|  |  |  |  |
| --- | --- | --- | --- |
| **题目：** | **萤火虫算法研究和改进** | | |
| **Title:** | **Research and Improvement of Firefly Algorithm** | | |
|  | |  | | |
| **学 科、专 业：** | | **计算机科学与技术** |
| **姓 名：** | | **冯春香** |
| **学 号：** | | **1821011739** |
| **导 师：** | | **接标 教授** |

# 摘 要

仿生群智能优化算法是近些年来国内外学者研究的热点问题，其主要的思想是研究或者模仿自然界群体生活的生物的社会行为而构造的随机搜索方法。目前研究比较多的有两种算法：蚁群算法（ACO）和粒子群算法（PSO）。有研究结果表明，仿生群智能优化算法为许多应用领域提供了新思路和新方法。

2005年，印度学者K.N.Krishnanand和D.Ghose在IEEE群体智能会议上提出了一种新的群智能优化算法，人工萤火虫群优化（Glowworm Swarm Optimization, GSO）算法。2009年，剑桥学者Xin-She Yang根据自然界中萤火虫的发光行为提出萤火虫算法（Firefly Algorithm, FA）。自这两种萤火虫算法提出以来，各国学者对这两种算法进行了研究、改进和应用。经过几年的发展，在连续空间的寻优过程和一些生产调度方面萤火虫算法具有良好的应用前景。

GSO和FA有相似的方面，但在具体实现方面有一定差异。本文具体介绍和分析了FA算法及其改进算法。

**1** **FA算法**

**1.1 算法应用原理**

萤火虫算法(FireflyAlgorithm)源于模拟自然界萤火虫在晚上的群聚活动的自然现象而提出的，在萤火虫的群聚活动中，每只萤火虫通过散发荧光素与同伴进行寻觅食物以及求偶等信息交流。一般来说，荧光素越亮的萤火虫其号召力也就越强，最终会出现很多萤火虫聚集在一些荧光素较亮的萤火虫周围。人工萤火虫算法就是根据这种现象而提出的一种新型的仿生群智能优化算法。在人工萤火虫群优化算法中，每只萤火虫被视为解空间的一个解，萤火虫种群作为初始解随机的分布在搜索空间中，然后根据自然界萤火虫的移动方式进行解空间中每只萤火虫的移动。通过每一代的移动，最终使的萤火虫聚集到较好的萤火虫周围，也即是找到多个极值点，从而达到种群寻优的目的。

把空间各点看成萤火虫，利用发光强的萤火虫会吸引发光弱的萤火虫的特点。在发光弱的萤火虫向发光强的萤火虫移动的过程中，完成位置的迭代，从而找出最优位置，即完成了寻优过程。

萤火虫算法有以下条件：

1. 假设所有萤火虫都是同一性别且相互吸引;

2. 吸引度只与发光强度和距离有关，发光强的萤火虫会吸引周围发光弱 的萤火虫，但是随着距离的增大吸引度逐渐减小，发光强的萤火虫会做随机运动;

3. 发光强弱由目标函数决定，在制定区域内与指定函数成比例关系。

搜索过程和萤火虫的两个重要参数有关:萤火虫的发光亮度和相互吸引度，发光亮的萤火虫会吸引发光弱的萤火虫向它移动，发光越亮代表其位置越好，最亮萤火虫即代表函数的最优解。发光越亮的萤火虫对周围萤火虫的吸引度越高，若发光亮度一样，则萤火虫做随机运动，这两个重要参数都与距离成反比，距离越大吸引度越小。

**1.2 算法的数学描述**

（1）萤火虫的相对荧光亮度：



其中，I0表示最亮萤火虫的亮度，即自身（r=0处）荧光亮度，与目标函数值相关，目标函数组越优，自身亮度越高；γ表示光吸收系数，因为荧光会随着距离的增加和传播媒介的吸收逐渐减弱，所以设置光强吸收系数以体现此特性，可设置为常数；rij表示萤火虫i与j之间的距离。

（2）相互吸引度β



其中，β0表示最大吸引度，即光源处（r=0处）的吸引度。

（3）最优目标迭代



其中，Xi与Xj表示i、j两个萤火虫的空间位置，α为步长因子，rand为[0,1][0,1]上服从均匀分布的随机因子。

**1.3 算法基本流程**

（1）初始化算法基本参数。设置萤火虫数目n，最大吸引度β0，光强吸收系数γ，步长因子α，最大迭代次数MaxGeneration或搜索精度ε；

（2）随机初始化萤火虫的位置，计算萤火虫的目标函数值作为各自最大荧光亮度I0；

（3）计算群体中萤火虫的相对亮度I和吸引度β，根据相对亮度决定萤火虫的移动方向；

（4）更新萤火虫的空间位置，对处在最佳位置的萤火虫进行随机移动；

（5）根据更新后萤火虫的位置，重新计算萤火虫的亮度；

（6）当满足搜索精度或达到最大搜索次数，则转下一步；否则，搜索次数增加1，转第3步，进行下一次搜索。

（7）输出全局极值点和最优个体值。

**2 算法不足**

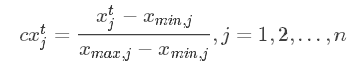
发现率低、求解精读不高、求解速度慢。

**3 萤火虫算法改进实例**

**3.1 混沌萤火虫算法**

**3.11 混沌算法步骤**

（1）令t=0，将萤火虫位置信息Xtj，j=1,2,...,n按下式映射为0到1之间的混沌参量cxtj。



映射公式中：变量xmax，j和变量xmin，j分别为搜索空间中第j维的上界和下界。

（2）根据cxtj，计算得到下一步迭代的混沌参量cxjt+1。

（3）将混沌参量cxjt+1转换为位置信息xjt+1



（4）根据位置信息xjt+1，j=1,2,...,n，对所得新解进行性能评价。

（5）若所得新解优于初始解X(0)=[x0i,...,x0n]或者混沌搜索已到预先设定的精度或迭代次数，则新解作为算法的最终结果，否则令t=t+1并返回步骤2。

**3.12 改进混沌萤火虫算法流程**

（1）系统初始化，生成萤火虫初始种群的规模、位置信息，设置光强吸收系数、最大吸引因子和步长因子等参数；

（2）计算群体中每个萤火虫的荧光亮度；

（3）更新权重公式计算惯性权重；

（4）根据位置更新公式更新萤火虫位置，最亮的萤火虫个体随机移动；

（5）计算位置更新后群体中每个萤火虫的荧光亮度；

（6）计算位置更新后群体中适应度最高的10%的个体，进行混沌局部搜索；

（7）计算位置更新后群体中每个萤火虫的荧光亮度；

（8）判断是否满足终止条件，如果满足终止条件则跳出循环并输出全局最优解，如果不满足条件则继续；

（9）按下列两式收缩搜索区域：





其中，xmaxlight，j表示当前最亮萤火虫个体的第j为变量的值。

在收缩后的新搜索区域内随机产生群体中剩余80%的萤火虫，然后返回步骤3。

**3.2 变步长萤火虫算法**

萤火虫i被亮度更大的萤火虫j吸引向其移动而更新自己的位置，位置更新公式如下：



式中，α为步长因子，一般取[0,1]上的常数；rand为[0,1]上服从均匀分布的随机因子。

萤火虫算法寻优是通过萤火虫之间的相互吸引来实现的，而随着迭代次数的增加，萤火虫群会在最优值附件聚集。此时萤火虫个体与最优值之间的距离已经非常小，在个体向最优值趋近的过程中，很可能会出现萤火虫移动的距离大于个体与最优值间距的情况，而导致个体更新自己位置时跳过了最优值，出现震荡，将会导致最优值发现率降低，影响算法的收敛精度和速度。

为了尽量避免由上述原因造成的收敛较慢情况，在算法开始时，将初始步长设定为相对较大值，而后随着迭代次数以及萤火虫之间距离增加设定一个判定条件：当个体距离小于某一固定步长时，使步长减小。则萤火虫算法将在开始时具有较好的全局寻优能力，迅速定位在接近全局最优解的区域，而后期也具有良好的局部搜索能力，能精确得到全局最优解。

初始步长α设定为0.25，当笛卡尔距离l>0.25时，位置更新公式变为：



式中k为根据实际情况可调整的正相关系数，即用klij代替固定步长α。

**4 FA代码详细解释**

**FA\_SVM\_exmp.m**

tic % 计时器

%% 清空环境变量

close all

clear

clc

format compact

%% 数据提取

% 载入测试数据wine,其中包含的数据为classnumber = 3,wine:178\*13的矩阵,wine\_labes:178\*1的列向量

load wine.mat

% 选定训练集和测试集

% 将第一类的1-30,第二类的60-95,第三类的131-153做为训练集

train\_wine = [wine(1:30,:);wine(60:95,:);wine(131:153,:)];

% 相应的训练集的标签也要分离出来

train\_wine\_labels = [wine\_labels(1:30);wine\_labels(60:95);wine\_labels(131:153)];

% 将第一类的31-59,第二类的96-130,第三类的154-178做为测试集

test\_wine = [wine(31:59,:);wine(96:130,:);wine(154:178,:)];

% 相应的测试集的标签也要分离出来

test\_wine\_labels = [wine\_labels(31:59);wine\_labels(96:130);wine\_labels(154:178)];

%% 数据预处理

% 数据预处理,将训练集和测试集归一化到[0,1]区间

[mtrain,ntrain] = size(train\_wine);

[mtest,ntest] = size(test\_wine);

dataset = [train\_wine;test\_wine];

% mapminmax为MATLAB自带的归一化函数

[dataset\_scale,ps] = mapminmax(dataset',0,1);

dataset\_scale = dataset\_scale';

train\_wine = dataset\_scale(1:mtrain,:);

test\_wine = dataset\_scale( (mtrain+1):(mtrain+mtest),: );

%% FA优化参数

% 参数向量 parameters [n N\_iteration alpha betamin gamma]

% n为种群规模，N\_iteration为迭代次数

para=[10,50,0.5,0.2,1];

% 待优化参数上下界 Simple bounds/limits for d-dimensional problems

d=2; % 待优化参数个数

Lb=[0.01,0.01]; % 下界

Ub=[100,100]; % 上界

% 参数初始化 Initial random guess

u0=Lb+(Ub-Lb).\*rand(1,d);

% 迭代寻优 Display results

[bestsolutio,bestojb]=ffa\_mincon\_svm(@objfun\_svm,u0,Lb,Ub,para,train\_wine\_labels,train\_wine,test\_wine\_labels,test\_wine);

%% 打印参数选择结果

bestc=bestsolutio(1);

bestg=bestsolutio(2);

disp('打印选择结果');

str=sprintf('Best c = %g，Best g = %g',bestc,bestg);

disp(str)

%% 利用最佳的参数进行SVM网络训练

cmd\_gwosvm = ['-c ',num2str(bestc),' -g ',num2str(bestg)];

model\_gwosvm = svmtrain(train\_wine\_labels,train\_wine,cmd\_gwosvm);

% model\_gwosvm = fitcsvm(train\_wine\_labels,train\_wine,cmd\_gwosvm);

%% SVM网络预测

[predict\_label,accuracy] = svmpredict(test\_wine\_labels,test\_wine,model\_gwosvm);

% 打印测试集分类准确率

total = length(test\_wine\_labels);

right = sum(predict\_label == test\_wine\_labels);

disp('打印测试集分类准确率');

str = sprintf( 'Accuracy = %g%% (%d/%d)',accuracy(1),right,total);

disp(str);

%% 结果分析

% 测试集的实际分类和预测分类图

figure;

hold on;

plot(test\_wine\_labels,'o');

plot(predict\_label,'r\*');

xlabel('测试集样本','FontSize',12);

ylabel('类别标签','FontSize',12);

legend('实际测试集分类','预测测试集分类');

title('测试集的实际分类和预测分类图','FontSize',12);

grid on

snapnow

%% 显示程序运行时间

toc

**ffa\_mincon\_svm.m**

% 算法主程序开始 Start FA

function [nbest,fbest]=ffa\_mincon\_svm(costfhandle,u0, Lb, Ub, para,train\_wine\_labels,train\_wine,test\_wine\_labels,test\_wine)

% 检查输入参数 Check input parameters (otherwise set as default values)

if nargin<5

para=[20 100 0.25 0.20 1];

end

if nargin<4

Ub=[];

end

if nargin<3

Lb=[];

end

if nargin<2

disp('Usuage: FA\_mincon(@cost,u0,Lb,Ub,para)');

end

n=para(1);

MaxGeneration=para(2);

alpha=para(3);

betamin=para(4);

gamma=para(5);

% 检查上界向量与下界向量长度是否相同 Check if the upper bound & lower bound are the same size

if length(Lb) ~=length(Ub)

disp('Simple bounds/limits are improper!')

return

end

% 计算待优化参数维度 Calcualte dimension

d=length(u0);

% 初始化目标函数值 Initial values of an array

zn=ones(n,1)\*10^100;

% 初始化萤火虫位置 generating the initial locations of n fireflies

[ns,Lightn]=init\_ffa(n,d,Lb,Ub,u0);

for k=1:MaxGeneration % 迭代开始

% 更新alpha（可选）This line of reducing alpha is optional

alpha=alpha\_new(alpha,MaxGeneration);

% 对每个萤火虫计算目标函数值 Evaluate new solutions (for all n fireflies)

for i=1:n

zn(i)=costfhandle(ns(i,:),train\_wine\_labels,train\_wine,test\_wine\_labels,test\_wine);

Lightn(i)=zn(i);

end

% 根据亮度排序 Ranking fireflies by their light intensity/objectives

[Lightn,Index]=sort(zn);

ns\_tmp=ns;

for i=1:n

ns(i,:)=ns\_tmp(Index(i),:);

end

%% 找出当前最优 Find the current best

nso=ns;

Lighto=Lightn;

nbest=ns(1,:);

Lightbest=Lightn(1);

% 另存最优值 For output only

fbest=Lightbest;

% 向较优方向移动 Move all fireflies to the better locations

[ns]=ffa\_move(n,d,ns,Lightn,nso,Lighto,alpha,betamin,gamma,Lb,Ub);

end

% ----- All the subfunctions are listed here ------------

% 初始化萤火虫位置 The initial locations of n fireflies

function [ns,Lightn]=init\_ffa(n,d,Lb,Ub,u0)

ns=zeros(n,d);

if ~isempty(Lb) % 如果参数界限不为空 if there are bounds/limits

for i=1:n

ns(i,:)=Lb+(Ub-Lb).\*rand(1,d); % 则在取值范围内随机取值

end

else % 如果没有设置参数界限

for i=1:n

ns(i,:)=u0+randn(1,d); % 在原有参数上加白噪声

end

end

% 初始化目标函数 initial value before function evaluations

Lightn=ones(n,1)\*10^100;

% Move all fireflies toward brighter ones

function [ns]=ffa\_move(n,d,ns,Lightn,nso,Lighto,alpha,betamin,gamma,Lb,Ub)

% 参数取值范围绝对值 Scaling of the system

scale=abs(Ub-Lb);

% 更新萤火虫 Updating fireflies

for i=1:n

% The attractiveness parameter beta=exp(-gamma\*r)

for j=1:n

r=sqrt(sum((ns(i,:)-ns(j,:)).^2));

% Update moves

if Lightn(i)>Lighto(j) % 如果i比j亮度更强 Brighter and more attractive

beta0=1;

beta=(beta0-betamin)\*exp(-gamma\*r.^2)+betamin;

tmpf=alpha.\*(rand(1,d)-0.5).\*scale;

ns(i,:)=ns(i,:).\*(1-beta)+nso(j,:).\*beta+tmpf;

end

end % end for j

end % end for i

% 防止越界 Check if the updated solutions/locations are within limits

[ns]=findlimits(n,ns,Lb,Ub);

% alpha参数更新函数

function alpha=alpha\_new(alpha,NGen)

% alpha\_n=alpha\_0(1-delta)^NGen=10^(-4);

% alpha\_0=0.9

delta=1-(10^(-4)/0.9)^(1/NGen);

alpha=(1-delta)\*alpha;

% 防止越界 Make sure the fireflies are within the bounds/limits

function [ns]=findlimits(n,ns,Lb,Ub)

for i=1:n

% Apply the lower bound

ns\_tmp=ns(i,:);

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

**objfun\_svm.m**

%% Objective Function

function f=objfun\_svm(cv,train\_wine\_labels,train\_wine,test\_wine\_labels,test\_wine)

% cv为长度为2的横向量，即SVM中参数c和v的值

cmd = [' -c ',num2str(cv(1)),' -g ',num2str(cv(2))];

model=svmtrain(train\_wine\_labels,train\_wine,cmd); % SVM模型训练

[~,fitness]=svmpredict(test\_wine\_labels,test\_wine,model); % SVM模型预测及其精度

f=1-fitness(1)/100; % 以分类预测错误率作为优化的目标函数值

**5 总结**

萤火虫算法(Firely Algorithm,FA)是一种启发式算法，这种算法启发于晚上萤火虫发光的行为。萤火虫的闪光，其主要目的是作为一个信号系统，以吸引其他的萤火虫。剑桥大学的Xin-She Yang教授在2009年提出了萤火虫算法，其假设为：

1、萤火虫不分性别，它将会被吸引到所有其他比它更亮的萤火虫那去；

2、萤火虫的吸引力和亮度成正比，对于任何两只萤火虫，其中一只会向着比它更亮的另一只移动，然而，亮度是随着距离的增加而减少的；

3、如果没有找到一个比给定的萤火虫更亮，它会随机移动。