Lecture4-2 启发式与元气启发式算法

1.可接受启发式算法 Admissible heuristics

只有启发式算法具有“良好的品质”时，它可以是强大的

一个好的启发式算法是**可接受的 admissible**

一个**可接受的**启发式永远不会高估达到目标的成本，也就是说它是**乐观**的，一个启发式算法是可接受的当

$$\forall node\ n, h(n) \leq h^*(n)$$

- 其中，$h*$ 是真实的从节点 n 到目标的开销

贪心算法是可接受的

上文中贪心算法里的 h_{SLD} 是可接受的，因为它是两个点之间的最短路线距离（一定小于等于开销）

A* 算法是可接受的

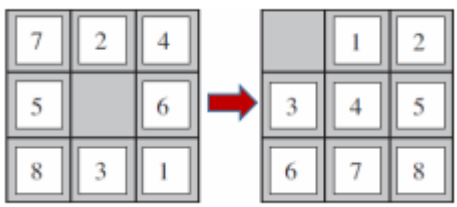
证明：

- 假设 G_o 是最优路径的目的地节点
  假设 G_S 是次优路径的目的地节点
  假设节点 n 是到 G_o 最短路径上存在的一个节点
- 因为 $h(G_S) = 0$，所以 $f(G_S) = g(G_s)$
  因为 $h(G_o) = 0$，所以 $f(G_o) = g(G_o)$
  因为 G_S 是次优的，所以 $g(G_s) > g(G_o)$
  所以 $f(G_s) > f(G_o)$
- 因为 h 是可接受的，所以 $h(n) \leq h^*(n)$
  $g(n) + h(n) \leq g(n) + h^*(n) = g(G_o) = f(G_o)$
  所以 $f(n) \leq f(G_o)$

从 $f(G_s) > f(G_o)$ 和 $f(n) \leq f(G_o)$ 中可以推出，A* 算法永远不会在搜索中选择 G_s 的走法，所以 A* 算法是最佳的

2. 启发式算法的搜索效率

回顾 8-拼图问题



有两个属性可以选择

- $h_{mis}(s)$：没有对准的方片个数 ∈[0,8] 是可接受的
- $h_{1stp}(s)$：到目标情况时所移动的方片次数

因为 $h_{1stp}(s) \geq h_{mis}(s)$ 所以选择 $h_{1stp}(s)$ 作为启发式比选择 $h_{mis}(s)$ **更好**

主导 dominate

对于两个**可接受**的启发式 h_1 和 h_2，如果对于 $\forall s$，$h_1(s) \geq h_2(s)$，那么我们说 h_1 **主导 dominates** h_2，且对于**搜索的效率更高**

对于任何**可接受**的启发式 h_1 和 h_2，定义 $h(s) = max\{h_1(s), h_2(s)\}$，此时 $h(s)$ 是**可接受的**，并且主导 h_1和 h_2

搜索效率的量化

有效分支因子 b^* 量化启发式方法的搜索效率

- 对于A^* 算法找到的最优解，计算有效分支因子b^* 满足$N = b^* + (b^*)^2 + \ldots + (b^*)^d$
- N：找到解前搜索的节点数量
- d：解在搜索树中的深度
- 例如：A^* 算法在搜素了 52 个节点后找到了深度为 5 的节点，则 $b^* = 1.92$

良好的启发式算法的 b^* 接近 1，以合理的计算成本解决了很大的问题

生成可接受的启发式算法

如何生成“好的”启发式方法?

从松弛问题 Relaxed Problem 中产生

规则被放松了的问题

- 例如，8-拼图问题中，一个方片可以与其它方片重叠

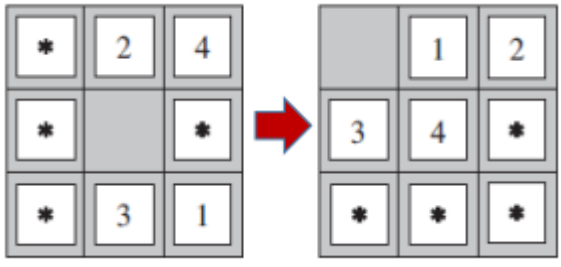
定理：一个松弛问题的最优解的代价是原问题的一个可接受的启发式算法

从子问题中产生

子问题

- 任务：让 1，2，3，4 方片移动到正确的位置

定理：一个松弛问题的最优解的代价小于原问题最优解的代价



不相关的子问题 Disjoint Subproblems

- 如果将两个子问题
  - 让 1，2，3，4 方片移动到正确的位置
  - 让 5，6，7，8 方片移动到正确的位置

  相加，考虑这两个子问题是否没有重叠？
- 不，因为它们总是共享一些移动

如果这些共享的移动不算在内的话

- $h_{sub}^{(1,2,3,4)}(s) + h_{sub}^{(5,6,7,8)}(s) \leq c^*(s)$，是可接受的

从经验中产生 —— 学习

- 解决许多 8-拼图问题，以获得许多例子
- 每个示例都由来自解决方案路径的状态和从该点开始的解的实际成本组成
- 这些例子是我们解决这个问题的“**经验**”

什么是好的经验？

- 与预测状态对目标的成本相关
  - $x_1(s)$：放置位置错误的方片个数
  - $x_2(s)$：相邻的，不是目标状态中按顺序相连的方片个数

如何从这些相关的经验特中学习代价函数 h？

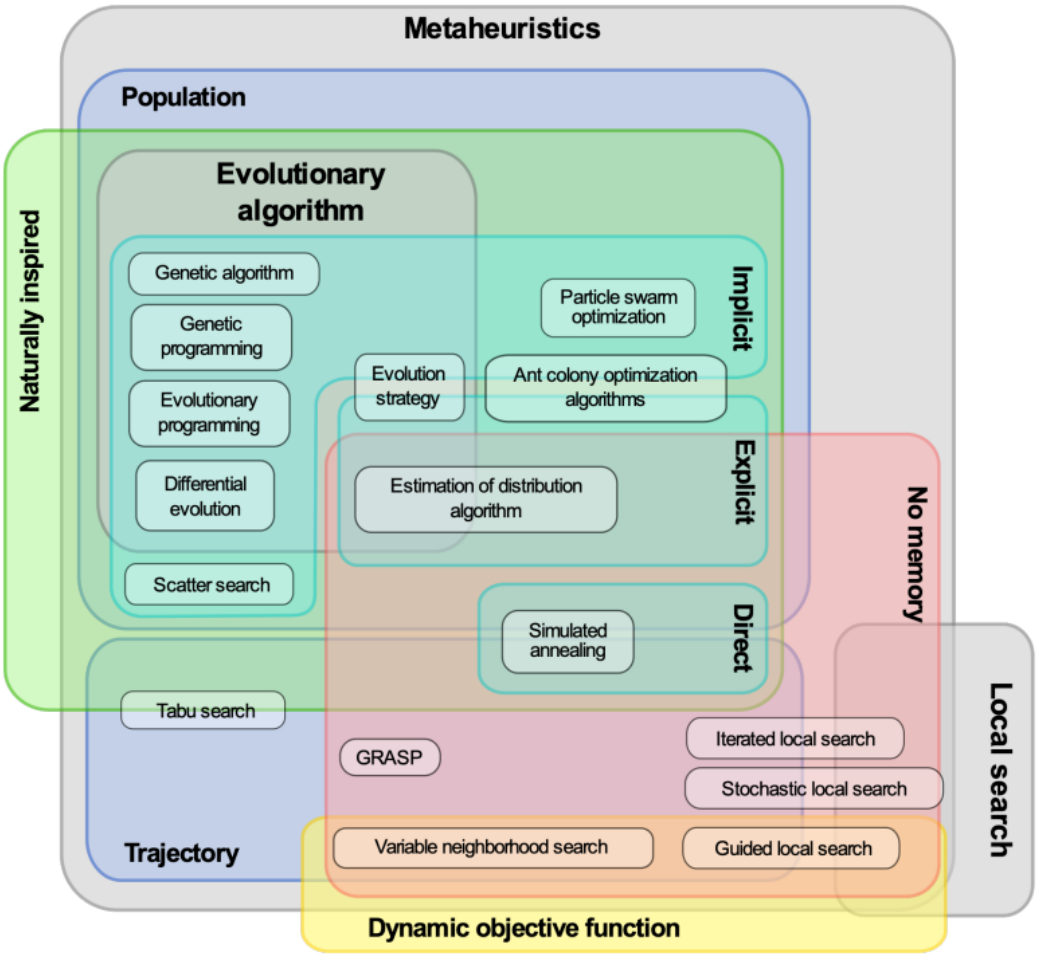
- 构建线性模型 $h(s) = w_1 x_1(s) + w_2 x_2(s)$
- 其中，w_1，w_2 是通过神经网络和决策树等学习方法从训练数据中学习的模型参数

3. 元启发式算法 Metaheuristic

启发式 vs 元启发式算法

启发式	元启发式
针对不同问题采取不同的策略	与问题本身无关
可以被一个元启发式利用	可以使用不同的启发式或者组合启发式

元启发式的范围

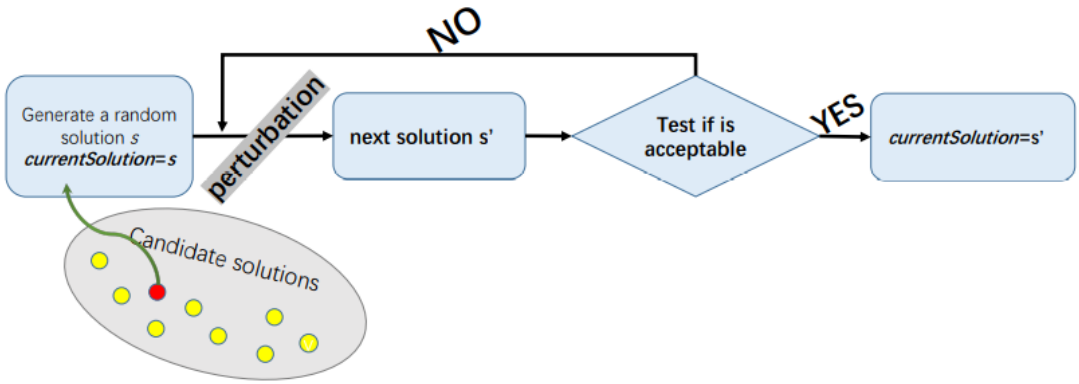


演化计算 Evolutionary Computation

> "One general law, leading to the advancement of all organic beings, namely, **multiply**, **vary**, **let the strongest live and weakest die**."
>
> - multiply：**Reproduction(crossover)**
> - vary: **Mutation**
> - let the strongest live and weakest die: **Selection**
>
> —— Charles Darwin, The Origin of Species

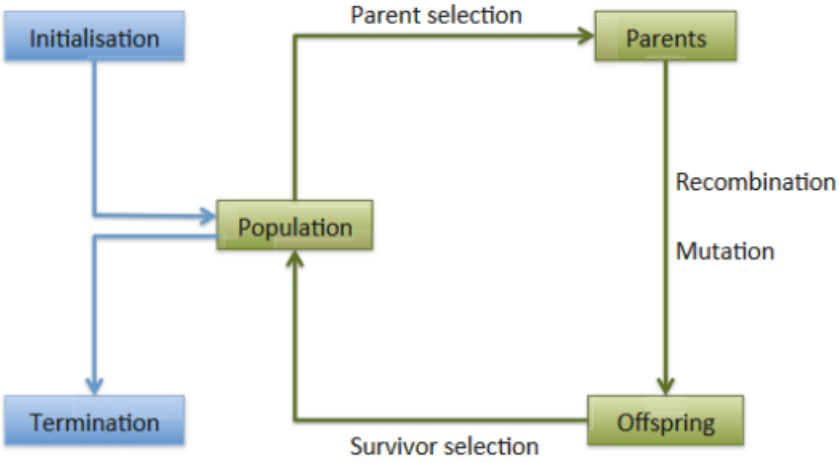
- 它是利用自然进化的思想和灵感来研究计算系统的学科
- 其中，借鉴了达尔文的一个原则，适者生存
- **演化计算 Evolutionary Computation (EC)** 技术可以用于优化、学习和设计
- 演化计算技术不需要使用丰富的领域知识，但是可以将领域知识整合到演化计算中

生成与测试 Generate-and-Test



1. 随机生成**初始解 initial solution**，然后把它声明称**当前解 current solution**
2. 通过**扰动 perturbation** 从当前解中生成**下一代解 next solution**
3. 检查，新生成的解是否可以**接受的 acceptable**
    1. 如果更好，那么接受
    2. 否则，保持当前的解不变
4. 重复第 2 步，直到进化足够多代

- **生成 Generate**：对于种群中的个体，使用 Mutate 或者 Recombine
- **测试 Test**：从 parents 和 offspring 中选择合适的作为下一代



伪代码

```
function EA()
    return a fittest solution

    Generate the initial population P(0) at random
    i = 0 // Generation counter
    solution = best(P(0))
    while halting citeria is not satisfied
        [Evaluate] the fittness of each individual in P(i)
        [Select] parents from P(i) based on their fieness in P(i)
        [Generate] offspring from the parents using crossover and mutation to
from P(i+1)
        solution = best(P(i))
        i ← i+1
    return solution
```

演化计算示例

问题

最大化指定函数

$$f(x) = x^2$$

- 其中 x 在一个整数的取值区间 $[0, 31]$

种群

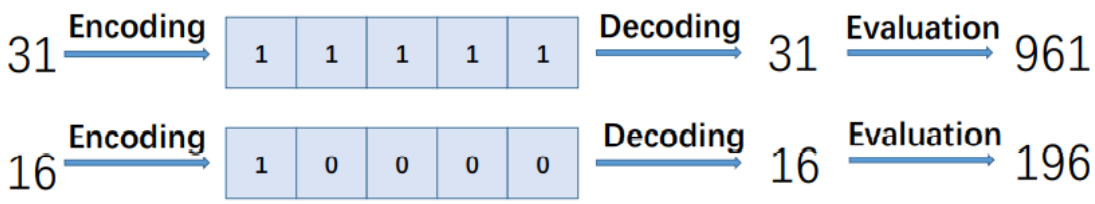
我们来使用一个大小为 4 的种群来进行演化计算

- 种群数量等于 4 等价于有 4 个个体 individual

个体的编解码

EA 的第一步是编码 encoding（即，如何表示个体和染色体）

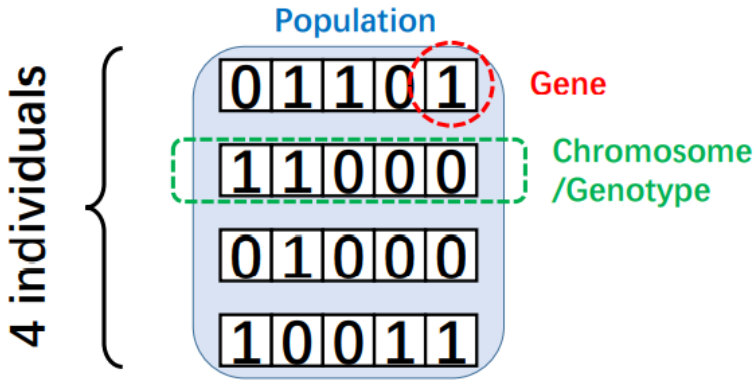
- 我们采用整数的二进制表示法
- 5 个 bit 用于表达整数从 0 到 31



- 基因型：`11111`（即为一个染色体）
- 表现型：31

生成初始种群

随机生成初始种群



- 这里由于个体只有一条染色体（表达 1-31），所以 4 个个体有 4 条染色体

计算适应性函数 fitness function

适应性函数用来评估一个个体的好坏

- 将个体的染色体解码表现型
  - **01101 -> 13; 11000 -> 24, 01000 -> 8, 10011 -> 19;**
- 根据表现型评估适应性函数
  - 这里 $fixness(x) = x^2$
  - $\boldsymbol{f(13) = 169, f(24) = 576, f(8) = 64, f(19) = 361.}$

选择和交叉 Selection & Crossover

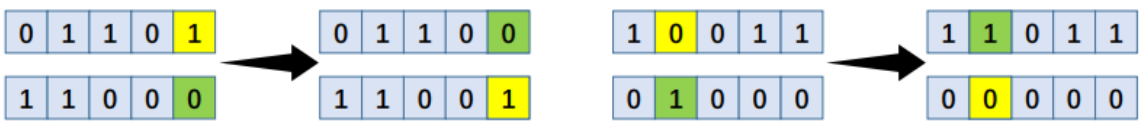
根据适应性函数选择两个个体，可以使用轮盘赌 roulette-wheel selection

$$P_i = \frac{f_i}{\sum_j f_j}$$

- $P_1(13) = \frac{169}{1170} = 0.14,\ P_2(24) = \frac{576}{1170} = 0.49,\ P_3(8) = \frac{64}{1170} = 0.06,\ P_4(19) = \frac{361}{1170} = 0.31$

交叉通常产生两个后代，并添加到中间暂存的新种群中

在我们的例子中，重复这个步骤，直到中间暂存的新种群大小也等于原始种群大小

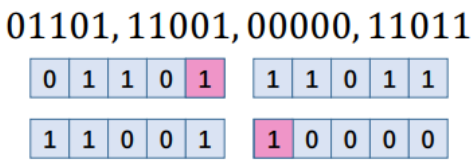


- 现在，中间暂存的种群为 `01000`，`11001`，`11011`，`00000`

变异 Mutation

将突变应用于中间群体中的个体，其概率很小，一个简单的突变是位翻转

例如，随机突变后，我们可能会有如下的新种群



- 红色的地方为突变基因

演化计算不同算法

有很多的熟知的 EA 算法，它们有着不同的

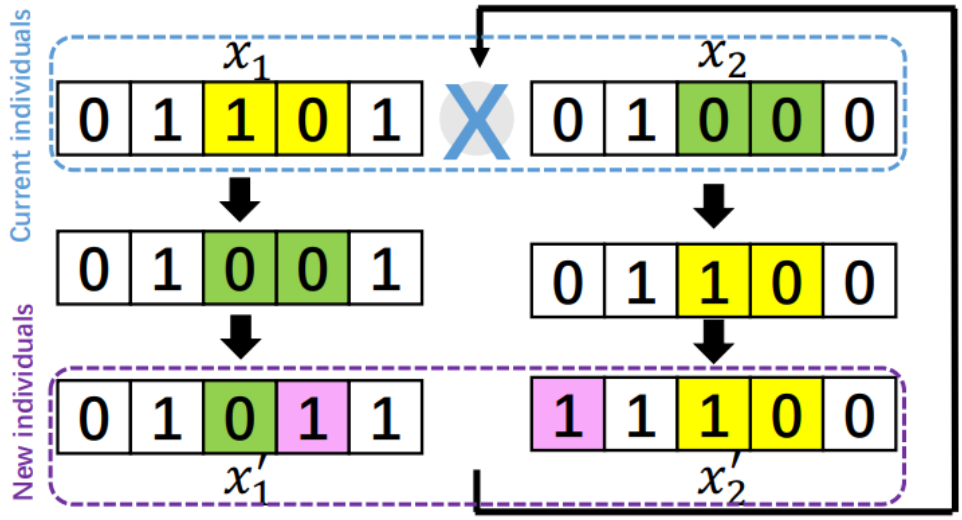
- 历史背景
- 表达
- Mutation 的操作
- Selection 的方案

事实上，EA 指的是一整个算法家族，而不是单一的算法

EA 算法家族

- 遗传算法 Genetic Algorithm（GA）
- 演化编程 Evoutionary Programming（EP）
- 演化策略 Evolution Strategies（ES）
- 遗传编程 Genetic Programming（GP）

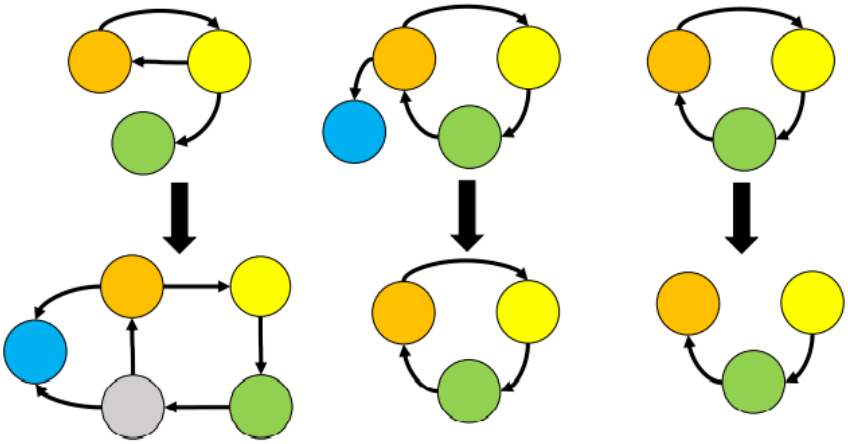
遗传算法 Genetic Algorithm（GA）



Representation	Bit-strings
Recombination	1-Point crossover
Mutation	Bit flip
Parent selection	Fitness proportional - implemented by Roulette Wheel
Survival selection	Generational

- 首先由 Holland 和他的学生在 60 年代中期到 70 年代中期为自适应搜索和优化而提出
- 二进制字符串作为个体（染色体）被广泛使用
- 模拟**达尔文进化论**
- 搜索算子只应用于个体的基因型表示（染色体）
- 强调**重组（交叉） Crossover** 的作用，**变异 Mutation** 只用作背景运算符
- 经常使用**轮盘赌**进行**选择 Selection**

演化编程 Evoutionary Programming（EP）



- Fogel 等人在 20 世纪 60 年代中期为了模拟智能而首次提出
- 用**有限状态机 (FSMs)** 表示个体，尽管实值向量一直用于数值优化
- 它更接近**拉马克进化论**
- 搜索操作符（仅限突变）应用于个体表现型表示
- 它没有使用任何的重组（交叉）
- 通常使用**竞标赛**进行选择 Selection

演化策略 Evolution Strategies（ES）

$$[x_0^{(1)}, x_1^{(1)}, \ldots, x_{d-1}^{(1)}] \quad \times \quad [x_0^{(2)}, x_1^{(2)}, \ldots, x_{d-1}^{(2)}]$$

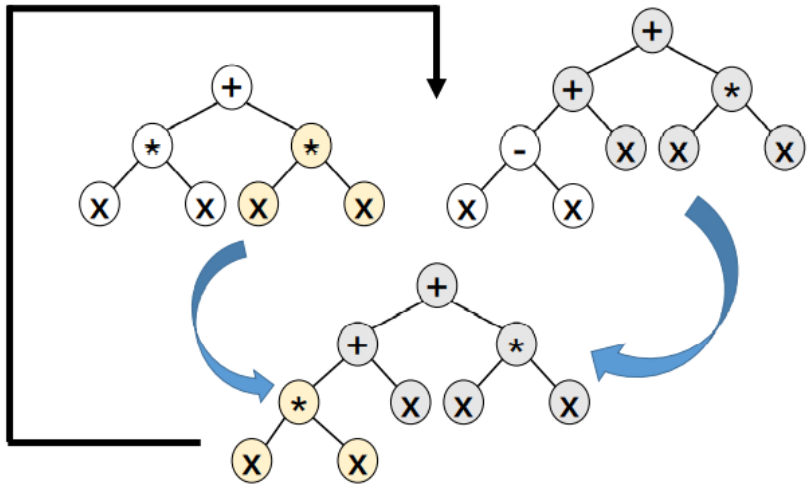
$$\downarrow \qquad\qquad\qquad \downarrow$$

$$[x'^{(1)}_0, x'^{(1)}_1, \ldots, x'^{(1)}_{d-1}] \qquad [x'^{(2)}_0, x'^{(2)}_1, \ldots, x'^{(2)}_{d-1}]$$

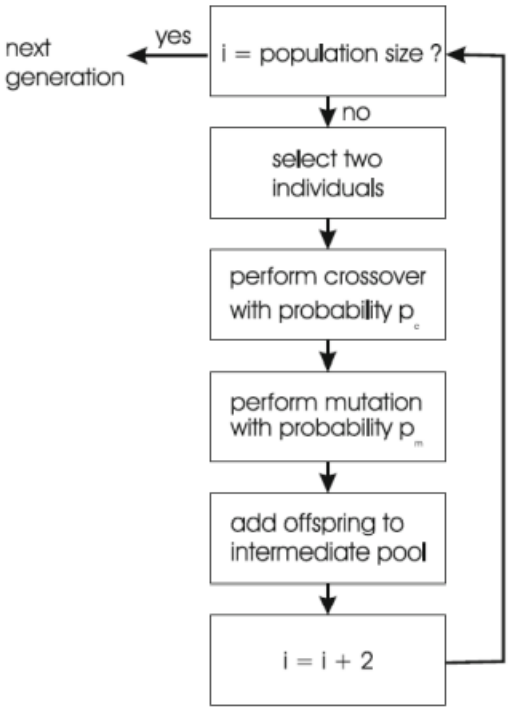
Representation	Real-valued vectors
Recombination	Discrete or intermediary
Mutation	Gaussian perturbation
Parent selection	Uniform random
Survivor selection	Deterministic elitist replacement by (μ, λ) or $(\mu + \lambda)$
Speciality	Self-adaptation of mutation step sizes

- 由 Rechenberg 和 Schwefel 在 20 世纪 60 年代中期首次为解决数值优化提出的方法
- 用**实数向量**表示个体
- 它们更接近拉马克进化论
- 它们使用重组（交叉）
- 它们使用**自适应变异 self-adaptive mutations**

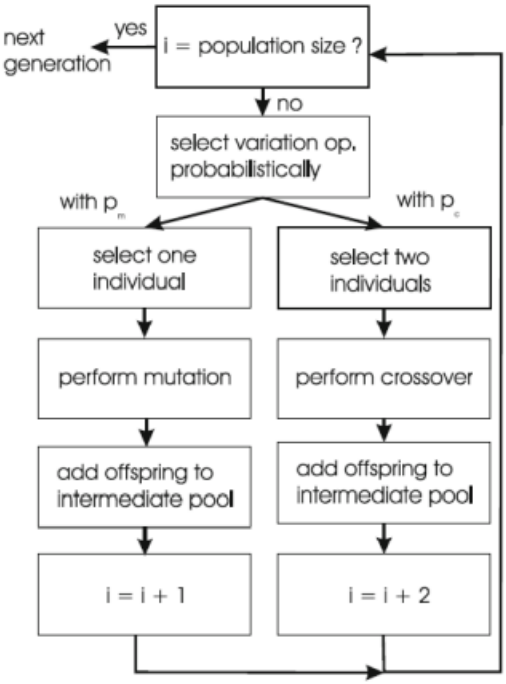
遗传编程 Genetic Programming（GP）



- 首先由 de Garis 用来表示人工神经网络的进化，但由 Koza 用来表示计算机程序的进化
- 使用树（特别是 Lisp expression 树）来表达个体
- 使用交叉和变异



GA loop



GP loop

总结

- EA 算法家族面临的问题就像那些经典的 AI 算法一样，如何表示，如何搜索
- 尽管 GAs、EP、ES 和 GP 是不同的，但它们都是遵循种群的，生成和测试的框架下的不同变体，他们的共同点比不同点多
- 演化计算的技术非常的灵活，且具有鲁棒性
- 一个更好更通用的术语是演化算法 Evolutionary Algorithms

操作符变种

- **交叉/重组 Crossover / Recombination**：单点交叉，两点交叉，统一交叉，中间交叉
- **变异 Mutation**：反转比特，高斯突变，柯西突变
- **选择 Selection**：轮盘赌，竞标赛选择，基于排名的选择
- **替代 Replacement**：根据后代的迭代，稳态迭代
- 特别的操作符：多亲本重组，反转，基于顺序的交叉

解决问题

- 数值优化
- 组合优化（NP-hard 问题）
- 混合优化
- 约束优化
- 多目标优化
- 动态环境优化（动态的 fitness function）

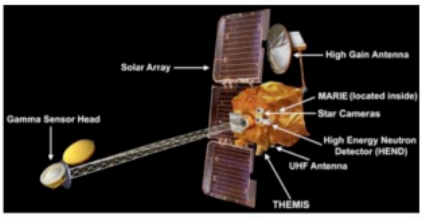
实际应用

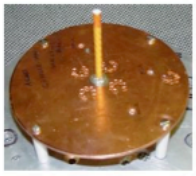
- 高铁头部的设计

Series 700 Designed by human

Series N700 Designed by EA

- X-波段天线设计

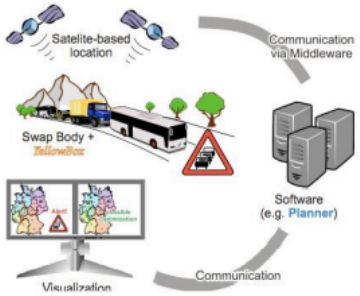




Human

EA

- 交通规划系统



- 鸟巢