Dong Wang

现代机器学习技术导论

2018 年 1 月 11 日

Springer

Contents

Chapter 1
机器学习概述

Chapter 2
线性模型

Chapter 3
神经模型

Chapter 4
深度学习

Chapter 5
核方法

Chapter 6
图模型

Chapter 7
非监督学习

Chapter 8
非参数模型

Chapter 9
演化学习

我们已经讨论了各种模型，这些模型假设某种学习结构，用参数或非参数的方法来表达这种结构，再基于某种优化方法对结构进行优化。对这些结构（包括结构本身及其参数）进行优化时，除了少数有闭式解的情况外，多数采用类似‘爬山’的优化策略，即基于当前结构，依任务的性质（如目标函数的几何性质，任务本身的限制条件等）计算出下一个更优的结构，如此迭代进行，逐渐逼近最优结构。我们前面提到的EM算法、变分法、MCMC、SGD，以及下一章要介绍的TD算法等都属于这种方法。这些方法都会充分利用当前解的信息，并依此信息‘直接’求解更好的解。

这一直接优化方法的好处是直接高效，但存在若干缺点。首先，对绝大部分问题我们能看到的信息只是局部的，因此很多时候只能得到局部最优解；另一方面，一些问题比较复杂，如何有效利用局部信息并不明确。例如SGD需要计算梯度信息，但一些任务的目标函数是不连续的，这时梯度无法计算。再如图模型的变分法中，很多复杂模型求期望函数本身就是很困难的事，也就无法完成迭代优化。

演化学习（Evolutional Learning, EL）提供了另一种学习方法，这种学习方法不是直接计算下一步的优化解，而是通过随机生成一些可能的解，对这些解进行优化选择，如此迭代到得到满意的解为止。这种‘Try-and-Error’的学习方法可称为‘采样法’，传统直接优化方法可称为‘推理法’。和传统方法相比，采样法简单直观，事实上是生物界进化的基础动力。因此，EA经常被一些人工智能研究者认为是实现机器智能的普适方法。本章将讨论两种主要演化学习方法：遗传算法（GA）和遗传编程（GP）。同时，EA方法中的两个主要成份：群体学习和随机优化也被研究者单独借鉴，独立发展成两种基于采样的优化方法。本章将讨论若干有代表性的群体学习方法和随机优化方法。

9.1 基于采样的优化方法

对机器学习而言，基于先验知识设计合理的模型很多时候是很直观的，但对模型进行优化则是相当困难的问题。传统方法基于观测数据直接计算模型参数的优化值，这一方法可称为‘推理优化法’。这一方法对简单模型是有效的，但在实际应用中，很多模型可能非常复杂，直接计算优化参数往往是不可行的。‘采样优化法’通过随机生成新解，并对生成的解进行优化选择，不受模型复杂度的限制，因而可在任何优化任务中普适应用。采样优化法最初受生物进化理论启发被提出，发展成后面的演化学习方法；其后，研究者将演化学习中的群体学习和随机优化两种思路进一步拓展，发展成更广泛的采样优化理论。

9.1.1 演化学习

科学家们从人工智能早期就开始关注模型优化问题，特别是生物进化过程给了他们很大启发。生物基因系统是如此精密复杂，不可能有一种推理优化方式实现对一系统的适应与学习。达尔文告诉我们，这一复杂系统的学习方式远比我们想象的要简单：仅仅通过不段尝试新的基因组合方式并从中选择适应性更强的基因，通过长时间的世代演化之后，即可实现非常强大的基因学习和物种进化 [13]。人工智能的先驱者将这种学习方式引入到机器学习中，称为演化学习（EL）。这一学习方法和我们以前介绍过的学习方法完全不同：以前的学习方式是一种推理优化，是一种由果推因的反向学习，而生物进化所执行的是一种采样学习，通过模拟和选择来实现随机的、整体的优化，事实上是一种正向学习。

具体而言，演化学习模拟生物基因进化过程来对模型或过程进行优化。这里的模型可以是简单的线性模型，复杂的神经网络，或极度复杂的混合模型；同样，这里的过程可以是简单的邮路问题，也可以是复杂的芯片制造流程。EL将每一个候选解法或过程视为种群中的一个个体，从一个随机种群出发，选择优质个体（解法或过程）进行繁衍生成一代种群。通过若干代的演化，种群的整体质量会逐渐提高，直到发现让人满意的解法或过程。

事实上，演化学习从一开始就受到机器学习研究者的重视，甚至被认为是实现智能机器的最终方式。例如，图灵在1948年的著作《Intelligent Machinery》一文中就提到 [63]：

There is the genetical or evolutionary search by which a combination of genes is looked for, the criterion being the survival.

这实上是最早的演化学习思想。1950年，图灵进一步指出 [64]：

We cannot expect to find a good child-machine at the first attempt. One must experiment with teaching one such machine and see how well it learns. One can then try another and see if it is better or worse.

人工智能的先驱们对演化学习的钟爱不仅源于这一方法的简单直观，更源于他们对人工智能的哲学思考：通用的智能必须基于某种简单而普适的学习的方法，否则只能是特定领域的个案学习，最终会流于编程技巧。演化学习正是这样一种通用的方法，它效率不高，但只要给以足够耐心，就可以解决任何可定义的学习任务。这正是进化仿生学派的基本思路 [21]。

演化学习早在50年代就发展起来 [10, 5]，这些早期工作多集中在用计算机对生物进化过程进行模拟 [24, 12, 20]。60年代Rechenberg 提出进化策略（Evolutionary Strategy），通过个体变异操作寻找优化解。Rechenberg的工作使得演化学习开始受到重视。同样在60年代，Fogel提出进化编程方法（Evolutionary Programming, EP），通过对有限状态自动机（FSM）参数的进化遗传学习得到更好的预测模型 [22]。70年代，John Holland 及其学生对演化算法做了形式化整理并进行了一系列理论探讨 [33]，使得演化学习得以成形。

传统的演化学习可以认为是对解空间的随机搜索方法，例如解一个非常复杂的拟合问题，将模型的所有参数的所有取值作为解空间，在这一解空间中搜索最优点。这一方法通常称为遗传算法（Genetic Algorithm, GA） [33]。进一步，可以将任何一个计算机程序作为优化目标，在所有可能的程序空间寻找最优解，这一方法通常称为遗传编程（Genetic Programming, GP） [44]。

9.1.2 群体学习与随机优化

演化学习包含两个主要成分：群体学习和随机优化。群体学习通过个体之间的交互共享知识，保证总体学习方向的稳定；随机优化可以创造新的个体，帮助学习过程摆脱局部最优。这两种学习方法具有天然联系：在群体学习中知识共享一般是随机的，个体只会部分服从群体目标；随机优化经常需要利用群体信息以保证随机过程的合理性。值得说明的是，这里讨论的群体学习和随机优化都基于采样优化，即对个体进行随机生成和优化选择。

典型的群体学习方法包括峰群算法（Artificial Bee Colony Algorithm）[36]、粒子群优化算法（Particle Swarm Optimization, PSO ） [38, 58]、人工蜂群算法（Artificial bee colony algorithm，ABC）等。典型的随机优化方法是模拟退火算法（Simulated Annealing, SA） [39]，也包括具有弱群体学习特征的禁忌搜索算法（Tabu Search, TS） [28]、和声搜索算法（Harmony Search，HS） [26]等。这些算法和典型的演化学习方法有所不同，一是它们并没有模拟生物进化过程，二是仅包含了演化学习的部分思想（或者群体学习或者个体变异）。然而，这些方法绝大部分都模拟了生物种群的某些行为（如觅食、捕猎），且都基于演化学习中的优化选择，因此可以认为是演化学习的近邻算法。

本章将从典型的演化学习方法–遗传算法（GA）开始讨论，介绍演化学习的基本思路，再将该方法扩展到遗传编程（GP）。之后我们将介绍一些典型的群体学习方法和随机优化方法，并比较它们的优劣。

9.2 遗传算法（GA）

遗传算法（GA）是最早发展起来的演化学习方法。这一算法模拟生物进化方式，首先随机生成一个种群，种群中每个个体代表一个目标问题的解。通过优化选择质量较高的个体，基于这些优质个体通过交叉繁衍和个体变异生成新一代种群。经过若干代演化后即可得到优化的种群，其中的最优个体即对应目标问题的优化解。GA算法通常用于常规优化方法难以解决的问题。图 9.1是一个应用GA进行复杂问题求解的著名例子：科学家用GA算法来设计2006 NASA ST5 宇宙飞船上的天线，经过一系列繁衍优化后，GA发现了如图所示的非常复杂的天线模式。

9.2.1 算法框架

GA算法包括种群初始化、个体选择、种群繁衍三个步聚，通过迭代方式逐步提高种群质量。具体描述如下：

Fig. 9.1 用GA算法设计的2016 NASA ST5宇宙飞船天线，称为‘演化天线’（Evolved Antenna）。

- 种群初始化：初始化一个种群$G_0 = \{\xi_i\}$，其中每个个体ξ_i 由一个基因代表，又称为‘染色体’（Chromosome）[1]。在GA中，一个基因对应目标任务的一个具体解法，如模型优化问题中的模型参数，路径搜索问题中的一条合法路径等。初始化方式一般是随机的，也可以是基于先验知识得到的某些近似解。基因编码方式通常是二值串，每一位代表一个基因位；也可以是浮点值、离散类别或更复杂的结构，其中的每个元素代表一个基因位。
- 个体选择：对当前种群中的个体进行评价，选择质量较高的个体进行繁衍。这一评价需要一个‘适应函数’（Fitness function）$f(\cdot)$。在生物进化中，这一函数代表个体对环境的适应性或生存能力，在优化任务中，这一函数代表某一解法的优劣，一般取任务的目标函数作为GA的适应函数。基于这一适应函数，对适应性较强的个体给以更高的选择概率以保证后代的高质量，同时对适应性较弱的个体也给以一定的概率，以保证种群的差异

[1] 严格来说，一个基因包含一串基因位，一个染色体中包含多个基因。本章中，我们对基因和染色体不做区分。

性。经过这一选择后，即得到一个中间种群G_0'。相对G_0，这一种群中包含更大比例的优质个体。

- 种群繁衍：基于中间种群G_0'中的个体生成新种群G_1。繁衍过程一般包括两种方式：交叉繁衍和个体变异。在交叉繁衍中，随机选择两个个体，这两个个体通过互相交换基因片段生成新个体。这一方式对应有性繁殖。在个体变异中，随机选择某一个体，对其基因中某些基因位做随机扰动从而生成新个体。这一方式对应无性繁殖。经过上述繁衍过程，即得到新一代种群G_1。

上述个体选择和种群繁衍过程迭代进行，在第i代种群G_i基础上选择出种群G_i'，再繁衍出第$i+1$代种群G_{i+1}。注意到在个体选择过程中质量较高的个体有更多机会被选中，因此新一代种群总是倾向于包含更高质量的个体。同时，这一选择又是随机进行的，质量不高的个体依然有一定概率被选择和繁衍，因此可以降低局部优化的风险。当上述演化过程达到指定的迭代次数或质量要求时（如两代种群最优个体质量足够接近），GA算法完成，种群中的最优个体即可作为目标任务的最优解。

上述GA过程如图 9.2所示，其中每个星号代表一个个体，星号的大小表示个体的质量（适应函数值）。从图中可以看到，那些优质（适应函数值较高）个体有更高的概率被选择和繁衍；对那些质量较低的个体，虽然概率比较低，但依然有一定机会进化到下一代。这一进化过程的结果是，下一代种群总有更多机会包括更优质的个体（图中表示为种群中的星号由小变大），同时保持种群的多样性和差异化。

9.2.2 算法细节

GA算法的基本思路非常简单，但有些实现细节可能会显著影响算法的运行特性，需要特别注意。这些细节包括：基因编码方式、初始化策略、个体选择方法、种群繁衍方式、进化结束条件等。我们对这些内容做一简述，更多信息可参考 [67]。

编码方式

基因编码方式可能是GA算法中影响最大的因素，编码方式不同直接决定交叉和变异操作的实现方式并显著影响整个算法的收敛效果。最简单的编

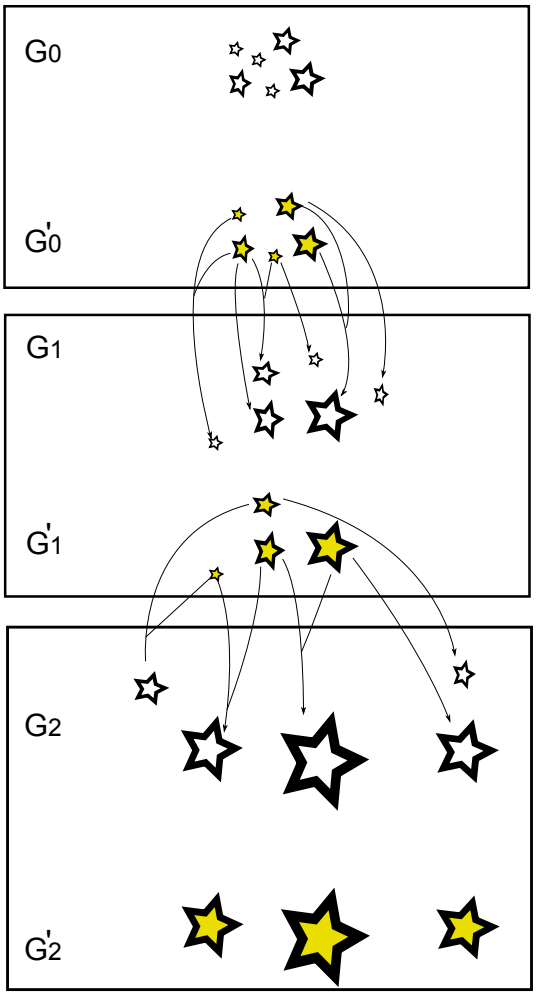

Fig. 9.2 GA算法中的个体选择与种群繁衍过程。自上而下，每个方框代表一代种群，每个星号代表一个个体，星号的大小表示个体的质量（适应函数值），带颜色的星号表示被选择的个体。在每一代种群中，较优质的个体有更大机会被选择；在生成下一代种群时，被选择的个体通过交叉繁衍和个体变异生成新个体。这一选择-繁衍机制使得那些适应性较强个体的基因有更大机会在下一代中得到继承和延续。

码方式是二进制编码（Bit string）。这一编码可用来表示类别变量，也可用来表示整数，如利用Gray编码 [32]。基于二进制编码，交叉操作可简单实现为按位交换或其它二进制操作，变异操作可表示为基于伯努利分布的按位随机采样。下一节要讨论的Holland进化理论很大程度上基于二进制串编码 [33]。

除了二进制串编码，一些遗传算法也基于浮点编码，这时需要特别设计相应的交叉和变异操作。例如，这种编码下的变异操作需要基于连续变分布（如高斯分布），而不是二进制编码中的伯努利分布。实验表明，浮点编码在一些任务上表现出比二进制编码更好的效果 [35]。

进一步，上述数值编码可扩展到任意数据结构的指针编码（如序列、堆栈、树、哈稀等）。在这些编码上的交叉和变异操作更需特殊设计，特别是结合任务需求和数据形式进行针对性设计。

上述编码方式都是对'数据'的编码，如模型参数、搜索状态等。如果我们将数据编码扩展到对操作进行编码，则得到类似二进制程序的过程编码，基于这一编码GA可以学习如何完成一个任务的过程和方式，即是我们后面要讨论的遗传编程（GP）。

个体选择策略

从当前种群G_i进行个体选择以生成中间种群G_i'也有很多种方案。最直观的想法是使得对每个个体的选择概率与个体的适应函数成正比。设第i个个体的适应函数为f_i，所有个体的平均适应函数为$\bar{f}$，则第i个个体的选择概率正比于$r_i = \frac{f_i}{\bar{f}}$。一种简单的做法如下：首先取$r_i$的整数部分$\lfloor r_i \rfloor$，将$\xi_i$复制$\lfloor r_i \rfloor$份，然后取分数部分$r_i - \lfloor r_i \rfloor$，以$r_i - \lfloor r_i \rfloor$为概率对$\xi_i$进行随机复制。例如，如果$r_i = 1.5$，则首先将$\xi_i$复制一份，再以0.5为概率对$\xi_i$进行随机复制。因此$x_i$被复制2份和3份的概率各有50%。如果$r_i = 0.5$，则$\xi_i$仅有50%的概率被复制一份，另有50%的概率不被选中。这种方法称为余量随机采样（Remainder Random Sampling）。

上述个体选择策略需要计算适应函数f_i。如果这一函数复杂度很高，则会严重影响GA的效率。一种可能的方法是对适应函数做近似。因为个体选择本身就是带有随机性的，因此不精确的适应函数值并不会对个体选择产生特别大的影响。还有一种可能的方法是不计算f_i的具体值，只要得到个体的适应性排序，即可给每个个体赋予一定的选择概率，实现合理的个体选择。还有一种情况是有些个体的适应性并不直观，必须经过一个复杂的生成过程才能得到。如人类的繁衍，一个初生婴儿的适应性是不容易判断的，需

要一个复杂的成长过程才能看到他的真正能力，这一过程可能充满了很强的随机性。这时需要一个模拟过程将基因发育成个体，由此来判断每个个体的适应函数值。上述办法虽然都可以一定程度上解决适应函数的计算问题，但GA算法一般仅适用于适应函数比较简单的场景，如果这一条件得不到满足，可以需要考虑用其它优化方法或建模方式。

最后，我们强调个体选择不仅要考虑个体适应函数的值，还需考虑任务的特殊限制，如种群大小、个体相似性、基因编码限制等。因此在设计GA算法时，需要根据具体任务对选择策略做灵活设计。

繁衍策略

交叉和变异是最常用两种繁衍策略。交叉策略是两个基因互换片段，形成新个体。最简单的交叉策略是单点交叉，即对两个待交叉的父本基因随机选择一个交叉点（注意两个父本基因长度相等，因此该交叉点对这两个基因是等位交差点），在交叉点处对两个父本基因切分，并进行基因片段互换，如图 9.3 （a）所示。稍复杂一些的交叉策略是两点交叉。这种交叉方法随机选择两个交叉点，取这两个交叉点间的基因片段互相交换，如图 9.3（b）所示。其它更为复杂的交叉策略也偶尔用到，如多点交叉或对全局交叉（Universal Crossover，每一位独立随机互换）等，但这些方法并不普遍。

变异策略一般对每一个基因位独立随机进行。对于二进制编码来说，这相当于对每一位取值进行随机反转，对浮点编码来说，可能需要引入高斯随机扰动。更复杂的变异策略很少用到，例如可以先随机确定某一个可变异的基因片段，再对该片段中各基因位随机变换。一些研究表明，交叉策略可能并不必要，在很多任务中仅有变异策略也可以得到很好的结果 [21]。

如前所述，这些策略的具体实现方式与基因编码方式和具体任务相关。不论是交叉还是变异，都是基于个体选择得到的中间种群，因此繁衍得到的下一代通常有更好的适应性。注意，这两种策略本身都具有随机性：在交叉策略中，在两个基因的什么位置进行交叉，交叉多少都是随机的；在变异中，由原基因生成新基因本身即是一个随机函数。控制这两种操作的随机性非常重要：过强的随机性会导致当前优质个体的丢失，过小的随机性会减弱种群的个体差异性。

除了交叉和变异，还有一些可能用到的繁衍策略。例如，可以将个体分组，每一组代表一个地域性种群，除了在地域性种群内部进行交叉和变异，还允许不同组间的‘移民’（类似不同种族间的通婚），以提高整个种群的丰富

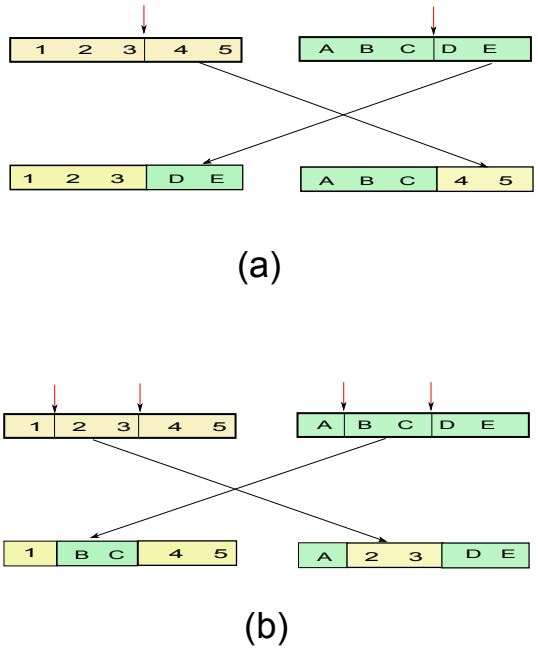

Fig. 9.3 交叉策略。（a）单点交叉策略：随机选择一个交叉点（红色箭头），从交叉点处切段两个父本染色体，互换基因片段得到两个新染色体。（b）两点交叉策略：在两个父本染色体中用两个交叉点确定一个基因片段，交换这一基因片段，得到新染色体。

性。同时，新一代个体可以合并或重新分组，部分地域种群可以消亡。这些操作将基因层的操作扩展到种群层，可减小因近亲繁殖导致的整个种群质量下降 [1]。

有些GA算法为了保证前一代的优质个体不因交叉和变异过程中的随机性而丢失，在繁衍过程中强制将上一代若干最好的个体复制到下一代。这种方法称为精英策略（Elitism） [3]。

最后，GA只是生物进化的模拟，但未必完全符合生物进化原则。一些繁衍策略虽然未必完全符合生物原则，但依然可以在GA算法中得到好效果。例如我们可以允许多个父本基因共同交叉生成一个后代，这在生物界显然是不可能的，但在GA中却可表现出不错的性能 [17]。

结束条件

和传统基于推理的迭代优化过程（如EM、变分）不同，GA的收敛性是无法保证的，因此需要设定一些结束条件来判断进化过程是否需要结束。如何设计结束条件也与任务相关，一些常见的结束条件包括：

- 进化已经达到指定的最大迭代次数；
- 当前种群中已经发现了让人满意的解；
- 最近几代种群中没有发现更好的解；
- 达到其它任务相关的限制，如运行时间。

依上述结束条件完成的GA过程并不能保证得到优化解，因此通常要对GA运行若干次以观察得到的解是否可靠。如果多次运行得到类似的结果，则我们有更大把握得到了较优化的解。注意GA一般用于求解形式复杂的问题，容易产生过训练，因此要用开发集控制生成解的质量，或用假设检验方法检测解的可扩展性。

参数调节

和大多数机器学习方法一样，GA依赖合理的参数调节，包括（1）种群的大小；（2）个体选择的概率形式；（3）交叉和变异策略的随机性；（4）最大迭代次数。依目标任务的性质和适应函数的形式，这些参数对算法表现的影响程度也各不相同。通常可以基于一个开发集来确定这些参数。因为GA算法本身的随机性，可能需要多次运行GA过程来确认某一参数的影响。同时，研究者还提出一些自适应参数方法。例如，利用当前种群的适应性或聚类信息可以调节交叉和变异策略的概率。这一方法称为自适应GA（Adaptive GA） [61]。

9.2.3 进化理论

直观上，GA每次生成新种群时都会选择较高质量的个体参与繁衍（交叉和变异），同时可引入一定限制条件（如精英策略）来保证下一代种群的质量，因此可以想象GA的演化过程会逐步提高种群的质量。然而，在理论上说明GA的有效性并不容易。研究者在这方面进行了一系列探索，发现在某些限定场景下（较简单的基因表示，较简单的繁衍策略，较简单的适应函

数等），是可以建立起表达GA过程的理论模型的。本章以Holland的超平面随机采样（Hyperplane Sampling）理论为例，说明GA的有效性。

超平面随机采样理论将搜索空间分成多个不同阶的超平面，每个超平面包含若干当前种群中的个体（可称为代表样本），将这些个体的适应函数值做平均，即得到该超平面的适应函数。可以证明，GA中的个体选择过程事实上是以一定数概率对所有超平面进行同步采样的过程，其中每个超平面获得样本点的概率正比于该超平面的适应函数。即某个超平面的适应函数平均值越大，该超平面在中间种群中的代表样本越多。交叉和变异操作在这一采样的基础上调整采样概率，在提高样本多样性的同时，保证在每个超平面上其采样概率与适应函数的正比关系不会偏离过远。因此，GA的迭代过程使得采样向适应函数高的超平面聚拢。

以三维搜索空间为例，设基因的二进制编码为$[b_1b_2b_3]$，则搜索空间可表示为三维空间中的一个立方体，如图 9.4 所示。如果我们允许该编码中一位或两位可取其任意值并用通配符$*$表示，则该编码可代表一个样本集合，这些样本分布在一个超平面上。超平面的阶数由编码中非通配符的个数确定，含有一个非通配符的编码代表一阶超平面，含有n个非通配符的编码代表一个n阶超平面。例如$[1*0]$是一个二阶超平面，在图 9.4中表示为蓝色边；$[**1]$为一个一阶超平面，在图 9.4中表示为桔色平面。很容易看到，对一个长为L的二进制编码，可以定义3^L-1个超平面，因为每一位都有$0,1,*$三种选择，而$[***]$作为全空间不计入超平面中。

在一个高维空间上进行搜索有组合爆炸问题，GA算法用种群中的个体对搜索空间中的超平面进行采样。由于每个个体都属于2^L-1 个超平面（每一位或保留原值，或被$*$代替，且不记所有位为$*$的全空间特例），因此对任意一个个体计算其适应函数时，相当于对2^L-1个超平面同时加入了一个采样值。如果一个超平面H的适应函数为$\bar{f}_H$，而该超平面内的采样数为c_H，则我们希望经过个体选择后，H 内的采样应向$\bar{f}_H$较大的方向调整。如果我们采用余量随机采样策略，则可发现中间种群内每个超平面H的采样个数C'_H将接近$\bar{f}_H \times c_H$。图 9.5给出一个例子。由图中可知，中间种群中每个超平面上的采样数与基于适应函数的期望采样数非常接近。值得注意的是，余量随机采样策略以每一个个体的适应函数值为基础进行个体复制和随机生成，并没有考虑到每个超平面的采样情况，但这一策略确实使得经过个体选择后所有超平面上的样本数与原种群中该超平面的适应函数值成正比。这意味着计算每个个体适应函数的时候，即相当于同时隐性计算了所有超平面的适应函数值。

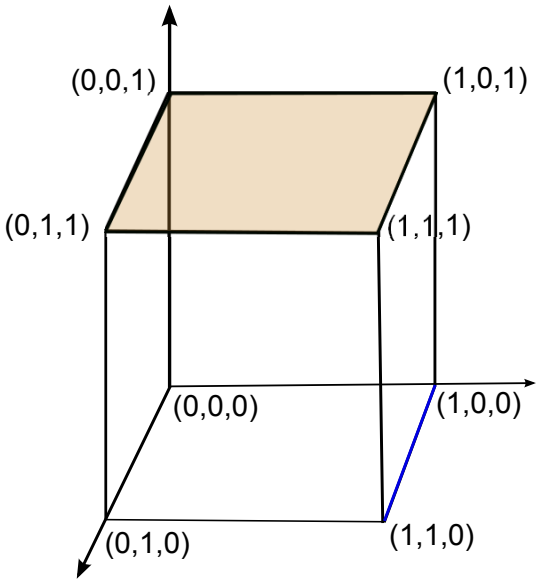

Fig. 9.4 三维空间中的二进制编码和超平面。立方体的每个顶点代表一个个体的基因编码，每条线段为一个二阶超平面，每个平面为一个一阶超平面。例如，蓝色边表示以$[1*0]$为编码的二阶超平面，桔色面表示以$[**1]$为编码的一阶超平面。

Schemata and Fitness Values									
Schema	Mean	Count	Expect	Obs	Schema	Mean	Count	Expect	Obs
101*...*	1.70	2	3.4	3	*0**...*	0.991	11	10.9	9
111*...*	1.70	2	3.4	4	00**...*	0.967	6	5.8	4
1*1*...*	1.70	4	6.8	7	0***...*	0.933	12	11.2	10
01...*	1.38	5	6.9	6	011*...*	0.900	3	2.7	4
**1*...*	1.30	10	13.0	14	010*...*	0.900	3	2.7	2
11...*	1.22	5	6.1	8	01**...*	0.900	6	5.4	6
11**...*	1.175	4	4.7	6	0*0*...*	0.833	6	5.0	3
001*...*	1.166	3	3.5	3	*10*...*	0.800	5	4.0	4
1***...*	1.089	9	9.8	11	000*...*	0.767	3	2.3	1
0*1*...*	1.033	6	6.2	7	**0*...*	0.727	11	8.0	7
10**...*	1.020	5	5.1	5	*00*...*	0.667	6	4.0	3
*1**...*	1.010	10	10.1	12	110*...*	0.650	2	1.3	2
****...*	1.000	21	21.0	21	1*0*...*	0.600	5	3.0	4
					100*...*	0.566	3	1.70	2

Fig. 9.5 每个超平面（由Schema表示）的的适应函数（Mean）、当前种群采样数（Count）、期望采样数（Expect）和基于余量随机采样策略生成的中间种群中该超平面上的实际采样数。从表中可知，这些变量间具有如下关系：期望采样数=适应函数× 当前种群采样数。图片来自 [4]。

上述结论可形式化如下：设第t代种群中超平面H 上的采样数为$M(H,t)$，以$t+inter$为经过个体选择后的中间种群。依余量采样策略的个体复制方式，可知：

$$M(t+inter)=M(H,t)\frac{f(H,t)}{\bar{f}}, \tag{9.1}$$

其中$f(H,t)$为第t代种群中超平面H 上包含的所有采样的适应函数平均值，$\bar{f}$为所有样本的平均值。

值得注意的是，个体选择仅是当前种群中个体的复制。这一复制机制可能会改变每个超平面内采样的分布，但不会产生新采样。为产生新采样，需要繁衍策略。我们首先考虑交叉繁衍。为简便起见，仅考虑单点交叉策略，即随机选择一个交叉点对两个父本基因在该交叉点处同时切断，再互相交换基因片段重新组合，如图 9.3 （a）所示。

我们考虑某一个超平面H，考察单点交叉策略如何影响在该超平面上的采样。特别是，当执行完交叉策略后，如果产生成的个体不在该超平面上，则将对该超平面产生'破坏'，即降低下一代种群在该超平面上的概率。显然，对于一阶超平面，任何交叉策略都不会对该超平面上的分布产生影响，因为交叉后生成的个体里总有一个保持在原空间。例如一阶超平面$[1****]$，不论采用何种交叉策略，以该超平中某一样本作为其中一个父样本，其生成的子样本里总有一个在第一位是1，不论另一个父样本是否属于同一个超平面。

对于2阶以上的超平面，交叉策略会使得生成的子样本脱离该超平面，因而对超平面产生破坏。特别的，即使对同阶超平面，交叉策略产生的破坏也是不同的。例如如下两个超平面:

$$[11******] \qquad [1******1],$$

对于第一个超平面，如果交叉点选在第2个1之后，则产生的子样本总会有一个在该超平面上，而对于第二个超平面，任何一个交叉点都可能产生两个不属于该超平面的子本样。为描述这种不同模式的超平面在交叉策略下的破坏作用，我们定义一个超平面的相隔距离（Defining Length）$\Delta(H)$为H的编码中第一个非$*$ 位和最后一个非$*$位的距离，例如，$[*1**0*]$的相隔距离为3，$[**1****0***1]$的相隔距离为9。交叉策略产生的破坏与$\Delta(H)$有正比关系，即：

$$\hbar=\frac{\Delta(H)}{L-1},$$

$\hbar$代表产生的破坏，L为基因的编码长度。注意到在交叉操作中，一个父样本由H中产生，因此只有当另一个父样本由该超平面的补集中产生时，才会产生破坏，因此有：

$$\hbar = \frac{\Delta(H)}{L-1}(1-P'(H,t))$$

其中$P'(H,t)$为在中间种群中超平面H 中采样点所占的比例。依式 9.1，可知：

$$P'(H,t) = P(H,t)\frac{f(H,t)}{\bar{f}},$$

其中$P(H,t)$是在第t代种群中超平面H中采样点的比例，因而有：

$$\hbar = \frac{\Delta(H)}{L-1}(1-P(H,t)\frac{f(H,t)}{\bar{f}}). \tag{9.2}$$

设种群中样本参与交叉操作的概率为p_c，对不参与交叉操作的个体进行直接复制，则有第$t+1$代种群中超平面H中采样个数为：

$$M(H,t+1) \geq (1-p_c)M(H,t)\frac{f(H,t)}{\bar{f}} + p_c[M(H,t)\frac{f(H,t)}{\bar{f}}(1-\hbar)] \tag{9.3}$$

上式之所以是一个不等式，是因为前面讨论的交叉操作带来的破坏量事实上是一个上界。某些看起来可能带来破坏的操作事实上有一定比例不会带来破坏。例如对超平面[11***]，当该平面中的样本与[10000]做交叉操作且交叉点选在第一位之后时，所生成的后代依然有一个在原超平面内。将式 9.2代入式 9.3，并做整理，可得：

$$P(H,t+1) \geq P(H,t)\frac{f(H,t)}{\bar{f}}[1-p_c\frac{\Delta(H)}{L-1}(1-P(H,t)\frac{f(H,t)}{\bar{f}})]$$

上式称为基于单点交叉的Schema定理。

现在可以引入变异操作。设每一位上变异的概率为p_m，并记超平面H的阶数（即编码中非$*$的个数）为$o(H)$。首先注意到对H中任何非$*$字符（0或1）的变异都会导致破坏，因此不被破坏的概率为$(1-p_m)^{o(H)}$。由于交叉和变异是串行的，只有两者都不产生破坏时才能保证繁衍过程不产生破坏,因此两者的概率是乘积关系。故而得到如下基于单点交叉和变异的Schema定理：

$$P(H,t+1) \geq P(H,t)\frac{f(H,t)}{\bar{f}}[1-p_c\frac{\Delta(H)}{L-1}(1-P(H,t)\frac{f(H,t)}{\bar{f}})](1-p_m)^{o(H)}.$$

通过Schema定义的推导过程我们看到，GA算法通过个体选择来调整在各个超平面的采样比例，使得采样向适应函数更高的超平面聚集；同时，交叉和变异操作在提供新样本的同时，保持任意一个超平面中所包含的采样比例满足一个下界。这一过程也可以理解为一种'Divide and Conquer' 方法：那些低阶的、低相隔距离的、具有较高适应函数值的超平面被越来越多地采样、繁衍，并成为构造高阶超平面的模块（称为Building Block）。因为具有较高的适应函数值，GA的采样过程保证这些模块被更好地代表；由于较低的相隔距离，GA的交叉繁衍过程不易对其产生破坏；由于具有低阶性，这些模块可以互相组合（注意，不是交叉操作），形成高质量的高阶模块，即更集中的解空间。

Schema定理给我们提供了一个GA过程的理论解释，具有很强的指导意义。然而，这一理论也有很大局限性，特别是它没有考虑在繁衍过程中各个超平面之间的相互关系，因此很难得到对GA趋势的全局性判断。另一个问题是这一理论对变异策略在GA中的作用解释不足。由Schema定理，变异操作部分$(1-p_m)^{o(H)}$是一种指数级的破坏，特别是对低相隔距离的优质模型产生破坏，从而导致第$t+1$代种群在各个超平面上的分布极大偏离第t代种群，引起算法不稳定。然而，在现实应用中，如果没有这种破坏，交叉操作可能很快进入局部最优，产生所谓的'未成熟收敛'（Prematurely Converge）现象。很多研中发现，变异操作可减少这种局部收敛的概率。事实上，即使只使用变异操作，也可以得到扩展性非常好的解。

9.3 遗传编程（GP）

标准GA的操作对象是数据，如模型的参数或搜索任务中的路径。如果我们将操作对象换作一组连续操作，基于和GA中类似的演化原则，即对操作过程进行学习。这种基于演化原则对操作过程进行学习的方法称为遗传编程（Genetic Programming, GP）。如果我们将GA比作选择最好的积木搭出最美的房子，GP则是学习给定一批积木，学习如何搭出最美房子的过程。GP可以认为是GA的扩展。

9.3.1 算法基础

图 9.6给出GP的基本流程。在GP中，每个个体对应一个算法流程，GP的目的是搜索一个有效的算法，使得运行该算法时得到的目标收益最大化。图9.6给出这一搜索过程：首先初始化一个种群，种群中每个个体对应一个算法；对这些个体依适应函数进行个体选择，其中算法的适应函数值通过运行该算法产生的结果时行评价；选择完成后，应用交叉繁衍和变异策略生成新个体，得到下一代种群。上述个体选择和种群繁衍过程迭代进行，直到得到质量让人满意的算法。上述过程和GA中的演化过程是一致的，只有在个体对象和适应函数的选择上有所区别。

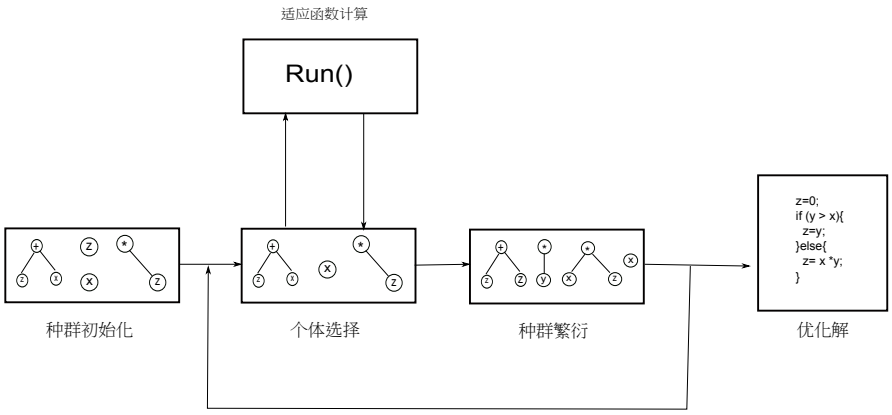

Fig. 9.6 GP基本框架：初始化一个算法种群，对每个个体通过运行其所对应的算法计算其适应函数值；基于该适应函数值进行个体选择，并应用交叉繁衍和个体变异操作生成新种群。经过反复迭代，得到优化的种群，其中最优个体即对应得到的优化算法。

基因编码

GP中最重要的问题是如何对算法流程进行编码。和GA不同，GP中不同个体可能对应一个计算过程，其长度可能是不等的，且语义信息更丰富。语法树编码是GP中通用的编码方式。该编码的符号表包括若干操作符和操作对象，将这些符号用树结构组织起来，基于一定语法规则，这一树结构即可表示一个算法流程。图 9.7给出一个语法树的例子，其中每个中间节点代表一个操作符，用来对其直接子节点进行操作，得到的结果送入父节点做上一

层操作。定义好上述语法规则后，每个语法树对应唯一一个算法。注意，该语法树中的操作符可以是任意一个函数，该函数以其所有子节点为变量，因此该语法树每个节点可能包含多个子节点，每个节点对应操作符的复杂性也可能有很大差别。上述语法树结构可以进一步扩展，每个节点可以是一个子图，代表一个子过程，如图 9.8。将图划分为有层次的嵌套子图具有重要意义，它使得在GP过程中某些固定计算单元可以作为整体参与演化，因而极大提高学习效率。

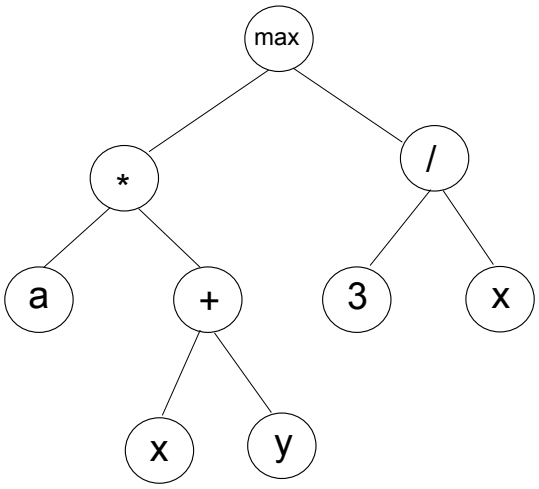

Fig. 9.7 GP中用语法树结构表示的算法流程。每个叶子节点代表一个被操作对象，每个中间节点表示一个操作符。图中所示语法树对应算法为：$max(a*(x+y),3/x)$。图片来自[44]。

初始化

树状编码使得GP的初始化更加复杂。较常用的初始化方法包括两种：全路径初始化和增长型初始化。不论哪种初始化，都需要指定树的深度，即从根节点到叶子节点的最长路径。例如，图 9.7中编码树的深度为3，最长路径为（$max,*,+,x$）和（$max,*,+,y$）。确定了树的深度d，全路径初始化从根结点出发直到$d-1$层的所有中间节点都随机选择一个操作符，直到到达第d层节点时，随机选择操作对象。因此，全路径初始化生成的树都是完全的，即所有路径的长度都是d。图 9.9给出全路径初始化的过程。

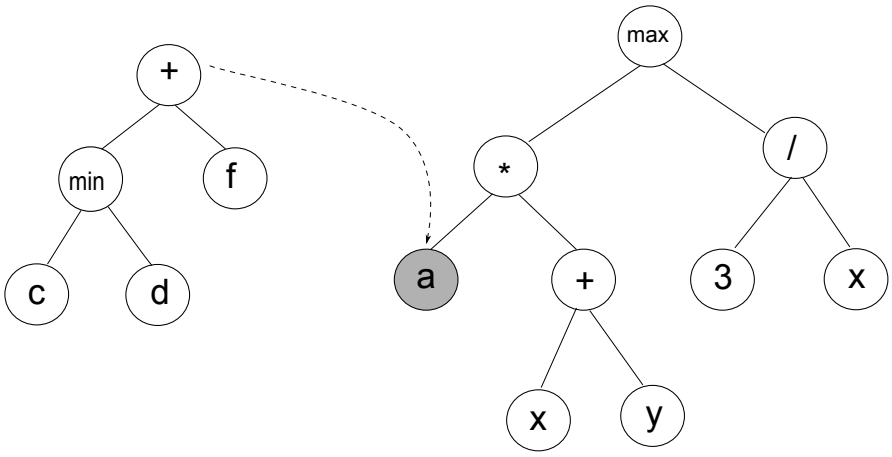

Fig. 9.8 GP中带子图的语法树结构，其中节点a为一个子图，代表的子算法为$a = min(c,d)+f$。图片来自 [44]。

在增长型初始化方法中，生成每个节点时既可以选择一个操作对象，也可以选择一个操作符。当选择操作对象时，该路径结束；当选择一个操作符时，将该节点当作中间节点并扩展到下一层继续路径生成。由于这一初始化方式可能在任一节上点选择操作对象，因而生成的树是不完全的，有些路径长度小于d。图 9.10 给出增长型初始化的过程。

对比上述两种初始化方式，全路径生成的树更均衡，增长方式生成的种群更具多样性。实践中可以将两种方式结合起来，一半用全路径生成，另一半用增长方式生成（称为Ramped Half-and-Half）。其它初始化方式包括对所有可能树结构做均匀采样 [42]，或基于领域知识进行初始化 [2]。

个体选择

GA中一般通过计算每个个体的适应函数值或适应性的相对排序来确定如何对个体进行选择，如余量随机采样方法。这一方法也可以用于GP，只不过GP中每个个体是一段程序，因此需要运行每个个体所代表的程序，并通过运行结果来确定个体的适应函数。这一过程通常是很耗时的，不适合大规模GP学习。

GP中常用的个体选择方法是‘比武决胜’（Tournament Selection, TS）法。例如我们要选出一个样本执行变异策略，TS方法在当前种群中随机选择一个子集，在子集中挑选一个最好的个体作为变异样本。这一方法不需对所有样

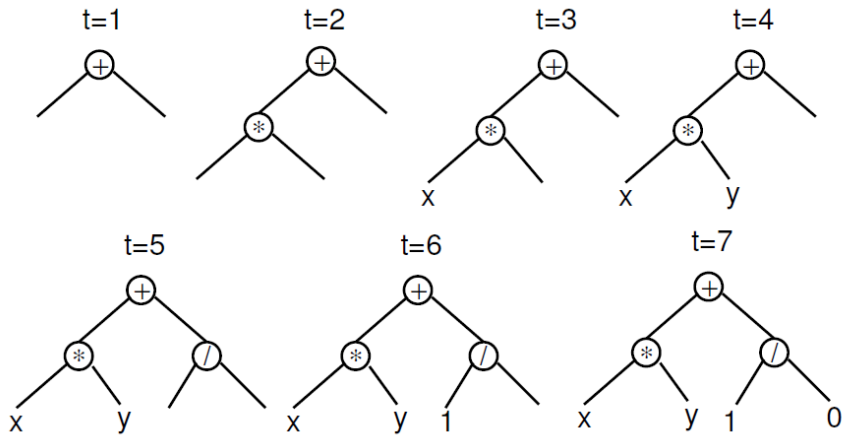

Fig. 9.9 GP的全路径初始化过程，*t*表示生成步骤。该初始化过程生成的编码树是完全的，树中所有路径长度相等。

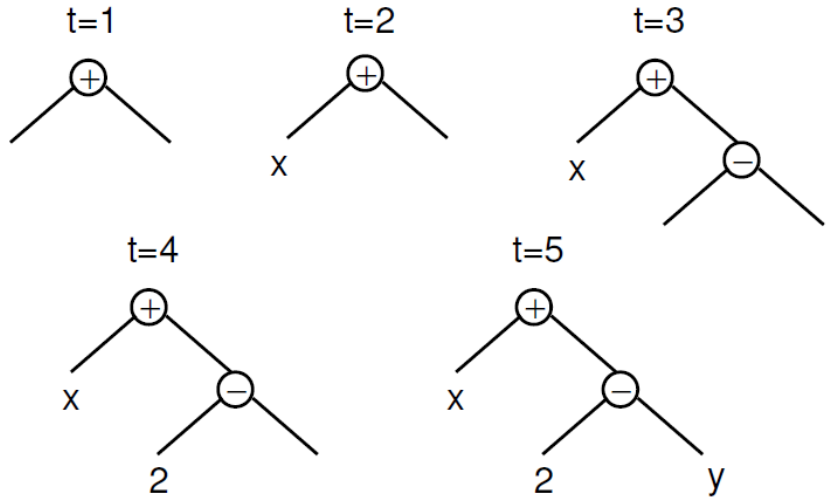

Fig. 9.10 GP的增长型初始化过程，*t*表示生成步骤。该初始化过程得以的树是不完全的，树中路径长度不等。

本做适应函数计算，因此计算量不受当前种群大小的影响；同时，我们只关注子集中的最好个体，而不是适应函数的具体值，因此可以用一些近似方法来对适应性进行排序，而不必要实际运行每个个体代表的程序。对交叉繁衍策略，同样可以用TS方法选出父样本，只不过需要做两次TS，得到两个父样本。

TS方法是一种保守进化方法，它不对当前种群中的最好个体做过度推荐，而是通过个体间的互相比较选择相对高质量的个体。这一方法似乎更符合生物进化的实际情况，可以保持足够的个体多样性，特别适合较大规模种群的进化。

交叉与变异

基于其树状表示，GP中的交叉策略和GA中的交叉策略有显著区别。最常用的GP交叉策略是子树交叉。在这一策略中，在两个父样本的语法树T_1和T_2中各自随机确定一个交叉点h_1和h_2，从T_2中截取以h_2为根节点的子树S_{h_2}来替换T_1中以h_1为根节点的子树S_{h_1}，形成新的个体，如图 9.11 所示。注意，图 9.11中父个体T_2中非S_{h_2}的部分和T_1中的S_{h_1}可以被丢弃不用，但这两部分也可以组合起来成为另一个新个体。

上述交叉策略依赖两个父样本中交叉点的选择。一般来说我们希望控制交叉过程带来的随机性，因此需要替换的子树不能过大。一种方法是在选择交叉节点时考虑每个节点包含的子节点个数，包含子节点数越多的节点其子树越大，越不应该被轻易替换。在所有节点上设计一个非平均匀随机分布，使得子树越大的子节点被选作交叉节的概率越小，可以有效控制随机风险。

关于变异策略，一种常见的方法是随机选择一个变异点，将该变异点以下的子树替换成一个随机生成的树。这一随机树可用全路径方式（Full）或成长方式（Grow）生成。注意这一子树变异方式可认为是交叉策略的特例，其T_2为一个随机生成树，且交叉节点h_2是T_2的根节点。和交叉策略相似，在变异策略中我们也需要控制变异带来的随机性，因此依子树大小设计的非均匀概率分布对变异节点的选择优为重要。另一种变异操作是单点变异，即只随机改变树中的某个节点，而不是重新初始化其子树。其它变异方式包括升级变异（Hoist Mutation）, 缩减变异（Shrink Mutation）, 置换变异（Permutation Mutation）等，具体可见 [44]。一些早期研究者认为变异操作可能是不必要的，如Koza [40]，但后来研究者发现小规模的变异是有价值的。

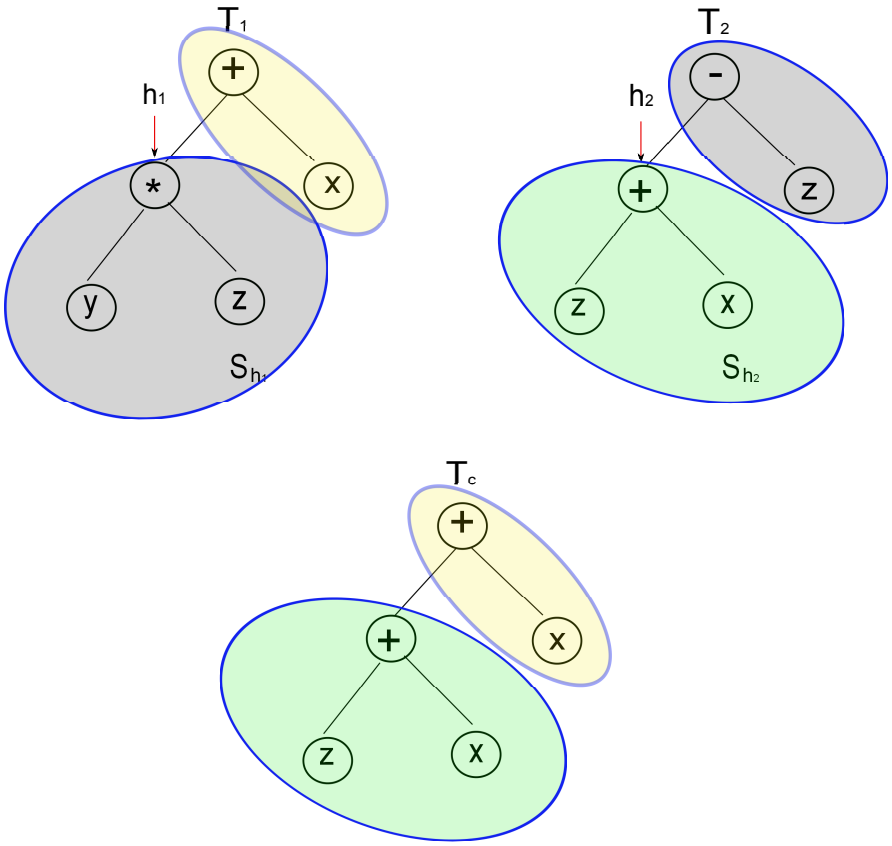

Fig. 9.11 GP的子树交叉策略。在两个父个体对应的树T_1和T_2上随机选择交叉点h_1和h_2（红色箭头所示），T_1的子树S_{h_1}被替换成T_2的子树S_{h_2}，形成新个体T_c。T_1和T_2中未被选择的部分（灰色阴影部分）被丢弃。

现在一般认为绝大部分的操作应该是交叉操作（90%），只有极少数操作是变异操作（小于1%）。

上述交叉和变异策略假设任何交叉组合都是合法的，这对复杂算法来说是不现实的。假设某一节点是一个操作函数，这个函数所能接受的信息在种类、值域、语义上都有一定限制。这些限制在初始化、交叉、变异等各个环节都需要认真考虑，保证生成的算法是合法性。

GP演化过程中经常会看到一种突变（Bloat）现象：在GP初期，种群中个体的平均长度（编码树的节点数）保持相对稳定，但经过一段时间的演化后，这一平均长度会忽然增长，但整个种群的适应函数值并没有显著提高。研究者从理论上提出了很多解释，如基于Schema理论的种群平均长度理论

[52]。突变现象对GP的性能造成了破坏，不仅加重了计算负担，也降低了结果的可扩展性。一种控制突变现象的简单方法是对个体长度进行惩罚，降低个体选择过程中冗长个体的接受概率，或减少繁衍过程中冗长个体的生成概率 [71]。

9.3.2 GP高级话题

线性编码

GP不仅可用来生成计算公式，更重要的是可以自动生成计算程序，其中每一步可执行某一基本操作。通过对操作序列做演化学习，可得到完成任务的计算机程序。然而，通用的树结构编码并不适合程序学习，一是这种表达效率不高，特别是在做交叉或变异操作时比较复杂，二是需要特别的解释器，通用性不够。程序指令是用二进制表示的，因此很自然的想法是用二进制编码来代替树结构编码，称为线性编码。

和树结构不同的是，线性编码中不包含具体的操作数据，而是附加若干个寄存器结构，程序中的指令（对应基因位）从某些寄存器中读取数据，再将计算结果存回某一寄存器中。换句话说，线性编码中只包含操作指令，持行器将依编码中保存的指令顺序依次执行。执行器可以是机器的CPU，也可以是领域相关的指令解释器。一些典型的GP自动编程机可参考 [41, 23]。

基于上述二进制编码方式，交叉和变异操作变得相对简单，例如可以用双交叉点方式在父本基因上选择一个片段，这些片段可以在两个父本间进行交换以得到新个体。这种交互可以是等位的（即只有父本间对应位置的基因片段可互换），也可以是非等位的。变异操作可对某一位置指令进行随机替换，但需要保证变换后的指令满足程序的合法性（如寄存器的读取顺序、取值范围等）。

图编码

前面提到过，模块化编程是GP得以提高性能的重要手段。模块化编程需要改变树编码结构，允许不同节点可以共享同一个子树，如图 9.12所示。可以看到，在（a）所示树结构编码中，两个灰色结点表示的子树具有相同结

构，因此只需保留一个子树，并将其连接到相关的结点上，形成如（b）所示的图结构。这种允许子树共享的编码方式称为图编码。

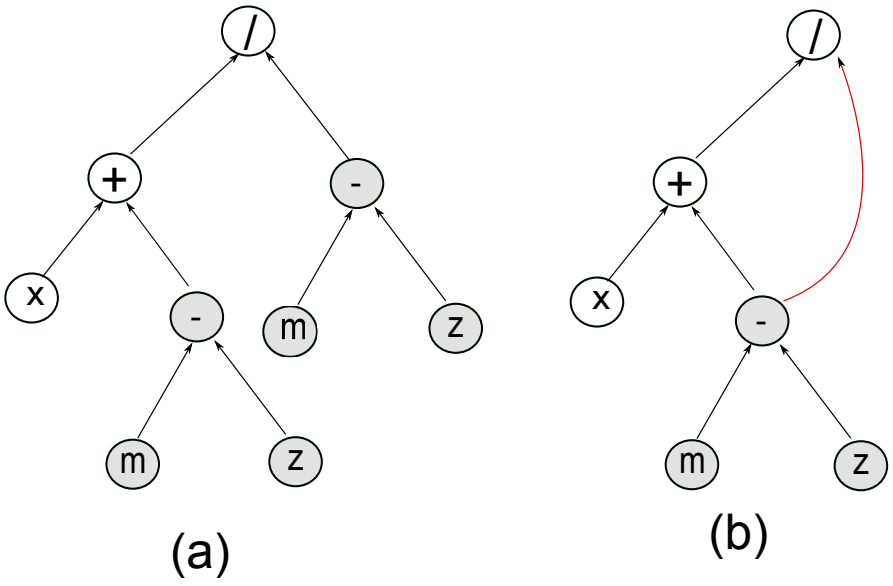

Fig. 9.12 GP的图编码方式。（a）朴素树结构，包含两个同样子树（灰色结点）；（b）等价图结构，两个节点连接到同一个子树。

概率GP

传统GA/GP基于个体选择生成中间种群，并基于交叉和变异生成新种群。前面我们在讨论Schema定理时提到，这一过程相当于对具有较高适应函数的超平面给以更多采样。显然，这一采样过程是隐性的。一种可能的代替方法是不依赖选择与繁衍等进化操作，而是基于当前种群中个体的适应函数信息直接采样出新个体。这一方法虽然不依赖进化操作，但依然是一种种群进化方法，称为概率GP。概率估计算法（Estimation of Distribution Algorithm, EDA）是一种典型的概率GP方法。这一方法基于当前种群信息估计一个采样分布，并基于该分布直接采样出下一代种群的个体。注意，这一采样分布随着种群的进化逐渐发生改变，使其越来越倾向于生成具有更高适应函数值的新个体。简单的EDA假设个体基因中所有基因位是互相独立的，复杂的EDA依任务要求在基因位间建立适当的相关性。另外，将EDA应

用到GP时，需要对每个基因位放置哪些操作符进行建模，或对不同操作符在基因中的相对位置进行建模 [43]。

概率GP可以认为是一种软性语法和语义限制，因此依概率生成的个体有更大可能性是符合任务要求的合法个体，因而可极大减少错误生成率，提高生成效率。同时，也需要对每个个体做严格检查，保证其合法性。另一方面，我们也可以将任务信息作为一种语法和语义规则引入到个体生成中，这些规则可以是确定的，也可以是随机的。这种依知识的生成的个体有更大概率是合法个体。一些研究者将语法和语义规则和概率GP结合起来，取得了不错效果 [57]。

GP的理论解释

前面讨论过GA的Schema理论，这一理论同样可用来解释GP过程。Schema理论将解空间分成众多超平面，并估计每个超平面上的随机样本数 [33]。研究者扩展了这一理论来解释GP的动态行为特性 [50, 51]。其它对GP的理论解释包括马尔可夫模型和统计概率理论等 [14, 53]。上述对GP的解释有一定指导意义，但绝大多数理论都基于较强的假设，对复杂问题下的行为还不能解释。然而我们之所以用GP，就是因为它可以解决复杂问题。从这个角度来说，很难有一种理论可以解释真实应用场景下的GP行为。

9.3.3 其它演化学习方法

上一节我们介绍了GA，本节介绍了GP，这两者都基于同样的生物进化原则。基于这一原则的优化算法还有若干变体，这些变体只是在编码方式、适应函数计算、条件限制等方面有所差异，有些是演化学习发展早期的初级形式。我们对这些方法不作一一展开，仅列举他们的一些主要特征。

- 进化编程（Evolutionary Programming，EP）。与GP类似，但不允许改变程序结构，只允许改变程序中的数据参数 [22]。EP由Fogel在60年代提出，当时以一个有限状态机（FSM）为操作对象，通过EP选择合理的参数来提高FSM的预测能力。EP发展到今天，已经和GP没有太大区别。
- 进化策略（Evolution Strategy，ES） [7]。ES由Rochenberg [55]等在60年代提出。ES中一般采用浮点数编码，并以变异操作作为主要演化工具。基本的ES种群中只保留一个个体，对父个体执行变异操作，直到得到一个比

父个体更优的个体，即得到下一代种群。这一方法称为$1+1$-ES。在$1,\lambda$-ES中，执行变异操作得到λ个个体，这些个体参与选择，最优个体被保留到下一代，当前个体被丢弃。ES算法事实上是GA的早期形式。

- 基因表达编程（Gene Expression Programming，GEP）是一种线性编码的遗传编程方法 [18]。和标准线性编码GP不同，GEP的编码是等长的，且编码中只有部分基因片段可被表达成个体（树状表达）。因此一个GEP的基因可表达多个不等长的算法。这类似于人类基因中只有少部分在成长过程中被表达出来，因而同样的基因在发展过程中会生成不同性状的个体（如双胞胎）。在进化算法中，基因编码形式通常称为Genotype，由该编码表达出的个体通常称为Genetype。GEP即是用定长的Genotype表达不定长的Genetype的GP方法。
- 差异进化（Differential Evolution, DE）。DE是另一种简单的演化学习方法 [62]。这一方法多用于数值优化算法中。简单的DE算法从种群中随机选取4个体，每个个体是一个数值向量，记为x,a,b,c，计算$y=a+F\times(b-c)$，将x与y做随机交叉，其中每一维的交叉操作都是独立随机的。如果我们将这一过程视为x与a,b,c的交叉，这样当于包含多个父个体的交叉演化策略。
- 神经进化（Neuro Evolution，NE）。NE类似于GP和EP，但其操作对象为一个神经网络，利用演化学习策略生成神经网络的参数和结构 [19]。传统神经网络学习一般采用基于梯度下降的BP算法，这一方法无法改变网络结构，对包含非连续激活函数的复杂网络无法训练。基于NE，只需随机出一些神经网络的结构和参数，并确定这些网络在特定任务上的性能，即可选出优化的网络结构和参数。如在游戏任务上，游戏结果很难通过BP对网络参数进行学习，特别是对网络结构进行调整。通过NE，即可选择合理的结构和参数。这一方法特别适用于那些具有非连续激发函数的网络，如基于冲激序列的仿生神经网络 [27, 37]。
- 分类器学系统（Learning Classifier System, LCS）。LCS最早由John Henry Holland提出，用GA算法来生成人类可读的规则 [34]。这些规则通过在线学习训练样例自动学习，并通过剪裁进行压缩。这一方法称为Michigan-style LCS。80年代，Kenneth de Jong 和Stephen Smith用离线学习方法提取规则[65]，这一方法称为Pittsburgh-style LCS。不论哪种方法，早期的LCS都试图通过训练数据总结一系列形如'IF-THEN'的规则，并对规则的适用性进行赋值。特别重要的是，这些规则首先由训练数据样例得到，并通过演化策略进行扩展。90年代，Wilson进一步发展了LCS系统，引

入Q-learning算法中的反馈信号作为适应函数，使之可以处理强化学习任务 [68]。

9.4 群体学习方法

前面我们讨论的演化学习算法主要模拟生物种群的进化过程。这一过程的重要特征之一是群体学习，通过种群间不同个体互相交换信息来协调种群的整体优化方向。这一群体学习方式不仅体现在遗传进化过程中，也是许多物种的日常生活方式。一般来说，群居物种群都具有某些特定的协同行为，这种协同行为具有自组性和分工合作两种属性。自组织性通过个体间的局部交互行为形成群体行为模式。这些个体交互包括正向反馈（如个体间互相跟随）、负向反馈（如猎食时的互相竞争）、随机行为等。分工合作通常使种群的工作效率更高。这些基于个体交互衍生出的群体行为往往表现的比个体行为更有目的性，因而也称为群体智能（Swarm Intelligence） [8]。

本节介绍几种基于群体智能的优化算法，这些算法严格来说不算进化学习，但在思路和目标上都与进化学习有很强相关性。例如两者都基于适应函数对个体进行选择；他们都依赖某种受限制的随机性来搜索潜在的求解子空间，以对抗组合爆炸问题。

9.4.1 蚁群优化算法（*ACO*）

Dorigo于1992年提出蚁群优化算法（Ant Colony Optimization）用来解决图上的路径搜索问题 [16]。目前，ACO已经被广泛用于组合优化任务中，如资源规划 [6, 25]，路径规划 [56, 15]，任务分配 [54]，图象处理 [45]等。

ACO算法模拟蚁群寻找目标的行为方式：每只蚂蚁在找到一条有效路径后，会在路上洒下信息素，信息素的多少与路径长短（或其它激励）相关；其它蚂蚁会依局部目标的远近和前面同伴留下信息素的多少来决定往哪个方向寻找。这一搜索可顺序分批进行，每一次放出一批蚂蚁，一批蚂蚁搜索完成后在路径上留下信息素，作为下一批蚂蚁寻找路径的后验知识。经过多轮搜索，蚂蚁们即可找到高质量的优化路径。信息素是蚂蚁间进行个体交互的主要方式，也是蚁群协调行动的重要机制。相应的，将历史搜索结果作为先验信息指导后续搜索，是ACO算法的主要特征。

以旅行商问题为例来说明ACO的具体步骤。旅行商问题（Traveling Salesman Problem, TSP）的目的是在地图上寻找一条最短路径，这条路径通过所有城市，且每个城市只去一次。ACO用如下步聚解决TSP：

1. 初始化所有城市间的信息素为一个小量$\tau_{ij} = \delta$，其中i, j代表城市编号。
2. 选择N只蚂蚁，从初始点开始尝试进行路径搜索。搜索的每一步依如下概率选择目标城市：
$$p_{ij} \propto \tau_{ij}^{\beta} d_{ij}^{\alpha},$$
其中d_{ij}为城市i, j之间的路径长度, α, β为参数。注意在选择目标城市时，还需加入TSP问题的具体限制，即每个蚂蚁每个城市只能去一次。
3. 待所有N只蚂蚁路径搜索后，每只蚂蚁得到一条完整合法的旅行路线。此时更新路径的信息素如下：
$$\tau_{ij} = (1-\rho)\tau_{ij} + \rho \sum_{n=1}^{N} \delta_{ij \in \xi_n} \frac{1}{L_n},$$
其中ρ是更新参数（对应前一批蚂蚁留下的信息素的挥发系数），ξ_n代表第n只蚂蚁走过的路径，L_n代表ξ_n的总长度。上式表明，某只蚂蚁发现的路径越短，它留下的信息素就越多。
4. 返回到第2步，派出下一批蚂蚁重新开始路径搜索。由于前一批蚂蚁已经留下了很多有价值的信息，新的搜索空间会更加优化。

类似TSP的组合优化问题一般用启发搜索或GA等方法求解。和这些方法相比，ACO的一个优点是它具有动态学习能力。例如当路径中加入堵车、商品价格变化等因素，TSP问题的最优解将随时间动态变化。ACO可通过派出新蚁群实现对新环境的适应学习。这种适应学习来源于ACO 的概率估计过程：每一批新蚂蚁返回后，信息素会即时更新，这相当于将后验概率转化为先验概率，因此过去的知识可被新环境所利用，同时又可以被及时更新以适应新环境。这种概率估计方法与我们在监督学习中的贝叶斯在线学习有相似性，虽然信息素的更新公式并不直接对应一个合理的后验估计。

值得一提的是，ACO与我们前面提到的EDA（Estimation of Distribution Algorithm）有相似之处：二者都是通过当前信息对解空间做概率估计，并基于该概率生成可供继续搜索的候选解。ACO与EDA不同在于，EDA基于上述概率信息直接生成解空间采样，而ACO将上述概率与历史信息结合起来，更新局部路径的概率值，再基于该局部概率进行解空间采样。

9.4.2 人工蜂群算法（*ABC*）

人工蜂群算法（Artificial bee colony algorithm，ABC）是另一种群体优化方法 [36]。ABC算法模拟蜂群的工作方式。一个蜂群系统包括三类职能的工蜂：采蜜蜂（Employed bee）、驻守蜂（Outlook bee）和巡逻蜂（Scout bee）。最开始时，有若干巡逻蜂被派出去寻找蜜源，一旦找到蜜源即开始采蜜，这些巡逻蜂相应转化成采蜜蜂。这些采蜜蜂把采到的蜜带回蜂巢，并通过舞蹈的方式把蜜源的信息共享给驻守蜂。驻守蜂得到这些蜜源消息后，依蜜源的质量随机选择不同蜜源进行采蜜。蜜源的质量越好，跟着去采蜜的驻守蜂越多。当蜜源被开采到一定程度，其质量会逐渐下降，直到没有继续开采价值后被放弃。这时一些蜜蜂再次转变成巡逻蜂，飞出去寻找新的蜜源[2]。

ABC算法借鉴蜜蜂的这一群体行为方式来处理优化任务。将每个蜜源认为是一个可能的解空间，这一解空间中的每个解都有一个相应的适应函数。采蜜过程相当于这一解空间中通过发掘可能的解，只要不断有更好的解被发现，就意味着这一蜜源没有枯竭。每个解空间中的信息，即最优解的质量由采蜜蜂带回，驻守蜂们基于这些解的相对优劣决定去不同解空间去寻找更优化的解，其中对解空间的选择概率正比于该解空间中最优解的适应函数。如果一个解空间长期没有发现更好的解，意味着该蜜源已经枯竭，则丢掉该解空间，并随机一个新的解空间重新探索。这一蜂群算法的基本流程可以有很多变种，算法 1给出一种简单的实现方法。

另一种模拟蜂群行为的优化算法由Pham在2005提出，称为Bees Algorithm （BA） [49]。这一算法和ABC算法的思想基本一致，但实现方法上有如下差别：（1）BA算法中采蜜蜂和驻守蜂会在自己选定的蜜源附近随机搜索，而ABC借用其它蜜源的位置进行搜索；（2）BA算法中如果某个蜜源在某次迭代中没有找到更优解，下次搜索时缩小搜索范围。因此，当几次搜索都无法找到更优解时，当前蜜源位置即为局部最优解，不必继续搜索下去；（3）BA算法将蜜源分成优质蜜源、有价值蜜源和无价值蜜源，对优质蜜源指派更多驻守蜂去发掘，对无价值蜜源则直接丢弃，重新派巡逻蜂寻找新蜜源。[3]

---

[2] http://www.scholarpedia.org/article/Artificial_bee_colony_algorithm

[3] https://en.wikipedia.org/wiki/Bees_algorithm

Input: N, M, α, γ;
Initialization:
$\mathbf{x}_n = Rand()$; $n = 1, ..., N$; //init each honey source
$c_n = 0$; $n = 1, ..., N$; //set extinction clock
while *True* **do**
  for *n:*=1 *to N* **do**
    $p_n = \frac{f(\mathbf{x}_n)}{\sum_i f(\mathbf{x}_i)}$; //fraction of outlookers to source n
    $M_n = Rand(M, p_n)$; //number of outlookers to source n
    //search new honey by each outlooker
    $\mathbf{x}'_n = \mathbf{x}_n$;
    for *m :*=1 *to* M_n **do**
      //use the position of anther honey source to find new honey
      $j = Random(1, N)$; $j \neq i$
      $\mathbf{x}_n = \mathbf{x}_n + \alpha(\mathbf{x}_n - \mathbf{x}_j)$;
      //accept better honey and reset extinction clock
      if $f(\mathbf{y}_n) > f(\mathbf{x}'_n)$ **then**
        $\mathbf{x}'_n = \mathbf{y}_n$; $c_n = 0$;
      end
    end
    if $f(\mathbf{x}'_n) > f(\mathbf{x}_n)$ **then**
      $\mathbf{x}_n = \mathbf{x}'_n$; //update the honey source if better found
    end
    else
      $c_n = c_n + 1$; //update the extinction clock
    end
    if $c_n == \gamma$ **then**
      $\mathbf{x}_i = Rand()$; $c_n = 0$; //Research new source if distinction clock reach γ
    end
  end
  if *Converged*() **then**
    break;
  end
  Output: $\arg\max_{\mathbf{x}_n} f(\mathbf{x}_n)$;
end

Algorithm 1: Artificial Bee Colony （ABC）优化算法。N为解空间个数（采蜜蜂个数）；M为在各个解空间寻找新解的总个数（驻守蜂个数）；α为生成新解的插值参数；γ为放弃解空间需要的迭代次数；$\mathbf{x}_i$是第i个解空间的最优解，$f(\mathbf{x})$为解x的适应函数；c_i是第i个解空间未找到更优解的次数。当c_i达到γ时，说明第$\mathbf{x}_i$已经几次迭代没有更新（相当于蜜源已经枯竭），因此要被放弃重新采样。Converged()是判断迭代是否收敛的函数。Rand(M, p_i)依概率p_i确定在第i个解空间中搜索新解的个数。

9.4.3 粒子群算法（PSO）

粒子群算法（Particle Swarm Optimization, PSO）最初提出来是为了模拟羊群或鱼群的整体行为 [38, 58]。这些动物在寻最目标的时候，单个个体的搜索策略既受到自己过去认知的影响，也受到群体全局认知的影响，这两者共同决定每个个体每一步的行为方式。全局认知是使得动物可以成群的原因，而个体认知提供了较好的多样性。简单的PSO算法如算法 2所示，该算法包括如下步骤：

1. 对每个个体n的位置$\mathbf{x}_n$和目标优化方向$\mathbf{v}_n$作随机初始化；
2. 计算每个个体的适应函数$f(\mathbf{x}_n)$，并初始化局部优化位置为$\mathbf{p}_n = \mathbf{x}_n$:
3. 对所有个体互相比较得到全局最优位置$\mathbf{g}$;
4. 对每个个体n，基于局部最优位置$\mathbf{p}_n$和全局最优位置$\mathbf{g}$计算目标优化方向$\mathbf{v}_n$;
5. 对每个个体n，如果目标优化方向$\mathbf{v}_n$使得这一个体的适应函数值增加，则接受该方向，并更新局部优化位置$\mathbf{p}_n$;
6. 更新全局优化位置$\mathbf{g}$
7. 如果得到满意的全局最优解$\mathbf{g}$，退出程序，否则回到步骤4。

Input: $N, K, \omega, \phi_p, \phi_g$;
Initialization: $\mathbf{x}_n = Rand(); \mathbf{v}_n = Rand();\ n = 1, ..., N$;
//set local optimal $\mathbf{p}_n$;
$\mathbf{p}_n = \mathbf{x}_n$;
//set global opitmal**g**
$\mathbf{g} = \arg\max_{\mathbf{x}_i} f(\mathbf{x}_i)$ **while** *True* **do**
    //adjust each individual **for** $n :=$ 1 *to* N **do**
        //update each dimension (variable) of each individual
        for $k :=$1 *to* K **do**
            $r_p \sim U(0,1)$;
            $r_g \sim U(0,1)$;
            $v_{nk} = \omega v_{nk} + \phi_p r_p(p_{nk} - x_{nk}) + \phi_g r_g(g_k - x_{nk})$;
        end
        //update individual position
        $\mathbf{x}_n = \mathbf{x}_n + \mathbf{v}_n$;
        //update local optimal
        if $f(\mathbf{x}_n) > f(\mathbf{p}_n)$ **then**
            $\mathbf{p}_n = \mathbf{x}_n$;
        end
        //update global optimal
        $\mathbf{g} = \arg\max_{\mathbf{x}_n} f(\mathbf{x}_n)$;
    end
    //break if converged
    if *Converged()*==*True* **then**
        Break;
    end
end
Output: **g**;

Algorithm 2: PSO算法。N为种群大小，K为解的维度，$\mathbf{v}_n$为第n个个体的位置（候选解法），$\mathbf{v}_n$为第n个个体的速度，$\mathbf{p}_n$为第n个个体发现的最优解，$\mathbf{g}$为全局最优解，ω, ϕ_p, ϕ_g为参数。

PSO算法受参数影响较大。如果对全局优化点过于重视，群体有更强的总体方向性，可很快收敛，但容易陷入局部最优；如果对个体优化点过于重视，群体行为过于散乱，收敛较慢，但陷入局部最优的风险较小。研究者提

出了各种方法对参数进行合理选择 [59, 66]。关于PSO算法的稳定性和收敛性问题也有很多相关研究 [11, 9]。

9.4.4 捕猎者搜索（*HuS)*

捕猎者搜索（Hunting Search，HuS ）是另一种基于群体行为的优化算法[47]。这一算法模拟狼群等群体性捕猎者的猎食行为，将优化目标视为猎物，将每个捕猎者个体所在位置视为一个有效解，该解的目标函数作为适应函数，对应捕猎者的位置优势。HuS初始化一组捕猎者作为候选解，找到位置最优的捕猎者作为领头捕猎者，所有捕猎者向领头捕猎者随机靠拢，如果靠拢后位置更优时，则停留在该位置，否则退回。到达新位置后，每个捕猎者对自己位置做进一步调整，或者加入随机扰动，或者与其它捕猎者交换部分解（等价于GA中的变异和交叉操作）。上述过程重复进行，直到找到满意的解（对应领头捕猎者的位置）。为防止搜索陷入局部最优，HuS提出群体重组策略，保持领头捕猎者不变，其余捕猎者重新随机选择位置。

9.4.5 荧火虫算法（*FA*）

荧火虫算法（Firefly Algorithm, FA）模拟荧火虫飞行时的互相吸引行为[69]。FA将每个荧火虫的亮光类比为优化函数的值，并假设荧火虫间通过光亮互相吸引：亮度较低的荧火虫会受到亮度较高的荧火虫吸引，向更亮的荧火虫趋近；群体中最亮的个体不受其它个体吸引，自身随机搜索。FA事实上是PSO的一种变种，只不过PSO中个体受到全局最优个体吸引，FA中受到所有比当前个体更优个体的吸引。

9.5 其它随机优化方法

群体优化方法基于个体间的信息交互保证群体学习方向的一致性。一些研究者认为这种群体信息交互具有重要意义，使得群体行为比个体行为更具有智能性 [8]。然而，有些研究者认为复杂的信息交互并不能带来明显的学习效率，至少就优化任务而言，随机性是更重要的内容，仅依靠随机性即可

实现演化学习和群体学习的大部分目标。本节我们介绍几种典型的随机优化方法，其中模拟退火算法最为典型。

9.5.1 模拟退火算法（*SA*）

金属在加热时如果缓慢降温，会形成更大晶粒，表现出更好的物理性能。这一过程称为‘退火‘。模拟退火算法（Simulated Annealing，SA）模拟这一物理过程，通过一个温度参数调节优化过程中的随机性，使得优化过程得以摆脱局部最优 [39]。设我们的目标是最小化某一函数，并设当前解为s，需要寻找更好的解s'来代替s。传统的爬山算法选择使梯度下降最大的s'。在模拟退火算法中，从s的邻域中随机生成一个新的解s'，对这一解设计一个接受概率$P(s,s',T)$，其中T是温度参数。重要的是，即使s'的函数值高于s（即s'质量低于s），$P(s,s',T)$还是具有大于零的概率，意味着s' 依然可以被接受。当然，$s'-s$越小，$P(s,s',T)$的概率越大。T的作用是控制$P(s,s',T)$的形状，T越大$P(s,s',T)$对$s'-s$越不敏感。例如，$P(s,s',T)$可定义如下：

$$P(s,s',T)=\begin{cases}1 & if\ s'<s\\ e^{-(s'-s)/T} & otherwise.\end{cases}$$

在模拟退火算法中，开始时对T设一个较大值，使一些质量较差的s'也有较高的接收概率，搜索过程相比混乱。这种混乱有利于在搜索初期摆脱局部最优。在继续优化过程中，T的取值被逐渐降低，较差的解越来越难以被接受；当$T=0$时，只有$s'<s$的解会被接受，这时模拟退火算法回归到传统爬山算法，帮助我们快速找到局部最优解。因为在算法初期已经有很大概率摆脱了局部最优，后期找到的局部最优解往往是全局最优解。一个简单的退火优化流程如算法 3所示。

Input: K;
Initialization: $s = Rand()$;
//loop for K times
for k:=1 *to* K **do**
    //set temperature of the iteration
    $T = k/K$;
    //search candidate solution s' around s
    $s' = Rand(s)$;
    //test if s' is accepted
    $u \sim U(0,1)$;
    if $P(s,s',T) \geq u$ **then**
        $s = s'$;
    end
    Output: s;
end

Algorithm 3: 简单的模拟退火算法。$U(0,1)$是0-1上的均匀分布。K是最大迭代次数。

和演化学习、群体学习等方法相比，模拟退火算法非常简单，仅在搜索过程中引入了随机性。有研究者认为，这一算法在绝大部分情况下已经足以克服局部最优问题，并不需要象GA或PSO这种复杂的群优化方法 [60][4]。

9.5.2 杜鹃搜索（*CS*）

一些杜鹃将蛋产在别人的窝里，让别的鸟帮它们孵化后代 [48]。为了提高自己子女的存活率，杜鹃妈妈会想办法把别人的蛋替换成自己的蛋。然而，这种偷换的杜鹃蛋为也经常会被宿主鸟类发现，被清除出去。为此，杜鹃妈妈们想尽了各种办法来冒充宿主鸟类，在蛋的大小、颜色方面进行模仿，甚至小杜鹃都会模仿宿主鸟的叫声，以增加自己被喂食的机会。

杜鹃搜索算法（Cuckoo Search）借鉴杜鹃的这种寄生行为到优化任务中 [70]。在这种算法中，每个杜鹃蛋代表一种解法，蛋的质量正比于该解法的适应函数。假设有N个宿主巢，每随机生成一个新蛋，杜鹃妈妈随机选一个宿主巢j，如果新蛋的质量高于该巢中原有蛋的质量，则用新蛋替换原来的

[4] https://en.wikipedia.org/wiki/Genetic_algorithm

蛋（相当于给出了更优解法），反之则保持原巢中的蛋不变。每次放好新蛋后，为模拟被宿主发现的风险，对当前N个巢中的蛋质量排序，以p_a为比例扔掉最差的蛋并随机生成新蛋。上述CS算法每个巢里只允许有一个蛋，也可以扩展到允许多个蛋的情况。

比较重要的特点是如何生成新的蛋。一种简单方法可以基于布朗运动，每次用均匀分布生成一个新方向，在新方向上的位移距离由一个高斯分布得到，即：

$$\mathbf{x}^t = \mathbf{x}^{t-1} + \mu\mathbf{v}; \quad \mu \sim N(0,\sigma).$$

其中$\mathbf{x}^t$是第t时刻生成的新蛋（即新生成的解），$\mathbf{v}$是任一方向的单位向量。Yang等人发现这种生成方式生成的$\mathbf{x}^t$多样性不足，因此建议使用基于Levy分布的位移距离。

CS算法最大的好处是简单易实现，需要调节的参数少（仅有丢弃比例p_a一个参数）。注意CS算法中虽然生成了N个蛋，但群体学习特征并不明显，个体间的信息交换只是通过质量竞争和排序，因此我们将之视为一种随机优化方法。

9.5.3 和声搜索（HS）

和声搜索（Harmony Search，HS）是受爵士乐演奏启发提出的一种优化方法 [26]。在该算法中，每个音乐弹奏一种乐器，对应解中的一个变量（即搜索空间中的一个维度），弹奏出的音符对应该变量的取值。所有音乐家一起弹奏，得到一个解，弹奏的和谐程度对应该解的目标函数值。HS算法很简单：初始化若干解$\{\mathbf{x}_i\}$，生成一个新的解$\mathbf{x}'$时，对每一维x'_j随机独立生成，生成方式以一定概率p从现有解$\{\mathbf{x}_i\}$的对应维度$\{x_{ij}\}$ 随机选择，或以概率$1-p$随机生成。对从现有解中得到的x'_j，还需加入一定噪声（高斯的或离散的）。如果$\mathbf{x}'$ 优于现有解中质量最差的解，则替换该解，成为现有解集中的解。上述过程迭代进行，直到解集中解的质量不再提高为止。

HS算法和GA算法有很强相似性，事实上$\mathbf{x}'$的生成过程即包括了交叉和变异两种操作，只不过这里的交叉不是两个父本间的交叉，而是解集中所有个体的随机交叉。仅管存在这种个体间的信息交互，但这一交互仅用于生成新的解，因此群体学习特征并不明确。我们将HS归类为随机学习方法。

9.5.4 禁忌搜索（*TS*）

禁忌搜索（Tabu Search, TS） [28]是一个组合搜索算法。该算法模拟人类在搜索时头脑里的记忆功能，在搜索过程基于当前局部最优解搜索附近的解，在搜索过程中对已经保存在记中的解不再考虑。一个简单的TS搜索过程如算法 4所示，其中记忆是一种短时记忆，即超过一定时间的记忆被丢弃。和杜鹃搜索、和声搜索类似，TS只要依赖新解法的随机性，群体学习特点不明显，因而应该认为是一种随机优经方法。

```
Input: K;
Initialization:
//init state
s = Rand();
//init tabu list
G = s;
//init global optimal s_g and local optimal s_l
s_g = s_l = s;
while True do
    //search for the best around s_l and not in the tab list
    f' = −∞;
    s' = 0;
    for s ∈ Nb(s_l) do
        if (s ∉ G) AND (f(s) > f') then
            s' = s;
            f' = f(s);
        end
    end
    //reset local optimal
    s_l = s';
    //update global optimal
    if f(s_l) > f(s_g) then
        s_g = s_l;
    end
    //update tabu list, and purge the old memory
    G = Clean(G ∪ {s_l});
    Output: s_g;
end
```

Algorithm 4: 简单的禁忌搜索，其中记忆为短时记忆。G为禁忌列表，或历史记忆; $\mathbf{s}_g$和$\mathbf{s}_l$ 分别为全局和局部优化点。$Nb(\cdot)$是新样本生成函数。$Clean(\cdot)$为对记忆的整理函数，如去掉超过一定时间的记忆。

9.6 本章小节

本章讨论了演化学习方法及其相关技术，包括群体学习方法和随机优化算法。我们首先讨论了最基本的演化学习方法–遗传算法（GA）。GA包含了演化学习的所有基本概念：基于适应函数的个体选择、基于交叉和变异的种群繁衍、基于世代更迭的群体进化。我们讨论了GA的Schema理论，这一理论表明通过个体选择，种群中的个体会越来越集中到适应函数较高的搜索空间，而交叉和变异等繁衍操作在提供有效随机性的同时，使得新一代种群不会偏离上一代种群过远。

我们讨论了遗传编程（GP）。GP和GA的不同在于其种群中的个体不再局限于静态数据（模型参数或搜索路径），而是引入了对行为过程的编码。这意味着我们不仅可以学习既定模型的优化参数，而且可以学习完成某一任务的过程和方法。换句话说，GP可以突破人为设计的局限，只需给定任务目标，即可自主学习如何完成该目标的行动流程，实现自主编程。这种目标驱动的学习方式类似于我们在第 3章讨论的神经图灵机（Neural Turing Machine, NTM） [30] 或可微分神经计算机（Differentiable Neural Computer, DNC） [31]。不同的是，NTM和DNC是基于可微分的参数模型，用推理方法训练，而GP基于采样法和优化选择进行训练。另一个不同是NTM/DNC可以学习元操作本身，而GP通常是在定义好的原操作上进行选择。GP与下一章要介绍的强化学习也具有某些方面的相似性：二者都是以目标为导向的过程学习，但强化学习会设计过程中各步骤之间的概率关系（如马尔可夫随机性），因此可利用这些结构化信息提高学习效率。GP没有结构化信息可利用，是纯粹的采样重试，效率较低，但应用范围更广。

我们进一步讨论了若干群体学习方法。这些群体学习方法不具有生物进化的背景，但绝大多数方法模拟了生物种群的某些行为，如蚂蚁和蜜蜂的觅食、狼群的捕猎等。这些群体学习方法希望通过模拟种群居动物个体间的通讯机制来提高优化任务的效率。这些仿生方法曾给研究者很大启发，但也带来了一些不好的趋势，使得研究者热衷于为自己的算法寻找生物学背景，满足于提出一些看似新奇，但在算法本身并无特点的伪创新，事实上反而限制了真正的算法创新。

我们也讨化了以模拟退火算法为代表的随机优化方法。如果从优化角度来看，随机优化方法大很大程度上已经可以克服局部最优问题，在很多任务上表现出良好性能。基于模拟退火算法完善的理论基础和简单的算法流程，人们对包括GA在内的群体学习的实际价值提出了一定质疑。总而言之，选

择哪种方法可能还是需要依实际应用场景具体确定，但优先选择简单的算法确实是机器学习里的一条基本原则。

上述所有方法的共同点在于‘尝试-选择’这一基本流程，即采样法。这也是这些方法区别于我们前面所述各种模型的根本所在。这种方法的优点在于可以摆脱局部最优，但更重要的是对复杂模型的建模。从某种意上讲，这二者是一致的，因为越复杂的模型越容易产生局部最优问题。因而，应用采样法的一个基本原则是：只有当对任务的先验知识极为缺乏或模型非常复杂时，才考虑用采样法，用时间换性能；当模型比较清晰简单时，应该首先考虑基于推理的学习方法，因为这类方法的效率更高。

9.7 相关资源

- 关于GA，本章参考了Banzhaf的综述文章《Genetic programming: an introduction》 [4]。
- 关于GP，本章参考了McPhee等人的综述文章《Field guide to genetic programming》 [44]。
- 关于群体学习和随优化算法，本章参考了Wiki上的相关信息[5]。
- 关于GA的早期著作，请参考 [29, 67, 46]。

[5] https://en.wikipedia.org/wiki/Evolutionary_algorithm

Chapter 10
强化学习

Chapter 11
优化方法

References

[1] Akbari R, Ziarati K (2011) A multilevel evolutionary algorithm for optimizing numerical functions. International Journal of Industrial Engineering Computations 2(2):419–430

[2] Aler R, Borrajo D, Isasi P (2002) Using genetic programming to learn and improve control knowledge. Artificial Intelligence 141(1):29–56

[3] Baluja S, Caruana R (1995) Removing the genetics from the standard genetic algorithm. In: Machine Learning: Proceedings of the Twelfth International Conference, pp 38–46

[4] Banzhaf W, Nordin P, Keller RE, Francone FD (1998) Genetic programming: an introduction, vol 1. Morgan Kaufmann San Francisco

[5] Barricelli NA (1957) Symbiogenetic evolution processes realized by artificial methods. Methodos 9(35-36):143–182

[6] Bauer A, Bullnheimer B, Hartl RF, Strauss C (2000) Minimizing total tardiness on a single machine using ant colony optimization. Central European Journal of Operations Research 8(2):125–141

[7] Beyer HG, Schwefel HP (2002) Evolution strategies–a comprehensive introduction. Natural computing 1(1):3–52

[8] Bonabeau E, Dorigo M, Theraulaz G (1999) Swarm intelligence: from natural to artificial systems. 1, Oxford university press

[9] Bonyadi MR, Michalewicz Z (2014) A locally convergent rotationally invariant particle swarm optimization algorithm. Swarm Intelligence 8(3):159–198

[10] Box GE (1957) Evolutionary operation: A method for increasing industrial productivity. Applied statistics pp 81–101

[11] Clerc M, Kennedy J (2002) The particle swarm-explosion, stability, and convergence in a multidimensional complex space. IEEE transactions on Evolutionary Computation 6(1):58–73

[12] Crosby JL, et al (1973) Computer simulation in genetics.

[13] Darwin C, Beer G (1951) The origin of species. Dent

[14] Davis TE, Principe JC (1993) A markov chain framework for the simple genetic algorithm. Evolutionary computation 1(3):269–288

[15] Donati AV, Montemanni R, Casagrande N, Rizzoli AE, Gambardella LM (2008) Time dependent vehicle routing problem with a multi ant colony system. European journal of operational research 185(3):1174–1191

[16] Dorigo M (1992) Optimization, learning and natural algorithms. Ph D Thesis, Politecnico di Milano, Italy

[17] Eiben AE, Raue PE, Ruttkay Z (1994) Genetic algorithms with multi-parent recombination. In: International Conference on Parallel Problem Solving from Nature, Springer, pp 78–87

[18] Ferreira C, Gepsoft U (2008) What is gene expression programming

[19] Floreano D, Dürr P, Mattiussi C (2008) Neuroevolution: from architectures to learning. Evolutionary Intelligence 1(1):47–62

[20] Fogel DB (1998) Evolutionary computation: the fossil record. Wiley-IEEE Press

[21] Fogel DB (2006) Evolutionary computation: toward a new philosophy of machine intelligence, vol 1. John Wiley & Sons

[22] Fogel LJ, Owens AJ, Walsh MJ (1966) Artificial intelligence through simulated evolution

[23] Foster JA (2001) Discipulus: A commercial genetic programming system. Genetic Programming and Evolvable Machines 2(2):201–203

[24] Fraser A, Burnell D, et al (1970) Computer models in genetics. Computer models in genetics

[25] Gagné C, Price W, Gravel M, et al (2002) Comparing an aco algorithm with other heuristics for the single machine scheduling problem with sequence-dependent setup times. Journal of the Operational Research Society 53(8):895–906

[26] Geem ZW, Kim JH, Loganathan G (2001) A new heuristic optimization algorithm: harmony search. simulation 76(2):60–68

[27] Ghosh-Dastidar S, Adeli H (2009) Spiking neural networks. International journal of neural systems 19(04):295–308

[28] Glover F (1986) Future paths for integer programming and links to artificial intelligence. Computers & operations research 13(5):533–549

[29] Goldberg DE, et al (1989) Genetic algorithms in search optimization and machine learning

[30] Graves A, Wayne G, Danihelka I (2014) Neural turing machines. arXiv preprint arXiv:14105401

[31] Graves A, Wayne G, Reynolds M, Harley T, Danihelka I, Grabska-Barwińska A, Colmenarejo SG, Grefenstette E, Ramalho T, Agapiou J, et al (2016) Hybrid computing using a neural network with dynamic external memory. Nature 538(7626):471–476

[32] Gray F (1953) Pulse code communication. US Patent 2,632,058

[33] Holland JH (1992) Adaptation in natural and artificial systems: an introductory analysis with applications to biology, control, and artificial intelligence. MIT press

[34] Holland JH, Reitman JS (1977) Cognitive systems based on adaptive algorithms. Acm Sigart Bulletin (63):49–49

[35] Janikow CZ, Michalewicz Z (1991) An experimental comparison of binary and floating point representations in genetic algorithms. In: ICGA, pp 31–36

[36] Karaboga D (2005) An idea based on honey bee swarm for numerical optimization. Tech. rep., Technical report-tr06, Erciyes university, engineering faculty, computer engineering department

[37] Kasabov NK (2014) Neucube: A spiking neural network architecture for mapping, learning and understanding of spatio-temporal brain data. Neural Networks 52:62–76

[38] Kennedy J (2011) Particle swarm optimization. In: Encyclopedia of machine learning, Springer, pp 760–766

[39] Kirkpatrick S, Gelatt CD, Vecchi MP, et al (1983) Optimization by simulated annealing. science 220(4598):671–680

[40] Koza JR (1994) Genetic programming ii: Automatic discovery of reusable subprograms. Cambridge, MA, USA

[41] Koza JR, Goldberg DE, Fogel DB, Riolo RL (1996) Genetic Programming 1996: proceedings of the first annual conference. Mit Press

[42] Langdon WB (2000) Size fair and homologous tree crossovers for tree genetic programming. Genetic programming and evolvable machines 1(1-2):95–119

[43] Larrañaga P, Lozano JA (2001) Estimation of distribution algorithms: A new tool for evolutionary computation, vol 2. Springer Science & Business Media

[44] McPhee NF, Poli R, Langdon WB (2008) Field guide to genetic programming

[45] Meshoul S, Batouche M (2002) Ant colony system with extremal dynamics for point matching and pose estimation. In: Pattern Recognition, 2002. Proceedings. 16th International Conference on, IEEE, vol 3, pp 823–826

[46] Mitchell M (1998) An introduction to genetic algorithms. MIT press

[47] Oftadeh R, Mahjoob M, Shariatpanahi M (2010) A novel meta-heuristic optimization algorithm inspired by group hunting of animals: Hunting search. Computers & Mathematics with Applications 60(7):2087–2098

[48] Payne RB, Sorensen MD (2005) The cuckoos, vol 15. Oxford University Press

[49] Pham D, Ghanbarzadeh A, Koc E, Otri S, Rahim S, Zaidi M (2011) The bees algorithm-a novel tool for complex optimisation. In: Intelligent Production Machines and Systems-2nd I* PROMS Virtual International Conference (3-14 July 2006), sn

[50] Poli R (2001) Exact schema theory for genetic programming and variable-length genetic algorithms with one-point crossover. Genetic Programming and Evolvable Machines 2(2):123–163

[51] Poli R (2001) General schema theory for genetic programming with subtree-swapping crossover. In: European Conference on Genetic Programming, Springer, pp 143–159

[52] Poli R (2003) A simple but theoretically-motivated method to control bloat in genetic programming. Genetic programming pp 43–76

[53] Prügel-Bennett A, Shapiro J (1997) An analysis of genetic algorithms using statistical mechanics. Physica D 104:75–114

[54] Ramalhinho Lourenço H, Serra D (2002) Adaptive search heuristics for the generalized assignment problem. Mathware & soft computing 2002 Vol 9 Núm 2 [-3]

[55] Rechenberg I (1973) Evolutionsstrategie optimierung technischer systeme nach prinzipien der biologishen evolution. PhD thesis

[56] Secomandi N (2000) Comparing neuro-dynamic programming algorithms for the vehicle routing problem with stochastic demands. Computers & Operations Research 27(11):1201–1225

[57] Shan Y, McKay R, Essam D, Abbass H (2006) A survey of probabilistic model building genetic programming. Scalable Optimization via Probabilistic Modeling pp 121–160

[58] Shi Y, Eberhart R (1998) A modified particle swarm optimizer. In: Evolutionary Computation Proceedings, 1998. IEEE World Congress on Computational Intelligence., The 1998 IEEE International Conference on, IEEE, pp 69–73

[59] Shi Y, Eberhart RC (1998) Parameter selection in particle swarm optimization. In: International conference on evolutionary programming, Springer, pp 591–600

[60] Skiena SS (1998) The algorithm design manual, vol 1. Springer Science & Business Media

[61] Srinivas M, Patnaik LM (1994) Adaptive probabilities of crossover and mutation in genetic algorithms. IEEE Transactions on Systems, Man, and Cybernetics 24(4):656–667

[62] Storn R, Price K (1997) Differential evolution–a simple and efficient heuristic for global optimization over continuous spaces. Journal of global optimization 11(4):341–359

[63] Turing A (1948) Intelligent machinery. report for national physical laboratory. reprinted in ince, dc (editor). 1992. mechanical intelligence: Collected works of am turing

[64] Turing AM (1950) Computing machinery and intelligence. Mind 59(236):433–460

[65] Urbanowicz RJ, Moore JH (2009) Learning classifier systems: a complete introduction, review, and roadmap. Journal of Artificial Evolution and Applications 2009:1

[66] Van Den Bergh F (2007) An analysis of particle swarm optimizers. PhD thesis, University of Pretoria

[67] Whitley D (1994) A genetic algorithm tutorial. Statistics and computing 4(2):65–85

[68] Wilson SW (1995) Classifier fitness based on accuracy. Evolutionary computation 3(2):149–175

[69] Yang XS (2010) Nature-inspired metaheuristic algorithms. Luniver press

[70] Yang XS, Deb S (2009) Cuckoo search via lévy flights. In: Nature & Biologically Inspired Computing, 2009. NaBIC 2009. World Congress on, IEEE, pp 210–214

[71] Zhang BT, Ohm P, Mühlenbein H (1997) Evolutionary induction of sparse neural trees. Evolutionary Computation 5(2):213–236