13

Firefly Algorithm

Xin-She Yang
School of Science and Technology
Middlesex University, London, United Kingdom

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

CONTENTS

13.1 Introduction 163
13.2 Original firefly algorithm 164
  13.2.1 Description of the standard firefly algorithm 164
  13.2.2 Pseudo-code of FA 165
  13.2.3 Parameters in the firefly algorithm 165
13.3 Source code of firefly algorithm in Matlab 166
13.4 Source code in C++ 168
13.5 A worked example 170
13.6 Handling constraints 172
13.7 Conclusion 173
References 173

13.1 Introduction

The original firefly algorithm (FA) was first developed by Xin-She Yang in late 2007 and early 2008 [1]; this mimics the main characteristics of flashing behaviour of tropical fireflies. Due to the rich characteristics of FA as a nonlinear system, it can solve multimodal optimization problems effectively [2, 3]. It can also deal with stochastic functions and non-convex problems [4, 5]. A detailed comparison by Senthilnath et al. [6] suggests that the FA can obtain the best results with the least amount of computing time for clustering. Other studies show that FA can solve various problems in different applications effectively, including software testing [7], modeling to generate design alternatives [8], and scheduling problems [9]. In addition, it is possible to enhance the performance of FA further by introducing other components such as chaotic maps [10].

The rest of the chapter provides all the fundamentals of the firefly algorithm with the main pseudo-code, and both Matlab and C++ demo codes.

13.2 Original firefly algorithm

For an optimization problem such as function optimization with an objective function $f(\boldsymbol{x})$, the decision variables $\boldsymbol{x}$ can be considered as a D-dimensional vector formed by D independent variables

$$\boldsymbol{x} = [x_1,\ x_2,\ x_3,\ ...,\ x_D], \tag{13.1}$$

where we have used a row vector. Obviously, it can be changed into a column vector by a transpose (T) action. The problem is to minimize an objective

$$\text{Minimize } f(\boldsymbol{x}), \quad \boldsymbol{x} \in \mathbb{R}^D. \tag{13.2}$$

This is an unconstrained optimization problem, which forms our starting point. Once we understand the main idea of the firefly algorithm and its implementation, we can easily extend it to solve constrained optimization problems by incorporating constraints properly using constraint-handling techniques to be discussed later.

13.2.1 Description of the standard firefly algorithm

The position of a firefly such as firefly i can be considered as a solution vector to an optimization problem. Thus, the movement of positions is equivalent to the search moves in the decision space. The search process is an iterative process with a pseudo-time counter t, starting with $t = 0$. For simplicity, we use the notation $\boldsymbol{x}_i^t$ to denote the position of firefly i at iteration t. Here t should not be confused with the exponent of an exponential function. In fact, in many textbooks, another notation $\boldsymbol{x}_i^{(t)}$ is commonly used. However, as both notations are popular, we will use $\boldsymbol{x}_i^t$ here as it causes no confusion in this context.

The main equation for the firefly algorithm is

$$\boldsymbol{x}_i^{t+1} = \boldsymbol{x}_i^t + \beta_0 e^{-\gamma r_{ij}^2}(\boldsymbol{x}_j^t - \boldsymbol{x}_i^t) + \alpha \boldsymbol{\epsilon}_i^t, \tag{13.3}$$

where β_0 is the attractiveness at zero distance and α is a scaling factor. In addition, γ is an absorption coefficient, and the distance r_{ij} is defined as

$$r_{ij} = ||\boldsymbol{x}_i - \boldsymbol{x}_j||_2 = \sqrt{\sum_{k=1}^{D} \big(x_{i,k} - x_{j,k}\big)^2}, \tag{13.4}$$

which is the Cartesian distance of two fireflies. Here, $x_{i,k}$ means the kth component or variable of the solution vector $\boldsymbol{x}_i$.

The vector $\boldsymbol{\epsilon}_i^t$ is a vector of random numbers that are normally distributed with a zero mean and unity variance. This means that this vector is drawn from a normal distribution $N(0, 1)$ and updated at every iteration.

The second term $\beta_0 \exp[-\gamma r_{ij}^2]$ is the attractiveness term, which mimics the attractiveness seen by two fireflies i and j. This term comes from the combination of the light intensity variation due to the inverse-square law with distance and exponential absorption in the media.

13.2.2 Pseudo-code of FA

The main steps of the FA consists of two loops over all fireflies. The updates of positions or solution vectors are done iteratively, and evaluations of new solutions are carried out within the loops. The pseudo-code for the firefly algorithm is presented in Algorithm 12.

Algorithm 12 Pseudo-code of the firefly algorithm.

```
1:  Define the objective function $f(\boldsymbol{x})$
2:  Set all the parameters $\alpha_0$, $\theta$, $\beta_0$ and $\gamma$
3:  Initialize the population of $n$ fireflies
4:  Set the iteration counter $t = 0$
5:  while ($t < t_{\max}$) do
6:      for $i = 1 : n$ do
7:          for $j = 1 : i$ do
8:              Calculate the distance $r_{ij} = ||\boldsymbol{x}_i - \boldsymbol{x}_j||$
9:              Draw a random vector $\boldsymbol{\epsilon}$ from a normal distribution
10:             Calculate the positions/solutions of fireflies $i$ and $j$
11:             Evaluate $f(\boldsymbol{x}_i)$ and $f(\boldsymbol{x}_j)$
12:             if $f(\boldsymbol{x}_i) < f(\boldsymbol{x}_j)$ then
13:                 Move firefly $j$ towards $i$ (for minimization) using Eq. (13.3)
14:             end if
15:         end for
16:     end for
17:     Rank the firefly population and find the current best solution $\boldsymbol{g}_*$
18: end while
19: Output the best solution
20: Visualize the results
```

13.2.3 Parameters in the firefly algorithm

There are three parameters in the FA, and they are: α, β_0 and γ.

The attraction strength or attractiveness β of two fireflies is governed by

$$\beta = \beta_0 \exp[-\gamma r_{ij}^2]. \tag{13.5}$$

Obviously, $\beta = \beta_0$ if $r_{ij} = 0$. If γ is too small (i.e., $\gamma \to 0$), we have $\beta \to \beta_0$, which means that the light intensity or attractiveness remains close to a constant β_0. The visibility distance is large, the nonlinear term in Eq. (13.3) becomes linear. This corresponds to the case where a flashing firefly can be seen by all the other fireflies in the search space. This also means that collective information is used by all the fireflies. On the other hand, if $\gamma \to +\infty$, then $\beta \to 0$. The visibility distance for fireflies is short, which corresponds to a situation in a dense fog where one firefly cannot see other fireflies. Thus, the system becomes highly nonlinear, and each firefly moves by performing local random walks. This means that the exploitation ability of the firefly system is at the minimum, but its exploration ability is at the maximum.

So a simple rule of setting γ is to allow $\gamma r_{ij}^2 = 1$. That is

$$\gamma = \frac{1}{L^2}, \tag{13.6}$$

where L is the average length scale of the problem. For example, if a variable varies from -5 to $+5$, its scale is $L = 10$, so we can use $\gamma = 1/10^2 = 0.01$.

The attractiveness β_0 can be taken as $\beta_0 = 1$ for most applications, while $\gamma > 0$ should relate to the scales of the modes of the problem. If there is no prior knowledge of the problem modality, $\gamma = 0.01$ to 1 can be used first.

The scaling factor α in the third term controls the step size of the movements of the fireflies. If α is large, the steps are large, which can explore a large space; however, the solution generated may be too far away from the current region, even potentially jumping out of the search region. If α is too small, the steps are also small, which limits the exploration ability, and the moves become primarily local. Thus, there is a fine balance: the steps should not be too large or too small. Overall, the third term in Eq. (13.3) controls the randomness in the algorithm. It can be expected that the overall random components should become smaller and smaller as the population may move towards the true optimal position. Thus, ideally, α should be gradually reduced. For example, we can use

$$\alpha = \alpha_0 \theta^t, \tag{13.7}$$

where $0 < \theta < 1$ is a constant, and α_0 is the initial value of α. In most cases, we can use $\theta = 0.9$ to 0.99 and $\alpha_0 = 1$.

13.3 Source code of firefly algorithm in Matlab

As the algorithm is simple with only one main governing equation, it is straightforward to implement in any programming language. Here, we present a detail description of a simple Matlab implementation to find the minimum of the function $f(\boldsymbol{x})$

$$\text{minimize} f(\boldsymbol{x}) = (x_1 - 1)^2 + (x_2 - 1)^2 + ... + (x_D - 1)^2, \quad x_i \in \mathbb{R}, \tag{13.8}$$

which has a global minimum of $\boldsymbol{x}_* = (1, 1, ..., 1)$. Though the algorithm can search the whole domain, it would be more realistic if we impose some simple regular bounds. For example, for the above function to be optimized, we can use the following lower bound (Lb) and upper bound (Ub):

$$Lb = [-5,\ -5,\ ...,\ -5], \quad Ub = [+5,\ +5,\ ...,\ +5]. \tag{13.9}$$

The Matlab code here consists of three parts: initialization, main loops, and the objective function. These lines of codes can simply be put together line by line in sequential order, which should run smoothly to give the optimal solution of the minimum $f(\boldsymbol{x})$.

The first part is mainly initialization of parameter values such as α_0, β_0 and γ, as well as the generation of the initial population of $n = 20$ fireflies. The cost function is the objective function to be given later in a Matlab function.

```
function fa_demo % Start the FA demo
n=20;            % Population size (number of fireflies)
alpha=1.0;       % Randomness 0--1 (highly random)
beta0=1.0;       % beta value
gamma=0.1;       % Absorption coefficient
theta=0.97;      % Randomness reduction factor for alpha
d=10;            % Number of dimensions
tMax=500;        % Maximum number of iterations
Lb=-5*ones(1,d); % Lower bounds/limits
Ub=5*ones(1,d);  % Upper bounds/limits
% Generating the initial locations of n fireflies
   for i=1:n,
   ns(i,:)=Lb+(Ub-Lb).*rand(1,d);
   Lightn(i)=cost(ns(i,:));   % Evaluate objectives
   end
```

Listing 13.1
FA demo initialization.

The second part is the main part, consisting of two loops and the update of the firefly position vectors by the algorithmic equation (13.3). Then, the solutions are ranked according to their fitness or objective values. The best solution is found. In addition, new solutions are checked to see if they are within the simple bounds.

```
for k=1:tMax,         %%%%% start iterations %%%%%
 alpha=alpha*theta;   % Reduce alpha by a factor theta
 scale=abs(Ub-Lb);    % Scaling of the system
% Two loops over all fireflies
for i=1:n,
  for j=1:n,
      % Evaluate the objective values of current solutions
      Lightn(i)=cost(ns(i,:));   % Call the objective
      % Update moves
      if Lightn(i)>Lightn(j),    % Brighter/more attractive
      r=sqrt(sum((ns(i,:)-ns(j,:)).^2));
      beta=beta0*exp(-gamma*r.^2);   % Attractiveness
      steps=alpha.*(rand(1,d)-0.5).*scale;
      % The FA update equation
      ns(i,:)=ns(i,:)+beta*(ns(j,:)-ns(i,:))+steps;
      end
   end % end for j
end % end for i
% Check if the updated solutions/locations are within limits
```

```
ns=findlimits(n,ns,Lb,Ub);
%% Ranking fireflies by their light intensity/objectives
[Lightn,Index]=sort(Lightn);
ns_tmp=ns;
for i=1:n,
 ns(i,:)=ns_tmp(Index(i),:);
end
%% Find the current best and display outputs
fbest=Lightn(1), nbest=ns(1,:)
end % End of t loop (up to tMax)
```

Listing 13.2
The main part of the firefly algorithm.

The third part is the objective function and ways for implementing the lower and upper bounds. This will ensure that all solution vectors should be within the regular bounds, and any new solution can be evaluated by calling the objective or cost function.

```
% Make sure the fireflies are within the bounds/limits
function [ns]=findlimits(n,ns,Lb,Ub)
for i=1:n,
    ns_tmp=ns(i,:);
  % Apply the lower bound
  I=ns_tmp<Lb;  ns_tmp(I)=Lb(I);
  % Apply the upper bounds
  J=ns_tmp>Ub;  ns_tmp(J)=Ub(J);
  % Update this new move
  ns(i,:)=ns_tmp;
end
%% Cost or Objective function
function z=cost(x)
z=sum((x-1).^2); % Solutions should be (1,1,...,1)
```

Listing 13.3
Variable limits and objective function.

13.4 Source code in C++

The same FA steps can be implemented in C++. In order to be consistent, the same function and limits are used, and the C++ code is given as follows:

```
#include <iostream>
#include <time.h>
#include <tgmath.h>
#include <algorithm>
using namespace std;

// Definition of the objective function OF(.)
float cost(float x[], int size_array)
{
float t=0;
for(int i=0; i<size_array; i++){
    t=t+(x[i]-1)*(x[i]-1);}
return t;
}
// Generate pseudo random values from the range [0, 1)
```

```
float ra(){return (float)(rand()%1000)/1000;}

int main()
{
int n=20;//Population size (number of fireflies)
float alpha=1.0;  //Randomness 0--1 (highly random)
float beta0=1.0;  //beta value
float gamma=0.1;  //Absorption coefficient
float theta=0.97; //Randomness reduction factor for alpha
int d=10;         //Number of dimensions
int tMax=500;     //Maximum number of iterations
float Lb[d], Ub[d]; //Lower bounds/limits, Upper bounds/limits
float ns[n][d], ns_tmp[n][d];
float Lightn[n], Lightn_tmp[n], fbest, nbest[d];
float scale[d], r, beta, sum, steps[d];
// Initialization of the pseudo-random generator
srand(time(NULL));
// Initialization of the bounds
for(int j=0; j<d; j++){Lb[j]=-5; Ub[j]=5;}
// Generating the initial locations of n fireflies
for(int i=0; i<n; i++)
{
    for(int j=0; j<d; j++)
    {
        ns[i][j]=Lb[j]+(Ub[j]-Lb[j])*ra();
    }
    Lightn[i]=cost(ns[i],d); //Evaluate objectives
}
//Iterations for pseudo-time marching
for(int k=0; k<tMax; k++)    //Start iterations
{
    alpha=alpha*theta;
    //scaling of the system
    for(int i=0; i<d; i++)
    {
        scale[i]=fabs(Ub[i]-Lb[i]);
    }
    //updating fireflies
    for(int i=0; i<n; i++)
    {
        for(int j=0; j<n; j++)
        {   //Evaluate objective value of current solutions
                    Lightn[i]=cost(ns[i],d); //Call the objective
            //Update moves
            if(Lightn[i]>Lightn[j]) //Brighter and more attractive
            {
                sum=0;
                for(int h=0; h<d; h++)
                {
                    sum=sum+(ns[i,h]-ns[j,h])*(ns[i,h]-ns[j,h]);
                }
                r=sqrt(sum);
                beta=beta0*exp(-gamma*r*r); //Attractiveness
                for(int h=0; h<d; h++)
                {
                    steps[h]=alpha*(ra() -0.5)*scale[h];
                }
                //The FA update equation
                for(int h=0; h<d; h++)
                {
                  ns[i][h]=ns[i][h]+beta*(ns[j][h]-ns[i][h])+steps[h];
                }
            } // end if
        } // end for j
    } // end for i
//Check if the updated solution/locations are within limits
for(int i=0; i<n; i++)
{
    for(int j=0; j<d; j++)
    {
        if (ns[i][j]<Lb[j]){ns[i][j]=Lb[j];}
        if (ns[i][j]>Ub[j]){ns[i][j]=Ub[j];}
    }
}
//Ranking fireflies by their light intensity/objectives
for (int i=0; i<n; i++)
{
    for(int j=0; j<d; j++)
    {
        ns_tmp[i][j]=ns[i][j];
    }
    Lightn_tmp[i]=Lightn[i];
}
float worst_fit=*max_element(Lightn_tmp,Lightn_tmp+d)+1;
for(int i=0; i<n; i++)
{
    int best_fit=min_element(Lightn_tmp, Lightn_tmp+d)-Lightn_tmp;
    for(int j=0; j<d; j++)
    {
        ns[i][j]=ns_tmp[best_fit][j];
    }
    Lightn[i]=Lightn_tmp[best_fit];
    Lightn_tmp[best_fit]=worst_fit;
}
fbest=Lightn[0];
for(int i=0; i<d; i++)
{
    nbest[i]=ns[0][i];
}
} // End for k
cout<<"#### fbest: "<<fbest<<endl;
cout<<"#### nbest: [ ";
for(int i=0; i<d; i++){cout<<nbest[i]<<" ";}
cout<<"]"<<endl;
getchar();
return 0;
}
```

Listing 13.4
Firefly algorithm in C++.

13.5 A worked example

Let us use the firefly algorithm to find the minimum of

$$f(\boldsymbol{x}) = (x_1 - 1)^2 + (x_2 - 1)^2 + (x_3 - 1)^2, \tag{13.10}$$

in the simple ranges of $-5 \le x_i \le 5$. For a purely demonstrative purpose, we only use $n = 5$ fireflies. Suppose the initial population is randomly initialized and their objective (fitness) values are as follows:

$$\begin{cases} \boldsymbol{x}_1 = \begin{pmatrix} 2.00 & 2.00 & 3.00 \end{pmatrix}, & f_1 = f(\boldsymbol{x}_1) = 6.00, \\ \boldsymbol{x}_2 = \begin{pmatrix} 5.00 & 0.00 & 5.00 \end{pmatrix}, & f_2 = f(\boldsymbol{x}_2) = 33.00, \\ \boldsymbol{x}_3 = \begin{pmatrix} -3.00 & -2.00 & 0.00 \end{pmatrix}, & f_3 = f(\boldsymbol{x}_3) = 26.00, \\ \boldsymbol{x}_4 = \begin{pmatrix} -5.00 & 0.00 & 5.00 \end{pmatrix}, & f_4 = f(\boldsymbol{x}_4) = 53.00, \\ \boldsymbol{x}_5 = \begin{pmatrix} 3.00 & 4.00 & 5.00 \end{pmatrix}, & f_5 = f(\boldsymbol{x}_5) = 29.00, \end{cases} \tag{13.11}$$

where we have only used two decimal places for simplicity.

The best solution with the lowest (or best) value of the objective function is $\boldsymbol{x}_1$ with $f_1 = 6.00$ for this population at $t = 0$. We did this intentionally so that $\boldsymbol{x}_1$ is the best solution. Otherwise, we should re-arrange the order of the population so that the first solution is the best solution.

For simplicity, we can use $\alpha_0 = 1$, $\beta_0 = 1$, $\theta = 0.97$ and $\gamma = 0.1$, though these parameters are not good and they are not used in the actual Matlab implementation. However, such settings allow us to show and calculate new solutions easily so that we can focus on the main principle and procedure of the firefly algorithm.

The first loop is to compare each pair of the fireflies, and update their positions at $t = 1$.

- By comparing $\boldsymbol{x}_2$ and $\boldsymbol{x}_1$, we know that $\boldsymbol{x}_1$ is better because $f_1 < f_2$, so we should move $\boldsymbol{x}_2$ towards $\boldsymbol{x}_1$ by

$$\boldsymbol{x}_2^{t+1} = \boldsymbol{x}_2 + \beta_0 e^{-\gamma r_{21}^2}(\boldsymbol{x}_1 - \boldsymbol{x}_2) + \alpha\epsilon_2, \tag{13.12}$$

where all the values on the right-hand side are values at iteration t.

For simplicity, let us draw a random vector and we get $\epsilon_2 = [-0.50.40.1]$. Since the distance between $\boldsymbol{x}_1 = [2, 2, 3]$ and $\boldsymbol{x}_2 = [5, 0, 5]$ is

$$r_{21} = \sqrt{(2-3)^2 + (2-0)^2 + (3-5)^2} = \sqrt{17}, \tag{13.13}$$

we have

$$\boldsymbol{x}_2^{t+1} = \begin{pmatrix} 5 \\ 0 \\ 5 \end{pmatrix}^T + 1 \times e^{-0.1\times(\sqrt{17})^2}\left[\begin{pmatrix} 2 \\ 2 \\ 3 \end{pmatrix}^T - \begin{pmatrix} 5 \\ 0 \\ 5 \end{pmatrix}^T\right] + 0.97 \times \begin{pmatrix} -0.5 \\ 0.4 \\ 0.1 \end{pmatrix}^T$$
$$= \begin{pmatrix} 3.97 \\ 0.75 \\ 4.73 \end{pmatrix}^T, \tag{13.14}$$

where the transpose turns a row vector into a column vector. This new $\boldsymbol{x}_2$ at $t + 1$ gives the objective value

$$f(\boldsymbol{x}_2^{t+1}) = 22.79, \tag{13.15}$$

which is better than the original f_2. Even though the new objective 22.79 is still higher than $f_1 = 6$, the move from $\boldsymbol{x}_2$ to $\boldsymbol{x}_2^{t+1}$ is an improvement over the original f_2, and this move should be accepted. Therefore, the new solution $\boldsymbol{x}_2$ should be immediately updated as

$$\boldsymbol{x}_2^{t+1} = \begin{pmatrix} 3.97 & 0.75 & 4.73 \end{pmatrix}, \quad f_2^{t+1} = 22.79. \tag{13.16}$$

- Now we do a similar update by comparing $\boldsymbol{x}_3$ and $\boldsymbol{x}_1$. The distance r_{31} is

$$r_{31} = \sqrt{(-3-2)^2 + (-2-2)^2 + (0-3)^2} = \sqrt{50}. \tag{13.17}$$

If we draw $\epsilon_3 = [-0.2, 0.1, -0.5]$, we have

$$\boldsymbol{x}_3^{t+1} = \begin{pmatrix} -3 \\ -2 \\ 0 \end{pmatrix}^T + 1 \times e^{-0.1 \times (\sqrt{50})^2} \left[\begin{pmatrix} 2 \\ 2 \\ 3 \end{pmatrix}^T - \begin{pmatrix} -3 \\ -2 \\ 0 \end{pmatrix}^T \right] + 0.97 \times \begin{pmatrix} -0.2 \\ 0.1 \\ -0.5 \end{pmatrix}^T$$

$$= \begin{pmatrix} -3.16 \\ -1.88 \\ -0.46 \end{pmatrix}^T, \tag{13.18}$$

which gives $f(\boldsymbol{x}_3^{t+1}) = 27.73$, which is higher than the original $f_3 = 26.00$, which means that we should not update this move.

- We use a loop over the rest of the population such as $\boldsymbol{x}_4$ and $\boldsymbol{x}_5$, and update them in a similar manner. If the new moves jump out of the range, we compare them with the nearest bounds and force the new solutions to be within the simple domain bounded by Lb and Ub.

Once the population has been updated once, the solutions are ranked according to their objective values so that the first solution $\boldsymbol{x}_1^{t+1}$ should be the best solution. Then, a new round is carried out by setting $t \leftarrow t+1$. Once a predefined maximum number of iterations such as $t_{\max} = 1000$, the best solution to the problem under consideration can be obtained.

In general, the firefly algorithm can be efficient. For the objective function, the optimal solution $\boldsymbol{x}_* = [1, 1, 1]$ can be obtained within about 100 iterations. For example, in one run after 100 iterations with a population $n = 20$, the best solution we got is

$$\boldsymbol{x}_1^{100} = \begin{pmatrix} 1.020 & 0.997 & 1.001 \end{pmatrix}, \quad f(\boldsymbol{x}_1^{100}) = 4.1 \times 10^{-4}. \tag{13.19}$$

After 200 iterations with the same population, we can get $f_{\min} = 2.9 \times 10^{-7}$. Similarly, after 500 iterations, we can typically get $f_{\min} = 2.3 \times 10^{-14}$.

13.6 Handling constraints

It is worth pointing out that the above implementations or demo codes are mainly for unconstrained optimization problems. For solving nonlinear constrained optimization problems, constraints should be handled properly using constraint-handling techniques such as penalty methods and the method of Lagrangian multipliers. In the simplest cases, a penalty parameter can be introduced so as to incorporate all the constraints into the modified objective; thus the constrained problem becomes a corresponding unconstrained one.

In general, a mathematical optimization problem in a D-dimensional design space can be written as

$$\text{minimize} \quad f(\boldsymbol{x}), \quad \boldsymbol{x} = (x_1, x_2, ..., x_D) \in \mathbb{R}^D, \tag{13.20}$$

subject to

$$\phi_i(\boldsymbol{x}) = 0, \quad (i = 1, 2, ..., M), \tag{13.21}$$

$$\psi_j(\boldsymbol{x}) \leq 0, \quad (j = 1, 2, ..., N), \tag{13.22}$$

where $\boldsymbol{x}$ is the vector of D design variables, and $\phi_i(\boldsymbol{x})$ and $\psi_j(\boldsymbol{x})$ are the equality constraints and inequality constraints, respectively. The penalty-based method transforms the objective $f(\boldsymbol{x})$ into a modified objective Θ in the following form:

$$\Theta(\boldsymbol{x}) = f(\boldsymbol{x})[\text{objective}] + P(\boldsymbol{x})[\text{penalty}], \tag{13.23}$$

where the penalty term $P(\boldsymbol{x})$ can take different forms, depending on the actual ways or variants of constraint-handling methods. For example, a static penalty method uses

$$P(\boldsymbol{x}) = \lambda \Big[\sum_{i=1}^{M} \phi_i^2(\boldsymbol{x}) + \sum_{j=1}^{N} \max\{0, \psi_j(\boldsymbol{x})\}^2 \Big]. \tag{13.24}$$

Since $\lambda > 0$ is fixed, independent of the iteration t, we can extend the above Matlab code for unconstrained problems to solve this type of problem.

13.7 Conclusion

The firefly algorithm is a simple, flexible and yet efficient algorithm for solving multimodal optimization problems. As we have seen from the above explanation of the main steps, the implementation is relatively straightforward, which requires a minimum amount of memory and computation costs.

This algorithm has been extended to other forms with many variants. Interested readers can refer to more advanced literature [3, 11].

References

1. X.S. Yang, *Nature-Inspired Metaheuristic Algorithms*, Luniver Press, UK, 2008.
2. X.S. Yang, "Firefly algorithm for multimodal optimisation", in *Proceedings of 5th Symposium on Stochastic Algorithms, Foundation and Applications*, Lecture Notes in Computer Science, volume 5792, pp. 169-178, 2009.

3. I. Fister, I. Fister Jr., X.S. Yang, J. Brest, "A comprehensive review of firefly algorithms". *Swarm and Evolutionary Computation*, 13(1):34-46, 2013.
4. X.S. Yang, "Firefly algorithm, stochastic test functions and design optimisation", *Int. Journal of Bio-Inspired Computation*, 2(2): 78-84, 2010.
5. X.S. Yang, S.S. Hosseini, A.H. Gandomi, "Firefly algorithm for solving non-convex economic dispatch problems with valve loading effect", *Applied Soft Computing*, 12(3): 1180–1186, 2012.
6. J. Senthilnath, S.N. Omkar, V. Mani, "Clustering using firefly algorithm: performance study", *Swarm and Evolutionary Computation*, 1(3): 163-171, 2011.
7. P. R. Srivastava, B. Mallikarjun, X.S. Yang, "Optimal test sequence generation using firefly algorithm", *Swarm and Evolutionary Computation*, 8(1): 44-53, 2013.
8. R. Imanirad, X.S. Yang, J.S. Yeomans, "Modeling-to-generate-alternatives via the firefly algorithm", *J. Appl. Oper. Res.*, 5(1): 14-21, 2013.
9. A. Yousif, A.H. Abdullah, S.M. Nor, A. Abdelaziz, "Scheduling jobs on grid computing using firefly algorithm", *J. Theor. Appl.Inform. Technol.*, 33(2): 155-164, 2011.
10. A.H. Gandomi, X.S. Yang, S. Talatahari, A. H. Alavi, "Firefly algorithm with chaos", *Commun. Nonlinear Sci. Numer. Simulation*, 18(1): 89-98, 2013.
11. X.S. Yang, *Nature-Inspired Optimization Algorithms*, Elsevier, 2014.