第3章

进化和群智能优化

摘要 本章介绍了基础的进化算法，包括经典的遗传算法(Genetic Algorithm，GA)、实数编码遗传算法、进化策略(Evolution Strategies，ES)、遗传规划(Genetic Programming，GP)、蚁群优化(Ant Colony Optimization，ACO)算法、差分进化算法和粒子群优化(Particle Swarm Optimization，PSO)算法。此外，还描述了结合进化搜索和局部搜索的模因算法，以及使用概率模型生成后代解的分布估计算法(Estimation of Distribution Algorithm，EDA)。最后，介绍了求解多目标和高维多目标优化问题的基本方法。

3.1 引言

元启发式算法一般可以分为两大类：一类是在抽象层面模拟自然进化过程的进化算法，另一类是模拟群居动物(如鸟群和蚁群)群体行为的群智能优化算法。这两类元启发式算法都是基于种群的随机搜索方法，原则上他们都能用于求解各种类型的白盒和黑盒优化问题。但需要注意的是，进化算法和群智能优化算法也可以作为设计全局自适应系统的通用工具，或用于模拟和理解人工生命等自然智能。

作为人工智能的一个研究领域，进化算法可以追溯到20世纪60年代，当时美国的Lawrence J. Fogel提出了进化编程[61]，德国Ingo Rechenberg和Hans-Paul Schwefeln提出了进化策略[62]，美国John Hollandn提出了遗传算法并随着David Goldberg出版的一本书[63]而为人们熟识起来。最新的两类进化算法分别是由John Koza在20世纪80年代末提出的遗传规划[64]，以及由Storn和Price在20世纪90年代末提出的差分进化算法[65]。多年来，不同种类的进化算法已经融合在一起[66]。

遗传算法、进化策略和遗传规划的一般框架非常相似，图3.1给出了进化算法的一般框架。给定求解问题的表示方法(编码机制)，所有进化算法都从初始化父代种群开始。从优化的角度来看，用于搜索的初始种群由一组随机初始化的起点组成。随后，对父代种群进行交叉和变异产生新的候选解，称为子代种群。配偶选择是指用于完成交叉的父代选择。所有子代个体在执行环境选择之前进行适应度评估。需要注意的是，环境选择可能只基于子

代种群(非精英)或者基于父代和子代种群(精英)。所选择的解将成为下一代父代种群。此过程重复执行,直到满足终止条件。

群体优化算法是另一类元启发式搜索方法,其灵感主要来自于群居动物的集体行为,如鸟群和蚁群。最流行的群体优化算法是 PSO 算法和蚁群优化算法。与进化算法不同,群体优化算法通常没有交叉或选择操作。然而,值得注意的是,PSO 算法和蚁群优化算法的工作机制是非常不同的。蚁群的集体行为是蚂蚁之间基于信息素的局部通信实现的,而在 PSO 算法中,粒子会向性能更好的粒子学习,从而使种群的收敛性能达到一个平衡状态。

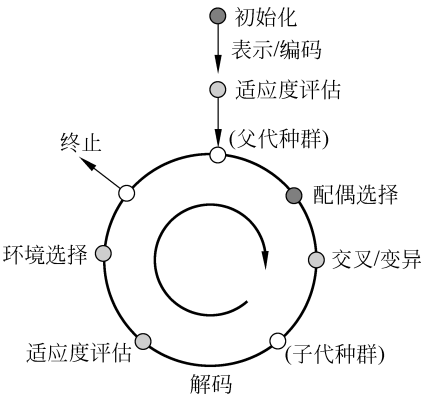


图 3.1　进化算法的一般框架

3.2　遗传算法

3.2.1　定义

GA 旨在模拟自然进化的过程,对每个生物组织的基因型-表现型映射关系进行建模、通过个体之间的有性繁殖实现染色体的交叉、在个体层面进行遗传突变以及在群体层面进行自然选择。

遗传算法中的大多数术语都来源于生物学。在遗传算法中,种群由一组个体构成,种群中个体的数量称为种群大小。一个个体由一条或多条染色体组成,每条染色体由基因组成并具有一个特定的等位基因。个体的一个特征由一个或多个基因编码,例如鸟的羽毛颜色;然而,也有可能多个基因编码一个特征(称为多基因性),或者一个基因影响多个特征(多效性)。

每个个体都有一个适应度,在生物系统中,适应度对应个体的生存和繁殖能力。在优化中,适应度代表解的质量,用于确定适应度的数学描述称为适应度函数。

染色体的完整集合称为个体的基因型,而特征的完整集合称为表现型。图 3.2(a)举例说明了一个由 8 个基因和 4 个特征组成的基因型-表现型映射关系,图 3.2(b)和图 3.2(c)分别是多基因性和多效性的例子。

3.2.2　表示

传统的遗传算法中一般采用二进制表示来模拟生物系统中的 DNA 序列。例如,若使用一个 4 位二进制字符串来编码一个整数决策变量,那么一个 12 位的字符串'1 0 1 1 0 1 1 0 1 1 1 0'表示的是 3 个整数,从左到右,'1 0 1 1'表示整数 11,'0 1 1 0'表示整数 6,'1 1 1 0'表

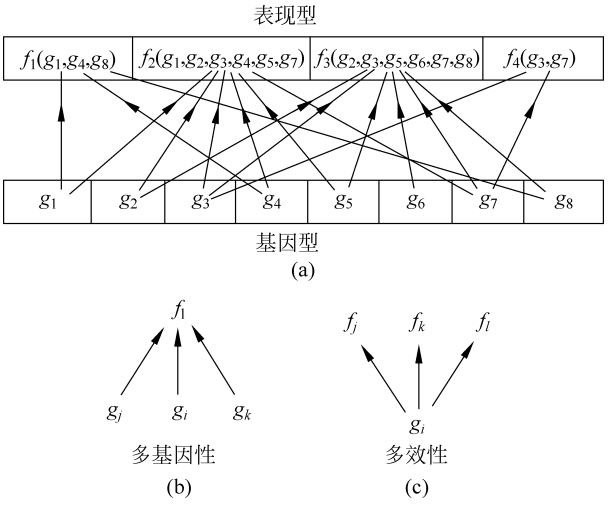


图 3.2 基因型-表现型的映射关系

示整数 14。

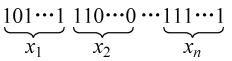


图 3.3 对 n 个决策变量进行编码的二进制字符串

图 3.3 所示的二进制串表示 n 个决策变量，每个变量由 l 位二进制编码。因此，若编码的决策变量是整数，则二进制字符串可以通过以下形式来进行解码：

$$x_i = \sum_{j=0}^{l-1} s_j 2^j, \quad s_j = \{0,1\} \tag{3.1}$$

通常，二进制编码的遗传算法可用于整数优化问题和组合优化问题。在以往，它们也被广泛用于连续优化问题，其中实值决策变量使用二进制字符串进行编码。在这种情况下，若给定解码的实值决策变量范围：

$$x_i = a_i + (b_i - a_i)\frac{1}{2^l - 1}\sum_{j=0}^{l-1} s_j 2^j, \quad s_j = \{0,1\} \tag{3.2}$$

则一个 l 位二进制串可以解码为一个实数。其中，$x_i \in [a_i, b_i]$，a_i 和 b_i 分别是决策变量 x_i 的上界和下界。

需要注意的是，若实数 $a_i \leqslant x_i \leqslant b_i$ 被编码为 l 位，那么编码范围需要离散化为 $2^l - 1$ 等份并引入量化误差。也就是编码的决策变量可能无法达到目标函数的最优值。打个比方，如果在[−1,1]区间的一个决策变量使用 3 位字符串进行编码，编码后的 8 个数字为 $\{-1,-5/7,-3/7,-1/7,1/7,3/7,5/7,1\}$。如果目标函数的最优值在 $x=0$ 位置，那么它永远无法通过使用 3 位二进制字符串找到最优值。很明显，l 越大，量化误差越小。但是，如果决策变量的数量很大，那么使用较大的 l 会导致巨大的搜索空间。因此，一般来说，二进制编码的遗传算法并不适合连续问题的优化。解决搜索准确度和效率平衡问题的一种方法是使用可变长编码方法，其中编码长度可以变化，通常随着进化过程而增加。

二进制遗传算法的另一个问题是所谓的海明悬崖。通过海明悬崖，编码十进制数字每增加1，二进制数字就会发生最大的变化。例如，一个4位二进制字符串'0 1 1 1'表示7，而'1 0 0 0'表示8。因此，要从7更改到8，所有4位都必须要更改（最大更改）。为了解决这一问题，可采用格雷编码来代替二进制编码。

3.2.3　交叉和变异

变化的主要驱动力来自于交叉操作算子。交叉操作模拟生物体的有性繁殖，其中选用两个父代个体（称为配偶选择）来产生新的候选解，称为子代个体。在交叉过程中，随机选择一个或多个交叉点，然后交换交叉点（二进制字符串片段）之间双亲的遗传物质。图3.4列举了3种不同的交叉情况，即单点交叉、两点交叉和均匀交叉。从理论上讲，对于一个 l 位长度的字符串，可以有一个点、两个点……（$l-1$）个点交叉，并且交叉点是随机选择的。一种更一般的交叉方法称为均匀交叉，其中和染色体长度相同的二进制掩码是随机产生的。如果掩码字符位上是'1'，则进行位的交换，否则如果掩码字符位上为'0'，则不进行交换。

考虑到交叉是传统遗传算法中的主要遗传算子，因此交叉概率通常设置较高。此外，还可以基于多个父代个体进行交叉。

随后对交叉所产生的子代个体进行变异。对于使用二进制编码或格雷编码的遗传算法，变异只是简单地将'1'改为'0'，或将'0'改为'1'，这也称为翻转。对点的变异概率通常设置得较低，一般可以通过 $1/l$ 计算得到，其中 l 是染色体的总长度。

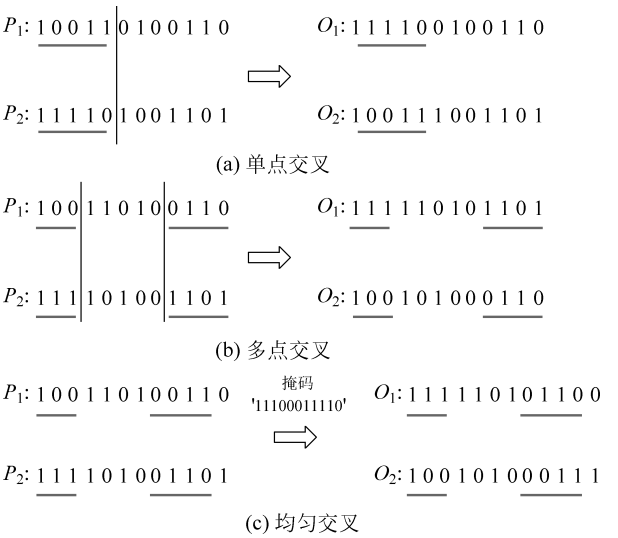


图3.4　交叉图示

3.2.4 环境选择

遗传算法中的环境选择模仿自然进化中达尔文的“适者生存”理论。通过环境选择，从子代种群中选择下一代父代种群。目前已提出了很多环境选择方法，在遗传算法中大多通过概率来选择。广泛使用的环境选择策略包括适应度比例选择（也称为轮盘赌选择）、等级比例选择和锦标赛选择。

在适应度比例选择中，按照以下概率来选择子代个体作为下一代的父代个体：

$$p(i)=\frac{f_i}{\sum_{i=1}^{N} f_i} \tag{3.3}$$

其中，f_i 为父代种群中第 i 个个体的适应度，N 为种群大小。需要注意的是，对于最大化问题，$f(i)$可以表示第 i 个个体的目标值。然而，对于最小化问题，需要将目标值转换成一个适应度，这样目标值越小，个体的适应度也就越小。这个目标值的转换可以通过以下方法来实现：

$$f'(i)=\frac{1}{f(i)+\varepsilon} \tag{3.4}$$

其中，$\varepsilon>0$ 表示一个很小的常数，用于避免除数为零。

在适应度比例选择中，如果个体之间的适应度相差很大，那么弱势个体将很难被选择到，从而导致选择压力过大。为了解决这个问题，可以使用排序比例选择方法。在这个方法中，子代种群中的个体从差到好进行排序，也就是说，最差的个体排序为 1，最好的个体排序为 N。然后使用以下选择比例实现父代个体的选择：

$$p(i)=\frac{r_i}{\sum_{i=1}^{N} r_i} \tag{3.5}$$

其中，$1\leqslant r(i)\leqslant N$ 表示个体 i 的排序。因此，适应度远小于其他个体的个体将有更大的机会被选中。需要指出的是，排序选择并不总是会降低选择压力。例如，如果种群中个体的适应度非常相似，那么将会导致排序比例选择产生很大的选择压力。

锦标赛选择是从子代个体中随机挑选 $2\leqslant k<N$ 个个体，然后从 k 个个体中选择最好的个体作为下一代父代个体，其中 k 称为锦标赛规模，重复以上操作直至选择够 N 个父代个体。当 $k=2$ 时，锦标赛选择称为二进制锦标赛。通常情况下，k 比种群规模 N 小得多。

然而，即使最优个体比子代种群中的所有个体都优秀，也没有上述哪一种选择策略能够将其保留在父代种群中，这些选择策略统称为非精英策略。因此，可以引进一种精英策略用于将最优个体的选择和概率选择方法相结合。

同样需要注意的是，在标准的遗传算法中，每一代中所有父代个体都会被子代个体替换

掉，这被称为一般的进化算法。相反，当只允许产生的子代个体来替换最差的父代个体时，称为稳态进化算法。

3.3　实数编码的遗传算法

尽管传统的二进制遗传算法可以用于整数优化、组合优化及连续优化，但当决策变量数量很多并且每个决策变量用一个很长的二进制字符串表示时，它的优化效率就会很低。若一个优化问题有 100 个决策变量，每个决策变量用 20 位二进制串表示，那么染色体的长度就是 2000，这就会形成一个很大的搜索空间，导致搜索效率很低。

3.3.1　实值表示

相比于二进制编码，实数编码更加直接。对于一个 n 维问题，染色体长度就是 n，而且不需要上下界的编码，如图 3.5 所示。

x_1	x_2	…	x_n

图 3.5　决策变量的实数编码

因此，二进制编码遗传算法的 n 点交叉或均匀交叉不能直接用于实数编码遗传算法，必须设计新的不同操作。目前，已经提出了很多的交叉算子，大致可以分为以种群为中心（均值中心）或者以父代为中心，以种群为中心表示交叉后产生的子代均值分布在父代种群的中心周围，而以父代为中心主要表示所产生的子代集中在父代个体周围。在实数编码遗传算法的各种交叉算子中，混合交叉（BLX-α）和模拟二进制交叉（Simulated Binary Crossover，SBX）的应用最为广泛。

3.3.2　混合交叉

以下为混合交叉（BLX-α）的描述。

首先从 t 代父代种群中选择两个父代个体：$\boldsymbol{x}_1(t)$ 和 $\boldsymbol{x}_2(t)$，其中

$$\boldsymbol{x}_1(t)=\{x_{11}(t),x_{21}(t),\cdots,x_{i1}(t),\cdots,x_{n1}(t)\},\quad i=1,2,\cdots,n$$

n 是决策变量的数量。

然后根据如下操作生成两个子代个体 $\boldsymbol{x}_1(t+1)$ 和 $\boldsymbol{x}_2(t+1)$。

对于 for $i=1$ to n，执行

(1) $d_i=|x_{i1}(t)-x_{i2}(t)|$；

(2) 从下列区间中随机选择一个均匀分布的实数 u_1

$$[\min\{x_{i1}(t),x_{i2}(t)\}-\alpha d_i,\max\{x_{i1}(t),x_{i2}(t)\}+\alpha d_i]$$

(3) $x_{i1}(t+1)=u_1$（第一个子代个体）；

(4) 从下列区间中随机选择一个均匀分布的实数 u_2

$$[\min\{x_{i1}(t),x_{i2}(t)\}-\alpha d_i,\max\{x_{i1}(t),x_{i2}(t)\}+\alpha d_i]$$

(5) $x_{i2}(t+1)=u_2$(第二个子代个体)。

从上面的过程中可以看到,混合交叉是两个父代个体之间的随机内插或者外插操作。

3.3.3 模拟二进制交叉和多项式变异

SBX 是实数编码遗传算法中另一种更流行的交叉算子。SBX 思想来源于二进制交叉过程中,即两个父代个体的平均值和两个子代个体的平均值是相同的。例如,给定两个二进制编码的父代个体,P_1:1001,P_2:1110,解码后的数分别是 9 和 11,其平均值为 10.5。如果在第二位(从左开始)和第三位之间进行单点交叉,那么子代个体就分别为 O_1:1000、O_2:1101,解码后其值分别为 8 和 13,可得其平均数也为 10.5。

因此,对于实数编码的遗传算法,SBX 的目标是二进制交叉的动态行为,使得子代的均值和父代的均值相同。为了实现这个想法,定义一个传播因子

$$\beta=\frac{|O_1-O_2|}{|P_1-P_2|} \tag{3.6}$$

其中,P_1、P_2、O_1、O_2 分别是两个父代个体和两个子代。可以看出,当父代个体之间的差异和子代个体之间的差异相同时,传播因子为 1。当 $\beta<1$ 时,子代个体之间的差异小于父代个体之间的差异,从而导致了收缩效应。反过来,当 $\beta>1$ 时,意味着两个子代个体之间的差异大于两个父代个体之间的差异,从而引发扩张效应。通常,产生的子代个体会比较接近于父代个体。给定两个父代个体,SBX 通过以下步骤产生两个子代。

(1) 在产生一个 0～1 的随机数 u。

(2) 通过以下公式计算传播因子

$$\beta(u)=\begin{cases}(2u)^{\frac{1}{\eta_c+1}}, & u\leqslant 0.5\\ \dfrac{1}{2(1-u)^{\frac{1}{\eta_c+1}}}, & u>0\end{cases} \tag{3.7}$$

其中,$\eta_c>1$ 称为分布因子,由用户自定义所得。

(3) 给定两个父代个体 $x_{i1}(t)$和 $x_{i2}(t)$,$i=1,2,\cdots,n$ 表示第 t 代 n 维决策变量的第 i 个元素,通过以下公式生成子代个体:

$$x_{i1}(t+1)=0.5[(1-\beta(u))x_{i1}(t)+(1+\beta(u))x_{i2}(t)] \tag{3.8}$$

$$x_{i2}(t+1)=0.5[(1+\beta(u))x_{i1}(t)+(1-\beta(u))x_{i2}(t)] \tag{3.9}$$

图 3.6 给出了在给定父代和分布因子的情况下子代个体分布的概率密度函数。可以看出,密度函数依赖于父代个体的位置以及分布因子(η_c)。分布因子越大,搜索的开采能力越强。

与 SBX 相似,实数编码的遗传算法同样也需要变异算子,称为多项式变异。假设 $x_i\in$

$[x_i^L, x_i^U]$是实数编码遗传算法的一个决策变量，其经过变异后生成 x_i'：

$$x_i' = \begin{cases} x_i + \delta_L (x_i - x_i^L), & u \leqslant 0.5 \\ x_i + \delta_R (x_i^U - x_i), & u > 0.5 \end{cases} \tag{3.10}$$

其中，u 是[0,1]区间的随机数，δ_L 和 δ_R 表示两个参数，其定义为：

$$\delta_L = (2u)^{\frac{1}{1+\eta_m}} - 1, \quad u \leqslant 0.5 \tag{3.11}$$

$$\delta_R = 1 - (2(1-u))^{\frac{1}{1+\eta_m}} - 1, \quad u > 0.5 \tag{3.12}$$

其中，η_m 表示变异操作的分布因子。

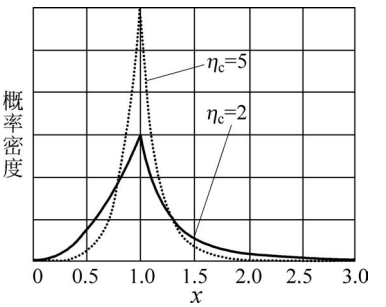


图 3.6　给定父代个体 $x=1$ 以及 $\eta_c=2$ 和 $\eta_c=5$ 时后代个体的分布密度函数

SBX 和多项式变异应该是实数编码遗传算法中应用最广泛的交叉和变异算子。

3.4　进化策略

ES 最初是针对连续优化问题而提出来的一类进化算法，它最早的版本是在 20 世纪 60 年代提出的 1+1-ES 算法。该算法与随机搜索非常相似，不同的是在该算法中提出了一种变异强度自适应的规则，称为“1/5 成功规则”。显然，通过使用参数和决策变量共同演化的参数自适应方法是进化策略和遗传算法的主要不同点。其他主要的不同点包括进化策略的子代种群规模通常要大于父代种群的规模，而且高斯变异是进化策略的主要驱动力，而不是遗传算法中的交叉操作。

3.4.1　(1+1)-ES

如其所命名，(1+1)-ES 中一个父代个体产生一个子代个体，因此其并不是真正的基于种群的搜索算法，但其包含了最重要的参数自适应思想。对于最小化实数函数 $f(\boldsymbol{x})$的优化问题，$\boldsymbol{x} \in \mathbf{R}^n$，(1+1)-ES 算法的主要步骤描述如下。

(1) 设置初始代数 $t=0$，初始化步长为 $\sigma(t)$，随机产生一个初始解

$$\boldsymbol{x}(t) = (x_1(t), x_2(t), \cdots, x_n(t))$$

(2) 对于每个 $i=1,2,\cdots,n$，从正态分布(高斯分布)中产生 z_i

$$x_i(t+1) = x_i(t) + z_i, \quad z_i \sim N(0, \sigma(t+1)) \quad \text{// 变异} \tag{3.13}$$

(3) 若 $f(\boldsymbol{x}(t)) < f(\boldsymbol{x}(t+1))$，则 $\boldsymbol{x}(t+1) = \boldsymbol{x}(t)$　//变异不成功

(4) $t \leftarrow t+1$。

(5) 根据 1/5 成功规则自适应步长 $\sigma(t)$。

(6) 若不满足终止条件，那么返回步骤(2)。

1/5 成功规则在每 k 代后调整步长 $\sigma(t)$($k\geqslant 5$ 由用户自行定义)。

$$\sigma(t+1)=\begin{cases}\sigma(t)/c, & p_s>1/5 \quad //\sigma \text{ 增大};\\ \sigma(t)\cdot c, & p_s<1/5 \quad //\sigma \text{ 减小};\\ \sigma(t), & p_s=1/5\end{cases} \tag{3.14}$$

其中,p_s 是过去 k 代中成功变异的比例,$0.8\leqslant c\leqslant 1$。也就是说,若在过去 k 代中成功变异的比例大于 1/5,则增大步长,否则减小步长。

3.4.2 基于全局步长的进化策略

基于种群的进化策略同样也有许多变种,其主要区别在于步长大小的数量、步长自适应的方式及其选择策略。具有全局步长的进化策略包含两个染色体:一个包含 n 个实值决策变量,另一个是步长大小 σ。换句话说,n 个决策变量使用相同的步长进行变异。全局步长和决策变量通过下列形式进行变异:

$$\sigma(t+1)=\sigma(t)\exp(\tau N(0,1)) \tag{3.15}$$

$$x_i(t+1)=x_i(t)+\sigma(t+1)N(0,1),\quad i=1,2,\cdots,n \tag{3.16}$$

其中,$N(0,1)$表示满足正态分布的随机数,$\tau\sim 1/\sqrt{2n}$ 是常数,n 表示决策变量的数量。

从上述公式中可以观察到,像决策变量一样,全局步长 σ 也会进化。需要注意的是,变异顺序是很重要的,即步长大小必须在决策变量变异之前进行变异。

3.4.3 基于个体步长大小的进化策略

显然,对所有决策变量使用一个全局步长并不是一个很好的主意,尤其是当不同决策变量的范围有很大差异或者决策变量相关的时候。因此,对每个决策变量使用一个步长大小会更合理。在这种情况下,将会有 n 种步长大小。

类似地,如下给出了基于个体步长大小的进化策略对步长大小和决策变量的变异公式:

$$\sigma_i(t+1)=\sigma_i(t)\exp(\tau' N(0,1)+\tau N_i(0,1)),\quad i=1,2,\cdots,n \tag{3.17}$$

$$x_i(t+1)=x_i(t)+\sigma(t+1)N_i(0,1),\quad i=1,2,\cdots,n \tag{3.18}$$

其中,$\tau'\sim 1/\sqrt{2n}$ 和 $\tau\sim 1/\sqrt{2\sqrt{n}}$ 是两个参数。$N(0,1)$表示从正态分布采样的一个随机数,并用于所有的决策变量变异;而 $N_i(0,1)$表示对每个决策变量重新采样的随机数。同样,步长大小的变异必须在决策变量变异之前。

3.4.4 繁殖与环境选择

进化策略中的环境选择与遗传算法中的环境选择有很大的不同,其不同主要在于前者采用的是一种确定性的选择策略。它通过产生多于父代个体数的子代种群来实现。也就是说,给定 μ 个父代个体,将通过上述描述的变异策略产生 $\lambda>\mu$ 个子代个体。比如,(15,

100)-ES 中，$\mu=15$ 表示父代种群规模，$\lambda=100$ 是产生的子代种群大小。在繁殖过程中，随机选择一个父代个体，分别通过对其步长大小和决策变量的变异来产生一个子代个体。这个过程重复 100 次后产生 100 个子代个体。

在进化策略中有两种环境选择策略的变种：一种称为非精英策略，即(μ,λ)-ES；另一种称为精英策略，即$(\mu+\lambda)$-ES。在非精英选择策略中从 λ 个子代个体中选择最优的 μ 个个体作为下一代父代个体；而在精英选择策略中，从 $\mu+\lambda$ 个个体中选择最优的 μ 个个体作为下一代的父代个体。

进化策略的总体框架可以总结如下。

(1) 随机初始化 μ 个父代个体，其中决策变量在给定搜索空间中随机产生，初始步长大小在 0 和最大初始步长之间随机产生，一般设定为搜索范围的三分之一。

(2) 基于式(3.15)和式(3.16)或者式(3.17)和式(3.18)变异产生 λ 个个体。

(3) 从 λ 个子代个体中选择 μ 个最好的个体(非精英策略)；或者从 $\mu+\lambda$ 个个体中选择 μ 个最好的个体(精英策略)，作为下一代的父代个体。

(4) 如果不满足停止条件，则转到步骤(2)。

一开始进化策略并没有交叉或重组操作。然而，当最优解在给定搜索空间的中间时，加入交叉或重组操作能有效加快搜索速度。由于进化策略使用实值编码，以下 4 种重组方法，包括局部离散重组、局部中间重组、全局离散重组及全局中间重组，都可以在实行变异操作之前应用于决策变量和步长大小。

进化策略中的局部重组操作对所有变量使用相同的两个父代个体。在重组之前，从父代种群中随机选择两个父代个体 x_{P1} 和 x_{P2}，然后对每个决策变量执行以下操作。

(1) 随机选择两个父代个体 x_{P1} 和 x_{P2}。

(2) 对于每个个体 $i=1,2,\cdots,n$

$$x'_i=x_{\mathrm{P1},i} \text{ 或 } x_{\mathrm{P2},i}(\text{离散重组}) \tag{3.19}$$

$$x'_i=x_{\mathrm{P1},i}+\xi(x_{\mathrm{P2},i}-x_{\mathrm{P1},i})(\text{中间重组}) \tag{3.20}$$

其中，$\xi\in(0,1)$是用户自定义参数。

相反，全局重组操作可以对不同决策变量使用不同的父代个体。也就是说，对每个决策变量 x_i 随机选择两个父代个体，然后执行式(3.19)或式(3.20)中给出的重组操作。

3.4.5　协方差矩阵自适应进化策略

个体步长大小的进化策略提供了沿不同决策变量独自适应搜索过程的灵活性。然而，当不同决策变量之间存在相关性时，这种方法还是不够的。由于决策变量之间的成对依赖关系可以通过协方差矩阵来获得，因此协方差矩阵自适应进化策略(Covariance Matrix Adaptation Evolution Strategy，CMA-ES)旨在调整协方差矩阵从而高效地找到最优解。式(3.21)中概率密度函数的全协方差矩阵 $\boldsymbol{C}$ 适用于决策变量的变异，其中 n 是决策变量的

维度

$$f(\boldsymbol{z})=\frac{\sqrt{\det(\boldsymbol{C}^{-1})}}{(2\pi)^{n/2}}\exp\left(-\frac{1}{2}(\boldsymbol{z}^{\mathrm{T}}\boldsymbol{C}^{-1}\boldsymbol{z})\right) \tag{3.21}$$

去随机化的 CMA-ES[67]可以通过以下方式表示：

$$\boldsymbol{x}(t+1)=\boldsymbol{x}(t)+\delta(t)\boldsymbol{B}(t)\boldsymbol{z},\quad z_i\sim N(0,1) \tag{3.22}$$

其中，$\delta(t)$是第 t 代的整个步长大小，$z_i\sim N(0,1)$，$\boldsymbol{C}=\boldsymbol{B}\boldsymbol{B}^{\mathrm{T}}$，即 $\boldsymbol{B}\boldsymbol{z}\sim N(\boldsymbol{0},\boldsymbol{C})$。

协方差矩阵的自适应使用累积步长大小方法，通过两步来实现。其中 $c_{\mathrm{cov}}\in(0,1)$和 $c\in(0,1)$表示在累积自适应过程中过去对当前的影响。

$$\boldsymbol{s}(t+1)=(1-c)\boldsymbol{s}(t)+c_u\boldsymbol{B}(t)\boldsymbol{z} \tag{3.23}$$

$$\boldsymbol{C}(t+1)=(1-c_{\mathrm{cov}})\boldsymbol{C}(t)+c_{\mathrm{cov}}\boldsymbol{s}(t+1)\boldsymbol{s}^{\mathrm{T}}(t+1) \tag{3.24}$$

由于 $\boldsymbol{C}=\boldsymbol{B}\boldsymbol{B}^{\mathrm{T}}$ 并不足以推导出 $\boldsymbol{B}$，因此从 $\boldsymbol{C}$ 中计算矩阵 $\boldsymbol{B}$ 并不是一件小事。为此，我们可以使用 $\boldsymbol{C}$ 的特征向量作为 $\boldsymbol{B}$ 的列向量。这是因为整个步长大小 δ 必须要和上一步适应。为了实现这一点，我们需要知道累积向量 $\boldsymbol{s}_\delta$ 的期望长度。因此，依赖于 $\boldsymbol{B}$ 的变异(突变)向量应该服从分布 $N(0,1)$。若 $\boldsymbol{B}$ 的列向量是 $\boldsymbol{C}$ 的特征向量，则其可以通过使用相应特征值方差根来归一化 $\boldsymbol{B}$ 的列向量，从而生成矩阵 $\boldsymbol{B}_\delta$。最后，$\boldsymbol{s}_\delta$ 和 δ 可以通过下式进行自适应：

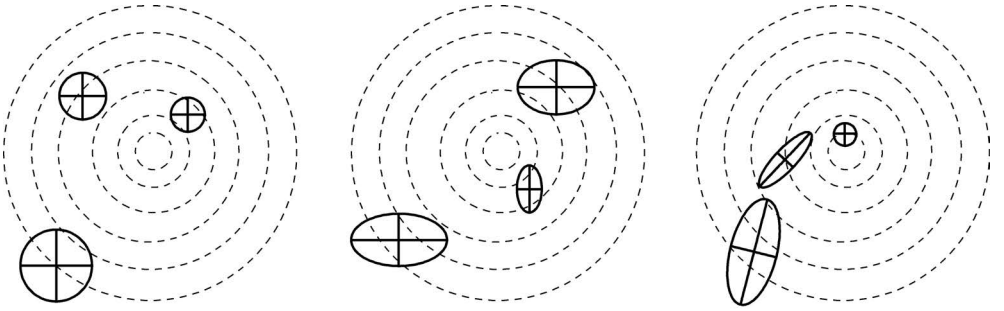
$$s_\delta(t+1)=(1-c)\boldsymbol{s}_\delta(t)+c_u\boldsymbol{B}_\delta(t)\boldsymbol{z} \tag{3.25}$$

$$\delta(t+1)=\delta(t)\exp(\beta(\|\boldsymbol{s}_\delta(t+1)\|-\hat{\chi}_n)) \tag{3.26}$$

其中，$\boldsymbol{B}_\delta$ 等于列向量归一化后的 $\boldsymbol{B}$，因此 $\boldsymbol{B}_\delta(t)\boldsymbol{z}$ 遵循 $N(0,1)$分布。$\hat{\chi}_n$ 表示 χ_n 分布的期望，这是一个长度分布，其是满足 $N(0,1)$分布的一个随机向量。

CMA-ES 包含几个需要指定的超参数，关于这些参数的更多细节可以查阅文献[67]。

具有全局步长的 ES、具有个体步长的 ES 以及 CMA-ES 的主要区别通过图 3.7 进行说明。在图 3.7(a)中，所有决策变量的步长大小相同，即产生变异样本的分布对所有决策变量(圆圈)具有相同的方差。相反，由于不同决策变量允许使用不同的步长大小，因此不同变



(a) 具有全局步长的ES　(b) 具有个体步长的ES　(c) CMA-ES

图 3.7　不同 ES

量的分布具有不同的方差(椭圆),如图 3.7(b)所示。另外,由于 CMA-ES 能够适应协方差矩阵,因此分布可以看作是一个可旋转的椭圆,从而获得最有效的搜索性能,如图 3.7(c)所示。大量研究表明,CMA-ES 可以很好地解决连续优化问题。

3.5　遗传规划

Canonical-PSO 代码

CSO 代码

LP-PSO 代码

遗传规划[68]是一类特殊的进化算法,其能够自动求解具有特殊的高级形式、模式或结构的优化问题。尽管遗传规划可以看成是一种采用不同表示形式的遗传算法,典型地通常使用决策树结构,由于其很适合求解回归和特征选择问题,因此其也可以认为是一种机器学习方法。

遗传规划已经在符号回归[69]、分类[70]和特征选择[71]中获得了应用,这些工作可以用于工程设计中,如天线[72]、拓扑结构[73]、电子电路[74]以及软件代码[75]。

3.5.1　基于树结构的遗传规划

遗传规划可以使用不同的表示方式,包括基于树结构的遗传规划、基于堆栈结构的遗传规划、线性遗传规划、基于图结构的遗传规划以及强类型的遗传规划。其中,树结构是遗传规划最早且最广泛使用的一种表示方式[76]。因此,本章采用基于树结构的遗传规划作为例子来讲解遗传规划的流程。

如图 3.8 所示,遗传规划包含进化算法的主要步骤:初始化、遗传变异、适应度评估以及环境选择。但是遗传规划和其他进化算法技术的不同点在于其编码(或表示)方式,因此其具有不同的初始化和变异操作。

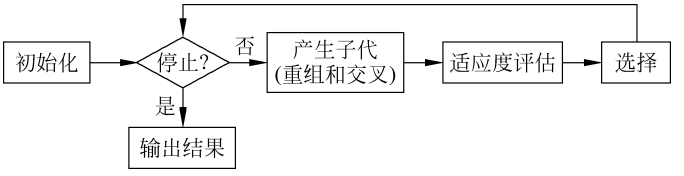


图 3.8　遗传规划的一般流程

在基于树结构的遗传规划中,程序或者数学表达式通过语法树而不是二进制数或实数来表示[77]。图 3.9 给出了数学表达式 $3a(3+(y+10))$ 的决策树表示形式。在这棵决策树中,变量或常量(3、a、y 和 10)称为叶节点(或终端节点),运算符或函数(+)称为内部节点。

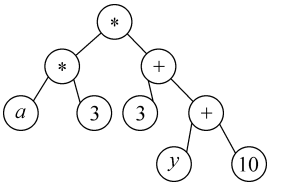


图 3.9　$3a(3+(y+10))$ 的树结构表示

在树结构中,所有可能的变量、常量、运算符和函数被称作基元,它们决定了遗传规划的搜索空间。基元集合包括存储叶节点

的终端节点集合以及存储内部节点的函数集合[78]。根据不同的问题，终端节点集合可以是外部输入，如决策变量或特征、常量；函数集合可以是算法、布尔、数学或者编程函数。表 3.1 是常见的基元。

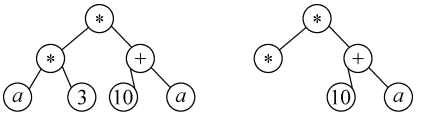
表 3.1 遗传规划中的常见基元

终端节点集合	
类别	**例子**
变量	x、y、a
常量	3、y
函数集合	
类别	**例子**
算法	+、*、−、/
布尔	AND、OR、NOT
数学	sin、tan、cos、exp
程序	WHILE、FOR、IF、ELSE、END

基元集合的选择对于遗传规划算法的有效性是非常重要的。一般来说，需要谨慎考虑以下两点：封闭性和充分性[64]。封闭性包含类型一致性和评价安全性，用于避免节点的随意性并确保变异中产生的表达式是可行的。充分性是指基元集合应该包含足够的表达式变种使其能够表示求解问题的最优解。

3.5.2 初始化

由于解的不同表示，遗传规划中其初始化方法是不同于其他进化算法的。两种基本的初始化方法（完全法和生长法）如图 3.10 所示。这两种方法均从基元集合中随机选取节点。完全法用于构建深度预定义的完全二叉树，而生长法使用深度优先搜索方法逐渐生成树，直到这棵树满足预设的深度。图 3.10 说明了这两种方法的不同之处。



(a) 完全法　　(b) 生长法

图 3.10 遗传规划中两种基本的初始化方法

尽管上述两种方法容易实现，但其树的形状多样性不够。因此，这两种方法的结合（称为半完全半生长法）[64]得到广泛采用，即初始种群中一半个体采用完全法产生，而另一半采用生长法产生。

3.5.3 交叉与变异

继承于进化算法，基于树结构的遗传规划在产生新的候选解时使用遗传算法相同的交

叉和变异操作，称为重组与变异。图 3.11 和图 3.12 给出了子树层的两种操作。重组操作从每个父代个体上选择一棵子树，组合成一棵新树作为子代个体。变异操作选择一个节点进行变异后来使用一个随机生成的子树来替代。

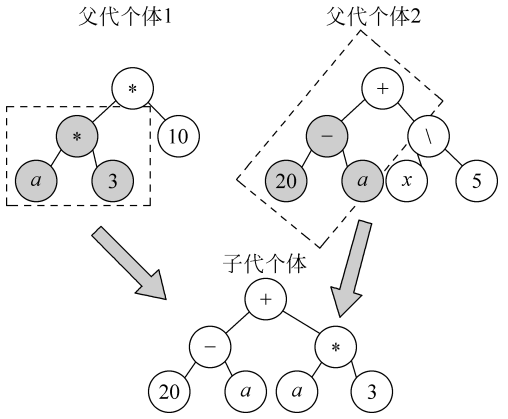


图 3.11　遗传规划中的重组操作示例

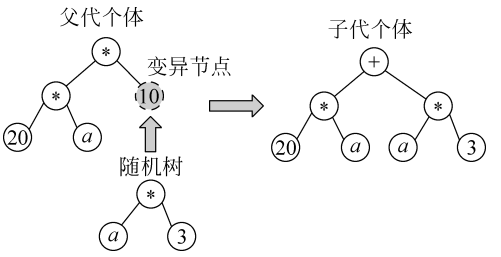


图 3.12　遗传规划中的变异操作示例

为了评价候选解的质量，遗传规划中的适应度函数可以根据问题定义成多种形式，可以是近似误差、分类准确率、系统到达预定状态所消耗的资源或时间，或者含有用户定义标准的结构。同样，一旦完成适应度评估，遗传算法中的选择机制[79]就可应用于遗传规划中，比如锦标赛选择机制以及基于适应度比例的轮盘赌选择机制。

3.6　蚁群优化算法

受自然界蚂蚁通过信息素交流的启发[80-81]，ACO 算法最初提出时用于求解旅行商问题[82]。ACO 算法采用 N 个人工蚂蚁（或称为智能体）来模拟蚁群寻找食物的过程，从而完成优化任务。由于 ACO 算法结合了随机和贪婪机制，它可以较好地求解 NP 难的组合优化问题[83]。

图 3.13 是 ACO 算法主要思想的一个简单示例，该问题的目标是寻找从起点 A 到终点 E 的最短路径。ACO 算法采用近乎相同数量的蚂蚁来初始化每一条路径，随后，每迭代一次，蚂蚁通过共享它们的信息素 τ 以显示它们到终点 E 的距离信息，并根据 τ 的分布调整它们的行动策略。显然，短的距离会产生高的信息素 τ，进而吸引更多的蚂蚁探索较短路径。当 ACO 算法结束时，蚂蚁就找到了到达终点 E 的最短路径。

3.6.1　整体框架

ACO 算法的一般过程见算法 3.1，其主要包括 4 个主要操作（初始化、解的构建、局部

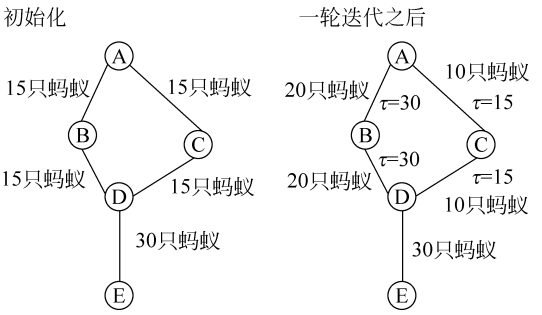


图 3.13　蚁群优化算法示例

搜索和信息素更新)。

算法 3.1　ACO 算法伪代码

```
1: 初始化
2: while 终止条件未达到 do
3:     构建 N 个蚂蚁个体
4:     进行局部搜索
5:     更新信息素
6: end while
输出: 最优解
```

ACO 算法在每次开始时,N 个蚂蚁个体的信息素均设为 τ_0。每个蚂蚁个体 $\boldsymbol{x}$ 基于启发式因子和信息素构建的。假设 $\boldsymbol{x}$ 的第 i 维有若干个可行值,那么分配为第 j 个可行值的概率为

$$p(x_i^j)=\frac{\tau_{ij}^{\alpha}(\eta(x_i^j))^{\beta}}{\sum\tau_{il}^{\alpha}(\eta(x_i^l))^{\beta}} \tag{3.27}$$

其中,τ_{ij} 和 $\eta(x_i^j)$分别是 x_i 的第 j 个可行值的信息素和启发式因子;α 和 β 是两个预定义参数,用于平衡启发式信息和信息素,作为可选项,当构建完 $\boldsymbol{x}$ 后可以应用局部搜索来提高解的质量。为了引导对最优解的搜索,每一代信息素通过两种机制来进行更新。首先,将优质解存放入集合 S_{upd} 中,如果 x_i^j 在集合 S_{upd} 中,则提高 τ_{ij}。随后,信息素随着迭代而逐渐衰减。因此,可用如下公式更新信息素:

$$\tau_{ij}=(1-\rho)\tau_{ij}+\sum_{\boldsymbol{x}\in S_{\text{upd}}|x_i^j\in\boldsymbol{x}}g(\boldsymbol{x}) \tag{3.28}$$

其中,ρ 为遗忘率;$g(\boldsymbol{x})$是为优质解的 x_i^j 部分分配信息素增益的函数,该函数依赖于适应度函数 $f(\boldsymbol{x})$。当满足终止条件时,ACO 算法输出其所获得的最优解。

3.6.2　扩展应用

ACO 算法已经被成功应用于求解很多组合优化问题,包括路径规划[84]、分配问题[85]、

车间调度问题[86]、子集选取问题[87]、分类问题[88]和生物信息学中的单倍型推断问题[89]。需要注意的是，ACO 算法也已经被应用于连续优化问题[90]。

3.7　差分进化算法

差分进化[65]算法是 1996 年由 Price 和 Storm 提出的一类基于种群的进化算法，其概念简单，能有效求解连续优化问题。给定一个初始种群，差分进化通过执行 3 种操作寻找 n 维问题的全局最优解。这 3 种操作包括变异、交叉和环境选择。当算法满足停止准则时，如达到迭代的最大次数或者达到最大的适应度评估次数，则输出最优解。与其他进化算法不同的是，对于当前种群中的任意解，差分进化算法选择当前种群中不同于该解的任意两个解的差向量对其进行扰动。

图 3.14 给出了差分进化算法的流程图。下面将详细介绍差分进化算法的主要组成部分。

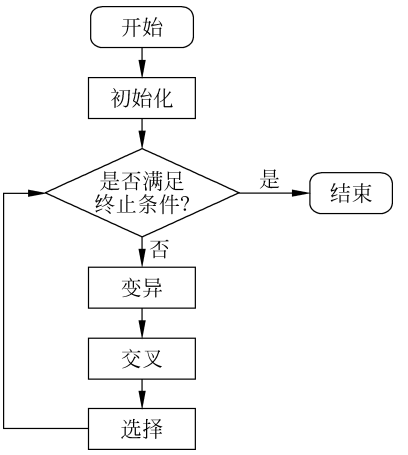


图 3.14　DE 流程图

3.7.1　初始化

在优化问题的决策空间 $\mathbf{R}^n$ 中随机产生 N 个个体组成种群 P，其中，n 表示决策变量的个数。任意个体 $\boldsymbol{x}_i$ 在第 j 个维度上的位置由下式产生：

$$x_{ij}(t)=x_{j,\min}+r^*(x_{j,\max}-x_{j,\min}) \tag{3.29}$$

其中，种群初始化时设置 $t=0$，r 是[0,1]范围内的一个随机数。$\boldsymbol{x}_{\min}=(x_{1,\min},x_{2,\min},\cdots,x_{n,\min})$和 $\boldsymbol{x}_{\max}=(x_{1,\max},x_{2,\max},\cdots,x_{n,\max})$分别是下界和上界。

3.7.2　差分变异

在进化算法中，差分变异可以视作是对决策变量的一个扰动。差分进化算法为种群 P 中的每个个体 i 通过对基向量添加一个或多个差向量来产生一个变体，该变体称为贡献向量 $\boldsymbol{u}$。下面给出了一些常用的变异策略：

(1) DE/rand/1

$$\boldsymbol{u}_i(t+1)=\boldsymbol{x}_{r1}(t)+F(\boldsymbol{x}_{r2}(t)-\boldsymbol{x}_{r3}(t)) \tag{3.30}$$

(2) DE/best/1

$$\boldsymbol{u}_i(t+1)=\boldsymbol{x}_{\text{best}}+F(\boldsymbol{x}_{r1}(t)-\boldsymbol{x}_{r2}(t)) \tag{3.31}$$

(3) DE/rand/2

$$\boldsymbol{u}_i(t+1)=\boldsymbol{x}_{r1}(t)+F(\boldsymbol{x}_{r2}(t)-\boldsymbol{x}_{r3}(t))+F(\boldsymbol{x}_{r4}(t)-\boldsymbol{x}_{r5}(t)) \tag{3.32}$$

(4) DE/best/2

$$\boldsymbol{u}_i(t+1)=\boldsymbol{x}_{\text{best}}+F(\boldsymbol{x}_{r1}(t)-\boldsymbol{x}_{r2}(t))+F(\boldsymbol{x}_{r3}(t)-\boldsymbol{x}_{r4}(t)) \tag{3.33}$$

(5) DE/current-to-best/1

$$\boldsymbol{u}_i(t+1)=\boldsymbol{x}_i(t)+F(\boldsymbol{x}_{\text{best}}-\boldsymbol{x}_i(t))+F(\boldsymbol{x}_{r1}(t)-\boldsymbol{x}_{r2}(t)) \tag{3.34}$$

(6) DE/rand-to-best/1

$$\boldsymbol{u}_i(t+1)=\boldsymbol{x}_{r1}(t)+F(\boldsymbol{x}_{\text{best}}-\boldsymbol{x}_{r1}(t))+F(\boldsymbol{x}_{r2}(t)-\boldsymbol{x}_{r3}(t)) \tag{3.35}$$

在式(3.30)～式(3.35)中，$\boldsymbol{u}_i(t+1)$表示个体 i 在第 $t+1$ 代时通过变异操作产生的向量，等式右边的第一项称为基向量。$\boldsymbol{x}_i(t)$和 $\boldsymbol{x}_{\text{best}}$ 分别表示个体 i 在第 t 代时的决策向量和迄今为止所找到的最优解。$r1$、$r2$、$r3$、$r4$ 和 $r5(r1\neq r2\neq r3\neq r4\neq r5\neq i)$表示随机数，$F$ 为缩放因子。

3.7.3 差分交叉

差分进化算法中交叉操作用于提升变异后种群的多样性。常见用于产生试验向量$\boldsymbol{v}$ 的两种交叉方法为二项式交叉和指数交叉。在二项式交叉中，试验向量$\boldsymbol{v}_i$ 的第 j 个决策变量的值根据设定的交叉率(Cr)从其父代向量 $\boldsymbol{x}_i$ 或贡献向量 $\boldsymbol{u}_i$ 中选取。交叉率(Cr)通常是[0,1]范围内的一个常数。式(3.36)给出了二项式交叉方式：

$$v_{ij}(t+1)=\begin{cases}u_{ij}(t+1), & (\text{rand}()\leqslant \text{Cr}\ \text{或}\ j=j_{\text{rand}})\\ x_{ij}(t), & \text{其他}\end{cases} \tag{3.36}$$

其中，rand()表示一个函数，用于生成[0,1]范围内的一个随机数，$j_{\text{rand}}\in\{1,2,\cdots,n\}$是随机选择的一个索引，用于确保至少有一个决策变量的值是选自贡献向量中的。

在指数交叉中，从[1,n]范围中随机选择一个整数，将其作为从父代向量中选择的起始点，然后从环形方式阵列的贡献向量 $\boldsymbol{u}$ 中选择连续的 $L(L<n)$个变量值。即在满足 rand()>Cr 之前或从贡献向量中获得变量总数达到 L 之前，试验向量的决策变量值将从贡献向量中获取，其所有其他决策变量值都来源于父代向量。算法 3.2 给出了指数交叉的伪代码。

算法 3.2 指数交叉

输入：$\boldsymbol{x}_i$ $\boldsymbol{u}_i$
输出：$\boldsymbol{v}_i$
1: $\boldsymbol{v}_i=\boldsymbol{x}_i$
2: 在集合{1,2,…,n}中随机选取一个整数 k;
3: $l=1$;
4: $j=k+l$;
5: 重复
6: $v_{ij}=u_{ij}$;
7: $l=l+1$;
8: $j=k+l$;
9: 直至满足(rand()>Cr 或者 $l>L$)

3.7.4　环境选择

最后，通过父代向量和试验向量的适应度来确定哪个解会保留到下一代。选择适应度不差于当前父代向量的向量替换当前父代向量：

$$\boldsymbol{x}_i(t+1)=\begin{cases}\boldsymbol{v}_i(t+1), & f(\boldsymbol{v}_i(t+1))\leqslant f(\boldsymbol{x}_i(t))\\ \boldsymbol{x}_i(t), & \text{其他}\end{cases} \tag{3.37}$$

3.8　粒子群优化算法

3.8.1　传统的粒子群优化算法

PSO 算法是 Kennedy 和 Eberhart 在 1995 年提出的模拟鸟群或鱼群觅食行为的优化算法[91]。它是一种基于种群的搜索方法，在这种方法中，一系列个体（或称为粒子）相互协作以找到问题的最优解。不同于遗传算法或差分进化算法，PSO 算法中的每个个体在每一代 t 不仅具有位置 $\boldsymbol{x}_i$，还有速度 $\boldsymbol{v}_i$。每个个体通过向自身经验和群体经验学习来更新自己的速度，可以用如式(3.38)所示的数学公式表示：

$$v_{ij}(t+1)=\omega v_{ij}(t)+c_1r_1(p_{ij}(t)-x_{ij}(t))+c_2r_2(g_j-x_{ij}(t)) \tag{3.38}$$

相应地，第 i 个粒子的位置可以通过式(3.39)进行更新：

$$x_{ij}(t+1)=x_{ij}(t)+v_{ij}(t+1) \tag{3.39}$$

其中，$\boldsymbol{v}_i(t)=(v_{i1}(t),v_{i2}(t),\cdots,v_{in}(t))$ 和 $\boldsymbol{x}_i(t)=(x_{i1}(t),x_{i2}(t),\cdots,x_{in}(t))$ 分别表示个体 i 在第 t 代时的速度和位置；$\boldsymbol{p}_i(t)=(p_{i1},p_{i2},\cdots,p_{in})$ 和 $\boldsymbol{g}(t)=(g_1,g_2,\cdots,g_n)$ 表示迄今为止个体 i 找到的最好位置以及种群所找到的最好位置：

$$\boldsymbol{p}_i(t)=\underset{k=0,1,2,\cdots,t}{\operatorname{argmin}} f(\boldsymbol{x}_i(k)) \tag{3.40}$$

$$\boldsymbol{g}(t)=\underset{k=0,1,2,\cdots,t}{\operatorname{argmin}} \{f(\boldsymbol{x}_1(k)),f(\boldsymbol{x}_2(k)),\cdots,\boldsymbol{x}_N(k)\} \tag{3.41}$$

其中，$\boldsymbol{w}$ 称为惯性权重；c_1 和 c_2 为加速度系数，均为正常数，其中 c_1 称为认知参数，c_2 称为社会参数；r_1 和 r_2 是在[0,1]范围内均匀产生的随机数。

从式(3.38)可以看出，个体的速度更新包含 3 部分。第一部分称为惯性向量，反映个体的原搜索方向。第二部分称为认知学习，这部分鼓励个体向自身找到的最优位置学习。第三部分是社会学习，这部分鼓励个体向迄今为止种群发现的最优位置学习。因此，种群的最优位置代表寻找全局最优解中协作行为的结果。

图 3.15 给出了一个简单的例子，用于说明一个个体在二维决策空间中如何获得下一个位置。在图 3.15 中，三条带箭头的虚线分别表示式(3.38)中的 3 部分。如图 3.15 所示，参数 $\boldsymbol{w}$ 用于控制动量，φ_1 和 φ_2 分别控制向个体最优位置和向至今找到的最优解的学习程

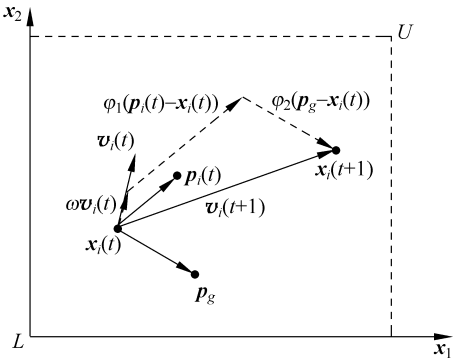


图 3.15 更新个体 i 位置的简单例子

度，有

$$\varphi_1 = c_1 r_1, \quad \varphi_2 = c_2 r_2$$

算法 3.3 给出了 PSO 算法的伪代码。首先在上下界范围内随机产生 N 个个体组成初始种群，每个个体均具有速度和位置。每个个体 i 的个体最优位置 $\boldsymbol{p}_i$ 即为每个个体的当前位置，记录所有个体至今找到的最优位置 $\boldsymbol{g}(t)$。初始化后，每个个体根据式(3.38)和式(3.39)更新自己的速度和位置，并评估其适应度。对每个个体 i，若其当前位置的适应度优于个体最优位置，则更新其个体最优位置。所有个体的最优位置用于更新迄今为止所找到的全局最优位置。当满足停止条件时，输出到目前为止找到的最优位置及其适应度。

PSO 算法由于其实现简单、收敛速度快而受到广泛欢迎。然而，粒子群算法很容易陷入局部最优，特别是在高维空间中，加剧了这种情况。因此，提出了许多新的学习策略[92-94]，其目的是提高种群的多样性，从而避免算法陷入局部最优。接下来，我们将介绍两种用于求解高维或大规模优化问题的变种 PSO 算法。

算法 3.3：PSO 算法

输入：N：种群规模；
输出：gbest：迄今为止找到的最优解.

```
初始化种群 P;
评估种群中每个个体 i 的适应度;
t = 0;
将每个个体 i 的当前位置 x_i(t), i = 1,2,…,N 设置为个体最优位置,即 p_i(t) = x_i(t);
在所有个体最优解中找出最好的位置作为到目前为止找到的最优位置 g(t);
while 不满足终止条件 do:
    for 每个个体 i
        使用式(3.38)和式(3.39)分别更新个体 i 的速度和位置;
        评价个体 i 的适应度;
        if f(x_i(t+1)) ≤ f(p_i(t))
            p_i(t+1) = x_i(t+1);
        else
            p_i(t+1) = p_i(t);
        end if
        if f(p_i(t+1)) ≤ f(g(t))
            g(t+1) = p_i(t+1);
        else
            g(t+1) = g(t);
        end if
    end for
end while
输出 g(t+1) 和 f(g(t+1)).
```

3.8.2　竞争粒子群优化器

收敛速度和全局搜索能力是基于种群优化算法的两个重要特性。为了减少早熟收敛，在同一个种群个体之间引入竞争机制，从而形成 PSO 算法的一种变种算法，称为竞争粒子群优化器(Competitive Swarm Optimizer，CSO)[93]。图 3.16 给出了竞争粒子群优化器的一般思路。在 CSO 中，假设种群规模 N 为偶数，将当前种群 $P(t)$ 随机平分为 $N/2$ 对。比如，在图 3.16 中，当前种群 $P(t)$ 包含的 6 个个体被随机平分为 3 组，然后每组中的两个个体根据适应度进行竞争。假设 $\boldsymbol{x}_i(t)$ 和 $\boldsymbol{x}_j(t)$ 是一对，它们将根据各自的适应度相互竞争，适应度好的个体定义为胜利者，另一个个体则被定义为失败者。胜利者会直接保留到下一代，而失败者则会通过向胜利者学习来更新自己的速度和位置。假设在给定的例子中，$\boldsymbol{x}_i(t)$ 比 $\boldsymbol{x}_j(t)$ 具有更好的适应度，那么如图 3.16 所示，$\boldsymbol{x}_i(t)$ 将直接传递到下一代，即 $\boldsymbol{x}_i(t+1)=\boldsymbol{x}_i(t)$，而 $\boldsymbol{x}_i(t)$ 则通过向胜利者 $\boldsymbol{x}_i(t)$ 学习来更新位置(和速度)，从而得到 $\boldsymbol{x}_j(t+1)$。所以从图 3.16 中可以看到，在每一代只有种群中的 $N/2$ 个个体需要更新。

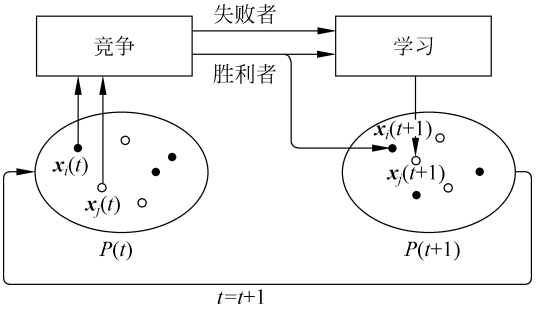


图 3.16　竞争群体优化器的一般思路

假设 $\boldsymbol{x}_{w,k}(t)$、$\boldsymbol{x}_{l,k}(t)$、$\boldsymbol{v}_{w,k}(t)$ 和 $\boldsymbol{v}_{l,k}(t)$ 分别表示在第 t 代、第 k 轮竞争中胜利者和失败者的位置和速度，其中，$1\leqslant k\leqslant N/2$。相应地，失败者的速度将通过以下学习策略进行更新：

$$\boldsymbol{v}_{l,k}(t+1)=\boldsymbol{R}_1(k,t)\,\boldsymbol{v}_{l,k}(t)+\boldsymbol{R}_2(k,t)(\boldsymbol{x}_{w,k}(t)-\boldsymbol{x}_{l,k}(t))+\varphi\boldsymbol{R}_3(k,t)(\bar{\boldsymbol{x}}_k(t)-\boldsymbol{x}_{l,k}(t)) \tag{3.42}$$

相应地，失败者的位置将通过以下进行更新：

$$\boldsymbol{x}_{l,k}(t+1)=\boldsymbol{x}_{l,k}(t)+\boldsymbol{v}_{l,k}(t+1) \tag{3.43}$$

在式(3.42)和式(3.43)中，$\boldsymbol{R}_1(k,t)$，$\boldsymbol{R}_2(k,t)$，$\boldsymbol{R}_3(k,t)\in[0,1]^n$ 是第 t 代、第 k 轮竞争中随机产生的 3 个向量，$\bar{\boldsymbol{x}}_k(t)$ 是相关粒子的平均位置，φ 是控制平均位置 $\bar{\boldsymbol{x}}_k(t)$ 产生影响的参数。可以使用两种方法来计算平均位置：一种是全局版本的平均位置，标记为 $\bar{\boldsymbol{x}}_k^g(t)$，其是当前种群 $P(t)$ 中所有粒子的平均位置；另一种是局部版本的平均位置，记为 $\bar{\boldsymbol{x}}_{l,k}^l(t)$，其是第 t 代、第 k 轮竞争中失败者邻域个体的平均位置。式(3.42)中的第一部分用于确保搜索过程的稳定性，这与传统 PSO 算法中保持之前方向的动量一样。式(3.42)的第二部分 $\boldsymbol{R}_2(k,t)(\boldsymbol{x}_{w,k}(t)-\boldsymbol{x}_{l,k}(t))$ 也被称为认知部分，然而，与传统的 PSO 算法不同的是，失败

个体是向同一对的胜利个体学习，而不是向其个体至今所发现的最优位置学习。从生物学的角度来说，这种方式能更好地体现动物的群体行为，因为很难要求所有的个体都能记住它们在过去所经历过的最优位置。式(3.42)的第三部分 $\varphi \boldsymbol{R}_3(k,t)(\bar{\boldsymbol{x}}_k(t)-\boldsymbol{x}_{l,k}(t))$，和传统PSO算法中一样，也被称为社会组成部分。然而，CSO中没有使用到目前为止种群所找到的最优位置，而是使用了种群的平均位置，且也不需要记录到目前为止找到的最优位置。

算法3.4给出了竞争粒子群优化器的伪代码。在决策空间中随机生成一个初始种群 P 并且使用适应度函数对其进行评价。将当前种群中具有最小适应度的解设置为到目前为止所找到的最好解，记为gbest。重复下列过程直到满足停止条件为止。将当前种群随机分为规模大小为 $N/2$ 的两个子种群，分别保存在集合 S_1 和 S_2 中。正如5-16行所示，对集合 S_1 和 S_2 中的每一对个体，比较它们的适应度大小，并将适应度好的个体作为 $\boldsymbol{x}_{w,k}(t)$，把剩下另一个作为 $\boldsymbol{x}_{l,k}(t)$。胜利个体将会直接保存到集合 U 中，而失败个体则使用式(3.42)和式(3.43)更新后保存到集合 U 中。集合 U 中的所有解即为下一代的父代粒子。当集合 U 中的所有解都经过适应度函数评价后，更新迄今为止所发现的最优解gbest。

算法3.4：竞争群体优化器的伪代码

```
输入：N：种群规模；
输出：gbest：迄今为止找到的最优解.
 1：初始化种群 P；
 2：评价种群中每个个体 i 的适应度；
 3：t = 0；
 4：找出种群 P 所包含解的最优位置，并将其设置为迄今为止找到的最优位置 gbest；
 5：while 终止条件不满足，则执行以下操作：
 6：     S1 = ∅；
 7：     S2 = ∅；
 8：     从种群 P 中随机分配 N/2 个解分配给 S1，剩下的解分配给 S2；
 9：     U = ∅；
10：     for k = 1 to N/2 do
11：         if f(S1(i))≤f(S2(i))，则
12：             xw,k(t) = S1(i)；
13：             xl,k(t) = S2(i)
14：         else
15：             xw,k(t) = S2(i)；
16：             xl,k(t) = S1(i)
17：         end if
18：         U = U∪xw,k(t)；
19：         使用式(3.42)和式(3.43)更新 xl,k(t)；
20：         将更新后的解保存到 U；
21：     end for
22：     P = U；
23：     t = t + 1；
24：     评估种群 P 中所有解的适应度，并更新迄今为止找到的最优解 gbest；
25：end while
26：输出迄今为止找到的最优解 gbest.
```

3.8.3　社会学习粒子群优化器

学习和模仿种群中较优个体的行为在社会性动物中随处可见，称为社会学习。与个体学习不同，社会学习的优势在于个体可以不需要通过试凑就可以从其他个体中学习行为。因此，很自然地可以将社会学习机制应用于基于种群的随机优化中。社会学习粒子群优化器(Social Learning Particle Swarm Optimizer，SL-PSO)是 PSO 算法的变种[94]，每个个体向当前种群中比其更好的任意粒子(也称为演示者)学习，而不是向历史最优位置学习。图 3.17 给出了 SL-PSO 的一般思路。对于最小化问题，当前种群 $P(t)$中的所有个体根据其适应度按降序排序(对于最大化问题，则按升序排序)。随后，当前种群中比 $\boldsymbol{x}_i(t)$适应度好的解将会成为 $\boldsymbol{x}_i(t)$的演示者，如图中浅蓝色网格部分。然后，$\boldsymbol{x}_i(t)$通过向它的演示者学习来更新它的速度和位置。需要注意的是，在 SL-PSO 中，除了最差的个体，每个解 $\boldsymbol{x}_i(t)$都可以作为不同模仿者(即比 $\boldsymbol{x}_i(t)$具有更差的适应度)的演示者。同样，除了最好的个体，每个解 $\boldsymbol{x}_i(t)$都可以向不同的演示者学习，且当前种群中的最优粒子不用更新。

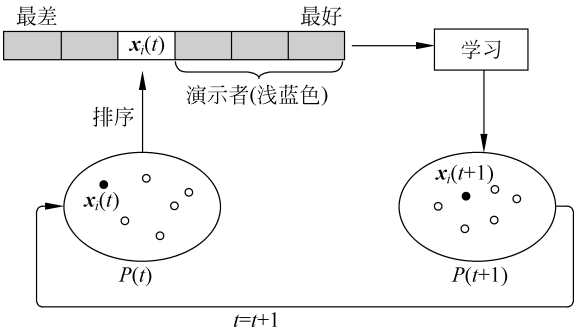


图 3.17　SL-PSO 的一般思想

在社会学习机制的启发下，模仿者会以以下的方式学习不同演示者的行为：

$$x_{ij}(t+1)=\begin{cases}x_{ij}(t)+\Delta x_{ij}(t+1), & p_i(t)\leqslant P_i^L\\ x_{ij}(t), & \text{其他}\end{cases} \tag{3.44}$$

其中，$x_{ij}(t)(i=1,2,\cdots,N;\ j=1,2,\cdots,n)$是第 t 代行为向量的第 j 个维度的行为，$\Delta x_{ij}(t+1)$是第 i 个粒子第 j 个维度上的修正行为。需要注意的是，在社会学习中，向更好个体学习的动机可能因人而异，因此为每个个体定义了学习概率 P_i^L。最终，只有当随机产生的概率 p_i 满足不等式条件 $0\leqslant p_i(t)\leqslant P_i^L\leqslant 1$ 时，个体 i 才使用式(3.45)向它的演示者学习：

$$\Delta x_{ij}(t+1)=r_1(t)\Delta x_{ij}(t)+r_2(t)(x_{kj}(t)-x_{ij}(t))+r_3(t)\in(\bar{x}_j(t)-x_{ij}(t)) \tag{3.45}$$

在式(3.45)中，$r_1(t)$、$r_2(t)$、$r_3(t)$分别表示在 t 代所产生的[0,1]范围内的 3 个随机数。$x_{kj}(t)$是第 t 代粒子 k 行为向量在第 j 维上的行为，$\bar{x}_j(t)=\dfrac{\sum_{i=1}^{N}x_{ij}(t)}{N}$是第 t 代当前种群所

有粒子在第 j 维度上的平均行为。参数 ε 称为社会影响因子，用于控制社会对这个粒子在这个维度上的影响程度。与传统 PSO 算法的速度更新类似，SL-PSO 算法中的速度更新也由 3 个部分组成。第一部分旨在保持先前方向上的动量，并确保搜索过程的稳定性。在式(3.45)的第二部分，个体 i 向它的演示者学习，这和传统粒子群算法中向个体历史最优位置学习是不同的。注意 $i < k \leqslant N$，个体 i 学习的演示者在不同维度上可能是不同的。在式(3.45)的第三部分也不同于传统的粒子群算法。在 SL-PSO 中，个体 i 是向整个种群学习，即向当前种群中所有粒子的平均行为学习，而不是传统 PSO 中向迄今为止种群所发现的最优位置学习。

算法 3.5 给出了 SL-PSO 的伪代码。生成一个初始种群并使用适应度函数对其个体进行评价。找出种群中所有个体的最优位置，并将其设为迄今为止所找到的最优位置，标记为 gbest。重复执行以下过程，直到满足停止条件为止。根据当前种群中所有个体的适应度按降序对其进行排序。除了最优个体，排序后的每个个体都通过向其演示者学习来更新其速度，当前种群中的个体位置也随之更新。对于分类种群中的每个个体，除了最优个体，都会通过向演示者学习来更新其速度。此后当前种群 P 的粒子位置将被更新。一旦个体 i 的每个维度都确定了其演示者，每个位置向量的组成将通过向不同的演示者和当前种群所有个体的平均位置学习而获得更新(第 7～14 行)。所有个体位置更新后对其进行评价，并相应地更新迄今为止所找到的最优位置 gbest。

算法 3.5：SL-PSO 的伪代码

```
输入：N：种群规模；
输出：gbest：迄今为止找到的最优解.
 1：初始化种群 P；
 2：评价种群中每个个体 i 的适应度；
 3：t = 0；
 4：找出种群 P 中所有解的最优位置，并将其设置为迄今为止所找到的最优位置 gbest；
 5：while 终止条件不满足，执行以下操作：
 6：     根据适应度对当前种群 P 按降序排序，相应地，更新种群中个体的索引；
 7：     for i = 1 to N − 1 do
 8：         for j = 1 to n do
 9：             在[i + 1,N]范围内产生随机整数 k；
10：             在[0,1]范围内随机产生 3 个数 r1(t),r2(t)和 r3(t)；
11：             计算当前种群所有个体在第 j 维上的平均位置；
12：             使用式(3.44)更新粒子 i 在第 j 维上的位置；
13：         end for
14：         评价更新后粒子 i 的适应度；
15：     end for
16：     使用当前种群所有个体中迄今为止找到的最优位置 gbest；
17：     t = t + 1；
18：end while
19：输出迄今为止找到的最优解 gbest.
```

3.9　模因算法

3.9.1　基本概念

模因算法是将局部搜索(也称为终身学习)嵌入进化算法的混合搜索方法。因此,模因算法既可以从全局的、随机的一般搜索中获益,也可以从局部的、特别是贪婪的局部搜索中获益。图 3.18 给出了模因算法的一般描述,可以看出,模因算法不同于进化算法的主要地方在于其在子代个体评价之前需要进行局部搜索。

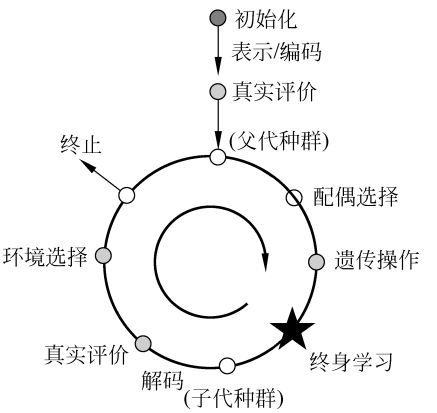


图 3.18　模因算法的一般框架

在进行局部搜索之前,有几个问题需要回答。例如,可以使用哪个局部搜索算法?局部搜索需要执行多少次迭代?所有的个体是否都需要进行局部搜索?不同的设置可能导致不同的搜索性能,这也跟优化问题息息相关。尽管局部搜索在原则上可以和二进制编码以及实值编码进化算法进行混合,但更普遍的做法是将局部搜索和二进制或灰度遗传算法相结合,因为这样能更好地发挥全局搜索和局部搜索相结合所获得的协同效用。

3.9.2　拉马克方法和鲍德温方法

模因算法有两种稍有差异的变种,分别为拉马克进化和鲍德温进化。需要注意的是,局部搜索是在表现型空间进行的,而遗传算法是在基因型空间进行的。因此,两种方法的主要区别在于局部搜索中决策变量的变化(表现型变化)是否会直接影响基因型,并使变化具有可遗传性。图 3.19 说明了模因算法中拉马克方法和鲍德温方法之间的区别。在图的左侧,对解 x_1 的表现型进行局部搜索后得 x_1',同时,改变的表现型直接编码到个体的基因型,从而个体的基因型经过局部搜索后从 g_1 到了 g_1',这种情况就是拉马克进化。反过来,在图的右侧,在解 x_2 上进行局部搜索后得 x_2'。然而,解的基因型没有改变,这种情况下就是鲍德温进化。需要注意的是,在环境选择中,x_1 的适应度为 $f(x_1')$,x_2 的适应度为 $f(x_2')$。因此,无论是拉马克进化还是鲍德温进化,局部搜索的结果都会影响环境选择。它们唯一的区别是,在拉马克进化中 x_1 的基因型会随着局部搜索而变化,而在鲍德温进化中仅有 x_2 的适应度发生了变化。

3.9.3　多目标模因算法

在单目标优化中,局部搜索中使用的目标函数通常与一般搜索相同。然而,由于很多如

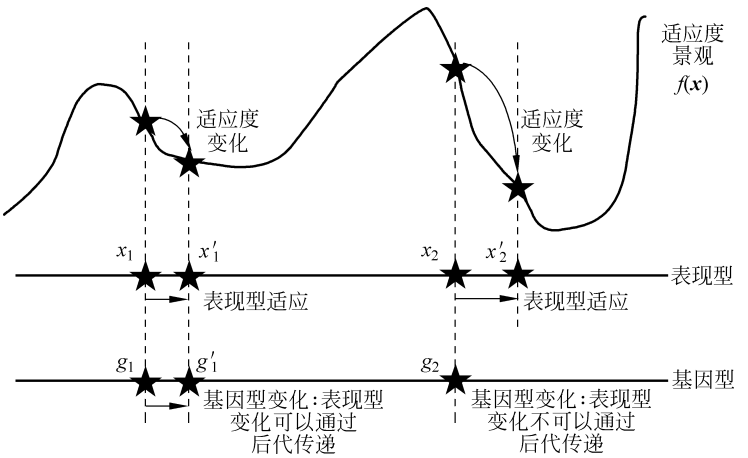


图 3.19 拉马克和鲍德温进化

基于梯度的方法的局部搜索算法在进行局部搜索之前都需要先将多目标优化问题转换成单目标优化问题，因此其不适合于进行多目标优化，多目标优化问题需要首先转换为单目标问题，才能进行局部搜索。为此，可以为种群中的每个个体分配一个随机生成的权重向量，从而可在多个方向上执行局部搜索。例如，对于求解具有 m 个目标的优化问题，可使用以下单目标问题进行局部搜索：

$$f(\boldsymbol{x})=\sum_{i=1}^{m} w_i f_i(\boldsymbol{x}) \tag{3.46}$$

其中，$w_i>0$，$\sum_{i=1}^{m} w_i=1$。

此外，也可以为当前种群中的每个个体定义一组伪权重。假设第 i 个目标函数的最小值和最大值分别是 $f_i^{\min}$ 和 $f_i^{\max}$，那么对于第 i 个目标值为 $f_i(x)$的个体伪权重可以通过以下方式计算：

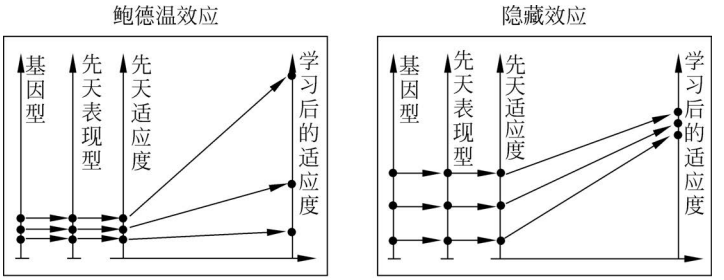
$$w_i=\frac{f_i^{\max}-f_i(\boldsymbol{x})}{f_i^{\max}-f_i^{\min}}\Bigg/\sum_{j=1}^{m}\frac{f_i^{\max}-f_i(\boldsymbol{x})}{f_i^{\max}-f_i^{\min}} \tag{3.47}$$

在某些情况下，也可以仅对决策变量的子集进行局部搜索。例如，在神经网络多目标优化中，权重和结构需要同时优化，我们可以只对权重使用梯度下降[95]进行局部搜索。

3.9.4 鲍德温效应与隐藏效应

局部搜索与进化搜索的结合旨在加速搜索过程，这被称为鲍德温效应。然而，这并不总是正确的，因为局部搜索也可能会减缓进化，这被称为隐藏效应。通常，如果局部搜索增大了当前种群中个体之间固有的适应度差异，则会出现鲍德温效应；而如果局部搜索减小了当前种群中个体之间固有的适应度差异，则会出现隐藏效应，如图 3.20 所示。在图 3.20(a)

中，3 个个体的固有适应度变大，因此，较弱的个体在终身学习后留下来的概率更小，从而形成较高的选择压力，导致出现鲍德温效应。相比之下，在图 3.20(b)中，3 个个体之间的固有适应度在终身学习后变小，导致形成较小的选择压力，从而产生了隐藏效应。需要注意的是，上述讨论中假设采用了适应度比例选择的策略。



(a) 鲍德温效应　　(b) 隐藏效应

图 3.20　局部搜索与进化搜索的关系

接下来，给出一个简单的数学证明，用于说明鲍德温或隐藏效应发生的条件[96]。假设一个最大化优化问题的固有适应度地形(没有学习)是单调递增并且是连续的，即 $f(\boldsymbol{x})>0$，它的一阶导数函数 $f'(\boldsymbol{x})>0$，并且它的一阶至四阶导数函数 $f^{(i)}(\boldsymbol{x})$，$i=1,2,3,4$ 是连续的。终身学习后其适应度地形标记为 $f_l(\boldsymbol{x})$，并且也是正向和单调递增的。对于一个规模大小为 N 的种群，其平均基因型计算如下：

$$\bar{\boldsymbol{x}}=\frac{1}{N}\sum_{i=1}^{N}\boldsymbol{x}_i \tag{3.48}$$

如果对种群使用适应度比例选择策略，则选择后其平均基因型变为

$$\bar{\boldsymbol{x}}^*=\frac{\sum_{i=1}^{N}\boldsymbol{x}_i f(\boldsymbol{x}_i)}{\sum_{i=1}^{N}f(\boldsymbol{x}_i)} \tag{3.49}$$

因此，选择后基因型的期望变化为

$$\Delta\bar{\boldsymbol{x}}=\frac{\sum_{i=1}^{N}\boldsymbol{x}_i f(\boldsymbol{x}_i)}{\sum_{i=1}^{N}f(\boldsymbol{x}_i)}-\frac{1}{N}\sum_{i=1}^{N}\boldsymbol{x}_i \tag{3.50}$$

所以以下表达式若为正的(负的)，则通过学习后进化就会加速(减速)：

$$\operatorname{sign}(\Delta\bar{\boldsymbol{x}}_l-\Delta\bar{\boldsymbol{x}})=\operatorname{sign}\left(\frac{\sum_{i=1}^{N}\boldsymbol{x}_i f_l(\boldsymbol{x}_i)}{\sum_{i=1}^{N}f_l(\boldsymbol{x}_i)}-\frac{\sum_{i=1}^{N}\boldsymbol{x}_i f(\boldsymbol{x}_i)}{\sum_{i=1}^{N}f(\boldsymbol{x}_i)}\right) \tag{3.51}$$

对于一个给定的固有适应度函数 $f(\boldsymbol{x})$和一个学习后得到的适应度函数 $f_l(\boldsymbol{x})$，我们可以定义以下的增益函数来判断学习是否能够加速进化：

$$g(\boldsymbol{x})=\frac{f_l(\boldsymbol{x})}{f(\boldsymbol{x})} \tag{3.52}$$

可以证明：

$$g'(\boldsymbol{x})=\begin{cases}>0\Leftrightarrow\Delta\bar{\boldsymbol{x}}_l-\Delta\bar{\boldsymbol{x}}>0 & \text{（加速）}\\ <0\Leftrightarrow\Delta\bar{\boldsymbol{x}}_l-\Delta\bar{\boldsymbol{x}}<0 & \text{（减速）}\\ =0\Leftrightarrow\Delta\bar{\boldsymbol{x}}_l-\Delta\bar{\boldsymbol{x}}=0 & \text{（无影响）}\end{cases} \tag{3.53}$$

需要注意的是，上述讨论的适应度函数是非常特殊的，而对于在求解复杂优化问题时局部搜索是否能够加速进化优化的判断是很困难的。

另外需要注意的是，在自然界中，选择压力的主要来源是生存和繁殖，而不是一种快速的进化。一个很自然的问题是在基因型和表现型级别上的自适应机制能带来什么样的好处。文献[97]中给出的一些见解表明通过利用隐藏效应，种群能够保持足够程度的基因型多样性，这对种群快速适应动态环境尤为重要。这也意味着在频繁变化的环境中鲍德温和隐藏效应可能有助于种群的进化。

3.10 分布估计算法

进化算法依赖于交叉和突变等遗传变异操作来搜索新的优秀解。然而，这些操作执行的是随机搜索，不能显式地学习到问题的结构，也不能在搜索中利用这些知识。

EDA 是一类基于种群的搜索算法，它通过对优秀候选解建立和采样显式概率模型来引导对最优解的搜索。图 3.21 给出了一般的分布估计算法示意图。可以看到，除了遗传操作被建立概率模型、从模型中采样子代个体所代替之外，EDA 的主要构成和进化算法非常类似。

因此，有效构建概率模型成为了设计 EDA 需要关注的焦点问题。接下来介绍一些广泛应用于 EDA 中的概率模型。

3.10.1 一个简单的 EDA

EDA 的基本思想可以使用所谓的 onemax 问题来解释，其中每个决策变量都是一个二进制位，目标函数为

$$\text{onemax}(x_1,x_2,\cdots,x_n)=\sum_{i=1}^{n}x_i \tag{3.54}$$

其中，n 表示决策变量的数目。该问题具有一个全局最优解，其所有决策变量值都为 1。

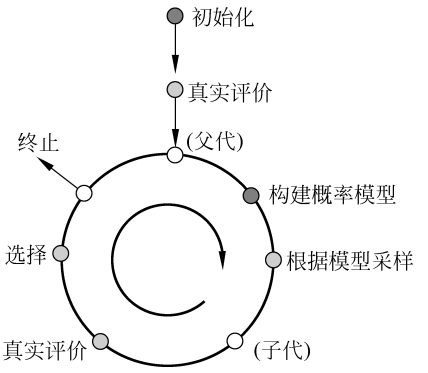


图 3.21　分布估计算法示意图

假设 onemax 有 4 个变量，在第 t 代父代种群中包含 3 个个体 $\boldsymbol{x}^1(t)=\{1,0,1,1\}$、$\boldsymbol{x}^2(t)=\{0,1,1,0\}$和 $\boldsymbol{x}^3(t)=\{0,1,1,0\}$。因此，可以基于 3 个个体计算 $x_i=1(i=1,2,3,4)$的概率，从而建立一个概率模型：

$$\begin{aligned} p_1(x_1=1)&=1/3\\ p_2(x_2=1)&=2/3\\ p_3(x_3=1)&=1\\ p_4(x_4=1)&=1/3 \end{aligned} \tag{3.55}$$

然后，可以使用上述概率对每个决策变量 $p_i(i=1,2,3,4)$采样产生 3 个子代个体，记为$\boldsymbol{x}'^1(t)=\{0,1,1,1\}$，$\boldsymbol{x}'^2(t)=\{1,0,1,0\}$和$\boldsymbol{x}'^3(t)=\{1,1,1,0\}$。随后，使用这一种选择方法来选择第 $t+1$ 代的父代个体，即 $\boldsymbol{x}^1(t+1)=\{0,1,1,1\}$、$\boldsymbol{x}^2(t+1)=\{0,1,1,1\}$和 $\boldsymbol{x}^3(t+1)=\{1,1,1,0\}$。随后，概率模型将更新如下：

$$\begin{aligned} p_1(x_1=1)&=1/3\\ p_2(x_2=1)&=1\\ p_3(x_3=1)&=1\\ p_4(x_4=1)&=2/3 \end{aligned} \tag{3.56}$$

持续执行上述过程，直到种群收敛到最优解(1,1,1,1)。由于不需要考虑变量之间的交互性，因此该算法也被称为单变量边缘分布算法(Univariate Marginal Distribution Algorithm，UMDA)。

3.10.2　求解离散优化问题的 EDA

3.10.1 节中描述的 UMDA 可以扩展用于求解一般的离散优化问题[98]。一种想法是逐步更新概率模型，称为基于种群的增量学习(Population-Based Incremental Learning，PBIL)[99]：

$$p_i=(1-\alpha)p_i+\alpha\frac{1}{N}\sum_{k=1}^{N}x_i^k,\quad i=1,2,\cdots,n \tag{3.57}$$

其中，α 是用户定义的学习率，$\boldsymbol{x}^k=\{x_1^k,x_2^k,\cdots,x_n^k\}$，$N$ 是从 $M\geqslant N$ 个子代个体中选择的下一父代个体数。

PBIL 的一种变种称为紧凑遗传算法(Compact Genetic Algorithm，CGA)[100]。以概率 $p_i=0.5(i=1,2,\cdots,n)$ 初始化概率模型。然后根据概率模型采样两个个体，朝着最优个体更新概率模型。持续执行该过程，直到概率模型收敛为止。

虽然 UMDA、PBIL 和 CGA 没有考虑决策变量之间的交互作用，但更复杂的 EDA 可以使用双变量或多元概率模型用于捕获两个或多个变量之间的交互作用。例如，扩展的紧凑遗传算法将决策变量分成若干类，然后每个类用一种概率[101]进行建模。因式分解分布算法(Factorized Distribution Algorithm，FDA)[102]采用固定的因子分解，将分布分解为条件分布和边缘分布用于求解可分离散问题。贝叶斯优化算法(Bayesian Optimization Algorithm，BOA)[103]使用贝叶斯网络对多个决策变量之间的交互建模，这些决策变量之间的交互通过节点之间的连接边表示。需要注意的是，BOA 完全不同于 5.4 节中的贝叶斯优化。

3.10.3 求解连续优化问题的 EDA

也有很多分布估计算法是针对求解连续优化问题而提出的。同样，这些模型可以分为单变量概率模型和多变量概率模型。对于规模大小为 N 的父代种群，一个单变量分解高斯模型可以描述为

$$\mu_i=\frac{1}{N}\sum_{i=1}^{N}x_i \tag{3.58}$$

$$\delta_i=\sqrt{\frac{1}{N}\sum_{i=1}^{N}(x_i-\mu_i)^2} \tag{3.59}$$

其中，N 表示选择的个体数量(父代)，随后，对每个决策变量构建一个高斯模型：

$$p_i(x_i)=\frac{1}{\delta_i\sqrt{2\pi}}\exp\left\{-\frac{(x_i-\mu_i)^2}{2(\delta_i)^2}\right\} \tag{3.60}$$

其中，μ_i 和 δ_i 表示为变量 $i=1,2,\cdots,n$ 训练的高斯模型的平均值和标准差。

一旦高斯模型 $p_i(x_i)$ 构建后，子代个体(通常超过 N 个)就可以从高斯分布中生成。经过选择后，更新概率模型，然后再重新采样新解。

然而，这样的单变量模型并不能捕获决策变量之间的相关性。为了解决这个问题，可以采用联合高斯分布模型。然而，在高维空间中建立一个完整的联合分布模型将面临维数灾难，因此在实际中是不可行的。为了缓解这一问题，可以将种群聚成几类，然后为每一类建立联合分布模型。对于第 k 个聚类，$1\leqslant k\leqslant K$，构建以下联合分布模型：

$$p^k(\boldsymbol{x})=\frac{1}{(2\pi)^{n/2}\mid\boldsymbol{\Sigma}^k\mid^{n/2}}\exp\left\{-\frac{1}{2}(\boldsymbol{x}-\boldsymbol{\Lambda}^k)^{\mathrm{T}}(\boldsymbol{\Sigma}^k)^{-1}(\boldsymbol{x}-\boldsymbol{\Lambda}^k)\right\} \tag{3.61}$$

其中，$\boldsymbol{x}$ 是 n 维设计向量，$\boldsymbol{\Lambda}^k$ 是均值的 n 维向量，$\boldsymbol{\Sigma}^k$ 由第 k 个类中的个体估计得到的 $n\times n$ 协方差矩阵。

与单变量分解模型不同，第 k 个类的概率模型如下所示：

$$p(k)=\frac{N^k}{\sum_{k=1}^{K}N^k} \tag{3.62}$$

其中，N^k 表示第 k 个类中的个体的数量。

3.10.4　多目标 EDA

原则上，所有为单目标优化而提出的 EDA 都可以通过替代选择机制扩展到多目标优化上。然而，很多多目标 EDA 的提出是用于对 Pareto 前沿面特征的建模。

一类多目标 EDA 在 $m-1$ 维流形上建立概率模型，称为规律建模多目标进化算法(Regularity Modeling Multi-objective Evolutionary Algorithm，RM-MOEA)。在适当条件下，m 目标优化问题的 Pareto 前沿面是一个 $m-1$ 维的流形[104]。为此，RM-MOEA 的概率模型由一条局部主曲线和一些高斯分布模型组成。

假设一个种群分为 K 类。在第 k 个类(标记为 C^k)中的解可以用一个 $m-1$ 维流形 M^k 的均匀分布来描述：

$$P^k(\boldsymbol{S})=\begin{cases}\dfrac{1}{V^k}, & \boldsymbol{S}\in M^k\\ 0, & \text{其他}\end{cases} \tag{3.63}$$

其中，$\boldsymbol{S}$ 是潜在空间中的一个 $m-1$ 维随机向量，V^k 是 M^k 的体积，其边界为

$$a_i^k\leqslant s_i\leqslant b_i^k,\quad i=1,2,\cdots,(m-1) \tag{3.64}$$

其中，

$$a_i^k=\min_{\boldsymbol{x}\in C^k}(\boldsymbol{x}-\bar{\boldsymbol{x}}^k)^{\mathrm{T}}U_i^k \tag{3.65}$$

$$b_i^k=\max_{\boldsymbol{x}\in C^k}(\boldsymbol{x}-\bar{\boldsymbol{x}}^k)^{\mathrm{T}}U_i^k \tag{3.66}$$

$\bar{\boldsymbol{x}}^k$ 是 C^k 中个体的平均值，U_i^k 是第 i 个类 C^k 中数据的主成分。

除了由主曲线定义的均匀分布模型外，在设计空间中还构建了 n 维零均值高斯分布：

$$N^k(\boldsymbol{x})=\frac{1}{(2\pi)^{n/2}\mid\boldsymbol{\Sigma}^k\mid^{n/2}}\exp\left\{-\frac{1}{2}\boldsymbol{x}^{\mathrm{T}}\boldsymbol{\Sigma}^k\boldsymbol{x}\right\} \tag{3.67}$$

其中，$\boldsymbol{\Sigma}^k=\delta^k\boldsymbol{I}$，$\boldsymbol{I}$ 是 $n\times n$ 维单位矩阵，δ^k 由下式计算：

$$\delta^k=\frac{1}{n-m+1}\sum_{i=m}^{n}\lambda_i^k \tag{3.68}$$

其中，λ_i^k 表示 k 类中所有解的协方差矩阵中第 i 个最大特征值。

选择类 k 的模型以进行子代个体采样的概率定义为：

$$p(k)=\frac{V^k}{\sum_{k=1}^{K}V^k} \tag{3.69}$$

其中，V^k 是 $m-1$ 维流形的体积，若是在曲线的情况下，那么它就是指曲线的长度。

采样的新解包括 3 个步骤。第一步，根据式(3.63)在 $m-1$ 维流形 M^k 上产生一个解，然后映射到 n 维决策空间上。假设 $\boldsymbol{S}$ 是在 M^k 中为第 k 个类生成的一个 $m-1$ 维随机向量，若流形 M^k 是一个一阶主曲线，那么它将以以下方式映射到 n 维决策空间中：

$$\boldsymbol{x}_1=\boldsymbol{\Theta}_0^k+\boldsymbol{\Theta}_1^k\boldsymbol{S} \tag{3.70}$$

其中，$\boldsymbol{x}_1$ 是 n 维随机向量，$\boldsymbol{\Theta}_0^k$ 是类 $C(\boldsymbol{x})^k$ 中数据的均值，$\boldsymbol{\Theta}_1^k$ 是 $n\times(m-1)$维矩阵，由 $m-1$ 个最大特征值所对应的特征向量组成。

接下来，通过式(3.67)中的高斯模型生成一个 n 维的决策向量 $\boldsymbol{x}_2$。最后，与 $\boldsymbol{x}_1$ 相加获得一个新的候选解：

$$\boldsymbol{x}=\boldsymbol{x}_1+\boldsymbol{x}_2 \tag{3.71}$$

重复上述过程以产生一个子代种群。

虽然大多数多变量 EDA 用于捕获决策变量之间的交互信息，但 RM-MOEA 能够对目标之间的关系进行建模，并已用于处理噪声评估[105]。进一步地，还可以考虑决策变量之间、目标之间以及决策变量和目标之间交互。例如，文献[106]中使用了一个多维贝叶斯网络(Multi-dimensional Bayesian Network，MBN)用于捕获上述 3 种不同的交互信息。

在 MBN 中，根据贝叶斯网络结构，在给定父代变量的不同值组合情况下，节点代表决策变量，弧线描述三变量之间的条件依赖关系，参数表示每个变量值的条件概率。图 3.22 是 MBN 的一个例子，其中顶层由 3 个节点表示，代表 3 个类变量，底层有 4 个节点表示，代表 4 个决策变量。因此，MBN 能够使用弧线(有向连接)来捕获决策变量和目标之间的相互作用，从而能够建立一个更有效的 EDA 用于解决多目标优化问题。

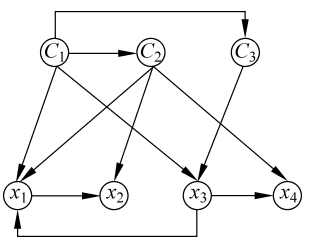


图 3.22 由 4 个决策变量和 3 个类组成的多维贝叶斯网络示例

3.11 参数自适应和算法选择

到目前为止，已介绍了 6 类最流行的基于种群进化的元启发式搜索算法，包括遗传算法、进化策略、遗传规划、ACO 算法、差分进化和 PSO 算法。这些算法在不同问题上的求解

性能表现不同。给定一个问题，目前还没有明确的理论证明应该使用哪种算法对其求解，因此，实践中若没有大量的试错，很难选择正确的优化算法；此外，大部分元启发式算法包含很多需要用户自定义的参数。接下来，将介绍用于选择和推荐元启发式搜索算法的常用做法。

3.11.1　自动参数调优

元启发式算法中参数的定义并非易事。元启发式算法中的大多数参数，如种群大小、交叉和变异概率、迭代数以及编码方法，对它们的性能有很大的影响，但目前还没有针对这些超参数设定的具体指导策略。进化策略可能是一个例外，因为它们同时也对一些重要的参数进行编码并在搜索中对其进行共同演化，例如染色体的步长。此外，进化策略还有一些确定种群大小的准则，特别是在协方差矩阵适应进化策略中的应用。

元启发式算法中的参数自适应具有挑战性，其原因包括以下几方面。首先，参数不是独立的，它们共同影响搜索的性能。其次，在算法能够获得有用知识来有效调节参数之前需要大量的迭代次数。最后，参数的最优设置通常和问题本身有关。

通常，参数既可以直接调整，也可以根据搜索动态的反馈来进行调整。例如，在搜索的早期阶段应当鼓励其进行探索，而在后期阶段则进行开采，可以通过控制参数来反映这些需求。因此，一些参数可以根据预先指定的适应机制来控制，特别是在迭代过程中改变参数。

另一个想法是根据搜索性能或算法状态（如收敛性和种群多样性）来改变参数。更多关于参数自适应的讨论可以参考文献[107]。

除了参数自适应外，进化算法的表示也可以自适应。针对二进制遗传算法表示的自适应，一个基本的想法是使用可变长的染色体代替固定长度的染色体[108]。另一种想法是使用如图 3.23 所示的混合表示法，图中开关 R 表示激活的是实值表示，而 B 表示激活的是二

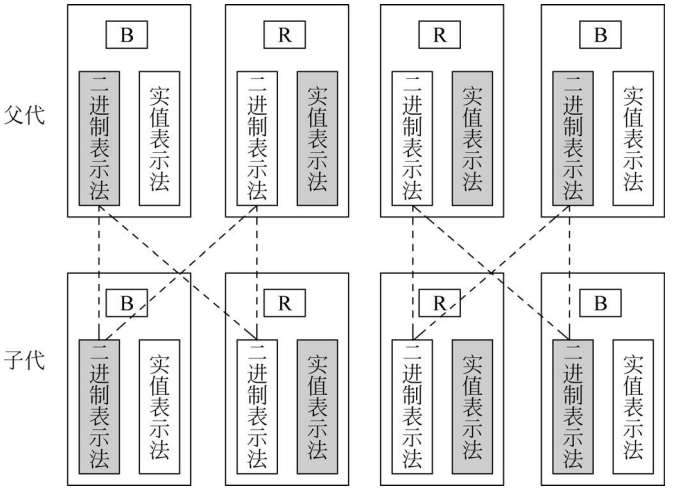


图 3.23　对每个决策变量使用二进制和实值表示法进行编码的混合表示法

进制表示，即对每个决策变量都有二进制和实值两种表示法，至于激活哪种表示法则取决于各自表示法的性能[109]。

需要注意的是，这两种表示方法需要协作使得交叉和突变后它们表示的是相同的值。

3.11.2 超启发式算法

根据无免费午餐理论，没有哪一种元启发式或数学优化算法可以在所有类型的优化问题上胜过其他算法。当然，在搜索过程中自动选择性能最好的算法是很有意义的。超启发式算法的目的是自动选择或组合启发式算法以有效求解给定的问题。元启发式算法的自动选择和生成通常借助机器学习技术和统计分析来实现。图 3.24 给出了超启发式算法的通用框架。

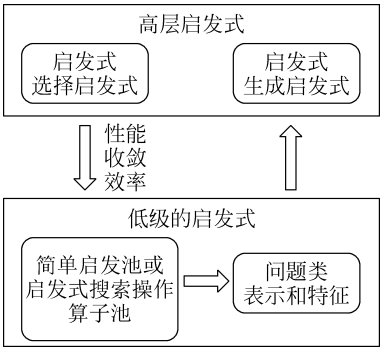


图 3.24 通用超启发式算法

超启发式可以分为在线方法和离线方法。在离线方法中，通过规则的方式提取以往求解各种问题的知识，并基于案例推理来选择算法。在在线方法中，通过在搜索过程中应用机器学习算法，如强化学习，来寻找求解给定问题最有效的搜索启发式算法。

3.11.3 适应度地形分析

适应度地形分析是理解启发式算法在求解优化问题时理解搜索性能的另一类方法。为此，我们试图研究求解优化问题的困难特征的主要特性。找出给定问题的正确特征，将其和搜索算法或搜索操作关联起来能够为选择正确的算法和搜索操作提供参考。目前已被广泛分析的适应度地形特征主要包括[110]：

(1) 模态意指适应度地形包含一个还是多个最优解。它是适应度地形的显式特征，用于刻画问题的求解难度以及决定哪些搜索操作是更有效的。

(2) 引力盆地主要定义了最小值指尖的宽度和深度。盆地分为具有强引力(无条件)和有条件(弱)引力。

(3) 崎岖性是描述最优解的另一个特征。不同于盆地，它反映了局部极小值变化的频

率或密度。

(4) 障碍定义的是从一个最小值经过任意路径到另一个最小值的最小适应度。

(5) 演化性、中立性和上位性等特征借用生物网络的概念,用于描述适应度地形相邻点之间的关系。这些与地形走势分析密切相关,可将适应度地形视为一个复杂的网络。

还有一些其他的适应度地形分析方法,包括适应度距离相关性[111]和光谱地形分析[112]。文献[113]中也提出了信息理论方法。

3.11.4　自动推荐系统

一种经验性的想法是开发推荐系统对于给定问题推荐特定的启发式算法,这也是比较可靠的想法。与超启发式算法不同,推荐系统旨在离线识别算法性能和适应度地形特征之间的关系。推荐系统类似于分类系统,它将表示优化问题难度或结构的特征作为输入,并将相对应执行最优的元启发式算法作为输出(标签)。不同的搜索算法用于求解大量的基准测试问题,并将每个问题上的获胜算法记录作为标签。利用这些数据训练分类器。为了获得一个高性能算法的推荐系统,收集足够数量的基准问题、从基准问题中抽取最重要的特征以及选择具有代表性的元启发式算法来解决这些问题是非常重要的。

尽管大多数工作是基于适应度地形分析的[114],但最近提出的一个想法是使用决策树结构来表示任意白盒或黑盒目标函数。决策树结构由一些基本的算术运算符(如加法运算符、乘法运算符),和一些基本函数(包括平方、平方根、指数和正弦等)组成[115]。对于黑盒优化问题,可以采用基于遗传规划的符号回归来预测树的表示。这样做的好处是人们可以获得任意优化问题的统一表示,并且可以通过基本的算术运算符和基本函数产生随机树,从而生成无限基准问题用于手机大量训练数据。随后,利用这些数据训练深度学习模型,使其能够预测任何给定函数的性能,从而推荐性能最好的启发式算法。图 3.25 给出了推荐算法的示意图。

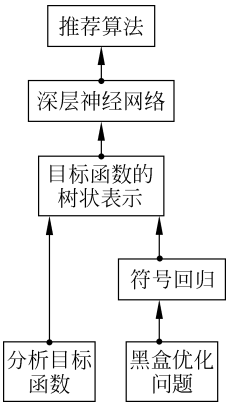


图 3.25　基于深度学习的元启发式算法推荐系统

3.12　总结

基于种群的元启发式算法在大量数学规划方法无法求解的测试问题上已展示出了它的有效性。他们还在求解黑盒实际优化问题上也获得了很大成功,如空气动力设计优化、工业过程优化以及混合动力电动汽车控制器设计。然而,进化算法的能力在很大程度上仍然需要在重点应用中来证明。为了实现这一目标,设计高效的数据驱动进化优化算法是必不可少的。