ISSN: 2664-9640

Program design for slime mould algorithm based on Python

Junlong Zheng [1,2, a,*], Fengshan Yang [3,b]

[1] Guangxi Electrical Polytechnic Institute, Nanning 530007, China

[2] King Mongkut Institute of Technology, Thailand

[3] Guangxi Electrical Polytechnic Institute, Nanning 530007, Guangxi, China

[a]long8889@126.com, [b]long8889@126.com

*Corresponding Author

Abstract

Because of many papers on Optimization algorithms, most of them only write the principles, mathematical models and pseudo-programs of the algorithm, and few of them provide complete programs for a computer programming language, especially for SMA newly proposed in 2020, it is difficult to find a complete computer program. However, for many scholars, it is very difficult to refer to pseudo programs to write a complete program. This paper provides a complete SMA computer program, and applies SMA to two well-known test functions to provide an important reference for optimizing engineering projects.

Keywords

Slime mould algorithm; Python; Program design.

1. Introduction

Most real-world problems have a high complication, nonlinear limitations, interdependent parameters, and a wide limit of solutions. This ensures technical application with the ability to find a solution for complex optimization problems in realtime[3]. Metaheuristic algorithms are one of these techniques[4]. These algorithms are optimization methods that provide logical and good solutions in a reasonable amount of time. Optimization means that, given the constraints imposed, we achieve a set of valuable variables to achieve the goal of minimizing the subject of the objective function[5]

However, each optimization algorithm has its own advantages and limitations. Here, this paper introduces an optimization algorithm based on bionics - slime mould algorithm [1], abbreviated SMA below .Which provides one more algorithm for optimization calculation. In order to facilitate the application and promotion of the algorithm in engineering projects, this paper uses Python computer language to program the algorithm, and applies the algorithm to several famous test functions to verify the correctness of the program design

The slime mold is a strange organism that is similar to an amoeba but has distinctive features. In the food presentation, each of these organisms behaves like an amoeba. They move in the soil and swallow bacteria. This organism is haploid (n chromosomal). But under conditions of environmental stress (food shortages), they come together to form a full-cell colony, resembling a shell-less snail (animal-like or plasmodium-like). The colony migrates from this environment and stops in the next stage of movement and forms a base and capsule (mushroomlike). Inside the capsule, spores are produced. As spores grow, they form in the environment of amoebic organisms, and this cycle continues. Of course, cellular mucosal molds also reproduce sexually. At this stage, two amoeba-like organisms combine to form an egg cell (chromosomal 2n), which is the only diploid cell in the life cycle of this organism. The egg cell

ISSN: 2664-9640

divides into meiosis, forming four cells that become amoebic-like single-celled organisms in the environment and resume the asexual cycle. However, this creature has no brain, it can do clever things and find its way in the winding paths for searching the food source. It carefully balances its diet and measures food quality and the risk . To have another chance of looking for another location, the time of leaving the area with a low food supply should be decided by the slime mold. The heuristic on the currently available insufficient knowledge is a proper incentive for a slime mold to assess the time of leaving the current spot. The mathematical modeling of this algorithm is described below.

1.1. Approach food[1]

To model the approaching behavior of slime mould as a mathematical equation, the following rule is proposed to imitate the contraction mode:

$$\overrightarrow{X(t+1)} = \begin{cases} \overrightarrow{X_b(t)} + \overrightarrow{vb} \cdot \left(\vec{W} \cdot \overrightarrow{X_A(t)} - \overrightarrow{X_B(t)}\right), r < p \\ \overrightarrow{vc} \cdot \overrightarrow{X(t)}, r \geq p \end{cases} \tag{1-1}$$

where $\overrightarrow{vb}$ is a parameter with a range of $[-a, a]$, $\overrightarrow{vc}$ decreases linearly from one to zero. t represents the current iteration, $\overrightarrow{X_b}$ represents the individual location with the highest odor concentration currently found, $\vec{X}$ represents the location of slime mould, $\overrightarrow{X_A}$ and $\overrightarrow{X_B}$ represent two individuals randomly selected from the swarm, $\vec{W}$ represents the weight of slime mould.

The formula of p is as follows:

$$p = \tanh|S(i) - DF| \tag{1-2}$$

where $i \in 1,2,\ldots,n$, $S(i)$ represents the fitness of $\vec{X}$, DF represents the best fitness obtained in all iterations.

The formula of $\overrightarrow{vb}$ is as follows:

$$\overrightarrow{vb} = [-a, a] \tag{1-3}$$

$$a = \operatorname{arctanh}\left(-\left(\frac{t}{\max\_t}\right) + 1\right) \tag{1-4}$$

The formula of $\vec{W}$ is listed as follows:

$$\overrightarrow{W(SmellIndex(\imath))} = \begin{cases} 1 + r \cdot log\left(\frac{bF - S(i)}{bF - wF} + 1\right), condition \\ 1 - r \cdot log\left(\frac{bF - S(i)}{bF - wF} + 1\right), \quad others \end{cases} \tag{1-5}$$

$$SmellIndex = sort(S) \tag{1-6}$$

where $condition$ indicates that $S(i)$ ranks first half of the population, r denotes the random value in the interval of $[0,1]$, bF denotes the optimal fitness obtained in the current iterative process, wF denotes the worst fitness value obtained in the iterative process currently, $SmellIndex$ denotes the sequence of fitness values sorted(ascends in the minimum value problem).

1.2. Wrap food[1]

The mathematical formula for updating the location of slime mould is as follows:

ISSN: 2664-9640

$$\overrightarrow{X^*} = \begin{cases} rand \cdot (UB - LB) + LB, rand < z \\ \overrightarrow{X_b(t)} + \overrightarrow{vb} \cdot \left(W \cdot \overrightarrow{X_A(t)} - \overrightarrow{X_B(t)}\right), r < p \\ \overrightarrow{vc} \cdot \overrightarrow{X(t)}, r \geq p \end{cases} \quad (1\text{-}7)$$

where LB and UB denote the lower and upper boundaries of the search range, $rand$ and r denote the random value in [0,1].

1.3. Grabble food[1]

The value of $\overrightarrow{vb}$ oscillates randomly between $[-a, a]$ and gradually approaches zero as the iterations increase. The value of $\overrightarrow{vc}$ oscillates between [-1,1] and tends to zero eventually.

2. Algorithm Pseudo-code of SMA

Algorithm Pseudo-code of SMA
Initialize the parameters popsize, $Max\_iteraition$;
Initialize the positions of slime mould $X_i (i = 1,2, \dots, n)$;
While $(t \leq Max\_iteraition)$
Calculate the fitness of all slime mould;
$Update\ bestFitness,\ X_b$
Calculate the W by Eq. (5);
For $each\ search\ portion$
$Update\ p, vb, vc$;
$Update\ positions\ by$ **Eq**. (7);
End **For**
$t = t + 1$;
End While
Return $bestFitness,\ X_b$;

3. Program design for SMA based on Python language

In order to facilitate the application of engineering projects, we use Python computer programming language to program SMA as follows:

```
1  import numpy as np
2  from matplotlib import pyplot as plt
3  import random
4  import math
5  import copy
6
7  #Population initialization function
8  def initial(SMApop, dim, ub, lb):
9      X = np.zeros([SMApop, dim])
```

ISSN: 2664-9640

```
    for i in range(SMApop):
        for j in range(dim):
            X[i, j] = random.random()\
 * (ub[j] - lb[j]) + lb[j]
    return X, lb, ub

#Boundary check function
def BorderCheck(X, ub, lb, SMApop, dim):
    for i in range(SMApop):
        for j in range(dim):
            if X[i, j] > ub[j]:
                X[i, j] = ub[j]
            elif X[i, j] < lb[j]:
                X[i, j] = lb[j]
    return X

'# Calculate fitness function
def CaculateFitness(X, fun,nn,\
Xdraw,Ydraw,Zdraw):
    SMApop = X.shape[0]
    fitness = np.zeros([SMApop, 1])
    for i in range(SMApop):
        fitness[i],nn,Xdraw,Ydraw,Zdraw\
= fun(X[i, :],nn,Xdraw,Ydraw,Zdraw)
    return fitness,nn,Xdraw,Ydraw,Zdraw

#--Function of maximum problem--
def SortFitness(Fit):
    fitness = np.sort(Fit, axis=0)[::-1]
    index = np.argsort(Fit, axis=0)[::-1]
    return fitness, index
'''#--Function of minimum problem--
def SortFitness(Fit):
```

ISSN: 2664-9640

```
    fitness = np.sort(Fit, axis=0)
    index = np.argsort(Fit, axis=0)
    return fitness, index'''

#-- Sort locations according to fitness --
def SortPosition(X, index):
    Xnew = np.zeros(X.shape)
    for i in range(X.shape[0]):
        Xnew[i, :] = X[index[i], :]
    return Xnew

#--SMA Main function--
def SMA(low,up,SMApop,dim,SMALoop,\
nn,Xdraw,Ydraw,Zdraw,fun):
    z = 0.03
    lb = low * np.ones([dim, 1])
    ub = up * np.ones([dim, 1])
    X, lb, ub = initial(SMApop, dim, ub, lb)
    fitness,nn,Xdraw,Ydraw,Zdraw \
= CaculateFitness(X, fun,nn,Xdraw,Ydraw,Zdraw)
    fitness, sortIndex = SortFitness(fitness)
    X = SortPosition(X, sortIndex)
    GbestScore = copy.copy(fitness[0])
    GbestPositon = copy.copy(X[0, :])
    Curve =[]
    iteration=[]
    for f in range(fitness.shape[0]):
        Curve.append(fitness[-(f+1),0])
        iteration.append(f+1)
    W = np.zeros([SMApop, dim])
    for t in range(1,SMALoop):
        worstFitness = fitness[-1]
        bestFitness = fitness[0]
```

ISSN: 2664-9640

```
77          S = bestFitness - worstFitness + 10E-8
78          for i in range(SMApop):
79              if i < SMApop / 2:
80                  W[i, :] = 1 +\
81   np.random.random([1, dim])\
82  * np.log10((bestFitness - fitness[i]) / (S) + 1)
83              else:
84                  W[i, :] = 1 - \
85  np.random.random([1, dim])\
86  * np.log10((bestFitness - fitness[i]) / (S) + 1)
87          tt = -(t / SMALoop) + 1
88          if tt != -1 and tt != 1:
89              a = math.atanh(tt)
90          else:
91              a = 1
92          b = 1 - t / SMALoop
93          for i in range(SMApop):
94              if np.random.random() < z:
95                  X[i, :] = (ub.T - lb.T) * \
96  np.random.random([1, dim]) + lb.T
97              else:
98                  p = np.tanh(abs(fitness[i] - GbestScore))
99                  vb = 2 * a * np.random.random([1, dim]) - a
100                 vc = 2 * b * np.random.random([1, dim]) - b
101                 for j in range(dim):
102                     r = np.random.random()
103                     A = np.random.randint(SMApop)
104                     B = np.random.randint(SMApop)
105                     if r < p:
106                         X[i, j] = GbestPositon[j] \
107 + vb[0, j] * (W[i, j] * X[A, j] - X[B, j])
108                     else:
109                         X[i, j] = vc[0, j] * X[i, j]
```

ISSN: 2664-9640

```
        X = BorderCheck(X, ub, lb, SMApop, dim)
        fitness,nn,Xdraw,Ydraw,Zdraw \
= CaculateFitness(X, fun,nn,Xdraw,Ydraw,Zdraw)
        fitness, sortIndex = SortFitness(fitness)
        X = SortPosition(X, sortIndex)
        if (fitness[0] <= GbestScore):
            GbestScore = copy.copy(fitness[0])
            GbestPositon = copy.copy(X[0, :])
        Curve.append(GbestScore)
        iteration.append(nn)
    return GbestScore, GbestPositon,\
iteration,Curve,nn,Xdraw,Ydraw,Zdraw
```

The following is the SMA global main function

```
#-- SMA global main function--
import numpy as np
from matplotlib import pyplot as plt
from SMA import SMA as SMA
from Ackley import F2 as fun  # Import objective function

# Set general parameters
low=-0.8
up=0.8
dim = 2
nn=0
# Defines the variables used to draw the graph
Xdraw=[]
Ydraw=[]
Zdraw=[]
#--SMA Special parameters--
SMApop=100
SMALoop = 20
# Call SMA ----
GbestScore,GbestPositon,iteration,Curve,nn,Xdraw,Ydraw,Zdraw
```

ISSN: 2664-9640

```
=
SMA(low,up,SMApop,dim,SMALoop,nn,Xdraw,Ydraw,Zdraw,fun)
print('bestScore：',GbestScore)
print('bestPositon:',GbestPositon)
print('Iterations,Zdraw:',nn,Zdraw[-1])

# Draw fitness iteration curve
plt.figure()
plt.semilogx(iteration,Curve,'r-',linewidth=2)
#plt.loglog(iteration,Curve,'r-',linewidth=2)
#plt.semilogy(iteration,Curve,'r-',linewidth=2)
plt.plot(iteration,Curve, color='green', label='training accuracy')
plt.xlabel('Iteration',fontsize='medium')
plt.ylabel("Fitness",fontsize='medium')
plt.title('SMA',fontsize='large')
plt.rcParams['axes.unicode_minus']=False
plt.show()
```

4. SMA is applied to two well-known test functions

4.1. Ackley function

The first function is Ackley. It is characterized by a nearly flat outer region in its two-dimensional shape, as seen in the Fig 4-1, In this almost flat area, many valleys or peaks modulated by cosine wave are superimposed, resulting in uneven surface and a large hole in the middle. The function poses a risk of being stuck in one of its many local minima for optimization algorithms, particularly hill-climbing algorithms. The limitationof the variables is in the range [−5, 5]. The formulation for this function is given below[2]:

$$f_1(x) = -20exp(-0.2\sqrt{\frac{1}{D}\sum_{i=1}^{D}(x_i^2)})$$

$$-exp(\frac{1}{D}\sum_{i=1}^{D}(\cos(2\pi x_i))) + 20 + exp(1) \quad (4\text{-}1)$$

The expression of the two-dimensional variable of the function is written into a python program:

ISSN: 2664-9640

```
1  import numpy as np
2  def F2(X,nn, Xdraw, Ydraw, Zdraw):
3      x,y=X[0],X[1]
4      Fxy=-20 * np.exp(-0.2 * \
5  np.sqrt(0.5 * (x**2 + y**2))) - \
6  (np.exp(0.5 * (np.cos(2 * np.pi * x)\
7   + np.cos(2 * np.pi * y)))) + np.e + 20
8      Xdraw.append(x)
9      Ydraw.append(y)
10     Zdraw.append(Fxy)
11     nn = nn + 1
12     return Fxy, nn, Xdraw, Ydraw, Zdraw
```

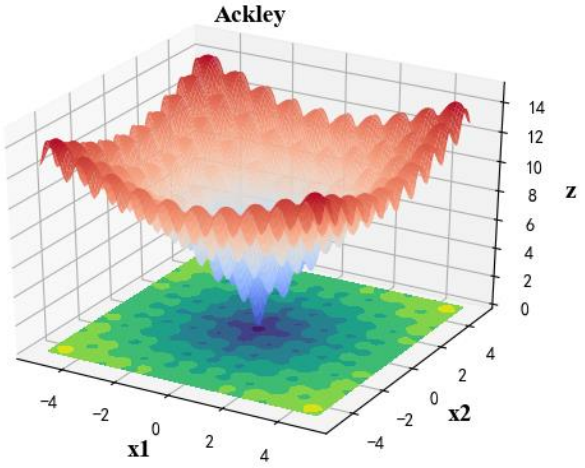

Fig 4-1 Ackley function

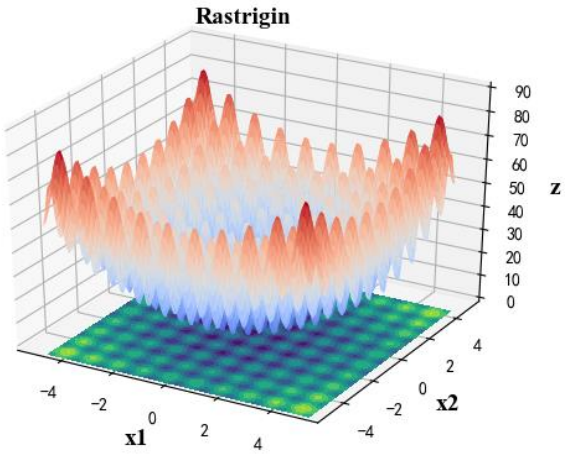

Fig 4-2 Rastrigin function

4.2. Rastrigin function

The second test function is Rastrigin. This function gives numerous local minima. It is highly multimodal, the minimum positions are spread frequently. But the global minimum is 0, in the middle. The two-dimensional variable value range of this function is [- 5, 5],and its figure and formulation is as follows:

$$f_2(x) = 10D + \sum_{i=1}^{D} (x_i^2 - 10\cos(2\pi x_i)) \quad (4\text{-}2)\ [2]$$

4.3. SMA is applied to the two functions

In order to unify opinions, the number of iterations is defined as one operation by substituting independent variables into the objective function each time, which is counted as one iteration.

ISSN: 2664-9640

SMA applies to both of these test functions, with the number of iterations set to 1000 and the operation set to the minimum. In order to clearly understand the operation principle of SMA, we draw the iteration curve, especially the iteration points in the process of program operation.

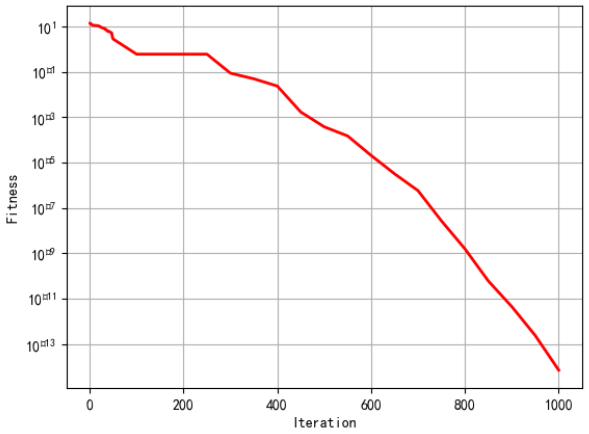

FIG 4.3 the iterative curve for SMA applied to Ackley function

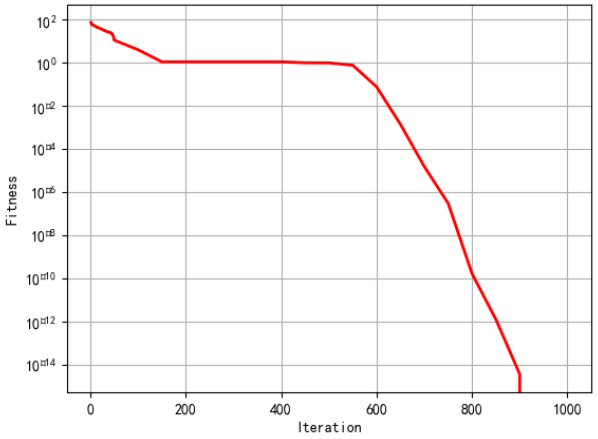

FIG 4.4 the iterative curve for SMA applied to Rastrigin function

From the analysis of the iterative curve of SMA applied to the two test functions, it can be seen that the application of SMA to the optimization operation of the two test functions has fast convergence speed, and only 1000 iterations can achieve more appropriate results. We also apply SMA to several other commonly used optimization algorithm test functions, such as sphere function and schwefe function, which achieve the same effect.

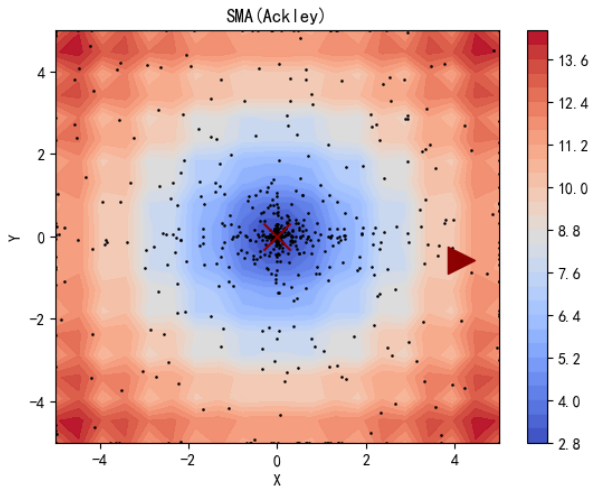

FIG 4.5 the operation position point for SMA applied to Ackley function

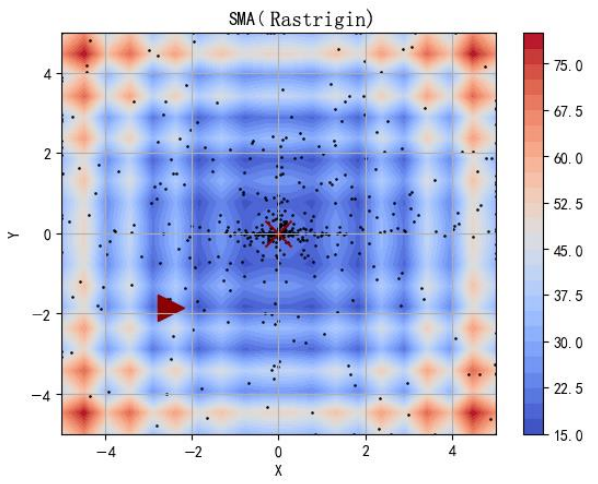

FIG 4.6 the operation position point for SMA applied to Rastrigin function

In the two figures representing the operation position points, "▷ " indicates the starting point of the operation the starting point. "×" Indicates the optimal fitness position .The color difference in the figure represents the height of the position, and the scattered points in the figure represent the " slime mould " randomly distributed by SMA. During the operation, it gradually converges to the optimal position.

ISSN: 2664-9640

5. Conclusion

We have read many papers on Optimization algorithms. Most of them only write the principles, mathematical models and pseudo-programs of the algorithms. Few programs are provided for a computer programming language, especially for SMA, which was just introduced in 2020. It is difficult to find a complete computer program. However, for many scholars, it is very difficult to refer to pseudo programs to write a complete program. This paper provides a complete SMA computer program, and applies SMA to two well-known test functions to provide an important reference for optimizing engineering projects.

Acknowledgements

1.2022 Basic Research Capability Promotion Project for Young and Middle-aged Teachers in Guangxi Higher Education Schools "Research on Automatic Positioning System for Wireless Charging of Automobile Based on Maritime Search and Rescue Algorithm", No. 2022KY1335.

2. Research results of the high-level innovation team of new energy automotive electronics technology in Guangxi Electrical Polytechnic Institute, No. GEPI[2020] 268.

References

[1] Shimin Li, Huiling Chen, Mingjing Wang, Ali Asghar Heidari, Seyedali Mirjalili, Slime mould algorithm: A new method for stochastic optimization, Future Generation Computer Systems, 2020.

[2] Marcin Molga, Czesław Smutnicki. Test functions for optimization needs (2005). Retrieved in Nov.2021.

[3] Khodaei, Hossein, et al., 2018. Fuzzy-based heat and power hub modelsfor cost-emission operation of an industrial consumer using compromiseprogramming. Appl. Therm. Eng. 137, 395–405.

[4] Gollou, A.R., Noradin, Gh., 2017. A new feature selection and hybrid forecastengine for day-ahead price forecasting of electricity markets. J. Intell. FuzzySystems 32 (6), 4031–4045.

[5] Dehghani, Moslem, et al., 2021. Blockchain-based securing of data exchange in apower transmission system considering congestion management and socialwelfare. Sustainability 13 (1), 90