Genetic Algorithm(遗传算法)

这种算法的思维来自于进化论

一些基本概念:

- Population: individual(个体)的集合
- Selection: 越好的个体被选择的可能性越大
- Crossover: 字染色体的产生
- Mutation: 染色体在随机一定条件下会随机变异

一般情况下,会把搜索集的个体进行编码,主要是二进制编码.然后从种群中选择个体, 进行交叉, 变异, 形成新的种群.直到达到停止条件.

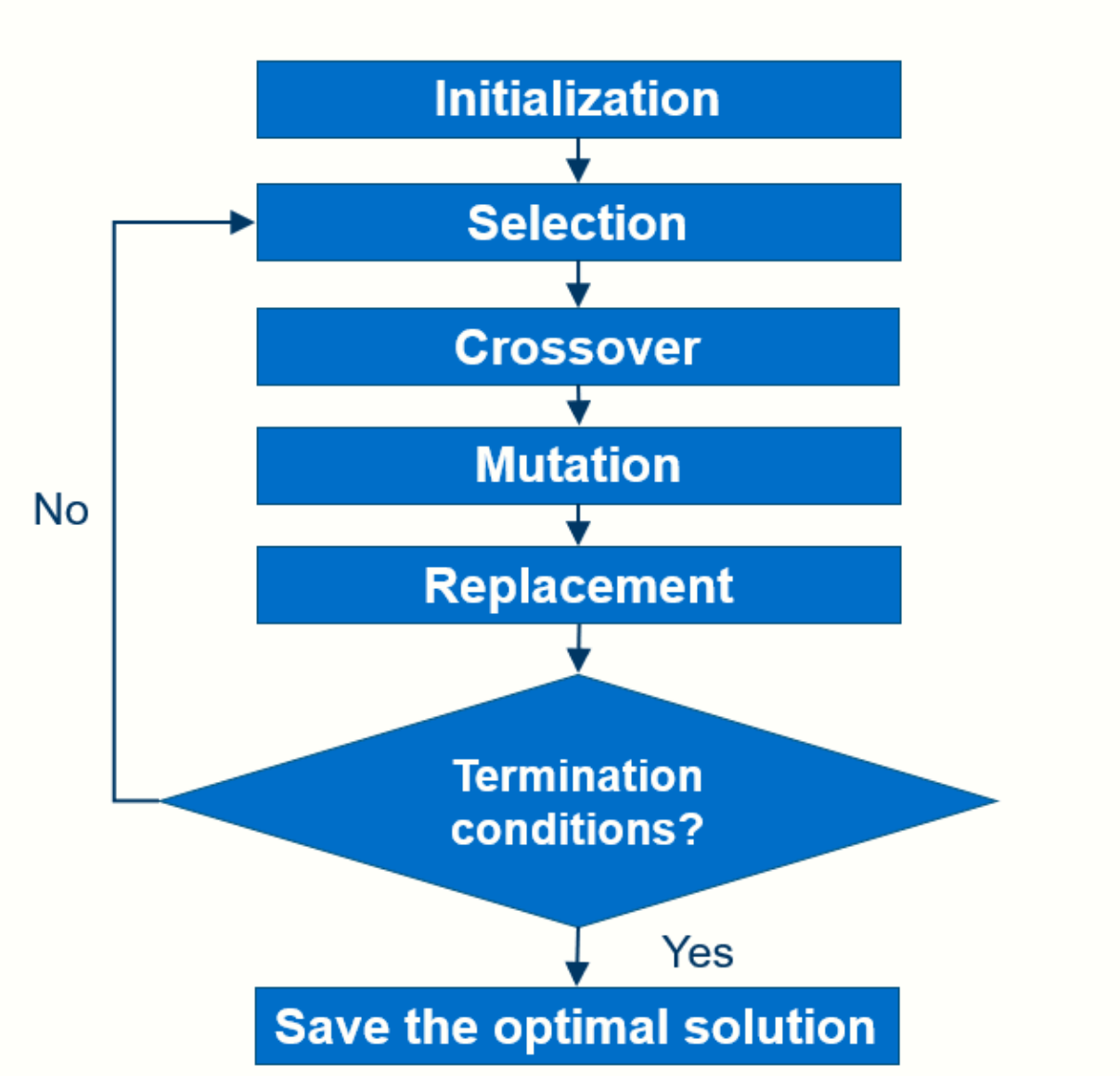

Figure 1: 遗传算法的流程

编码

用于表示个体的一种数据结构, 最常见的编码方式是用二进制编码.

初始种群(initial population)

- Population size:种群里有多少个个体
- The initialization method: 如何得到初始种群的?

Selection(选择)

类似于生物进化,依据适应性选择,更好的个体被选择的几率越大,由 fitness function 来计算适应度

$$prob(x_i) = \frac{G(x_i)}{\sum_1^{pop\_size} G(x_i)}$$

Figure 2: 选择的概率计算, 计算适应度占总和的比例

这种计算方式就像抽奖转盘一样, 所以叫做轮盘赌选择(Roulette Wheel)

具体如何用计算机实现?

G_n代表个体 n 的适应度 $S_1 = G_1$,$S_2 = G_1 + G_2$,$S_3 = G_1 + G_2 + G_3$, 直接从 0 到 S_3 中取得一个随机数, 在$[0, S_1]$就选个体 1, 在$[S_1, S_2]$选个体 2,在$[S_2, S_3]$就选个体 3.

更多的个体能够被选择带 mating pool(交配池), 负面的效果是减少了多样性(diversity).不断的选择只能找到初始种群中最好的个体,不能真正解决优化问题.

Crossover

这种操作一般设计为和适应度无关, 一般分为:

- One-Point Crossover

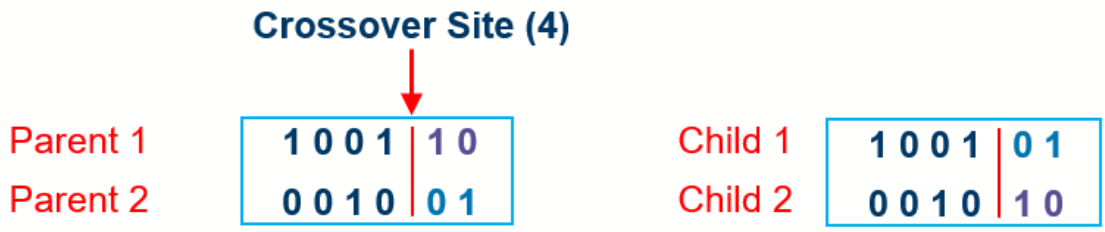

Figure 3: 单点交叉

- K-Point Crossover

找到 K 个分割点(Crossover sites),然后交换其中的子串.

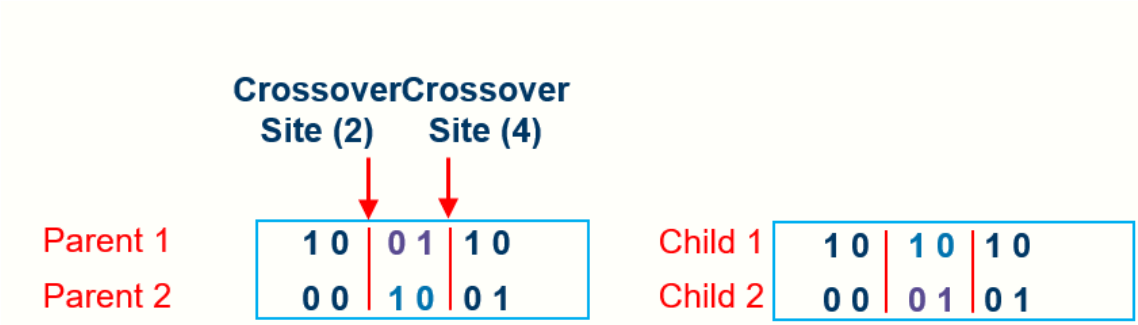

Figure 4: K=2

p_c:交叉率, 在交配池中的一对染色体发生交叉的概率, 一般比较高(0.6-0.9)

- 父母:parents
- 子代:offspring

Mutation

较低的变异概率$P_m \approx 0.01 \sim 0.001$

停止条件

- 达到最大代数
- 得到了满足要求的解
- 种群收敛(整个种群（population）的个体变得越来越相似，最终可能集中在少数解（或单个解）附近的现象)

遗传算法的优点

- 通用的优化算法
- 种群中的候选解可以并行评估
- 遗传算法通过进化过程能够发现多样化的解,从而在全局搜索中避免局部最优问题
- 过程实现起来不难

遗传算法的缺点

- 它可能不会得到一个非常高质量的最优解
- 该算法有几个参数，这些参数的选择影响着解的质量
- 速度慢

常用编码方式

- Binary(二进制字符串)
- Real vector(实数向量, 适合处理连续优化问题)
- 排列向量(表示特定顺序的)

初始化

对于二进制编码,可以用 0,1 以各 50% 的概率随机生成.
对于实数表示, 在可行区间内随机生成.
尽量平均化, 解决最优的答案
根据相关领域的特定知识

变异

- 变异操作要使得理论上所有在搜索集里的结果都有可能出现
- 变异的强度(变异后和变异前的差异)
- 变异后的结果必须是有效的(valid)

二进制:0-1 交换
实数变异可以用随机加上一个符合正态分布的偏置值的办法解决
对于排列,可以随机交换两个.

Crossover

本质上:自带要继承一部分父代的东西

- 产生的子代要是有效的(valid)
- 最好对于具体问题设计算法

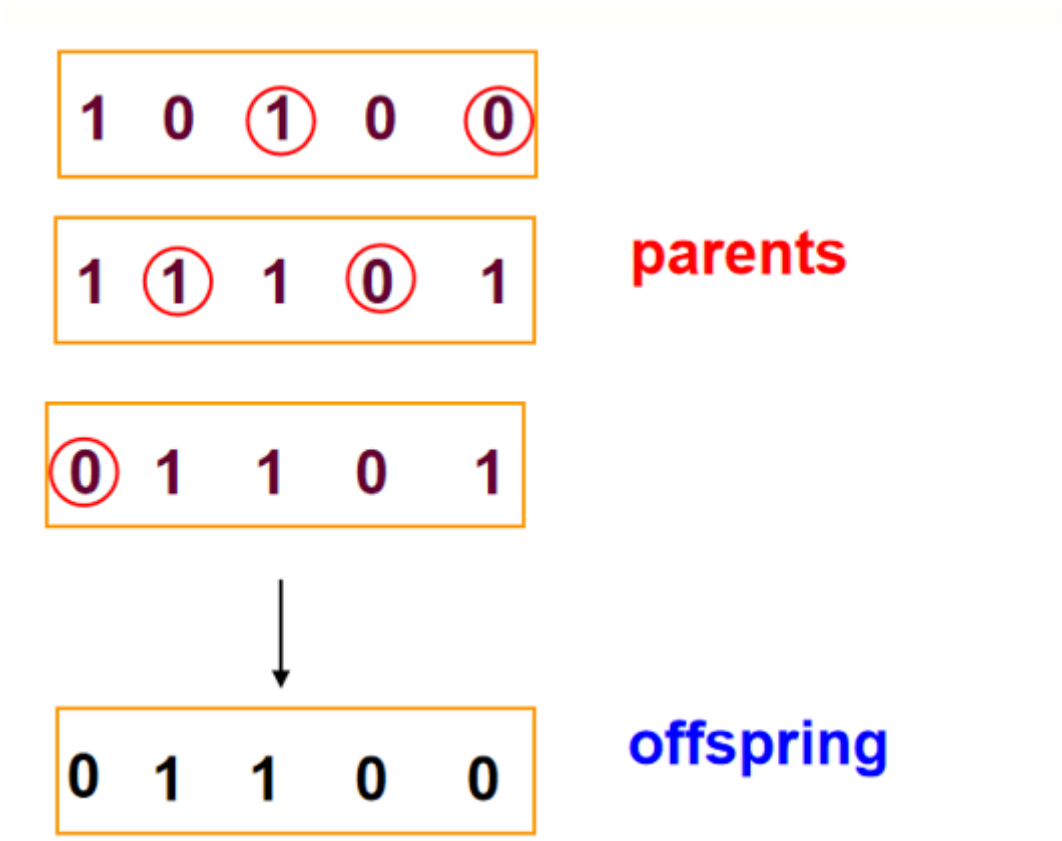

Figure 5: 一种产生子代的方式

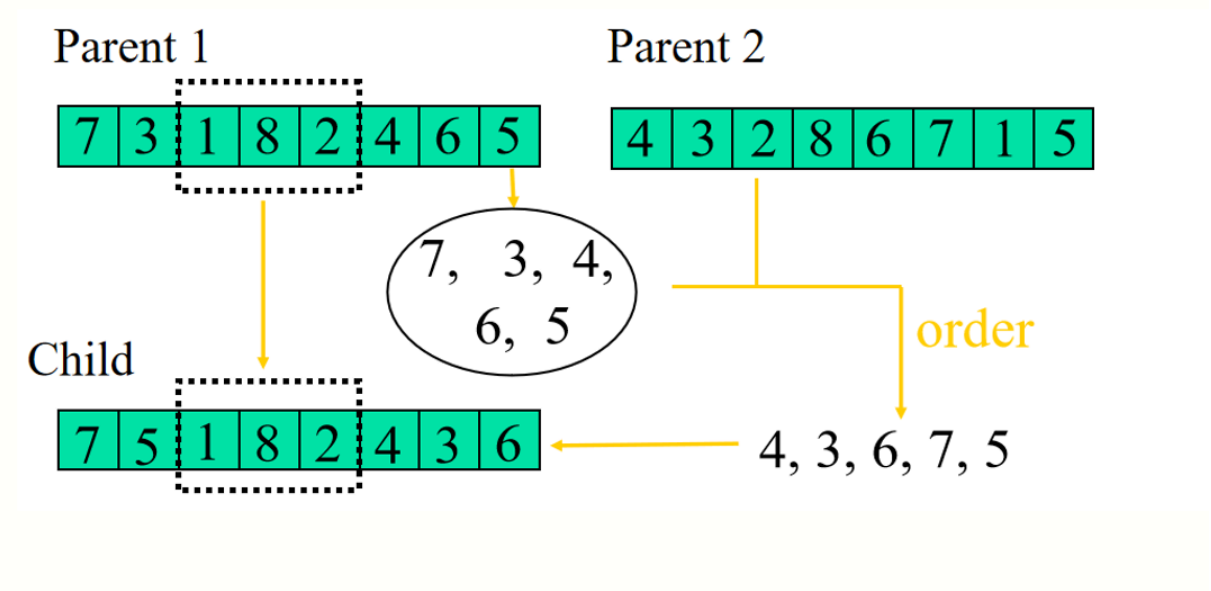

Figure 6: 一种从排列产生子代的方式

选择

Figure 7: 计算在交配池中某一父代的平均个数

轮盘赌选择法的缺陷:

- 难以选择不好的个体, 交配池很快就被优秀个体占据了(对每个适应度加一个较大的值可以减少他们之间被选中概率的差异)
- 如何这些个体的适应性差不多, 难以体现选择性(对每个个体减一个值可以增大它们被选择概率的差异)
- 可以用适应度函数的值进行排序, 把排名直接作为计算概率的标准
- Tournament selection:随机选 k 个个体, 然后从这 k 个中选择最好的

Replacement(新旧替代)

- 旧种群完全由新种群替代
- 部分替代, 子代和父代竞争, 更加优秀的替代原来的

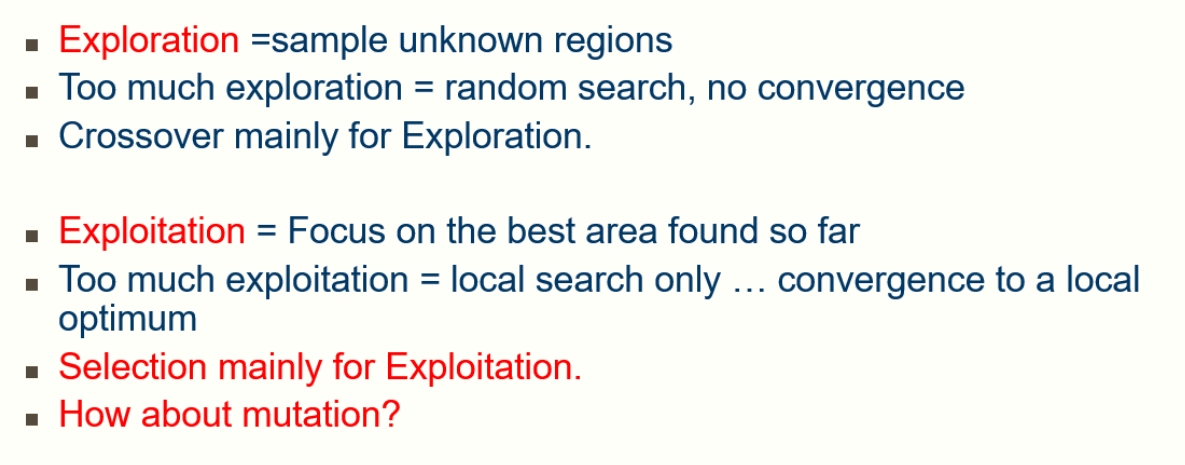

Figure 8: 从探索与开发方面理解

有约束的优化 Constrained Optimization

- feasible(可行), infeasible(不可行), 由约束条件决定

比较规则

1. 可行解一定比不可行的好
2. 可行的当中, 让目标函数最优化的更好

3. 不可行当中,违背约束条件越小的越好
4. 多个约束条件可能有优先级的区别

如何处理约束条件

- 惩罚函数

比如:

$$\min f(x)$$

$$g(x) \leq c$$

$$\min f(x) + k * h(g(x), c)$$

$$h(g(x), c) = \max\{0, g(x) - c\}$$

k是惩罚系数,用于控制惩罚的强度

- 决定惩罚系数k

这张图描述了一种经验方法，用于确定惩罚系数 (k)。具体步骤如下：

Step 1: 确定可接受的误差 **(\epsilon)**
首先需要估计一个可接受的误差 (\epsilon)，即问题中可以容忍的一种微小的约束违反值。在现实问题中，这个值通常不会为零，而是一个略微大于零的值。

例如：在约束优化中，有时我们无法完全满足约束条件，可以允许一个很小的误差。

Step 2: 估计目标函数的期望最优值 **(ilde{f})**
接下来估计目标函数的期望最优值 (ilde{f})，即根据问题对目标函数的预期，这个值不一定是精确的，但应接近真实问题的目标值范围。

Step 3: 确定惩罚系数 **(k)**
为了让 (k \cdot \epsilon) 的数量级与目标函数的期望最优值成合理比例，这里设定 (k \cdot \epsilon) 的大小是 (ilde{f}) 的 50 倍。因此：

[k \cdot \epsilon = 50 \cdot ilde{f}]

从而可以解出 (k) 的计算公式：

[k = \frac{50 \cdot ilde{f}}{\epsilon}]

总结：
惩罚系数 (k) 的大小由以下因素共同决定：

1. 可接受误差 **(\epsilon)**：越小的 (\epsilon) 会导致 (k) 越大，从而对违反约束的惩罚更为严厉。
2. 目标函数期望值 **(ilde{f})**：越大的 (ilde{f}) 会导致 (k) 增大，从而让惩罚项和目标函数在数量级上保持一致。
3. 比例因子 **50**：这里固定为一个经验值，表示惩罚项的量级应为目标函数的 50 倍。

通过这种方法，可以合理选择 (k) 的值，使得约束违反的影响能够与目标函数的优化平衡起来。