# 《人工智能》课程系列

遗传算法基础*

武汉纺织大学数学与计算机学院

杜小勤

2018/11/26

## Contents



*本系列文档属于讲义性质，仅用于学习目的。Last updated on: November 23, 2019。

# 1 概述

遗传算法属于演化计算的一个分支。演化计算是一种模拟生物进化机制的通用问题求解方法。它采用简单的编码技术来表示问题的解结构，然后将类似于生物的遗传操作直接作用于这些解形成的群体，并利用优胜劣汰的选择机制来生成新的解群体，经过若干代的进化之后，较优的解将会进化而出。对于一些问题，设计良好的演化计算算法，甚至可以找到最优解。演化计算最核心的思想是优胜劣汰-适者生存。

演化计算最初形成了三大分支，即遗传算法（Genetic Algorithm，GA）、演化规划（Evolutionary Programming，EP）、演化策略（Evolution Strategy，ES）。在 20 世纪 90 年代初，在遗传算法的基础上又发展出了另一个分支——遗传程序设计（Genetic Programming，GP）。虽然这几个分支在算法的实现方面有一些差别，但是它们都是借助于生物演化的思想与原理来进行问题求解。

在演化计算的发展过程中，还有一个出现于上世纪 80 年代的方法——Neuroevolution，即神经演化。它是将演化计算应用于神经网络结构、权值优化等方面的产物。最初出现时，它只是用于固定拓扑结构的神经网络的权值训练。后来，随着应用领域的不断拓展及需求的刺激，研究者们相继研究出了 NEAT、HyperNEAT、Novelty Search 等关键技术，神经网络的结构因而也能够不断地得到演化而变得越来越复杂与强大。彼时，这一研究领域也伴随着深度学习的崛起而大放异彩。在诸如机器人控制、游戏角色控制及游戏内容的自动生成等应用中，Neuroevolution 丝毫不逊色于深度学习技术。在 NEAT 与 HyperNEAT 的发明者 Stanley 看来，Neuroevolution 是另一种不同形式的深度学习，它是一项最接近模拟人类大脑的生物进化过程的技术。

把计算机科学与生物进化论结合起来的尝试开始于上世纪 50 年代末，但由于缺乏有效的编码方案，演化操作主要依赖于变异（Mutation）操作，而不是杂交或交叉（Crossover）操作，研究进展缓慢。到了 20 世纪 60 年代中期，John Holland 在 A. S. Fraser 和 H. J. Bremermann 等人工作的基础上，提出了位串编码技术。这种编码技术非常适合变异操作与杂交操作。另外，Holland 还强调应该将杂交

操作作为主要的遗传操作。1975 年，Holland 出版了开创性著作《Adaptation in Natural and Artificial Systems》。后来，这一方法被正式命名为遗传算法。

综上所述，遗传算法运用了生物遗传与进化概念，通过繁殖、变异、竞争等方法，实现优胜劣汰，以逐步得到问题的最优解或次优解。

## 2 基本概念

遗传算法在求解问题时，从初始的解种群（Population）开始，首先对种群中的 $N$ 个个体随机初始化，并计算每个个体的适应度函数。如果不满足优化准则，开始新一代计算：按照适应度选择个体，进行基因重组（杂交或交叉），按一定的概率进行变异，重新计算适应度，替换上一代个体。上述过程循环往复，直到满足优化准则为止。

种群（记为 $P(t)$, $t$ 表示迭代步），是由一定数目（记为 $N$）的个体（Individual）或染色体（Chromosome）组成的，记为 $x_1(t), x_2(t), ..., x_N(t)$。一般地，$P(t)$ 的数目 $N$ 在整个演化过程中是不变的。

在进行遗传迭代时，要选择当前解种群中的 2 个个体进行杂交以产生新的个体，这 2 个个体被称为新个体的父亲（Parent），产生的新个体被称为后代（Offspring）或儿子（Son）。

一般地，遗传算法首先需要对问题的解进行编码，即通过变换将 $X$ 映射到另一空间（基因空间）$I$。要求该变换 $F : X \to I$ 是可逆的，记 $F^{-1} : I \to X$ 为解码变换，以便将求解到的解进行解码，转换到问题域中。

为讨论方便，假设使用二进制对问题解进行二进制编码。注意，遗传算法不限于二进制编码，其它形式的编码也是可行的。应用时，要依据问题的性质，选择适当的编码形式对问题解进行编码。

遗传算法可以形式化如下：

$$GA = (P_0, B, I, N, l, s, g, p, f, C)$$

其中，$P_0 = \{x_1(0), x_2(0), ..., x_N(0)\}$ 表示初始种群，$x_i(0) \in I$；$B = \{0, 1\}$ 表示编码字符集；$I = B^l = \{0, 1\}^l$ 表示长度为 $l$ 的二进制串全体；$N$ 表示种群中个体的数目，$l$ 表示二进制串的长度；$s : I \to I$ 表示选择策略；$g$ 表示遗传算子：繁殖算子 $O_r : I \to I$、杂交算子 $O_c : I \times I \to I \times I$ 和变异算子 $O_m : I \to I$；$p$ 表示遗传

算子的操作概率：繁殖概率 $p_r$、杂交概率 $p_c$ 和变异概率 $p_m$；$f : I \to R^+$ 是适应度函数；$C$ 是终止条件。

在种群中，个体 $x_i \in I$，也被称为染色体。染色体的每一位被称为基因（Gene），基因的取值被称为等位基因（Allele），基因所在染色体中的位置被称为基因位（Locus）。依照生物术语，$I$ 中的个体 $x_i$ 也被称为基因型（Genetype），而 $F^{-1}(x_i)$ 被称为 $x_i$ 的表现型（Phenotype）。

# 3 基本算法

## 3.1 基本框架

不同的问题，具有不同的编码方案、选择策略和遗传算子，并且遗传算法有很多变体。但是，遗传算法的基本框架是一样的。下面给出算法的基本框架：

```
def GA0():
    Initialize the Population P randomly
    Calculate the fitness of each individual in P
    while the terminal condition does not meet:
        Generate the new population newP from P, by individual's fitness
        Calculate the fitness of each individual in newP
```

显然，上述框架只是算法的一个较为粗略的表示，后文将会给出较为详细的算法描述。

根据种群的生成方式，可以将算法分为：

- 非重叠的种群生成

  新种群会整个替换掉原来的种群；

- 重叠的种群生成

  新种群替换掉原来种群中较差的部分个体；

## 3.2 主要步骤

基本的遗传算法也称为简单的遗传算法（Simple Genetic Algorithm, SGA），包括几个方面：编码方法、个体评价函数、初始种群、群体大小、选择算子、交叉

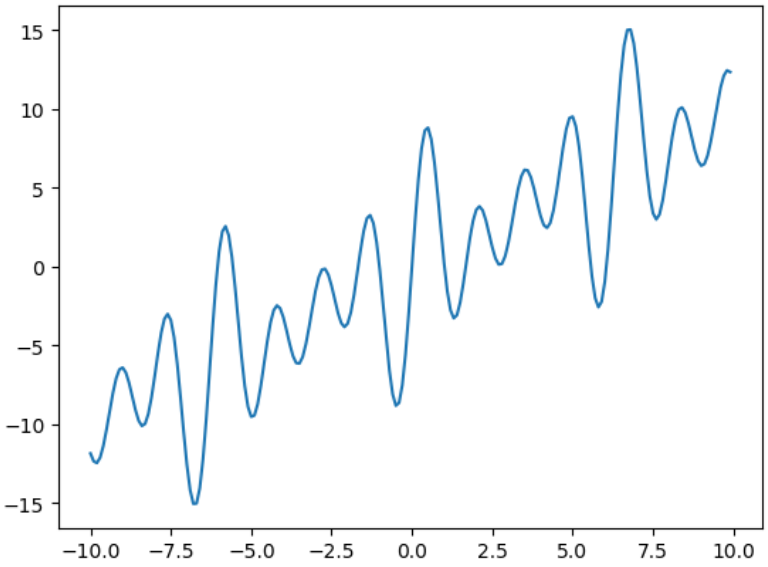


图 3-1: 函数曲线示例

算子、变异算子、算法终止条件等。编码方法采用固定长度的二进制编码，个体评价函数采用非负数，初始种群随机产生，仅使用选择、交叉和变异三种遗传算子。选择方法采用赌轮方法，交叉方法采用单点交叉，变异方法采用基本位变异，算法终止条件为指定的迭代次数或找到最优解（或次优解）。

例如，现在考虑如下优化问题：

$$\max\{f(x)|x \in X\}$$

其中，$f$ 是 $X$ 上的非负函数，即 $\forall x \in X$，$f(x) \geq 0$，$X$ 是问题的解空间。

为便于描述，设 $f(x) = x + 2\sin 2x + 3\sin 3x + 4\sin 4x$，其函数曲线如图3-1所示[1]。

如何利用遗传算法计算该函数在定义域 $x \in [0, 10]$ 上的最大值（精确到 4 位小数）?

首先需要明确，如果使用穷举法，需要执行至少 100000 次计算。下面是遗传算法需要考虑的步骤：

1. 编码与解码

[1] 曲线绘制程序参见5附录。

该问题需要对 100000 个解（数）进行编解码，这些解均匀地分布于区间 $[0,10]$ 上。如果采用二进制编码方式，因为 $2^{16} < 100000 < 2^{17}$，所以总共需要 17 位二进制编码。设 $x \in [0,10]$，$x' = (b_{16}b_{15}b_{14}...b_2b_1b_0)_2$，它们之间的关系如下：

$$x = 0 + \frac{x'}{2^{17}-1} \times 10$$

该公式将用于编解码。从生物学的角度看，$x$ 是表现型，$x'$ 是基因型；

2. 群体的生成

在确定了编解码方式后，可以随机地生成 $N$ 个个体，它们组成了遗传演化的初始群体；

3. 适应度函数

适应度函数用于确定每个个体的质量。一般而言，需要根据问题来选择合适的适应度函数。个体的适应度越高，生存的机会就越大。本例中，函数 $f(x)$ 可以作为适应度函数[2]；

4. 选择操作

这是群体的优胜劣汰操作。可用的选择策略有：基于适应值比例的策略、基于排名的策略、基于局部竞争机制的选择等。

基于适应值比例的策略有：

(a) 繁殖池（Breeding Pool）策略

繁殖池策略根据个体的适应值，首先计算每个个体的相对适应值 $rel_i = \frac{f_i}{\sum_{i=1}^{N} f_i}$，其中 $f_i$ 是群体中第 $i$ 个个体的适应度，$N$ 是群体中个体数目；然后计算每个个体的繁殖量为 $N_i = round(rel_i \times N)$，其中 $round(x)$ 表示与 $x$ 距离最小的整数；

(b) 轮盘赌（Roulette）策略

这种选择策略在实际应用中得到了广泛的使用。与繁殖池策略一样，它也是计算每个个体的相对适应值 $rel_i = \frac{f_i}{\sum_{i=1}^{N} f_i}$，但是，它直接将 $rel_i$ 作为

[2]实际上，在定义域 $[0,10]$ 内，一些 $x$ 的函数值会小于 0。针对这些 $x$，可以简单地设置其适应度为 0，因为我们的目标是搜索最大的函数值。

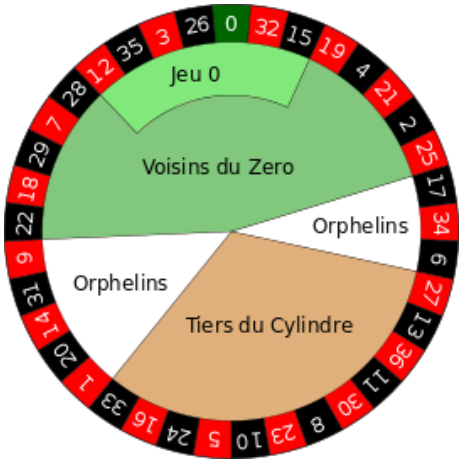


图 3-2: 轮盘赌示意图

每个个体被选择的概率。轮盘赌示意图如图3-2所示。显然，个体的适应度越大，扇区的面积越大，个体被选中的机会就越大。对于本例，可以选择轮盘赌策略；

(c) Boltzmann 策略

它使用函数 $\delta(f_i) = exp(f_i/T)$，将适应度进行变换，以改变原始的选择压力，其中 $T$ 是一个控制参数，$T$ 越大，选择压力越小，$T$ 越小，选择压力越大；

基于排名的选择策略有：线性排名选择、非线性排名选择。基于局部竞争机制的策略有：锦标赛（Tournament）策略、$(\mu, \lambda)$ 和 $\mu + \lambda$ 策略、Boltzmann 锦标赛策略等。

一般而言，无论是哪种选择策略，个体的适应度越高，生存的机会就越大；反之，生存的机会就越低。

5. 杂交或交叉（Crossover）操作

杂交操作分为点式杂交和均匀杂交。

点式杂交分为单点式杂交和多点式杂交。单点式杂交随机地在两个父串中选择一个杂交点，然后交换这 2 个父串中的子串，其中杂交点的位置可以随机设定。多点式杂交则是一次随机生成多个杂交点，然后间隔地交换 2 个父串中的子串。

均匀杂交依照概率交换两个父串中的每一位。其过程是，先随机地生成一个与父串具有相同长度的杂交模板（Template），其中 0 表示不交换，1 表示交换；然后依据杂交模板对 2 个父串进行杂交，所得的 2 个新串即为后代串。

显然，通过交叉操作，可以生成新的个体。下面比较 2 种杂交操作的优缺点：

- 点式杂交对个体的破坏性小，在搜索过程中，能以较大的概率保护好个体的模式。但是，这种策略能够搜索到的个体模式数也少。在群体规模较小时，搜索能力受到一定的影响；
- 均匀杂交对个体中的每个基因位均匀对待，破坏个体模式的可能性比较大。它能够搜索到点式杂交无法搜索到的模式。因此，在群体规模较小时，具有较强的搜索能力；

基于上述原因，当群体规模较小时，使用均匀杂交操作较合适。当群体规模较大时，群体的内在多样性本身可以弥补点式杂交的缺点，而点式杂交对个体的破坏性小，可以让好的个体模式得到更多的保护，算法收敛较快；

6. 变异（Mutation）操作

变异操作非常简单，它以一定的变异概率 $p_m$ 改变个体的基因位。在二进制编码中，它的操作是取反。在其它编码中，它以变异概率将每个基因位改变为任意合法的编码。

一般而言，交叉的概率比较大，变异的概率很低。这也符合生物进化的自然规律。

变异操作是十分微妙的遗传操作，需要与交叉操作妥善配合使用，目的是挖掘群体中个体的多样性，克服有可能限于局部解的弊病。

7. 主要控制参数的优化与选取

在遗传算法中，主要的控制参数包括：群体的规模 $N$（个体的数目）、杂交概率 $p_c$、变异概率 $p_m$。

这些控制参数对遗传算法的性能及收敛性会产生较大的影响。对于一个具体的问题，要想获得最优的参数配置，可以采取试验的方法，这一过程被称为调参。例如，针对每个参数，可以在某个区间上按一定的间隔取有限个值，

然后针对每一种组合，执行遗传算法并比较它们的性能。最后，选出最佳的参数组合。

但是，这种方法的计算量很大。在实际应用中，可以依据经验值选取控制参数，然后多执行几次以获得较优的结果。

根据经验，对于执行重叠的遗传算法，参数可以设置为：$N = 20 \sim 100$、$p_c = 0.60 \sim 0.95$、$p_m = 0.001 \sim 0.01$（或取 $1/l$，$l$ 为编码长度）；对于执行非重叠的遗传算法，参数可以设置为：$N = 20 \sim 100$、$p_c = 0.5 \sim 0.7$、$p_m = 0.2 \sim 0.4$。

## 3.3 基本算法

在明确了遗传算法的主要步骤之后，下面给出计算函数 $f(x) = x + 2\sin 2x + 3\sin 3x + 4\sin 4x$ 在定义域 $[0, 10]$ 上的最大值的算法伪代码：

```
Input:
    None
Output:
    the best solution
def GA():
    seed()
    generation_num = 500
    population_num = 20
    codebit_num = 17
    prob_crossover = 0.7
    prob_mutation = 0.02
    POPULATION = []
    FITNESS = []
    PROB = []
    Init()
    for t in range(generation_num):
        fitness_sum = 0
        P_TMP = POPULATION[:]
```

```
        for i in range(0, population_num, 2):
            d1 = Select(P_TMP)
            d2 = Select(P_TMP)
            Crossover(d1, d2)
            Mutate(d1)
            Mutate(d2)
            POPULATION[i] = d1
            POPULATION[i+1] = d2

            f1 = CalculateFitness(d1)
            FITNESS[i] = f1
            fitness_sum += f1
            f2 = CalculateFitness(d2)
            FITNESS[i+1] = f2
            fitness_sum += f2
        #Update the statistics of population
        PROB[0] = FITNESS[0]/fitness_sum
        for i in range(1, len(FITNESS)):
            PROB[i] = PROB[i-1]+FITNESS[i]/fitness_sum

    return x with MAX_FITNESS of POPULATION

    def Init():
        fitness_sum = 0
        for i in range(population_num):
            generate an individual randomly
            POPULATION.append(individual)
            f = CalculateFitness(individual)
            FITNESS.append(f)
            fitness_num += f
        PROB.append(FITNESS[0]/fitness_sum)
```

```
        for i in range(1, len(FITNESS)):
            PROB.append(PROB[i-1]+FITNESS[i]/fitness_sum)

    def Fitness(xx):
        x = 0 + xx*10/131071#pow(2,17)-1=131071
        return x + 2*sin(2*x) + 3*sin(3*x) + 4*sin(4*x)

    def Select(population):
        r = random()
        for i in range(len(PROB)):
            if r <= PROB[i]:
                return population[i][:]

    def Crossover(d1, d2):
        r = random()
        if r <= prob_crossover:
            point = randint(0, len(d1)-1)
            d1[point:], d2[point:] = d2[point:], d1[point:]

    def Mutate(d):
        for i in range(len(d)):
            if random() <= prob_mutation:
                d[i] = str((int(d[i])+1) % 2)

    def CalculateFitness(d):
        xx = int(''.join(d), base = 2)
        f = Fitness(xx)
        if f < 0:#abandon negative value
            f = 0
        return f
```

实际上，遗传算法存在诸多的变体算法，可以根据实际问题，选取合适的算法变

体。

# 4 理论基础

遗传算法是一种基于群体进化的计算模型，在这个进化过程中，包含了大量的随机操作。这些随机操作对群体性能优化起到何种作用以及它们的内在原理是什么，有必要做进一步的分析与研究。

20 世纪 70 年代，Holland 提出了基因模式理论。该理论以二进制位串为基础，深入讨论了模拟生物染色体的遗传算法的内在机制，为遗传算法奠定了理论基础。

模式（Schema）表示基因空间 $I$ 中的一些特定子集。如果我们使用 “*” 作为通配符，即它既可以表示 “0”，也可以表示 “1”，或者认为，我们对它的取值并不关心，则空间 $V = \{0, 1, *\}^l$ 表示长度为 $l$ 的所有模式全体。例如，模式 “H=*1101*” 表示集合 $\{011010, 011011, 111010, 111011\}$，其中 $l = 6$。

为了定量描述模式的性质，引入模式阶（Order）和定义距（Defining Length）这 2 个重要的概念，它们为遗传算法的模式定理分析提供了重要保障。

模式阶指的是模式 $H$ 中已有明确值的字符个数，记作 $O(H)$。例如，$O(*1101*) = 4$，$O(**101*) = 3$。显然，在模式长度 $l$ 一定时，模式阶越低，模式所表示的个体就越多，模式的概括性就越强，模式的确定性就越低；反之，模式阶越高，模式所表示的个体就越少，模式的概括性就越弱，模式的确定性就越高。因此，模式阶反映了模式的确定性与概括性。

模式定义距指的是模式 $H$ 中第 1 个和最后 1 个具有明确值的字符之间的距离，记作 $\delta(H)$。例如，$\delta(1**01*) = 4$，$\delta(*110**) = 2$，$\delta(1*****) = 0$。在遗传算法的迭代中，模式的定义距越长，模式被破坏的可能性越大；反之，模式长度越短，模式被破坏的可能性越小。因此，模式定义距反映了模式被破坏的可能性。

模式、模式阶与模式定义距为分析遗传算子对个体及其模式的作用效果提供了一种基本方法。另外，遗传算法的模式定理及内在并行性都是基于模式概念的，这些相关结论统称为模式理论。

在遗传进化的过程中，对个体的遗传操作实际上就是对个体模式的操作，不同的模式在群体进化中不断发生着改变。模式理论就是对模式及运行规律进行研究与分析的理论。

## 4.1 选择算子作用下的模式

假设在遗传迭代的过程中，群体中的个体依其适应度大小被选择。设第 $t$ 代群体为 $P_t$，该群体中模式 $H$ 出现的次数为 $N_H^t$，那么，以第 $t$ 代群体 $P_t$ 为基础进行选择操作后，得到了第 $t+1$ 代群体 $P_{t+1}$。于是，出现模式 H 的次数为：

$$N_H^{t+1} = N_H^t \cdot N \cdot \frac{f_H^t}{\sum f_j^t} \tag{1}$$

其中，$N$ 为群体的个数，$f_H^t$ 为模式 $H$ 代表的所有个体的平均适应度。

利用公式 $\overline{f^t} = \frac{1}{N} \cdot \sum f_j^t$，公式 (1) 可以写成：

$$N_H^{t+1} = N_H^t \cdot \frac{f_H^t}{\overline{f^t}} \tag{2}$$

其中，$\overline{f^t} = \frac{1}{N} \cdot \sum f_j^t$ 表示群体的平均适应度。

公式 (2) 表明，当模式 $H$ 的平均适应度大于群体的平均适应度时，模式 $H$ 的个数将增加，反之，模式 $H$ 的个数将减少。自然界生物遗传的优胜劣汰机制，在模式个数增长的关系中得到了充分的体现。遗传算法不同于一般的随机搜索方法，群体进化是向优良基因模式和适应度高的个体逼近的过程。

假设在第 $t$ 代，群体中某一特定模式 $H$ 的平均适应度为 $f_H^t = (1+c)\overline{f^t}$，则在第 $t+1$ 代，它的个体数量将是 $N_H^{t+1} = (1+c) \cdot N_H^t$ 。假设上述适应度关系一直维持，那么随着群体的进化，经过 $T$ 代之后，它的个体数量将是：

$$N_H^{t+T} = (1+c)^T \cdot N_H^t \tag{3}$$

上式表明，模式 $H$ 的数目将按照指数规律变化。

许多不同的模式将按照上述规律相应地增加或减少：优良个体得到较多的选择和复制机会，劣质个体逐渐减少。整个群体在此过程中没有出现新的个体。选择与复制不会搜索新的相似点，即没有搜索问题空间的新区域，而交叉与变异是产生新个体或新搜索区域的主要遗传算子。

## 4.2 单点交叉算子作用下的模式

模式 $H$ 只有当交叉点位于定义距之外才能生存，在单点交叉的情况下，$H$ 遭破坏的概率为 $\frac{\delta(H)}{l-1}$，其中，$l$ 是染色体或个体编码的位数，那么 $H$ 的生存概率为 $1-\frac{\delta(H)}{l-1}$。

当考虑交叉概率 $p_c$ 时，上述 2 个量分别为 $p_c \cdot \frac{\delta(H)}{l-1}$ 和 $1 - p_c \cdot \frac{\delta(H)}{l-1}$。

还要考虑到一个因素：即使交叉发生在定义距内，模式 $H$ 也不一定被破坏，这是因为其配偶可能在相同的基因位上有相同的基因，因此，模式 $H$ 遭破坏的概率 $\leq p_c \cdot \frac{\delta(H)}{l-1}$，那么模式 $H$ 的生存概率 $\geq 1 - p_c \cdot \frac{\delta(H)}{l-1}$。

显然，从上述公式可以看出，具有短定义距的模式更容易生存。

## 4.3 选择与单点交叉共同作用下的模式

如果同时考虑选择与单点交叉算子，那么可以得到：

$$N_H^{t+1} \geq N_H^t \cdot \frac{f_H^t}{\overline{f^t}} \cdot [1 - p_c \cdot \frac{\delta(H)}{l-1}] \tag{4}$$

上式表明，那些在种群平均适应度之上且定义距又短的模式将更易生存。

## 4.4 变异算子作用下的模式

变异操作以概率 $p_m$ 随机地改变一个字符串编码位，每一位的存活概率是 $1-p_m$。

对于模式 $H$，其阶次为 $O(H)$，该模式的存活概率是 $(1-p_m)^{O(H)}$。一般情况下，变异概率 $p_m \ll 1$，那么有 $(1-p_m)^{O(H)} \approx 1-O(H)p_m$。

## 4.5 选择、单点交叉及变异共同作用下的模式

在上述三种算子的共同作用下，有：

$$\begin{aligned} N_H^{t+1} &\geq N_H^t \cdot \frac{f_H^t}{\overline{f^t}} \cdot [1 - p_c \cdot \frac{\delta(H)}{l-1}][1 - O(H) \cdot p_m] \\ &\approx N_H^t \cdot \frac{f_H^t}{\overline{f^t}} \cdot [1 - p_c \cdot \frac{\delta(H)}{l-1} - O(H) \cdot p_m] \end{aligned} \tag{5}$$

## 4.6 模式定理与积木块假设

实际上，公式 (5) 就是模式定理（Schema Theorem）。由模式定理，可以得出一个重要的推论：如果模式的定义距较短、阶次较低、适应度大于群体的平均适应度，那么随着群体的进化，该模式在群体中出现的次数将按指数规律增长。

模式定理是遗传算法的理论基础，深刻地揭示了遗传算法中优胜劣汰的内在原因，为解释遗传算法的机理提供了一种有力的分析工具。

由模式定理可知，具有某种结构特征的模式在群体进化的过程中，其样本数将按指数级增长。这些结构特征是：模式阶较低、模式定义距较短、模式的平均适应度高于群体的平均适应度，我们把这类模式称为基因块或积木块（Building Block）。

积木块在遗传算法中有非常重要的作用，它们在遗传操作下相互结合，进而能够产生长定义距、高阶及平均适应度更高的模式，并最终能够找到接近全局最优的解。这就是积木块假设所揭示的内容。

虽然模式定理可以在一定程度上解释遗传算法的有效性，但是具有以下几个缺点：

- 模式定理只对二进制编码有效，其它的编码方案尚没有相应的结论成立；
- 模式定理只是提供了模式 $H$ 个数的下限，并不能据此推断出算法的收敛性；
- 模式定理对算法的设计（例如控制参数的选择）并没有提供理论指导；

# 5 附录

函数曲线的 Python 绘制程序如下：

```
import numpy as np
import matplotlib.pyplot as plt
x = np.arange(-10,10,0.1)
y=x+2*np.sin(2*x)+3*np.sin(3*x)+4*np.sin(4*x)
plt.plot(x,y)
plt.show()
```

# 6 练习

1. 编写遗传程序，计算3.2节函数 $f(x) = x + 2\sin 2x + 3\sin 3x + 4\sin 4x$ 在定义域 $[0, 10]$ 上的最大值；
2. 实现 TicTacToe 的遗传算法对弈程序；

# 7 参考文献

1. Stuart J. Russell, Peter Norvig, 殷建平等译。《人工智能：一种现代的方法》，第 3 版，清华大学出版社。

2. 潘正君，康立山，陈毓屏。演化计算，清华大学出版社，广西科学技术出版社，1998 年 7 月第 1 版。

3. Neuroevolution: A different kind of deep learning.

4. 夏定纯，徐涛。《计算智能》，科学出版社，2008 年。

5. 遗传算法与其 python 实现。

6. Wikipedia: Roulette.

7. Wikipedia: Genetic algorithm.

8. Chapter 4: Genetic Algorithm.