CONTENTS

引言

基本原理

应用实例

优缺点

算法改进

研究点

差分演化算法

许金良[1]

北京邮电大学 网络技术研究院

November 20, 2015

[1]个人博客：http://blog.csdn.net/u012176591

Outline

1. 引言
2. 基本原理
3. 应用实例
4. 优缺点
5. 算法改进
6. 研究点

优化问题和近似最优解

CONTENTS

引言
基本原理
应用实例
优缺点
算法改进
研究点

- **优化问题**是一种以数学为基础，用于求解各种工程问题的应用技术。
- 绝大多数的工程问题的求解都可以转换为优化问题，但是部分问题属于NP问题，很难找到解析解，比如:0 – 1背包、组合优化问题、任务指派等。某些情况下，退而求其次，找到**近似最优解**即可。
- 针对优化问题的近似解求解，目前已成为了当前一个热点研究方向，催生出一系列的**智能算法**。

CONTENTS

引言
基本原理
应用实例
优缺点
算法改进
研究点

智能算法的研究

- 1975年：J.Holland 根据生物进化过程提出了**遗传算法**。
- 1982年：Kirkpatrick模拟冶金学的退火过程提出了**模拟退火算法**。
- 1991年：Dorigo.M 根据蚂蚁觅食的群体行为提出了**蚁群算法**。
- 1995年：Kennedy根据鸟类觅食的群体行为提出了**粒子群算法**。
- 1997年：Rainer Storn 和Kenneth Price在遗传算法等进化思想的基础上，提出了**差分进化算法** (Differential Evolution, DE)。

差分进化算法简介

CONTENTS

引言

基本原理

应用实例

优缺点

算法改进

研究点

- 由Rainer Storn 和Kenneth Price 在1997年为求解切比雪夫多项式而提出。
- 是一种随机的并行直接搜索算法，它可以对非线性、不可微、连续空间函数进行最小化，以其易用性、稳健性和强大的全局寻优能力在多个领域取得成功。
- 应用：在约束优化计算、聚类优化计算、飞线性优化控制、神经网络优化、滤波器设计、阵列天线方向图综合等。

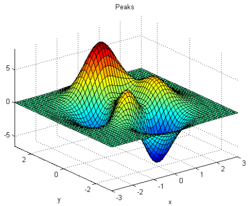
参考文献

- Storn, Rainer and Price, Kenneth. *Differential evolution: a simple and efficient heuristic for global optimization over continuous spaces*. Journal of global optimization, 1997.
- 杨启文, 蔡亮, 薛云灿. 差分进化算法综述. 模式识别与人工智能, 2008.
- 王培崇, 钱旭, 王月, 虎晓红. 差分进化计算研究综述. 计算机工程应用, 2009.
- Das, Swagatam and Suganthan, Ponnuthurai Nagaratnam. *Differential evolution: a survey of the state-of-the-art*. Evolutionary Computation, IEEE Transactions on, 2011.

优化问题表示

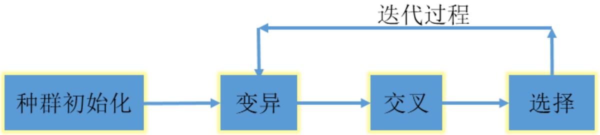
- 左图是两个参数的函数的3-D图像，可以将x-y平面的矩形作为解空间，优化问题就是从解空间中搜索最大最小值。

- 右侧是最优化问题的形式化描述。第一行是目标函数，表示求函数极小值；然后是约束条件。



$$
\begin{aligned}
\min \quad & f(x_1, x_2, \cdots, x_n) \\
s.t. \quad & x_j \in [L_j, U_j], \\
& 1 \le j \le n
\end{aligned}
$$

算法框架



- **种群初始化**在解空间中随机、均匀地产生M个个体，每个个体由n个染色体组成，作为第0代种群，标记为

$$X_i(0) = \left(x_{i,1}(0), x_{i,2}(0), \cdots, x_{i,n}(0)\right)$$
$$i = 1, 2, \cdots, M$$

- **变异、交叉、选择**三步操作迭代执行，直到算法收敛。第g次迭代的第i个个体标记为

$$X_i(g) = \left(x_{i,1}(g), x_{i,2}(g), \cdots, x_{i,n}(g)\right)$$
$$i = 1, 2, \cdots, M$$

种群初始化

差分演化算法

许金良

CONTENTS

引言

基本原理

应用实例

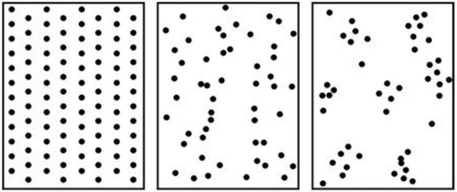
优缺点

算法改进

研究点

在n维空间里随机产生满足约束条件的M个染色体，第i个染色体的第j个维取值方式如下(rand(0,1)产生0到1的均匀分布的随机数)：

$$
\begin{aligned}
x_{i,j}(0) &= L_j + rand(0,1)\left(U_j - L_j\right) \\
i &= 1, 2, \cdots, M \\
j &= 1, 2, \cdots, n
\end{aligned}
$$



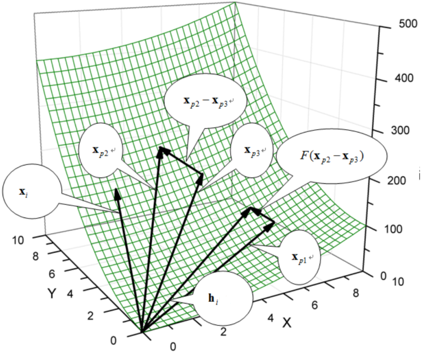
均匀分布　随机分布　聚群分布

变异算子

在第g次迭代中，对个体$X_i(g) = \big(x_{i,1}(g), x_{i,2}(g), \cdots, x_{i,n}(g)\big)$，从种群中随机选择3个个体$X_{p1}(g), X_{p2}(g), X_{p3}(g)$，且$p1 \neq p2 \neq p3 \neq i$，则

$$H_i(g) = X_{p1}(g) + F \cdot \big(X_{p2}(g) - X_{p3}(g)\big)$$

其中$\Delta_{p2,p3}(g) = X_{p2}(g) - X_{p3}(g)$是差分向量；$F$是缩放因子，用于控制差分向量的影响力.

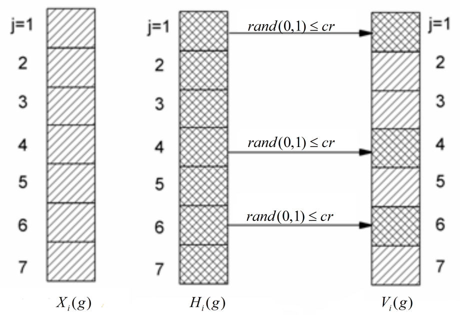


交叉算子

交叉操作可以增加种群的多样性，方法如下：

$$v_{i,j}(g) = \begin{cases} h_{i,j}(g), rand(0,1) \leq cr \\ x_{i,j}(g), else \end{cases}$$

其中$cr \in [0,1]$为交叉概率，$rand(0,1)$是$[0,1]$上服从均匀分布的随机数。



选择算子

首先查看根据评价函数选择$V_i(g)$ 或$X_i(g)$ 作为$X_i(g+1)$

$$X_i(g+1)=\begin{cases} V_i(g), & \text{if} \quad f(V_i(g))<f(X_i(g)) \\ X_i(g), & else \end{cases}$$

可以看出：

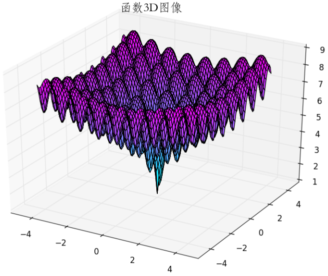
- 对每个个体，$X_i(g+1)$要好于或持平$X_i(g)$。
- 肯定会收敛于最优点（可能是局部最优）。
- **变异、交叉** 操作有助于突破局部最优到达全局最优。

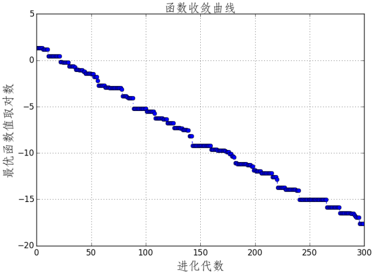
差分进化算法寻找函数最优解

定义关于参数x, y的函数，函数图像如左图所示

$$f(x,y) = -20e^{-0.2\sqrt{\frac{x^2+y^2}{2}}} - e^{\frac{\cos 2\pi x + \cos 2\pi y}{2}} + 20 + e$$

用差分进化算法求解，效果如右图所示（参数设置：$N = 20, F = 0.5, cr = 0.5$，迭代次数$T = 300$）





Pros and Cons

和其他进化算法相比，差异进化算法具有以下优点：

1. 在非凸、多峰、非线性、连续不可微函数优化问题上表现出极强的稳健性。
2. 收敛速度要快。
3. 擅长求解多变量的函数优化问题。
4. 操作简单，容易实现。

缺点：

1. 算法后期个体间差异逐渐缩小，收敛速度慢，容易陷入局部最优。
2. 控制参数和学习策略对算法性能有着重要的影响，并且高度依赖于优化问题的本质。
3. 没有利用个体的先验信息，有时需要过多的迭代才能搜索到全局最优。

CONTENTS

引言
基本原理
应用实例
优缺点
算法改进
参数控制
变异策略
研究点

参数的选取

参数选择主要涉及群体规模M，缩放因子F，以及交叉概率cr的设定[2]

- M：一般介于$5 \times n$与$10 \times n$之间，但不能少于4，否则变异算子无法进行；
- F：一般在$[0, 2]$之间选择，通常取0.5；
- cr：一般在$[0, 1]$之间选择，比较好的选择应在0.3左右。cr取值偏大，收敛速度会加快，但易发生早熟现象。

---

[2]各研究人员得到的经验参数值往往不一致，甚至相互矛盾，所以要具体问题具体分析。

参数的自适应调整(F)

将变异算子中随机选择的三个个体进行从优到劣的排序，得到X_b, X_m, X_w，对应适应度f_b, f_m, f_w，则变异算子改为：

$$V_i = X_b + F_i\left(X_m - X_w\right)$$

同时，F的取值根据生成差分向量的两个个体自适应变化，平衡全局搜索和局部搜索之间的矛盾。

$$F_i = F_l + \left(F_u - F_l\right)\frac{f_m - f_b}{f_w - f_b}$$

其中，

$$F_l = 0.1, F_u = 0.9$$

参数的自适应调整(cr)

对于适应度好的解，取较小的cr，使得该解进入下一代的机会增大；对于适应度差的解，则取交大的cr，加快改变该个体的结构，使该解被淘汰掉。

$$cr_i = \begin{cases} cr_l + (cr_u - cr_l)\frac{f_i - f_{min}}{f_{max} - f_{min}} & \text{if} \quad f_i > \bar{f} \\ cr_l & \text{if} \quad f_i < \bar{f} \end{cases}$$

其中f_i是个体X_i的适应度，f_{min}和f_{max}分别是当前种群中最差和最优个体的适应度，$\bar{f}$是当前种群适应度平均值，cr_l和cr_u分别是cr的下限和上限，一般$cr_l = 0.1, cr_u = 0.6$。

CONTENTS

引言
基本原理
应用实例
优缺点
算法改进
参数控制
变异策略
研究点

变异策略

变异策略表示为$DE/a/b$，其中a表明被变异个体的选择方式，b表明差向量的个数。

1. **DE/rand/1:**
$$V_i = X_{p1} + F\left(X_{p2} - X_{p3}\right)$$
2. **DE/best/1:**
$$V_i = X_{best} + F\left(X_{p1} - X_{p2}\right)$$
3. **DE/current to best/1:**
$$V_i = X_i + F\left(X_{best} - X_i\right) + F\left(X_{p1} - X_{p2}\right)$$
4. **DE/best/2:**
$$V_i = X_{best} + F\left(X_{p1} - X_{p2}\right) + F\left(X_{p3} - X_{p4}\right)$$
5. **DE/rand/2:**
$$V_i = X_{p1} + F\left(X_{p2} - X_{p3}\right) + F\left(X_{p4} - X_{p5}\right)$$

差分进化算法的研究点

CONTENTS
引言
基本原理
应用实例
优缺点
算法改进
研究点

差分进化算法是一种新兴的智能优化算法，在以下方向需要进一步研究：

1. **理论研究：** 如何执行搜索操作？为什么能取得很好效果？目前，相关的理论解释非常有限。此外，仍需要对收敛性、稳定性等方面进行进一步的理论研究。
2. **算法改进：** 控制参数(种群规模、交叉概率和尺度因子)的自适应、变异策略的自适应方面还有很大的研究空间。此外，利用其它技术思想提出新的混合算法，来改善算法性能。
3. **应用研究：** 受关注程度远不及*GA*和*PSO*，一个关键因素是应用的研究还相对薄弱。

差分演化算法

许金良

差分进化算法实现代码

```python
def Rosenbrock(x):
    X=x[0];Y = x[1]
    return -20*np.exp(-0.2*np.sqrt(np.sqrt((X**2+Y**2)/2)))+
20+np.e-np.exp((np.cos(2*np.pi*X)+np.cos(2*np.pi*Y))/2)

N = 20;F = 0.5;cr = 0.3;T = 300

records = []
pops = (np.random.random((N,2))-0.5)*2*4.1
H = np.zeros((N,2))
minfunc = 10000
for n in range(N):
    if minfunc > Rosenbrock(pops[n]):
        minfunc = Rosenbrock(pops[n])

for t in range(T):
    records.append(minfunc)
    for n in range(N):
        sels = np.random.permutation(N)[:3]
        H[n] = pops[sels[0]] + F*(pops[sels[1]]-pops[sels[2]])
        for i in range(2):
            if np.random.random() > cr:
                H[n,i] = pops[n,i]
            if np.abs(H[n,i]) > 4.1:
                H[n,i] = (np.random.random()-0.5)*2*4.1
        if Rosenbrock(pops[n]) > Rosenbrock(H[n]):
            pops[n] = H[n]
            if minfunc > Rosenbrock(pops[n]):
                minfunc = Rosenbrock(pops[n])

plt.plot(np.log(records),'-.o')
```