遗传算法入门到掌握

读完这个讲义，你将基本掌握遗传算法，要有耐心看完。

想了很久，应该用一个怎么样的例子带领大家走进遗传 算法的神奇世界呢？遗传算法的有趣应用很多，诸如寻路问题，8 数码问题，囚犯困境，动作控制，找圆心问题（这是一个国外网友的建议：在一个不规则的多边形 中，寻找一个包含在该多边形内的最大圆圈的圆心。），TSP 问题（在以后的章节里面将做详细介绍。），生产调度问题，人工生命模拟等。直到最后看到一个非 常有趣的比喻，觉得由此引出的袋鼠跳问题（暂且这么叫它吧），既有趣直观又直达遗传算法的本质，确实非常适合作为初学者入门的例子。这一章将告诉读者，我 们怎么让袋鼠跳到珠穆朗玛峰上去(如果它没有过早被冻坏的话)。

问题的提出与解决方案

让我们先来考虑考虑下面这个问题的解决办法。

已知一元函数：$f(x)=x\sin(10\pi\cdot x)+2.0 \qquad x\in[-1,2]$

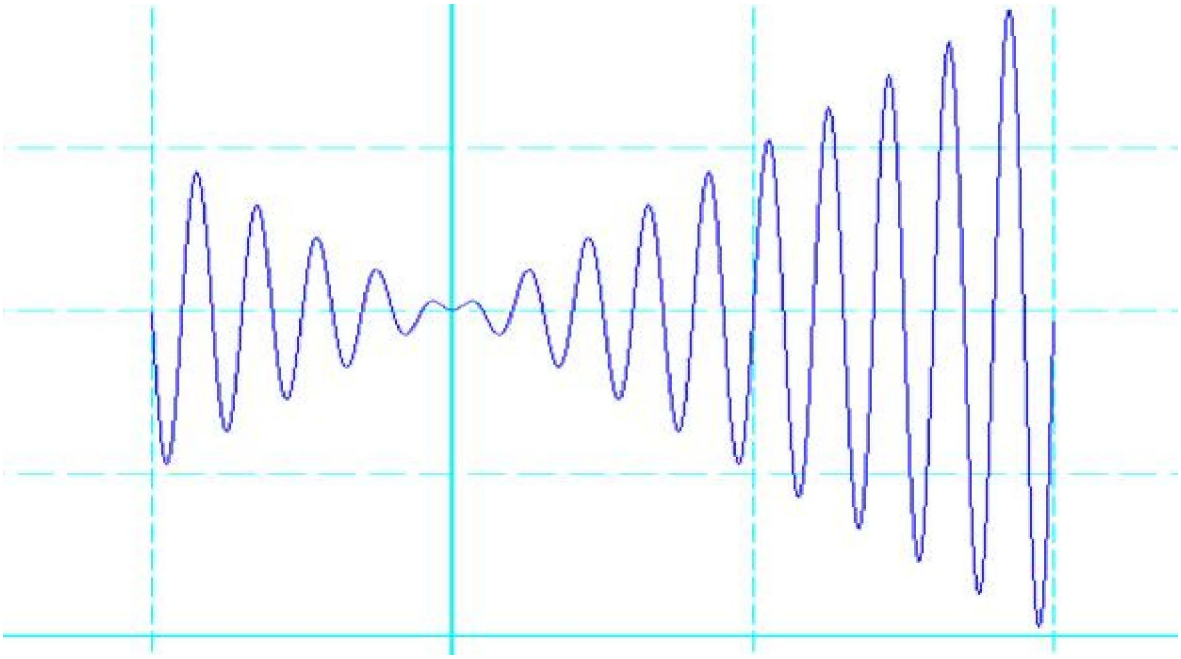


图 2-1

现在要求在既定的区间内找出函数的最大值。函数图像如图 2-1 所示。

极大值、最大值、局部最优解、全局最优解

在解决上面提出的问题之前我们有必要先澄清几个以后将常常会碰到的概念：极大值、最大值、局部最优解、全局最优解。学过高中数学的人都知道极大值在一个小邻域里面左边的函数值递增，右边的函数值递减，在图 2.1 里面的表现就是一 个“山峰”。当然，在图上有很多个“山峰”，所以这个函数有很多个极大值。而对于一个函数来说，最大值就是在所有极大值当中，最大的那个。所以极大值具有 局部性，而最大值则具有全局性。

因为遗传算法中每一条染色体，对应着遗传算法的一个 解决方案，一般我们用适应性函数（fitness function）来衡量这个解决方案的优劣。所以从一个基因组到其解的适应度形成一个映射。所以也可以把遗传算法的过程看作是一个在多元函数里面求最优 解的过程。在这个多维曲面里面也有数不清的“山峰”，而这些最优解所对应的就是局部最优解。而其中也会有一个“山峰”的海拔最高的，那么这个就是全局最优 解。而遗传算法的任务就是尽量爬到最高峰，而不是陷落在一些小山峰。（另外，值得注意的是遗传算法不一定要找“最高的山峰”，如果问题的适应度评价越小越 好的话，那么全局最优解就是函数的最小值，对应的，遗传算法所要找的就是“最深的谷底”）如果至今你还不太理解的话，那么你先往下看。本章的示例程序将会 非常形象的表现出这个情景。

“袋鼠跳”问题

既然我们把 函数曲线理解成一个一个山峰和山谷组成的山脉。那么我们可以设想所得到的每一个解就是一只袋鼠，我们希望它们不断的向着更高处跳去，直到跳到最高的山峰 （尽管袋鼠本身不见得愿意那么做）。所以求最大值的过程就转化成一个“袋鼠跳”的过程。下面介绍介绍“袋鼠跳”的几种方式。

爬山法、模拟退火和遗传算法

解决寻找最大值问题的几种常见的算法：

1. 爬山法（最速上升爬山法）：

从搜索空间中随机产生邻近的点，从中选择对应解最优的个体，替换原来的个体，不断 重复上述过程。因为只对“邻近”的点作比较，所以目光比较“短浅”，常常只能收敛到离开初始位置比较近的局部最优解上面。对于存在很多局部最优点的问题， 通过一个简单的迭代找出全局最优解的机会非常渺茫。（在爬山法中，袋鼠最有希望到达最靠近它出发点的山顶，但不能保证该山顶是珠穆朗玛峰，或者是一个非常 高的山峰。因为一路上它只顾上坡，没有下坡。）

2. 模拟退火：

这个方法来自金属热加工过程的启发。在金属热加工过程中，当金属的温度超过它的熔点（Melting Point） 时，原子就会激烈地随机运动。与所有的其它的物理系统相类似，原子的这种运动趋向于寻找其能量的极小状态。在这个能量的变

迁过程中，开始时。温度非常高， 使得原子具有很高的能量。随着温度不断降低，金属逐渐冷却，金属中的原子的能量就越来越小，最后达到所有可能的最低点。利用模拟退火的时候，让算法从较大 的跳跃开始，使到它有足够的“能量”逃离可能“路过”的局部最优解而不至于限制在其中，当它停在全局最优解附近的时候，逐渐的减小跳跃量，以便使其“落脚 ”到全局最优解上。（在模拟退火中，袋鼠喝醉了，而且随机地大跳跃了很长时间。运气好的话，它从一个山峰跳过山谷，到了另外一个更高的山峰上。但最后，它 渐渐清醒了并朝着它所在的峰顶跳去。）

3. 遗传算法：

模拟物竞天择的生物进化过程，通过维护一个潜在解的群体执行了多方向的搜索，并支持这些方向上的信息构成和交换。以面为单位的搜索，比以点为单位的搜索，更能发现全局最优解。（在遗传算法中，有很多袋鼠，它们降落到喜玛拉雅山脉的任意地方。这些袋鼠并不知道它们的任务是寻找珠穆朗玛峰。但每过几年，就在一些海拔高度较低的地方射杀一些袋鼠，并希望存活下来的袋鼠是多产的，在它们所处的地方生儿育女。） （后来，一个叫天行健的网游给我想了一个更恰切的故事：从前，有一大群袋鼠，它们被莫名其妙的零散地遗弃于喜马拉雅山脉。于是只好在那里艰苦的生活。海拔 低的地方弥漫着一种无色无味的毒气，海拔越高毒气越稀薄。可是可怜的袋鼠们对此全然不觉，还是习惯于活蹦乱跳。于是，不断有袋鼠死于海拔较低的地方，而越 是在海拔高的袋鼠越是能活得更久，也越有机会生儿育女。就这样经过许多年，这些袋鼠们竟然都不自觉地聚拢到了一个个的山峰上，可是在所有的袋鼠中，只有聚 拢到珠穆朗玛峰的袋鼠被带回了美丽的澳洲。 ）

下面主要介绍介绍遗传算法实现的过程。

遗传算法的实现过程

遗传算法的实现过程实际上就像自然界的进化过程那样。首先寻找一种对问题潜在解进行“数字化”编码的方案。（建立表现型和基因型的映射关系。）然后用随机 数初始化一个种群（那么第一批袋鼠就被随意地分散在山脉上。），种群里面的个体就是这些数字化的编码。接下来，通过适当的解码过程之后，（得到袋鼠的位置 坐标。）用适应性函数对每一个基因个体作一次适应度评估。（袋鼠爬得越高，越是受我们的喜爱，所以适应度相应越高。）用选择函数按照某种规定择优选 择。（我们要每隔一段时间，在山上射杀一些所在海拔较低的袋鼠，以保证袋鼠总体数目持平。）让个体基因交叉变异。（让袋鼠随机地跳一跳）然后产生子 代。（希望存活下来的袋鼠是多产的，并在那里生儿育女。）遗传算法并不保证你能获得问题的最优解，但是使用遗传算法的最大优点在于你不必去了解和操心如何 去“找”最优解。（你不必去指导袋鼠向那边跳，跳多远。）而只要简单的“否定”一些表现不好的个体就行了。（把那些总是爱走下坡路的袋鼠射杀。）以后你会 慢慢理解这句话，这是遗传算法的精粹！

题外话：

这里想提一提一个非主流的进化论观点：拉马克主义的进化论。

法 国学者拉马克（Jean-Baptiste de Lamarck,1744～1891）的进化论观点表述在他的《动物学哲学》（1809）一书中。该书提出生物自身存在一种是结构更加复杂化的“内驱力 ”，这种内驱力是与生俱来的，在动物中表现为“动物体新器官的产生来自它不断感觉到的新需要。”不过具体的生物能否变化，向什么方向变化，则要受环境的影 响。拉马克称其环境机制为“获得性遗传”，这一机制分为两个阶段：一是动物器官的用与不用（即“用进废退”：在环境的作用下，某一器官越用越发达，不使用 就会退化，甚至消失。）；二是在环境作用下，动物用与不用导致的后天变异通过繁殖传给后代（即“获得性遗传”）。

德 国动物学家魏斯曼（August Weismann,1834～1914）对获得性遗传提出坚决的质疑。他用老鼠做了一个著名的“去尾实验”，他切去老鼠的尾巴，并使之适应了短尾的生活。 用这样的老鼠进行繁殖，下一代老鼠再切去尾巴，一连切了22代老鼠的尾巴，第23代老鼠仍然长出正常的尾巴。由此魏斯曼认为后天后天获得性不能遗传。（择 自《怀疑----科学探索的起点》）

我 举出这个例子，一方面希望初学者能够更加了解正统的进化论思想，能够分辨进化论与伪进化论的区别。另一方面想让读者知道的是，遗传算法虽然是一种仿生的算 法，但我们不需要局限于仿生本身。大自然是非常智慧的，但不代表某些细节上人不能比她更智慧。另外，具体地说，大自然要解决的问题，毕竟不是我们要解决的 问题，所以解决方法上的偏差是非常正常和在所难免的。（下一章，读者就会看到一些非仿生而有效的算法改进。）譬如上面这个“获得性遗传”我们先不管它在自 然界存不存在，但是对于遗传算法的本身，有非常大的利用价值。即变异不一定发生在产生子代的过程中，而且变异方向不一定是随机性的。变异可以发生在适应性 评估的过程当中，而且可以是有方向性的。（当然，进一步的研究有待进行。）

所以我们总结出遗传算法的一般步骤：

开始循环直至找到满意的解。

1.评估每条染色体所对应个体的适应度。

2.遵照适应度越高，选择概率越大的原则，从种群中选择两个个体作为父方和母方。

3.抽取父母双方的染色体，进行交叉，产生子代。

4.对子代的染色体进行变异。

5.重复2，3，4步骤，直到新种群的产生。

结束循环。

接下来，我们将详细地剖析遗传算法过程的每一个细节。

编制袋鼠的染色体----基因的编码方式

通过前一章的学习，读者已经了解到人类染色体的编码符号集，由 4 种碱基的两种配合组成。共有 4 种情况，相当于 2 bit 的信息量。这是人类基因的编码方式，那么我们使用遗传算法的时候编码又该如何处理呢？

受到人类染色体结构的启发，我们可以设想一下，假设目前只有“0”，“1”两种碱基，我们也用一条链条把他们有序的串连在一起，因为每一个单位都能表现出 1 bit 的信息量，所以一条足够长的染色体就能为我们勾勒出一个个体的所有特征。这就是二进制编码法，染色体大致如下：

0100100110110111110111110

上面的编码方式虽然简单直观，但明显地，当个体特征比较复杂的时候，需要大量的编码才能精确地描述，相应的解码过程（类似于生物学中的 DNA 翻译过程，就是把基因型映射到表现型的过程。）将过份繁复，为改善遗传算法的计算复杂性、提高运算效率，提出了浮点数编码。染色体大致如下：

1.2 - 3.3 - 2.0 - 5.4 - 2.7 - 4.3

那么我们如何利用这两种编码方式来为袋鼠的染色体编码呢？因为编码的目的是建立表现型到基因型的映射关系，而表现型一般就被理解为个体的特征。比如人的基因型是 46 条染色体所描述的（总长度 两 米的纸条？），却能解码成一个个眼，耳，口，鼻等特征各不相同的活生生的人。所以我们要想为“袋鼠”的染色体编码，我们必须先来考虑“袋鼠”的“个体特 征”是什么。也许有的人会说，袋鼠的特征很多，比如性别，身长，体重，也许它喜欢吃什么也能算作其中一个特征。但具体在解决这个问题的情况下，我们应该进 一步思考：无论这只袋鼠是长短，肥瘦，只要它在低海拔就会被射杀，同时也没有规定身长的袋鼠能跳得远一些，身短的袋鼠跳得近一些。当然它爱吃什么就更不相 关了。我们由始至终都只关心一件事情：袋鼠在哪里。因为只要我们知道袋鼠在那里，我们就能做两件必须去做的事情：

（1）通过查阅喜玛拉雅山脉的地图来得知袋鼠所在的海拔高度（通过自变量求函数值。）以判断我们有没必要把它射杀。

（2）知道袋鼠跳一跳后去到哪个新位置。

如 果我们一时无法准确的判断哪些“个体特征”是必要的，哪些是非必要的，我们常常可以用到这样一种思维方式：比如你认为袋鼠的爱吃什么东西非常必要，那么你 就想一想，有两只袋鼠，它们其它的个体特征完全同等的情况下，一只爱吃草，另外一只爱吃果。你会马上发现，这不会对它们的命运有丝毫的影响，它们应该有同 等的概率被射杀！只因它们处于同一个地方。（值得一提的

是，如果你的基因编码设计中包含了袋鼠爱吃什么的信息，这其实不会影响到袋鼠的进化的过程，而那只 攀到珠穆朗玛峰的袋鼠吃什么也完全是随机的，但是它所在的位置却是非常确定的。）

以上是对遗传算法编码过程中经常经历的思维过程，必须把具体问题抽象成数学模型，突出主要矛盾，舍弃次要矛盾。只有这样才能简洁而有效的解决问题。希望初学者仔细琢磨。

既然确定了袋鼠的位置作为个体特征，具体来说位置就 是横坐标。那么接下来，我们就要建立表现型到基因型的映射关系。就是说如何用编码来表现出袋鼠所在的横坐标。由于横坐标是一个实数，所以说透了我们就是要 对这个实数编码。回顾我们上面所介绍的两种编码方式，读者最先想到的应该就是，对于二进制编码方式来说，编码会比较复杂，而对于浮点数编码方式来说，则会 比较简洁。恩，正如你所想的，用浮点数编码，仅仅需要一个浮点数而已。而下面则介绍如何建立二进制编码到一个实数的映射。

明显地，一定长度的二进制编码序列，只能表示一定精度的浮点数。譬如我们要求解精确到六位小数，由于区间长度为 2 - (-1) = 3 ,为了保证精度要求，至少把区间[-1,2]分为 3 × 10^6 等份。又因为

$$2097152 = 2^{21} < 3\times10^6 < 2^{22} = 4194304$$

所以编码的二进制串至少需要 22 位。

把一个二进制串 $(b_0\cdots b_{20}b_{21})$ 转化位区间 $[-1,2]$ 里面对应的实数值通过下面两个步骤。

（1）将一个二进制串 $(b_0\cdots b_{20}b_{21})$ 代表的二进制数转化为 10 进制数 x'：

$$(b_0\cdots b_{20}b_{21})_2 = (\sum_{i=0}^{21} b_i\cdot 2^i)_{10} = x'$$

（2） x' 对应区间 $[-1,2]$ 内的实数：

$$x = -0.1 + x'\cdot\frac{2-(-1)}{2^{22}-1}$$

例如一个二进制串<1000101110110101000111>表示实数值 0.637197。

$$x' = (1000101110110101000111)_2 = 2288967$$

$$x = -0.1 + 2288967 \cdot \frac{3}{2^{22}-1} = 0.637197$$

二进制串<00000000000000000000>和<11111111111111111111>则分别表示区间的两个端点值-1 和 2。

由于往下章节的示例程序几乎都只用到浮点数编码，所以这个“袋鼠跳”问题的解决方案也是采用浮点数编码的。往下的程序示例（包括装载基因的类，突变函数） 都是针对浮点数编码的。（对于二进制编码这里只作简单的介绍，不过这个“袋鼠跳”完全可以用二进制编码来解决的，而且更有效一些。所以读者可以自己尝试用 二进制编码来解决。）

小知识：vector（容器）的使用。

在具体写代码的过程中，读者将会频繁用到 vector 这种数据结构，所以大家必须先对它有所了解。

std::vector 是 STL（standard template library）库里面的现成的模板类。它用起来就像动态数组。利用 vector（容器）我们可以方便而且高效的对容器里面的元素进行操作。示例如下：

```
1.  //添加头文件，并使用 std 名空间。
2.
3.  #include<vector>
4.
5.  using namespace std;
6.
7.  //定义一个 vector,<>内的是这个 vector 所装载的类型。
8.
9.  vector<int> MyVector;
10.
11. //为 vector 后面添加一个整型元素 0。
12.
13. MyVector.push_back(0);
14.
15. //把 vector 的第一个元素的值赋给变量 a。值得注意的是如果 vector 的长度只有 1，
    而你
16.
17. //去访问它的下一个元素的话，编译和运行都不会报错，它会返回一个随机值给你，
    所以使
18.
19. //用的时候一定要注意这个潜伏的 BUG。
20.
```

```
int a = MyVector[0];

//把 vector 里面的元素全部清空。

MyVector.clear();

//返回 vector 里面的元素的个数。

MyVector.size()
```

呵呵，如果你没用过这个模板类，请完全不必介意，因为现在为止，你已经学会了在本书里面将用到的所有功能。

另 外，我也顺便提一提，为什么我用 vector 而不用其它数据结构比如数组，来承载一条基因，还有后面我们将会学到的神经网络中的权值向量。诚然，用数组作 为基因或者权值向量的载体，速度会快一些。但是我用 vector 主要出于下面几个考虑。首先，vector 的使用比较方便，方便得到其大小，也方便添加和 访问元素，还有排序。其次，使用 vector 也便于代码的维护与及重用（在这本书的学习过程中，学习者将会逐步建立起遗传算法和人工神经网络的引擎，通过 对代码少量的修改就能用于解决新的问题。）。另外，我还希望在研究更前缘的应用方向－－通过遗传算法动态改变神经网络的拓扑结构的时候，大家仍然可以通过 少量的修改后继续利用这些代码。（因为动态地改变神经网络的拓扑结构非常需要不限定大小的容器。）

我们定义一个类作为袋鼠基因的载体。（细心的人会提 出这样的疑问：为什么我用浮点数的容器来储藏袋鼠的基因呢？袋鼠的基因不是只用一个浮点数来表示就行吗？恩，没错，事实上对于这个实例，我们只需要用上一 个浮点数就行了。我们这里用上容器是为了方便以后利用这些代码处理那些编码需要一串浮点数的问题。）

```
class CGenome

{

public:

    //定义装载基因的容器（事实上从英文解释来看，Weights 是权值的意思，这用来表示

//基因的确有点名不符实，呵呵。这主要是因为这些代码来自于 GA-ANN 引擎，所以在

//它里面基因实质就是神经网络的权值，所以习惯性的把它引入过来就只好这样了。）

    vector <double> vecWeights;
```

```
14.
15.     // dFitness 用于存储对该基因的适应性评估。
16.
17.     double dFitness;
18.
19.     //类的无参数初始化参数。
20.
21.     CGenome():dFitness(0){}
22.
23.     //类的带参数初始化参数。
24.
25.     CGenome(vector <double> w, double f): vecWeights(w), dFitness(f){}
26.
27. };
```

好了，目前为止我们把袋鼠的染色体给研究透了，让我们继续跟进袋鼠的进化旅程。

物竞天择－－适应性评分与及选择函数。

1.物竞－－适应度函数（fitness function）

自然界生物竞争过程往往包含两个方面：生物相互间的搏斗与及生物与客观环境的搏斗过程。但在我们这个实例里面，你可以想象到，袋鼠相互之间是非常友好的，它们并不需要互相搏斗以争取生存的权利。它们的生死存亡更多是取决于你的判断。因为你要衡量哪只袋鼠该杀，哪只袋鼠不该杀，所以你必须制定一个衡量的标准。而对于这个问题，这个衡量的标准比较容易制定：袋鼠所在的海拔高度。（因为你单纯地希望袋鼠爬得越高越好。）所以我们直接用袋鼠的海拔高度作为它们的适应性评分。即适应度函数直接返回函数值就行了。

2.天择－－选择函数（selection）

自然界中，越适应的个体就越有可能繁殖后代。但是也不能说适应度越高的就肯定后代越多，只能是从概率上来说更多。（毕竟有些所处海拔高度较低的袋鼠很幸运，逃过了你的眼睛。）那么我们怎么来建立这种概率关系呢？下面我们介绍一种常用的选择方法－－轮盘赌（Roulette Wheel Selection）选择法。假设种群数目n,某个个体i其适应度为f_i,则其被选中的概率为：

$$P_i = \frac{f_i}{\sum_{i=1}^{n} f_i}$$

比如我们有 5 条染色体，他们所对应的适应度评分分别为：5，7，10，13，15。

所以累计总适应度为：$$F=\sum_{i=1}^{n} f_i = 5+7+10+13+15=50$$

所以各个个体被选中的概率分别为：

$$P_1=\frac{f_1}{F}\times 100\% = \frac{5}{50}\times 100\% = 10\%$$

$$P_2=\frac{f_2}{F}\times 100\% = \frac{7}{50}\times 100\% = 14\%$$

$$P_3=\frac{f_3}{F}\times 100\% = \frac{10}{50}\times 100\% = 20\%$$

$$P_4=\frac{f_4}{F}\times 100\% = \frac{13}{50}\times 100\% = 26\%$$

$$P_5=\frac{f_5}{F}\times 100\% = \frac{15}{50}\times 100\% = 30\%$$

呵呵，有人会问为什么我们把它叫成轮盘赌选择法啊？其实你只要看看图 2-2 的轮盘就会明白了。这个轮盘是按照各个个体的适应度比例进行分块的。你可以想象一下，我们转动轮盘，轮盘停下来的时候，指针会随机地指向某一个个体所代表的区域，那么非常幸运地，这个个体被选中了。（很明显，适应度评分越高的个体被选中的概率越大。）

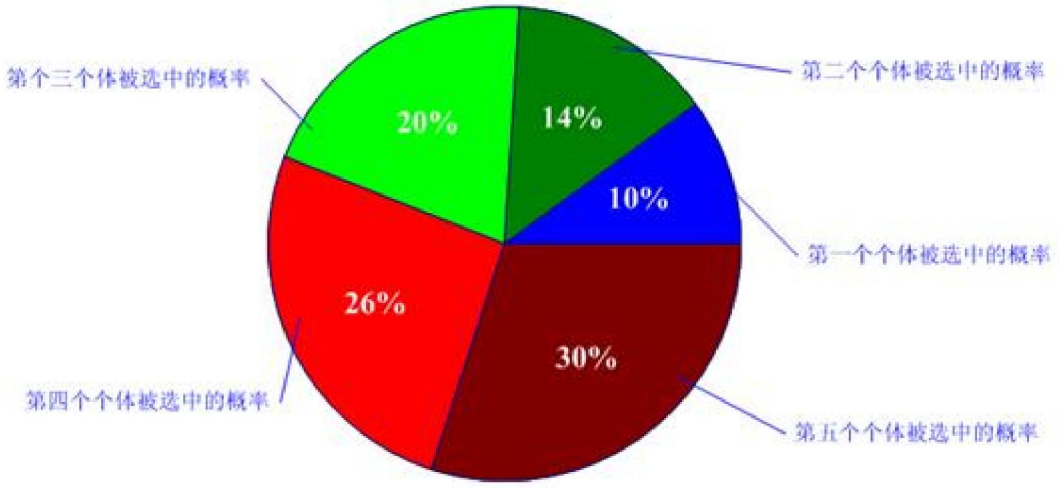


图 2-2

那么接下来我们看看如何用代码去实现轮盘赌。

```
1.  //轮盘赌函数
2.
```

```
CGenome GetChromoRoulette()

{

    //产生一个 0 到人口总适应性评分总和之间的随机数.

    //中 m_dTotalFitness 记录了整个种群的适应性分数总和)

    double Slice = (RandFloat()) * m_dTotalFitness;

    //这个基因将承载转盘所选出来的那个个体.

    CGenome TheChosenOne;

    //累计适应性分数的和.

    double FitnessSoFar = 0;

    //遍历总人口里面的每一条染色体。

    for (int i=0; i<m_iPopSize; ++i)

    {

      //累计适应性分数.

      FitnessSoFar += m_vecPop[i].dFitness;

      //如果累计分数大于随机数,就选择此时的基因.

      if (FitnessSoFar >= Slice)

      {

        TheChosenOne = m_vecPop[i];

        break;

      }

    }

    //返回转盘选出来的个体基因
```

```
47.     return TheChosenOne;
48.
49. }
```

遗传变异－－基因重组（交叉）与基因突变。

应该说这两个步骤就是使到子代不同于父代的根本原因（注意，我没有说是子代优于父代的原因，只有经过自然的选择后，才会出现子代优于父代的倾向。）。对于 这两种遗传操作，二进制编码和浮点型编码在处理上有很大的差异，其中二进制编码的遗传操作过程，比较类似于自然界里面的过程，下面将分开讲述。

1.基因重组/交叉(recombination/crossover)

（1）二进制编码

回顾上一章介绍的基因交叉过程：同源染色体联会的过程中，非姐妹染色单体（分别来自父母双方）之间常常发生交叉，并且相互交换一部分染色体，如图 2-3。事实上，二进制编码的基因交换过程也非常类似这个过程－－随机把其中几个位于同一位置的编码进行交换，产生新的个体，如图 2-4 所示。

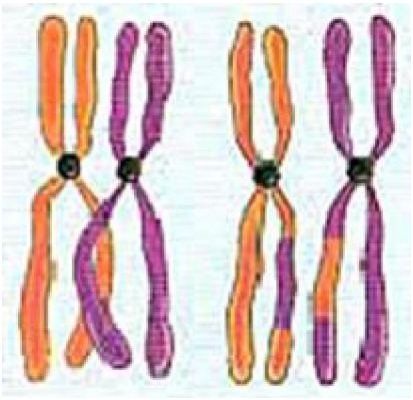


图 2-3

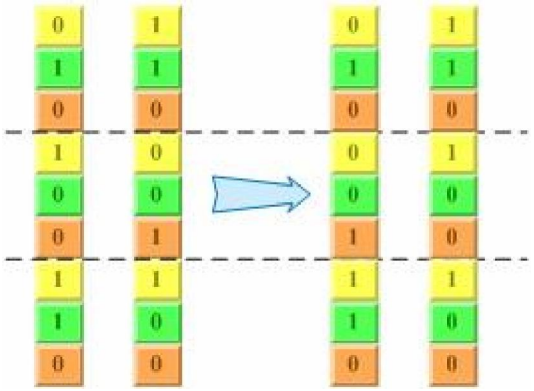


图 2-4

（2）浮点数编码

如果一条基因中含有多个浮点数编码，那么也可以用跟上面类似的方法进行基因交叉，不同的是进行交叉的基本单位不是二进制码，而是浮点数。而如果对于单个浮点数的基因交叉，就有其它不同的重组方式了，比如中间重组：

$$\text{子代编码值}=\text{父代编码在值}+\theta(\text{母方编码值}-\text{父方编码值})\quad \text{其中}\theta\in[0,1]$$

这样只要随机产生θ就能得到介于父代基因编码值和母代基因编码值之间的值作为子代基因编码的值。

考 虑到“袋鼠跳”问题的具体情况－－袋鼠的个体特征仅仅表现为它所处的位置。可以想象，同一个位置的袋鼠的基因是完全相同的，而两条相同的基因进行交叉后， 相当于什么都没有做，所以我们不打算在这个例子里面使用交叉这一个遗传操作步骤。（当然硬要这个操作步骤也不是不行的，你可以把两只异地的袋鼠捉到一起， 让它们交配，然后产生子代，再把它们送到它们应该到的地方。）

题外话：

性的起源

生命进化中另一个主要的重大进展是伴随着两性的发育－－两个生物个体间遗传物质的交换而来的。正是这种交换提供了自然选择可以发生作用的变异水平。

性 可能起源于在某种同类相食中。一个生物吞噬了另一个生物。含有双倍遗传物质的吞噬后生物为了解救自己而一分为二。这时，一种单倍遗传物质与双倍遗传物质的 单位持续相互交换替的模式就会产生。直至到达一个各项规则都适合于双倍系统的环境。在这个系统中，从双倍体到单倍体的分裂只发生在性细胞或配子形成中，然 后来自不同母体的配子结合成一个新的个体而恢复正常的双倍体系统。由于两性的出现，使进化的步伐加快了。（择自《吉尼斯－百科全书》1999 年版）

由 于基因交叉和两性有莫大的关联，所以我们可以从这个角度去深入了解基因交叉。性别的出现是在生物已经进化得相对复杂的时候。那个时候生物的基因基本形成了 一种功能分块的架构。而自然界的基因交叉过程又一般不是单个基因，或者随便几个基因的交叉，而是一块基因，往往是表现某种个体性状的那块基因，所以从宏观 上看，基因交叉的表现是性状的分离（孟德尔在实验中总结出来的规律。）。而性状又往往表现相对独立的个体特征。比如豌豆的高茎和矮茎，圆滑和皱缩。（参照 第一章对孟德尔实验的介绍。）这些都是相对独立的特征，它们之间可以自由组合互相搭配。这时候，交叉过程将起到从宏观上调整基因块之间搭配的作用。经过物 竞天择的过程，最后就能得到相对较好的特征组合方式，从而产生更优的个体。我想这才是基因交叉的意义所在吧。所以对于很多问题，使用基因交叉操作的效果不 太明显，往往只能充当跳出局部最优解，相当于大变异的功能。真正意义上的基因交叉应该使用在大规模参数的进化过程当中，它将承担起对基因块进行组合优化的 职责，从更宏观的角度去优化个体。对于交叉操作以后还将进行更具体的探讨。

2.基因突变(Mutation)

（1）二进制编码

同样回顾一下上一章所介绍的基因突变过程：基因突变是染色体的某一个位点上基因的改变。基因突变使一个基因变成它的等位基因，并且通常会引起一定的表现型变化。恩，正如上面所说，二进制编码的遗传操作过程和生物学中的过程非常相类似，基因串上的 “ 0”或“ 1”有一定几率变成与之相反的“ 1”或“ 0”。例如下面这串二进制编码：

101101001011001

经过基因突变后，可能变成以下这串新的编码：

001101011011001

（2）浮点型编码

浮点型编码的基因突变过程一般是对原来的浮点数增加或者减少一个小随机数。比如原来的浮点数串如下：

1.2, 3.4, 5.1, 6.0, 4.5

变异后，可能得到如下的浮点数串：

1.3, 3.1, 4.9, 6.3, 4.4

当 然，这个小随机数也有大小之分，我们一般管它叫“步长”。（想想“袋鼠跳”问题，袋鼠跳的长短就是这个步长。）一般来说步长越大，开始时进化的速度会比较 快，但是后来比较难收敛到精确的点上。而小步长却能较精确的收敛到一个点上。所以很多时候为了加快遗传算法的进化速度，而又能保证后期能够比较精确地收敛 到最优解上面，会采取动态改变步长的方法。其实这个过程与前面介绍的模拟退火过程比较相类似，读者可以做简单的回顾。

下面是针对浮点型编码的基因突变函数的写法：

```
1.  //基因突变函数
2.
3.  void Mutate(vector<double> &chromo)
4.
5.  {
6.
7.      //遵循预定的突变概率,对基因进行突变
8.
9.      for (int i=0; i<chromo.size(); ++i)
10.
11.     {
12.
13.         //如果发生突变的话
```

```
14.
15.        if (RandFloat() < m_dMutationRate)
16.
17.        {
18.
19.            //使该权值增加或者减少一个很小的随机数值
20.
21.            chromo[i] += (RandomClamped() * g_dMaxPerturbation);
22.
23.            //保证袋鼠不至于跳出自然保护区.
24.
25.            if(chromo[i] < g_LeftPoint)
26.
27.            {
28.
29.                chromo[i] = g_RightPoint;
30.
31.            }
32.
33.            else if(chromo[i] > g_RightPoint)
34.
35.            {
36.
37.                chromo[i] = g_LeftPoint;
38.
39.            }
40.
41.            //以上代码非基因变异的一般性代码只是用来保证基因编码的可行性。
42.
43.        }
44.
45.    }
46.
47. }
```

值得一提的是遗传算法中基因突变的特点和上一章提到的生物学中的基因突变的特点非常相类似，这里回顾一下：

1. 基因突变是随机发生的，且突变频率很低。（不过某些应用中需要高概率的变异）

2. 大多数基因变异对生物本身是有害的。

3.基因突变是不定向的。

题外话：

染色体变异

基因突变是染色体的某一个位点上基因的改变，这种改变在光学显微镜下是无法直接观察到的。而染色体变异（chromosomal variations）是可以用显微镜直接观察到的，如染色体结构的改变、染色体数目的增减等。

1.染色体结构的变异

人类的许多遗传病是由染色体结构改变引起的。例如，猫叫综合征是人的第5号染色体部分缺失引起的遗传病，因为患病儿童哭声轻，音调高，很像猫叫而得名。猫叫综合症患者的生长发育迟缓，而且存在严重的智力障碍。

在自然条件或人为因素的影响下，染色体发生的结构变异主要有以下4种类型。（如图组2-5）

（1）染色体某一段缺失引起变异。

（2）染色体中增加某一片段引起变异。

（3）染色体某一片段移接到另一条非同源染色体上引起变异。

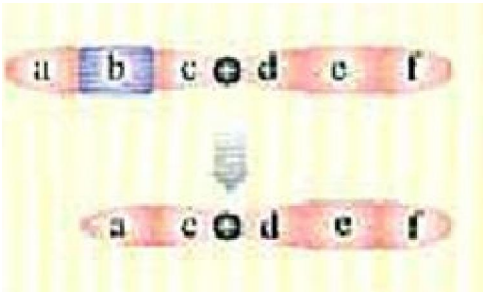
（4）染色体中某一片段位置颠倒也可引起变异。

上述染色体结构的改变，都会使排列在染色体上的基因的数目和排列顺序发生改变，从而导致性状的变异。大多数染色体结构变异对生物体是不利的，有的甚至导致生物体死亡。

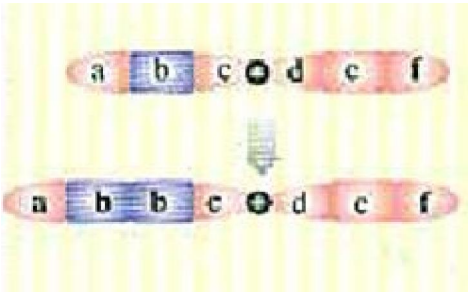
2. 染色体数目的变异

一般来说，每一种生物的染染色体数目都是稳定的，但是，在某些特定的环境条件下生物体的染色体数目会发生改变，从而产生可遗传变异。染色体数目的变异可以分为两类：一类是细胞内的个别染色体增加或减少，另一类是细胞内的染色体数目以染色体组的形式成倍地增加或减少。（择自《高中生物课本》）

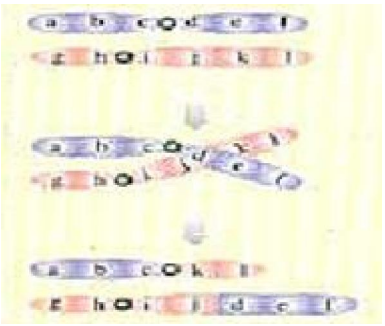
读 者应该察觉到我们用在遗传算法上的基因突变也没有包括染色体的变异过程。因为一般来说这种大规模的变异对原来的个体的基因序列破坏性比较大。所以一般来说 很难得到一个适应度高的个体。但是染色体变异，特别是染色体数目的突变使到生物从简单进化到复杂成为了可能，这也是非常具有意义的。



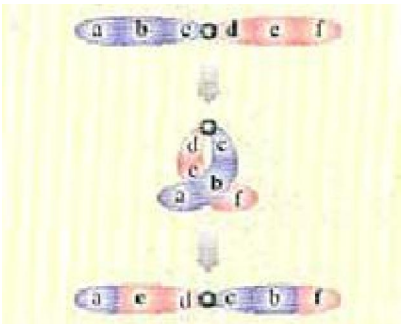
1. 染色体某一段缺失引起变异。



2. 染色体中增加某一片段引起变异。



3. 染色体某一片段移接到另一条非同源染色体上引起变异。



4. 染色体中某一片段位置颠倒也可引起变异。

图组 2-5

好了，到此为止，基因编码，基因适应度评估，基因选择，基因变异都一一实现了，剩下来的就是把这些遗传过程的“零件”装配起来了。先让我们定义一个遗传算法的类：CGenAlg

遗传算法引擎－－CGenAlg

```
1.  class CGenAlg
2.
3.  {
4.
5.  public:
6.
7.  //这个容器将储存每一个个体的染色体
8.
9.  vector <CGenome>       m_vecPop;
10.
11. //人口(种群)数量
12.
13. int m_iPopSize;
14.
15. //每一条染色体的基因的总数目
16.
17. int m_iChromoLength;
```

```
18.
19. //所有个体对应的适应性评分的总和
20.
21. double m_dTotalFitness;
22.
23. //在所有个体当中最适应的个体的适应性评分
24.
25. double m_dBestFitness;
26.
27. //所有个体的适应性评分的平均值
28.
29. double m_dAverageFitness;
30.
31. //在所有个体当中最不适应的个体的适应性评分
32.
33. double m_dWorstFitness;
34.
35. //最适应的个体在 m_vecPop 容器里面的索引号
36.
37. int m_iFittestGenome;
38.
39. //基因突变的概率,一般介于 0.05 和 0.3 之间
40.
41. double m_dMutationRate;
42.
43. //基因交叉的概率一般设为 0.7
44.
45. double m_dCrossoverRate;
46.
47. //代数的记数器
48.
49. int m_cGeneration;
50.
51. //构造函数
52.
53. CGenAlg();
54.
55. //初始化 m_dTotalFitness, m_dBestFitness, m_dWorstFitness, m_dAverageFitness 等变
    量
56.
57. void Reset();
58.
59. //初始化函数
60.
```

```
61.  void init(int popsize, double MutRate, double CrossRate, int GenLenght);
62.
63.  //计算 m_dTotalFitness, m_dBestFitness, m_dWorstFitness, m_dAverageFitness 等变量
64.
65.  void CalculateBestWorstAvTot();
66.
67.  //轮盘赌选择函数
68.
69.  CGenome GetChromoRoulette();
70.
71.  //基因变异函数
72.
73.  void Mutate(vector<double> &chromo);
74.
75.  //这函数产生新一代基因
76.
77.  void Epoch(vector<CGenome> &vecNewPop);
78.
79.  };
```

其中 Reset()函数，init()函数和 CalculateBestWorstAvTot()函数都比较简单，读者查看示例程序的代码就能明白了。而下面分别介绍 init 函数和 Epoch 函数。

类的初始化函数－－init 函数

init 函数主要充当 CGenAlg 类的初始化工作，把一些成员变量都变成可供重新开始遗传算法的状态。（为什么我不在构造函数里面做这些工作呢？因为我的程序里面 CGenAlg 类是 View 类的成员变量，只会构造一次，所以需要另外的初始化函数。）下面是 init 函数的代码：

```
1.   void CGenAlg::init(int popsize, double MutRate, double CrossRate, int GenLenght)
2.
3.   {
4.
5.       m_iPopSize = popsize;
6.
7.       m_dMutationRate = MutRate;
8.
9.       m_dCrossoverRate = CrossRate;
10.
11.      m_iChromoLength = GenLenght;
12.
```

```
13.       m_dTotalFitness = 0;
14.
15.       m_cGeneration = 0;
16.
17.       m_iFittestGenome = 0;
18.
19.       m_dBestFitness = 0;
20.
21.       m_dWorstFitness = 99999999;
22.
23.       m_dAverageFitness = 0;
24.
25.       //清空种群容器，以初始化
26.
27.       m_vecPop.clear();
28.
29.       for (int i=0; i<m_iPopSize; i++)
30.
31.       {
32.
33.            //类的构造函数已经把适应性评分初始化为 0
34.
35.            m_vecPop.push_back(CGenome());
36.
37.            //把所有的基因编码初始化为函数区间内的随机数。
38.
39.            for (int j=0; j<m_iChromoLength; j++)
40.
41.            {
42.
43.  m_vecPop[i].vecWeights.push_back(RandFloat() *
44.
45.  (g_RightPoint - g_LeftPoint) + g_LeftPoint);
46.
47.            }
48.
49.       }
50.
51. }
```

恩，正如我之前说的，我们这个程序不但要应付基因编码只有一个浮点数的“袋鼠跳”问题的情况，还希望以后在处理一串浮点数编码的时候也一样适用，所以从这里开始我们就把基因当成串来对待。

开创新的纪元－－Epoch 函数

现在万事具备了，只差把所有现成的“零件”装配起来而已。而 Epoch 函数就正好充当这个职能。下面是这个函数的实现：

```
1.  //此函数产生新的一代,见证着整个进化的全过程.
2.
3.  //以父代种群的基因组容器作为参数传进去,该函数将往该容器里放入新一代的基因组(当然是经过了优胜劣汰的)
4.
5.  void Epoch(vector<CGenome> &vecNewPop)
6.
7.  {
8.
9.      //用类的成员变量来储存父代的基因组(在此之前 m_vecPop 储存的是不带估值的所有基因组)
10.
11.     m_vecPop = vecNewPop;
12.
13.     //初始化相关变量
14.
15.     Reset();
16.
17.     //为相关变量赋值
18.
19.     CalculateBestWorstAvTot();
20.
21.     //清空装载新种群的容器
22.
23. vecNewPop.clear();
24.
25.     //产生新一代的所有基因组
26.
27.     while (vecNewPop.size() < m_iPopSize)
28.
29.     {
30.
31.         //转盘随机抽出两个基因
32.
33.         CGenome mum = GetChromoRoulette();
```

```
34.
35.             CGenome dad = GetChromoRoulette();
36.
37.             //创建两个子代基因组
38.
39.             vector<double> baby1, baby2;
40.
41.             //先把他们分别设置成父方和母方的基因
42.
43.             baby1 = mum.vecWeights;
44.
45.             baby2 = dad.vecWeights;
46.
47.             //使子代基因发生基因突变
48.
49.             Mutate(baby1);
50.
51.             Mutate(baby2);
52.
53.             //把两个子代基因组放到新的基因组容器里面
54.
55.             vecNewPop.push_back( CGenome(baby1, 0) );
56.
57.             vecNewPop.push_back( CGenome(baby2, 0) );
58.
59.         }//子代产生完毕
60.
61.         //如果你设置的人口总数非单数的话,就会出现报错
62.
63.         if(vecNewPop.size() != m_iPopSize)
64.
65.         {
66.
67.             AfxMessageBox("你的人口数目不是单数!!!");
68.
69.             return;
70.
71.         }
72.
73. }
```

呵呵，现在我们可以为袋鼠传宗接代了。（细心的读者会发现，上面每次处理两个基因个体其实是没必要的，恩，那也是为了以后能够使用交叉函数而准备的，因为交叉函数需要两个相异的个体参与。）接下来，我们要把命令袋鼠跳正式开始的函数（大家注意，这个函数非 CGenAlg 类的成员函数，而是 CSearchMaxView 类的成员函数，因为这个命令并非 CGenAlg 类自发的，而是由你“通知”CSearchMaxView 类，然后再由 CSearchMaxView 类通知 CGenAlg 类的。）也一并实现：

上帝的一声令下－－OnStartGenAlg 函数

下面将列出 OnStartGenAlg 函数的主要代码（为了不要太占版面，只列出那些关键性的代码及其解释。），读者要注意里面的适应度评价是怎么实现的。

```
1.  void CSearchMaxView::OnStartGenAlg()
2.
3.  {
4.
5.      //产生随机数
6.
7.      srand( (unsigned)time( NULL ) );
8.
9.      //初始化遗传算法引擎
10.
11.     GenAlg.init(g_popsize, g_dMutationRate, g_dCrossoverRate, g_numGen);
12.
13.     //清空种群容器
14.
15.     m_population.clear();
16.
17.     //种群容器装进经过随机初始化的种群
18.
19.     m_population = GenAlg.m_vecPop;
20.
21.     //定义两个容器，以装进函数的输入与及输出（我们这个函数是单输入单输出的，但是以后往往不会那么简单，所以我们这里先做好这样的准备。）
22.
23.     vector <double> input, output;
24.
25.     input.push_back(0);
26.
27.     for(int Generation = 0;Generation <= g_Generation;Generation++)
28.
29.     {
30.
```

```
31.             //里面是对每一条染色体进行操作
32.
33.  for(int i=0;i<g_popsize;i++)
34.
35.             {
36.
37.  input = m_population[i].vecWeights;
38.
39.  //为每一个个体做适应性评价，如之前说的，评价分数就是函数值。其
40.
41.  //Function 函数的作用是输入自变量返回函数值，读者可以参考其代码。
42.
43.                     output = Curve.Function(input);
44.
45.                     m_population[i].dFitness = output[0];
46.
47.             }
48.
49.             //由父代种群进化出子代种群（长江后浪退前浪。）
50.
51.             GenAlg.Epoch(m_population);
52.
53.         }
54.
55.  }
```

恩，到这里“袋鼠跳”的主要代码就完成了。(其它一些代码，比如图形曲线的显示，和 MFC 的相关代码在这就不作介绍了，建议初学者不必理会那些代码，只要读懂算法引擎部分就可以了。)下面就只等着我们下达命令了！

让袋鼠在你的电脑里进化－－程序的运行

我 想没有什么别的方法比自己亲手写一个程序然后通过修改相关参数不断调试程序，更能理解并且掌握一种算法了。不知道你还记不记得你初学程序的日子，我想你上 机动手写程序比坐在那里看一本厚厚的程序开发指南效率不知高上多少倍，兴趣也特命浓厚，激情也特别高涨。恩，你就是需要那样的感觉，学遗传算法也是一样 的。你需要把自己的代码运行起来，然后看看程序是否按照你所想象的去运行，如果没有，你就要思考原因，按照你的想法去改善代码，试着去弄清其中的内在联 系。这是一个思维激活的过程，你大脑中的神经网络正在剧烈抖动（呵呵，或许学到后面你就知道你大脑的神经网络是如何“抖动”的。），试图去接受这新鲜而有 趣的知识。遗传算法（包括以后要学到的人工神经网络）包含大量的可控参数，比如进化代数、人口数目、选择概率、交叉概

率、变异概率、变异的步长还有以后学 到的很多。这些参数之间的搭配关系，不能指望别人用“灌输”的方式让你被动接受，这需要你自己在不断的尝试，不断的调整中去形成一种“感觉”的。很多时候 一个参数的量变在整个算法中会表现出质的变化。而算法的效果又能从宏观上反映参数的设置。

现在就让我们来对这个程序做简单的说明。

参数的设置：

这个程序有很多的需要预先设置好的参数，为了方便修改，我把它们都定义为全局变量，定义和初始化都放在 Parameter.h 的头文件里面。下面对几个主要参数的说明：

```
1.  //目标函数的左右区间，目前的设置是[-1,2]
2.
3.  double g_LeftPoint = -1;
4.
5.  double g_RightPoint = 2;
6.
7.  ////遗传算法相关参数////
8.
9.  int g_numGen = 1;      //每条染色体的编码个数，这里是 1 个
10.
11. int g_Generation = 1000;     //进化的代数
12.
13. int g_popsize = 50;      //种群的人口数目（就是说你要放多少只袋鼠到山上）
14.
15. double g_dMutationRate = 0.8;   //基因变异的概率
16.
17. double g_dMaxPerturbation = 0.005;  //基因变异的步长（袋鼠跳的最大距离）
```

当然，一些主要的参数在程序运行后可以通过参数设置选项进行设置。（其中缓冲时间是每进化一代之后，暂停的时间，单位为毫秒）如图 2-6。

图 2-6

运行程序：

程序运行后请选择菜单项：控制—>让袋鼠们开始跳吧，开始遗传算法的过程。其中蓝色的线条是函数曲线（恩，那就是喜玛拉雅山脉。其中最高的波峰，就是珠穆朗玛峰。）绿色的点是一只只袋鼠。上方的黑色曲线图表是对每一代最优的个体的适应性评分的统计图表。下方的黑色曲线图表是对每一代所有个体的平均适应性评分的统计图表。（如果你认为它们阻碍了你的视线，你可以在参数设置里面取消掉。）如图 2-7 所示。另外还可以用键盘的上下左右键来控制视窗的移动，加减键控制函数曲线的放缩。

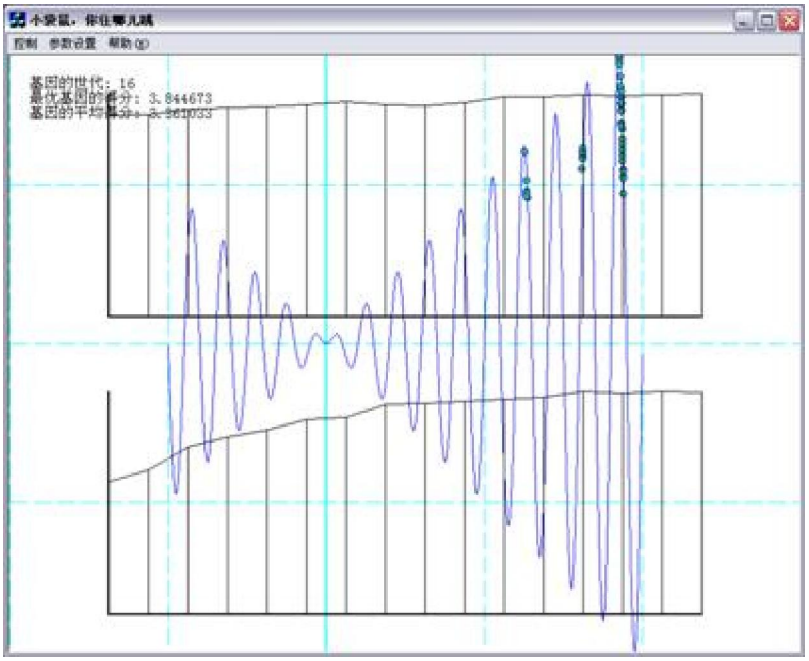


图 2-7

刚开始的时候，袋鼠分布得比较分散它们遍布了各个山岭，有的在高峰上，有的在深谷里。（如图 2-8）

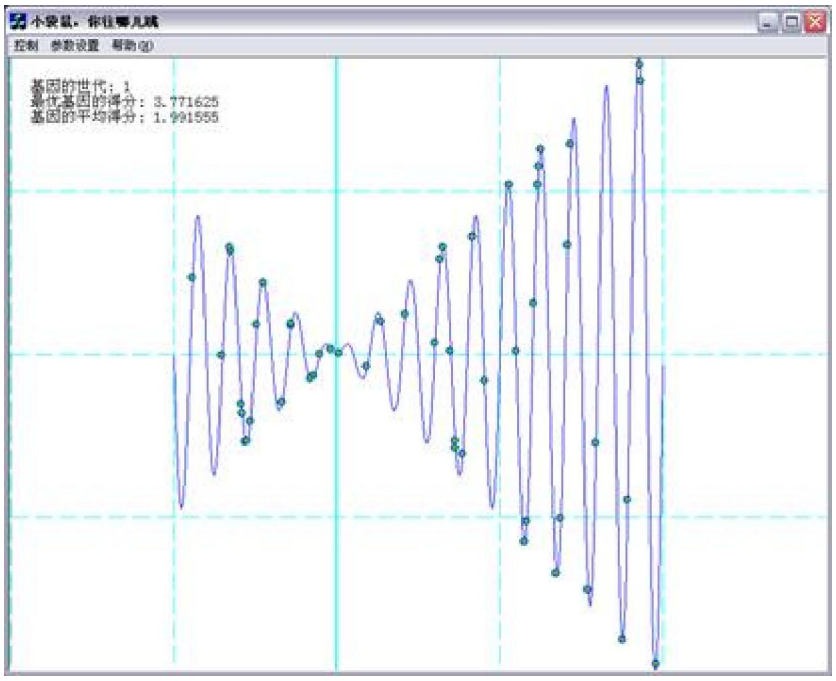


图 2-8

经过了几代的进化后，一些海拔高度比较低的都被我们射杀了，而海拔较高的袋鼠却不断的生儿育女。（如图 2-9）

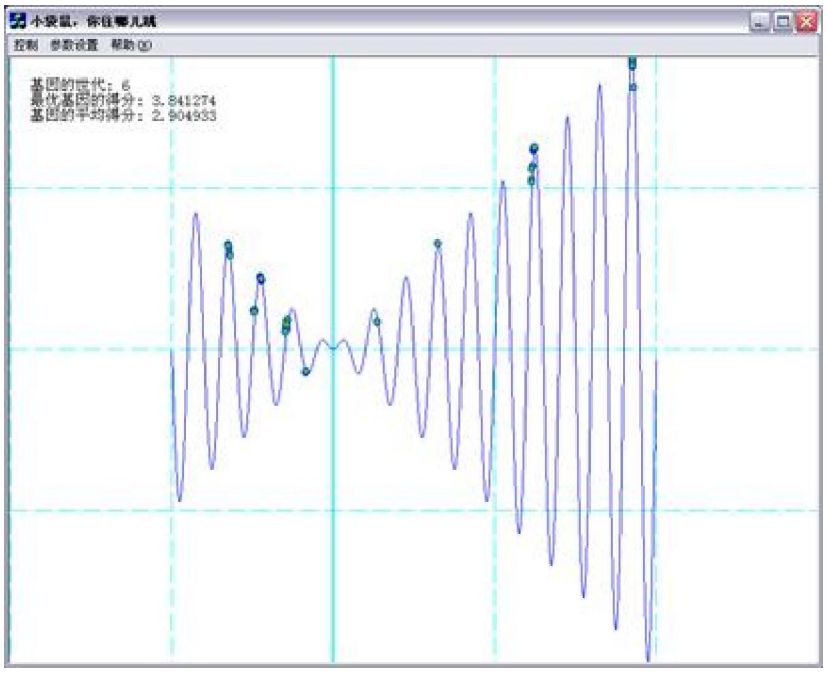


图 2-9

最后整个袋鼠种群就只出现在最高峰上面（最优解上）。（如图 2-10）

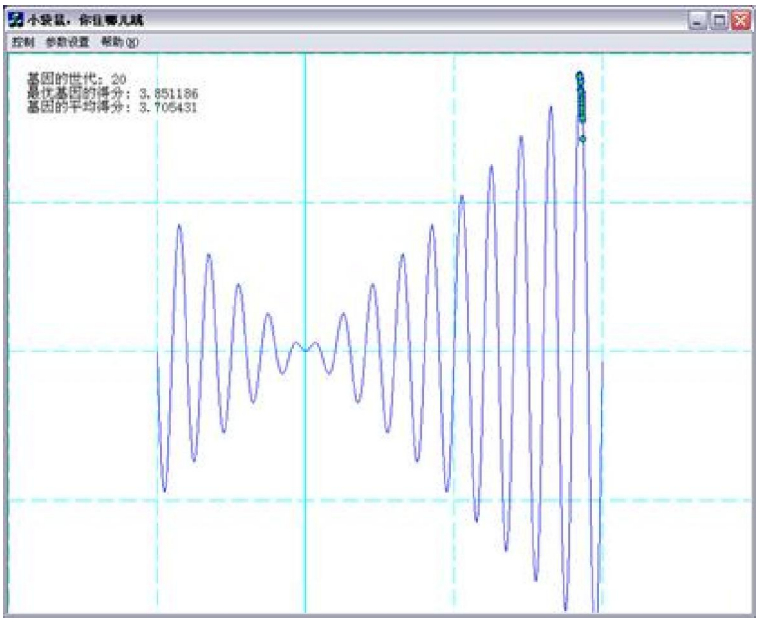


图 2-10

当然，袋鼠不是每一次都能跳到珠穆朗玛峰的，如图 2-11 所 示。（就是说不是每次都能收敛到最优解）也许它们跳到了某一个山峰，就自大的认为它们已经“会当凌绝顶”了。（当然，事实上是因为不管它们向前还是向后跳 都只能得到更小的适应度，所以不等它们跳过山谷，再跳到旁边更高的山峰，就被我们射杀了。）所以，我们为了使到袋鼠每次都尽可能的攀到珠穆朗玛峰，而不是 留恋在某一个低一些的山峰，我们有两个改进的办法，其一是初始人口数目更多一些，以使最好有一些袋鼠一开始就降落到最高峰的附近，但是这种方法对于搜索空 间非常大的问题往往是无能为力的。我们常常采用的方法是使袋鼠有一定的概率跳一个很大的步长，以使袋鼠有可能跳过一个山谷到更高的山峰去。这些改进的方法留给读者自己去实现。

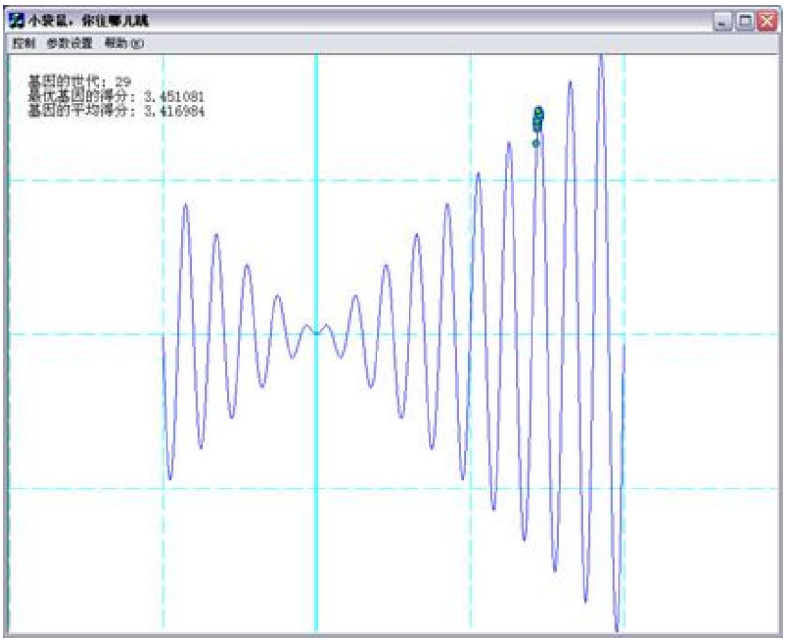


图 2-11

另外，如果把变异的机率调得比较高，那么就会出现袋鼠跳得比较活跃的局面，但是很可能丢失了最优解；而如果变异的机率比较低的话，袋鼠跳得不太活跃，找到最优解的速度就会慢一些，这也留给读者自己去体验。

作为一个寻找大值的程序，这个的效率还 很低。我希望留给初学者更多改进的空间，大家不必受限于现有的方法，大可以发挥丰富的想象力，自己想办法去提高程序的效率，然后自己去实现它，让事实去验 证你的想法是否真的能提高效率，抑或刚好相反。恩，在这个过程当中，大家不知不觉地走进了遗传算法的圣殿了，胜于一切繁复公式的摆设和教条式的讲解。

总结与及扩充

经 过本章的学习，我想读者应该能基本上把握遗传算法的基本步骤与及隐约的看到了她的本质。当然同时还会带着许多许多的疑问和不解。好的，不必急躁，让我们在 以后的章节中慢慢领会。下面我们回顾一下前面所学过的内容，同时也做一些扩充。（为了适应学习新知识的客观规律，我对知识点的介绍所遵循的原则是：先对理 论作简单的介绍，目的是让读者对新鲜理论有一个感性的认识。然后用实际的例子实现理论并且在实践中加深对理论的理解。最后对理论作更为深入系统的总结与及 扩充。）

对编码方式的回顾与扩充

1．二进制编码

二进制编码的编码符号集由 0 和 1 组成，因此染色体是 一个二进制符号串，其优点在于编码、解码操作简单，交叉、变异等遗传操作便于实现，对于全局搜索能力有一定的优势;其缺点在于，不便于反映所求间题的特定 知识，对于一些连续函数的优化问题等，也由于遗传算法的随机特性而使得其局部搜索能力较差，对于一些多维、高精度要求的连续函数优化，二进制编码存在着连 续函数离散化时的映射误差，个体编码串较短时，可能达不到精度要求;而个体编码串的长度较长时，虽然能提高精度，但却会使算法的搜索空间急剧扩大。如果个 体编码串特别长时，会造成遗传算法的性能降低。

2．浮点数编码

浮点数编码方式，以浮点数为编码的单位。就二进制编码和浮点数编码比较而言，浮点数编码一些情况下比较能反映所求问题的特定知识，编码结构一般比二进制来得简单些。一般二进制编码比浮点数编码搜索能力强，但浮点数编码比二进制编码在变异操作上能够保持更好的种群多样性。

3．其它编码方式

其实编码的方式是多种多样的，有时候还会用到混合编 码，而且编码形式对具体问题的依赖性比较强。设计编码的时候不必拘泥于现有的几种编码方式，解决具体问题的时候，很多情况下需要为具体问题“度身定做”。 有时候一种合适的编码方式，配合合适的交换算子，变异算子（交换算子和变异算子常常需要适合特定的编码方式。），这些都会影响到解决问题的效率，在以后的 深入学习过程中大家将会有深刻体会。（下一章的例子将用到混合编码。）

接下来总结出遗传算法选取编码过程的几个原则：

1.完全性，原则上问题的所有可能的解都能找到与之对应的编码组合。

2.合法性，每个基因编码都对应一个可接受的个体。

3.多重性，多个基因型解码成一个表现型，即从基因型到相应的表现型空间是多对一的关系，这是基因的多重性。若相同的基因型被解码成不同的表现型，这是表现型多重性。当然，基因型到表现型的映射关系最好是一对一的关系。

4.紧致性,若两种基因编码能解码成相同的个体，那么占用空间越小的编码方式就越紧致。

5.复杂性，指基因型结构的复杂性，解码的复杂性，计算时空的复杂性。

这些特征常常是鱼与熊掌，不可兼得的。（整理《遗传算法－－理论、应用与软件实现》相关资料而来）

对适应性函数的回顾与扩充

适 应性函数有一个更形象的名字－－压力函数。为什么这样说呢？如果你对遗传算法没有一定程度上的理解的话很难把握它的意思,但是经过上面那个例子－－对“袋 鼠逃”问题解决，读者会发现经过一段时间的进化过程，袋鼠都被无形的力“压”到了山顶。这其实是适应性函数的力量，如果你喜欢的话，你可以通过对适应性函 数的作简单修改，就能把袋鼠“压”到谷底。（建议初学者自己尝试尝试如何修改，虽然简单，但是你不一定那么容易成功的。）由此可见适应性函数是一个影响进 化趋势的函数，有非常重要的地位。

尺度变换(fitness scaling)

并不是每个问题的适应性函数都像“袋鼠跳”问题的那么简单明了。我们常常需要对目标函数值作一些变换。这种对目标函数值域的映射变换就称为适应度的尺度变换(fitness scaling)。下面是几种常见的尺度变换。（$F(x)$为适应度函数，$f(x)$为目标函数）

（1）当目标函数为最大问题时：（C_{min}是$f(x)$的最小值估计）

$$F(x)=\begin{cases} f(x)-C_{min}, & f(x)>C_{min} \\ 0, & f(x)\le C_{min} \end{cases}$$

当目标函数为最小问题时：（C_{max}是$f(x)$的最大值估计）

$$F(x)=\begin{cases} C_{max}-f(x), & f(x)<C_{max} \\ 0, & f(x)\ge C_{max} \end{cases}$$

（2）当目标函数为最大问题时：（C是$f(x)$界限的保守估计值）

$$F(x)=\frac{1}{1+c-f(x)} \qquad c\ge 0, c-f(x)\ge 0$$

当目标函数为最小问题时：

$$F(x)=\frac{1}{1+c+f(x)} \qquad c\ge 0, c+f(x)\ge 0$$

下面介绍一下适应度函数应该遵循的几个原则。

(1)适应性评分应该是非负的：这主要是因为轮盘赌函数只能处理非负的适应度评分，否则会打乱个体的选择概率的。（回想前面我要大家尝试一下把袋鼠“压”到山谷下，我想很多初学者都会想到把适应性评分取$-f(x)$，但是这样做的话会出现问题的，为什么呢？主要是因为轮盘赌函数的问题，它需要正数才能正常执行，所以就需要上面提到的尺度变换的第一类办法，就是保证适应度评价是正数。）

(2)合理、一致性：要求适应度值反映对应解的优劣程度。

(3)计算量小：适应度函数设计应尽可能简单，这样可以减少计算时间和空间上的复杂

性，降低计算成本。

欺骗

在使用遗传算法解决实际问题的时候我们常常遇到一些简单的问题，但是遗传算法却又难以收敛到最优解上面。是什么原因导致一个简单的问题对遗传算法来说是难以求解的呢？我们称那些引导遗传算法出错的函数编码组合为遗传算法的欺骗问题。一般来说，欺骗的出现可能是下面的原因：

（1） 在遗传进化的初期，产生一些适应性评分特别高的个体，若按照比例选择法，这些个体因竞争力太突出而控制了选择过程，影响算法的全局优化性能。（在袋鼠跳问 题里面，大家也许看见过这样的情景：当种群初始化的时候，有些袋鼠非常幸运的降落到一个比较高的山峰，但是那不是珠穆朗玛峰，由于其它的适应度较高，生的 儿女特别多，为了保持人口总数的平衡，那些爬在珠穆朗玛峰半山腰的就不幸被杀了。）

（2）在遗传进化的后期，即算法接近收敛时，由于种群中个体适应度差异较小时，继续优化的潜能降低，可能获得某个局部最优解。

后记

如 果读者还能硬着头皮看到这里的话，那么恭喜你了，你已经走进遗传算法的殿堂。当然你现在还未能把每个房间都仔细观摩，更没来得及去端详那些油光亮瓦。事实 上，你也许还在纳闷，遗传算法能做什么，遗传算法的魔力何在？呵呵，不必着急，只管带着这些问题继续往下读，要记住一句话：把宝贝用到适当的地方去，就会 发挥其巨大作用。

经过这章的学习，大家将基本把握遗传算法的一般步 骤，而且能够利用遗传算法解决简单的实际问题。但是那距离高效而有实际应用价值的遗传算法还有一段距离，但是你的确已经跨过了那道最深的鸿沟，接下来的路 子将更加平坦而舒适，更加有趣而更具实际意义。好的，让我们带着满脑的疑问，与及美好的憧憬进入下一次的学习－－星际旅行中的人工智能(1)，让我们看看科幻影片里面的人工智能是如何代替人类操纵宇宙飞船飞出险境的！