约束优化问题

(Constrained Optimization Problems)

东北大学 系统工程研究所

2011.09

主要内容

- 无约束优化；
- 非线性规划；

现实中的各类优化问题，多数存在有各类约束条件，本章重点了解**GA**中约束处理的基本技术。

无约束优化问题

(Unconstraint Optimization Problems)

Ackley函数

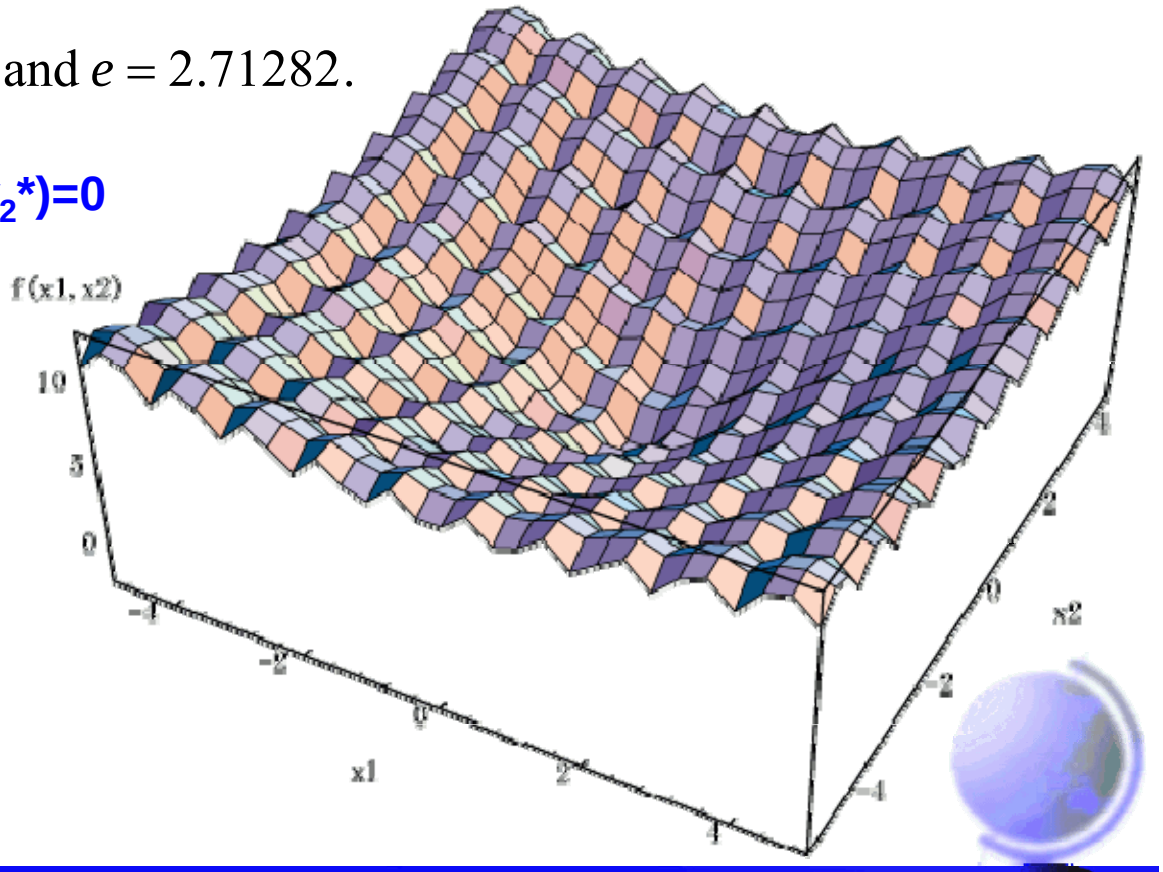
- 多峰的一个测试函数

$$\min\ f(x_1,x_2) = -c_1 \cdot \exp\left(-c_2\sqrt{\frac{1}{2}\sum_{j=1}^{2}x_j^2}\right) - \exp\left(\frac{1}{2}\sum_{j=1}^{2}\cos(c_3 \cdot x_j)\right) + c_1 + e$$

$$-5 \le x_j \le 5, \quad j = 1, 2$$

where $c_1 = 20, c_2 = 0.2, c_3 = 2\pi$, and $e = 2.71282$.

最优解: $(x_1^*, x_2^*)=(0, 0)$, $f(x_1^*, x_2^*)=0$



GA求解

- 实数编码（Real Number Encoding）

  $v = [x_1, x_2, \cdots, x_n]$

  x_i：real number，$i = 1, 2, \cdots, n$

- 算术交叉（Arithmetic Crossover）
  - 算术交叉定义为两个染色体的组合

    $v_1' = \lambda v_1 + (1-\lambda) v_2$

    $v_2' = \lambda v_2 + (1-\lambda) v_1$

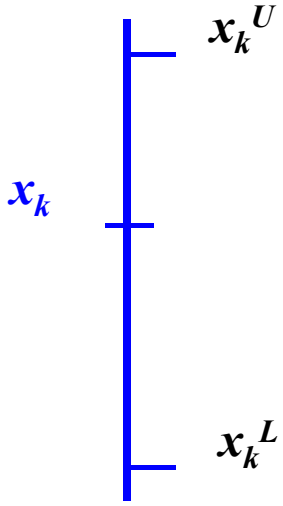
    where $\lambda \in (0, 1)$.

GA求解

- 均匀变异（Uniform Mutation）
  - 对于给定的父代 v，若它的元素 x_k 被选来变异，则生成的后代如下：

$$v = [x_1, \cdots, x_k, \cdots, x_n]$$

$$v' = [x_1, \cdots, x_k', \cdots, x_n]$$



  - x_k' 在 $[x_k^L, x_k^R]$,中的随机选取
  - 这里 x_k^U and x_k^L 分别是 x_k 的上下界。

GA求解

- 适值函数（Fitness Function）
  - 对于本例求极小问题，适值函数可取如下形式：

$$eval(v) = C - f(x)$$

  - 目标函数到适值函数的映射，需要保证以下两点：
    - 映射后的适值是非负的
    - 目标函数的优化方向，应对应适值的增大方向
- 选择（Selection）
  - 采用“最好种群选择法”（Top PopSize Selection）
  - 从父代和后代中为下一代选择最好的 *pop-size* 个染色体。
- 种群初始化（Initial Population）
  - 在变量的上下界之间随机产生

GA求解

- **GA求解过程**

```
procedure: GA for Unconstraint Optimization
input: objective function, GA parameters
output: best solution
begin
    t ← 0;
    initialize P(t) by real number encoding;
    fitness eval(P);
    while (not termination condition) do
        crossover P(t) to yield C(t) by arithm
        mutation P(t) to yield C(t) by nonunif
        fitness eval(C);
        select P(t+1) from P(t) and C(t) by to
        t ← t + 1;
    end
    output best solution;
end
```

目标函数：$f(x)= -0.2*exp(\ldots\ldots,$

变量上下界：$x_1^U=5$，$x_1^L= --5\ldots$

种群大小：10.

采用实数编码，根据种群大小，在变量

将染色体（实质就是问题的解）代入如下适值函数：$eval(v) = 15 - f(x)$, 计算适值，

$v_1 = [1.23, 2.45]$, $eval = 6.52$

$v_2 = [2.41, 3.52]$, $eval = 3.63$

对交叉变异后新产生的染色体，代入如下适值函数：$eval(v) = 15 - f(x)$, 计算适值：

v_2'= [3.18，2.59]，$eval$ = 4.50（变好）

v_5'= [4.21，1.35]，$eval$ = 3.83（变好）

v_7'= [1.55，4.14]，$eval$ = 3.85（变好）

v_8'= [1.02，3.56]，$eval$ = 5.17（变坏）

v_6'= [2.12，2.20]，$eval$ = 6.95（变好）

$v_{10} = [3.28, 2.85]$, $eval = 4.33$

v_7'= [1.55，4.14]，$eval$ = 3.85

非线性规划问题

(Nonlinear Programming Problems)

非线性规划问题

- 在等式和/或不等式约束的前提下，优化某个目标函数的问题，称为非线性规划。
- 由于许多实际问题不能成功地表述为线性规划模型，因此，非线性规划对于工程、数学和运筹学的各个领域都是极其重要的工具。
- 一般非线性规划可描述如下：

$$\begin{aligned} \min \quad & f(\boldsymbol{x}) \\ \text{s. t.} \quad & g_i(\boldsymbol{x}) \le 0, \quad i = 1,2,\ldots,m_1 \\ & h_i(\boldsymbol{x}) = 0, \quad i = m_1+1,\ldots,m_1+m_2 \\ & \boldsymbol{x} \in \boldsymbol{X} \end{aligned}$$

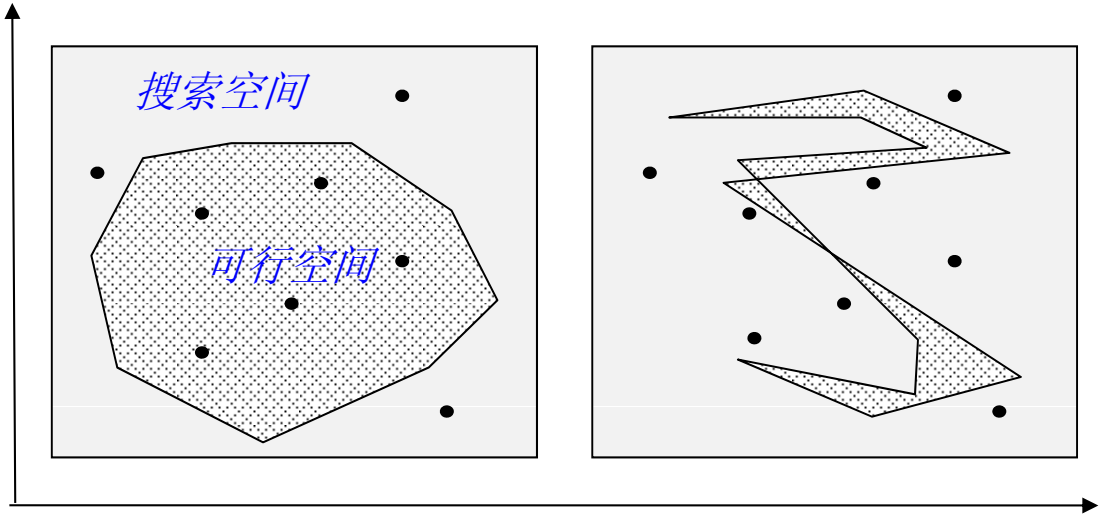
- 非线性规划问题就是寻找一点 $\boldsymbol{y}$ 使得对任意的可行解$\boldsymbol{x}$ 都存在 $f(y) \le f(x)$
- 和线性规划不同,传统的非线性规划方法十分复杂且效率不高。

约束条件的处理

- 几种处理约束的技术：
  - 拒绝策略 (Rejecting Strategy);
  - 修复策略 (Repairing Strategy);
  - 改进遗传算子策略 (Modifying Genetic Operator Strategy);
  - 惩罚策略 (Penalty Strategy)。
- 各种策略都有不同的优点和缺点。

拒绝策略

- 拒绝策略抛弃所有遗传过程中产生的不可行的染色体。
- 这是遗传算法中普遍的作法。
- 当可行的搜索空间是凸的，且为整个搜索空间的适当的一部分时，这种方法应该是有效的。
- 然而，这是很严格的限制。
  - 例如，对许多约束优化问题初始种群可能全由非可行染色体构成，全部拒绝会导致算法无法运行。



修复策略

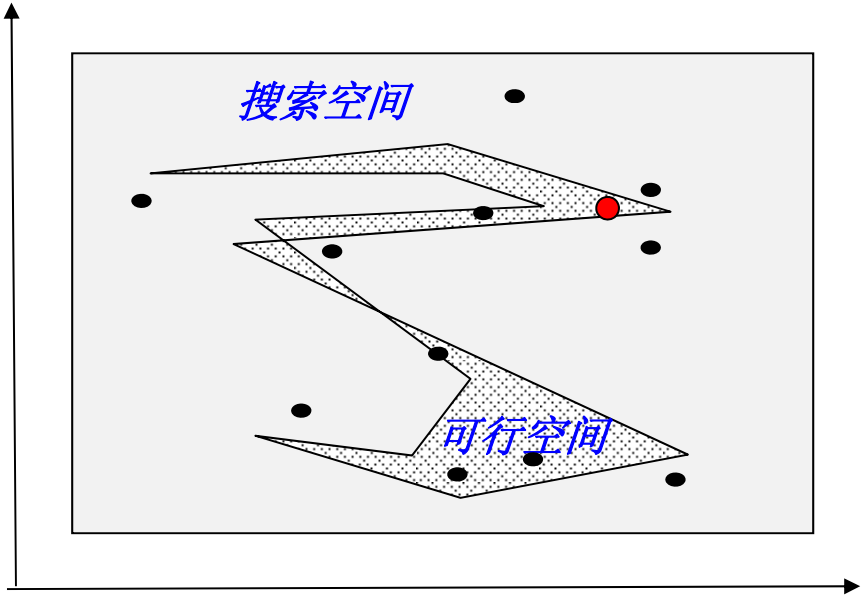
- 修复策略对不可行染色体采用“修复程序”使之变为可行。
  - 对于某些系统，特别是可行搜索空间非凸时，通过对不可行染色体的修复，往往更容易达到最优解。
  - 对于许多组合优化问题，构造修复程序相对比较容易。
  - 能否采取修复策略，取决于是否存在一个可将不可行染色体转化为可行染色体的修复程序。
- 该方法的缺点是：
  - 它对*问题本身的依赖性*，*对于每个具体问题必须设计专门的修复程序*。
  - 对于某些问题，*修复过程甚至比原问题的求解更复杂。*
- 最近，**Orvosh和Davis提出了所谓的5%规则:**
  - 该规则对于多数组合优化问题，若令**5%**的修复过的染色体替代原染色体，则带有修复程序的遗传算法可取得最好的效果。
- **Michalewicx** 等则认为对有非线性约束的优化问题，**15%**的替代律为最好。

改进遗传算子策略

- 解决可行性问题的一个合理办法是设计针对问题的表达方式以及专门的遗传算子来维持染色体的可行性。
- 许多领域中的实际工作者采用“专门的问题表达方式和遗传算子”构造了非常成功的遗传算法，这已是一个十分普遍的趋势。
- 这种方法通常比基于惩罚的遗传算法更可靠。
- 但是，该方法的遗传搜索被限制在可行域内。无法搜索可行域外的点。

惩罚策略

- 对于约束严的问题，不可行解在种群中的比例很大。如果将搜索限制在可行域内，就很难找到可行解。
- **Glover 和 Greenberg 建议的约束处理技术：**
  - 允许在搜索空间的不可行域中进行搜索，要比将搜索限制在可行域内的方法能更快地获得最优解或较好解。
- 惩罚策略就是这类*在遗传搜索中考虑不可行解的技术*。



惩罚策略

- 惩罚策略本质上是通过惩罚不可行解，将约束问题转化为无约束问题。
  - 例如，对如下约束优化问题：

$$\begin{aligned} \min \quad & f(\boldsymbol{x}) \\ \text{s. t.} \quad & h(\boldsymbol{x}) = 0 \end{aligned}$$

通过乘子转化为无约束优化问题：

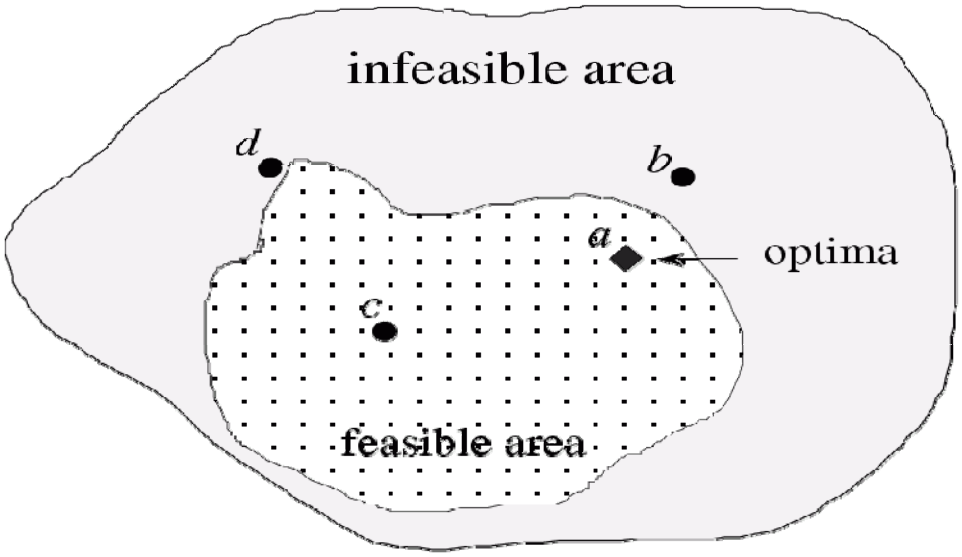
$$\min \quad f(\boldsymbol{x}) + \lambda \left| h(\boldsymbol{x}) \right|$$

将不可行解对约束的违反程度作为惩罚项加入目标函数

  - 对 **GA** 而言，这种惩罚反应在适值函数上，通过对违反约束的染色体进行惩罚，使之具有较低的适值。

惩罚策略

- 惩罚技术用来在每代的种群中保持部分不可行解，使遗传搜索可以从可行域和不可行域两边来达到最优解
- 要想保留不可行解，惩罚必须适当，否则要么被淘汰、要么最后找不到可行解。



- 惩罚策略的主要问题是：
  - 如何设计一个惩罚函数 $P(X)$，从而能有效地引导遗传搜索达到解空间的最好区域。

惩罚函数

- 设计惩罚函数没有一般规则，仍要依赖于待解的问题。
- 带惩罚项的适值函数
  - 加法形式：

$$eval(\boldsymbol{x}) = f(\boldsymbol{x}) + p(\boldsymbol{x})$$

$$p(\boldsymbol{x}) = 0 \quad \boldsymbol{x} \text{ 可行解}$$
$$p(\boldsymbol{x}) < 0 \qquad \text{其它}$$

  - 乘法形式：

$$eval(\boldsymbol{x}) = f(\boldsymbol{x}) \cdot p(\boldsymbol{x})$$

$$p(\boldsymbol{x}) = 1 \quad \boldsymbol{x} \text{ 可行解}$$
$$0 \le p(\boldsymbol{x}) < 1 \quad \text{其它}$$

假设问题是求极大。这里 $\boldsymbol{x}$ 表示一个染色体，$f(\boldsymbol{x})$ 表示目标函数，$p(\boldsymbol{x})$ 表示惩罚项。

惩罚函数

- 惩罚函数的分类
  - 基本上，惩罚是距可行域的距离的函数。这个距离可按以下三种方式测量:
    - 单一不可行解的绝对距离的函数;
    - 现行种群中所有不可行解的相对距离的函数;
    - 自适应惩罚项的函数。
  - 一般可分为两类
    - 定量惩罚
      - 不考虑约束违反程度，惩罚量是一定的。
      - 定量惩罚法，对于复杂问题不太有效
    - 变量惩罚。一般包含两部分:
      - 可变惩罚率：可按约束违反程度和 GA 的迭代次数进行调节。
      - 违反约束的惩罚量.

惩罚函数

- 惩罚函数的分类
  - 惩罚法可进一步分为
    - 问题依赖的;
      - 多数惩罚技术属于问题依赖的
    - 问题独立的。
  - 惩罚法还可分为
    - 带参数的;
      - 多数惩罚技术属于带参数的，
      - 参数惩罚函数基本都是问题依赖的
    - 不带参数的。

典型惩罚函数（1）

- **Homaifar，Qi 和 Lai 方法**

考虑如下非线性规划问题

$$\begin{aligned} \max \quad & f(\mathbf{x}) \\ \text{s.t.} \quad & g_i(\mathbf{x}) \geqslant 0;\ i = 1,2,\cdots,m \end{aligned}$$

取加法形式的惩罚函数

$$eval(\mathbf{x}) = f(\mathbf{x}) + p(\mathbf{x})$$

$$p(\mathbf{x}) = \begin{cases} 0, & \text{若 } \mathbf{x} \text{ 可行} \\ \sum_{i=1}^{m} r_i\, g_i(\mathbf{x}), & \text{其他} \end{cases}$$

仅包含违反约束的项(违反约束为负数），而不是全部约束

其中 r_i 是约束 i 的可变惩罚系数。

- 惩罚函数由两部分构成:
  - 可变惩罚因子
  - 违反约束惩罚。
- **MIichalewicz**最近指出，解的质量严重地依赖于这些惩罚系数的值。当惩罚系数不适当时，算法可能收敛于不可行解;另一方面，惩罚系数过大时该方法等价于拒绝策略。

典型惩罚函数（2）

- **Joines 和 Houck 方法**

考虑如下非线性规划问题

$$\begin{aligned} \max \quad & f(\mathbf{x}) \\ \text{s.t.} \quad & g_i(\mathbf{x}) \geqslant 0;\ i = 1,2,\cdots,m_1 \\ & h_i(\mathbf{x}) = 0;\ i = m_1+1,2,\cdots,m\ (=m_1+m_2) \end{aligned}$$

取加法形式的评估函数：

$$eval(\mathbf{x}) = f(\mathbf{x}) + p(t,\mathbf{x})$$

$$p(t,\mathbf{x}) = -\rho_t^{\alpha}\sum_{i=1}^{m} d_i^{\beta}(\mathbf{x})$$

$$d_i(\mathbf{x}) = \begin{cases} 0, & \text{若 } \mathbf{x} \text{ 可行} \\ |g_i(\mathbf{x})|, & \text{否则 } 1 \leqslant i \leqslant m_1 \\ |h_i(\mathbf{x})|, & \text{否则 } m_1+1 \leqslant i \leqslant m \end{cases}$$

$$\rho_t = C \times t$$

仅包含违反约束的项，而不是全部约束

t 是遗传算法的迭代次数，α 和 β 是调节惩罚值大小的参数。

C 是常数。通过 ρ_t 对不可行染色体的惩罚随进化过程进展而增加。

- 惩罚函数由两部分构成:
  - 可变惩罚因子
  - 违反约束惩罚。
- 可变惩罚因子随迭代次数而变。解的质量对三个参数的值很灵敏。
- 在遗传算法迭代的后期，该方法将给不可行染色体以死亡惩罚。该方法会因惩罚过大而过早收敛。

典型惩罚函数（3）

- **Michalewicz 和 Attia 方法**

考虑如下非线性规划问题

$$\begin{aligned} \max \quad & f(\mathbf{x}) \\ \text{s.t.} \quad & g_i(\mathbf{x}) \geqslant 0;\ i = 1,2,\cdots,m_1 \\ & h_i(\mathbf{x}) = 0;\ i = m_1+1,\cdots,m(=m_1+m_2) \end{aligned}$$

取加法形式的评估函数：

$$eval(\mathbf{x}) = f(\mathbf{x}) + p(\tau,\mathbf{x})$$

$$p(\tau,\mathbf{x}) = -\frac{1}{2\tau}\sum_{i\in A} d_i^2(\mathbf{x})$$

$$d_i(\mathbf{x}) = \begin{cases} \min\{0, g_i(\mathbf{x})\}, & \text{若 } 1 \leqslant i \leqslant m_1 \\ |h_i(\mathbf{x})|, & \text{若 } m_1+1 \leqslant i \leqslant m \end{cases}$$

A 是起作用的约束集，它由所有等式约束和不能满足的不等式约束构成。

τ 是可变惩罚因子，称为温度。从初始温度 τ_0 开始，终止在冻结温度 τ_f

- 惩罚函数由两部分构成:
  - 可变惩罚因子
  - 违反约束惩罚。
- 此方法对参数值十分灵敏。存在的问题是对具体问题应如何设定参数。

典型惩罚函数（4）

- **Smith，Tate 和 Coit 方法**

考虑如下非线性规划问题：

$$\max \quad f(\mathbf{x})$$

$$\text{s.t.} \quad g_i(\mathbf{x}) \leqslant b_i;\ i = 1,2,\cdots,m$$

取加法形式的评估函数：

$$eval(\mathbf{x}) = f(\mathbf{x}) + p(\mathbf{x})$$

$$p(\mathbf{x}) = -\sum_{i=1}^{m}\left(\frac{\Delta b_i(\mathbf{x})}{\Delta b_i^{\text{nef}}}\right)^{\alpha}(f_{\text{all}}^{*} - f_{\text{feas}}^{*})$$

α 是用来调节惩罚严厉性的参数；$\Delta b_i(\mathbf{x})$ 为约束 i 的违反量；

Δb_i^{nef} 是约束 i 的近可行性阈，如何确定适当的 nef 取决于具体问题；

f_{feas}^{*} 是已经找到的最好可行解的目标函数值，

f_{all}^{*} 是已获得的未受惩罚的最好解的目标函数值。

自适应项 $(f_{\text{all}}^{*} - f_{\text{feas}}^{*})$ 可能带来两个问题：①零惩罚和②过惩罚。

当 $f_{\text{all}}^{*} = f_{\text{feas}}^{*}$ 时，该方法对所有不可行解也给予零惩罚；

进化初始阶段，目标值 f_{all}^{*} 很大时，会给所有不可行解加上过大的惩罚。

- 惩罚函数由两部分构成：
  - 违反约束的相对惩罚系数
  - 自适应惩罚项。

典型惩罚函数（5）

- **Yokota，Gen，Ida 和 Taguchi 方法**

考虑如下非线性规划问题：

$$\max \quad f(\mathbf{x})$$

$$\text{s.t.} \quad g_i(\mathbf{x}) \leqslant b_i;\ i = 1,2,\cdots,m$$

取乘法形式的评估函数：

$$eval(\mathbf{x}) = f(\mathbf{x})\ p(\mathbf{x})$$

$$p(\mathbf{x}) = 1 - \frac{1}{m}\sum_{i=1}^{m}\left(\frac{\Delta b_i(\mathbf{x})}{b_i}\right)$$

$$\Delta b_i(\mathbf{x}) = \max\{0,\ g_i(\mathbf{x}) - b_i\}$$

$\Delta b_i(\mathbf{x})$是约束 i 的违反量。

- 惩罚函数由一部分构成:
  - 违反约束的相对惩罚系数
- 该方法的惩罚比Smith等的方法相对温和一些。
- 特别应注意，该惩罚函数是不含参数的，且不依赖于问题。

典型惩罚函数（6）

- **Gen 和 Cheng 方法**
  - 为了给不可行解更严厉的惩罚 **Gen** 和 **Cheng** 进一步改进了以上方法。

考虑如下非线性规划问题：

$$\max \quad f(\mathbf{x})$$

$$\text{s.t.} \quad g_i(\mathbf{x}) \leqslant b_i;\ i = 1,2,\cdots,m$$

取乘法形式的评估函数：

$$eval(\mathbf{x}) = f(\mathbf{x})\ p(\mathbf{x})$$

$$p(\mathbf{x}) = 1 - \frac{1}{m}\sum_{i=1}^{m}\left(\frac{\Delta b_i(\mathbf{x})}{\Delta b_i^{\max}}\right)$$

$$\Delta b_i(\mathbf{x}) = \max\{0,\ g_i(\mathbf{x}) - b_i\}$$

$$\Delta b_i^{\max} = \max\{\varepsilon,\ \Delta b_i(\mathbf{x});\ x \in P(t)\}$$

$\Delta b_i(\mathbf{x})$是约束 i 在 $\mathbf{x}$ 的违反量；

$\Delta b_i^{\max}$ 是当前种群中约束 i 的最大违反量；

ε 为避免除零的小正数。

  - 对严约束优化问题，每代中不可行解在种群中的比例较大。该惩罚方法可以逐代自动调节惩罚比，从而避免过惩罚。

GA求解

- 实数编码（Real Coding Technique）
  - 每个染色体编码为一个和解向量维数相同的实向量。

$$v = [x_1, x_2, \cdots, x_n]$$

  - 又称为浮点表达，实数表达或连续表达。
- 遗传运算方式可分为：
  - 传统运算（Conventional Operators）；
    - 将二进制表达的运算扩展到实数表达的即为传统运算
  - 算术运算（Arithmetic Operators）；
    - 针对实数编码，借用凸集理论中向量线性组合的概念，而形成的运算
  - 基于方向的运算（Direction-based Operators）。
    - 是将近似梯度(次梯度)或负梯度方向引入遗传算法得来的运算。

传统运算

- 简单交叉（Simple Crossover）
  - 一般的交叉运算可模仿二进制表达的交叉。
  - 最基本的是单点交叉。

crossing point at *k*th position

parents

$$\boldsymbol{x}=[x_1,x_2,\ldots,x_k,x_{k+1},x_{k+2},\ldots,x_n]$$

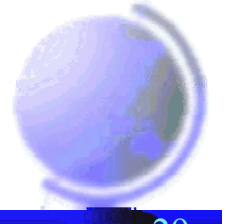
$$\boldsymbol{y}=[y_1,y_2,\ldots,y_k,y_{k+1},y_{k+2},\ldots,y_n]$$

offspring

$$\boldsymbol{x}'=[x_1,x_2,\ldots,x_k,y_{k+1},y_{k+2},\ldots,y_n]$$

$$\boldsymbol{y}'=[y_1,y_2,\ldots,y_k,x_{k+1},x_{k+2},\ldots,x_n]$$

  - 进一步，可将二进制表达的两点、多点和均匀交叉扩展到实数表达中



传统运算

- 均匀交叉（Uniform Crossover）
  - 单点和多点交叉的定义使得个体在交叉点处分成片段。均匀交叉更加广义化，将每个点都作为潜在的交叉点。
  - 随机地产生与个体等长的0-1掩码，掩码中的1表示交换基因，0表示不交换基因。

parents $\boldsymbol{x}=[x_1,x_2,x_3,x_4,x_5,x_6,x_7,x_8,x_9,x_{10}]$

$\boldsymbol{y}=[y_1,y_2,y_3,y_4,y_5,y_6,y_7,y_8,y_9,y_{10}]$

mask code $[1\quad 0\quad 0\quad 1\quad 1\quad 1\quad 0\quad 0\quad 1\quad 0]$

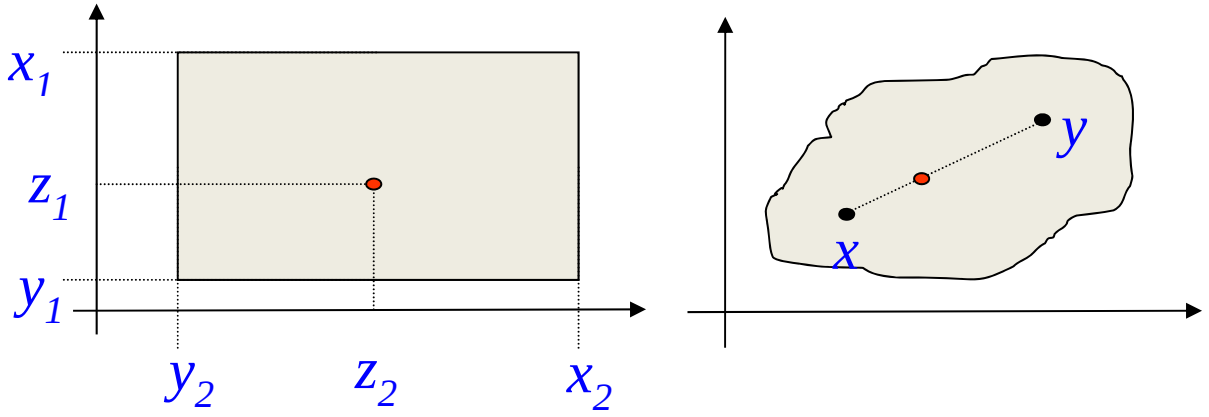
offspring $\boldsymbol{x}'=[y_1,x_2,x_3,y_4,y_5,y_6,x_7,x_8,y_9,x_{10}]$

$\boldsymbol{y}'=[x_1,y_2,y_3,x_4,x_5,x_6,y_7,y_8,x_9,y_{10}]$

传统运算

- 随机交叉（Random Crossover）
  - 本质上，该交叉运算是在由双亲界定的超矩形中随机地产生后代。

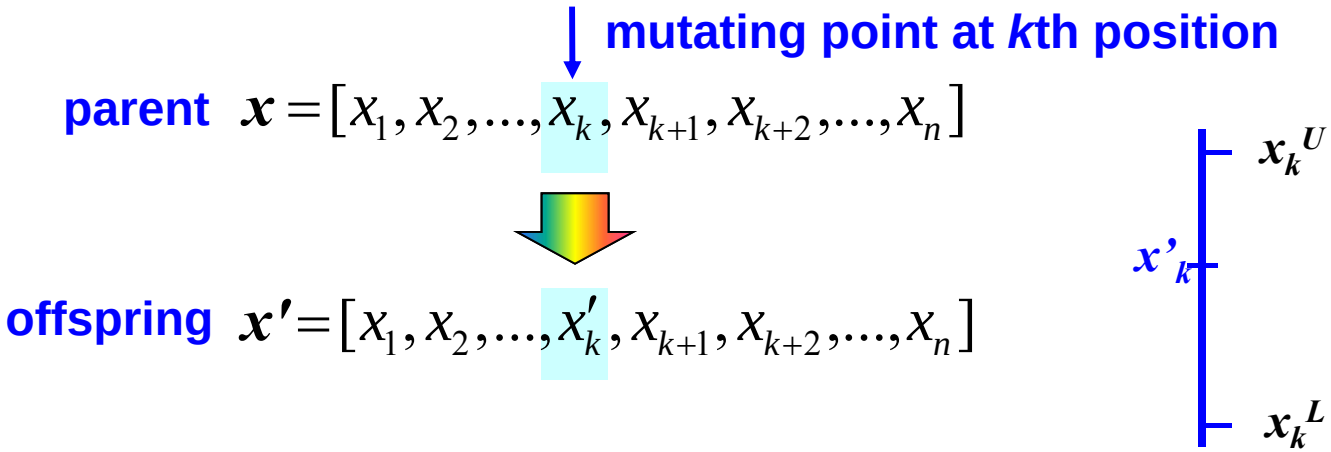
$$\boldsymbol{x}=[x_1,x_2,\ldots,x_n]$$
$$\boldsymbol{y}=[y_1,y_2,\ldots,y_n]$$
$$z=[z_1,z_2,\ldots,z_n]$$



  - 其基本方法由 **Radeliffe** 给出，称为浮点交叉。
    - 对于每个基因，它在双亲的两个基因值之间按均匀分布随机地取一个值作为后代的相应基因的值。
  - **Eshelman** 和 **Schaffer** 扩展了 **Radcliffe** 的工作
    - 在包含双亲的两点间均匀地选取后代的基因值，称之为盲目交叉。

传统运算

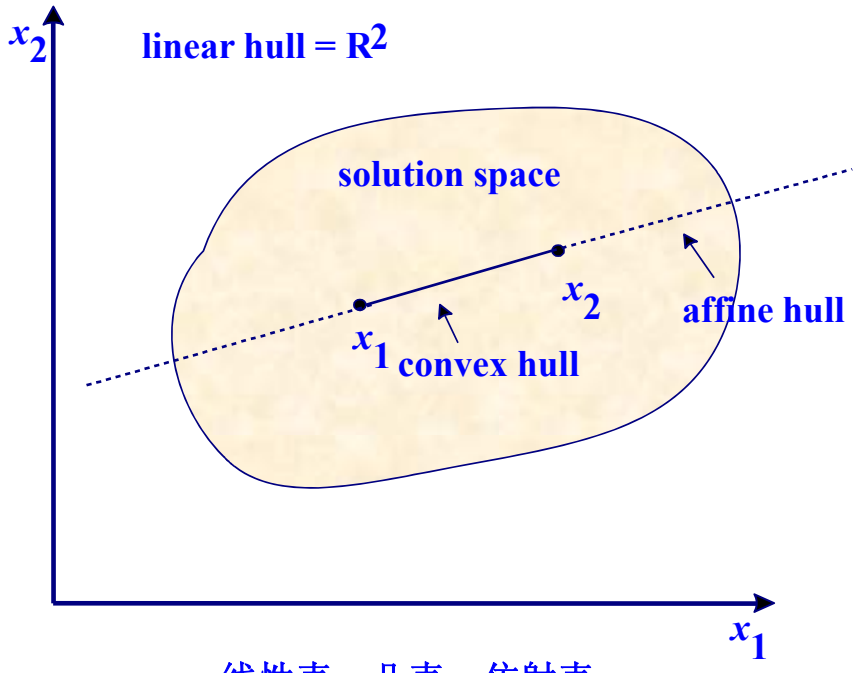
- 变异
  - 最基本的一种是均匀变异（Uniform Mutation），即简单地在指定范围内随机选一个实数替代原基因。



  - x_k^L, x_k^R 通常取变量x_k 的上下界
  - x_k' 是 $[x_k^L, x_k^R]$,中的均匀随机数。
  - 若 x_k' 等概率地由 x_k^L, x_k^R 确定，则称之为边界变异（boundary mutation）
  - 若用 $[x_{k-1}, x_{k+1}]$ 代替上下界，则称之为简易变异（plain mutation）

算数运算

- 交叉
  - 这种运算的基本概念源于凸集理论。
  - 两个父代 x_1 和 x_2 的子代，可以通过$\lambda_1 x_1+\lambda_2 x_2$ 的组合得到。



线性壳、凸壳、仿射壳

$$x_1' = \lambda_1 x_1 + \lambda_2 x_2$$
$$x_2' = \lambda_1 x_2 + \lambda_2 x_1$$

- 凸交叉（***Convex crossover***）

If $\lambda_1+\lambda_2=1$, $\lambda_1>0$, $\lambda_2>0$

- 仿射交叉 （***Affine crossover***）

If $\lambda_1+\lambda_2=1$，去掉非负限制
例如限定取值在：[-0.5,1.5]之间

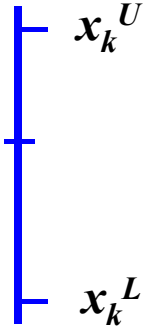
- 线性交叉 (***Linear crossover***)

仅要求乘子属于实数空间
例如限定： $\lambda_1+\lambda_2 \le 2$, $\lambda_1>0$, $\lambda_2>0$

算数运算

- 动态变异（Dynamic mutation）
  - 又称为非均匀变异（Nonuniform Mutation），是由 Janilow 和 Michalewicz 提出的。是为提高精度，增加细调能力而设计的。
  - 对于父代 x，若元素 x_k 被选出作变异，则后代为 $x'=[x_1 \cdots x'_k \cdots x_n]$，$x'_k$ 有如下两种可能的选择：

$$x'_k = x_k + \Delta(t, x_k^U - x_k) \quad \text{or} \quad x'_k = x_k - \Delta(t, x_k - x_k^L)$$



  - x_k^U, x_k^L 是变量x_k 的上下界
  - 函数 $\Delta(t, y)$ 返回一个 $[0, y]$ 之间的值，随代数 t 的增加而趋于零

$$\Delta(t, y) = y \cdot r \cdot \left(1 - \frac{t}{T}\right)^b$$

  - r 是一个[0, 1]间的随机数， T 是最大代数， b 是确定不均匀度的参数
  - 特点：
    - 初始迭代时，搜索均匀分布在整个空间，而到后期则分布在局部范围内。

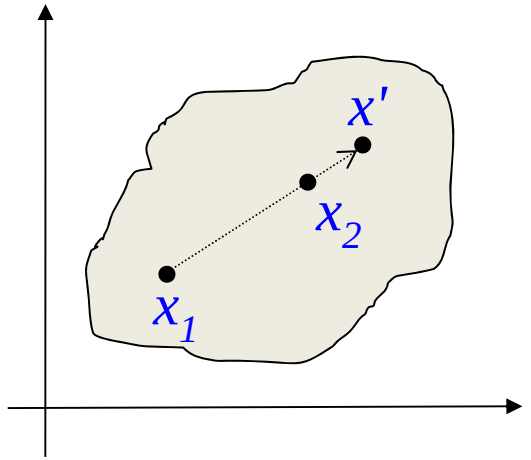
基于方向的运算

- 基于方向的运算，将问题的知识引入遗传算法。

- 基于方向的交叉（**Direction-based crossover**）
  - 采用目标函数的信息来确定遗传操作的方向
  - 根据下述信息，从两个双亲中产生一个后代。

$$x' = r \cdot (x_2 - x_1) + x_2 \quad \text{where } 0 < r \leq 1.$$

  - 假设双亲中 x_2 不比 x_1 坏，
  - 即对极大问题，$f(x_2) \geq f(x_1)$，
  - 对极小问题，$f(x_2) \leq f(x_1)$ 。



基于方向的运算

- 基于方向的变异（**Directional mutation**）
  - 变异后的子代由如下公式给出：

$$\boldsymbol{x}' = \boldsymbol{x} + r \cdot \boldsymbol{d}$$

$$\boldsymbol{d} = \frac{f(x_1, \cdots, x_i + \Delta x_i, \cdots, x_n) - f(x_1, \cdots, x_i, \cdots, x_n)}{\Delta x_i}$$

$$\boldsymbol{r} = \textit{随机非负实数}$$

  - $\boldsymbol{d}$ 是由上式计算的近似梯度
  - $\boldsymbol{r}$ 的选取应使后代是可行解
  - 问题：
    - 当染色体靠近边界时，按上述方法给出的变异方向可能都指向边界，这时就会发生拥挤
  - 为避免染色体都挤到一个角落里，有时也可随机地选一个自由方向替代给定的方向。

实例

$$\begin{aligned}\min \quad & f(\mathbf{x}) = (x_1 - 2)^2 + (x_2 - 1)^2 \\ \text{s.t.} \quad & g_1(\mathbf{x}) = x_1 - 2x_2 + 1 = 0 \\ & g_2(\mathbf{x}) = x_1^2 / 4 - x_2^2 + 1 \geqslant 0\end{aligned}$$

该问题源于 Bracken 和 McCormick 的书

Homaifar，Qi 和 Lai 用遗传算法 求解

End