# 遗传算法及其在 TSP 问题求解中的应用*

赵雪梅

(盐城工学院 计算机基础中心, 江苏 盐城　224001)

**摘要**:介绍了遗传算法的基本原理,讨论了遗传算法中有关编码表示和遗传算子(包括选择算子、交叉算子、变异算子)设计等方面的技术. 针对 TSP(旅行商)问题,运用基本遗传算法,研究了种群规模、进化代数、实现选择操作、交叉操作、变异操作等方面的问题,并对遗传算法的求解结果和求解效率的影响因素进行分析,最后对遗传算法解决 TSP 问题的前景进行展望.

**关键词**:遗传算法;组合优化;TSP; NP 难

**中图分类号**:TP18　　**文献标识码**:A　　**文章编号**:1006 - 0707(2009)11 - 0022 - 06

遗传算法是一种基于达尔文的生物进化论而产生的种群导向随机搜索技术和方法,它模拟生物进化的步骤,将繁殖、杂交、变异、竞争和选择等概念引入到算法中,通过对一组可行解的重新组合,改进可行解在多维空间内的移动轨迹或趋向,最终得到最优解.

遗传算法求解问题的基本思想是:从问题的解出发的,将问题的一些可行解进行编码,这些已编码的解即被当作种群中的个体(染色体);个体对环境适应能力的评价函数就是问题的目标函数;模拟遗传学中的杂交、变异、复制来设计遗传算子,用优胜劣汰的自然选择法则来指导学习和确定搜索方向;对由个体组成的种群进行演化,利用遗传算子来产生具有更高平均适应值和更好个体的种群,经过若干代后,选出适应能力最好的个体,它就是问题的最优解或近似最优解[1-2]. 本文中基于遗传算法的基本思想,讨论遗传算法的设计方法及在 TSP 问题上的应用.

## 1　遗传算法原理与算法设计

### 1.1　基本原理

遗传算法是一种借鉴生物界自然选择和自然遗传机制的随机搜索算法,由美国 J. Holland 教授提出,其主要内容是种群搜索策略和种群中个体之间的信息交换,搜索不依赖于梯度信息. 该算法是一种全局搜索算法,尤其适用于传统搜索算法难于解决的复杂和非线性问题.

选择(selection)算子、交叉(crossover)算子和变异(mutation)算子是遗传算法的 3 个主要操作算子. 遗传算法中包含了如下 5 个基本要素:①对参数进行编码;②设定初始种群大小;③设计适应度函数;④设计遗传操作;⑤设定控制参数(包括种群大小、最大进化代数、交叉率、变异率等).

### 1.2　算法描述

基于对自然界中生物遗传与进化机理的模仿,针对不同的问题,很多学者设计了许多不同的编码方法来表示问题的可行解,开发出了许多不同的遗传算子来模仿不同环境下生物的遗传特性. 这样,由不同的编码方法和不同的遗传算子构成了各种不同的遗传算法. 但这些遗传算法都有共同的特点,即通过对生物遗传和进化过程中选择、交叉、变异机理的模仿,来完成对问题最优解的自适应搜索过程. 基于这个特点,Goldberg 总结出了一种统一的最基本的遗传算法——基本遗传算法(simple genetic algorithms, SGA). 基本遗传算法只使用选择算子、交叉算子和变异算子这 3 种基本遗传算子,其遗传进化操作过程简单,容易理解,是其他一些遗传算法的雏形和基础. 它不仅给各种遗传算法提供了一个基本框架,同时也具有一定的应用价值.

基本遗传算法只使用选择算子、交叉算子、变异算子,可以形式化定义为一个 8 元组:$SGA = (C, E, Po, M, \phi, \tau, \psi, T)$,其中:$C$ 为个体的编码方法;$E$ 为个体适应度评价函数;$Po$ 为初始种群;$M$ 为种群大小;$\phi$ 为选择算子;$\tau$ 为交叉算子;$\psi$ 为变异算子;$T$ 为遗传运算终止条件.

遗传算法的应用步骤:

第 1 步　确定决策变量及其各种约束条件,即确定出个体的表现型 $X$ 和问题的解空间.

第 2 步　建立优化模型,即确定出目标函数的类型(是求目标函数的最大值还是求目标函数的最小值)及其数学描述形式或量化方法.

第 3 步　确定表示可行解的染色体编码方法,也即确定出个体的基因型 $X$ 及遗传算法的搜索空间.

第 4 步　确定解码方法,即确定出个体基因型 $X$ 到个体表现型 $X$ 的对应关系转换方法.

* **收稿日期**:2009 - 09 - 25

**作者简介**:赵雪梅(1975—),女,江苏盐城人,讲师,主要从事计算机软件开发研究.

第 5 步　确定个体适应度的量化评价方法,即确定出由目标函数值 $f(X)$ 到个体适应度 $F(X)$ 的转换规则.

第 6 步　设计遗传算子,即确定出选择运算、交叉运算、变异运算等遗传算子的具体操作方法.

第 7 步　确定遗传算法的有关运行参数,即确定出遗传算法的 $M$, $T$, $Pc$, $Pm$ 等参数.

1.3　算法设计

1.3.1　编码方法. 在遗传算法中如何描述问题的可行解,即把一个问题的可行解从其解空间转换到遗传算法所能处理的搜索空间的转换方法称为编码. 编码是应用遗传算法时要解决的主要问题,也是设计遗传算法的一个关键步骤. 编码方法除了决定个体染色体排列形式之外,还决定了个体从搜索空间的基因型变换到解空间的表现型时的解码方法. 编码方法也影响到交叉算子、变异算子等遗传算子的运算方法.

针对一个具体问题,如何设计一个完美的编码方案一直是遗传算法的应用难点之一,也是遗传算法的一个重要研究方向. 目前还没有一套既严密又完整的指导理论及评价准则能够指导我们设计编码方案. De Jong 曾提出了 2 条操作性较强的实用编码原则: ①编码原则 1(有意义积木块编码原则):应使用能易于产生与所求问题相关的且具有低阶、短定义长度的编码方案; ②编码原则 2(最小字符集编码原则):应使用能使问题得到自然表示或描述的具有最小字符集的编码方案.

迄今为止人们已经提出了许多的编码方法,总的来说可以分为 3 类:二进制编码方法、浮点数编码方法和符号编码方法.

1.3.2　适应度函数. 在研究自然界中生物的遗传和进化现象时,生物学家使用适应度这个术语来度量某个物种对其生存环境的适应程度. 对生存环境适应程度较高的物种将有更多的繁殖机会,而对生存环境适应度较低的物种,其繁殖的机会就相对较少,甚至会逐渐灭绝. 即适应度较高的个体遗传到下一代的概率就相对大一些,而适应度较低的个体遗传到下一代的概率就相对较小一些. 度量个体适应度的函数就称为适应度函数.

个体适应度大小决定该个体被遗传到下一代种群中的概率. 遗传算法仅使用所求问题的目标函数值就可以得到下一步的有关搜索信息. 目标函数值的使用是通过评价个体适应度来体现的.

评价个体适应度的一般过程: ①对个体编码串进行解码处理后,可得到个体的表现型; ②由个体的表现型可计算出对应个体的目标函数值; ③根据最优化问题的类型,由目标函数值按一定的转换规则求出个体的适应度.

但是,在仅用适应度函数来计算个体适应度时,有些遗传算法收敛很快,有些遗传算法收敛很慢,因此在运行到不同的阶段时须对个体的适应度进行适当的扩大或缩小.

适应度尺度变换(fitness scaling)即是对个体的适应度进行适当的扩大或缩小变换. 个体适应度尺度变换的 3 种方法:

1) 线性尺度变换,公式为

$$F' = aF + b$$

其中: $F$ 为原适应度; $F'$ 为尺度变换后的新适应度; $a$ 和 $b$ 为系数.

2) 乘幂尺度变换,公式为

$$F' = F^n$$

其中幂指数 $n$ 与所求解的问题有关.

3) 指数尺度变换,公式为

$$F' = \exp(-\beta F)$$

其中系数 $\beta$ 决定了选择的强制性.

1.3.3　选择算子设计. 遗传算法中使用选择算子(也叫复制算子 reproduction operator)来对种群中的个体进行优胜劣汰操作:适应度较高的个体被遗传到下一代的概率较大;适应度较低的个体被遗传到下一代的概率较小. 遗传算法中的选择操作就是用来确定如何从父代种群中按某种方法选取哪些个体遗传到下一代种群中的一种遗传运算. 选择操作建立在对个体的适应度进行评价的基础上. 选择的主要目的为了避免基因缺失,提高全局收敛性和计算效率. 最常用的几种选择策略有以下几种:

1) 轮盘赌选择. 个体适应度按比例转换为选中概率,将轮盘分成 $n$ 个扇区进行 $n$ 次选择,产生 $n$ 个 0～1 之间的随机数,相当于转动 $n$ 次轮盘,可以获得 $n$ 次转盘停止时指针位置,当指针停放在某一扇区时,该扇区代表的个体被选中. 概率越大所占转盘面积越大,被选中的机率越大.

2) 最优保存策略选择. 在遗传算法的运行过程中,通过对个体进行交叉、变异等遗传操作而不断产生新的个体. 虽然随着种群的进化过程会产生出越来越多的优良个体,但由于遗传操作的随机性,它们也有可能破坏掉当前种群中适应度最好的个体. 我们希望适应度最好的个体要尽可能的保存到下一代种群中,为达到这个目的使用最优保存策略进化模型.

3) 排序选择方法. 主要思想是:对种群中的所有个体按其适应度大小进行排序,基于这个排序来分配各个个体被选中的概率.

4) 比例选择. 个体被选中的概率与其适应度大小成正比. 由于随机操作的原因,这种选择方法的选择误差较大,有时连适应度高的个体也选不上.

5) 确定式采样选择. 按照一种确定的方式进行选择操作.

6) 无放回随机选择. 根据每个个体在下一代种群中的生存期望值来进行随机选择运算.

7) 随机联赛选择 . 每次选取适应度最高的一个个体遗传到下一代种群中.

1.3.4　交叉算子设计. 在生物的自然进化中,2 个同源染色体通过交配而重组形成新的染色体,从而产生新的个体或物种. 模仿这个环节,在遗传算法中也使用交叉算子来产生新个体. 交叉运算是指 2 个相互配对的染色体按某种方式交换其部分基因,从而形成新个体. 交叉运算是遗传算法区别于其他进化算法的重要特征,在遗传算法中起关键作用,是产生新个体的主要方法. 最常用的交叉算子是单点交叉算子. 它是在个体编码串中只随机设置一个交叉

点,然后在该点相互交换 2 个配对个体的部分染色体.

例 1 以下 2 个个体在第 4 个基因座之后进行单点交叉运算,得到 2 个新个体,如图 1 所示.

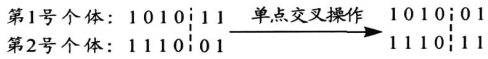


图 1 单点交叉运算得到 2 个新个体

单点交叉的重要特点是:若邻接基因座之间的关系能够提供较好的个体性状和较高的个体适应度的话,则单点交叉操作破坏这种个体性状和降低个体适应度的可能性最小.

1.3.5 变异算子设计. 在生物的遗传和自然进化中,其细胞分裂复制环节有可能会因为某些偶然因素的影响而产生一些复制差错,就会导致生物的某些基因发生某种变异,从而产生新的染色体,表现出新的生物性状. 在遗传算法中的变异运算是指将基因座上的基因值用该基因座的等位基因来替换,从而产生新个体. 从遗传运算过程中产生新个体的能力方面来说,交叉运算是产生新个体的主要方法,它决定了遗传算法的全局搜索能力;而变异运算只是产生新个体的辅助方法,但它也是一个必不可少的运算步骤,因为它决定了遗传算法的局部搜索能力. 它们共同完成对搜索空间的全局搜索和局部搜索,使遗传算法能够以良好的搜索性能完成最优化问题的寻优过程. 变异算子有以下 2 个作用:①改善遗传算法的局部搜索能力;②维持种群多样性,防止早熟现象. 较常用的有基本位变异、均匀变异、边界变异、非均匀变异和高斯变异等.

1.3.6 运行参数设计. 在遗传算法中需要选择的运行参数主要有个体编码串长度 $L$、种群大小 $M$、交叉概率 $Pc$、变异概率 $Pm$、终止代数 $T$、代数 $G$ 等.

1) 编码串长度 $L$. 使用二进制编码时,编码串长度 $L$ 的选取与问题所要求的求解精度有关;使用浮点数编码时,编码串长度 $L$ 与决策变量的个数 $n$ 相等;使用符号变量来表示个体时,编码串长度 $L$ 由问题的编码方式来确定;另外,也可以使用变长度的编码来表示个体.

2) 种群大小 $M$. 表示种群中所含个体的数量. 当 $M$ 较小时,可提高遗传算法的运算速度,却降低了种群的多样性,有可能会引起遗传算法早熟现象;而当 $M$ 较大时,又会使得遗传算法的运行效率降低. 一般 $M$ 建议取 20～100.

3) 交叉概率 $Pc$. 交叉操作是产生新个体的主要方法,一般应取较大值. 但取值过大的话,它会破坏种群中的优良模式,对进化运算反而产生不利影响;取值过小的话,产生新个体的速度较慢. 一般取 0.4～0.99.

4) 变异概率 $Pm$. 取值较大,虽然能产生较多的新个体,但有可能破坏掉很多的较好模式,使得遗传算法的性能近似于随机搜索算法的性能;取值过小的话,则变异操作产生新个体的能力和抑制早熟的能力就差. 一般取 0.000 1～0.1.

5) 终止代数 $T$. 表示遗传算法运行到指定的进化代数之后就停止,并将当前种群中的最佳个体作为所求问题的最优解输出. 一般为 100～1 000.

6) 代数 $G$. 表示每一代种群中被替换掉的个体所占的百分率,即每一代中有 $M \times G$ 个个体被替换掉.

## 2 遗传算法在 TSP 问题上的应用

### 2.1 TSP 问题的建模与描述

货郎担问题(TSP)也称为巡回旅行商问题,是一个较古老的问题,最早可以追溯到 1759 年 Euler 提出的骑士旅行问题. TSP 问题是一个典型的、容易描述但是难以处理的 NP 难问题,同时 TSP 问题也是诸多领域内出现的多种复杂问题的集中概括和简化形式. 目前求解 TSP 问题的主要方法有启发式搜索法、模拟退火算法、遗传算法、Hopfield 神经网络算法、二叉树描述算法等. 所以,有效解决 TSP 问题在计算理论上和实际应用上都有很高的价值,而且 TSP 问题由于其典型性已经成为各种启发式的搜索、优化算法的间接比较标准. 就其本质来说,遗传算法主要是解决复杂问题的一种鲁棒性强的启发式随机搜索算法,因此遗传算法在 TSP 问题求解方面的应用研究,对于构造合适的遗传算法框架、建立有效的遗传操作以及有效地解决 TSP 问题等有着多方面的重要意义[3-4].

TSP 问题可描述为:给定一组 $n$ 个城市和它们两两之间的直达距离,寻找一条闭合的旅程,使得每个城市刚好经过一次而且总的旅行路径最短. TSP 问题的描述很简单,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 $X=\{1,2,\cdots,n\}$($X$ 中的元素表示对 $n$ 个城市的编号)的一个排列 $\pi(X)=\{v_1,v_2,\cdots,v_n\}$,使 $T=\sum d(v_i,v_{i+1})+d(v_i,v_n)$ 取最小值. 式中的 $d(v_i,v_{i+1})$ 表示城市 $v_i$ 到城市 $v_{i+1}$ 的距离.

产生初始种群的方法通常有 2 种:一种是完全随机的方法产生,它适合于对问题的解无任何先验知识的情况;另一种是某些先验知识必须满足一组要求,然后在满足这些要求的解中再随机地选取样本,这样选择初始种群可使遗传算法更快地得到最优解.

下面给出了 C 语言描述的完全随机产生新种群的函数:

```
GenerateInitialPopulation() / *random */
{ int x,y;
/ * Seed the random - number generator with current time so
that the numbers will be different every time we run. */
srand( (unsigned)time( NULL) );
  for(x = 0;x < POPSIZE;x + + )
  {
  for(y = 0;y < Pathlength;y + + )
  {
  if(y = = 0)
  code[x][y] = 1;
  else
  code[x][y] = rand() %(Pathlength - y) + 1;
}
}
```

```
for(x = 0;x < POPSIZE;x + + )
code-path(x);
for(x = 0;x < POPSIZE;x + + )
  { printf("%3d ",x + 1);
  for(y = 0;y < Pathlength;y + + )
  printf("%3d",code[x][y]) ;
  printf("\ n");
}
}
```

### 2.2 对 TSP 问题的遗传基因编码实现

在遗传算法的应用中,遗传基因的编码是一个很重要的问题.对遗传基因进行编码时,要考虑到是否适合或有利于交叉和变异操作.在遗传算法中有 2 种基于串的基因编码形式:一种是基于二进制编码的遗传算法(binary coded GA);另一种是基于顺序表编码的遗传算法(the order - based GA).在求解 TSP 问题时,采用基于顺序表编码的遗传算法.

在求解 TSP 问题时,1985 年 Grefenstette 等提出了基于顺序表示(ordinal representation)的遗传基因编码方法.顺序表示是指所有城市依次排列构成一个顺序表(order list),对于一条旅程,可以依旅行经过顺序处理每个城市,每个城市在顺序表中的顺序就是一个遗传因子的表示.每处理完一个城市,从顺序表中去掉该城市.处理完所有城市以后,将每个城市的遗传因子表示连接起来,就是一条旅程的基因表示.例如,顺序表 $C = (1,2,3,4,5,6,7,8,9)$ 中一条旅程为 5—17—8—9—4—6—3—2.按照这种编码方法,这条旅程的编码 $L = (5\ 1\ 5\ 5\ 5\ 3\ 3\ 3\ 2\ 1)$.但是这种表示方法在进行单点交叉的时候,交叉点右侧部分的旅程发生了随机变化,而交叉点左侧部分的旅程未发生改变.由于存在这样的缺点,所以顺序表示的方法的适用性存在一定的问题.下面给出顺序表编码的函数"

```
pathr-code(int n) / *path change into code */
{
  int b[MaxPathlength];
  int i,j,k;
  for(i = 0;i < MaxPathlength;i + + )
  b[i] = 0;
  for(i = 0;i < Pathlength;i + + )
  { k = 0;
    for(j = 1;j < Pathlength + 1;j + + )
    { if((b[j] = = 0))
      { if(population[n].Path[i] = = j)
        { code[n][i] = j - k;
          b[j] = 1;
          break;
          } }
  else
  k + + ;
    }
    }
```

```
}
```

### 2.3 对 TSP 问题的遗传操作实现

2.3.1 选择操作实现.对于求解 TSP 问题,常用的选择机制有轮盘赌选择(roulette wheel selection)机制、最佳个体保存(elitist model)选择机制、期望值模型(expected value model)机制、排序模型(rank-based model)选择机制、联赛选择模型(tournament selection model)机制、排挤模型(crowding model)等.遗传算法中一个要求解决的问题是如何防止"早熟"收敛现象.为了保证遗传算法的全局收敛性,就要维持种群中个体的多样性,避免有效基因的丢失.Rudolhp C[5]提出采用精英选择策略,即保持种群中最好的个体不丢失,以保证算法的收敛性.Konstantin Boukreev 采用联赛选择方法和最优个体保存方法相结合的方法,通过在VC ++ 6.0环境下编程实现了求解 TSP 问题的遗传算法,取得了很好的效果.谢胜利[6]等提出浓度控制策略,即当某种个体的浓度超过给定的浓度阀值时减少该种个体的数量,使之控制在给定的浓度阀值之内,并随机产生新的个体以补足种群的规模.他们通过实验证明该策略很好地解决了遗传算法中种群的多样性问题.

笔者在该遗传算法中采用轮盘赌选择,即个体适应度按比例转换为选中概率,将轮盘分成 $n$ 个扇区进行 $n$ 次选择,产生 $n$ 个 0～1 之间的随机数,相当于转动 $n$ 次轮盘,可以获得 $n$ 次轮盘停止时的指针位置,指针停放在某一扇区,则该扇区代表的个体被选中.概率越大所占转盘面积越大,被选中的机率越大.

2.3.2 交叉操作实现.本文中用于旅行商问题求解的交叉算子用的是常规单点交叉.单点交叉又称为简单交叉,是指在个体编码串中只随机设置一个交叉点,然后在该点相互交换 2 个配对个体的部分染色体.单点交叉运算的示意图见图 2.

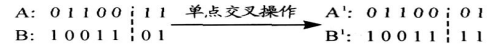


图 2 单点交叉

单点交叉运算函数如下:

```
one_point_crossover(int a,int b)
{int i,j;
  int t;
  for(i = 0;i < POPSIZE;i + + )
  path_code(i);
  / * for(i = 0;i < POPSIZE;i + + )
  {
    for(j = 0;j < Pathlength;j + + )
    printf("%d",population[i].Path[j]);
    printf("\ n");
  } */
  i = rand() %(Pathlength_2) + 1;
  for(i;i < Pathlength;i + + )
  {
  t = code[a][i];
```

```
    code[a][i] = code[b][i];
    code[b][i] = t;
    }
  }
```

2.3.3 **变异操作实现.遗传算法强调的是交叉的功能.从遗传算法的观点来看,解的进化主要靠选择机制和交叉策略来完成,变异只是为选择、交叉过程中可能丢失的某些遗传基因进行修复和补充,变异在遗传算法的全局意义上只是一个背景操作.针对** TSP **问题,陈国良**[7]**等介绍了** 4 **种主要的变异技术:**

1) **点位变异,即仅以一定的概率(通常很小)对串的某些位作值的变异.**

```
simple_mutation()
{
  int i,j,p;
  i = rand() %(POPSIZE);
  j = rand() %(Pathlength);
  p = rand() %(Pathlength-j) + 1;
  while(j = = Pathlength-1)
  j = rand() %(Pathlength-1);
  code[i][j] = p;
  for(i = 0;i < POPSIZE;i + + )
    code_path(i);
}
```

2) **逆转变异,即在编码串中,随机选择** 2 **点,再将这** 2 **点内的子串按反序插入到原来的位置中.**

```
inversion_mutation()
{
  int i;
   int x,temp,p1,p2;
   int insert[MaxPathlength];
   x = rand() %(POPSIZE);
   p1 = rand() %Pathlength;
   p2 = rand() %(Pathlength - 2) + 2;
   while(abs(p2 - p1) < 2)
   {
  srand( (unsigned)time( NULL ) );
   p2 = rand() %Pathlength;
   }
  if(p1 > p2)
   {
     temp = p1;
     p1 = p2;
     p2 = temp;
   }
   for(i = p1 + 1;i < = p2;i + + )
  {
      insert[i - p1 - 1] = population[x].Path[i];
   }
  for(i = p2;i > = p1 + 1;i - - )
   {
     population[x].Path[i] = insert[p2 - i];
   }
}
```

3) **对换变异,随机选择串中的** 2 **点,交换其码值.**

```
change_mutation()
{
  int i;
   int x,temp,p1,p2;
   int insert[MaxPathlength];
   x = rand() %(POPSIZE);
   p1 = rand() %Pathlength;
   p2 = rand() %(Pathlength - 1) + 1;
   while(abs(p2 - p1) < 1)
   {
     p2 = rand() %Pathlength;
   }
   if(p1 > p2)
   {
     temp = p1;
      p1 = p2;
     p2 = temp;
   }
      temp = population[x].Path[p1];
      population[x].Path[p1] = population[x].Path[p2];
      population[x].Path[p2] = temp;
}
```

4) **插入变异,即从串中随机选择** 1 **个码,将此码插入随机选择的插入点之后.**

```
insert_mutation()
{
  int i;
   int x,temp,p1,p2;
   int insert[MaxPathlength];
   x = rand() %(POPSIZE);
   p1 = rand() %Pathlength;
   p2 = rand() %(Pathlength - 2) + 2;
   while(abs(p2 - p1) < 2)
   {
     p2 = rand() %Pathlength;
   }
   if(p1 > p2)
   {
      temp = p1;
      p1 = p2;
      p2 = temp;
   }
     temp = population[x].Path[p2];
     for(i = p2;i > p1;i - - )
     population[x].Path[i] = population[x].Path[i - 1];
```

```
    population[x].Path[p1 + 1] = temp;
}
```

此外,对于变异操作还有一些变体形式,如 Sushil J. Louis 提出的贪心对换变异(greedy-swap mutation),其基本思想是从一个染色体中随机的选择 2 个城市(即 2 个码值),然后交换它们,得到新的染色体,以旅程长度为依据比较交换后的染色体与原来的染色体的大小,保留旅程长度值小的染色体. 谢胜利[6]等提出倒位变异算子,即在个体编码串中随机选择 2 个城市,将第 1 个城市的右边城市与第 2 个城市之间的编码倒序排列,从而产生一个新个体. Hiroaki S. 等采用 Grefenstette J J. 方法的思想,提出了 greedy subtour crossover(GSC)变异算子. 这种方法与贪心对换变异有相同的思想,但是更易扩张,更有效率. Konstantin Boukreev 实验发现:当城市大小在 200 之内时,该变异算子可以大大改善程序的运行速度,随着城市数目的增加,尤其是当城市数目达到1 000以上时,程序运行速度则非常慢[8-11].

在应用遗传算法求解 TSP 问题时,有 TSP 顺序表示和路径表示 2 种方法,这 2 种方法从根本上属于遗传基因码的向量式表示,是否充分包含一条遗传信息是令人怀疑的. 1992 年,Fox 和 McMahon[9]等提出了旅程的矩阵表示方法. 将一条旅程定义为一个有优先权的布尔矩阵 $M$,当且仅当城市 $i$ 排列在城市 $j$ 之前时矩阵元素 $M_{ij}=1$. 针对矩阵表示方法,他们设计了交(intersection)和并(union)算子. 当矩阵中位进行交运算就可获得一个合法的旅程矩阵. 一个矩阵的位子集可以安全地与另一个矩阵的位子集合并,只要这 2 个子集的交为空. 该算子将城市集合分割成 2 个分离的组:第 1 组城市从第 1 个矩阵拷贝位,第 2 组城市从第 2 个矩阵拷贝位. 由于时间和自身的水平关系,笔者没有尝试在程序中实现.

确切地说,针对 TSP 问题,在随机产生的一组数中用遗传算法产生的最优解是近似最优解. TSP 搜索空间随着城市数 $n$ 的增加而增大,所有的旅程路线组合为 $(n-1)!/2$. 当城市很多时,遗传算法表现了其强大的优势.

## 3 结果与分析

设 7 个点的 TSP 问题的测试数据如下:

0,23,93,18,40,34,13
23,0,75,4,72,74,36
93,75,0,64,21,73,51
18,4,64,0,55,52,8
40,72,21,55,0,43,64
34,74,73,52,43,0,43
13,36,51,8,64,43,0

测试的结果以及跟回溯算法的结果比较如表 1 所示.

从表 1 的结果可以看出,基本遗传算法可以求出 TSP 问题的近似最优解,但随着运行代数的增加,也可能会破坏适应度最好的个体,而得不到最优解,反而不如其它算法. 而如果把每一代适应度最好的个体保存下来,最后再选出最好的,这样随着运行代数的增加,得到的结果就越接近最优解,这就是最优保存策略. 影响遗传算法求解结果和求解效率的因素有很多,包括交叉概率、变异概率和运行代数,采用最优保存策略能使遗传算法更快收敛到某一稳定的最优解.

表 1 2 种算法结果比较

| 方法 | 权值 | 路径 |
|---|---|---|
| 回溯算法 | 184 | 1,2,4,7,3,5,6 |
| | $Pc=0.8$ | $Pm=0.05$ |
| 基本遗传算法 | 运行代数 | 权值 | 路径 |
| | 27 | 207 | 1,6,5,3,2,4,7 |
| | 36 | 184 | 1,2,4,7,3,5,6 |
| | 42 | 198 | 1,4,2,6,5,3,7 |

## 4 结束语

综上所述可以看到,遗传算法可用于求解 TSP 问题的近似最优解. TSP 问题属于 NP 难的,随着问题规模和复杂度的增加,单一的算法已经不能得到让人满意的解,混合优化策略方法必然成为研究趋势. 但就目前的研究状况来看,各种算法结合的方式虽然比较多,但还没有哪一种能够完美地解决 TSP 问题,因此混合优化策略方法是亟待发展的一个方向.

## 参考文献:


[1] 杜明,王江晴. 一个基于遗传算法的 TSP 问题解决方案[J]. 中南民族大学学报,2007(1):77-79.

[2] 杨尚达,李世平. 遗传算法研究[J]. 兵工自动化,2008,27(9):60-62.

[3] 凌应标. 基于子句权重学习的求解 TSP 问题的遗传算法[J]. 计算机学报,2005(9):1467-1475.

[4] 易敬,王平,李哲. 基于遗传算法的 TSP 问题研究[J]. 信息技术,2006(7):56-70.

[5] Rudolhp C. Convergence Properties of Canonical Genetic Algorithms[J]. IEEE Trans. Neural Networks, 2006(5): 96-101.

[6] 谢胜利. 求解 TSP 问题的一种改进的遗传算法[J]. 计算机工程与应用,2002(8):58-245.

[7] 陈国良. 遗传算法及其应用[M]. 北京:人民邮电出版社,1995.

[8] 王俊海. TSP 问题的一种高效 Memetic 算法[J]. 交通与计算机,2002(20):14-17.

[9] 黎永壹,黄泽. 遗传算法编程求解 TSP 问题[J]. 河池学院学报,2007(2):8-11.

[10] 彭青松,戴炳荣. 求解 TSP 问题的遗传算法新方法研究[J]. 福建电脑,2007(3):135-139.

[11] Fox B R, McMahon M B. Genetic Operators for the Sequencing Problems. Foundations of Genetic Algorithms[J]. Morgan Kaufmann Publishers, 2007(9):284-300.