# 6

# *Chicken Swarm Optimization*

**Dorin Moldovan**
*Department of Computer Science*
*Technical University of Cluj-Napoca, Romania*

**Adam Slowik**
*Department of Electronics and Computer Science*
*Koszalin University of Technology, Koszalin, Poland*

**CONTENTS**



## 6.1 Introduction

Chicken Swarm Optimization (CSO) was introduced in [1] and its main inspiration is represented by the behavior of chicken swarms, in particular the behavior of hens, chicks and roosters, and by the hierarchical order of the chicken swarms. The CSO algorithm can be applied in various optimization problems in which the solutions have a fitness function and they can be represented in a $D$-dimensional space. The solutions are represented by chickens and each chicken has a position and a fitness value. The chicken with the best fitness value represents the best solution. CSO identifies three types of chickens: roosters, hens and chicks. The mathematical version of CSO respects four rules which are briefly presented next and described in more detail in the next section of the chapter: (1) a chicken swarm has several groups, (2) the types of the chickens depend on the fitness values of the chickens, (3) the statuses in each group are updated every $G$ time steps and (4) the chickens update their positions in each time step respecting rules which are inspired from the searching for food behavior. Several modifications of the CSO algorithm are Binary Chicken Swarm Optimization (BCSO) [2], Mutation Chicken Swarm Optimization (MCSO) [3] and Chaotic Chicken Swarm Optimization (CCSO) [4]. CSO was also used in combination with other algorithms and two illustrative examples are Bat-Chicken Swarm Optimization (B-CSO) [5] and Cuckoo Search-Chicken Swarm Optimization (CS-CSO) [6]. In literature it was applied for solving various types of optimization problems such as the 0-1 knapsack problem [7], features selection [8], classification problems [9], attitude determination [10], tuning the number of nodes for a deep learning model [11] and constrained optimization problems [12]. The paper is organized as follows: Section 6.2 presents the original CSO algorithm, Section 6.3 presents the source-code of the global version of the CSO algorithm in Matlab, Section 6.4 presents the source-code of the global version of the CSO algorithm in C++, Section 6.5 presents a numerical example of the global version of the CSO algorithm and Section 6.6 presents the main conclusions.

## 6.2 Original CSO algorithm

### 6.2.1 Pseudo-code of global version of CSO algorithm

The global version of the CSO algorithm is presented in Algorithm 5 and it is adapted after the one from [1]. The input parameters of the algorithm are: $D$ - the number of dimensions of the search space, $N$ - the number of chickens, $RN$ - the number of roosters, $HN$ - the number of hens, $CN$ - the number of chicks, $MN$ - the number of mother hens, $FL$ - a random number from the interval $[0.5, 0.9]$, $G$ - a number which indicates how often the hierarchy of the swarm is changed, $I_{max}$ - the maximum number of iterations and $[C_{min}, C_{max}]$ - an interval that describes the minimum and the maximum possible values of the positions of the chickens. The output of the algorithm is represented by the global best which is the best position achieved by a chicken so far. The initial population of $N$ chickens $C_i$ with $i = 1, ..., N$ is created randomly in *step 1* such that each chicken is represented by a $D$-dimensional vector. In *step 2* of the algorithm are computed the fitness values of all chickens, in *step 3* the *Gbest* $D$-dimensional vector is created and in *step 4* the best chicken according to the fitness value is assigned to $Gbest$. Initially the value of $t$ is equal to 0 (step 5). While $t$ is less than a maximum number of iterations $I_{max}$ repeat the following sequence of steps. In *step 7* the condition ($t$ modulo $G$) $== 0$ is verified and if the outcome is positive then in *step 8* the fitness values of the chickens are ranked establishing the hierarchical order of the swarm and in *step 9* the swarm of chickens is divided into several groups and for each group the relations between the chicks and the associated mother are established. Next for each chicken $C_i$ from the swarm of $N$ chickens the new positions are updated using formulas that are different for each type of chicken (steps 12-20). In *step 21* the value of $C_{i,j}^{t+1}$ is updated if it is not in the interval $[C_{min}, C_{max}]$ as follows: if the value is less than $C_{min}$ then it takes the value $C_{min}$ and if the value is greater than $C_{max}$ then it takes the value $C_{max}$. The fitness value of the new solution is evaluated in *step 22* and if the new solution is better than the current solution *Gbest* then the value of *Gbest* is updated (step 23). In *step 25* the current iteration $t$ is updated using the formula $t = t + 1$ and finally in *step 27 Gbest* is returned as the final result of the algorithm.

**Algorithm 5** Pseudo-code of the global version of CSO.

1: create the initial population of chickens $C_i (i = 1, 2, ..., N)$ randomly such that each chicken is represented by a $D$-dimensional vector
2: evaluate the fitness values of all $N$ chickens
3: create the $D$-dimensional vector *Gbest*
4: assign the best chicken $C_i$ to *Gbest*
5: $t = 0$
6: **while** $t < I_{max}$ **do**
7:     **if** $t$ modulo $G == 0$ **then**
8:         rank the fitness values of the chickens and establish the hierarchical order of the swarm
9:         divide the swarm of chickens into several groups and determine the relations between the chicks and the associated mother hens in each group
10:     **end if**
11:     **for** $i = 1 : N$ **do**
12:         **if** $C_i ==$ rooster **then**
13:             $C_{i,j}^{t+1} = C_{i,j}^{t} \times \left(1 + \mathcal{N}(0, \sigma^2)\right)$
14:         **end if**
15:         **if** $C_i ==$ hen **then**
16:             $C_{i,j}^{t+1} = C_{i,j}^{t} + S_1 \times R_1 \times \left(C_{r_1,j}^{t} - C_{i,j}^{t}\right) + S_2 \times R_2 \times \left(C_{r_2,j}^{t} - C_{i,j}^{t}\right)$
17:         **end if**
18:         **if** $C_i ==$ chick **then**
19:             $C_{i,j}^{t+1} = C_{i,j}^{t} + FL \times (C_{m,j}^{t} - C_{i,j}^{t})$
20:         **end if**
21:         update the value $C_{i,j}^{t+1}$ if it is not in the interval $[C_{min}, C_{max}]$
22:         evaluate the fitness value of the new solution
23:         if the new solution is better than *Gbest* then update *Gbest*
24:     **end for**
25:     $t = t + 1$
26: **end while**
27: return the *Gbest* as a result

### 6.2.2 Description of global version of CSO algorithm

This section presents the global version of the CSO algorithm. Initially the following parameters of the algorithm are defined: $D$, $N$, $RN$, $HN$, $CN$, $MN$, $FL$, $G$, $I_{max}$ and $[C_{min}, C_{max}]$. The value of $FL$ is usually in the interval $[0.5, 0.9]$ and the value of $G$ is in the interval $[2, 20]$. In [1] the original authors of the algorithm use the following values for the initial parameters of the algorithm: $RN = 0.2 \times N$, $HN = 0.6 \times N$, $CN = N - RN - HN$, $MN = 0.1 \times N$ and $G = 10$. The values of $C_{min}$ and $C_{max}$ are problem dependent. The initial population of chickens $C_i$ with $i = 1, 2, ..., N$ is created randomly and each chicken is described by a $D$-dimensional vector. For each chicken from the set of $N$ chickens the fitness value is evaluated and the best chicken is assigned to the *Gbest* $D$-dimensional vector. For a number of iterations $I_{max}$ that is given as input a sequence of steps is repeated. If the current iteration is divisible without rest by $G$ then the hierarchy of the swarm of chickens is established considering the fitness values of the chickens and the swarm is then divided into several groups that have a dominant rooster, several hens and chicks. The chickens that have the best $RN$ fitness values are roosters, the chickens with the next best $HN$ fitness values are hens and the remaining $CN$ chickens are chicks. The relations between the chicks and the mother hens are determined for each group considering the fitness values of the chickens. For each chicken $C_i$ from the group of $N$ chickens where $i = 1, 2, ..., N$ the positions in the next iteration of the algorithm are updated considering the type of the chicken.

If the chicken is a rooster then the formula that is used for updating the position is:

$$C_{i,j}^{t+1} = C_{i,j}^{t} \times \left(1 + \mathcal{N}(0, \sigma^2)\right) \tag{6.1}$$

where $t+1$ is the next iteration, $t$ is the current iteration, $j$ takes values from the set $\{1, ..., D\}$ and $\mathcal{N}(0, \sigma^2)$ is a Gaussian with standard deviation $\sigma^2$ and mean 0. The value of $\sigma^2$ is given by the following formula:

$$\sigma^2 = \begin{cases} 1 & \text{if } F_i \le F_k \\ e^{\frac{F_k - F_i}{|F_i| + \epsilon}} & \text{otherwise} \end{cases} \tag{6.2}$$

where $\epsilon$ is a small positive constant that is used in order to avoid division by 0, $F_i$ is the fitness value of the $i$-th chicken and $F_k$ is the fitness value of a rooster that is selected randomly from the group of roosters.

If the chicken is a hen then the formula which is used for updating the position is:

$$C_{i,j}^{t+1} = C_{i,j}^{t} + S_1 \times R_1 \times \left(C_{r_1,j}^{t} - C_{i,j}^{t}\right) + S_2 \times R_2 \times \left(C_{r_2,j}^{t} - C_{i,j}^{t}\right) \tag{6.3}$$

where $R_1$ is a random number from $[0, 1]$, $R_2$ is a random number from $[0, 1]$, $r_1$ is the index of the rooster which is the hen's mate and $r_2$ is the index of a rooster or of a hen which is randomly chosen from the swarm such that $r_1 \neq r_2$. The rooster that is a given hen's mate is selected randomly from the set of roosters when the hierarchical order of the swarm is updated.

The formulas for $S_1$ and $S_2$ are the following:

$$S_1 = e^{\frac{F_i - F_{r_1}}{|F_i| + \epsilon}} \tag{6.4}$$

$$S_2 = e^{(F_{r_2} - F_i)} \tag{6.5}$$

where $F_i$ is the fitness value of the chicken $C_i$, $F_{r_1}$ is the fitness value of the rooster $r_1$ and $F_{r_2}$ is the fitness value of the rooster $r_2$.

If the chicken is a chick then the formula that is used for updating the position is:

$$C_{i,j}^{t+1} = C_{i,j}^{t} + FL \times (C_{m,j}^{t} - C_{i,j}^{t}) \tag{6.6}$$

where $m$ is the mother of the $i$-th chicken. We decided that $m$ is the mother of the $i$-th chicken when we updated the hierarchy of the swarm.

## 6.3 Source-code of global version of CSO algorithm in Matlab

Listing 6.1 presents the source-code of the objective function that is optimized by the CSO algorithm. In the function $OF(x, D)$ the input parameter $x$ is the vector of decision variables, and $D$ is the number of dimensions. The objective function is given by formula 6.7.

$$OF(x, D) = \sum_{i=1}^{D} x_i^2 \qquad \text{where } -5.12 \leqslant x_i \leqslant 5.12 \tag{6.7}$$

```
function [y]=OF(x,D)
y=0;
for i=1:D
y=y+x(1,i)*x(1,i);
end
end
```

**Listing 6.1**
Definition of objective function $OF(x,D)$ in Matlab.

```
D=5; N=10; G=10; FL_MIN=0.5; FL_MAX=0.9; e=10^(-9); RP=20;
HP=60; MAX=30; TheBest=1; Cmin=zeros(1,D); Cmax=zeros(1,D);
cType=zeros(1,N); cPositionInSwarm=zeros(1,N); C=zeros(N,D);
E=zeros(1,N); Gbest=zeros(1,D); Cmin(1,:)=-5.12; Cmax(1,:)=5.12;
nRoosters=round((N*RP)/100); nHens=round((N*HP)/100);
hens=zeros(1,nHens); nChicks=N-nRoosters-nHens;
chicks=zeros(1,nChicks); roosters=zeros(1,nRoosters);
cPositionInSwarm(1,:)=linspace(1,N,N);
hRoosterRelation=zeros(1,N); cHenRelation=zeros(1,N);
for i=1:N
  for j=1:D
    C(i,j)=(Cmax(1,j)-Cmin(1,j))*rand()+Cmin(1,j);
  end
  E(1,i)=OF(C(i,:),D);
```

```
  if E(1,i)<E(1,TheBest)
    TheBest=i;
  end
end
Gbest(1,:)=C(TheBest,:); EGbest=E(1,TheBest);
t=0;
while(t<=MAX)
  if mod(t,G)==0
  f_values=E(1,:);
  cType(1,:)=0;
  f_values=sort(f_values);
  t_roosters=f_values(1,nRoosters);
  t_hens=f_values(1,nRoosters+nHens);
  rIndex=1; hIndex=1; cIndex=1;
  for i=1:N
    if E(1,i)<=t_roosters
      cType(1,i)=2; roosters(1,rIndex)=i;
      if rIndex>nRoosters
        break;
      end
      rIndex=rIndex+1;
    end
  end
  for i=1:N
    if E(1,i)<=t_hens && cType(1,i)~=2
      cType(1,i)=1; hens(1,hIndex)=i;
      if hIndex>nHens
        break;
      end
      hIndex=hIndex+1;
    end
  end
  for i=1:N
    if cType(1,i)==0
      chicks(1,cIndex)=i;
      cIndex=cIndex+1;
    end
  end
  for i=1:nHens
    hIndex=hens(1,i);
    r=randi(nRoosters); rIndex=roosters(1,r);
    hRoosterRelation(1,hIndex)=rIndex;
  end
  for i=1:nChicks
    cIndex=chicks(1,i);
    hen=randi(nHens); hIndex=hens(1,hen);
    cHenRelation(1,cIndex)=hIndex;
  end
  end
  for i=1:N
    for j=1:D
      if cType(1,i)==2
        sigma_squared=1; k=randi(nRoosters);
        if E(1,k)<E(1,i)
          sigma_squared=exp((E(1,k) - E(1,i))/(abs(E(1,i)) + e));
        end
        C(i,j)=C(i,j)*(1+sigma_squared*randn());
      elseif cType(1,i)==1
        r1=hRoosterRelation(1,i); r2=r1;
        while (r2~=r1)
          type=randi(2);
          if type==0
            r2=roosters(1,randi(nRoosters));
          else
            r2=hens(randi(nHens));
          end
        end
```

```
        s1=exp((E(1,i) - E(1,r1))/(abs(E(1,i)) + e));
        s2=exp(E(1,r2) - E(1,i));
        C(i,j)=C(i,j)+s1*rand()*(C(r1,j)-C(i,j))
          +rand()*s2*(C(r2,j)-C(i,j));
      else
        m=cHenRelation(1,i);
        C(i,j)=C(i,j)+(FL_MIN+rand()
          *(FL_MAX-FL_MIN))*(C(m,j)-C(i,j));
      end
      if C(i,j) < Cmin(1,j)
        C(i,j) = Cmin(1,j);
      end
      if C(i,j) > Cmax(1,j)
        C(i,j) = Cmax(1,j);
      end
    end
  end
  TheBest=1;
  for i=1:N
    E(1,i)=OF(C(i,:),D);
    if E(1,i)<E(1,TheBest)
      TheBest=i;
    end
  end
  if E(1,TheBest)<EGbest
    EGbest=E(1,TheBest);
    Gbest(1,:)=C(TheBest,:);
  end
  t=t+1;
end
disp('FINAL RESULT'); disp(Gbest(1,:)); disp(EGbest);
```

**Listing 6.2**
Source-code of the global version of CSO in Matlab.

## 6.4 Source-code of global version of CSO algorithm in C++

```
#include <iostream>
#include <cstdlib>
#include <ctime>
#include <string>
#include <bits/stdc++.h>
#include <math.h>
using namespace std;
float OF(float x[], int size_array) {
  float t = 0;
  for(int i = 0; i < size_array; i++) t = t + x[i] * x[i];
  return t;
}
float r() {return (float)(rand()%1000)/1000;}
double rand(double min, double max) {
  return min + (double) rand() / RAND_MAX * (max - min);
}
int main() {
  srand(time(NULL));
  int D = 5; int N = 10; int G = 10; double FL_MIN = 0.5;
  double FL_MAX = 0.9; double e = 0.000000001; int RP = 20;
  int HP = 60; float Cmin[D]; float Cmax[D]; int TheBest = 0;
```

```
int MAX = 30; int cType[N]; int cPositionInSwarm[N];
for(int i = 0; i < D; i++) {Cmin[i] = -5.12; Cmax[i] = 5.12;}
float C[N][D]; float E[N]; float Gbest[D]; float EGbest;
int nRoosters = (N * RP) / 100; int nHens = (N * HP) / 100;
int hens[nHens]; int nChicks = N - nRoosters - nHens;
int chicks[nChicks]; int roosters[nRoosters];
unordered_map<int, int> hRoosterRelation;
unordered_map<int, int> cHenRelation;
for(int i = 0; i < N; i++) cPositionInSwarm[i] = i;
for(int i = 0; i < N; i++) {
  for(int j = 0; j < D; j++)
    C[i][j] = (Cmax[j] - Cmin[j]) * r() + Cmin[j];
  E[i] = OF(C[i], D); if(E[i] < E[TheBest]) TheBest = i;
}
for(int j = 0; j < D; j++) Gbest[j] = C[TheBest][j];
EGbest = E[TheBest]; int t = 0;
while(t < MAX) {
  if(t % G == 0) {
    vector<double> f_values;
    for(int i = 0; i < N; i++) f_values.push_back(E[i]);
    for(int i = 0; i < N; i++) cType[i] = 0;
    sort(f_values.begin(), f_values.end());
    double t_roosters = f_values[nRoosters - 1];
    double t_hens = f_values[nRoosters + nHens - 1];
    int rIndex = 0; int hIndex = 0; int cIndex = 0;
    for(int i = 0; i < N; i++) {
      if(E[i] <= t_roosters) {
        cType[i] = 2; roosters[rIndex++] = i;
        if(rIndex >= nRoosters) break;
      }
    }
    for(int i = 0; i < N; i++) {
      if(E[i] <= t_hens && cType[i] != 2) {
        cType[i] = 1; hens[hIndex++] = i;
        if(hIndex >= nHens) break;
      }
    }
    for(int i = 0; i < N; i++)
      if(cType[i] == 0) chicks[cIndex++] = i;
    for(int i = 0; i < nHens; i++) {
      int hIndex = hens[i]; int r = rand() % nRoosters;
      int rIndex = roosters[r];
      hRoosterRelation[hIndex] = rIndex;
    }
    for(int i = 0; i < nChicks; i++) {
      int cIndex = chicks[i]; int hen = rand() % nHens;
      int hIndex = hens[hen]; cHenRelation[cIndex] = hIndex;
    }
  }
  for(int i = 0; i < N; i++) {
    for(int j = 0; j < D; j++) {
      if (cType[i] == 2) {
        double sigma_squared = 1; int k = rand() % nRoosters;
        if(E[k] < E[i])
          sigma_squared = exp((E[k] - E[i])/(abs(E[i]) + e));
        default_random_engine generator;
        normal_distribution<double> d(0,sigma_squared);
        C[i][j] = C[i][j] * (1 + d(generator));
      } else if (cType[i] == 1) {
        int r1 = hRoosterRelation[i]; int r2 = r1;
        while(r2 != r1) {
          int type = rand() % 2;
          if(type == 0) {r2 = roosters[rand() % nRoosters];}
          else {r2 = hens[rand() % nHens];}
        }
        double s1 = exp((E[i] - E[r1])/(abs(E[i]) + e));
        double s2 = exp(E[r2] - E[i]);
```

```
          C[i][j] = C[i][j] + s1 * rand(0, 1) * (C[r1][j]
            - C[i][j]) + rand(0, 1) * s2 * (C[r2][j] - C[i][j]);
        } else {
          int m = cHenRelation[i];
          C[i][j] = C[i][j] + rand(FL_MIN, FL_MAX)
            * (C[m][j] - C[i][j]);
        }
        if(C[i][j] < Cmin[j]) C[i][j] = Cmin[j];
        if(C[i][j] > Cmax[j]) C[i][j] = Cmax[j];
      }
    }
    TheBest = 0;
    for(int i = 0; i < N; i++) {
      E[i] = OF(C[i], D);
      TheBest = (E[i] < E[TheBest]) ? i : TheBest;
    }
    if(E[TheBest] < EGbest) {
      EGbest = E[TheBest];
      for(int j = 0; j < D; j++) Gbest[j] = C[TheBest][j];
    }
    t = t + 1;
  }
  cout << EGbest << endl; return 0;
}
```

**Listing 6.3**
Source-code of the global version of CSO in C++.

## 6.5 Step-by-step numerical example of global version of CSO algorithm

In the first step, we assume that we want to minimize the objective function $OF(x, D)$ given by equation 6.7 where the number of dimensions $D$ is equal to 5. For simplicity in this step-by-step numerical example we consider that $OF(C) = OF(C, D) = OF(C, 5)$ where $C$ is an array that describes the position of a chicken.

In the second step, we initialize the parameters of CSO such that the number of chickens $N$ is equal to 10, the swarm of chickens updates the hierarchy every $G = 10$ iterations, $FL = [0.5, 0.9]$, $\epsilon = 10^{-9}$, the roosters percent is 20%, the hens percent is 60%, the maximum number of iterations is 30, $C_{min} = -5.12$ and $C_{max} = 5.12$.

In the third step, the chicken swarm that consists of 10 chickens is randomly created and each chicken is represented by a 5-dimensional vector that describes the position of the chicken.
$C_1 = \{-4.700, -0.337, -1.699, 0, -3.389\}$
$C_2 = \{2.293, -0.225, -1.454, 4.730, -0.368\}$
$C_3 = \{2.099, -3.635, -2.242, 3.348, 4.720\}$
$C_4 = \{-0.092, 5.068, 4.526, 3.348, -0.655\}$
$C_5 = \{-1.116, 1.064, 4.116, -3.553, -2.129\}$
$C_6 = \{-1.208, -0.808, 2.211, 2.232, 4.044\}$

$C_7 = \{-0.542, 2.314, 2.775, 0.389, 3.778\}$
$C_8 = \{4.218, 1.710, -2.058, -4.761, 4.034\}$
$C_9 = \{2.078, 3.184, -1.822, -1.710, 1.771\}$
$C_{10} = \{1.679, -3.676, 2.160, -2.529, 3.768\}$

In the fourth step, the 5-dimensional $Gbest$ vector is created.
$Gbest = \{0, 0, 0, 0, 0\}$

In the fifth step, we evaluate each chicken $C_i$ using the objective function $OF(.)$.
$E_1 = OF(C_1) = 36.583, E_2 = OF(C_2) = 29.943,$
$E_3 = OF(C_3) = 56.147, E_4 = OF(C_4) = 57.828,$
$E_5 = OF(C_5) = 36.487, E_6 = OF(C_6) = 28.350,$
$E_7 = OF(C_7) = 27.780, E_8 = OF(C_8) = 63.910,$
$E_9 = OF(C_9) = 23.848, E_{10} = OF(C_{10}) = 41.600$

In the sixth step, we consider the best chicken $C_i$ which has the smallest value for the objective function because the goal of the algorithm is to minimize the value of $OF(.)$ and we assign the vector that represents the chicken with the smallest $OF(.)$ value to $Gbest$. The best chicken is $C_9$ and thus $Gbest = \{2.078, 3.184, -1.822, -1.710, 1.771\}$ and $OF(Gbest) = 23.848$.

In the seventh step of the algorithm, the main loop is started. We check whether the current iteration is less than $I_{max} = 30$ and if that condition is true the algorithm jumps to the final step, otherwise the algorithm continues with the eighth step.

In the eighth step of the algorithm, we check whether the value of the current iteration is divisible without rest by $G = 10$ and if that condition is true then the algorithm updates the hierarchy of the chicken swarm, otherwise the algorithm continues with the ninth step.

**Chicken swarm hierarchy update**

In the first substep, the chickens are sorted according to their fitness value and the best $20\% \times 10 = 2$ chickens are considered roosters, the next best $60\% \times 10 = 6$ chickens are considered hens and the rest of the chickens are considered chicks.
$TypeOf(C_1) = Hen, TypeOf(C_2) = Hen,$
$TypeOf(C_3) = Hen, TypeOf(C_4) = Chick,$
$TypeOf(C_5) = Hen, TypeOf(C_6) = Hen,$
$TypeOf(C_7) = Rooster, TypeOf(C_8) = Chick,$
$TypeOf(C_9) = Rooster, TypeOf(C_{10}) = Hen$

In the second substep, each hen is associated with a rooster randomly and that rooster is considered the head of the group the hen belongs to.
$RoosterOf(C_1) = C_9, RoosterOf(C_2) = C_7,$
$RoosterOf(C_3) = C_7, RoosterOf(C_5) = C_9,$
$RoosterOf(C_6) = C_9, RoosterOf(C_{10}) = C_9$

In the third substep, each chick is associated with a hen randomly and that hen is considered the mother of the chick.
$HenMotherOf(C_4) = C_{10}$
$HenMotherOf(C_8) = C_5$

In the ninth step of the algorithm, each chicken updates its current position considering one of the equations 6.1, 6.3 or 6.6 such that the roosters use the equation 6.1, the hens use the equation 6.3 and the chicks use the equation 6.6. Then we check whether the value of each decision variable is within the range $[C_{min}, C_{max}]$. The new positions of the chickens are:
$C_1 = \{3.366, 1.537, -1.817, -0.021, -1.372\}$
$C_2 = \{0.275, 2.273, 0.850, 1.026, 2.125\}$
$C_3 = \{-1.160, 2.271, 3.869, 0.540, 4.484\}$
$C_4 = \{1.094, -2.112, 2.854, -1.917, 2.886\}$
$C_5 = \{0.444, 3.035, 3.582, -2.237, 3.108\}$
$C_6 = \{2.304, 0.628, 1.835, -0.931, 1.686\}$
$C_7 = \{-0.476, 2.031, 2.436, 0.341, 3.317\}$
$C_8 = \{2.097, 2.639, 2.414, -3.090, 3.467\}$
$C_9 = \{1.825, 2.796, -1.600, -1.501, 1.555\}$
$C_{10} = \{1.706, 4.601, -0.818, -1.858, 0.549\}$

In the tenth step of the algorithm, we compute the new fitness values of the chickens.
$E_1 = OF(C_1) = 18.886, E_2 = OF(C_2) = 11.536,$
$E_3 = OF(C_3) = 41.880, E_4 = OF(C_4) = 25.815,$
$E_5 = OF(C_5) = 36.916, E_6 = OF(C_6) = 12.786,$
$E_7 = OF(C_7) = 21.416, E_8 = OF(C_8) = 38.768,$
$E_9 = OF(C_9) = 18.385, E_{10} = OF(C_{10}) = 28.513$

In the eleventh step of the algorithm, we update the value of *Gbest* if the algorithm finds a better value. Then we return to the seventh step of the algorithm.
$Gbest = \{0.275, 2.273, 0.850, 1.026, 2.125\}$
$OF(Gbest) = 11.536$

In the twelfth step the algorithm returns *Gbest* as the final result. After 30 iterations the value of *Gbest* is $\{0.043, 0.064, -0.038, -0.036, 0.035\}$ and the value of $OF(Gbest)$ is 0.010.

## 6.6 Conclusions

This chapter presented the main principles of the Chicken Swarm Optimization algorithm. We presented the pseudo code of the algorithm, the source code in Matlab and the source code in C++. In addition we showed how this algorithm works in the global version providing a step-by-step numerical example. We believe that this chapter will facilitate the development of other versions of the algorithm in other programming languages.

## References


1. X. Meng, Y. Liu, X. Gao, H. Zhang. "A New Bio-inspired Algorithm: Chicken Swarm Optimization" in *Lecture Notes in Computer Science*, vol. 8794, 2014, pp. 86-94.
2. M. Han, S. Liu. "An Improved Binary Chicken Swarm Optimization Algorithm for Solving 0-1 Knapsack Problem" in *Proc. of the 2017 13th International Conference on Computational Intelligence and Security (CIS)*, 2017, pp. 207-210.
3. K. Wang, Z. Li, H. Cheng, K. Zhang. "Mutation Chicken Swarm Optimization Based on Nonlinear Inertia Weight" in *Proc. of the 2017 3rd IEEE International Conference on Computer and Communications (ICCC)*, 2017, pp. 2206-2211.
4. K. Ahmed, A. E. Hassanien, S. Bhattacharyya. "A Novel Chaotic Chicken Swarm Optimization Algorithm for Feature Selection" in *Proc. of the 2017 Third International Conference on Research in Computational Intelligence and Communication Networks (ICRCICN)*, 2017, pp. 259-264.
5. S. Liang, T. Feng, G. Sun, J. Zhang, H. Zhang. "Transmission Power Optimization for Reducing Sidelobe via Bat-chicken Swarm Optimization in Distributed Collaborative Beamforming" in *Proc. of the 2016 2nd IEEE International Conference on Computer and Communications (ICCC)*, 2016, pp. 2164-2168.
6. S. Liang, T. Feng, G. Sun. "Sidelobe-level suppression for linear and circular antenna arrays via the cuckoo search-chicken swarm optimisation algorithm" in *IET Microwaves, Antennas & Propagation*, vol. 11, 2017, pp. 209-218.
7. M. Han, S. Liu. "An Improved Binary Chicken Swarm Optimization Algorithm for Solving 0-1 Knapsack Problem" in *Proc. of the 2017*

*13th International Conference on Computational Intelligence and Security (CIS)*, 2017, pp. 207-210.

8. A. I. Hafez, H. M. Zawbaa, E. Emary, H. A. Mahmoud, A. E. Hassanien. "An Innovative Approach for Feature Selection Based on Chicken Swarm Optimization" in *Proc. of the 2015 7th International Conference of Soft Computing and Pattern Recognition (SoCPaR)*, 2015, pp. 19-24.
9. Roslina, M. Zarlis, I. T. R. Yanto, D. Hartama. "A Framework of Training ANFIS using Chicken Swarm Optimization for Solving Classification Problems" in *Proc. of the 2016 International Conference on Informatics and Computing (ICIC)*, 2016, pp. 437-441.
10. Y. Ji, Y. Wu, X. Sun, S. Yan, Q. Chen, B. Du. "A New Algorithm for Attitude Determination Based on Chicken Swarm Optimization" in *Proc. of the 2018 7th International Conference on Digital Home (ICDH)*, 2018, pp. 121-126.
11. D. Moldovan, V. Chifu, C. Pop, T. Cioara, I. Anghel, I. Salomie. "Chicken Swarm Optimization and Deep Learning for Manufacturing Processes" in *Proc. of the 2018 17th RoEduNet Conference: Networking in Education and Research (RoEduNet)*, 2018, pp. 1-6.
12. J. Wang, Z. Cheng, O. K. Ersoy, M. Zhang, K. Sun, Y. Bi. "Improvement and Application of Chicken Swarm Optimization for Constrained Optimization" in *IEEE Access*, vol. 7, 2019, pp. 58053-58072.