Chimp Optimization Algorithm (ChOA)

Houssam Fouki[1], Léna Aix[1], Maxime Mulot[2], Amine Zaamoun[2], Hassan Amraoui[3], and Kenza Baddou[3]

[1] *Explained the Mathematical theory behind the Chimp Optimization Algorithm - Typed the report in LaTeX*
[2] *Implemented the Chimp Optimization Algorithm using Python - Took care of all the coding parts of the project*
[3] *Qualitatively explained the idea behind the Chimp Optimization Algorithm - Wrote the introduction and the conclusion*

Compiled November 28, 2022

Abstract — **For this optimization project, we used an article proposing a novel metaheuristic algorithm called Chimp Optimization Algorithm (ChOA) inspired by the individual intelligence and sexual motivation of chimps in their group hunting, which is different from the other social predators. ChOA, according to the author, is designed to further alleviate the two problems of slow convergence speed and trapping in local optima in solving high-dimensional problems such as a learning algorithm for high-dimension neural networks. In this report, we will first explain the mathematical model of diverse intelligence and sexual motivation proposed, and then we will show how we implemented the algorithm in the researcher's library.**

1. INTRODUCTION

A metaheuristic is a high-level problem-independent algorithmic framework that provides a set of guidelines or strategies to develop heuristic optimization algorithms. Metaheuristic algorithms are therefore always heuristic in nature and can be defined as optimization methods designed according to the strategies laid out in a metaheuristic framework. This distinguishes them from exact methods, that do come with proof that the optimal solution will be found in a finite (although often very large) amount of time. These methods (metaheuristics) are therefore developed specifically to find a "good enough" solution with a feasible computation time (small enough) to avoid combinatorial explosion. They have proven to be a viable, and often superior alternative to traditional methods as the trade-off between solution quality and computing time tends to favor them, especially when we're dealing with complex and large problems. Another advantage of these methods is that they are more flexible than the more classical ones in two important ways: First, metaheuristic frameworks are more general and can be adapted to fit the expected solution quality and allowed computing time of every problem. Secondly, metaheuristics do not put any constraints on the formulation of the problem. They wouldn't require the objective function to be in a specific type (linear or quadratic for example). The second one comes at the cost of requiring considerable problem-specific adaptation to achieve good performance. Evolutionary-based Algorithms (EAs), Swarm Intelligence-based (SI) algorithms, Human-based algorithms and Physics-based (PB) algorithms are the four types of metaheuristic algorithms. Evolutionary-based algorithms are the oldest and most common type of metaheuristics, these algorithms mimic the concepts of evolutionary behaviour of creatures in nature (the concept of survival of the fittest). In EAs, the initial random population evolves over generations to build new solutions and get rid of the worst ones in order to mend the fitness value. These algorithms commonly perform well in finding optimal or near-optimal solutions given that they do not make any credibility about the basic fitness landscape. A subcategory of EA are genetic algorithms which, by relying on biologically inspired operators such as mutation, crossover and selection, generate high-quality solutions to optimization and search problems. Swarm intelligence-basted algorithms are the leading class of metaheuristics for simulating the collective conduct of intelligent communities found in nature. In SI, each individual has its own wit and behaviour, but a combination of individuals in the algorithms offers more power to tackle complex optimization problems. This type of meta-heuristics is gaining more interest from researchers and is the type of method the researcher is using. The human-based algorithms and the physics-based algorithms are like their names suggest based (respectively) on human behaviour and physics fundamentals.

2. MATHEMATICAL MODEL AND ALGORITHM

In this section, we will present the mathematical model of the proposed algorithm. This meta-heuristic algorithm is inspired by the prey hunting of chimps. In order to hunt prey, chimps are dividing their individuals into 4 groups: the barriers, the drivers, the chasers and the attackers (Fig. 1). Each group has a specific role and strategy in prey hunting, and there are two main stages in the hunt: the exploration phase, and the exploitation phase. To adapt this behaviour to our mathematical problem, we are going to do an analogy: the chimps are our candidate solutions, and the prey is the optimal solution. Then, the algorithm is going to explore the search space as the chimps are exploring the space of their prey: this means that we are going to explore our search space with four different strategies. The exploitation phase can be described as discovering nearby promising solutions to develop their quality locally. And the exploration phase can be described as exploring the search space globally. Another factor, called chaotic behaviour, reflects the sexual motivation of chimps in the hunting process. It will be introduced later and allows the algorithm to avoid local optima.

A. Driving and Chasing the Prey

As mentioned before, the prey is hunted during the exploitation and exploration phases. To this end, we formulate chasing and driving the prey as follows:

$$\vec{d} = \left|\vec{c}.\vec{X}_{prey}(t) - \vec{m}.\vec{X}_{chimp}(t)\right| \tag{1}$$

$$\vec{X}_{chimp}(t+1) = \vec{X}_{prey}(t) - \vec{A}.\vec{d} \tag{2}$$

In which t denotes the current iteration, $\vec{X}_{chimp}$ indicates the current position of the chimp (the attacker, barrier, chaser, or driver). This means that at each step, the chimps are adjusting their positions according to the prey's position, their own previous position, and three coefficients: $\vec{c}$, $\vec{m}$, and $\vec{a}$. These vectors are calculated by Eq.s 3 to 5:

$$\vec{a} = 2.f.\vec{r}_1 - f \tag{3}$$

$$\vec{c} = 2.\vec{r}_2 \tag{4}$$

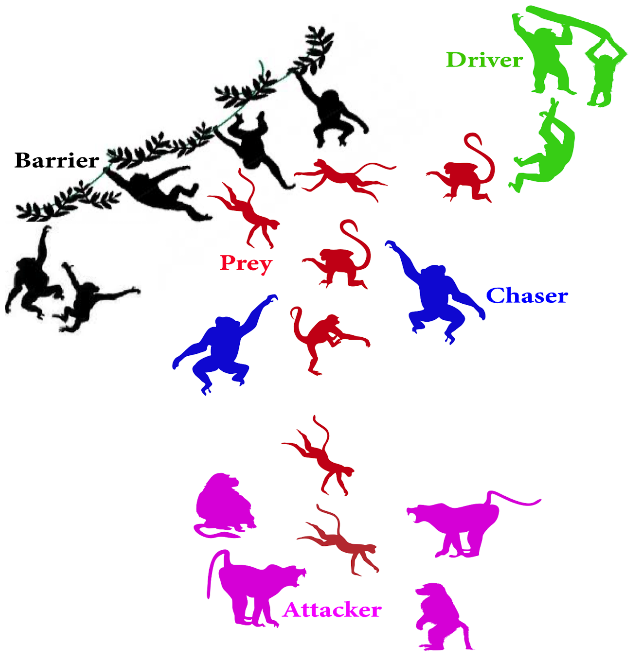


Fig. 1. Demonstration of the exploration (hunting) process.

$$\vec{m} = \text{Chaotic value} \tag{5}$$

Where $\vec{r}_1$ and $\vec{r}_2$ are random vectors that can vary in the range $[0, 1]$ for each chimp independently. $\vec{m}$ is a chaotic vector computed based on various chaotic maps to indicate the consequence of sexual motivation in the hunting process, we will detail it later. Lastly, f represents a control value that is diminished non-linearly from 2.5 to 0 through the iteration process (in both the exploitation and exploration phase). f is diminished according to a function common to each group: it represents the strategy of the group. The chosen function could be any decreasing continuous function of t (which means that at each iteration, f must decrease). The four independent groups use their own functions, and thus their own patterns to search the problem space locally and globally. The author tested different combinations of functions, and selected two combinations, called ChOA1 and ChOA2, providing the best performances. Those functions are summarized in Table 1. In this table, T represents the maximum number of iterations, and t indicates the current iteration.

B. Searching for Prey: Exploration phase

The previous equations allow both exploration and exploitation. Let's focus on the exploration for now. We want our equation to allow our chimp to explore the space in order to find all potentially interesting

Table 1. **The dynamic coefficients of f vector**

Groups	ChOA1	ChOA2
Group 1	$1.95 - 2.t^{\frac{1}{4}}/T^{\frac{1}{3}}$	$2.5 - (2\log(t)/\log(T))$
Group 2	$1.95 - 2.t^{\frac{1}{3}}/T^{\frac{1}{4}}$	$(-2t^3/T^3) + 2.5$
Group 3	$(-3t^3/T^3) + 1.5$	$0.5 + 2\exp[-(4t/T)^2]$
Group 4	$(-2t^3/T^3) + 1.5$	$2.5 + 2(t/T)^2 - 2(2t/T)$

areas to exploit during the later phase. Two of the coefficients are allowing it: c and a. Indeed, as r_1 and r_2 can vary in the range $[0,1]$, $2.r_1 - 1$ vary in the range $[-1,1]$, so a vary in the range $[-f, f]$. Thus, a allows to model a divergence behavior: when $|a| > 1$, the chimp is going to get distant from the prey and explore new areas. Similarly, c varies in the range $[0,2]$ so c provides random weights to reinforce (when $c > 1$) or lessen the effect of the prey location in the future destination of the chimp. Those two factors help the algorithm avoid local optima.

C. Attacking Method (Exploitation Phase)

The hunting process is usually performed by attackers. Occasionally, drivers, barriers, and chasers participate in hunting. Apparently, there is no position information in a conceptual search area. To mathematically reproduce the behavior of chimps, the initial attacker (the optimal choice), driver, barrier, and chaser are better informed about the status of the potential prey. Thus, four of the best solutions yet realized are kept, and the other chimps must change places. This is expressed as follows:

$$\begin{aligned} \vec{d}_{Attacker} &= |c_1\vec{X}_{Attacker} - \vec{m}_1\vec{X}| \\ \vec{d}_{Barrier} &= |c_2\vec{X}_{Barrier} - \vec{m}_2\vec{X}| \\ \vec{d}_{Chaser} &= |c_3\vec{X}_{Chaser} - \vec{m}_3\vec{X}| \\ \vec{d}_{Driver} &= |c_4\vec{X}_{Driver} - \vec{m}_4\vec{X}| \end{aligned} \tag{6}$$

$$\begin{aligned} \vec{X}_1 &=_{Attacker} -\vec{a}_1(\vec{d}_{Attacker}), \vec{X}_2 =_{Barrier} -\vec{a}_2(\vec{d}_{Barrier}) \\ \vec{X}_3 &=_{Chaser} -\vec{a}_3(\vec{d}_{Chaser}), \vec{X}_4 =_{Driver} -\vec{a}_4(\vec{d}_{Driver}) \end{aligned} \tag{7}$$

$$\vec{X}(t+1) = \frac{\vec{X}_1 + \vec{X}_2 + \vec{X}_3 + \vec{X}_4}{4} \tag{8}$$

Fig. 2 shows the process of updating the search chimp's location in a two-dimensional search space regarding the position of other chimp positions. As can be seen, the final position is randomly within a circle defined by the positions of the attacker, the barrier, the chaser, and the driver. In other words, the position of prey is estimated by the four best groups and other chimps randomly update their positions in its vicinity.

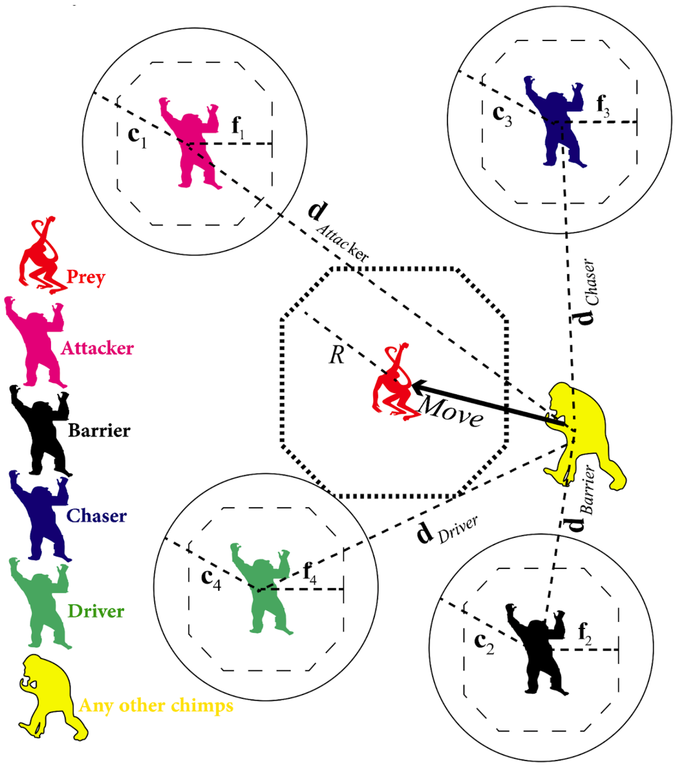


Fig. 2. Position updating in $ChOA$

D. Prey Attacking (Utilization)

The second and third roles of $\vec{a}$ are divergence and avoidance behaviors with respect to agent premises. As mentioned, the value of f is reduced non-linearly from 2.5 to 0. Since the value of $\vec{a}$ depends on f, it decreases by f. Otherwise speaking, $\vec{a}$ is a random variable in the interval of $[2f, 2f]$, whereas the f value reduces from 2.5 to 0 in the period of iterations. When the random values of $\vec{A}$ lie in the range of $[1,1]$, the next position of a chimp can be in any location between its current position and the position of the prey. To mathematically model the divergence behavior, the $\vec{a}$ vector bigger than 1 or smaller than 1 is utilized so that the search agents are compelled to diverge. This procedure depicts the exploration process and leads to searching globally. Another critical parameter in ChOA is the $\vec{c}$ vector. This random vector is very beneficial to ChOA for trapping local optima. This effect appears in both the initial and final iterations. It should be noted that chimps are sometimes progressing toward prey by certain obstacles placed along their progression path. Therefore, the random vector $\vec{c}$ can correctly model the phenomenon in nature.

E. Social Incentive (Sexual Motivation)

As mentioned earlier, chimpanzees come to abdicate their hunting responsibilities. That is why they are chaotically trying to get meat. This chaotic end-stage behavior helps chimps further mitigate both entrapment problems in locals and slow convergence rates in high-dimensional problems. This section describes the chaotic maps that have been used to improve the performance of ChOA. Six chaotic maps have been used in this article as shown in Fig. 3. These chaotic maps are deterministic processes that behave randomly. To model this simultaneous behavior, we assume that there is a 50% chance of choosing between the regular position update mechanism or the chaotic model to update the chimpanzee's position during optimization. The mathematical model is expressed by Eq. 9.

$$x_{chimp}(t+1) = \begin{cases} x_{prey}(t) - \vec{a}\vec{d} & \text{if } \mu < 0.5 \\ Chaotic\_Value & \text{if } \mu > 0.5 \end{cases} \tag{9}$$

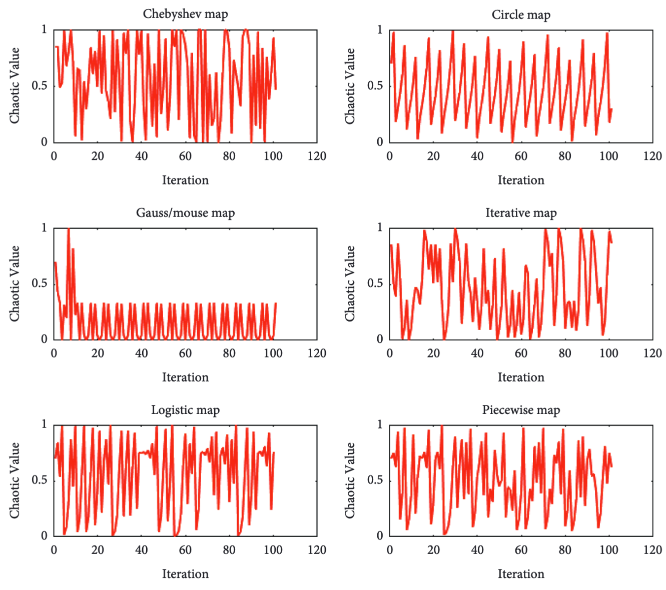
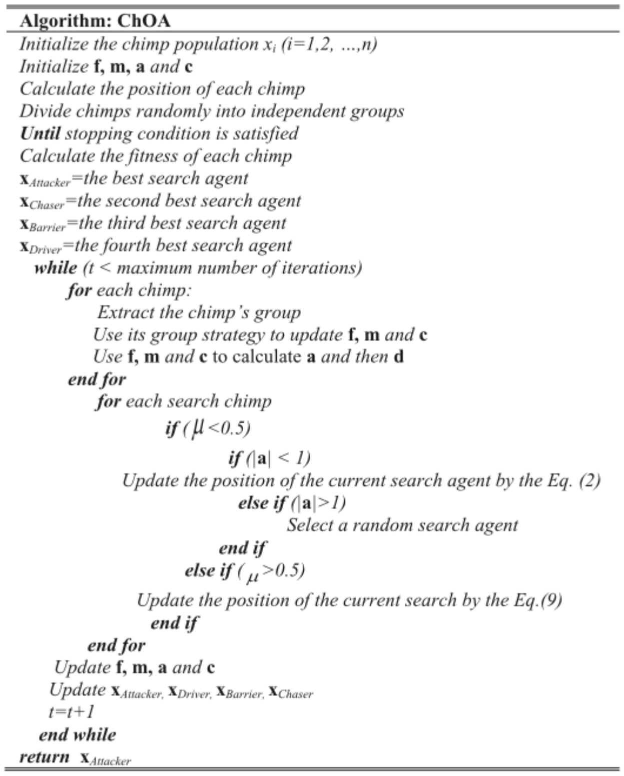


Fig. 3. The chaotic map diagrams

3. CODE CREATION

Based on the pseudo-code from the article (see Fig. 4), we wrote this additional optimizer in Python (see Listing. 1). We tried as much as possible to use the built-in functions of the existing package to create this new optimizer.



```
Algorithm: ChOA
Initialize the chimp population $x_i$ (i=1,2, ...,n)
Initialize f, m, a and c
Calculate the position of each chimp
Divide chimps randomly into independent groups
Until stopping condition is satisfied
Calculate the fitness of each chimp
$x_{Attacker}$=the best search agent
$x_{Chaser}$=the second best search agent
$x_{Barrier}$=the third best search agent
$x_{Driver}$=the fourth best search agent
  while ($t <$ maximum number of iterations)
      for each chimp:
          Extract the chimp's group
          Use its group strategy to update f, m and c
          Use f, m and c to calculate a and then d
      end for
          for each search chimp
              if ($\mu < 0.5$)
                  if ($|a| < 1$)
                      Update the position of the current search agent by the Eq. (2)
                  else if ($|a| > 1$)
                      Select a random search agent
                  end if
              else if ($\mu > 0.5$)
                  Update the position of the current search by the Eq.(9)
              end if
          end for
    Update f, m, a and c
    Update $x_{Attacker}$, $x_{Driver}$, $x_{Barrier}$, $x_{Chaser}$
    t=t+1
  end while
return $x_{Attacker}$
```

Fig. 4. Proposed pseudo-code

A. Chimp population creation

We first initialize the population by randomly creating chimp positions in the definition space and splitting them randomly into 4 groups (attackers, chasers, barriers, and drivers). Then we designate the first-best chimp of the population as the first attacker, the second-best chimp as the first chaser, the third-best as the first barrier, and the fourth-best as the first driver. These "best chimps" will be used to compute the new positions of all chimps.

B. The action phase

For each iteration (or until the condition is satisfied), we update the dynamic coefficient f (this coefficient varies from 2 to 0 until the last iteration), and the coefficients specific to the different roles of the chimps.

Additionally, for each search chimp, depending on a random value μ and a condition on a coefficient (see the pseudo-code for the condition), the chimp position is updated either according to its former position or by using a chaotic value. In this last case, the search agent position is updated in a deterministic way but with a "random behavior" (see Fig. 3).

Then, the position of the prey is estimated by calculating an average between the four best groups as explained earlier, and the other chimps update their positions accordingly (see Fig. 2).

In the last step of the loop, we update the best attacker, chaser, barrier, and driver as we did in the initialization phase.

4. CONCLUSION

This article proposes a new hunting-based optimization algorithm called ChOA. The proposed ChOA mimicked the social diversity and hunting behavior of chimps. Four hunting behaviors (driving, chasing, blocking, and attacking), multiple operators such as diverse intelligence and sexual motivation, and four types of chimpanzees have been proposed and mathematically modeled to provide ChOA with high exploitation and exploration. Each group has a specific role and strategy in prey hunting, and there are two main stages in the hunt: the exploration phase and the exploitation phase. To adapt this behavior to our mathematical problem, we did this analogy: the chimps are our candidate solutions, and the prey is the optimal solution. The algorithm will then explore the search space in a manner similar to how chimps look for their prey, which entails using four different ways to do so. Exploitation is the process of locating promising local solutions and enhancing their quality locally. Additionally, the exploration phase can be thought of as a global search space exploration. The final position is randomized in a circle defined by the positions of the attacker, the barrier, the chaser and the driver. In other words, the position of the prey is estimated by the four best groups and other chimpanzees update their positions randomly in its neighborhood. This algorithm benefits from convergence rate and high exploitation. Our main goal in this study was to understand how the algorithm works and implement it in the library created by the researcher. The ChOA was compared to nine well-known optimization algorithms in the literature. The results indicated that the ChOA provides very competitive results and outperforms other optimization algorithms in the majority of benchmark functions. Results also indicated that the ChOA is able to solve real-world optimization problems with unknown search spaces as well. So ChOA is able to be employed as an alternative optimizer for optimizing various high-dimensional optimization problems. The following are other conclusions that can be drawn from the study's findings:

1. Chimps being divided into several groups ensures that the search space is explored, especially for issues with higher dimensions.
2. The exploitation ability of the ChOA is emphasized by the suggested semi-deterministic property of chaotic maps.
3. The ChOA method can resolve local optima stagnations with the use of chaotic maps.
4. Since the ChOA method uses a four-type population of search agents to approximate the global optimum, local optimum avoidance is quite high.
5. The special decreasing shapes of various f parameter promotes exploitation and convergence rate as the iteration counter increases.
6. Chimpanzees memorize information from the search space during iteration.
7. ChOA generally has a few parameters to adjust and it almost uses memory to keep the best solution acquired so far.

5. REFERENCES

M. Khishe, M.R. Mosavi, Chimp optimization algorithm, Expert Systems with Applications, Volume 149, 2020, 113338, ISSN 0957-4174
Link to the request on GitHub

6. APPENDIX

```python
from numpy.lib.index_tricks import diag_indices_from
import numpy as np
from copy import deepcopy
from models.multiple_solution.root_multiple import RootAlgo
from random import random, randint

class BaseChOA(RootAlgo):

    ID_POS = 0
    ID_FIT = 1

    def __init__(self, root_algo_paras = None, choa_paras = None)
    :
        RootAlgo.__init__(self, root_algo_paras)
        self.epoch = choa_paras["epoch"]
        self.pop_size = choa_paras["pop_size"]
        self.chaotic_value = choa_paras["chaotic_value"]

    def _train__(self):

        '''
        Group IDs:
        Attacker = 0
        Barrier = 1
        Chaser = 2
        Driver = 3
        '''
        n_attackers = self.pop_size // 4
        n_barriers = n_attackers
        n_chasers = n_attackers
        n_drivers = n_attackers + self.pop_size % 4

        attackers = [self._create_solution__(minmax = 0) for _ in range(n_attackers)]
        barriers = [self._create_solution__(minmax = 0) for _ in range(n_barriers)]
        chasers = [self._create_solution__(minmax = 0) for _ in range(n_chasers)]
        drivers = [self._create_solution__(minmax = 0) for _ in range(n_drivers)]

        pop = [attackers, barriers, chasers, drivers]

        best_Attacker = self._get_global_best__(pop = attackers + barriers + chasers + drivers, id_fitness = self.ID_FIT, id_best = 0) # best chimp
        best_Barrier = self._get_global_best__(pop = attackers + barriers + chasers + drivers, id_fitness = self.ID_FIT, id_best = 1) # 2nd best chimp
        best_Chaser = self._get_global_best__(pop = attackers + barriers + chasers + drivers, id_fitness = self.ID_FIT, id_best = 2) # 3rd best chimp
        best_Driver = self._get_global_best__(pop = attackers + barriers + chasers + drivers, id_fitness = self.ID_FIT, id_best = 3) # 4th best chimp

        d_Attacker = np.zeros((self.problem_size, 1))
        d_Barrier = np.zeros((self.problem_size, 1))
        d_Chaser = np.zeros((self.problem_size, 1))
        d_Driver = np.zeros((self.problem_size, 1))

        for i in range(self.epoch):

          # The Dynamic Coefficient of f Vector as Table 1.
          f = (2 - i * ((2) / self.epoch))* np.ones((self.problem_size, 1)) # f decreases linearly from 2 to 0

          # Group 1
          ChOA1_G1 = 1.95 - ((2 * i^(1/4)) / (self.epoch^(1/3)))
          ChOA2_G1 = 2.5 - ((2 * np.log(i)) / (np.log(self.epoch)))

          # Group 2
          ChOA1_G2 = 1.95 - ((2 * i^(1/3)) / (self.epoch^(1/4)))
          ChOA2_G2 = (-2 * (i^3) / (self.epoch^3)) + 2.5

          # Group 3
          ChOA1_G3 = (-3 * (i^3) / (self.epoch^3)) + 1.5
          ChOA2_G3 = 0.5 + 2 * np.exp(-(4 * i / self.epoch)^2)

          # Group 4
          ChOA1_G4 = (-2 * (i^3) / (self.epoch^3)) + 1.5
          ChOA2_G4 = 2.5 + 2 * (i / self.epoch)^2 - 2 * (2 * i / self.epoch)

          for group in pop:
            for j in range(len(group)):

              r11 = ChOA1_G1 * np.random.rand(self.problem_size, 1) # r1 is a random number in [0,1]
              r12 = ChOA2_G1 * np.random.rand(self.problem_size, 1) # r2 is a random number in [0,1]

              r21 = ChOA1_G2 * np.random.rand(self.problem_size, 1)
              r22 = ChOA2_G2 * np.random.rand(self.problem_size, 1)

              r31 = ChOA1_G3 * np.random.rand(self.problem_size, 1)
              r32 = ChOA2_G3 * np.random.rand(self.problem_size, 1)

              r41 = ChOA1_G4 * np.random.rand(self.problem_size, 1)
              r42 = ChOA2_G4 * np.random.rand(self.problem_size, 1)

              a1 = 2 * f * r11 - f # equation (3)
              c1 = 2 * r12 # equation (4)
              m = Chaotic_value(dim = 1 , x1 = 1) # equation (5)

              a2 = 2 * f * r21 - f # equation (3)
              c2 = 2 * r22 # equation (4)

              a3 = 2 * f * r31 - f # equation (3)
              c3 = 2 * r32 # equation (4)

              a4 = 2 * f * r41 - f # equation (3)
              c4 = 2 * r42 # equation (4)

              if randint(0, 1): # the chimp choses to follow the strategy or to attack in a chaotic way

                  if np.linalg.norm(a1) < 1:
                    # Update the position of the current search agent by the Eq. (2)
                    d_Attacker = np.linalg.norm(c1 * best_Attacker[0] - m * group[j][0])
                    group[j][0] = best_Attacker[0] - a1 * d_Attacker
                  else:
                    # Select a random search agent
                    x_rand = self._create_solution__()
                    d_Attacker = np.linalg.norm(c1 * x_rand[0] - m * group[j][0])
                    group[j][0] = x_rand[0] - a1 * d_Attacker

                  if np.linalg.norm(a2) < 1:
                    d_Barrier = np.linalg.norm(c2 * best_Barrier[0] - m * group[j][0])
                    group[j][0] = best_Barrier[0] - a2 * d_Barrier
                  else:
                    x_rand = self._create_solution__()
                    d_Barrier = np.linalg.norm(c2 * x_rand[0] - m * group[j][0])
                    group[j][0] = x_rand[0] - a2 * d_Barrier

                  if np.linalg.norm(a3) < 1:
                    d_Chaser = np.linalg.norm(c3 * best_Chaser[0] - m * group[j][0])
                    group[j][0] = best_Chaser[0] - a3 * d_Chaser
                  else:
                    x_rand = self._create_solution__()
                    d_Chaser = np.linalg.norm(c3 * x_rand[0] - m * group[j][0])
                    group[j][0] = x_rand[0] - a3 * d_Chaser

                  if np.linalg.norm(a4) < 1:
                    d_Driver = np.linalg.norm(c4 * best_Driver[0] - m * group[j][0])
                    group[j][0] = best_Driver[0] - a4 * d_Driver
                  else:
                    x_rand = self._create_solution__()
                    d_Driver = np.linalg.norm(c4 * x_rand[0] - m * group[j][0])
                    group[j][0] = x_rand[0] - a4 * d_Driver

                  group[j][0] = (best_Attacker[0] + best_Barrier[0] + best_Chaser[0] + best_Driver[0] - a1 * d_Attacker - a2 * d_Barrier - a3 * d_Chaser - a4 * d_Driver) / 4
                  group[j][1] = self._fitness_model__(solution = group[j][0], minmax = 0)

              else:
                  # Update the position of the current search agent by the Eq. (9)
                  group[j][0] = Chaotic_value(dim = self.problem_size, x = group[j][0])
                  group[j][1] = self._fitness_model__(solution = group[j][0], minmax = 0)


          best_attacker = self._get_global_best__(pop = attackers + barriers + chasers + drivers, id_fitness = self.ID_FIT, id_best = 0) # best chimp
          best_barrier = self._get_global_best__(pop = attackers + barriers + chasers + drivers, id_fitness = self.ID_FIT, id_best = 1) # 2nd best chimp
          best_chaser = self._get_global_best__(pop = attackers + barriers + chasers + drivers, id_fitness = self.ID_FIT, id_best = 2) # 3rd best chimp
          best_driver = self._get_global_best__(pop = attackers + barriers + chasers + drivers, id_fitness = self.ID_FIT, id_best = 3) # 4th best chimp

          self.loss_train.append(best_attacker[self.ID_FIT])



        def Chaotic_value(dim , x1, index = 1): # x refers to the previous position of the chimp

          x = np.random.rand((dim, 1)) # x between 0 and 1, translates the state of "sexual excitment" of the chimp
          G = np.zeros((dim,1))

          match index:

            case 1: # Quadratic
            c = 1
            G = x**2 - c * np.ones((dim,1))

          case 2: # Gauss/mouse
            for i in range(0, dim):
              if x[i] == 0:
                G[i] = 0
              else:
                G[i] = (1 / x[i]) % 1

          case 3: # Logistic
            alpha = 4
            for i in range(0, dim):
              G[i] = alpha * x[i] * (1 - x[i])

          case 4:  # Singer
            mu = 1.07
            for i in range(0, dim):
              G[i] = mu * (7.86 * x[i] - 23.31 * x[i]^2 + 28.75
      * x[i]^3 - 13.302875 * x[i]^4)

          case 5: # Bernoulli
            for i in range(0, dim):
              G[i] = (2 * x[i]) % 1

          case 6: # Tent
            for i in range(0, dim):
              if x[i] < 0.7:
                G[i] = x[i] / 0.7
              if x[i] >= 0.7:
                G[i] = (10 / 3) * (1 - x[i])

        G = G * x1
        return G

      return best_attacker
```

Listing 1. Proposed Python code