【Experiment name】 Experiment 2 Solving TSP Problems with Genetic Algorithms

【Purpose】

Master the TSP problem solving method based on genetic algorithm.

【Experimental principle】

The TSP search space increases with the increase of the number of cities n, and the number of combinations of all journey routes is (n-1)!/2. There are many computational difficulties in seeking the optimal solution in such a huge search space for conventional methods and existing computing tools. It is a natural idea to solve the TSP problem with the help of the search ability of the genetic algorithm.

The basic genetic algorithm can be defined as an 8-tuple:

(SGA) = (C, E, P0, M, Φ, Γ, Ψ, T)

C - individual encoding method, SGA uses a fixed-length binary symbol string encoding method;

E ——Individual fitness evaluation function;

P0 - initial population;

M - the size of the group, generally 20-100;

Φ——selection operator, SGA uses proportional operator;

Γ——crossover operator, SGA uses a single-point crossover operator;

Ψ——mutation operator, SGA uses the basic bit mutation operator;

T—algorithm termination condition, the general termination evolution algebra is 100-500;

【Experimental content】

I initially tried to write this experiment in python , but due to some data format problems, I couldn't output the ideal results, so I switched to matlab .

```
        # # 变异函数
        # def mutate(path, prob):
        #     random=np.random.rand()
        #     if random <=prob:
        #         length=np.size(path[0])
        #         l=np.size(path)/length
        #         index1=np.random.randint(length)
        #         index2=np.random.randint(length)
        #         # 交换
        #         temp = path[1][index1]
        #         path[1][index1]=path[1][index2]
        #         path[1][index2]=temp
        #     mutatedPath=path
        #     return mutatedPath
```

Code interpretation:

Part 1: File reading and initialization

```
%% 文件读取与初始化
[cityNum,cities] = Readfile('map-10.tsp');
cities = cities';
gbest = Inf;
% 计算各城市距离
distances = getDistance(cities);

% 参数设置
maxIteration = 100;
popSize = 10;
crossoverRate = 0.8; %交叉概率
mutationRate = 0.1; %变异概率

% 生成种群，每个个体代表一个路径
pop = zeros(popSize, cityNum);     % 10条路径，每条路径上10座城市
for i=1:popSize
    pop(i,:) = randperm(cityNum);
end
offspring = zeros(popSize,cityNum); %
%保存每代的最小路径便于画图
minPathes = zeros(maxIteration,1);     % 只存一条路径
```

map-10.tsp - 记事本

文件(F) 编辑(E) 格式(O) 查看(V) 帮助(H)

```
1 116 40
2 121 31
3 109 34
4 114 22
5 118 25
6 103 36
7 102 25
8 114 31
9 127 46
10 88 44
```

① To obtain the data file, I selected 10 major cities across the country , and used their latitude and longitude respectively as shown in the figure above.

② Set parameters, where algebra T =100 , crossover probability 0.8 , mutation probability 0.2

③ Generating populations based on urban data for genetic processing

④ Save the minimum path of each generation

the second part:

```
%% 遗传算法
for  posterity=1:maxIteration
    % 计算适应度fitness
    [fval, sumDistance, minPath, maxPath] = fitness(distances, pop);
    % 每轮操作
    testSize=5; % 设置操作数大小
    for k=1:popSize    % 对父代进行交叉
        % 先选parent1
        PopDistances=zeros( testSize,1);
        for i=1:testSize
            randomRow = randi(popSize); % 整数随机均匀分布
            PopDistances(i,1) = sumDistance(randomRow,1);
        end
        parent1  = min(PopDistances);        % 获取距离最小的
        [parent1X,parent1Y] = find(sumDistance==parent1, 1, 'first');
        parent1Path = pop(parent1X(1,1),:);
        % 再选parent2
        for i=1:testSize
            randomRow = randi(popSize);
            PopDistances(i,1) = sumDistance(randomRow,1);
        end
        parent2  = min(PopDistances);
        [parent2X,parent2Y] = find(sumDistance==parent2, 1, 'first');
        parent2Path = pop(parent2X(1,1),:);

        % 先交叉再变异
        subPath = crossover(parent1Path, parent2Path, crossoverRate);%交叉
        subPath = mutation(subPath, mutationRate);%变异

        offspring(k,:) = subPath(1,:);
        minPathes(posterity,1) = minPath;
    end
    fprintf('代数T:%d\t最短路径:%.2fKM \n', posterity,minPath);
```

```
    % 显示当前最短路径地图
    if minPath < gbest
        gbest = minPath;
        show(cities, pop, gbest, sumDistance);
    end
end
```

①Calculation of fitness

② Operate each round to obtain the size of the operand

③ In the single-round operation, select the parent generation for crossover, and perform the same operation on parent1 and parent2 successively

④ Based on the parent processing results, do crossover-mutation operations, and perform multiple operations

⑤Display the current shortest path map, and perform genetic iteration according to algebra T

the third part:

```
%% 输出最终耗时
tEnd = toc(tStart);
fprintf('时间:%d 分  %f 秒.\n', floor(tEnd/60), rem(tEnd,60));
```

The final output is time-consuming, and the shortest path is based on the result of the last generation

Concrete function:

Read data file function:

```
%% 读数据文件
function [n_citys,city_position] = Readfile(filename)
fid = fopen(filename,'rt');
location=[];
A = [1 2];
tline = fgetl(fid);
while ischar(tline)
    if(strcmp(tline,'NODE_COORD_SECTION'))
        while ~isempty(A)
            A=fscanf(fid,'%f',[3,1]);
            if isempty(A)
                break;
            end
            location=[location;A(2:3)'];
        end
    end
    tline = fgetl(fid);
    if strcmp(tline,'EOF')
        break;
    end
end
[m,n]=size(location);
n_citys = m;
city_position=location;
fclose(fid);
end
```

Cross function:

```matlab
%% 交叉函数
function [childPath] = crossover(parent1Path, parent2Path, prob)
    random = rand();
    if prob >= random
        [l, length] = size(parent1Path);
        childPath = zeros(1,length);
        setSize = floor(length/2) -1;
        offset = randi(setSize);
        for i=offset:setSize+offset-1
            childPath(1,i) = parent1Path(1,i);
        end
        iterator = i+1;
        j = iterator;
        while any(childPath == 0)
            if j > length
                j = 1;
            end

            if iterator > length
                iterator = 1;
            end
            if ~any(childPath == parent2Path(1,j))
                childPath(1,iterator) = parent2Path(1,j);
               iterator = iterator + 1;
            end
            j = j + 1;
        end
    else
        childPath = parent1Path;
    end
end
```

Cross thinking:

The encoding method based on the path representation requires that no repeated gene codes are allowed in the chromosome encoding of an individual, that is to say, it must satisfy the constraint that any city must be visited only once. The individuals generated by the crossover operation of the basic genetic algorithm generally cannot satisfy this constraint. The partial matching crossover operation requires randomly selecting two intersection points in order to determine a matching segment, and generates two child individuals according to the mapping relationship given by the intermediate segment between the two intersection points in the two parent individuals.

fitness output

```
%% 整体种群适应度
function [ fitnessvar, sumDistances,minPath, maxPath ] = fitness( distances, pop )
    [popSize, col] = size(pop);
    sumDistances = zeros(popSize,1);
    fitnessvar = zeros(popSize,1);
    for i=1:popSize
       for j=1:col-1
          sumDistances(i) = sumDistances(i) + distances(pop(i,j),pop(i,j+1));
       end
    end
    minPath = min(sumDistances);
    maxPath = max(sumDistances);
    for i=1:length(sumDistances)
        fitnessvar(i,1)=(maxPath - sumDistances(i,1)+0.000001) / (maxPath-minPath+0.00000001);
    end
end
```

Adaptability idea:

Genetic algorithm basically does not use external information in the evolutionary search, only based on the fitness function, and uses the fitness value of each individual in the population to search. The goal of TSP is the shortest total path length, and the reciprocal of the total path length can be the fitness function of TSP

Path update function:

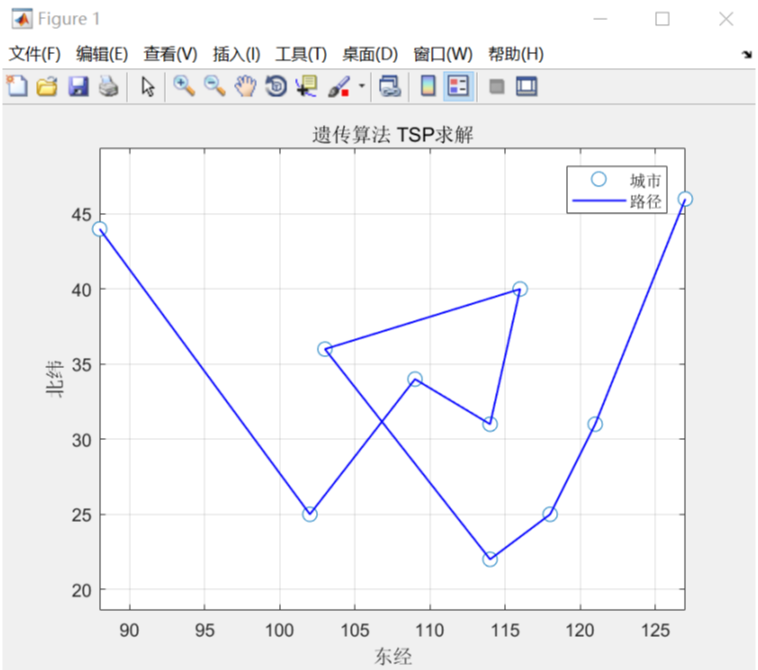
```
%% 对指定路径用指定概率更新
function [ mutatedPath ] = mutation( path, prob )
    random = rand();
    if random <= prob
        [l,length] = size(path);
        index1 = randi(length);
        index2 = randi(length);
        %交换
        temp = path(l,index1);
        path(l,index1) = path(l,index2);
        path(l,index2)=temp;
    end
        mutatedPath = path;
end
```

The path result generates a map display:

```
%% 生成地图展示
function [] = show( cities, pop, minPath, totalDistances)
    [~, length] = size(cities);
    xDots = cities(1,:);
    yDots = cities(2,:);
    title('遗传算法 TSP求解');
    plot(xDots,yDots, 'o', 'MarkerSize', 8, 'MarkerFaceColor', 'white');
    xlabel('东经');
    ylabel('北纬');
    axis equal
    hold on
    [minPathX,~] = find(totalDistances==minPath,1, 'first');
    bestPopPath = pop(minPathX, :);
    bestX = zeros(1,length);
    bestY = zeros(1,length);
    for j=1:length
       bestX(1,j) = cities(1,bestPopPath(1,j));
       bestY(1,j) = cities(2,bestPopPath(1,j));
    end
    title('遗传算法 TSP求解');
    plot(bestX(1,:),bestY(1,:), 'blue', 'LineWidth', 1);
    legend('城市', '路径');
    axis equal
    grid on
    drawnow
    hold off
end
```

【Experimental Results】

Output each iteration map and path display:



Generate results by paths of the specified algebra T:

```
命令行窗口
代数T:1 最短路径:135.48KM
代数T:2 最短路径:135.11KM
代数T:3 最短路径:135.11KM
代数T:4 最短路径:126.28KM
代数T:5 最短路径:126.28KM
代数T:6 最短路径:126.28KM
代数T:7 最短路径:119.25KM
代数T:8 最短路径:119.25KM
代数T:9 最短路径:119.25KM
代数T:10    最短路径:119.25KM
代数T:11    最短路径:114.93KM
代数T:12    最短路径:114.93KM
代数T:13    最短路径:114.93KM
代数T:14    最短路径:109.51KM
代数T:15    最短路径:109.51KM
```

```
命令行窗口
代数T:83    最短路径:103.64KM
代数T:84    最短路径:103.64KM
代数T:85    最短路径:103.64KM
代数T:86    最短路径:103.64KM
代数T:87    最短路径:103.64KM
代数T:88    最短路径:103.64KM
代数T:89    最短路径:103.64KM
代数T:90    最短路径:103.64KM
代数T:91    最短路径:103.64KM
代数T:92    最短路径:103.64KM
代数T:93    最短路径:103.64KM
代数T:94    最短路径:103.64KM
代数T:95    最短路径:103.64KM
代数T:96    最短路径:103.64KM
代数T:97    最短路径:103.64KM
代数T:98    最短路径:103.64KM
代数T:99    最短路径:103.64KM
代数T:100   最短路径:103.64KM
时间:0 分  4.270227 秒.
```

【Experiment Summary】

In this experiment, I re-understand and learn the TSP problem-solving method based on the genetic algorithm based on the experimental courseware. In the specific implementation, I initially tried to use python to implement it, but because of the problem of using some numpy data, I couldn't find the answer for a while. So I changed my mind and used matlab to complete this experiment. In terms of code composition, based on the idea of genetic algorithm, functions such as crossover are set to simulate the crossover process, using getdistance , Functions such as mu tation obtain the overall population fitness and perform path update operations. At the same time, functions such as Readfile and show are set to read data files and generate map display respectively. My algebra T is set to one hundred, as shown in the figure, and finally successfully simulated one hundred genetic iterations, and obtained the shortest path and The output was made.

After this experiment, I have a deeper understanding of the genetic algorithm and TSP problems, which has promoted the improvement of my experimental ability and the understanding of the algorithm of the subject of artificial intelligence.