**南京航空航天大学**

**研究生实验报告**

课程名称：MATLAB仿真技术与应用

课程代码：E030008

实验名称：基于GA的TSP问题研究

学生姓名：\_\_\_ 肖东 \_\_\_\_\_

班级学号：\_\_\_\_SX1503205\_\_\_

学科名称：\_导航制导与控制

所在学院：\_\_\_自动化学院\_\_\_\_

2015年 11月 29日

# 第一章 TSP 问题简介

巡回旅行商问题(Traveling Salesman Problem, TSP)是组合优化问题中的经典问题。组合优化问题的目标是从组合问题的可行解中求出最优解。在现实世界里存在大量组合优化问题,其中许多问题(如:TSP问题、图着色问题、分配问题、调度问题、布线问题及路由选择问题等)至今没有找到有效地多项式算法,这些问题已被证明是N'P完全问题。优化问题有三个基本要素:变量、约束和目标函数。在求解过程中选定的基本参数称为变量,对变量取值的限制称为约束,表示可行方案衡量标准的函数称为目标函数。组合优化问题就是在给定的约束条

件下,求目标函数最优值的问题。本文以TSP问题作为研究对象,正是由于TSP问题具有求解难度的代表性,激发了人们对优化技术的研究和对TSP问题本身的挑战,不仅TSP本身成为人们研究的热点,而且由于TSP问题的代表性,许多新的算法,理论和思想在被提出后也常常使用TSP作为测试其自身性能的标准。因此,TSP成为多种搜索、优化算法的间接比较标准,己经成为衡量优化技术成功与否的主要标准之一。

随着对优化技术的深入研究,以及计算机处理速度和内存容量的快速增长,在求解TsP问题过程中,人们取得了一个又一个的纪录。

1980年Crowder和Padberg，求解了318个城市的问题。

1987年Padberg和Rinaldi将这个城市数增加到了2392个。

1992年美国Rice大学的CRPC研究小组用50台工作站使用了基于“cutting planes”算法解决了3038个城市的问题,被《发现杂志》评为当年的前50条科学新闻。

1994年，Applegate，Bixby，Chvatal等人使用若干台SPARel作站组成的机群用了3-4年的CPU时间解决了7397个城市的TSP问题。

1998年,CRPC研究小组使用三台DigitalAlphaserver41005(12个处理器)组成的集群和32台Pentiuln一H个人计算机解决了美国13,509个城市组成的TSP问题。

2003年2月，HISao Tamaki使用了路径融合同Lin- Kernighan启发(LKH)的变种相结合的方法发现了TSPLIB中pla33810的一个次优解。2004年2月,KeldHe1Sgaun发现了plas5900问题的一个次优解。此外,他又于2003年12月发现了7,516,353,779个节点的世界TSP问题的一条比较好的解。

# 第二章 遗传算法

遗传算法主要是通过遗传操作对群体中具有某种结构形式的个体施加结构重组处理,从而不断地搜索出群体中个体间的结构相似性,形成并优化积木块以逐渐逼近最优解

## 2.1 编码

遗传算法不能直接处理问题空间的参数,必须把它们转换成遗传空间中的由基因按一定结构组成的染色体或个体,这一转换过程就叫做编码。编码是从表现型到基因型的映射。

评价一种编码策略常采用以下三个规范:

(l)完备性(completeness):问题空间中的所有解都能被构造出来。

(2)健全性(Soundness):GA中的染色体能对应所有问题空间中的候选解。

(3)非冗余性(nonredundan。y):染色体和候选解一一对应。

## 2.2 群体设定

遗传操作是对众多个体同时进行的,众多个体组成了群体。在遗传算法处理流程中,继编码设计后的任务是初始群体的设定,并以此为起点一代代进化,直到按某种进化停止准则终止进化过程,由此得到最后一代。群体设定的关键问题是—群体规模,即群体中包含个体的数目如何确定。其中有两个需考虑的因素:

(1)初始群体的设定;

(2)进化过程中各代的规模如何维持。

群体规模作为遗传算法的控制参数之一,它和交叉率、变异率等参数一样,

对于遗传算法效能的发挥有一定的影响。

**1、初始群体设定**

遗传算法中初始群体中的个体是随机产生的。一般来讲,初始群体的设定可

采用如下的策略:

(1)根据问题固有知识,设法掌握最优解所占空间在整个问题空间的分布范

围,然后在此分布范围内设定初始群体。

(2)先随机生成一定数目的个体,然后从中挑出最好的个体加到初始群体中。

这种过程不断迭代,直到初始群体中个体数达到了预先确定的规模。

**2、群体多样性**

群体规模的确定受遗传操作中选择操作的影响很大。群体规模越大,群体中个体的多样性越高,算法陷入局部解的危险就越小。所以,从考虑群体多样性出发,群体规模应较大。但是群体规模太大会带来若干弊病:一是群体越大,其适应度计算次数增加,计算量也增加,从而影响算法效率;二是群体中个体生存下来的概率大多采用和适应度成比例的方法,当群体中个体非常多时,少量适应度很高的个体会被选择而生存下来,但大多数个体却被淘汰,这会影响配对库的形成,从而影响交叉操作。另一方面,群体规模太小,会使遗传算法的搜索空间中分布范围有限,因而搜索有可能停止在未成熟阶段,引起未成熟收敛现象。显然,要避免未成熟收敛现象,必须保持群体的多样性,即群体规模不能太小。

## 2.3 适应度函数

遗传算法在进化搜索中基本上不用外部信息,仅以适应度函数为依据。适应度函数是由目标函数变换而成,对目标函数值域的某种映射变换称为适应度的尺度变换(fitnessscaling)。遗传算法的适应度函数不受连续可微的约束且定义域可以为任意集合。对适应度函数的唯一要求是,针对输入可快速计算出能加以比较的非负结果叫。这一特点使得遗传算法应用范围很广,在具体应用中,适应度函数的设计要结合求解问题本身的要求而定,适应度函数的计算是选择操作的依据,适应度函数的设计直接影响到遗传算法的性能。

## 2.4 遗传操作

遗传操作是模拟生物基因遗传的操作。在遗传算法中,通过编码组成初始群体后,遗传操作的任务就是对群体的个体按照它们对环境适应的程度施加一定的操作,从而实现优胜劣汰的进化过程。从优化搜索的角度而言,遗传操作可使问题的解,一代又一代地进化,并逼近最优解。遗传操作包括以下三个基本遗传算子:选择、交叉、变异。这三个遗传算子有如下特点:

(l)三个遗传算子的操作都是在随机扰动情况下进行的。即遗传操作是随机化操作,因此,群体中个体向最优解迁移的规则是随机的。

(2)遗传操作的效果和上述三个遗传算子所取的操作概率,编码方法,群体大小,初始群体以及适应度函数的设定密切相关。

(3)三个基本遗传算子的操作方法或操作策略随具体求解问题的不同而异。具体地讲,是和个体的编码方式直接有关。

# 第三章 遗传算法求解TSP问题的实现

## 3.1 遗传算法与组合优化

组合优化是遗传算法最基本的也是最重要的研究和应用领域之一。所谓组合优化是指在离散的、有限的数学结构上,寻找一个满足给定约束条件并使其目标函数值达到最大或最小的解。一般来说,组合优化问题通常带有大量的局部极值点,往往是不可微的、不连续的、多维的、有约束条件的、高度非线性的NP完全问题,因此,精确地求解组合优化问题的全局最优解一般是不可能的。遗传算法作为一种新型的、模拟生物进化过程的随机搜索、优化方法,近十几年来在组合优化领域得到了相当广泛的研究和应用,并已在解决诸多典型组合优化问题中显示了良好的性能和效果。

### 3.1.1 遗传算法求解TSP问题的实现

对采用遗传算法解决组合优化问题算法的一般性描述如下:

(1)确定群体规模n(整数),使用随机方法或其它方法产生n个可能解组成初始解群;

(2)对于每一个个体 (变量k为世代数,初始时k=1),计算其适应度:

(3)对于每一个体,计算其生存概率,,然后设一个随机选择器,依据以一定的随机方法产生配种个体。

(4)产生下一代种群。选取两个配种个体、,并依据一定的组合规则(如交叉、变异、逆转等将、,结合成两个新一代的个体

、,直至新一代n个个体形成完毕。

(5)重复(2)至(4)步,直至满足程序终结条件(如时间上的限制或解的质量

达到满意的范围等。

上述算法在合适的条件下,其适应度函数值会逐代增加,并收敛到某一最大值。这个算法涉及的主要问题有编码方案、适应度函数设计、约束条件处理、选择机制、遗传操作,遗传算法参数确定等。在组合优化问题的实际应用中,合理地处理以上问题,构造合适的遗传算法框架是遗传优化的关键所在。

### 3.1.2 TSP 巡回旅行商问题

TSP问题描述十分简单,简言之就是寻找一条最短的遍历n个城市的路径,或者说搜索整数子集(X 的元素表示对n个城市的编号)的一个排列使取最小值，表示城市到城市的距离

**1、编码和适应度函数**

根据TSP问题的要求,任意一个城市必须而且只能访问一次。因此在编码时,要求一个个体(即一条遍历路径)的染色体编码中不允许有重复的基因码「周。在求解TSP问题的各种遗传算法中,多采用以遍历城市的次序排列进行编码的方法,如码串82631457表示自城市8开始,依次经城市2,6,3,1,4,5,7,最后返回城市8的遍历路径。显然,这是一种针对TSP问题的最自然的编码方式,这一编码方案的主要缺陷在于造成了交叉操作的困难。另一种较为常用的编码方案是采用“边”的组合方式进行编码。例如码串24536871的第1个码2表示城市1到城市2的路径在TSP圈中,第2个码4表示城市2到城市4的路径在TSP圈中。这一编码方式有着与前面的“节点”遍历次序编码方式相类似的缺陷

适应度函数常取路径长度二的倒数,即。若结合TSP的约束条件(每个城市经过且只经过一次),则适应度函数可表示为:,其中,是对TSP路径不合法的度量(如取,为未遍历的城市的个数),。为惩罚系数,常取城市间最长距离的两倍多一点(如2.05x)。

**2、交叉策略**

基于TSP问题的顺序编码,若采取简单的一点交叉或多点交叉策略,必然产生未能完全遍历所有城市的非法路径。解决这一问题的一种处理方法是对交叉、变异等遗传操作作适当地修正,使其自动满足TSP的约束条件。针对TSP问题的交叉操作包括三种:部分匹配交叉(PMX)、顺序交叉(0X)、循环交叉(CX)，这里只介绍OX。

1985年Davis等人提出了基于路径表示的顺序交叉(Ox)操作,0x操作能够保留排列,并融合不同排列的有序结构单元。此方法开始也是选择一个匹配区域:

A：9 8 5|4 6 7|1 3 2 0

B：8 6 3|2 0 1|9 5 4 7

首先,两个交叉点之间的中间段保持不变,在其区域外的相应位置标记X,得到:

A’：X X X|4 6 7| X X X X

B’：X X X|2 0 1| X X X X

其次,记录父个体B从第二个交叉点开始城市码的排列顺序,当到达表尾时,返回表头继续记录城市码,直至到达第二个交叉点结束,这样便获得了父个体B从第二个交叉点开始的城市码排列顺序为9-5-4-7-8-6-3-2-0-l。对于父个体A而言,已有城市码4,6,7将它们从父个体B的城市码排列顺序中去掉,得到排列顺序9-5-8-3-2-0-1,再将这个排列顺序复制给父个体A,复制的起点也是从第二个交叉点开始,以此决定子个体1对应位置的未知码x,这样

新个体A”为:

A”=2 0 1 4 6 7 9 5 8 3

同样,可以产生子个体B”为:

B”= 4 6 7 2 0 1 3 9 8 5

**3、变异技术**

目前已有多种变异算子,如2一opt变异算子、倒位变异算子、启发式变异算子等,其中启发式变异算子相对其他变异算子更有效。启发式变异算子的操作过程如下:

设:P1=1 2 3 4 5 6 7 8 9

随机选择三个点,例如:2、6、8,任意交换位置可以得到5个不同个体:

AI =1 2 3 4 5 8 7 6 9

A2=1 6 3 4 5 2 7 8 9

A3=1 8 3 4 5 6 7 2 9

A4=1 6 3 4 5 8 7 2 9

A5=1 8 3 4 5 2 7 6 9

从中选择最好的作为新的个体。

# 第四章 程序代码

运用matlab的面向对象编程，程序结构清晰逻辑明了

## 4.1 个体类

classdef Individual

%%个体的类里面只存储基因和适应度

properties

gene

%%gene 是个一维数组，里面放的是走过的城市 如2,4,3,1,5 表示从2开始最后回到2的一个过程

fitness

end

methods

function obj=Individual(varargin)

%%构造函数

nargin=length(varargin);

if(nargin==0)

obj.gene=0;%脚本语言不存在数据类型的问题看着随便初始化一下就可以

obj.fitness=0;

elseif(nargin==2)

obj.gene=nargin{1};

obj.fitness=nargin{2};

else error('input error')

end

end

end

end

## 4.2 GA Engine

classdef Engine

%engine 此处显示有关此类的摘要

% 此处显示详细说明

%这个类感觉像是一个结构体和一坨static方法怎么用怎么不舒服

% GA 引擎，整个遗传算法所需要用到的方法都在这里面设置

%%注意由于matlab的类是一种伪类所以里面的方法都应写成对整个类一块功能的操作

properties

cityNum

%问题的阶数即城市的个数

distance

% distance 是二维距离数组

totalDistance

%distance的总距离

popSize

%种群数量

crossoverRate

%基因交叉概率

mutationRate

%基因突变概率

population

%种群 population是一唯数组里面存储的是Individual对象

bestFitness

worstFitness

totalFitness

aveFitness

end

methods

function obj=Engine(varargin)

%叼逼matlab 没有重载只能强行手动重载 注意参数顺序 distance,popSize,crossoverRate,mutationRate

%可以只给

nargin=length(varargin);

switch nargin

case 1 % 随便写几个case 就不写太多重载了

%只有distance

obj.distance=varargin{1};

obj.popSize=100;

obj.crossoverRate=0.7;

obj.mutationRate=0.06;

case 4

%全部都给

obj.distance=varargin{1};

obj.popSize=varargin{2};

obj.crossoverRate=varargin{3};

obj.mutationRate=varargin{4};

otherwise

error('输入信息错误')

end

obj.totalDistance=sum(sum(tril(obj.distance,-1)));%%取下三角的和

obj.cityNum=size(varargin{1},2);% size（A，1/2）1行数2列数

end

end

methods

% methods(Access='private')报错。。。

function obj=InitPop(obj)

% %初始化种群

%matlab list 无需定义直接用就可以了

for people=1:obj.popSize

obj.population(people).gene=randperm(obj.cityNum);

obj.population(people).fitness=obj.CalFitGene(obj.population(people).gene);

end

obj=obj.CalFit();%计算最佳适应度

end

function fit=CalFitGene(obj,gene)

fit=0;

for city=1:obj.cityNum-1

%% obj.population(people).gene(city)%%

%%obj.population(people).gene(city+1)%%计算的是这两者之间的距离

%%还需计算最后一个城市与第一个城市之间的距离因此加上了mod

fit=fit+obj.distance(gene(city),gene(city+1));

end

fit=fit+obj.distance(gene(1),gene(obj.cityNum));

fit=obj.totalDistance-fit;

end

function obj=CalFit(obj)

%计算种群的最佳适应度平均适应度最短距离

obj.bestFitness=obj.CalFitGene(obj.population(1).gene);

obj.worstFitness=obj.CalFitGene(obj.population(1).gene);

obj.totalFitness=0;

for people=1:obj.popSize

% obj.population(people).fitness=0;

% for city=1:obj.cityNum

% %% obj.population(people).gene(city)%%

% %%obj.population(people).gene(city+1)%%计算的是这两者之间的距离

% %%还需计算最后一个城市与第一个城市之间的距离因此加上了mod

% obj.population(people).fitness=obj.population(people).fitness+obj.distance(obj.population(people).gene(city),obj.population(people).gene(city+1));

% end

% obj.population(people).fitness=1/obj.population(people).fitness;%daoshu

obj.totalFitness= obj.totalFitness+obj.population(people).fitness;%计算总的适应度

if obj.bestFitness<obj.population(people).fitness

obj.bestFitness=obj.population(people).fitness;

end

if obj.worstFitness>obj.population(people).fitness

obj.worstFitness=obj.population(people).fitness;

end

end

obj.aveFitness=obj.totalFitness/obj.popSize;

end

end

methods

%%这里面主要是和繁殖相关的一些函数

function obj=Breed(obj)

%这个函数用于产生下一代跟换种群，

for i=1:2:obj.popSize

father=obj.Filter();

mather=obj.Filter();

babby=obj.Crossover(father,mather);

if(i<obj.popSize)

obj.population(i)=babby(1);

end

if(i+1<obj.popSize)

obj.population(i+1)=babby(2);

end

end

obj=obj.Mutation();%变异

obj=obj.CalFit();%计算最佳适应度

end

function father=Filter(obj)

%轮盘赌选出father mather

randNum=obj.totalFitness\*rand();

addNum=0;%累加数

father=obj.population(1);

for people=1:obj.popSize

addNum=addNum+obj.population(people).fitness;

if addNum>=randNum

father=obj.population(people);

break;

end

end

end

function babby=Crossover(obj,father,mather)

%这里的babby是个数组，里面有两个babby

%father mather都是Individual对象

%这里采用OX交叉

babby(1)=father;

babby(2)=mather;

if(rand()<obj.crossoverRate)

%满足要求则进行交叉

point1=randi(obj.cityNum);%采取包括point1和point2的操作

point2=randi(obj.cityNum);

if point1>point2

temp=point2;

point2=point1;

point1=temp;

end

%%始终让point2在后面

% for city=1:obj.cityNum

% babby(1).gene(city)=mather .gene(mod(city+point2,cityNum));%%babby2 是father

% babby(2).gene(city)=father .gene(mod(city+point2,cityNum));%%babby1 是mather

% end

babby(1).gene=circshift(mather.gene,[0,-point2+1]);%左移动point2

babby(2).gene=circshift(father.gene,[0,-point2+1]);

for pitch=point1:point2

repeatPoint1=find(babby(1).gene==father.gene(pitch));

babby(1).gene(repeatPoint1)=[];

repeatPoint2=find(babby(2).gene==mather.gene(pitch));

babby(2).gene(repeatPoint2)=[];

end

babby(1).gene=[father.gene(point1:point2),babby(1).gene];

babby(2).gene=[mather.gene(point1:point2),babby(2).gene];

babby(1).gene=circshift(babby(1).gene,[0,point1-1]);

babby(1).fitness=obj.CalFitGene( babby(1).gene);

babby(2).gene=circshift( babby(2).gene,[0,point1-1]);

babby(2).fitness=obj.CalFitGene( babby(2).gene);

%父亲代参与竞争

if(father.fitness>babby(1).fitness)

babby(1)=father;

end

if(mather.fitness>babby(2).fitness)

babby(2)=mather;

end

end

end

function obj=Mutation(obj)

%%变换三个位置得到一共6个选出一个适应度最高的

for people=1:obj.popSize

randNum=rand();

if randNum>obj.mutationRate

continue

else

end

obj.population(people)=obj.IndividualMutation(obj.population(people));

end

end

function mutation=IndividualMutation(obj,people)

%%%处理有点问题后期再改

mutationPoint=randperm(obj.cityNum,3);

mutation(1)=people;

for i=1:length(mutationPoint)

for j=i+1:length(mutationPoint)

%交换

mutation(i+j-1)=people;

mutation(i+j-1).gene(mutationPoint(i))=people.gene(mutationPoint(j));

mutation(i+j-1).gene(mutationPoint(j))=people.gene(mutationPoint(i));

end

end

for i=2:length(mutation)

if mutation(1).fitness<mutation(i).fitness

mutation(1)=mutation(i);

end

end

mutation= mutation(1);

end

end

end