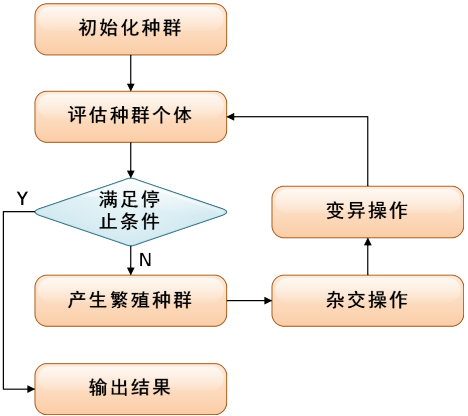


Figure: 遗传算法基本流程图

简单 GA 伪码流程

1. [Start]：产生初始种群;
2. [Fitness]：计算每个个体的适应值;
3. [New population]：产生新种群，直到达到停止条件
   - [Selection]：从上一代种群中选择合适的个体形成繁殖种群;
   - [Crossover]：按照杂交概率 p_c 对选择的父个体进行杂交;
   - [Mutation]：按照变异概率 p_m 对选择的父个体进行变异;
   - [Accepting]：利用新个体形成子种群;
4. [Replace]：通过生存选择在子种群和原始父种群形成新的父种群;
5. [Test]：如果满足终止条件则输出最优结果，并退出程序; 否则执行后续步骤;
6. [Loop] ：返回步骤 2.

智能优化技术

差分演化（进化）算法（Differential Evolution）

龚文引 (教授、博士生导师)

中国地质大学（武汉）计算机学院

1. 大纲

算法简介

算法基本流程

简单示例

小结

2. 算法简介

2. 算法简介

差分演化算法（差分进化算法）

- 演化算法是一种自适应, 并行的全局优化算法;
- 差分演化算法是一种演化算法;
- 演化算法还包括遗传算法, 演化策略, 进化规划和遗传编程等;
- 差分演化算法与其他演化算法的最大区别在于差分变异算子的应用.

R. Storn, K. Price, “Differential evolution – A simple and efficient heuristic for global optimization over continuous spaces," *J. of Global Optim.* 1997, 11 (4): 341 - 359.

差分演化算法

- Storn & Price 于 1995 年首次提出;
- 采用实数编码, 主要用于求解实数优化问题;
- 所求解优化问题可描述为:
  对于目标函数 $f : \mathbb{X} \subseteq \mathbb{R}^n \to \mathbb{R}$, 求最小解 $\mathbf{x}^*$

$$\mathbf{x}^* \in \mathbb{X} : f(\mathbf{x}^*) \leq f(\mathbf{x}), \forall \mathbf{x} \in \mathbb{X}$$

2. 算法简介

算法优点

- 结构简单，其核心代码只需约 30 行 C 语言代码;
- 容易使用，算法参数少，只有 3 个（群体大小 NP，杂交概率 Cr，缩放因子 F）;
- 收敛速度快;
- 鲁棒性好.

3. 算法基本流程

3. 算法基本流程

符号说明

- 设要求解一个 n 维实数优化问题;
- 设置群体大小为 NP ($NP \geq 4$);
- 个体编码为:

$$\mathbf{x}_i = \{x_{i,1}, x_{i,2}, \cdots, x_{i,n}\}$$

其中: $i = 1, 2, \cdots, NP$; $x_{i,j}$ 为一实数, 且 $L_j \leq x_{i,j} \leq U_j$, $j = 1, 2, \cdots, n$.

3. 算法基本流程

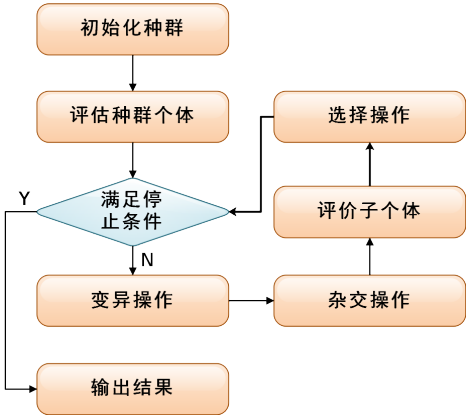
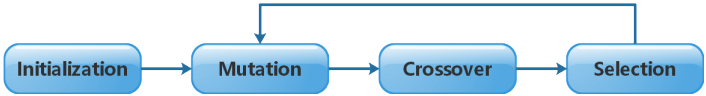


Figure: 差分演化算法基本流程图

3. 算法基本流程

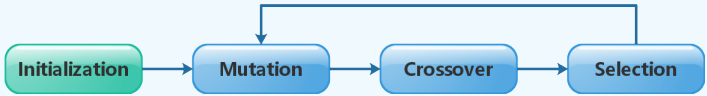


Algorithm 1: 经典差分演化算法流程

产生初始群体
计算群体中个体适应值
while 终止条件未达到 **do**
  for $i = 1$ *to* NP **do** /* 产生子个体 */
    选择三个随机父个体$r_1 \neq r_2 \neq r_3 \neq i$
    利用DE变异和杂交算子产生实验向量$\mathbf{u}_i$
    计算$\mathbf{u}_i$适应值
  for $i = 1$ *to* NP **do** /* 选择算子 */
    if $\mathbf{u}_i$优于$\mathbf{x}_i$ **then**
      $\mathbf{x}_i = \mathbf{u}_i$

3. 算法基本流程

群体初始化



群体中每个个体每一维 $x_{i,j}$ 在其自变量范围 $[L_j, U_j]$ 内采用均匀随机初始化:

$$x_{i,j} = \text{rndreal}(L_j, U_j)$$

3. 算法基本流程

随机函数

```
#define URAND       ((double)rand()/((double)RAND_MAX + 1.0))
double rndreal(double low,double high){    // 随机实数函数
   double  val;
   val = URAND*(high-low)+low;
   return (val);
}
int rndint(int low, int high){             // 随机整数函数
    int res;
    if (low >= high)    res = low;
    else{
        res = low + (int)(URAND*(high-low+1));
        if (res > high) res = high;
    }
    return (res);
}
```

3. 算法基本流程

个体表示

```
typedef struct
{
    double xreal[N_of_x];           /* decision variables */
    double fitness;                 /* fitness of the solution */
}individual;
```

3. 算法基本流程

群体初始化

```
void init_pop_uniform()
{
    int i, j;
    double low, up;

    for(i=0;i<pop_size;i++) {
        for(j=0;j<n;j++) {
            low = lowerBound[j];
            up  = upperBound[j];

            parent_pop[i].xreal[j] = rndreal(low, up);
        }
    }
}
```

3. 算法基本流程

计算个体适应值

$$f(\mathbf{x}) = \sum_{i=1}^{n} x_i^2$$

```
double evaluate_ind(double *x){
    double value = 0.0;
    int i;

    for (i=0;i<n;i++) {
        value += x[i]*x[i];
    }

    /* Increase the number of function evaluations */
    evaluations++;

    return(value);
}
```

3. 算法基本流程

差分变异算子

差分演化算法的核心算子是差分变异(Differential mutation)算子, 其中, 经典算子“DE/rand/1”为:

$$\mathbf{v}_i = \mathbf{x}_{r_1} + F \cdot \left(\mathbf{x}_{r_2} - \mathbf{x}_{r_3}\right)$$

- F为缩放因子 (scaling factor), $F \in (0, 1+)$;
- $\mathbf{v}_i$为变异向量 (mutant vector);
- $\mathbf{x}_{r_1}$为基向量 (base vector);
- $\mathbf{x}_{r_2} - \mathbf{x}_{r_3}$为差分向量 (differential vector);
- $r_1, r_2, r_3 \in \{1, NP\}$, 且 $r_1 \neq r_2 \neq r_3 \neq i$.

变异算子（父个体选择）

```
void parent_selection(int i, int *r1, int *r2, int *r3)
{
     do {
         *r1 = rndint(0, pop_size-1);
     } while(*r1 == i);
     do {
         *r2 = rndint(0, pop_size-1);
     } while(*r2 == i || *r2 == *r1);
     do {
         *r3 = rndint(0, pop_size-1);
     } while(*r3 == i || *r3 == *r2 || *r3 == *r1);
}
```

3. 算法基本流程

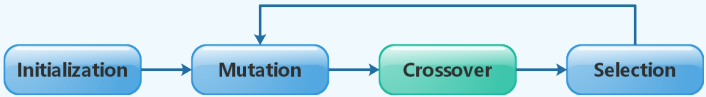
变异算子

```
void mutation(int i, int r1, int r2, int r3)
{
     int j;
     for (j=0;j<n;j++) {
         child_pop[i].xreal[j] = parent_pop[r1].xreal[j] +
             F*(parent_pop[r2].xreal[j]-parent_pop[r3].xreal[j]);

         /* Handle the boundary constraints */
         if (child_pop[i].xreal[j] < lowerBound[j] ||
             child_pop[i].xreal[j] > upperBound[j]) {
             child_pop[i].xreal[j] =
                 rndreal(lowerBound[j], upperBound[j]);
         }
     }
}
```

3. 算法基本流程

杂交算子



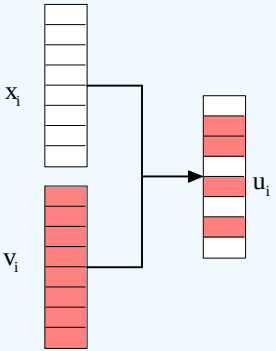
差分演化算法采用离散重组 (discrete recombination), 常用的二项式杂交算子为:

$$u_{i,j} = \begin{cases} v_{i,j}, & \text{if } \mathrm{rndreal}(0,1) < Cr \;||\; j == j_{\mathrm{rand}} \\ x_{i,j}, & \text{otherwise} \end{cases}$$

- $j_{\mathrm{rand}} = \mathrm{rndint}(0, n-1)$;
- $Cr \in [0,1]$ 为杂交概率;
- $\mathbf{x}_i$ 为目标向量 (target vector);
- $\mathbf{u}_i$ 为实验向量 (trial vector).

3. 算法基本流程

杂交算子



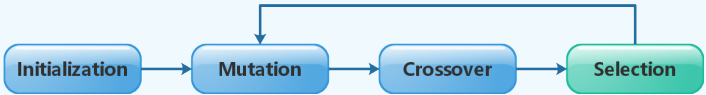
3. 算法基本流程

杂交算子

```
void bin_crossover(int i)
{
    int j, j_rand;
    j_rand = rndint(0, n-1);
    for (j=0;j<n;j++) {
        if (rndreal(0,1) < Cr || j == j_rand) {
            child_pop[i].xreal[j] = child_pop[i].xreal[j];
        }
        else {
            child_pop[i].xreal[j] = parent_pop[i].xreal[j];
        }
    }
}
```

3. 算法基本流程

选择算子



差分演化算法采用一对一锦标赛选择 (one-to-one tournament selection) 算子, 即目标向量 $\mathbf{x}_i$ 与实验向量 $\mathbf{u}_i$ 相比较, 较好个体保存到一下代:

$$\mathbf{x}_i' = \begin{cases} \mathbf{u}_i, & \text{if } f(\mathbf{u}_i) \leq f(\mathbf{x}_i) \\ \mathbf{x}_i, & \text{otherwise} \end{cases}$$

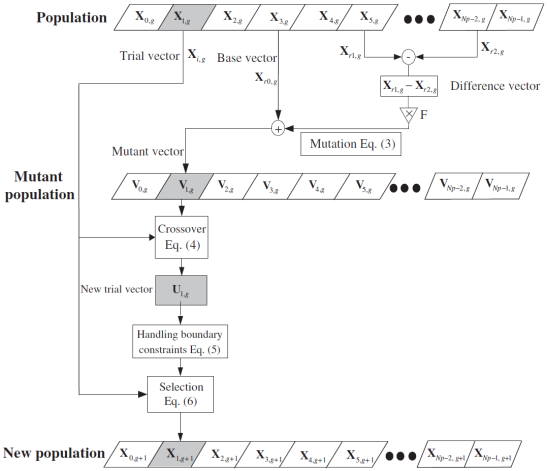
变异, 重组 (杂交) , 选择算子重复执行, 直到终止条件达到.

3. 算法基本流程

选择算子

```
void survival_selection ()
{
    int i;
    for (i=0;i<pop_size;i++) {
        if (child_pop[i].fitness <= parent_pop[i].fitness) {
            /* Replace the worse parent */
            parent_pop[i] = child_pop[i];
        }
    }
}
```

3. 算法基本流程


Population
Trial vector
Base vector
Difference vector
F
Mutation Eq. (3)
Mutant vector
Mutant population
Crossover Eq. (4)
New trial vector
Handling boundary constraints Eq. (5)
Selection Eq. (6)
New population

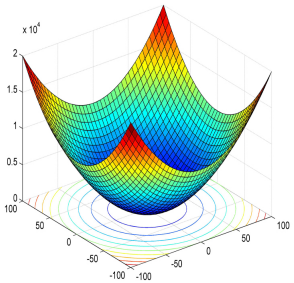
4. 简单示例

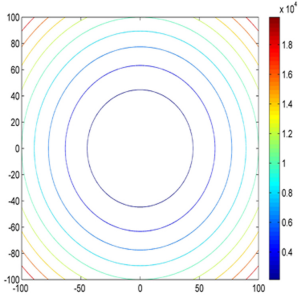
4. 简单示例

示例: 求 Sphere 函数最小值

- Sphere 函数
$$f(x_1, x_2) = x_1^2 + x_2^2$$
- 求解 $\mathbf{x}^* \in [-100, 100]$ 使得 $f(\mathbf{x}^*) \leq f(\mathbf{x}), \forall \mathbf{x} \in [-100, 100]$.
- $f(\mathbf{x}^*) = 0$, $\mathbf{x}^* = (0, 0)$.

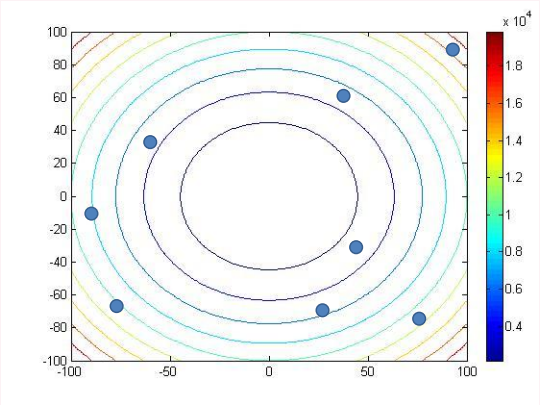
4. 简单示例


x 10^4
2
1.5
1
0.5
0
100
50
0
-50
-100
-100
-50
0
50
100


x 10^4
1.8
1.6
1.4
1.2
1
0.8
0.6
0.4
100
80
60
40
20
0
-20
-40
-60
-80
-100
-100
-50
0
50
100

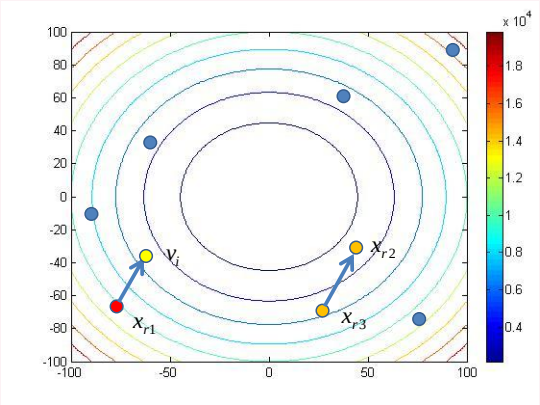
4. 简单示例

示例: 初始化


x 10^4
1.8
1.6
1.4
1.2
1
0.8
0.6
0.4
100
80
60
40
20
0
-20
-40
-60
-80
-100
-100
-50
0
50
100

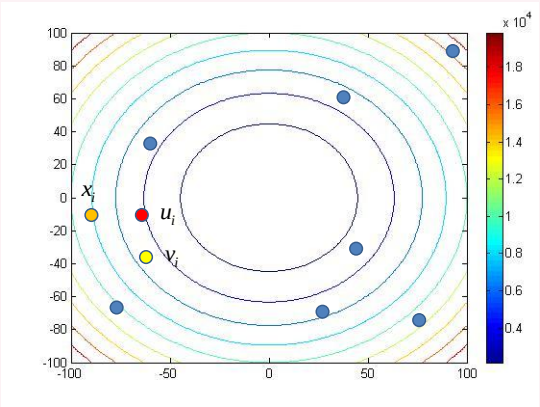
4. 简单示例

示例: 变异算子



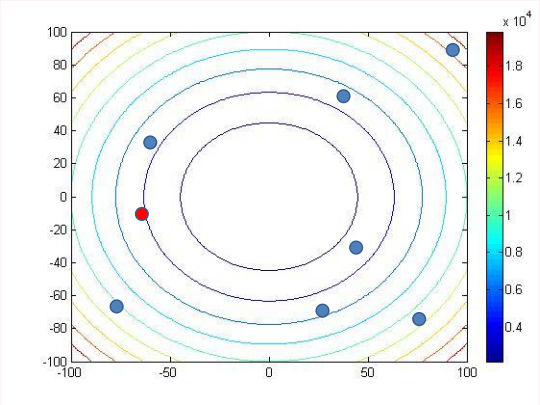
4. 简单示例

示例: 重组 (杂交) 算子



4. 简单示例

示例: 选择算子



算法性能

- 目前, DE 算法还没有收敛性证明;
- 但是, DE 算法在很多优化问题中取得了很好的优化结果;
- Storn和 Price 在 1997 年验证了 DE 算法优于模拟退火算法和遗传算法;
- Ali 和Törn 于 2004 年验证了 DE 算法在求解精度和有效性上优于控制随机搜索算法和另一遗传算法;
- Lampinen 和 Storn 于 2004 年验证了 DE 算法在求解精度和收敛速度上优于遗传算法, 进化规划和模拟退火算法.

算法应用

思考

- DE的产生与测试在何处？
- DE的搜索方向和搜索步长如何确定？
- DE如何实现信息共享？
- DE如何实现资源竞争？
- DE的勘探与开采如何实现？

进一步资料

- R. Storn, K. Price, “Differential evolution – A simple and efficient heuristic for global optimization over continuous spaces,” *J. of Global Optim.* 1997, 11 (4): 341 - 359.
- S. Das, P. N. Suganthan, “Differential evolution: A survey of the state-of-the-art,” *IEEE Trans. on Evol. Comput.* 2011, 15 (1): 4 - 31.
- K. Price and R. Storn, “Differential evolution homepage,” http://www1.icsi.berkeley.edu/~storn/code.html, 2018.

课外作业

在网上下载差分演化算法源程序，读懂，并尝试独立实现。要求：

1. 对每一个语句进行中文注释。

5. 小结

算法简介

算法基本流程

简单示例

小结

5. 小结

本章小结

1. 差分演化算法简介
2. 差分演化算法流程
3. 差分演化算法示例
4. 差分演化算法特点与应用

5. 小结

思考

在网上下载差分演化算法源程序，读懂，并尝试独立实现。思考以下问题：

1. 调整算法的三个参数，看是否对结果有影响？
2. 选择不同的差分变异算子，看是否对结果有影响？

Thank you!

AUTHOR: GONG, Wenyin
ADDRESS: School of Computer Science,
China University of Geosciences,
Wuhan, 430074, China
E-MAIL: wygong@cug.edu.cn
HOMEPAGE: http://grzy.cug.edu.cn/gongwenyin