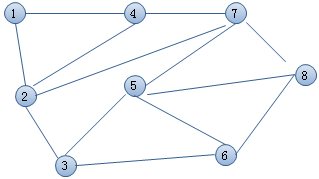
[遗传算法(Genetic Algorithm)](http://blog.chinaunix.net/uid-27105712-id-3886077.html) 2013-09-05 13:57:28

分类： LINUX

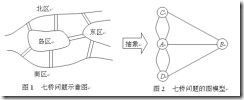
    遗传算法（Genetic Algorithm）又叫基因进化算法，或进化算法。属于启发式搜索算法一种，这个算法比较有趣，并且弄明白后很简单，写个100-200行代码就可以实现。在某些场合下简单有效。本文就花一些篇幅，尽量白话方式讲解一下。

    首先说一下问题。在我们学校数据结构这门功课的时候，时常会有一些比较经典的问题（而且比较复杂问题）作为学习素材，如八皇后，背包问题，染色问题等等。上面列出的几个问题都可以通过遗传算法去解决。本文列举的问题是TSP(Traveling Salesman Problem)类的问题。

      TSP问题实际上是”哈密顿回路问题”中的”哈密顿最短回路问题”.如下图，就是要把下面8个城市不重复的全部走一遍。有点像小时候玩的画笔画游戏，一笔到底不能重复。TSP不光是要求全部走一遍，并且是要求路径最短。就是有可能全部走一遍有很多走法，要找出其中总路程最短的走法。 



      和这个问题有点相似的是欧拉回路（下图）问题，它不是要求把每个点都走一遍，而是要求把每个边都不重复走一遍(点可以重复)，当然欧拉回路不是本算法研究的范畴。

[](http://blog.chinaunix.net/attachment/201309/5/27105712_137836063547a4.gif)

本文会从TSP引申出下面系列问题

1、  TSP问题：要求每个点都遍历到，而且要求每个点只被遍历一次，并且总路程最短。

2、  最短路径问题：要求从城市1 到城市8，找一条最短路径。

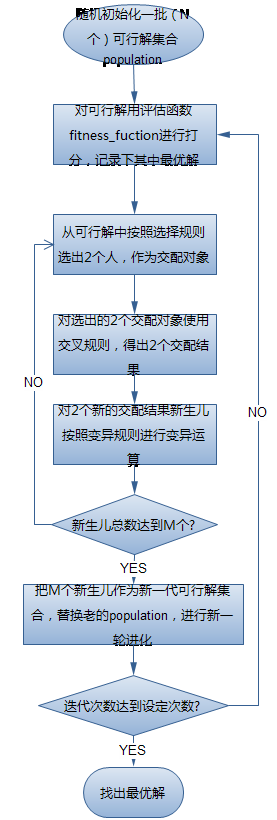
3、  遍历m个点，要求找出其距离最短的路线。(如果m=N总数，其实就是问题1了，所以问题1可以看成是问题3的特例 )。

遗传算法的理论是根据达尔文进化论而设计出来的算法: 人类是朝着好的方向（最优解）进化，进化过程中，会自动选择优良基因，淘汰劣等基因。

在上面tsp问题中，一个城市节点可以看成是一个基因，一个最优解就是一条路径，包含若干个点。就类似一条染色体有若干基因组成一样。所以求最短路径问题，可以抽象成求最优染色体的问题。

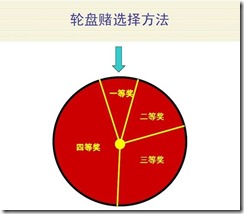
遗传算法很简单，没有什么分支判断，只有两个大循环，流程大概如下

   流程中有几个关键元素：

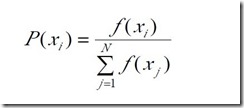


       1、  适度值评估函数。这个函数是算法的关键，就是对这个繁衍出来的后代进行评估打分，是优秀，还是一般，还是很差的畸形儿。用这个函数进行量化。在tsp中，路径越短，分数越高。函数可以可以这样 fitness = 1/total\_distance.  或者 fitness = MAX\_DISTANCE – total\_distance. 不同的计算方法会影响算法的收敛速度，直接影响结果和性能。

       2、  选择运算规则: 又称选择算子。对应着达尔文理论中适者生存，也有地方叫着精英主义原则，意思就是只有优秀的人才有更大的几率存活下来，拥有交配权。有权利拥有更多后代，传承下自己血脉基因。和现实中很相像，皇帝权臣遗留下来的子孙后代比较多。选择方法比较多。最常见的是round robin selection 算法，即轮盘赌算法, 这个算法比较简单有效。选择算法目前已有的有10来种之多。各种不同业务可以按需选择。

[](http://blog.chinaunix.net/attachment/201309/5/27105712_13783606395U55.jpg)

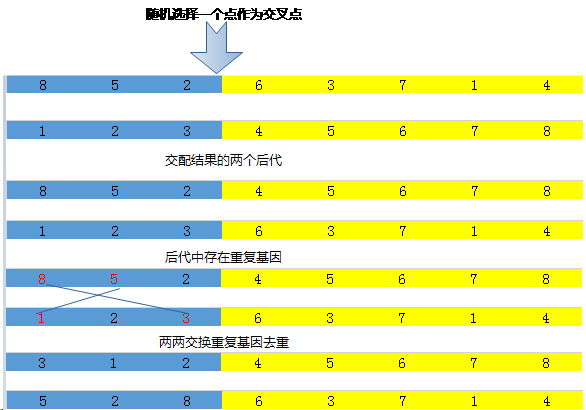
           选择公式如下:

[](http://blog.chinaunix.net/attachment/201309/5/27105712_1378360643OzwO.jpg) 

1. //选择运算---轮盘赌,此算法要求不能有负数.
2. int32\_t Genetic::Selection(Genome & selGenome)
3. {
4. //生成一个随机浮点数  
           //本算法在轮盘赌算法上加上了选择概率，提高最大可行解入围概率
5. double ftmp = (((random())%100001)/(100000 + 0.0000001));
6. if( ftmp > 0.9 )
7. {
8. GetBestGenome(selGenome);
9. return ESUCCESS;
10. }
11. //生成一个【0， m\_dTotalFitness】之间的随机浮点数
12. double dRange     = (((random()+ random())%100001)/(100000 + 0.0000001)) \* m\_dTotalFitness;
13. double dCursor    = 0.0;
14. size\_t i         = 0;
16. for(i = 0; i < m\_vGenome.size(); ++i)
17. {
18. dCursor += m\_vGenome[i].dFitness;
20. if (dCursor > dRange)
21. {
22. break;
23. }
24. }
25. selGenome = m\_vGenome[i];
27. return ESUCCESS;
28. }

      3、  交叉运算规则：又称交配规则，交叉算子。对应遗传学中的精子和卵子产生的受精卵含有精子的部分基因，也含有卵子的部分基因的现象。就像孩子有点像父亲，又有点像母亲的规律。交叉运算算法更多。作者可以天马行空的自己去想象。只要达到交叉结果中含有父母的基因就可以。最常见的是k-opt 交换。其中k可以是 1,2,3….等等。简称单点交换，两点交换，3点交换等等:

**单点交换**



           其中修复重复基因根据业务需要看是否需要。   
       两**点交换**



       4、  变异运算规则：又叫变异算子。在人类遗传进化过程中。会发生一些基因突变。这些突变有可能是好的突变，有可能是坏的突变。像癌细胞就是坏的突变。爱因斯坦的大脑估计是好的突变。突变方法也是可以天马行空的自己去发挥创造。 

这里讨论一下，为什么要有突变这道流程呢。从人类进化角度来说。人类基因有数十万种，在远古交流比较少的年代。都是部落内部通婚，但是整个部落内部居民可能都缺少某种好的基因，这样无论他们怎么交配，都不会产生好的基因，那么他们需要引入好的基因，于是和其他部落通婚。引入其他自己没有的基因，其实对于这个种群来说这就是一次基因变异。如果是好的变异，那么这个后代就很优秀，结果就是会产生更多子孙，把这个好的变异基因传承下去，如果不是好的变异基因，自然而然会在前面选择算子下淘汰，就是现实生活中的所谓的年幼夭折，痴呆无后，或先天畸形被淘汰，不会传承下去。

从计算机算法角度看：所有的启发式算法无外乎2种手段结合。局域搜索和全域搜索。局域搜索是在邻域范围内找出最优解。对应的是选择算子和交叉算子。在自己部落里面找最优秀的人。如果只有局域搜索的话，就容易陷入局域最优解。算法结果肯定是要找出全域最优解。这就要求跳出局域搜索。我们称之为“创新”。创新就是一次打破常规的突破——就是我们的“变异”算子。

这里拿最短路径路径举例子，求点1到点8之间的最短路径， 初始解是1——2——3——6——8

http://blog.chinaunix.net/attachment/201309/5/27105712_1378361513j9G0.png

**内变异：**所谓内变异就是在自己内部发生变异。严格来说其实不是一种变异。但是它又是一种比较有效的手段。

http://blog.chinaunix.net/attachment/201309/5/27105712_1378361542sZUs.png

**外变异：**外变异是引入创新，突破传统的质的飞跃, 也是启发算法中所谓的全域搜索。下面是充当前基因中引入外部基因（当前集合的补集）。

http://blog.chinaunix.net/attachment/201309/5/27105712_1378361576EJoQ.png

**结尾**：遗传算法除了上述这些几个主要算子之外，还有一些细节。如交叉概率pc，变异概率pm，这些虽然都是辅助手段，但是有时候对整个算法结果和性能带来截然不同的效果。这也是启发式算法的一个缺点。参数需要不停的在实践中摸索，没有万能的推荐参数。

还有细心的读者可能发现几个疑问，就是最短路径中变异或交叉结果可能产生无效解，如前面最短路径 1——6——3——2——8.  其中1和6之间根本没有通路。碰到这种情况，可以抛弃这条非法解，重新生成一条随机新解（其实这也是一次变异创新）。或者自己修复成可行解。反正框框在那里。具体手段可以自己天马行空发挥。

另一个比较实际的问题是：在最短路径中并不知道染色体长度是多少，不错。大部分人还是用定长染色体去解决问题，这样性能低下。算法不直观。这时候可以使用变长染色体来解决。其实我建议不管何种情况，都设计变长染色体模式。因为定长也是变长的一种特例。使用变长可以解决任何问题。不管是tsp还是最短路径问题。

还有一个编解码问题，就是把现实问题转换成基因，这些问题都比较容易解决，最简单的就是直接用数组下标表示。