4

Bat 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

4.1 Introduction

The original bat algorithm (BA) was first developed by Xin-She Yang in 2010 [1], which attempts to mimic the main characteristics of echolocation of microbats. Due to its good convergence, it has been applied to solve various optimization problems [2], and it has also been extended to solve a multi-objective optimization problem [3] and chaotic variants [4]. Its applications are diverse [5, 6], and a detailed review about the earlier literature was first given by Yang and He in 2013 [2]. In addition, many variants have been developed to improve its efficiency in various ways, including a hybrid bat algorithm [7], directional bat algorithm [8, 9], discrete bat algorithm [10], and others.

The global convergence of the standard bat algorithm has recently been proved mathematically using both a Markovian framework and dynamical system theory [11].

The rest of this chapter provides all the fundamentals of the bat algorithm with the main pseudo-code, Matlab code and C++ code, followed by some simple demonstration of how it works.

4.2 Original bat algorithm

Bats, especially microbats, use echolocation for navigation, which consists of emitting a series of short, ultrasonic pulses that typically last a few milliseconds with frequencies in the range of 25 kHz to about 150 kHz. The loudness of such bursts can be up to 110 dB. When homing for prey, microbats typically increase their pulse emission rates and frequencies, which is also accompanied by variation of their loudness. The main purpose of such frequency tuning and echolocation is for navigation and hunting so as to increase the accuracy of detection and success rate of capturing the prey. Such characteristics are simulated in the bat algorithm.

4.2.1 Description of the bat algorithm

For a population of n bats, each bat (say, bat i) is associated with a location or position vector $\boldsymbol{x}_i$ and a corresponding velocity vector $\boldsymbol{v}_i$. During the iterations, each bat can vary its pulse emission rate r_i, loudness A_i and its frequency f_i.

In general, a position vector is considered as a solution vector to an optimization problem in a D-dimensional search space with D independent design variables

$$\boldsymbol{x} = [x_1,\ x_2,\ x_3,\ ...,\ x_D], \tag{4.1}$$

which will vary with the iteration or a pseudo-time counter t. So, we use the notation $\boldsymbol{x}_i^t$ to denote the position of bat i at iteration t. Its corresponding velocity is thus denoted by $\boldsymbol{v}_i^t$.

Frequency tuning can be carried out by

$$f_i = f_{\min} + \beta(f_{\max} - f_{\min}), \tag{4.2}$$

where $f_{\min}$ and $f_{\max}$ are the minimum and maximum ranges, respectively, of the frequency f_i for each bat i. Here, β is a uniformly distributed number drawn from $[0, 1]$. In most applications, we should use $f_i = O(1)$. For example, we can start with $f_{\min} = 0$ and $f_{\max} = 2$.

This frequency variation is then used for updating the velocity of each bat

$$\boldsymbol{v}_i^{t+1} = \boldsymbol{v}_i^t + (\boldsymbol{x}_i^t - \boldsymbol{x}_*)f_i, \tag{4.3}$$

where $\boldsymbol{x}_*$ is the best solution found at iteration t.

The position or solution vector $\boldsymbol{x}_i$ is updated by

$$x_i^{t+1} = \boldsymbol{x}_i^t + (\Delta t)\boldsymbol{v}_i^{t+1}, \tag{4.4}$$

where Δt is the iteration or time increment. As all iterative algorithms are updated in a discrete manner, we usually set $\Delta t = 1$. Thus, we can simply consider the vectors without any physical units, and then write the update equation as

$$x_i^{t+1} = \boldsymbol{x}_i^t + \boldsymbol{v}_i^{t+1}. \tag{4.5}$$

The pulse emission rate r_i can monotonically increase from a lower value $r_i^{(0)}$, while the loudness can reduce from a higher value $A^{(0)} = 1$. We have

$$A_i^{t+1} = \alpha A_i^t, \quad r_i^{t+1} = r_i^{(0)}[1 - \exp(-\gamma t)], \tag{4.6}$$

where $0 < \alpha < 1$ and $\gamma > 0$ are constants. When t is large enough ($t \to \infty$), we have $A_i^t \to 0$ and $r_i^t \to r_i^{(0)}$. To start with the simple demonstration later, we can use $\alpha = 0.97$ and $\gamma = 0.1$.

For the local modification around a member of the best solution, we can use

$$\boldsymbol{x}_{\text{new}} = \boldsymbol{x}_{\text{old}} + \sigma\epsilon_t A^{(t)}, \tag{4.7}$$

where ϵ_t is a random number drawn from a normal distribution N(0,1), and σ is the standard deviation acting as a scaling factor. For simplicity, we can use $\sigma = 0.1$ in our later implementation. Here, $A^{(t)}$ is the average loudness at iteration t.

4.2.2 Pseudo-code of BA

Based on the above descriptions, the main steps of the BA consists of a single loop with some probabilistic switching during the iteration. Thus, the procedure of the bat algorithm is summarized in the pseudo-code shown in Algorithm 3.

4.2.3 Parameters in the bat algorithm

Apart from the population size n, there are two parameters and four initialization values in the bat algorithm. They are α, γ, $f_{\min}, f_{\max}, A_i^{(0)}$ and $r_i^{(0)}$. In our implementation, we have used $\alpha = 0.97$, $\gamma = 0.1$, $f_{\min} = 0$, $f_{\max} = 2$, and $A_i^{(0)} = r_i^{(0)} = 1$. Though n should vary for different applications, we have used $n = 10$ here.

Algorithm 3 Pseudo-code of the bat algorithm.

```
Define the objective function f(x)
Initialize the bat population x_i and v_i (i = 1, 2, ..., n)
Initialize frequencies f_i, pulse rates r_i and loudness A_i
Set the iteration counter t = 0
while (t < t_max) do
    Vary r_i and A_i
    Generate new solutions by adjusting frequencies
    Update velocities and locations/solutions via Eqs (4.3) and (4.5)
    if rand > r_i then
        Select a solution among the best solutions
        Generate a local solution around the selected best solution
    end if
    Generate a new solution by flying randomly
    if rand > A_i and f(x_i) < f(x_j) then
        Accept the new solution
    end if
    Rank the bats and find the current best solution x_*
end while
```

4.3 Source code of bat algorithm in Matlab

The implementation of this algorithm is relatively straightforward using Matlab. Here, we present a detailed description of a simple Matlab implementation to find the minimum of the function $f(\boldsymbol{x})$

$$\text{minimize} f(\boldsymbol{x}) = (x_1 - 1)^2 + (x_2 - 1)^2 + ... + (x_D - 1)^2, \quad x_i \in \mathbb{R}, \tag{4.8}$$

which has a global minimum of $\boldsymbol{x}_* = (1, 1, ..., 1)$. In theory, the algorithm can search the whole domain; we always impose some limits in practice. Here, for simplicity, we use the following lower bound (Lb) and upper bound (Ub):

$$Lb = [-5, \ -5, \ ..., \ -5], \quad Ub = [+5, \ +5, \ ..., \ +5]. \tag{4.9}$$

The Matlab code for the bat algorithm here consists of three parts: initialization, the main loop, and the objective function. The whole codes should consist of all the lines of codes in a sequential order. However, for ease of understanding, we split them into three parts.

The first part is mainly initialization of parameter values such as α and γ, as well as the generation of the initial population of $n = 10$ bats. The cost function is the objective function to be given later in a Matlab function.

It is worth pointing out that the simple bounds should be checked at each iteration when new solutions are generated. However, for simplicity, the

implementation here does not carry out this check, and care should be taken when extending the simple computer code here.

```
function [best,fmin]=bat_algorithm
% Default parameters
n=10;            % Population size, typically 10 to 25
A=1;             % Loudness  (constant or decreasing)
r=1;             % Pulse rate (constant or decreasing)
alpha=0.97;      % Parameter alpha
gamma=0.1;       % Parameter gamma
% Frequency range
Freq_min=0;      % Frequency minimum
Freq_max=2;      % Frequency maximum
% Max number of iterations
t_max=1000;
t=0;             % Initialize iteration counter
% Dimension of the search variables
d=5;
% Initialization of arrays
Freq=zeros(n,1);    % Frequency
v=zeros(n,d);       % Velocities
Lb=-5*ones(1,d);    % Lower bounds
Ub=5*ones(1,d);     % Upper bounds
% Initialize the population/solutions
for i=1:n,
  Sol(i,:)=Lb+(Ub-Lb).*rand(1,d);
  Fitness(i)=Fun(Sol(i,:));
end
% Find the best solution of the initial population
[fmin,I]=min(Fitness);
best=Sol(I,:);
```

Listing 4.1
BA demo initialization.

The second part is the main part, consisting a loop over the whole population at each iteration. The pulse emission rate and loudness are varied first, then new solutions are generated and evaluated. The new population is checked and the current best solution is found and updated. Though each bat can have its own pulse emission rate r_i and loudness A_i, we have used the same values of r and A for all bats. This means that the average of A_i is now simply A itself.

```
% Start the iterations -- Bat Algorithm
while (t<t_max)
   % Varying parameters
   r=r*(1-exp(-gamma*t));
   A=alpha*A;
   % Loop over all bats/solutions
 for i=1:n,
   Freq(i)=Freq_min+(Freq_max-Freq_min)*rand;
   v(i,:)=v(i,:)+(Sol(i,:)-best)*Freq(i);
   S(i,:)=Sol(i,:)+v(i,:);
   % Check a switching condition
   if rand>r,
   S(i,:)=best+0.1*randn(1,d)*A;
   end

   % Evaluate new solutions
   Fnew=Fun(S(i,:));
   % If the solution improves or not to loud
    if (Fnew<=Fitness(i)) & (rand>A) ,
      Sol(i,:)=S(i,:);
```

```
        Fitness(i)=Fnew;
      end

    % Update the current best solution
      if Fnew<=fmin,
        best=S(i,:);
        fmin=Fnew;
      end
    end % end of for
    t=t+1;  % Update iteration counter
end
% Output the best solution
disp(['Best =',num2str(best),' fmin=',num2str(fmin)]);
```

Listing 4.2
Main iterations for the bat algorithm.

The third part is the objective function and ways for implementing the lower and upper bounds. This will ensure the solution vector should be within the regular bounds; however, we have not implemented the bounds here for simplicity. A new solution is evaluated by calling the objective or cost function.

```
%% Cost or objective function
function z=cost(x)
z=sum((x-1).^2); % Solutions should be (1,1,...,1)
```

Listing 4.3
The objective function.

If we run this code with a maximum number of 1000 iterations, we can get the best minimum value as $f_{best} = 3.49 \times 10^{-29}$. Obviously, due to the random numbers used, these results are not exactly repeatable, but the order of the magnitude is easily reachable in the simulation.

4.4 Source code in C++

As the bat algorithm is simple, it can be implemented easily in any other programming language. For the same algorithm to solve the same problem as given above in Matlab, we can implement it in C++ as follows:

```
#include <iostream>
#include <time.h>
#include <random>
using namespace std;
//Definition of the objective function OF(.)
float Fun(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 value from the range [0, 1)
float ra() {return (float)rand()/RAND_MAX;}
int main()
{
```

```
16 //Normal distribution generator
17 std::default_random_engine generator;
18 std::normal_distribution<double> distribution(0,1.0);
19 //Initialization of pseudo random generator
20 srand(time(NULL));
21 int n=10;          //Population size, typically 10 to 25
22 float A=1.0;       //Loudness (constant or decreasing)
23 float r=1.0;       //Pulse rate (constant or increasing)
24 float alpha=0.97;//Parameter alpha
25 float gamma=0.1; //Parameter gamma
26 //Frequency range
27 float Freq_min=0.0;  //Frequency minimum
28 float Freq_max=2.0;  //Frequency maximum
29 //Max number of iterations
30 int t_max=1000;
31 //Dimension of the search variables
32 int d=5;
33 //Initialization of arrays
34 float Freq[n];    //Frequency
35 float v[n][d];    //Velocities
36 for(int i=0; i<n; i++)
37 {
38     for(int j=0; j<d; j++){v[i][j]=0;}
39 }
40 float Lb[d];  //Lower bounds
41 float Ub[d];  //Upper bounds
42 float Sol[n][d];    //Solutions
43 float Fitness[n];   //Fitness
44 float S[n][d];      //New solutions
45 float rnum;
46 float Fnew;
47 //Prepare arrays
48 for(int j=0; j<d; j++) {Lb[j]=-5.0; Ub[j]=5.0;}
49 //Initialize the population/solutions
50 for(int i=0; i<n;i++)
51 {
52     for(int j=0; j<d; j++)
53     {
54         Sol[i][j]=Lb[j]+(Ub[j]-Lb[j])*ra();
55     }
56     Fitness[i]=Fun(Sol[i],d);
57 }
58 //Find the best solution of the initial population
59 int I=0; float fmin;
60 fmin=Fitness[I];
61 for(int i=1; i<n; i++)
62 {
63     if(Fitness[i]<=fmin)
64     {
65         fmin=Fitness[i];
66         I=i;
67     }
68 }
69 float best[d];
70 for(int i=0; i<d; i++)
71     {
72         best[i]=Sol[I][i];
73     }
74 //Start the iterations -- Bat Algorithm
75 for(int t=1; t<=t_max; t++)
76 {
77     //Varying parameters
78     r=r*(1-exp(-gamma*t));
79     A=alpha*A;
80     //Loop over all bats/solutions
81     for(int i=0; i<n; i++)
82     {
```

```
83          Freq[i]=Freq_min+(Freq_max-Freq_min)*ra();
84          rnum=ra();
85          for(int j=0; j<d; j++)
86          {
87              v[i][j]=v[i][j]+(Sol[i][j]-best[j])*Freq[i];
88              S[i][j]=Sol[i][j]+v[i][j];
89              if (rnum>r)
90              {
91                  S[i][j]=best[j]+0.1*A*distribution(generator);
92                              }
93          }
94          //Evaluate new solutions
95          Fnew=Fun(S[i],d);
96          //If the solution improves or not to loud
97          if ((Fnew<=Fitness[i]) && (ra()>A))
98          {
99              Fitness[i]=Fnew;
100             for(int j=0; j<d; j++)
101             {
102                 Sol[i][j]=S[i][j];
103             }
104         }
105         //Update the current best solution
106         if (Fnew<=fmin)
107         {
108             fmin=Fnew;
109             for(int j=0; j<d; j++)
110             {
111                 best[j]=S[i][j];
112             }
113         }
114     }
115 }
116 //Output the best solution
117 cout<<"#### fmin = "<<fmin<<endl;
118 cout<<"#### Best = [ ";
119 for(int i=0; i<d; i++) {cout<<best[i]<<" ";}
120 cout<<"]"<<endl;
121 getchar();
122 return 0;
123 }
```

Listing 4.4
Bat algorithm in C++.

4.5 A worked example

Let us use the bat algorithm to find the minimum of

$$f(\boldsymbol{x}) = (x_1 - 1)^2 + (x_2 - 1)^2 + (x_3 - 1)^2, \tag{4.10}$$

in the simple ranges of $-5 \leq x_i \leq 5$. This simple 3-variable problem has the global minimum solution $\boldsymbol{x}_{\rm best} = [1, 1, 1]$.

For the purpose of algorithm demonstration, we only use $n = 5$ bats. Suppose the initial population is randomly initialized and their objective (fitness)

values are as follows:

$$\begin{cases} \boldsymbol{x}_1 = \begin{pmatrix} 5.00 & 0.00 & 5.00 \end{pmatrix}, & f_1 = f(\boldsymbol{x}_1) = 33.00, \\ \boldsymbol{x}_2 = \begin{pmatrix} 2.00 & 2.00 & 3.00 \end{pmatrix}, & f_2 = f(\boldsymbol{x}_2) = 6.00, \\ \boldsymbol{x}_3 = \begin{pmatrix} -3.00 & -2.00 & 0.00 \end{pmatrix}, & f_3 = f(\boldsymbol{x}_3) = 26.00, \\ \boldsymbol{x}_4 = \begin{pmatrix} -5.00 & 0.00 & 5.00 \end{pmatrix}, & f_4 = f(\boldsymbol{x}_4) = 53.00, \\ \boldsymbol{x}_5 = \begin{pmatrix} 3.00 & 4.00 & 5.00 \end{pmatrix}, & f_5 = f(\boldsymbol{x}_5) = 29.00, \end{cases} \tag{4.11}$$

where we have only used two decimal places for simplicity. Though the actual random numbers generated for the initial population can be any numbers in the simple bounds, here we have used the numbers that are rounded up as the initialization; this is purely for simplicity and clarity.

Obviously, the best solution with the lowest (or best) value of the objective function is $\boldsymbol{x}_2$ with $f_2 = 6.00$ for this population at $t = 0$. Thus, we have

$$\boldsymbol{x}_{\text{best}} = [2.00, \ \ 2.00, \ \ 3.00]. \tag{4.12}$$

In principle, each bat in the bat population can have its own pulse emission rate r_i and loudness A_i. However, for simplicity of implementation and discussion here, we can use the same r and A for all bats.

At the first iteration $t = 1$, starting with $r^{(0)} = 1$, $A^{(0)} = 1$, $\alpha = 0.97$ and $\gamma = 0.1$, we have

$$r = r^{(0)}[1 - \exp(-\gamma t)] = 1 \times [1 - \exp(-0.1 \times 1)] = 0.0952, \tag{4.13}$$

$$A = \alpha A^{(0)} = 0.97 \times 1 = 0.97. \tag{4.14}$$

When carrying out the loop over all the 5 bats, if we draw a random number $\beta = 0.2$, then the frequency becomes $f_i = 0 + 0.2 \times (2 - 0) = 0.4$, so the velocity $\boldsymbol{v}_1$ can be calculated by

$$\boldsymbol{v}_1^1 = \boldsymbol{v}_1^0 + (\boldsymbol{x}_1 - \boldsymbol{x}_{\text{best}}) \times 0.4 = [4.20, -2.80, 2.80], \tag{4.15}$$

where we have used $f_{\min} = 0$ and $f_{\max} = 2$. This gives the new solution

$$\boldsymbol{x}_1^1(\text{new}) = \boldsymbol{x}_1 + \boldsymbol{v}_1 = [9.20, -2.80, 7.8], \tag{4.16}$$

which in turn gives $f(\boldsymbol{x}_1^1) = 127.92$. This is worse than the previous solutions. However, the next step is to generate a uniformly distributed random number. Suppose we get $\beta = 0.5$, which is greater than $r = 0.0952$; we carry out a local search by

$$\boldsymbol{x}_1^1(\text{new}) = \boldsymbol{x}_{\text{best}} + 0.1\epsilon A. \tag{4.17}$$

If we draw a random number vector, we get

$$\epsilon = [-2, 0.5, -1.5], \tag{4.18}$$

and we have

$$\boldsymbol{x}_1^1(\text{new}) = \boldsymbol{x}_{\text{best}} + 0.1\epsilon \times 0.97 = [1.8060, 2.0485, 2.8545]. \tag{4.19}$$

This gives $f(\boldsymbol{x}_1^1) = 5.1882$, which is better than $\boldsymbol{x}_{\rm best}$. So we update it as the new best solution

$$\boldsymbol{x}_{\rm best} = [1.8060, 2.0485, 2.8545]. \tag{4.20}$$

Similarly, we do this for the rest of the other four solutions in the population. Once the first iteration is done, we vary r and A again, and then carry out another loop over the whole population.

The bat algorithm can be efficient. For the objective function, the optimal solution is $\boldsymbol{x}_* = [1, 1, 1]$. Using the above codes, we can typically get $f_{\rm best} = 4.9 \times 10^{-16}$ after 500 iterations.

4.6 Conclusion

The bat algorithm is a simple, flexible and yet efficient algorithm for solving optimization problems. In addition, this algorithm is relatively simple to implement with a minimum use of memory, and the computation cost is also low. The bat algorithm has been extended to other forms with many variants and applications [2, 10, 11].

References

1. X.S. Yang, "A new metaheuristic bat-inspired algorithm", in: *Nature Inspired Cooperative Strategies for Optimization (NICSO 2010)*, pp. 65-74, Springer, Berlin, 2010.
2. X.S. Yang, X.-S. He, "Bat algorithm: literature review and applications", *Int. J. Bio-Inspired Computation*, 5(3): 141-149, 2013.
3. X.S. Yang, "Bat algorithm for multi-objective optimisation", *Int. J. Bio-Inspired Computation*, 3(5):267-274, 2011.
4. A.H. Gandomi and X.-S. Yang, "Chaotic bat algorithm", *Journal of Computational Science*, 5(2): 224–232, 2014.
5. X.S. Yang and A. H. Gandomi, "Bat algorithm: a novel approach for global engineering optimization", *Engineering Computation*, 29(5): 464-483, 2012.
6. A.H. Gandomi, X.-S. Yang, A.H. Alavi, S. Talatahari, "Bat algorithm for constrained optimization tasks", *Neural Computing and Applications*, 22(6): 1239-1255, 2013.
7. I. Fister Jr., D. Fister, X.-S. Yang, "A hybrid bat algorithm", *Elekrotehnivški Vestn.*, 80(1-2): 1-7, 2013.

8. A. Chakri, R. Khelif, M. Benouaret, X.-S. Yang, "New directional bat algorithm for continuous optimization problems", *Expert Systems with Applications*, 69(1): 159-175, 2017.
9. A. Chakri, X.-S. Yang, R. Khelif, M. Benouaret, "Reliability-based design optimization using the directional bat algorithm", *Neural Computing and Applications*, 30(8): 2381-2402, 2018.
10. E. Osaba, X.-S. Yang, I. Fister Jr., J. Del Ser, P. López-García, A.J. Vazquez-Pardavila, "A discrete and improved bat algorithm for solving a medical good distribution problem with pharmacological waste collection", *Swarm and Evolutionary Computation*, 44(1): 273-286, 2019.
11. S. Chen, G.-H. Peng, X.-S. He, X.-S. Yang, "Global convergence analysis of the bat algorithm using a Markovian framework and dynamical system theory", *Expert Systems with Applications*, 114(1): 173-182, 2018.