
软件分析

# 搜索算法

熊英飞
北京大学
2018

# 判定问题和优化问题


- 判定问题：给定输入，输出“是”或者“否”的问题
  - 程序验证
- 优化问题：给定目标空间和一个适应性（fitness）函数f，f把任意目标空间的值映射成一个实数，求让f结果尽可能大的目标空间的值

# 优化问题举例


- 给定一个程序，求覆盖率最高的输入
- 给定一个带大量配置项的程序，求让程序运行效率最高的配置组合
- 给定一组程序员和要写的模块，求将不同模块分配到不同程序员的最佳方案

# 优化问题⇔判定问题

- 优化问题和判定问题可以互相转化
- 优化问题=>判定问题
  - 生成覆盖率高的测试输入
    - 是否存在覆盖率高于x%的测试输入
    - 是否存在测试输入能覆盖到xx语句
- 判定问题=>优化问题
  - 程序是否会执行某危险语句
    - 定义目标域：程序的输入空间
    - 定义适应度函数：某输入覆盖路径在控制流图中距离该危险语句的距离

# 优化问题


- 针对特定结构的优化问题存在高效算法
  - 针对连续优化问题，存在凸包等结构可以高效求解
  - 针对组合优化问题，存在贪心、动态规划等高效算法
- 但大量的软件分析优化问题并不存在良好的结构
  - 无法预期程序员会写出什么

# 搜索方法解决优化问题


- 搜索算法输入
  - 一个目标空间
  - 一个适应度函数fitness function
- 搜索算法输出
  - 目标空间的一个值，该值在适应度函数下最大

# 搜索算法的基本流程

```
while(! done()) {
  s = getNextValue()
  v = getFitness(s)
  analyzeValue(v, s)
}
printBestValue()
```

- done通常为
  - 达到一定的循环次数
  - 达到一定时间
  - fitness值不再增加
- analyzeValue通常记录下当前的值或仅记录下最佳值
- getNextValue是不同搜索算法的关键

# 随机搜索


- 最基本的搜索算法
- 随机产生下一个值

- 通常作为和其他搜索算法比较的基础
- 很多问题随机搜索已经能取得较好效果

# 元启发式搜索算法

- 采用问题相关的特定启发式规则进行搜索
  - 局部搜索
    - 爬山法
    - 模拟退火
    - 贝叶斯优化
    - 蒙特卡洛树搜索
  - 全局搜索
    - 粒子群算法
    - 遗传算法

# 爬山法


- 爬山法是最基本的启发式搜索算法
- 假设fitness是输入空间上的一个连续函数
- 下一个值为当前值的邻接值，直到达到局部最优点

# 爬山算法


vc ← 随机值

repeat

在vc的邻域中选择所有新点

计算新点的fitness，将最好的点记为vn

if fitness(vn) 好于 fitness(vc)

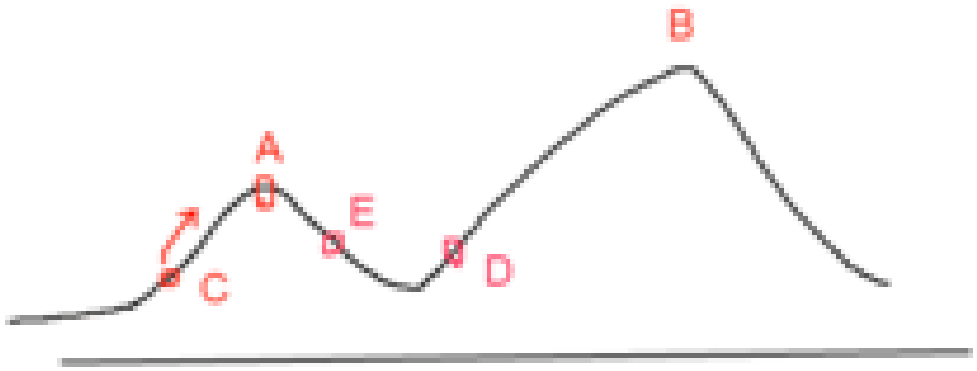
then vc ← vn

else localOptimal ← TRUE

until localOptimal

# 爬山法的缺点

- 只能达到局部最优，不能达到全局最优



# 模拟退火


- 除了上山也允许有一定几率下山

- 模拟退火的命名源自冶金学中的退火
  - 退火过快可能造成冶炼质量不好，所以要以适当速度退火
  - 模拟退火借用这个名字来说明搜索也不要收敛得太快

# 模拟退火算法

Let $s = s_0$

For $k$ = 0 through $k_{max}$ :

    $T \leftarrow k / k_{max}$

    Pick a random neighbour, $s_{new} \leftarrow$ neighbour($s$)

    If $P(fitness(s), fitness(s_{new}), T) \geq$ random(0, 1)

        $s \leftarrow s_{new}$

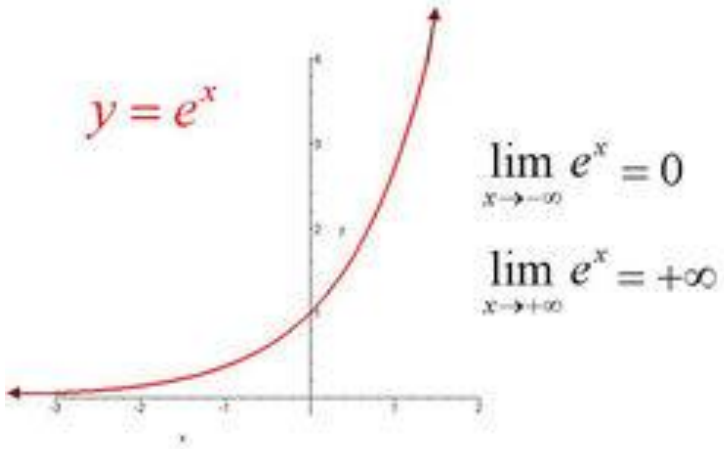
Output: the final state $s$

$s_{new}$越好，P越倾向于选择$s_{new}$

T越大，P越倾向于选择好的$s_{new}$

# P的定义方法

- P通常按照Metropolis准则定义为
  - exp((*fitness*($s_{new}$) - *fitness*(*s*))*T)
  - 求值大于等于1的时候概率为100%




exp函数

# 贝叶斯优化


- 爬山法和模拟退火仅假设适应度函数是连续的
- 如果我们对适应度函数有更多假设，能否带入到搜索过程中？

- 贝叶斯优化——假设适应度函数的形式，然后启发式搜索
  - 本身是一个强化学习过程
  - 较多用于机器学习的参数调优

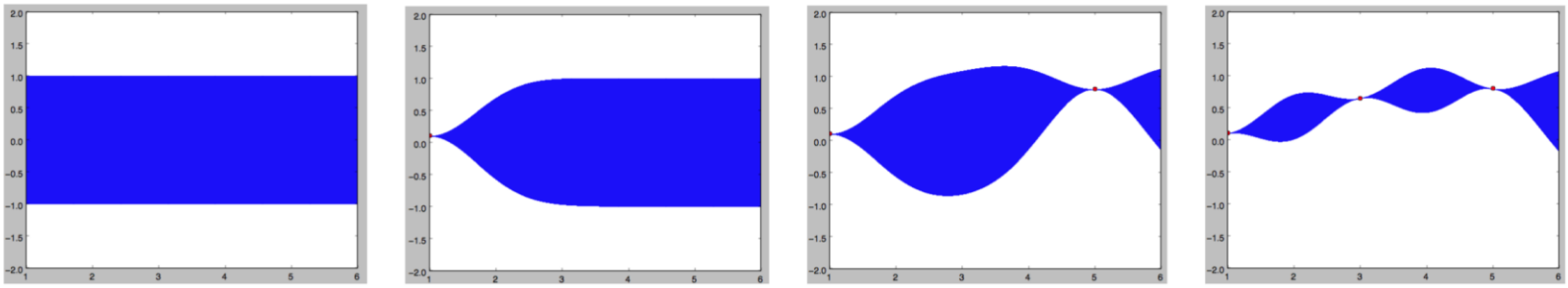
# 贝叶斯优化流程

- 假设适应度函数的形式，包括一组参数
  - 如线性函数+正态分布：$y(\boldsymbol{x}) \sim \mathcal{N}(\boldsymbol{w}\boldsymbol{x}+b, \sigma)$
- 假设参数的分布
  - $\boldsymbol{w} \sim \mathcal{N}(\boldsymbol{0}, \sigma' \boldsymbol{I})$
- 循环
  - 根据参数分布可能最优的输入并采样
    - 寻找$\boldsymbol{x}$，最大化期望值$\max(y(\boldsymbol{x}) - y_{max}, 0)$
      - $y_{max}$为当前找到的最好值
    - 即$y(\boldsymbol{x})$越大越好，同时不惩罚负数
  - 根据样本$\boldsymbol{D}$修正参数的分布
    - 根据贝叶斯公式计算
      - $argmax_{\boldsymbol{w}} P(\boldsymbol{w} \mid \boldsymbol{D}) = argmax_{\boldsymbol{w}} \left( \frac{p(\boldsymbol{D}|\boldsymbol{w}) p(\boldsymbol{w}))}{p(\boldsymbol{D})} \right)$

实际使用时通常采用优化库里提供的函数形式，较少自己推导

# 贝叶斯优化示例

假设适应度函数是高斯过程




最开始的分布　　首先采样x=0　　第二次采样　　第三次采样

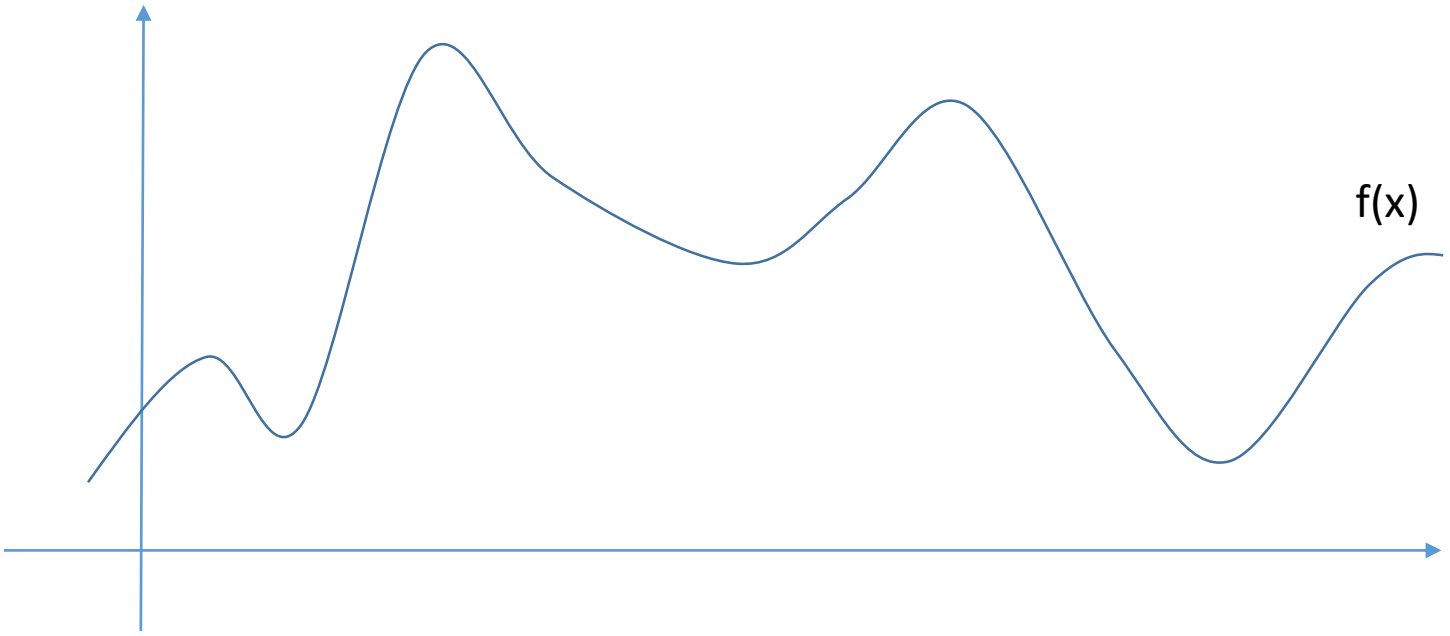
通过采用合适的模型，每次采样既有可能取得最大值，同时也能最大程度消除不确定性。

# 贝叶斯优化小结


- 采用先验知识来有效减少搜索次数
- 根据对适应度函数的假设，重新计算分布有可能会比较复杂
- 适合适应度函数本身计算较慢的情况

# 蒙特卡洛树搜索

- 蒙特卡洛方法——当精确代数解算不出来，通过扔硬币求近似解的轻松愉快方法




如何求f(x)在0-100之间的积分?

# 蒙特卡洛树搜索


- 广泛用于游戏，比如
  - AlphaGo
  - 罗马II：全面战争
- 假设产生一个样本需要经过若干步骤，每个步骤有若干选项，完成所有步骤之后才能计算样本的适应度函数
  - 游戏中每次行动一步，进行到终盘才知道输赢
  - 生成代码过程中每次生成一个token，全部生成完才知道对不对
  - 搜索一个数字的过程中每次生成一个bit，全部生成才能运行适应度函数
- 如何在每一步中选择最优？

# 蒙特卡洛树搜索


- 进行完一步之后，随机进行剩余步骤
- 采样多次，计算平均值或最大值或其他的统计指标
- 根据指标选择最好的步骤继续进行

# 蒙特卡洛树搜索小结


- 适合一次搜索需要较多步骤完成的情况
- 假设每个步骤对结果的影响有一定独立性

# 粒子群搜索


- 以上本地搜索算法一次只考虑一个候选值
- 全局搜索算法一次考虑多个候选值
- 粒子群算法
  - 粒子群算法(particle swarm optimization，PSO)，由Kennedy和Eberhart在1995年提出
  - 该算法模拟鸟群飞行觅食的行为，鸟之间通过集体的协作使群体达到最优目的，是一种基于Swarm Intelligence(群体智能)的优化方法。
  - 多次入选中科院和汤森路透的研究前沿

# 粒子群算法


- 假设目标空间是一个n维向量空间
- 空间中有若干粒子在飞行，每个粒子可以感知自己距离目标的距离（fitness）
- 粒子们不断根据自己发现的最佳位置和集体发现的最佳位置调整自己的行为

# 粒子群算法

- 每个结点保留两个参数
  - 结点位置$z_i$
  - 结点速度$v_i$
- 每次迭代根据下面的公式更新位置和速度
  - $v_i \leftarrow v_i + c_1 r_1 (p_i - z_i) + c_2 r_2 (p_g - z_i)$
  - $z_i \leftarrow z_i + v_i$
  - $c_1, c_2$为常数，$r_1, r_2$为随机值。
  - $p_i$为自己发现的最佳值，$p_g$为整体最佳值
- 迭代固定次数或者达到一定时间后结束

# 粒子群——最大速度


- 由于粒子群的飞行速度常常会过大，所以通常通过域值$v_{max}$来限制飞行速度

# 遗传算法


- 由美国密歇根大学的Holland于1975年首次提出
- 假设目标空间的每一个值是由一系列部件（基因）组成
- 整体fitness由每个部件的fitness和部件之间的局部组合决定
- 通过部件之间的组合（遗传）和演化（变异）来寻找新的值

- 例：用遗传算法解SAT问题
- 每个部件是对一个变量的赋值
- Fitness定义为当前满足的clause的数量

# 遗传算法


- 初始化：计算初代种群
  - 如：对所有变量随机赋true/false，生成n组
- 个体评价：计算适应度函数
  - 计算每组满足的clause数量
- 选择：选择可以繁殖的个体
  - 选择满足最多clause的n组
- 交叉：将若干个体的基因交换
  - 将n组两两组合，每组交换前一半变量的值
- 变异：将个体的基因改动
  - 每组随机翻转n个变量的值
- 下一代群体：取新值和老值的并集为下一代群体

# 编码

- 如何把目标空间编码成基因？
- 简单方法：根据二进制编码，每个二进制位是一个基因
  - 不能很好的表现整数的特征
- Gray Code：数值增加1，二进制位差别是1
- 最好针对特定问题设计编码方法

| Decimal | Binary | Gray | Gray as Decimal |
|---|---|---|---|
| 0 | 000 | 000 | 0 |
| 1 | 001 | 001 | 1 |
| 2 | 010 | 011 | 3 |
| 3 | 011 | 010 | 2 |
| 4 | 100 | 110 | 6 |
| 5 | 101 | 111 | 7 |
| 6 | 110 | 101 | 5 |
| 7 | 111 | 100 | 4 |

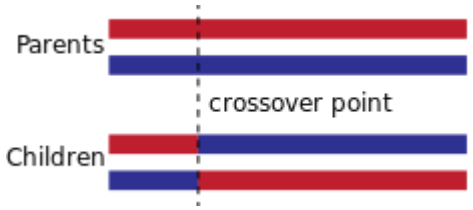
# 初始化

- 随机生成
- 等密度均匀生成

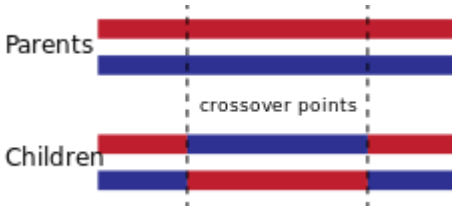
# 选择

- 基本方法：选择fitness最好的n个个体
  - 不利于物种多样性
- 轮盘法：每一轮，每个个体有$\frac{\text{该个体适应度}}{\text{总适应度}}$的概率被选择。重复n轮
  - 适应度差异较大（通常出现在前几轮），仍然不利于物种多样性
- 线性排序：先排序，个体被选择几率根据名次线性递减
- 竞技场选择：每次随机的从当前种群中挑选两个个体，以概率p选择较好个体
  - 避免排序的开销

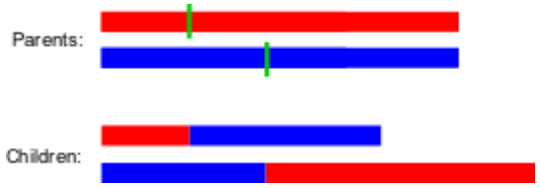
# 交叉




One-point crossover




Two-point crossover




Cut and splice
适用于个体长度不固定的情况


Uniform crossover
每个基因有50%的概率从父亲继承

通常需要针对问题设计交叉方法或者不使用交叉

# 变异

- 随机挑选n个基因，替换他们为新的随机基因
- 新的随机基因可以符合某种概率分布
  - 均匀分布
  - 正态分布
  - 非均匀分布——上一轮随机产生过的基因降低概率
- 非均匀变异
  - 一开始n值较大
  - 之后n值逐渐变小

# Hyper Heuristic/Meta Optimization


- 问题1：这么多种搜索算法，怎么知道哪个比较好？
- 问题2：每个搜索算法都有这么多参数，应该如何设置？

- 把搜索算法和他们的参数作为目标空间
- 给定训练集，fitness为搜索算法限定时间内在训练集上找到的最好结果
- 利用搜索算法找出最好的搜索算法和参数
- 可以看做是一个机器学习的过程

# 启发式搜索vs随机搜索


- 启发式搜索不一定就比随机搜索更快
  - 随机搜索判断出当前解不如最优解的时候就抛弃当前解
  - 启发式搜索必须完成完整的fitness计算并选择下一个值
    - 如果这个过程耗时较长，那么启发式搜索验证的个体数可能显著少于随机搜索
- 案例：基于遗传算法的程序缺陷修复