3.4 交叉算子

在生物的自然进化过程中，两个同源染色体通过交配而重组，形成新的染色体，从而产生出新的个体或物种。交配重组是生物遗传和进化过程中的一个主要环节。模仿这个环节，在遗传算法中也使用交叉算子来产生新的个体。

遗传算法中的所谓交叉运算，是指对两个相互配对的染色体按某种方式相互交换其部分基因，从而形成两个新的个体。交叉运算是遗传算法区别于其他进化算法的重要特征，它在遗传算法中起着关键作用，是产生新个体的主要方法。

遗传算法中，在交叉运算之前还必须先对群体中的个体进行配对。目前常用的配对策略是随机配对，即将群体中的 M 个个体以随机的方式组成 $\lfloor M/2 \rfloor$ 对配对个体组，交叉操作是在这些配对个体组中的两个个体之间进行的。

交叉算子的设计和实现与所研究的问题密切相关，一般要求它既不要太多地破坏个体编码串中表示优良性状的优良模式，又要能够有效地产生出一些较好的新个体模式。另外，交叉算子的设计要和个体编码设计统一考虑。

交叉算子的设计包括以下两方面的内容：

(1) 如何确定交叉点的位置？

(2) 如何进行部分基因交换？

最常用的交叉算子是单点交叉算子。但单点交叉操作有一定的适用范围，故人们发展了其他一些交叉算子。下面介绍几种适合于二进制编码个体或浮点数编码个体的交叉算子。

3.4.1 单点交叉

单点交叉(One-point Crossover)[3,4] 又称为简单交叉，它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两个配对个体的部分染色体。单点交叉的具体运算过程已在第二章中作过介绍，此处不再赘述。

单点交叉的重要特点是：若邻接基因座之间的关系能提供较好的个体性状和较高的个体适应度的话，则这种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。

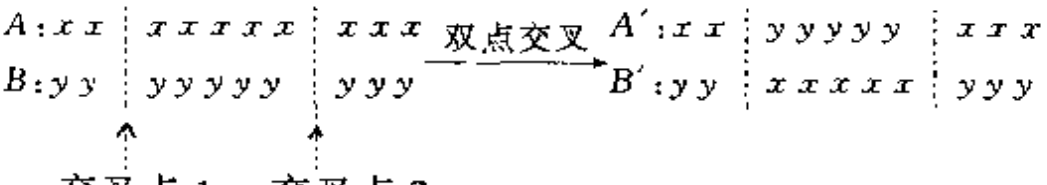
3.4.2 双点交叉与多点交叉

双点交叉(Two-point Crossover)[19，20] 是指在个体编码串中随机设置了二个交叉点，然后再进行部分基因交换。双点交叉的具体操作过程是：

(1) 在相互配对的两个个体编码串中随机设置两个交叉点。

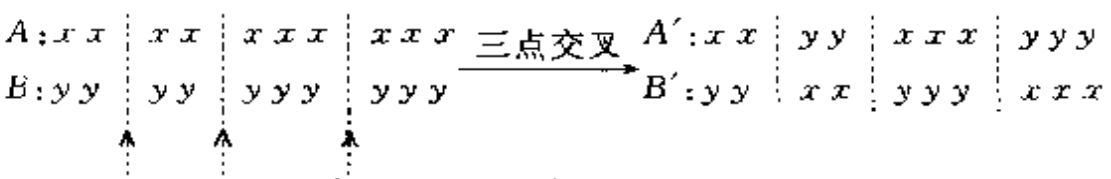
(2) 交换两个个体在所设定的两个交叉点之间的部分染色体。

例如，双点交叉操作的示例如下：

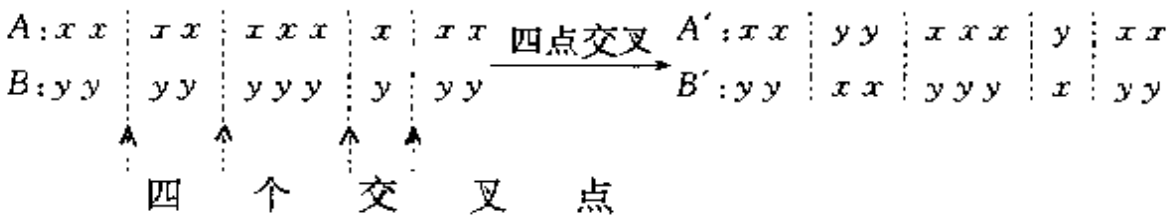


将单点交叉和双点交叉的概念加以推广，可得到多点交叉(multi-point crossover)[4] 的概念。即多点交叉是指在个体编码串中随机设置了多个交叉点，然后进行基因交换。多点交叉又称为广义交叉，其操作过程与单点交叉和双点交叉相类似。

如下所示为有三个交叉点时的交叉操作示例：



如下所示为有四个交叉点时的交叉操作示例：



需要说明的是，一般不太使用多点交叉算子，因为它有可能破坏一些好的模式。事实上，随着交叉点数的增多，个体的结构被破

坏的可能性也逐渐增大，这样就很难有效地保存较好的模式，从而影响遗传算法的性能。

3.4.3 均匀交叉

均匀交叉(Uniform Crossover)[21]是指两个配对个体的每一个基因座上的基因都以相同的交叉概率进行交换，从而形成两个新的个体。均匀交叉实际上可归属于多点交叉的范围，其具体运算可通过设置一屏蔽字来确定新个体的各个基因如何由哪一个父代个体来提供。均匀交叉的主要操作过程如下：

(1) 随机产生一个与个体编码串长度等长的屏蔽字 $W = w_1 w_2 \cdots w_i \cdots w_l$，其中 l 为个体编码串长度。

(2) 由下述规则从 A、B 两个父代个体中产生出两个新的子代个体 A'、B'：

● 若 $w_i = 0$，则 A' 在第 i 个基因座上的基因值继承 A 的对应基因值，B' 在第 i 个基因座上的基因值继承 B 的对应基因值；

● 若 $w_i = 1$，则 A' 在第 i 个基因座上的基因值继承 B 的对应基因值，B' 在第 i 个基因座上的基因值继承 A 的对应基因值。

均匀交叉操作的示例如下：

$$\begin{array}{l} A: x\,x\,x\,x\,x\,x\,x\,x\,x\,x \\ B: y\,y\,y\,y\,y\,y\,y\,y\,y\,y \end{array} \xrightarrow[W = 0101010101]{\text{均匀交叉}} \begin{array}{l} A': x\,y\,x\,y\,x\,y\,x\,y\,x\,y \\ B': y\,x\,y\,x\,y\,x\,y\,x\,y\,x \end{array}$$

3.4.4 算术交叉

算术交叉(Arithmetic Crossover)[22]是指由两个个体的线性组合而产生出两个新的个体。为了能够进行线性组合运算，算术交叉的操作对象一般是由浮点数编码所表示的个体。

假设在两个个体 X_A^t、X_B^t 之间进行算术交叉，则交叉运算后所产生出的两个新个体是：

$$\begin{cases} X_A^{t+1} = \alpha X_B^t + (1-\alpha) X_A^t \\ X_B^{t+1} = \alpha X_A^t + (1-\alpha) X_B^t \end{cases} \tag{3-14}$$

式中，α 为一参数，它可以是一个常数，此时所进行的交叉运算称为均匀算术交叉；它也可以是一个由进化代数所决定的变量，此时

所进行的交叉运算称为非均匀算术交叉。

算术交叉的主要操作过程是：

(1) 确定两个个体进行线性组合时的系数 α。

(2) 依据式(3-14) 生成两个新的个体。

3.5 变异算子

在生物的遗传和自然进化过程中，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，这样就会导致生物的某些基因发生某种变异，从而产生出新的染色体，表现出新的生物性状。虽然发生这种变异的可能性比较小，但它也是产生新物种的一个不可忽视的原因。模仿生物遗传和进化过程中的这个变异环节，在遗传算法中也引入了变异算子来产生出新的个体。

遗传算法中的所谓变异运算，是指将个体染色体编码串中的某些基因座上的基因值用该基因座的其他等位基因来替换，从而形成一个新的个体。例如，对于二进制编码的个体，其编码字符集为{0，1}，变异操作就是将个体在变异点上的基因值取反，即用 0 替换 1，或用 1 替换 0；对于浮点数编码的个体，若某一变异点处的基因值的取值范围为$[U_{\min}, U_{\max}]$，变异操作就是用该范围内的一个随机数去替换原基因值；对于符号编码的个体，若其编码字符集为{A, B, C, …}，变异操作就是用这个字符集中的一个随机指定的且与原基因值不相同的符号去替换变异点上的原有符号。

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，但它也是必不可少的一个运算步骤，因为它决定了遗传算法的局部搜索能力。交叉算子与变异算子的相互配合，共同完成对搜索空间的全局搜索和局部搜索，从而使得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。

在遗传算法中使用变异算子主要有以下两个目的：

(1) 改善遗传算法的局部搜索能力。遗传算法使用交叉算子已经从全局的角度出发找到了一些较好的个体编码结构，它们已接近或有助于接近问题的最优解。但仅使用交叉算子无法对搜索空间的细节进行局部搜索。这时若再使用变异算子来调整个体编码串中的部分基因值，就可以从局部的角度出发使个体更加逼近最优解，从而提高了遗传算法的局部搜索能力。

(2) 维持群体的多样性，防止出现早熟现象。变异算子用新的基因值替换原有基因值，从而可以改变个体编码串的结构，维持群体的多样性，这样就有利于防止出现早熟现象。

变异算子的设计包括如下两方面的内容：

- 如何确定变异点的位置？
- 如何进行基因值替换？

最简单的变异算子是基本位变异算子。为适应各种不同应用问题的求解需要，人们也开发出了其他一些变异算子。下面介绍其中较常用的几种变异操作方法，它们适合于二进制编码的个体和浮点数编码的个体。

3.5.1 基本位变异

基本位变异(Simple Mutation)操作是指对个体编码串中以变异概率 p_m 随机指定的某一位或某几位基因座上的基因值作变异运算[4]，其具体操作过程已在第二章中作过介绍，此处不再赘述。

基本位变异操作改变的只是个体编码串中的个别几个基因座上的基因值，并且变异发生的概率也比较小，所以其发挥的作用比较慢，作用的效果也不明显。

3.5.2 均匀变异

均匀变异(Uniform Mutation)[23]操作是指分别用符合某一范围内均匀分布的随机数，以某一较小的概率来替换个体编码串中各个基因座上的原有基因值。

均匀变异的具体操作过程是：

(1) 依次指定个体编码串中的每个基因座为变异点。

(2) 对每一个变异点，以变异概率 p_m 从对应基因的取值范围

内取一随机数来替代原有基因值。

假设有一个体为 $X = x_1x_2\cdots x_k\cdots x_l$，若 x_k 为变异点，其取值范围为$[U^k_{\min}, U^k_{\max}]$，在该点对个体 X 进行均匀变异操作后，可得到一个新的个体 $X = x_1\cdots x_2\cdots x'_k\cdots x_l$，其中变异点的新基因值是：

$$x'_k = U^k_{\min} + r \cdot (U^k_{\max} - U^k_{\min}) \tag{3-15}$$

式中，r 为[0，1]范围内符合均匀概率分布的一个随机数。

均匀变异操作特别适合应用于遗传算法的初期运行阶段，它使得搜索点可以在整个搜索空间内自由地移动，从而可以增加群体的多样性，使算法处理更多的模式。

3.5.3 边界变异

边界变异(Boundary Mutation)[22]操作是上述均匀变异操作的一个变形遗传算法。在进行边界变异操作时，随机地取基因座的二个对应边界基因值之一去替代原有基因值。

在进行由 $X = x_1x_2\cdots x_k\cdots x_l$ 向 $X' = x_1x_2\cdots x'_k\cdots x_l$ 的边界变异操作时，若变异点 x_k 处的基因值取值范围为$[U^k_{\min}, U^k_{\max}]$，则新的基因值 x'_k 由下式确定：

$$x'_k = \begin{cases} U^k_{\min}, & \text{if} \quad \text{random}(0,1) = 0 \\ U^k_{\max}, & \text{if} \quad \text{random}(0,1) = 1 \end{cases} \tag{3-16}$$

式中，random(0，1) 表示以均等的概率从 0、1 中任取其一。

当变量的取值范围特别宽，并且无其他约束条件时，边界变异会带来不好的作用。但它特别适用于最优点位于或接近于可行解的边界时的一类问题。

3.5.4 非均匀变异

均匀变异操作取某一范围内均匀分布的随机数来替换原有基因值，可使得个体在搜索空间内自由移动。但另一方面，它却不便于对某一重点区域进行局部搜索。为改进这个性能，我们不是取均匀分布的随机数去替换原有的基因值，而是对原有基因值作一随机扰动，以扰动后的结果作为变异后的新基因值。对每个基因座都

以相同的概率进行变异运算之后，相当于整个解向量在解空间中作了一个轻微的变动。这种变异操作方法就称为非均匀变异(Non-uniform Mutation)[23]。

非均匀变异的具体操作过程与均匀变异相类似，但它重点搜索原个体附近的微小区域。

在进行由 $X = x_1 x_2 \cdots x_k \cdots x_l$ 向 $X' = x_1 x_2 \cdots x'_k \cdots x_l$ 的非均匀变异操作时，若变异点 x_k 处的基因值取值范围为 $[U^k_{\min}, U^k_{\max}]$，则新的基因值 x'_k 由下式确定：

$$x'_k = \begin{cases} x_k + \Delta(t, U^k_{\max} - v_k), & \text{if} \quad \text{random}(0,1) = 0 \\ x_k - \Delta(t, v_k - U^k_{\min}), & \text{if} \quad \text{random}(0,1) = 1 \end{cases} \tag{3-17}$$

式中，$\Delta(t, y)$(y 代表 $U^k_{\max} - v_k$ 和 $v_k - U^k_{\min}$) 表示 $[0, y]$ 范围内符合非均匀分布的一个随机数，要求随着进化代数 t 的增加，$\Delta(t, y)$ 接近于 0 的概率也逐渐增加。例如，$\Delta(t, y)$ 可按下式定义：

$$\Delta(t, y) = y \cdot (1 - r^{(1-t/T)^b}) \tag{3-18}$$

式中，r 为[0，1]范围内符合均匀概率分布的一个随机数，T 是最大进化代数，b 是一个系统参数，它决定了随机扰动对进化代数 t 的依赖程度。

由式(3-17) 和式(3-18) 可知，非均匀变异可使得遗传算法在其初始运行阶段(t 较小时) 进行均匀随机搜索，而在其后期运行阶段(t 较接近于 T 时) 进行局部搜索，所以它产生的新基因值比均匀变异所产生的基因值更搜近于原有基因值。故随着遗传算法的运行，非均匀变异就使得最优解的搜索过程更加集中在某一最有希望的重点区域中。

3.5.5 高斯变异

高斯变异(Gaussian Mutation)[22] 是改进遗传算法对重点搜索区域的局部搜索性能的另一种变异操作方法。所谓高斯变异操作是指进行变异操作时，用符合均值为 μ、方差为 σ^2 的正态分布的一个随机数来替换原有基因值。

由正态分布的特性可知，高斯变异也是重点搜索原个体附近的某个局部区域。高斯变异的具体操作过程与均匀变异相类似。

具体实现高斯变异时，符合正态分布的随机数 Q 可由一些符合均匀分布的随机数利用公式来近似产生。假定有 12 个在[0, 1]范围内均匀分布的随机数 $r_i(i = 1, 2, \cdots, 12)$，则符合 $N(\mu, \sigma^2)$ 正态分布的一个随机数 Q 可由下式求得：

$$Q = \mu + \sigma \cdot \left(\sum_{i=1}^{12} r_i - 6 \right) \tag{3-19}$$

在进行由 $X = x_1 x_2 \cdots x_k \cdots x_l$ 向 $X' = x_1 x_2 \cdots x'_k \cdots x_l$ 的高斯变异操作时，若变异点 x_k 处的基因值取值范围为 $[U^k_{\min}, U^k_{\max}]$，并假设：

$$\mu = \frac{U^k_{\min} + U^k_{\max}}{2} \tag{3-20}$$

$$\sigma = \frac{U^k_{\max} - U^k_{\min}}{6} \tag{3-21}$$

则新的基因值 x'_k 可由下式确定：

$$x'_k = \frac{U^k_{\min} + U^k_{\max}}{2} + \frac{U^k_{\max} - U^k_{\min}}{6} \cdot \left(\sum_{i=1}^{12} r_i - 6 \right) \tag{3-22}$$

3.6 遗传算法的运行参数

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、群体大小 M、交叉概率 p_c、变异概率 p_m、终止代数 T、代沟 G 等。这些参数对遗传算法的运行性能影响较大，需认真选取。

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

(2) 群体大小 M。群体大小 M 表示群体中所含个体的数量。

当 M 取值较小时，可提高遗传算法的运算速度，但却降低了群体的多样性，有可能会引起遗传算法的早熟现象；而当 M 取值较大时，又会使得遗传算法的运行效率降低。一般建议的取值范围是 20 ~ 100。

(3) 交叉概率 p_c。交叉操作是遗传算法中产生新个体的主要方法，所以交叉概率一般应取较大值。但若取值过大的话，它又会破坏群体中的优良模式，对进化运算反而产生不利影响；若取值过小的话，产生新个体的速度又较慢。一般建议的取值范围是 0.4 ~ 0.99。另外，也可使用自适应的思想来确定交叉概率 p_c，如 Davis[24] 提出，随着遗传算法在线性能的提高，可以增大交叉概率 p_c 的取值。

(4) 变异概率 p_m。若变异值率 p_m 取值较大的话，虽然能够产生出较多的新个体，但也有可能破坏掉很多较好的模式，使得遗传算法的性能近似于随机搜索算法的性能；若变异概率 p_m 取值太小的话，则变异操作产生新个体的能力和抑制早熟现象的能力就会较差。一般建议的取值范围是 0.0001 ~ 0.1。另外，也可使用自适应的思想来确定变异概率 p_m，如 Davis[24] 提出，随着遗传算法在线性能的下降，可以减小变异概率 p_m 的取值；而在 Whitley[25] 提出的一种自适应变异策略中，p_m 与其上一代群体间的海明距离成反比，其结果显示出这种方法能够有效地维持群体的多样性。

(5) 终止代数 T。终止代数 T 是表示遗传算法运行结束条件的一个参数，它表示遗传算法运行到指定的进化代数之后就停止运行，并将当前群体中的最佳个体作为所求问题的最优解输出。一般建议的取值范围是 100 ~ 1000。

至于遗传算法的终止条件，还可以利用某种判定准则，当判定出群体已经进化成熟且不再有进化趋势时就可终止算法的运行过程。常用的判定准则有下面两种：

- 连续几代个体平均适应度的差异小于某一个极小的阈值；
- 群体中所有个体适应度的方差小于某一个极小的阈值。

(6) 代沟 G。代沟 G 是表示各代群体之间个体重叠程度的一

个参数，它表示每一代群体中被替换掉的个体在全部个体中所占的百分率，即每一代群体中有 $(M \times G)$ 个个体被替换掉。例如，$G = 1.0$ 表示群体中的全部个体都是新产生的，这也是最常见的一种情况；$G = 0.7$ 则表示 70% 的个体是新产生的，而随机保留了上一代群体中 30% 的个体。

3.7 约束条件的处理方法

实际应用中的优化问题一般都含有一定的约束条件，它们的描述形式各种各样。在遗传算法的应用中，必须对这些约束条件进行处理，而目前还未找到一种能够处理各种约束条件的一般化方法。所以对约束条件进行处理时，只能是针对具体应用问题及约束条件的特征，再考虑遗传算法中遗传算子的运行能力，选用不同的处理方法。

在构造遗传算法时，处理约束条件的常用方法主要有如下三种[26]：搜索空间限定法、可行解变换法、罚函数法。

3.7.1 搜索空间限定法

这种处理方法的基本思想是对遗传算法的搜索空间的大小加以限制，使得搜索空间中表示一个个体的点与解空间中表示一个可行解的点有一一对应的关系。此时的搜索空间与解空间的对应关系如图 3-3 所示。

对一些比较简单的约束条件(如 $a \leqslant x \leqslant b$ 之类)，在个体染色体的编码方法上着手，就能够达到这种搜索空间与解空间之间的一一对应的要求。用这种处理方法能够在遗传算法中设置最小的搜索空间，所以它能够提高遗传算法的搜索效率。但需要注意的是，除了在编码方法上想办法之外，也必须保证经过交叉、变异等遗传算子作用之后所产生出的新个体在解空间中也要有确定的对应解，而不会产生无效解。

这种处理约束条件的方法可由下面两种方法之一来实现：

方法一：用编码方法来保证总是能够产生出在解空间中有对

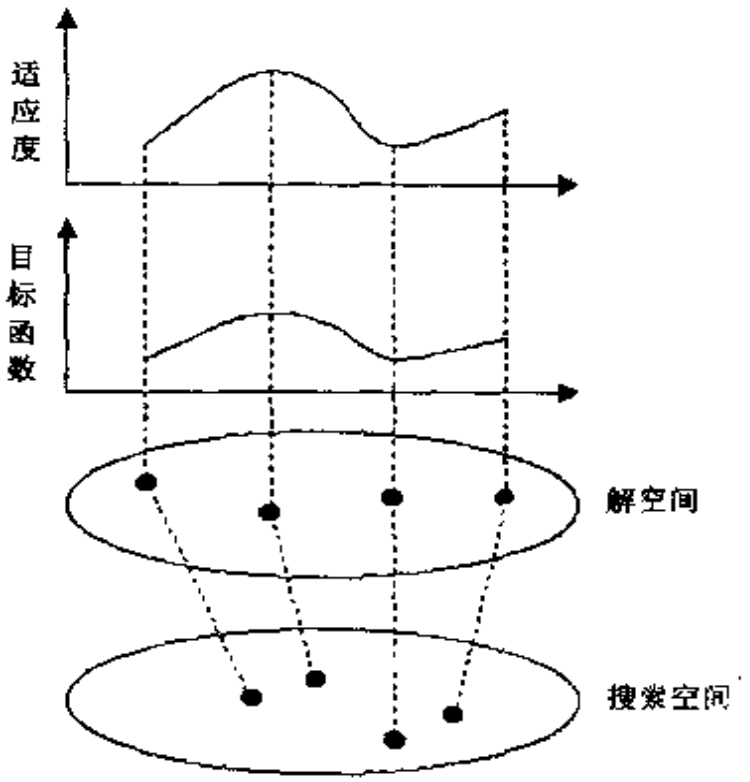


图 3-3　搜索空间与解空间之间的一对一的对应关系

应可行解的染色体。

这个实现方法要求我们设计出一种比较好的个体编码方案。例如，在处理 $a \leqslant x \leqslant b$ 这样的约束条件时，若使用二进制编码串来表示个体，我们将区间[a, b] 划分为(2^l-1) 个等分(其中 l 为个体编码串长度)，δ 为每个等分的长度，并且使编码时的对应关系如下：

$$
\begin{aligned}
&00000000\cdots00000000 = 0 && \longrightarrow a\\
&00000000\cdots00000001 = 1 && \longrightarrow a+\delta\\
&\quad\vdots\quad\vdots\quad\vdots && \quad\vdots\quad\vdots\\
&11111111\cdots11111111 = 2^l-1 && \longrightarrow b
\end{aligned}
$$

则介于 00000000…00000000 和 11111111…11111111 之间的任何编码都会满足上述这个约束条件。

方法二：用程序来保证直到产生出在解空间中有对应可行解的染色体之前，一直进行交叉运算和变异运算。

虽然这个实现方法对编码方法的要求不高，但它有可能需要反复地进行交叉运算和变异运算才能产生出一个满足约束条件的可行解，这样就有可能会降低遗传算法的运行效率。

3.7.2 可行解变换法

这种处理方法的基本思想是：在由个体基因型到个体表现型的变换中，增加使其满足约束条件的处理过程。即寻找出一种个体基因型和个体表现型之间的多对一的变换关系，使进化过程中所产生的个体总能够通过这个变换而转化成解空间中满足约束条件的一个可行解。图 3-4 所示为该方法的示意图。

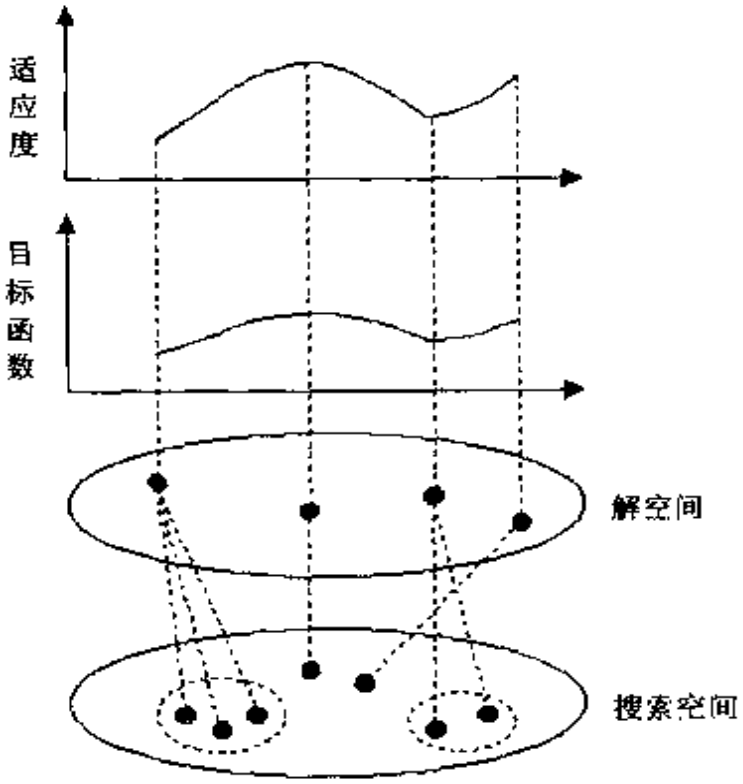


图 3-4 搜索空间与解空间之间的多对一的对应关系

这种处理方法虽然对个体的编码方法、交叉运算、变异运算等没有附加的要求，但它却是以扩大搜索空间为代价的，所以一般会使得遗传算法的运行效率有所下降。

例如，在处理 $x^2 + y^2 \leqslant 1$，并且 $x \in [-1,1]$，$y \in [-1,1]$ 这个约束条件时，如图 3-5 所示，可以把单位圆之外的点都通过该点到圆心的连线而变换到圆周之内。

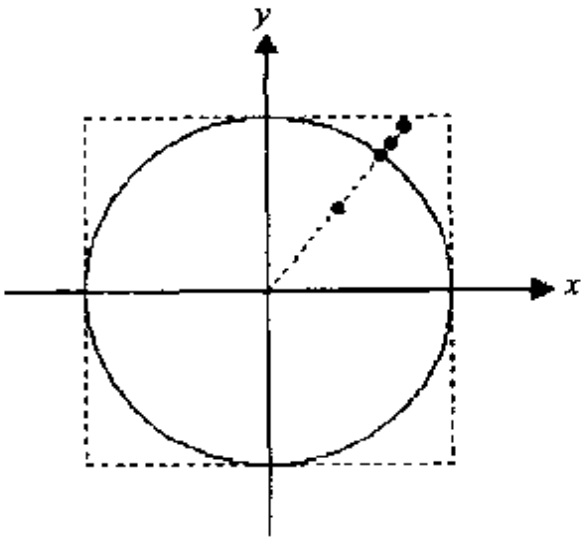


图 3-5　单位圆的外点向内点的变换处理

3.7.3　罚函数法

这种处理方法的基本思想是：对在解空间中无对应可行解的个体，计算其适应度时，处以一个罚函数，从而降低该个体适应度，使该个体被遗传到下一代群体中的机会减少。即用下式来对个体的适应度进行调整：

$$F'(X)=\begin{cases}F(X) & (X\text{ 满足约束条件时})\\ F(X)-P(X) & (X\text{ 不满足约束条件时})\end{cases} \tag{3-23}$$

式中，$F(X)$ 为原适应度，$F'(X)$ 为考虑了罚函数之后的新适应度，$P(X)$ 为罚函数。

例如，在处理 $x^2+y^2\leqslant 1$，并且 $x\in[-1,1]$，$y\in[-1,1]$ 这个约束条件时，融合罚函数的思想，可由下面的计算公式来计算个体的适应度：

$$F(x,y)=f(x,y)-\alpha\cdot\max\{0,x^2+y^2-1\}$$

式中，$\alpha>0$ 就是确定罚函数作用强度的一个系数。

如何确定合理的罚函数是这种处理方法的难点之所在，因为这时既要考虑如何度量解对约束条件不满足的程度，又要考虑遗传算法在计算效率上的要求。罚函数的强度太小的话，部分个体仍有可能破坏约束条件，所以保证不了遗传运算所得到的个体一定是满足约束条件的一个可行解；罚函数的强度太大的话，又有可能使个体的适应度差异不大，降低了个体之间的竞争力，从而影响遗

传算法的运行效率。

罚函数法的一种极端处理情况是，对在解空间中无对应可行解的个体，将其适应度降低为 0，从而使得该个体绝对不会遗传到下一代群体中。即：

$$F'(X) = \begin{cases} F(X) \\ 0 \end{cases} \tag{3-24}$$

例如，对于约束条件 $x^2 + y^2 \leqslant 1$，融合这种思想后，可以这样来处理个体的适应度：

$$F'(x,y) = \begin{cases} F(x,y) & \text{if} \quad x^2 + y^2 \leqslant 1 \\ 0 & \text{if} \quad x^2 + y^2 > 1 \end{cases}$$

这种适应度为 0 的染色体称为致死基因(Lethal Gene)。使用致死基因的方法处理约束条件时，虽然绝对不会产生破坏约束条件的个体，但当问题的约束条件比较严格时，由交叉算子或变异算子在搜索空间中生成新个体的能力就比较差，即使能够生成一些新的个体，群体的多样性也会有较大程度的降低，从而对遗传算法的运行带来不利的影响。所以这种方法应该谨慎使用。

3.8 遗传算法工具箱

基于对遗传算法基本实现技术的分析和归纳，我们在 Windows 环境下开发了一个遗传算法工具箱，它构成了一个研究遗传算法的简易实验平台[27]。

这个遗传算法工具箱的主要功能有：

- 能够方便地构造使用不同遗传算子的各种遗传算法；
- 能够方便地设置遗传算法的不同的运行参数；
- 能够选择不同的进化结果输出界面和输出形式；
- 能够方便地描绘群体进化过程中个体适应度的变化图；
- 能够方便地描绘群体进化过程中个体的分布图；
- 能够简单地对不同的遗传算法进行性能对比分析。