Artificial Intelligence
人工智能

第4章 高级搜索技术

第4章 高级搜索

4.1 爬山法搜索
4.2 模拟退火搜索
4.3 遗传算法
4.4 案例分析

搜索算法在内存中保留一条或多条路径并且记录哪些是已经探索过的，哪些是还没有探索过的。当找到目标时，到达目标的路径同时也构成了这个问题的一个解。在许多问题中，问题的解与到达目标的路径是无关的。例如，在八皇后问题中，重要的是最终皇后的布局，而不是加入皇后的次序。

这一类问题：

集成电路设计；工厂场地布局作业车间调度；自动程序设计电信网络优化；车辆寻径文件夹管理

局部搜索算法从单独的一个当前状态出发，通常只移动到与之相邻的状态。典型情况下，搜索的路径是不保留的。

- 优点：

（1）它们只用很少的内存

（2）它们通常能在很大状态空间中找到合理的解

除了找到目标，局部搜索算法对于解决纯粹的最优化问题是很有用的，其目标是根据一个目标函数找到最佳状态。

许多最优化问题不适合于“标准的”搜索模型。如，自然界提供了一个目标函数——繁殖适应性——达尔文的进化论可以被视为优化的尝试，但是这个问题没有“目标测试”和“路径耗散”。

- 地形图既有“位置”（用状态定义），又有“高度”（由启发式耗散函数或目标函数的值定义）。
- 如果高度对应于耗散，那么目标是找到最低谷——即一个全局最小值；
- 如果高度对应于目标函数，那么目标是找到最高峰——即一个全局最大值。
- 局部搜索算法就象对地形图的探索，如果存在解，那么完备的局部搜索算法总能找到解；最优的局部搜索算法总能找到全局最小值／最大值。

4.1 爬山法搜索

爬山法搜索——局部搜索

登高——一直向值增加的方向持续移动，将会在到达一个“峰顶”时终止，并且在相邻状态中没有比它更高的值。

这个算法不维护搜索树，因此当前节点的数据结构只需要记录当前状态和它的目标函数值。

爬山法不会预测与当前状态不直接相邻的那些状态的值。

八皇后问题。

局部搜索算法通常使用完全状态形式化，即每个状态都表示为在棋盘上放八个皇后，每列一个。

后继函数返回的是移动一个皇后到和它同一列的另一个方格中的所有可能的状态（因此每个状态有8×7：56个后继）。

启发式耗散函数h是可以彼此攻击的皇后对的数量，不管中间是否有障碍。

该函数的全局最小值是0，仅在找到完美解时才能得到这个值。图4.1(a)显示了一个h=17的状态。图中还显示了它的所有后继的值，最好的后继是h=12。爬山法算法通常在最佳后继的集合中随机选择一个进行扩展，如果这样的后继多于一个的话。

图4.1（a）

18	12	14	13	13	12	14	14
14	16	13	15	12	14	12	16
14	12	18	13	15	12	14	14
15	14	14	Q	13	16	13	16
Q	14	17	15	Q	14	16	16
17	Q	16	18	15	Q	15	Q
18	14	Q	15	15	14	Q	16
14	14	13	17	12	14	12	18

（b）

						Q	
				Q			
	Q						
			Q				
					Q		
							Q
		Q					
Q							

爬山法（贪婪局部搜索）。

爬山法能很快朝着解的方向进展。例如，从图4-1（a）中的状态，它只需要五步就能到达图4-1（b）中的状态，它的h=1，这基本上很接近于解了。可是，爬山法经常会遇到下面的问题：

(1) 局部极大值：

(2) 山脊：山脊造成的是一系列的局部极大值，贪婪算法处理这种情况是很难的。

(3) 高原：

- 局部极大值是一个比它的每个邻居状态都高的峰顶，但是比全局最大值要低。
- 爬山法算法到达局部极大值附近就会被拉向峰顶，然后被卡在局部极大值处无处可走。
- 更具体地，图4-1（b）中的状态事实上是一个局部极大值（即耗散h的局部极小值）；不管移动哪个皇后得到的情况都会比原来差。

高原是在状态空间地形图上评价函数值平坦的一块区域。它可能是一块平的局部极大值，不存在上山的出路，或者是一个山肩，从山肩还有可能取得进展。爬山法搜索可能无法找到离开高原的道路。

在各种情况下，爬山法算法都会达到无法取得进展的状态。

从一个随机生成的八皇后问题的状态开始，最陡上升的爬山法86%的情况下会被卡住，只有14%的问题实例能求解。

这个算法速度很快，成功找到最优解的平均步数是4步，被卡住的平均步数是3步——对于包含88个状态的状空间，这已经是不错的结果了。

针对爬山法的不足，有许多变化的形式。

如，随机爬山法，它在上山移动中随机地选择下一步；选择的概率随着上山移动的陡峭程度而变化。这种算法通常比最陡上升算法的收敛速度慢不少，但是在某些状态空间地形图上能找到更好的解。

再如，首选爬山法，它在实现随机爬山法的基础上，采用的方式是随机地生成后继节点直到生成一个优于当前节点的后继。这个算法在有很多后继节点的情况下有很好的效果。

- 爬山法算法是不完备的——它们经常会在目标存在的情况下因为被局部极大值卡住而找不到该目标。
- 随机重新开始爬山法，它通过随机生成的初始状态来进行一系列的爬山法搜索，找到目标时停止搜索。这个算法是完备的概率接近于1，原因是它最终会生成一个目标状态作为初始状态。

- 如果每次爬山法搜索成功的概率为p，那么需要重新开始搜索的期望次数为1／p。
- 对于不允许侧向移动的八皇后问题实例，p≈0.14，因此大概需要7次迭代就能找到目标（6次失败1次成功）。
- 所需步数的期望值为一次成功迭代的搜索步数加上失败的搜索步数与（1−p）／p的乘积，大约是22步。如果允许侧向移动，则平均需要迭代约1／0.94≈1.06次，平均步数为（1×21）+（0.06／0.94）×64≈25步。
- 那么对于八皇后问题，随机重新开始的爬山法实际上是非常有效的，甚至对于三百万个皇后，这个方法用不了一分钟就可以找到解。

第4章 高级搜索

4.1 爬山法搜索

4.2 模拟退火搜索

4.3 遗传算法

4.4 案例分析

- 模拟退火算法（Simulated Annealing，SA）的基本思想是模拟固体退火过程。固体退火是将固体加热，使其温度达到充分高，随着温度的升高，固体内部的粒子逐渐变为无序状态，然后再将固体徐徐降温，在降温的过程中，固体内部的粒子逐渐趋于有序状态，在每个温度都达到平衡，直至常温，固体达到基础状态。
- 类比优化搜索过程，在一定温度下，搜索从一个状态随机地变化到另一个状态，随着温度的不断降低，直至最低温度，搜索过程以接近 1 的概率停留在最优解。
- ＳＡ算法是对爬山算法的优化改进，爬山算法在搜索过程中只接受更优的邻近解，直至搜索不到更优解为止，算法简单易实现，但是却极有可能停止在局部最优解。

- ＳＡ算法引入一个温度参数，当搜索到较差的邻近解时，利用温度参数和目标函数值之差共同确定一个概率参数，利用此概率参数决定是否接受较差的邻近解，概率参数P的定义为

$$P = \mathrm{e}^{-\frac{\delta}{t}}$$

- 其中，δ为邻近解与当前解的目标函数之差，t为温度参数，该参数对应于固体退火过程中的温度，随着搜索过程的不断推进而不断减小，直至算法达到终止条件。
- 在温度下降足够慢时，算法找到全局最优解的概率接近１。

第4章 高级搜索

4.1 爬山法搜索

4.2 模拟退火搜索

4.3 遗传算法

4.4 案例分析

4.3.1 遗传算法的基本思想

- 遗传算法是从代表问题可能潜在解集的一个种群（population）开始的，而一个种群则由经过基因（gene）编码（coding）的一定数目的个体（individual）组成。
- 个体是染色体（chromosome）带有特征的实体。
- 染色体作为遗传物质的主要载体，即多个基因的集合，其内部表现（即基因型）是某种基因组合，它决定了个体形状的外部表现，如黑头发的特征是由染色体中控制这一特征的某种基因组合决定的。
- 需要从表现型到基因型的映射（编码）

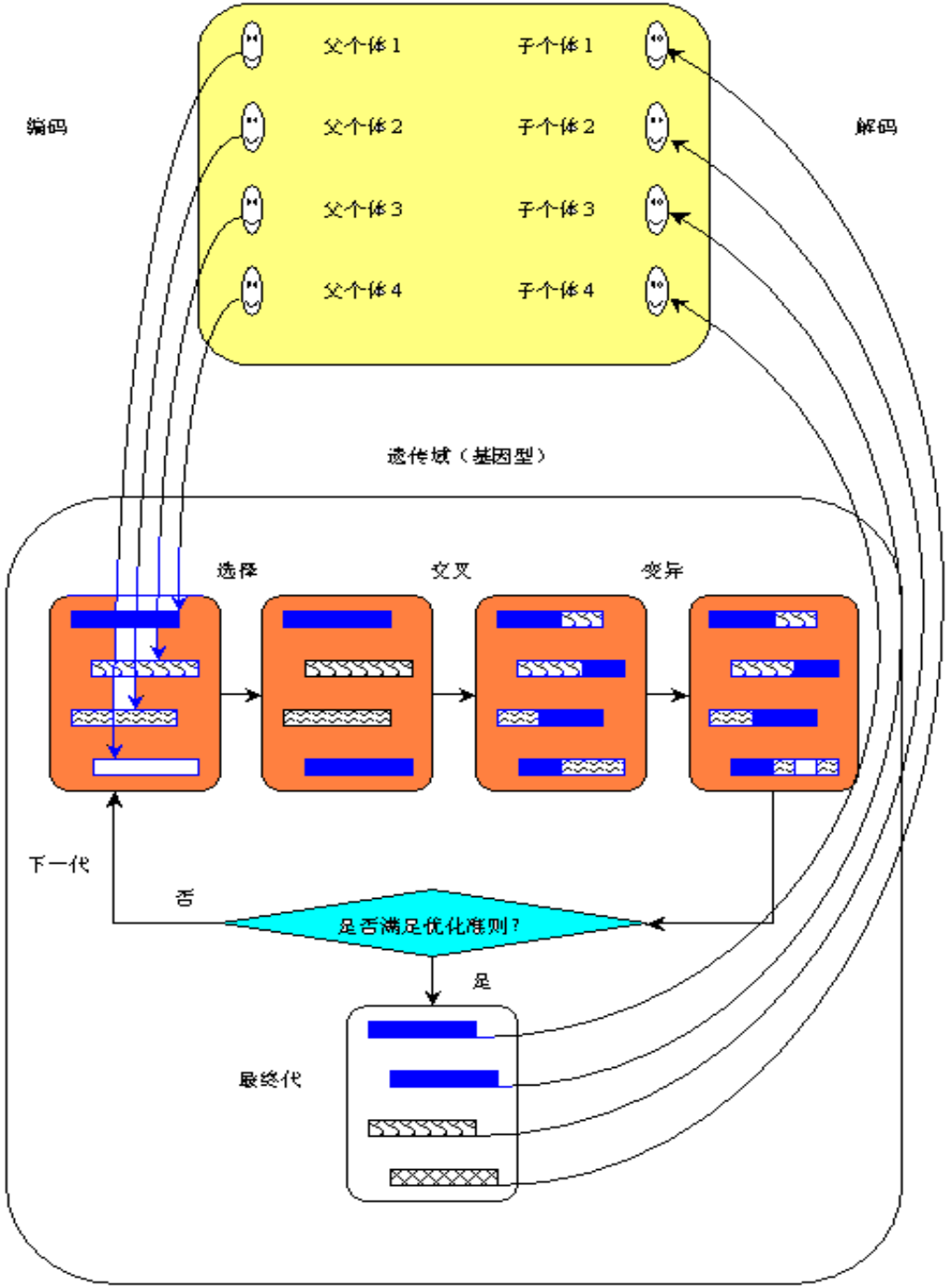
- 简化基因编码，如二进制编码。
- 初代种群产生之后，按照适者生存和优胜劣汰的原理，逐代(generation）演化产生出越来越好的近似解。

- 在每一代，根据问题域中个体的适应度（fitness）大小挑选个体，并借助于自然遗传学的遗传算子进行组合交叉和变异，产生出代表新的解集的种群。

- 这个过程将导致种群像自然进化一样的后代种群比前代更加适应于环境，末代种群中的最优个体经过解码，可以作为问题近似最优解。

遗传算法采纳了自然进化模型，如选择、交叉、变异、迁移、局域与邻域等。

图 遗传算法的过程



- 开始时，种群随机地初始化，并计算每个个体的适应度函数，初始代产生了。
- 如果不满足优化准则，开始产生新一代的计算。
- 为了产生下一代，按照适应度选择个体，父代进行基因重组（交叉）而产生子代。
- 所有的子代按一定概率变异。然后子代的适应度又被重新计算，子代被插入到种群中将父代取而代之，构成新的一代。
- 循环，直到满足优化准则。

遗传算法不同于传统的搜索和优化方法。区别：

1. 自组织、自适应和自学习性（智能性）。

- 应用遗传算法求解问题时，在编码方案、适应度函数及遗传算子确定后，算法将利用进化过程中获得的信息自行组织搜索。由于基于自然的选择策略为“适者生存，不适应者被淘汰”，因而适应度大的个体具有较高的生存概率。

适应度大的个体具有更适应环境的基因结构，再通过基因重组和基因突变等遗传操作，就可能产生更适应环境的后代。

利用遗传算法，可以解决那些复杂的非结构化问题。

2. 遗传算法的本质并行性。遗传算法按并行方式搜索一个种群数目的点，而不是单点。
3. 遗传算法不需要求导或其他辅助知识，而只需要影响搜索方向的目标函数和相应的适应度函数。
4. 遗传算法强调概率转换规则，而不是确定的转换规则。
5. 遗传算法可以更加直接地应用。
6. 遗传算法对给定问题，可以产生许多的潜在解，最终选择可以由使用者确定（在某些特殊情况下，如多目标优化问题不止一个解存在，有一组pareto最优解。这种遗传算法对于确认可替代解集而言是特别合适的）。

4.3.2 遗传算法的基本操作

选择
交叉
变异

1. 选择

- 选择是用来确定重组或交叉个体，以及被选个体将产生多少个子代个体。首先计算适应度：
  (1) 按比例的适应度计算；
  (2) 基于排序的适应度计算。
- 适应度计算之后是实际的选择，按照适应度进行父代个体的选择。

- 可以挑选以下的算法：
  ① 轮盘赌选择；
  ② 随机遍历抽样；
  ③ 局部选择；
  ④ 截断选择；
  ⑤ 锦标赛选择。

2．交叉或基因重组

- 基因重组是结合来自父代交配种群中的信息产生新的个体。

依据个体编码表示方法的不同，可以有以下的算法：

① 实值重组

离散重组；　　中间重组；

线性重组；　　扩展线性重组

② 二进制交叉

单点交叉；　　多点交叉；

均匀交叉；　　洗牌交叉；

缩小代理交叉

3．变异（ mutation ）

- 交叉之后子代经历的变异，实际上是子代基因按小概率扰动产生的变化。
- 依据个体编码表示方法的不同，可采用以下算法：① 实值变异；② 二进制变异。

- 例：考察一下二进制编码的轮盘赌选择、单点交叉和变异操作。

- 下图是一组二进制基因码构成的个体组成的初始种群。个体的适应度评价值经计算由括号内的数值表示，适应度越大代表这个个体越好。

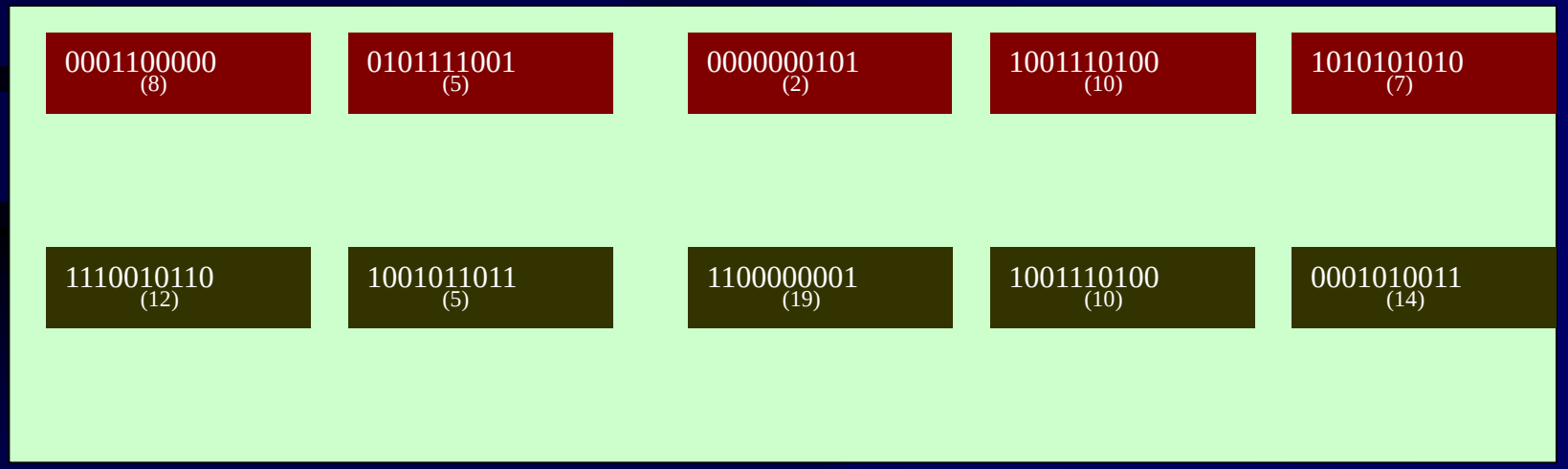
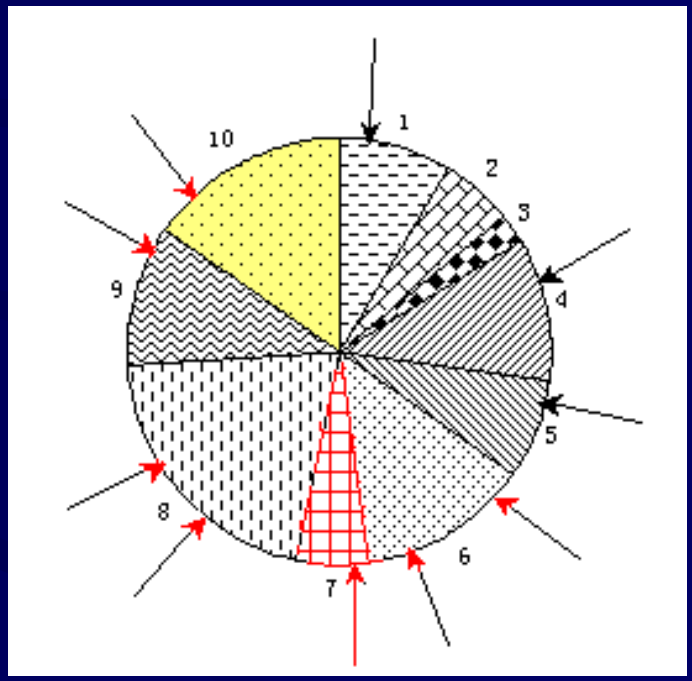


图4-5 初始种群的分布

表4-2 初始种群及其选择计算

个体	染色体	适应度	选择概率	累积概率
1	0001100000	8	0.086 957	0.086 957
2	0101111001	5	0.054 348	0.141 304
3	0000000101	2	0.021 739	0.163 043
4	1001110100	10	0.108 696	0.271 739
5	1010101010	7	0.076 087	0.347 826
6	1110010110	12	0.130 435	0.478 261
7	1001011011	5	0.054 348	0.532 609
8	1100000001	19	0.206 522	0.739 130
9	1001110100	10	0.108 696	0.847 826
10	0001010011	14	0.152 174	1.000 000

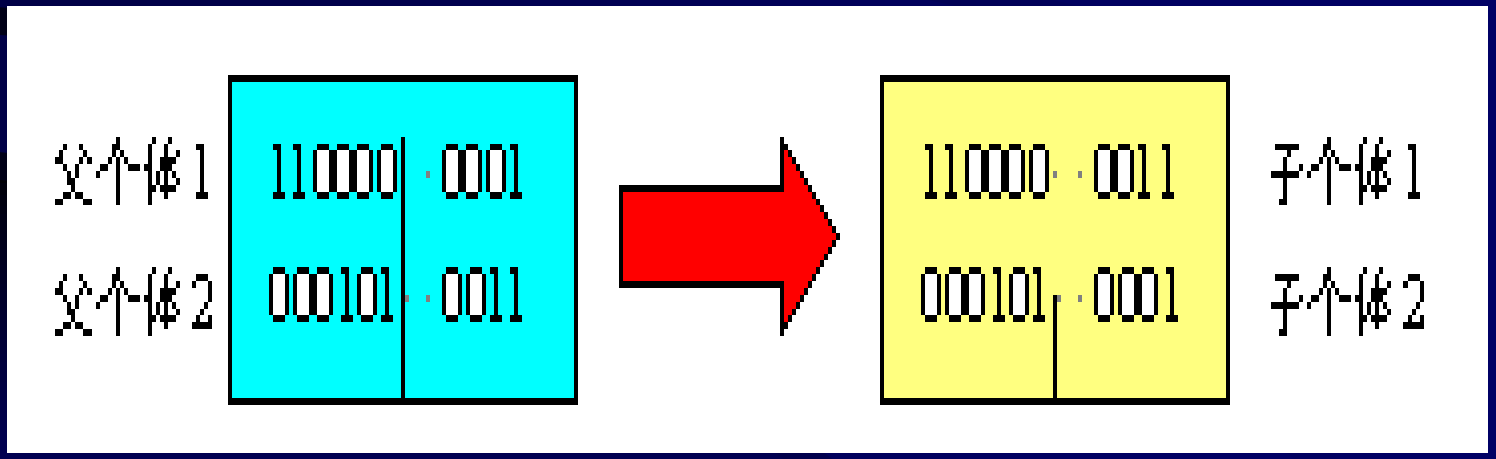
- 个体适应度按比例转化为选中概率，将轮盘分成10 个扇区，
- 因为要进行10 次选择，所以产生10 个[0, 1]之间的随机数，
- 相当于转动10 次轮盘，获得10 次转盘停止时指针位置，指针停止在某一扇区，
- 该扇区代表的个体即被选中。



轮盘赌

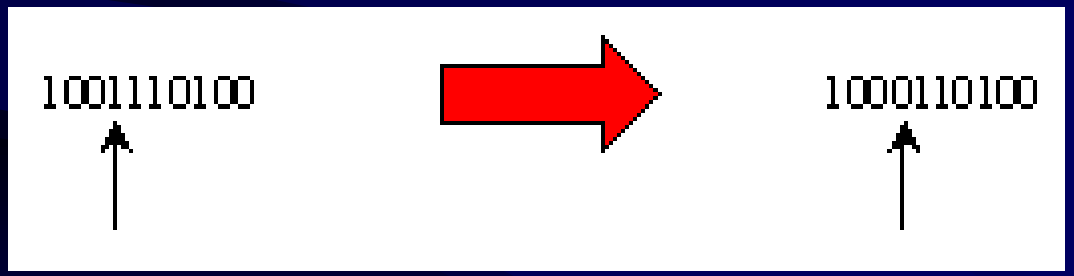
- 假设产生随机数序列为0.070 221 ， 0.545 929 ， 0.784 567 ， 0.446 93 ， 0.507 893 ， 0.291 198 ， 0.716 34 ， 0.272 901 ， 0.371 435 ， 0.854 641 ，将该随机序列与计算获得的累积概率比较，则依次序号为1 ， 8 ， 9 ， 6 ， 7 ， 5 ， 8 ， 4 ， 6 ， 10 个体被选中。
- 适应度高的个体被选中的概率大，而且可能被选中；而适应度低的个体则很有可能被淘汰。
- 在第一次生存竞争考验中，序号为2 的个体（0101111001）和3的个体（0000000101）被淘汰，代之以适应度较高的个体8和6，这个过程被称为再生。

- **交叉**：以单点交叉为例，任意挑选经过选择操作后种群中两个个体作为交叉对象，即两个父个体经过染色体交换重组产生两个子个体。
- 随机产生一个交叉点位置，父个体1 和父个体2 在交叉点位置之右的部分基因码互换，形成子个体1 和子个体2 。



交叉

- 为避免过早收敛，有必要在进化过程中加入具有新遗传基因的个体。
- 效法自然界生物变异：模仿生物变异的遗传操作，对于二进制的基因码组成的个体种群，实现基因码的小概率翻转，即达到变异的目的。



变异：以小概率决定第4个遗传因子翻转

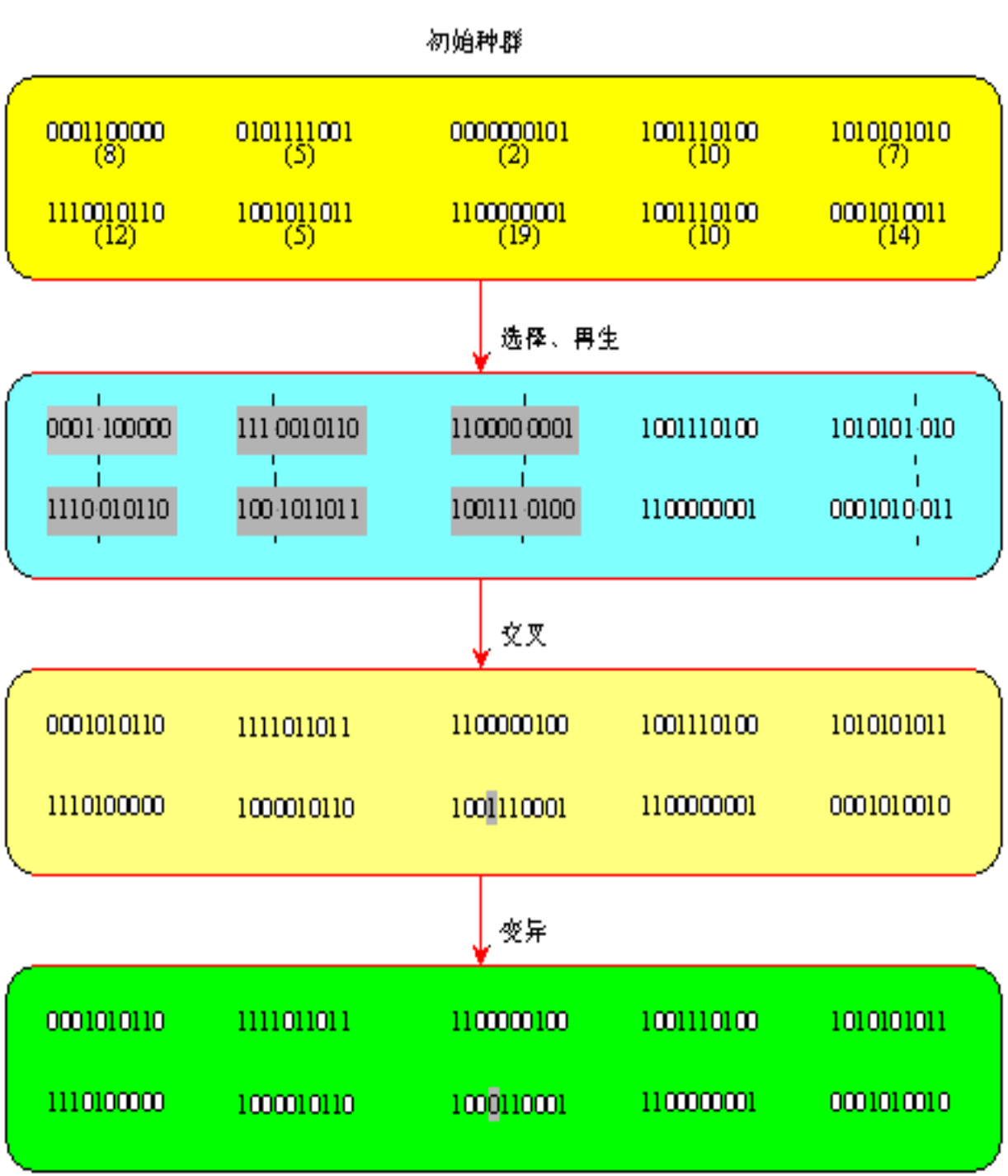
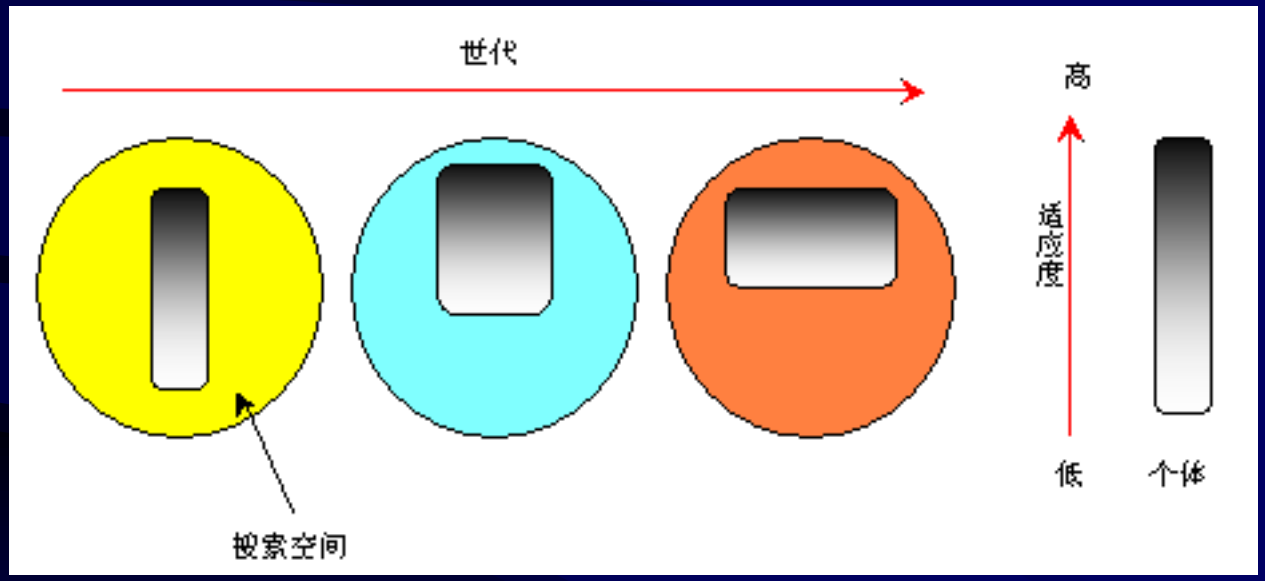


图4-9 遗传算法的进化过程

- 初始种群经过选择操作适应度较高的8号和6号个体分别复制出2个，
- 适应度较低的2号和3号遭到淘汰，
- 按一定概率选择了3对父个体分别完成交叉操作，在随机确定的“|”位置实行单点交叉生成3对子个体。
- 按小概率选中某个个体的基因码位置，产生变异。
- 形成了第一代的群体。
- 循环。

- 搜索空间中个体演变为最优个体，其在高适应度上的增殖概率是按世代递增的，图中表现个体的色彩浓淡表示个体增殖的概率分布。



遗传算法进化模式

遗传算法的流程图

第1步：随机产生初始种群，个体数目一定，每个个体表示为染色体的基因编码；

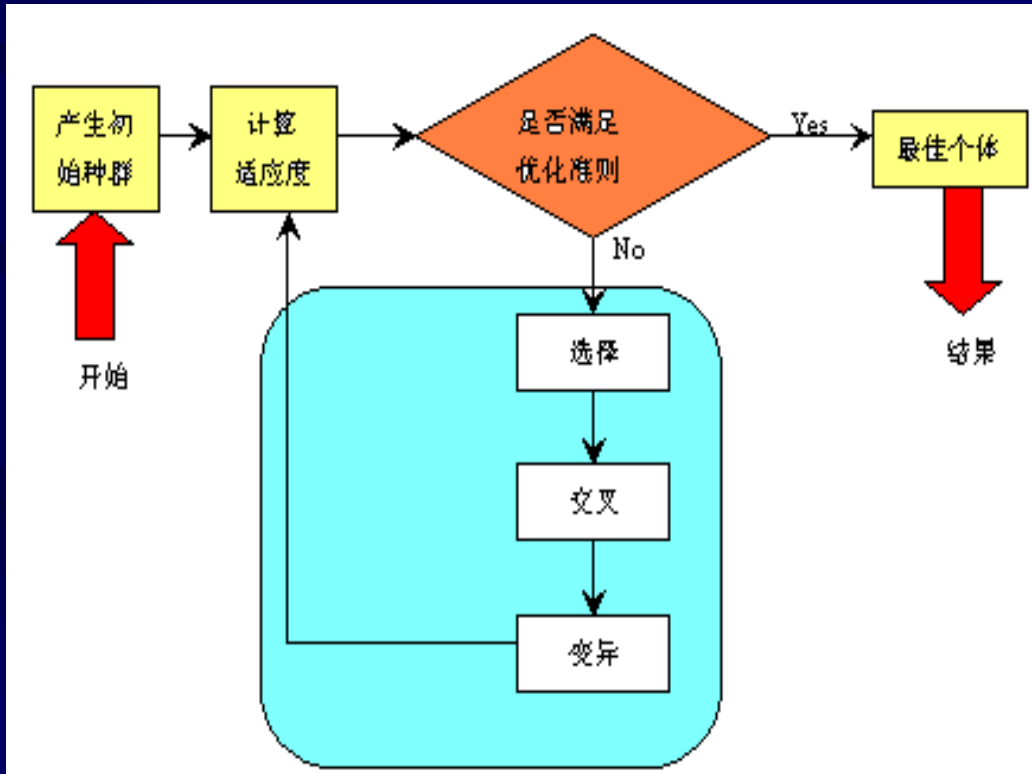
第2步：计算个体的适应度，并判断是否符合优化准则，若符合，输出最佳个体及其代表的最优解，并结束计算；否则转向第3 步；

第3步：依据适应度选择再生个体，适应度高的个体被选中的概率高，适应度低的个体可能被淘汰；

第4步：按照一定的交叉概率和交叉方法，生成新的个体；

第5步：按照一定的变异概率和变异方法，生成新的个体；

第6步：由交叉和变异产生新一代的种群，返回到第2步。



4.3.3 遗传算法的应用情况

(1) **函数优化** 对于一些非线性、多模型、多目标的函数优化问题，用其他优化方法较难求解，遗传算法却可以方便地得到较好的结果。

(2) **组合优化** 遗传算法对于组合优化中的NP 完全问题非常有效。例如，遗传算法已经在求解旅行商问题、背包问题、装箱问题、图形划分问题等方面得到成功的应用。

(3) **生产调度问题** 遗传算法已成为解决复杂调度问题的有效工具，在单件生产车间调度、流水线生产车间调度、生产规划、任务分配等方面遗传算法都得到了有效的应用。

(4) 自动控制 用遗传算法进行航空控制系统的优化、基于遗传算法的模糊控制器优化设计、基于遗传算法的参数辨识、利用遗传算法进行人工神经网络的结构优化设计和权值学习，都显示出了遗传算法在这些领域中应用的可能性。

(5) 机器人智能控制 遗传算法已经在移动机器人路径规划、关节机器人运动轨迹规划、机器人逆运动学求解、细胞机器人的结构优化和行动协调等方面得到研究和应用。

(6) 图像处理和模式识别 遗传算法已在图像恢复、图像边缘特征提取、几何形状识别等方面得到了应用。

(7) 人工生命 遗传算法已在其进化模型、学习模型、行为模型等方面显示了初步的应用能力。预见遗传算法在人工生命及复杂自适应系统的模拟与设计、复杂系统突现性理论研究中，将得到更为深入的发展。

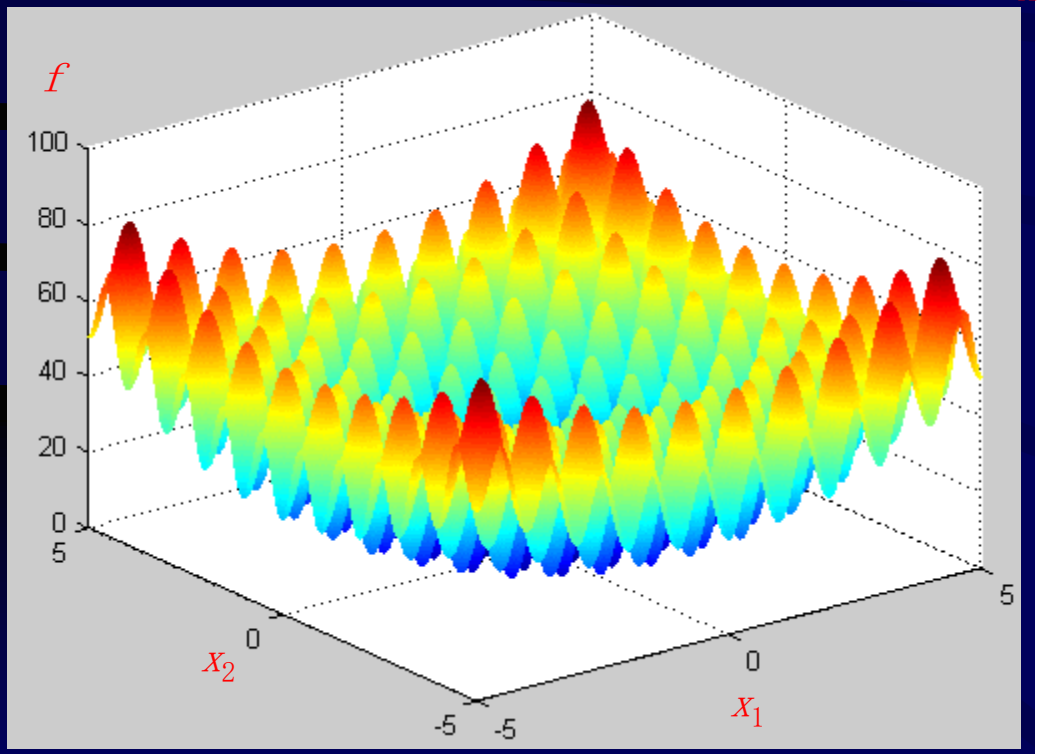
(8) 遗传程序设计 Koza 使用以LISP 语言所表示的编码方法，基于对一种树型结构所进行的遗传操作自动生成计算机程序。

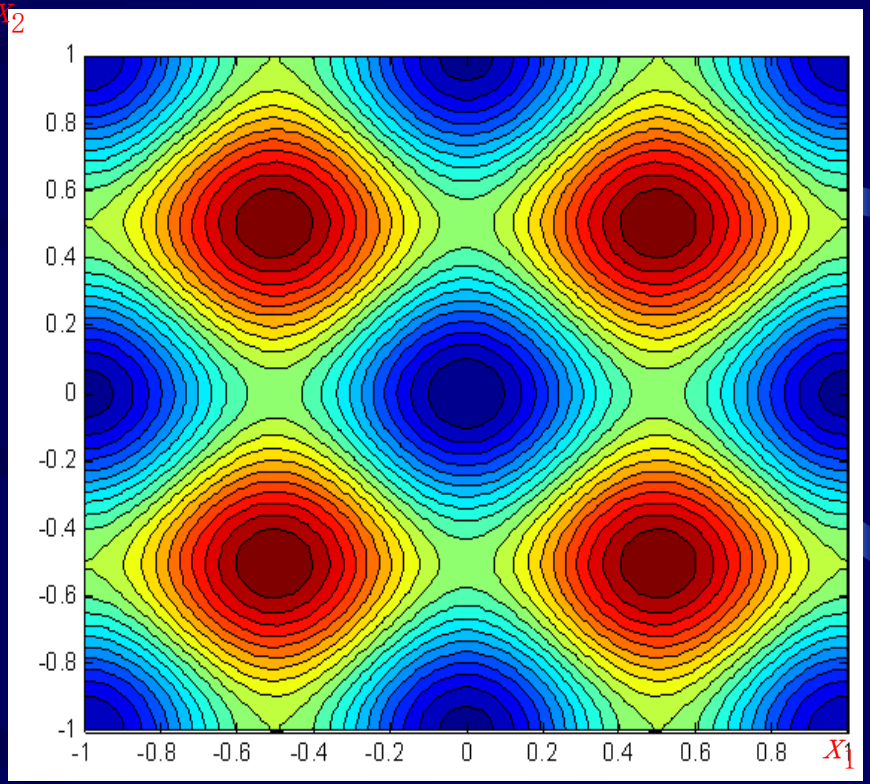
(9) 机器学习 遗传算法被用于模糊控制规则的学习，利用遗传算法学习隶属度函数，从而更好地改进了模糊系统的性能。基于遗传算法的机器学习可用于调整人工神经网络的连接权，也可用于神经网络结构的优化设计。分类器系统在多机器人路径规划系统中得到了成功的应用。

例： 用遗传算法求解下面一个Rastrigin函数的最小值。

$$f(x_1, x_2) = 20 + x_1^2 + x_2^2 - 10(\cos 2\pi x_1 + \cos 2\pi x_2)$$

$$-5 \le x_i \le 5 \qquad i = 1, 2$$



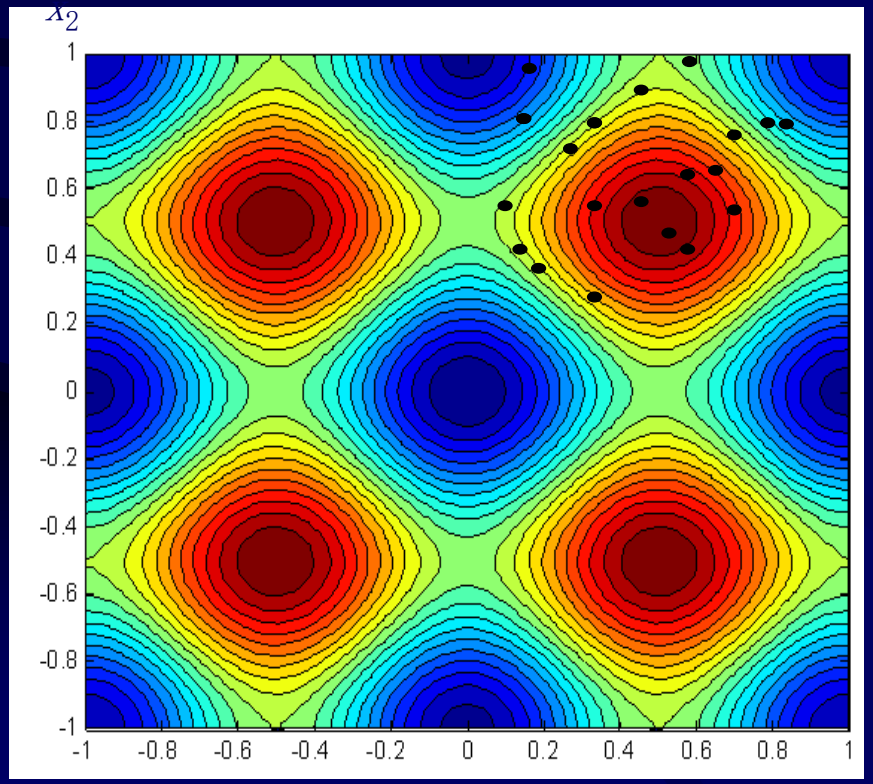


例： 用遗传算法求解下面一个Rastrigin函数的最小值。

$$f(x_1, x_2) = 20 + x_1^2 + x_2^2 - 10(\cos 2\pi x_1 + \cos 2\pi x_2)$$

$$-5 \le x_i \le 5 \qquad i = 1, 2$$

- 初始种群：

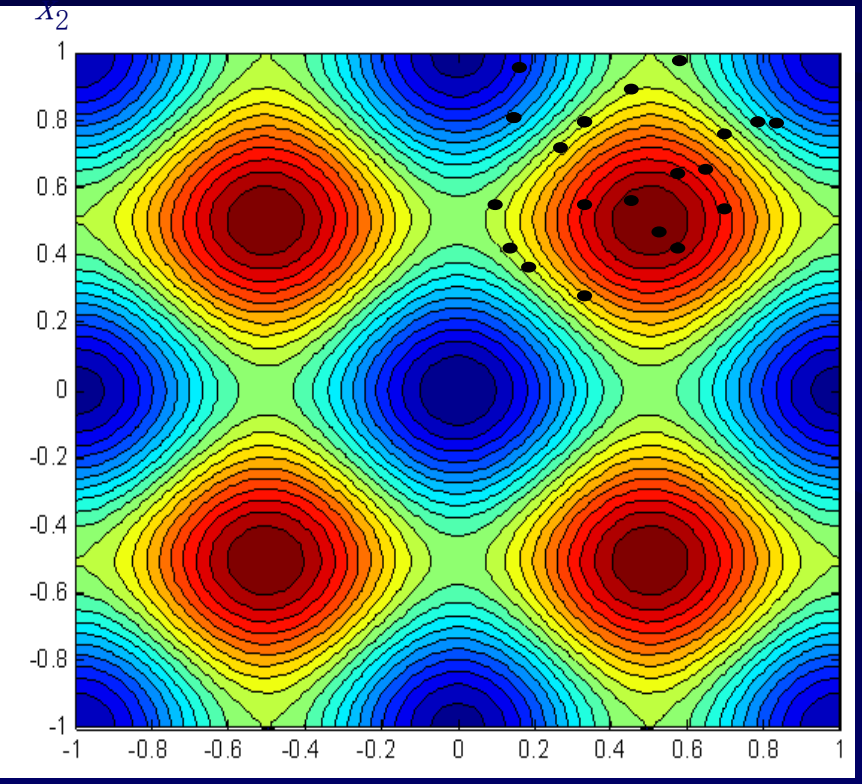


例： 用遗传算法求解下面一个Rastrigin函数的最小值。

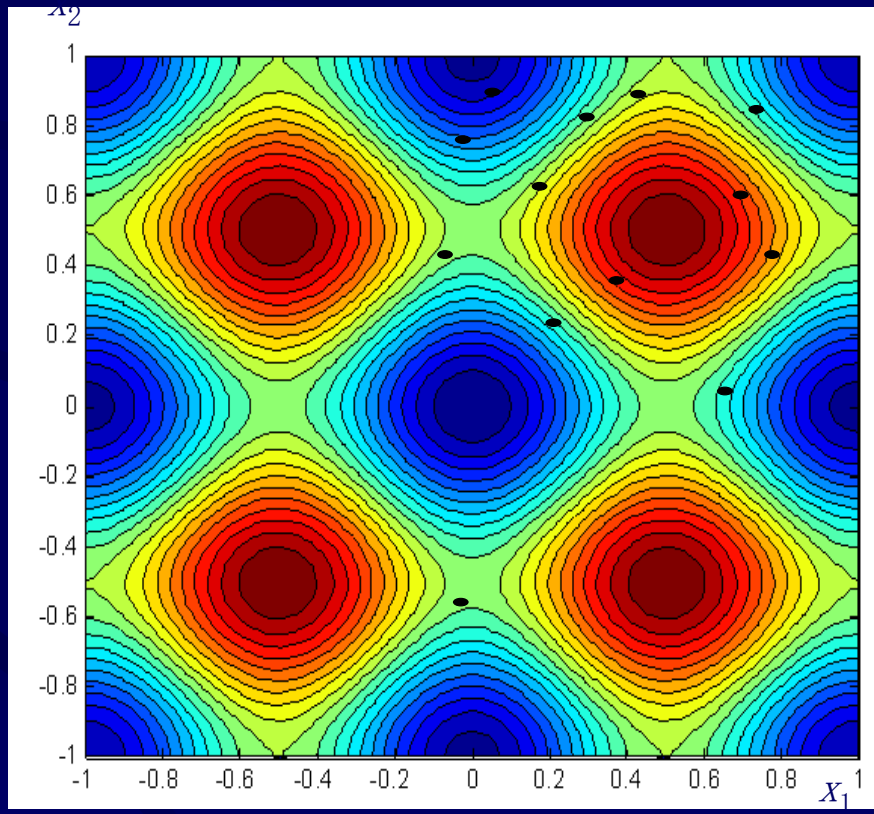
$$f(x_1, x_2) = 20 + x_1^2 + x_2^2 - 10(\cos 2\pi x_1 + \cos 2\pi x_2)$$

$$-5 \le x_i \le 5 \qquad i = 1, 2$$

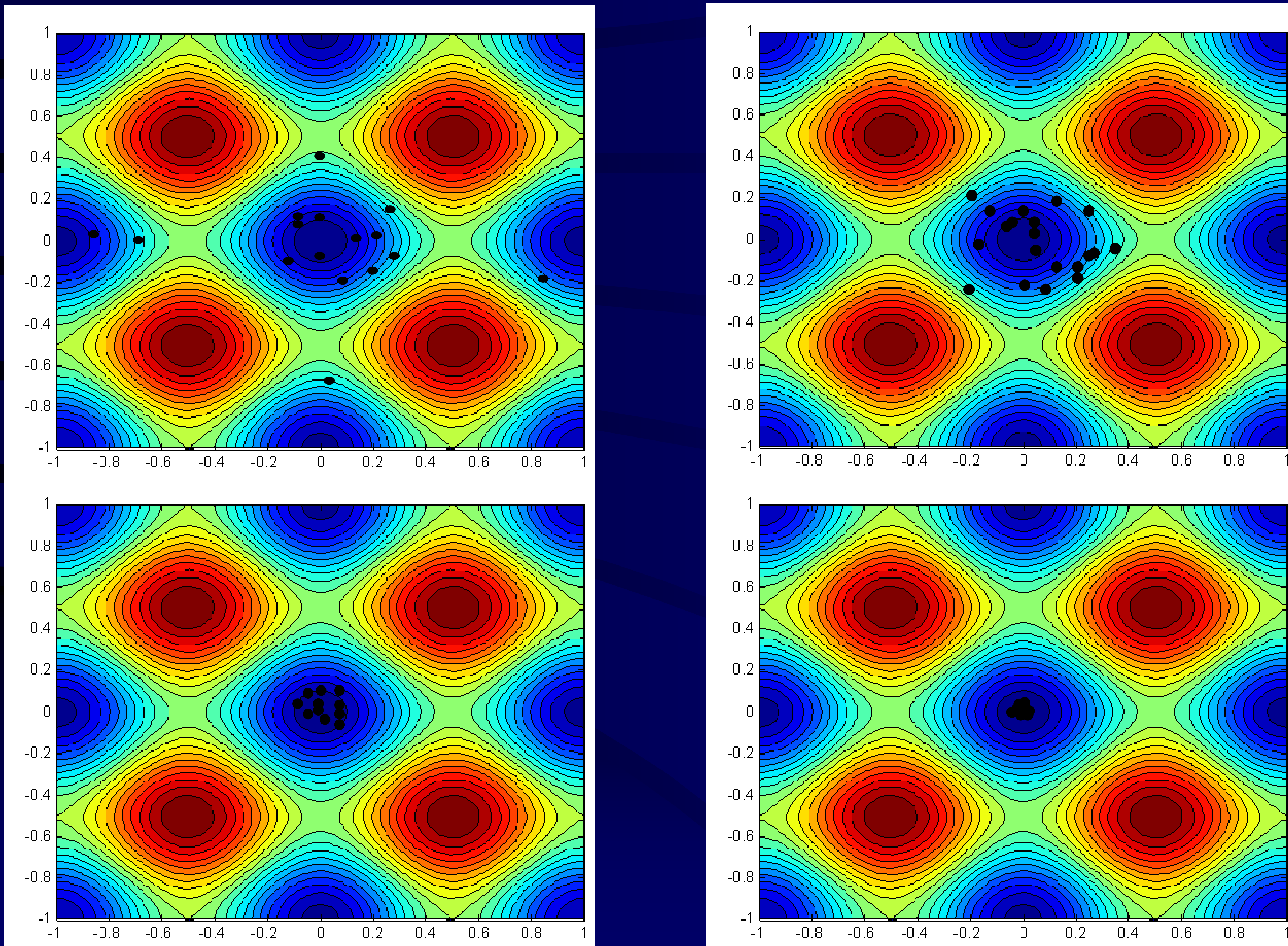
初始种群



第二代种群



➢ 在迭代60、80、95、100次时的种群



第4章 高级搜索

4.1 爬山法搜索

4.2 模拟退火搜索

4.3 遗传算法

4.4 案例分析

4.4.1爬山算法求解旅行商问题

- 设计一个解决旅行商问题的爬山算法。
- 解：假定有*k*个城市，候选解的形式为这*k*个城市构成的向量（其中不含重复的元素），记为：$s = <c_{s1}, c_{s2}, c_{s3}, \ldots, c_{sk}>$。距离矩阵Dist记录城市之间的距离。
- 定义搜索节点为Node，包含两个属性：属性*sol*记录该节点对应的候选解，属性*value*记录该候选解对应的距离代价，假定每个候选解的第1个城市都为旅行商的驻地城市。
- 定义一个评估函数evaluate()，其输入为一个Node型参数n，计算n的value值。
- 定义一个邻居节点生成函数neighbor_builder()，其输入为一个Node型参数n，返回n的所有邻居节点。

4.4.2模拟退火算法求解旅行商问题

```
cur_node, best_neighbor: Node;
1) 随机生成 cur_node 的 sol, evaluate(cur_node)
2) for t = 1 to ∞ do
3) →T = update(t)
4) →if T = 0 then return cur_node
5) →successors ← neighbor_builder( )
6) →从 successors 中随机选择一个节点，记它为 next_node
7) → ΔE← next_node.value - cur_node.value
8) →if  ΔE < 0 then cur_node ← next_node
9) →else 以概率 e^(-ΔE/T) 执行 cur_node ← next_node

function evaluate()
Input: n: Node
Output:n 对应的路径代价
int distance ← 0;
1) for i = 2 to k do
2) →distance ← distance + Dist(n.sol[i-1], n.sol[i])
3)  distance ← distance + Dist(n.sol[k], n.sol[1])
4）return distance;
```

```
function neighbor_builder
Input: n: Node
Output: a set of Node that is the neighbor of n.
neighbors = {}
neighbor : Node
1) for i = 2 to k-1 do
2) →for j = 3 to k do
3) → → neighbor ← n
4) → → 交换 neighbor.sol 中的第 i 个城市和第 j 个城市的位置
5) → → neighbors ← neighbors ∪ { neighbor }
6) return neighbors

function update()
Input: t, an integer that represents time
Output: T, an integer that represents temperature
if t = 0 then
  → T ← t0
else T ← λ^T
```

4.4.3遗传算法求解旅行商问题

```
# 设待解决的 TSP 问题有 k 个城市。
1) population ← { }
# 生成含有 10 个不相同个体的初始种群
2) i ← 0
3) do
4) → 随机生成一个长度为 k 的字符串，字符取自{0, …, k-1}且
     不重复，记此字符串为 individual
5) → if population 不含有 individual  then
6) → → population ← population ∪ { individual }
7) → → i ← i + 1
8) while (i >= 9)
9) num_iteration = 0;
10) repeat
11) → new_population ← { }
12) → for i = 1 to SIZE(population) do
13) → → parent1 ← random_select(population, evaluate())
14) → → parent2 ← random_select(population, evaluate())
15) → → child ← reproduce(parent1,parent2)
16) → → 在[0, 1]内生成一个随机数 r
17) → → if r ≤ β then
18) → → → mutate(child)
19) → → new_population ← new_population ∪ { child }
20) → population ← new_population
21) num_iteration++
22) until num_iteration ≥ 100000
```

```
function random_select
Input:
population, 种群集
evaluate, 评估个体的适应度函数，见 4.3.1 的解答
Output: individual, 被选择的个体
sum ← 0
for i = 1 to SIZE(population) do
  → evaluate(population[i])
  → sum ← sum + population[i].value
for i = 1 to SIZE(population) do
  → 生成一个[0,1]内的随机数 r
  → if (r < (population[i].value / sum)) then
  → → individual ← population[i]
  → → return individual

function reproduce
Input: x, y, 两个个体
Output: x 和 y 交叉得到的下一代个体 z
n ← LENGTH(x); c ← random number from 1 to n
return APPEND(SUBSTRING(x,1,c),SUBSTRING(y, c+1, n))
```

从群体智能到元启发：自然启发式优化算法

IEEE Computer计算科学评论
NOVEMBER 2016，第4期：52-60

杨新社（Xin-She Yang），苏阿什·戴布（Suash Deb），密德萨斯大学（Middlesex University）

在新型最优化算法中，有一类算法的基础是群体智能（swarm intelligence, SI）。

SI的设计思想是：群体（蚂蚁和蜜蜂等）中的有机个体，根据区域信息、行为主体（agent）之间的交流及其自身环境做出的决定，是群体智慧（collective intelligence）或社会智慧（social intelligence）的起源。

初始化种群和迭代计数

评估初始种群并找到最优解

while（未满足停止条件）

修改局部或全局现有的种群，生成新解

评估新解

if 新解更好，then接纳新解，更新种群

if 新解不好，then依照某些概率性准则接纳新解

更新迭代计数

结束

输出过程结果

【典型自然启发算法的伪代码】

如上页伪代码所示，所有自然启发算法都包括探索（exploration）和利用（exploitation）两种功能。为更有效地检索设计空间，备选解种群（population of candidate solutions）必须具有一定的差异性。

另一方面，为使算法以最少的迭代快速收敛，必须使用解空间信息（梯度、模型形状等）引导程序在可能性最高的区域进行搜索。

利用这类信息可以加快收敛的速度。但是，这样也会导致解种群差异性增长过快，引起所有解都具备相近的概率，致使程序崩溃。

算法类型

1. 蚁群优化算法（Ant colony optimization）
2. 蜂群优化算法(Bee colony optimization)
3. 蝙蝠算法(Bat algorithm)
4. 布谷鸟搜索算法（Cuckoo search）
5. 粒子群优化算法（Particle swarm optimization）
6. 萤火虫算法（Firefly algorithm)
7. 花朵授粉算法（Flower pollination algorithm）

其它这类算法还有很多很多，如引力（gravitational）、和声（harmony）、狼群（wolf)搜索算法，以及以生物地理学和免疫系统为基础的最优化算法。

蚁群优化算法（Ant colony optimization）

蚂蚁是一种社会性昆虫，生活在有组织的社群当中，单个社群的蚂蚁数量可以达到2,500万之多。蚂蚁通过一种可以指示自身存在的化学物质进行交流，这种物质称为信息素。任何蚂蚁都能跟随其它蚂蚁的信息素踪迹并留下自己的气味信息素，完成相互之间的交流。在踪迹选择中，信息素沉积（deposition）相当于正反馈机制；信息素消散（evaporation）相当于回避机制（escape mechanism）。

蚁群最优算法的提出是在1992年，它利用局域互动与信息素变化，从本质上模拟了蚂蚁的主要行为和社会特征。“它在处理最优化问题方面出乎意料地有效，不论是车辆路径问题（vehicle routing）还是著名的旅行推销员问题（traveling-saleman Problem，TSP）都可以解决。算法将蚂蚁行走的路径编码为实际路径（actual path）。蚂蚁在多条实际路径的交汇点做出的选择，取决于路径上信息素的浓度。路径上有信息素沉积说明这条路比较好；而信息素消散则确保早期搜索阶段中，那些不太好的路径不会很快地收敛。

蜂群优化算法(Bee colony optimization)

单个蜜蜂社群一般由2万到8万只蜜蜂组成，其中有一只蜂王和几百只雄蜂，其它都是雌性工蜂。工蜂会进行侦查、观望、守卫和采集花蜜等活动。侦查的工蜂发现新的花蜜源后，会用一种上下飞舞的舞蹈通知其它工蜂。2005年提出的人工蜂群优化算法，就模拟了蜜蜂搜索行为的主要特征。这个算法将蜜蜂按状态分为雇佣（employed）、侦查和观望（onlooker）三种，以花蜜源为解矢量（solution vector），并将其与目标解空间联系起来。虽然这个设定有些简单，但它抓住了蜜蜂搜索行为的主要特征。这个算法被用于训练神经网络，以及解决无约束数值优化问题和约束优化问题。

蝙蝠算法(Bat algorithm）

小型蝙蝠共有八百多种，其中大部分都用回声定位进行导航，为此蝙蝠需要每秒发出10到20次超声波脉冲，每个脉冲的持续时间只有几千分之一秒，频率在20到200千赫兹之间（人耳最高只能听到20千赫兹），音强可以达到120分贝（相当于大型喷气式客机起飞时的噪音强度）。蝙蝠发现昆虫后，在定向捕猎飞行中，超声波脉冲的发射频率上升到每秒200次，声波频率也更高。此时蝙蝠可以更准确地判断飞虫的大小、位置、飞行范围、速度和方向。

蝙蝠算法提出于2010年，利用了蝙蝠发射超声波脉冲和频率调节的特点，将蝙蝠的位置作为搜索空间（search space）中的解矢量。蝙蝠通过调节超声波频率，可以搜索范围更大的空间；而提高脉冲发射频率，则可以锁定附近区域中可能出现的局部解。在整个蝙蝠群中，有一个全局最优解，其它蝙蝠都趋向于飞向这个位置。因此，算法的收敛速度相对较快。这个算法由声波频率、脉冲发射频率及音强三个因素控制，可以应用于许多实际应用当中，如解决工业最优化问题、训练神经网络、图像处理以及解决TSP问题。

布谷鸟搜索算法（Cuckoo search）

一些布谷鸟很擅长巢寄生，它们将蛋生在其它鸟（比如林莺）的巢里，然后由宿主孵化喂养。这是因为布谷鸟的蛋可以模仿宿主鸟蛋的质感、颜色和大小，效果十分接近。即便如此，有些宿主还是可以认出布谷鸟的蛋，然后将它们踢出巢外或抛弃自己的鸟巢。这就在两个物种之间形成一种进化上的军备竞赛。

布谷鸟搜索算法提出于2009年，它将布谷鸟的蛋作为解矢量，筑巢区作为搜索空间。证据表明，布谷鸟和宿主鸟的飞行路线都遵循列维飞行（Lévy flight）模式，即偶尔的长距离飞行伴随着局部的随机行走，这使得大范围的搜索更加有效。鸟蛋相似可以转化为解相似，有助于迭代搜索过程达成收敛，而发现概率（discovery probability）则有利于全局探索。布谷鸟算法在工程最优化问题和图像处理问题中都有成功应用。

粒子群优化算法（Particle swarm optimization）

鸟和鱼类在移动时往往聚成一群，数量带来安全是其原因之一。每只鸟都遵循简单的飞行规则，只会追踪临近七只鸟的飞行状态。虽然飞行运动受牛顿力学支配，但奇怪的是，鸟群中的鸟却几乎完全不会相撞。从这类群体中，可以看出一些组织结构。

粒子群优化算法提出于1995年，它的基础就是上面提到的简单规则以及群体运动的特征。它以粒子的位置为解矢量，因此每个粒子都有历史最优位置，而当前的最优解由整个粒子群产生。粒子在上述简单规则下更新它们的位置和速度，而全体粒子都倾向于向质心（centroid）移动，那里是全局最优解。这个体系能在很多情形下迅速收敛。一方面，快速收敛使这个方法成为有效的优化控制器（optimizer）；另一方面，收敛的时机可能会太早，从而导致早熟收敛（premature convergence）。因为我们有许多解决早熟收敛问题的方法，所以这个算法有不少变种。粒子群优化算法几乎在科学和工程的每个领域都有应用，包括设计最优化问题、图像处理和调度问题（scheduling）等。

萤火虫算法（Firefly algorithm)

热带地区的萤火虫以生物发光的方式进行交流，每类萤火虫都有特定的闪光模式，作为各自的信号系统。萤火虫会被同类的闪光吸引，并向发光方向飞去。因为光强会随距离增加和空气污染而减弱，所以萤火虫只能看见几百米之内的闪光。

萤火虫算法提出于2008年，它的基础是上述闪光特征。萤火虫的位置对应于解矢量，目标解空间决定发光的亮度或吸引力。因为短距离的光比长距离的光更有吸引力，所以整个萤火虫算法可以自动分割为许多子群体（subgroups或subswarms），每个子群体在解空间的局部模型附近运动，形成峰（最大值）和谷（最小值）。因此，这个算法能同时找到多个最优解。在一定条件下（如正确的吸引范围及随机性单调递减），它的收敛速度可能比遗传算法（genetic algorithm）和粒子群算法都更快。萤火虫算法也和粒子群算法一样，从调度与分类问题，到图像处理、工业设计最优化问题等领域，都拥有广泛的应用。

花朵授粉算法（Flower pollination algorithm）

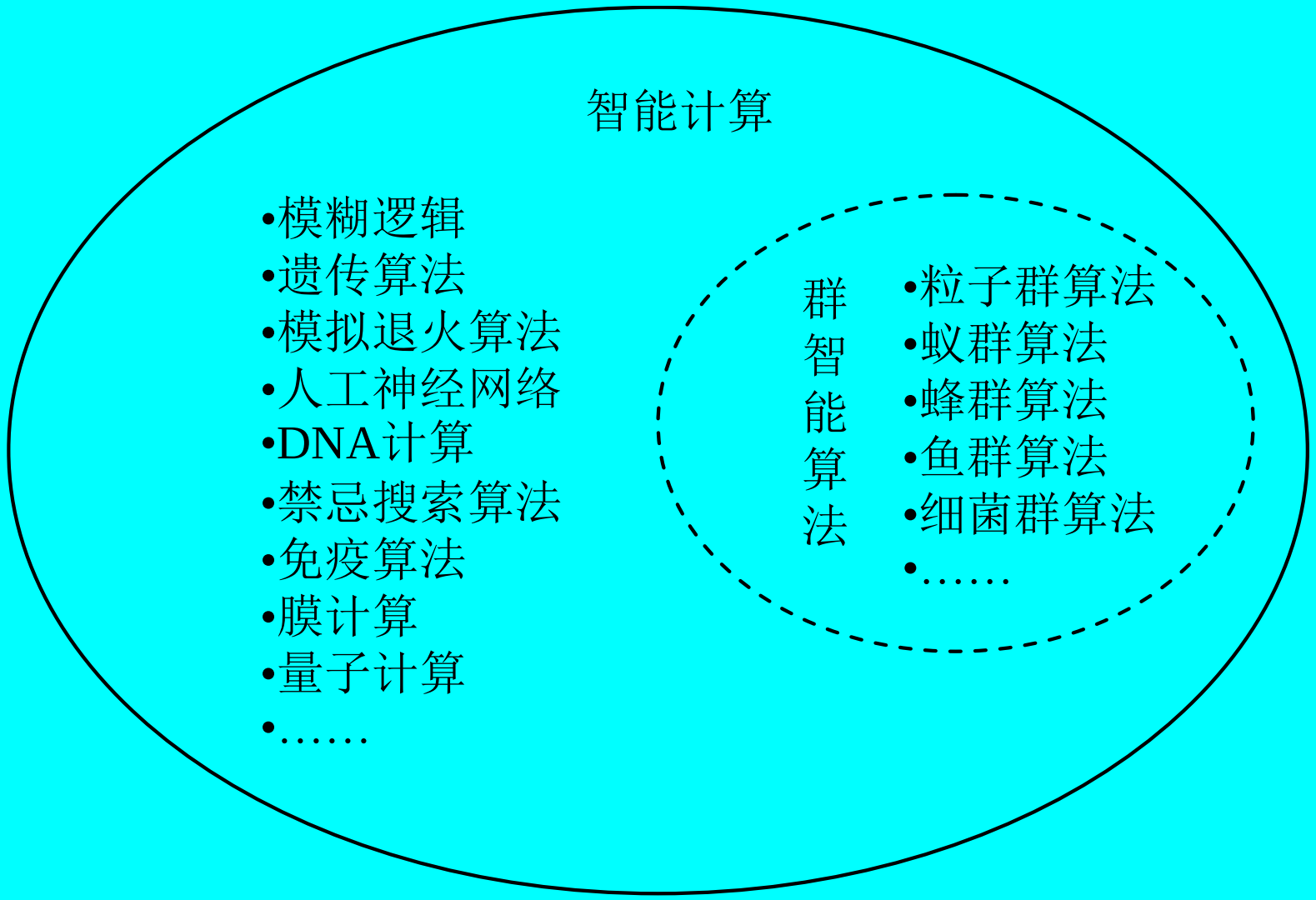
显然，不是所有算法都以群体行为作为基础。植物同样提供了许多巧妙的模型，比如授粉过程。在25万种开花植物中，有九成是生物授粉，靠蝙蝠、鸟类、蚂蚁和蜜蜂之类的动物传递花粉。某些传粉生物（如蜂鸟）只会接近特定种类的植物，它们具有访花恒定性（flower constancy）。剩下一成开花植物靠非生物授粉，花粉由风或水传播，局限在一定范围内。相比之下，生物授粉是全局性的，因为传粉生物可以移动很长的距离。

花朵授粉算法提出于2012年，以花粉配子作为问题的解矢量，用生物传粉来模拟全局搜索，而用非生物传粉模拟局部搜索。全局与局部的转换由一个概率控制，这个概率表示生物传粉与非生物传粉比例。这个算法被用于解决多目标最优化问题。

杂合（Hybrid）算法或许是个有趣的选择，因为它融合了多种算法的优点。如何设计一个更好的杂合算法本身就是一个更高层次的最优化问题——最优化算法的最优化问题。发展杂合算法不能只让它比非杂合算法的效率和稳健性更高，研究者必须跳出这种潮流。杂合算法的比较研究应该只涉及杂合之后的算法。

在物联网时代，算法必须具备自适应能力和智能。为一个特定问题找到最佳算法的应用程序太多了。相比之下，自适应方法能够自动选择适合给定任务集的算法，在没有或极少的用户干预下执行任务。自适应也意味着可以控制自身性能并自动调节参数，保证程序高效运行。最终目标是开发出一个智能工具箱（intelligent toolbox），只要用户按下开始按钮，就能解决所有给它的问题。

群智能算法


智能计算
•模糊逻辑
•遗传算法
•模拟退火算法
•人工神经网络
•DNA计算
•禁忌搜索算法
•免疫算法
•膜计算
•量子计算
•……
群智能算法
•粒子群算法
•蚁群算法
•蜂群算法
•鱼群算法
•细菌群算法
•……

THE END