查重（仅模型部分）

程序引用的是《30 个智能算法案例》，本文目前只对程序做了批注和小部分修改， 该算法实现智能选取打包点，然后根据设置的范围参数，对周围一定范围内的点进行打包。

免疫算法智能打包

免疫算法是一种模仿生物的免疫系统，该算法的自适应性较强，记忆、识别和学习能力较强，全局搜索能力强。

本文在进行任务的聚类打包时，制定以下原则：

①多个任务进行打包后，视为一个任务集，一任务点最多只能存在于一个任务集。

②一个任务集包含的任务点不能过多。

③一个任务集以某个任务点作为聚类中心，周围任务点与中心点的平均距离最小。

构建函数及约束条件如下：

$$\min A=\frac{\sum a_{ij}}{j} \tag{6}$$

约束条件：

$$a_{ij}=|h_i-z_{ij}|$$

$$D_{ij}=\begin{cases}0, & a_{ij}\geq s\\ 1, & a_{ij}<s\end{cases}$$

对于某一区域的点的任务点，要对该区域中的任务进行打包，首先算出中心点 h_i 到其余点 z_j 的距离 a_{ij}； a_{ij} 中 i 为该区域有 i 个打包中心点数，每个打包内除中心点外，还有 j 个其余点； s 为其余点到打包中心点的距离上限； D 为其余点与打包点的关系，如果在给定范围以内的取值为 1，表示可以打包为一类，为 0 则不打包； A 为一个打包内的所有点距中心的平均距离。

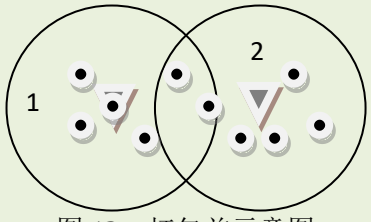

图 12　打包前示意图

免疫算法打包程序实现的步骤如下：

*Step*1:初始化参数并产生初始打包中心点，即初始种群；

*Step*2:对每个打包内的点进行距离分析；

*Step*3：分析每个点对所有打包中心点的距离，每个点都可能处于多个打包范围内；

*Step*4：通过比对步骤 2 与 3 中的距离，对于每个任务点而言，优先选择距离最近的一个中心点，并记录；

*Step*5：判断是否每个点都给予了分类，并检验是否满足条件，是则结束继续步骤 6；

*Step*6：产生新群体，基于步骤 4 的计算结果进行遗传方式产生新的打包群体；

*Step*7：比对前一个打包群体，对比各个打包内的 *A* 值，保留最小的群体，并循环步骤 4-7；

*Step*8：输出结果。

附录：免疫算法打包主程序	运行环境：Matlab2011a

main.m

```
%% step1 参数设置
sizepop=10;                % 种群规模
overbest=10;               % 记忆库容量大小
MAXGEN=100;                % 迭代次数
pcross=0.5;                % 免疫细胞遗传过程中的交叉概率
pmutation=0.4;             % 免疫细胞遗传过程中的变异概率
ps=0.95;                   % 多样性评价参数（一般设置为 0.95）
length=10;                 % 最大打包数
M=sizepop+overbest;
%% step2 将种群信息定义为一个结构体
individuals = struct('fitness',zeros(1,M), 'concentration',zeros(1,M),'excellence',zeros(1,M),'chrom',[]);
%struct 函数功能是生成一个具有指定字段名和相应数据的结构数组
%% step3 产生初始抗体种群
individuals.chrom = popinit(M,length);%“.chrom”是调用 individuals 数组
trace=[]; %初始化矩阵，用于记录每代最个体优适应度和平均适应度
%% 迭代寻优
for iii=1:MAXGEN
    %% step4 抗体群多样性评价
    for i=1:M
        individuals.fitness(i) = fitness(individuals.chrom(i,:));          % 每个个体适应度计算
        individuals.concentration(i) = concentration(i,M,individuals); % 计算第 i 个个体的打包中心点与种群中每个个体的打包中心点间的相似度
    end
    % 综合亲和度和浓度评价抗体优秀程度，得出繁殖概率
    individuals.excellence = excellence(individuals,M,ps);
    % 记录当代最佳个体和种群平均适应度
    [best,index] = min(individuals.fitness);     % 找出当前种群中适应值最小的中心点类，及较优的打包中心点
    bestchrom = individuals.chrom(index,:);      % 找出最优个体，即 min(individuals.fitness)对应的个体
    average = mean(individuals.fitness);         % 计算平均适应度
    trace = [trace;best,average];                % 记录最有个体
    %% step5 根据 excellence，形成父代群，更新记忆库（加入精英保留策略，可由 s 控制）
```

```
        bestindividuals = bestselect(individuals,M,overbest);     % 更新记忆库
        individuals = bestselect(individuals,M,sizepop);            % 更新父代群
        %% step6 选择，交叉，变异操作，再加入记忆库中抗体，产生新种群
        individuals = Select(individuals,sizepop);            % 选择
        individuals.chrom = Cross(pcross,individuals.chrom,sizepop,length);            % 交叉
        individuals.chrom = Mutation(pmutation,individuals.chrom,sizepop,length);     % 变异
        individuals = incorporate(individuals,sizepop,bestindividuals,overbest);      % 加入记忆库
中抗体
end
%% 画出免疫算法收敛曲线
figure(1)
plot(trace(:,1));
hold on
plot(trace(:,2),'--');
legend('最优适应度值','平均适应度值')
title('免疫算法收敛曲线','fontsize',12)
xlabel('迭代次数','fontsize',12)
ylabel('适应度值','fontsize',12)
%% 画出打包中心选址图
%城市坐标
city_coordinate=[1304,2312;3639,1315;4177,2244;3712,1399;3488,1535;3326,1556;3238,1229;4
196,1044;4312,790;4386,570;

3007,1970;2562,1756;2788,1491;2381,1676;1332,695;3715,1678;3918,2179;4061,2370;3780,221
2;3676,2578;

4029,2838;4263,2931;3429,1908;3507,2376;3394,2643;3439,3201;2935,3240;3140,3550;2545,23
57;2778,2826;2370,2975];
carge=[20,90,90,60,70,70,40,90,90,70,60,40,40,40,20,80,90,70,100,50,50,50,80,70,80,40,40,60,70
,50,30];
%找出最近打包点
for i=1:31
    distance(i,:)=dist(city_coordinate(i,:),city_coordinate(bestchrom,:)');
end
[a,b]=min(distance');
index=cell(1,length);
for i=1:length
%记录各个打包中心点的位置
index{i}=find(b==i);
end
figure(2)
title('最优打包图')
cargox=city_coordinate(bestchrom,1);
cargoy=city_coordinate(bestchrom,2);
plot(cargox,cargoy,'rs','LineWidth',2,'MarkerEdgeColor','r','MarkerFaceColor','b','MarkerSize',20)
hold on
plot(city_coordinate(:,1),city_coordinate(:,2),'o','LineWidth',2,'MarkerEdgeColor','k','MarkerFace
Color','g','MarkerSize',10)
for i=1:31
    x=[city_coordinate(i,1),city_coordinate(bestchrom(b(i)),1)];
    y=[city_coordinate(i,2),city_coordinate(bestchrom(b(i)),2)];
    plot(x,y,'c');hold on
end
```

popinit.m

```
function pop = popinit(n,length)
```

```
%种群初始化函数(记忆库库为空，全部随机产生)
% n            input       种群数量
% length     input       抗体长度
% pop          output      初始种群
for i=1:n
    flag=0;
    while flag==0
        aaa=rand(1,31); %一共有 31 个点
        [a,b]=sort(aaa);
        %b 矩阵记录的是 aaa 矩阵中元素数值大小的名次
        pop(i,:)=b(1:length);%pop 矩阵提取的是 b 矩阵中的前 length 个元素
        flag=test(pop(i,:));%test 为条件函数文件，如果满足条件，那么 flag 就=0，此时终止这次循环
    end
end
```

test.m

```
function flag=test(code)
%  检查个体是否满足距离约束
% code       input        个体
% flag       output       是否满足要求标志
city_coordinate=[1304,2312;3639,1315;4177,2244;3712,1399;3488,1535;3326,1556;3238,1229;4196,1044;4312,790;4386,570;

3007,1970;2562,1756;2788,1491;2381,1676;1332,695;3715,1678;3918,2179;4061,2370;3780,2212;3676,2578;

4029,2838;4263,2931;3429,1908;3507,2376;3394,2643;3439,3201;2935,3240;3140,3550;2545,2357;2778,2826;2370,2975];
flag=1;
if max( max(dist( city_coordinate(code,:)') ) )>3000%所有打包点之间的最大距离要大于 3000
    %dist 是一个欧式距离加权函数，给一个输入使用权值，去获得加权的输入
    %dist(pos)输入一个参数，pos——S×S 的神经元位置矩阵，返回 S×S 的距离矩阵
    flag=0;
end
end
```

fitness.m

```
function fit=fitness(individual)
%计算个体适应度值
%individual      input          个体
%fit                output         适应度值
%城市坐标
city_coordinate=[1304,2312;3639,1315;4177,2244;3712,1399;3488,1535;3326,1556;3238,1229;4196,1044;4312,790;4386,570;

3007,1970;2562,1756;2788,1491;2381,1676;1332,695;3715,1678;3918,2179;4061,2370;3780,2212;3676,2578;

4029,2838;4263,2931;3429,1908;3507,2376;3394,2643;3439,3201;2935,3240;3140,3550;2545,2357;2778,2826;2370,2975];
%计算出所有打包点到中心点的距离
for i=1:31
    distance(i,:)=dist(city_coordinate(i,:),city_coordinate(individual,:)');
end
%找出离中心点最近点，a 记录最近距离，b 记录最近的点
```

```
[a,b]=min(distance');
%如果距离大于 3000，那么就要加上一个惩罚值：4.0e+4*length，这里自己定，惩罚值尽可能大些，但也别太大
fit=sum(a) + 4.0e+4*length(find(a>3000));
end
```

concentration.m

```
function concentration = concentration(i,M,individuals)
% 计算个体浓度值
% i                 input       第 i 个抗体
% M                 input       种群规模
% individuals       input       个体
% concentration     output      浓度值
concentration=0;
for j=1:M
    xsd=similar(individuals.chrom(i,:),individuals.chrom(j,:));  % 第 i 个体与种群所有个体间的相似度
    % 相似度大于阀值就说明相似
    if xsd>0.7
        concentration=concentration+1;
    end
end
concentration=concentration/M;%计算相似度
end
```

similar.m

```
function resemble = similar(individual1,individual2)
% 计算个体 individual1 和 individual2 的相似度
% individual1,individual2     input       两个个体
% resemble                    output      相似度
k=zeros(1,length(individual1));
for i=1:length(individual1)
    if find(individual1(i)==individual2)
        k(i)=1;
    end
end
resemble=sum(k)/length(individual1);
end
```

excellence.m

```
function exc=excellence(individuals,M,ps)
% 计算个体繁殖概率
% individuals       input       种群
% M                 input       种群规模
% ps                input       多样性评价参数
% exc               output      繁殖概率
fit = 1./individuals.fitness;
sumfit = sum(fit);
con = individuals.concentration;
sumcon = sum(con);
for i=1:M
    exc(i) = fit(i)/sumfit*ps +con(i)/sumcon*(1-ps); %exc 为当前每个个体的一个评价值
end
end
```

bestselect.m

```
function rets=bestselect(individuals,m,n)
```

```
% 初始化记忆库,依据 excellence，将群体中高适应度低相似度的 overbest 个个体存入记忆库
% m                    input          抗体数
% n                    input          记忆库个体数\父代群规模
% individuals          input          抗体群
% bestindividuals      output         记忆库\父代群
% 精英保留策略，将 fitness 最好的 s 个个体先存起来，避免因其浓度高而被淘汰
s=3;
rets=struct('fitness',zeros(1,n), 'concentration',zeros(1,n),'excellence',zeros(1,n),'chrom',[]);
[fitness,index] = sort(individuals.fitness);
for i=1:s%s 为记忆库的最大含量
    rets.fitness(i) = individuals.fitness(index(i));
    rets.concentration(i) = individuals.concentration(index(i));
    rets.excellence(i) = individuals.excellence(index(i));
    rets.chrom(i,:) = individuals.chrom(index(i),:);
end
% 剩余 m-s 个个体
leftindividuals=struct('fitness',zeros(1,m-s),
'concentration',zeros(1,m-s),'excellence',zeros(1,m-s),'chrom',[]);
for k=1:m-s
    leftindividuals.fitness(k) = individuals.fitness(index(k+s));
    leftindividuals.concentration(k) = individuals.concentration(index(k+s));
    leftindividuals.excellence(k) = individuals.excellence(index(k+s));
    leftindividuals.chrom(k,:) = individuals.chrom(index(k+s),:);
end
% 将剩余抗体按 excellence 值排序
[excellence,index]=sort(1./leftindividuals.excellence);
% 在剩余群体中再选 n-s 个最好的个体，也就是总共筛选出记忆库的最大含量个个体，然后按优劣从上往下排序
for i=s+1:n
    rets.fitness(i) = leftindividuals.fitness(index(i-s));
    rets.concentration(i) = leftindividuals.concentration(index(i-s));
    rets.excellence(i) = leftindividuals.excellence(index(i-s));
    rets.chrom(i,:) = leftindividuals.chrom(index(i-s),:);
end
end
```

Select.m

```
function ret=Select(individuals,sizepop)
% 轮盘赌选择
% individuals input  ： 种群信息
% sizepop      input  ： 种群规模
% ret          output ： 选择后得到的种群
excellence=individuals.excellence;
pselect=excellence./sum(excellence);
% 事实上 pselect = excellence
index=[];
for i=1:sizepop     % 转 sizepop 次轮盘，sizepop 为种群数
    pick=rand;
    for j=1:sizepop
        pick=pick-pselect(j);
        if pick<0
            index=[index j];
            break;   % 寻找落入的区间，此次转轮盘选中了染色体 j
        end
    end
end
```

```
% 注意：在转 sizepop 次轮盘的过程中，有可能会重复选择某些染色体
individuals.chrom=individuals.chrom(index,:);
individuals.fitness=individuals.fitness(index);
individuals.concentration=individuals.concentration(index);
individuals.excellence=individuals.excellence(index);
ret=individuals;
end
```

Cross.m

```
function ret=Cross(pcross,chrom,sizepop,length)
% 交叉操作
% pcorss                    input  ：交叉概率
% chrom                     input  ：抗体群
% sizepop                   input  ：种群规模
% length                    input  ：抗体长度
% ret                       output：交叉得到的抗体群
% 每一轮 for 循环中，可能会进行一次交叉操作，随机选择染色体是和交叉位置，是否进行交叉操作则由交叉概率（continue）控制
for i=1:sizepop
    % 随机选择两个染色体进行交叉
    pick=rand;
    while prod(pick)==0
        pick=rand(1);
    end
    if pick>pcross
        continue;
    end
    % 找出交叉个体
    index(1)=unidrnd(sizepop);
    index(2)=unidrnd(sizepop);
    while index(2)==index(1)
        index(2)=unidrnd(sizepop);
    end
    % 选择交叉位置
    pos=ceil(length*rand);
    while pos==1
        pos=ceil(length*rand);
    end
    % 个体交叉
    chrom1=chrom(index(1),:);
    chrom2=chrom(index(2),:);
    k=chrom1(pos:length);
    chrom1(pos:length)=chrom2(pos:length);
    chrom2(pos:length)=k;
    % 满足约束条件赋予新种群
    flag1=test(chrom(index(1),:));
    flag2=test(chrom(index(2),:));
    if flag1*flag2==1
        chrom(index(1),:)=chrom1;
        chrom(index(2),:)=chrom2;
    end
end
ret=chrom;
end
```

Mutation.m

```
function ret=Mutation(pmutation,chrom,sizepop,length1)
```

```
% 变异操作
% pmutation          input  ：变异概率
% chrom              input  ：抗体群
% sizepop            input  ：种群规模
% iii                input  ：进化代数
% MAXGEN             input  ：最大进化代数
% length1           input  ：抗体长度
% ret                output：变异得到的抗体群
% 每一轮 for 循环中，可能会进行一次变异操作，染色体是随机选择的，变异位置也是随机选择的
for i=1:sizepop
    % 变异概率
    pick=rand;
    while pick==0
        pick=rand;
    end
    index=unidrnd(sizepop);
  % 判断是否变异
    if pick>pmutation
        continue;
    end
    pos=unidrnd(length1);
    while pos==1
        pos=unidrnd(length1);
    end
    nchrom=chrom(index,:);
    nchrom(pos)=unidrnd(31);
    while length(unique(nchrom))==(length1-1)
        nchrom(pos)=unidrnd(31);
    end
    flag=test(nchrom);
    if flag==1
        chrom(index,:)=nchrom;
    end
end
ret=chrom;
end
```

incorporate.m

```
function newindividuals = incorporate(individuals,sizepop,bestindividuals,overbest)
% 将记忆库中抗体加入，形成新种群
% individuals        input        抗体群
% sizepop            input        抗体数
% bestindividuals    input        记忆库
% overbest           input        记忆库容量
m = sizepop+overbest;
newindividuals = struct('fitness',zeros(1,m),'concentration',zeros(1,m),'excellence',zeros(1,m),'chrom',[]);
% 遗传操作得到的抗体
for i=1:sizepop
    newindividuals.fitness(i) = individuals.fitness(i);
    newindividuals.concentration(i) = individuals.concentration(i);
    newindividuals.excellence(i) = individuals.excellence(i);
    newindividuals.chrom(i,:) = individuals.chrom(i,:);
end
```

```
% 记忆库中抗体
for i=sizepop+1:m
    newindividuals.fitness(i) = bestindividuals.fitness(i-sizepop);
    newindividuals.concentration(i) = bestindividuals.concentration(i-sizepop);
    newindividuals.excellence(i) = bestindividuals.excellence(i-sizepop);
    newindividuals.chrom(i,:) = bestindividuals.chrom(i-sizepop,:);
end
end
```