　遗传算法 ( GA , Genetic Algorithm ) ，也称进化算法 。 遗传算法是受达尔文的进化论的启发，借鉴生物进化过程而提出的一种启发式搜索算法。因此在介绍遗传算法前有必要简单的介绍生物进化知识。

**一.进化论知识**

　　作为遗传算法生物背景的介绍，下面内容了解即可：

**种群(Population)：**生物的进化以群体的形式进行，这样的一个群体称为种群。

**个体**：组成种群的单个生物。

**基因 ( Gene ) ：**一个遗传因子。

**染色体 ( Chromosome )** ：包含一组的基因。

**生存竞争，适者生存**：对环境适应度高的、牛B的个体参与繁殖的机会比较多，后代就会越来越多。适应度低的个体参与繁殖的机会比较少，后代就会越来越少。

**遗传与变异**：新个体会遗传父母双方各一部分的基因，同时有一定的概率发生基因变异。

　　简单说来就是：繁殖过程，会发生基因交叉( Crossover ) ，基因突变 ( Mutation ) ，适应度( Fitness )低的个体会被逐步淘汰，而适应度高的个体会越来越多。那么经过N代的自然选择后，保存下来的个体都是适应度很高的，其中很可能包含史上产生的适应度最高 的那个个体。

**二.遗传算法思想**

　　借鉴生物进化论，遗传算法将要解决的问题模拟成一个生物进化的过程，通过复制、交叉、突变等操作产生下一代的解，并逐步淘汰掉适应度函数值低的解，增加适应度函数值高的解。这样进化N代后就很有可能会进化出适应度函数值很高的个体。

　　举个例子，使用遗传算法解决“0-1背包问题”的思路：0-1背包的解可以编 码为一串0-1字符串（0：不取，1：取） ；首先，随机产生M个0-1字符串，然后评价这些0-1字符串作为0-1背包问题的解的优劣；然后，随机选择一些字符串通过交叉、突变等操作产生下一代的 M个字符串，而且较优的解被选中的概率要比较高。这样经过G代的进化后就可能会产生出0-1背包问题的一个“近似最优解”。

**编码**：需要将问题的解编码成字符串的形式才能使用遗传算法。最简单的一种编码方式是二进制编码，即将问题的解编码成二进制位数组的形式。例如，问题的解是整数，那么可以将其编码成二进制位数组的形式。将0-1字符串作为0-1背包问题的解就属于二进制编码。

　　遗传算法有3个最基本的操作：选择，交叉，变异。

**选择**：选择一些染色体来产生下一代。一种常用的选择策略是 **“比例选择”**， 也就是个体被选中的概率与其适应度函数值成正比。假设群体的个体总数是M，那么那么一个体Xi被选中的概率为f(Xi)/( f(X1) + f(X2) + …….. + f(Xn) ) 。比例选择实现算法就是所谓的“轮盘赌算法”( Roulette Wheel Selection ) ，轮盘赌算法的一个简单的实现如下：

[复制代码](javascript:void(0);)

http://images.cnblogs.com/OutliningIndicators/ExpandedBlockStart.gif轮盘赌算法

/\*  
\* 按设定的概率，随机选中一个个体  
\* P[i]表示第i个个体被选中的概率  
\*/  
int RWS()  
{  
 m = 0;  
 r =Random(0,1); //r为0至1的随机数  
 for(i=1;i<=N; i++)  
 {  
 /\* 产生的随机数在m~m+P[i]间则认为选中了i  
 \* 因此i被选中的概率是P[i]  
 \*/  
 m = m + P[i];  
 if(r<=m) return i;  
 }  
}

[复制代码](javascript:void(0);)

**交叉(Crossover)**：2条染色体交换部分基因，来构造下一代的2条新的染色体。例如：

交叉前：

00000|011100000000|10000

11100|000001111110|00101

交叉后：

00000|000001111110|10000

11100|011100000000|00101

染色体交叉是以一定的概率发生的，这个概率记为Pc 。

**变异(Mutation)**：在繁殖过程，新产生的染色体中的基因会以一定的概率出错，称为变异。变异发生的概率记为Pm 。例如：

变异前：

000001110000000010000

变异后：

000001110000100010000

**适应度函数 ( Fitness Function )**： 用于评价某个染色体的适应度，用f(x)表示。有时需要区分染色体的适应度函数与问题的目标函数。例如：0-1背包问题的目标函数是所取得物品价值，但将 物品价值作为染色体的适应度函数可能并不一定适合。适应度函数与目标函数是正相关的，可对目标函数作一些变形来得到适应度函数。

**三.基本遗传算法的伪代码**

[复制代码](javascript:void(0);)

http://images.cnblogs.com/OutliningIndicators/ExpandedBlockStart.gif基本遗传算法伪代码

/\*  
\* Pc：交叉发生的概率  
\* Pm：变异发生的概率  
\* M：种群规模  
\* G：终止进化的代数  
\* Tf：进化产生的任何一个个体的适应度函数超过Tf，则可以终止进化过程  
\*/  
初始化Pm，Pc，M，G，Tf等参数。随机产生第一代种群Pop  
   
do  
{   
　　计算种群Pop中每一个体的适应度F(i)。  
　　初始化空种群newPop  
　　do  
　　{  
　　　　根据适应度以比例选择算法从种群Pop中选出2个个体  
　　　　if ( random ( 0 , 1 ) < Pc )  
　　　　{  
　　　　　　对2个个体按交叉概率Pc执行交叉操作  
　　　　}  
　　　　if ( random ( 0 , 1 ) < Pm )  
　　　　{  
　　　　　　对2个个体按变异概率Pm执行变异操作  
　　　　}  
 将2个新个体加入种群newPop中  
 } until ( M个子代被创建 )  
 用newPop取代Pop  
}until ( 任何染色体得分超过Tf， 或繁殖代数超过G )

[复制代码](javascript:void(0);)

**四.基本遗传算法优化**

 　　下面的方法可优化遗传算法的性能。

**精英主义(Elitist Strategy)选择**：是基本遗传算法的一种优化。为了防止进化过程中产生的最优解被交叉和变异所破坏，可以将每一代中的最优解原封不动的复制到下一代中。

**插入操作**：可在3个基本操作的基础上增加一个插入操作。插入操作将染色体中的某个随机的片段移位到另一个随机的位置。

**五. 使用AForge.Genetic解决TSP问题**

　　AForge.NET是一个C#实现的面向人工智能、计算机视觉等领域的开源架构。AForge.NET中包含有一个遗传算法的类库。

　　AForge.NET主页：<http://www.aforgenet.com/>

　　AForge.NET代码下载：<http://code.google.com/p/aforge/>

　　介绍一下AForge的遗传算法用法吧。AForge.Genetic的类结构如下：

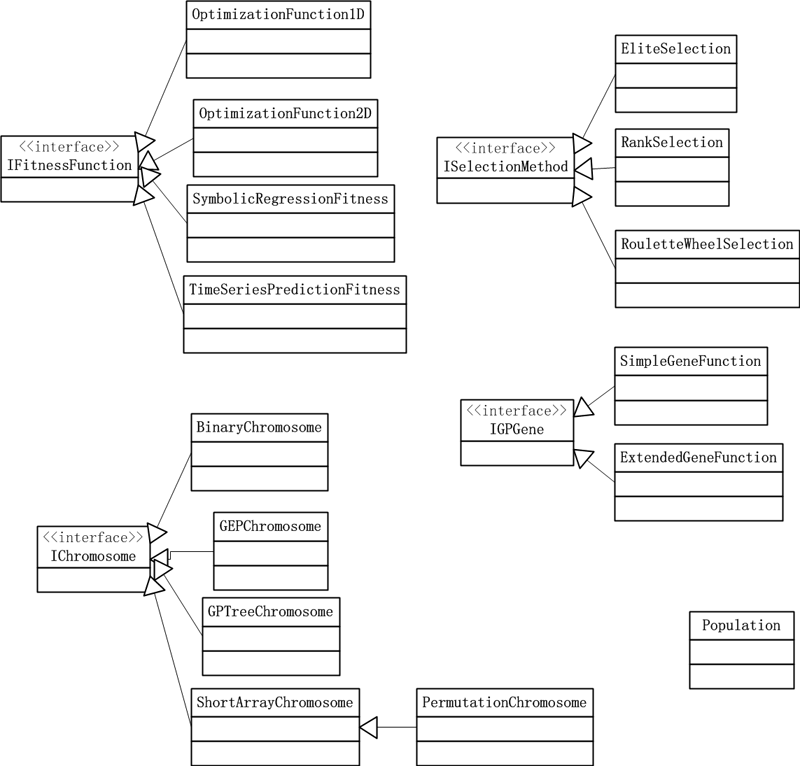


图1. AForge.Genetic的类图

   下面用AForge.Genetic写个解决TSP问题的最简单实例。测试数据集采用网上流传的中国31个省会城市的坐标:

[复制代码](javascript:void(0);)

1304 2312  
3639 1315  
4177 2244  
3712 1399  
3488 1535  
3326 1556  
3238 1229  
4196 1004  
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 2367  
3394 2643  
3439 3201  
2935 3240  
3140 3550  
2545 2357  
2778 2826  
2370 2975

[复制代码](javascript:void(0);)

**操作过程：**

   (1) 下载AForge.NET类库，网址：<http://code.google.com/p/aforge/downloads/list>

   (2) 创建C#空项目GenticTSP。然后在AForge目录下找到AForge.dll和AForge.Genetic.dll，将其拷贝到 TestTSP项目的bin/Debug目录下。再通过“Add Reference...”将这两个DLL添加到工程。

   (3) 将31个城市坐标数据保存为bin/Debug/Data.txt 。

   (4) 添加TSPFitnessFunction.cs，加入如下代码：

[复制代码](javascript:void(0);)

http://images.cnblogs.com/OutliningIndicators/ExpandedBlockStart.gifTSPFitnessFunction类

using System;  
using AForge.Genetic;  
  
namespace GenticTSP  
{  
 /// <summary>  
 /// Fitness function for TSP task (Travaling Salasman Problem)  
 /// </summary>  
 public class TSPFitnessFunction : IFitnessFunction  
 {  
 // map  
 private int[,] map = null;  
  
 // Constructor  
 public TSPFitnessFunction(int[,] map)  
 {  
 this.map = map;  
 }  
  
 /// <summary>  
 /// Evaluate chromosome - calculates its fitness value  
 /// </summary>  
 public double Evaluate(IChromosome chromosome)  
 {  
 return 1 / (PathLength(chromosome) + 1);  
 }  
  
 /// <summary>  
 /// Translate genotype to phenotype   
 /// </summary>  
 public object Translate(IChromosome chromosome)  
 {  
 return chromosome.ToString();  
 }  
  
 /// <summary>  
 /// Calculate path length represented by the specified chromosome   
 /// </summary>  
 public double PathLength(IChromosome chromosome)  
 {  
 // salesman path  
 ushort[] path = ((PermutationChromosome)chromosome).Value;  
  
 // check path size  
 if (path.Length != map.GetLength(0))  
 {  
 throw new ArgumentException("Invalid path specified - not all cities are visited");  
 }  
  
 // path length  
 int prev = path[0];  
 int curr = path[path.Length - 1];  
  
 // calculate distance between the last and the first city  
 double dx = map[curr, 0] - map[prev, 0];  
 double dy = map[curr, 1] - map[prev, 1];  
 double pathLength = Math.Sqrt(dx \* dx + dy \* dy);  
  
 // calculate the path length from the first city to the last  
 for (int i = 1, n = path.Length; i < n; i++)  
 {  
 // get current city  
 curr = path[i];  
  
 // calculate distance  
 dx = map[curr, 0] - map[prev, 0];  
 dy = map[curr, 1] - map[prev, 1];  
 pathLength += Math.Sqrt(dx \* dx + dy \* dy);  
  
 // put current city as previous  
 prev = curr;  
 }  
  
 return pathLength;  
 }  
 }  
}

[复制代码](javascript:void(0);)

   (5) 添加GenticTSP.cs，加入如下代码：

[复制代码](javascript:void(0);)

http://images.cnblogs.com/OutliningIndicators/ExpandedBlockStart.gifGenticTSP类

using System;  
using System.Collections.Generic;  
using System.Linq;  
using System.Text;  
using System.IO;  
  
using AForge;  
using AForge.Genetic;  
  
  
namespace GenticTSP  
{  
 class GenticTSP  
 {  
  
 static void Main()  
 {  
 StreamReader reader = new StreamReader("Data.txt");  
  
 int citiesCount = 31; //城市数  
  
 int[,] map = new int[citiesCount, 2];  
  
 for (int i = 0; i < citiesCount; i++)  
 {  
 string value = reader.ReadLine();  
 string[] temp = value.Split(' ');  
 map[i, 0] = int.Parse(temp[0]); //读取城市坐标  
 map[i, 1] = int.Parse(temp[1]);  
 }  
  
 // create fitness function  
 TSPFitnessFunction fitnessFunction = new TSPFitnessFunction(map);  
  
 int populationSize = 1000; //种群最大规模  
  
 /\*   
 \* 0：EliteSelection算法   
 \* 1：RankSelection算法   
 \* 其他：RouletteWheelSelection 算法  
 \* \*/  
 int selectionMethod = 0;  
  
 // create population  
 Population population = new Population(populationSize,  
 new PermutationChromosome(citiesCount),  
 fitnessFunction,  
 (selectionMethod == 0) ? (ISelectionMethod)new EliteSelection() :  
 (selectionMethod == 1) ? (ISelectionMethod)new RankSelection() :  
 (ISelectionMethod)new RouletteWheelSelection()  
 );  
  
 // iterations  
 int iter = 1;  
 int iterations = 5000; //迭代最大周期  
  
 // loop  
 while (iter < iterations)  
 {  
 // run one epoch of genetic algorithm  
 population.RunEpoch();  
  
 // increase current iteration  
 iter++;  
 }  
  
 System.Console.WriteLine("遍历路径是： {0}", ((PermutationChromosome)population.BestChromosome).ToString());  
 System.Console.WriteLine("总路程是：{0}", fitnessFunction.PathLength(population.BestChromosome));  
 System.Console.Read();  
  
 }  
 }  
}

[复制代码](javascript:void(0);)

网上据称这组TSP数据的最好的结果是 15404 ，上面的程序我刚才试了几次最好一次算出了15402.341，但是最差的时候也跑出了大于16000的结果。

我这还有一个版本，设置种群规模为1000，迭代5000次可以算出15408.508这个结果。源代码在文章最后可以下载。

总结一下使用AForge.Genetic解决问题的一般步骤：

   (1) 定义适应函数类，需要实现IFitnessFunction接口

   (2) 选定种群规模、使用的选择算法、染色体种类等参数，创建种群population

   (3)设定迭代的最大次数，使用RunEpoch开始计算