# 带局部增强算子的微分进化改进算法

赵光权,彭喜元,孙 宁
(哈尔滨工业大学科学园 3033 信箱,黑龙江哈尔滨 150080)

**摘 要**: 微分进化算法具有鲁棒性强、易于使用、控制参数少等优点.但在搜索中存在一定的盲目性,搜索效率不高.本文通过引入局部增强算子,使种群中的部分个体在当前最优个体附近寻优,以加快算法的收敛速度.利用五个标准的优化算法测试函数对改进算法进行了测试,并与动态微分进化算法和微粒群算法进行了比较.算法分析和仿真结果表明,本文提出的改进算法大大提高了算法搜索效率.

**关键词**: 全局优化; 微分进化算法; 局部增强算子
**中图分类号**: TP18 **文献标识码**: A **文章编号**: 0372-2112 (2007) 05-0849-05

# A Modified Differential Evolution Algorithm with Local Enhanced Operator

ZHAO Guang-quan, PENG Xi-yuan, SUN Ning
*(Science Park of Harbin Institute of Technology, P. O. Box 3033, Harbin, Heilongjiang 150080, China)*

Abstract: The differential evolution algorithm is robust, easy to use, and requires few control parameters. However, as the search of the algorithm is of some blindness, its efficiency is limited. To improve the efficiency of the algorithm, the local enhanced operator is proposed to make some individuals of the population search around the current best individual. Numerical study is carried out using five benchmark functions, and the result is compared with that of dynamic differential evolution and particle swarm optimization. Analysis and simulation results show that the efficiency of modified differential evolution is significantly improved.

Key words: global optimization; differential evolution algorithm; local enhanced operator

## 1 引言

现实中的许多问题都可以归结为连续空间上的全局优化问题,因此求解全局优化问题成为一个重要的课题.当问题是非线性的、不可微的甚至是没有函数表达式的时候,随机优化算法是比较好的选择.目前较流行的随机优化算法包括模拟退火算法、遗传算法、微粒群算法等.用户对优化算法的要求通常可以概括为以下四个方面:(1)能够处理非线性、不可微、多峰函数;(2)可实现并行计算;(3)算法易使用,也就是说算法所需控制参数少,控制参数易于选择,且鲁棒性好;(4)良好的收敛特性,也就是说算法连续多次运行都可以收敛于全局最优.1995 年 Rainer Storn 和 Kenneth Price 提出的微分进化算法(differential evolution algorithm,以下简称 DE 算法)满足以上所有要求[1].

DE 算法是一种实数编码的基于种群进化的全局优化算法,它在许多优化问题中都表现出优于自适应模拟退火算法、PSO 算法、GA 算法的性能.DE 算法在滤波器设计、神经网络参数训练、聚类分析、机器人路径规划等工程领域取得了良好的应用效果[2~5].目前已提出了一些 DE 改进算法,最新的改进算法有 Anyong Qing 提出的动态微分进化算法(简称 DDE)[6],Paul K Bergey 提出的改进算法 MDE[7], Swagatam Das 提出的 DERSF 和 DETVSF[8],Jianyong Sun 等提出的 DE/EDA 混合算法等[9],这些改进不同程度地提高了 DE 算法的搜索效率.尽管如此,微分进化算法在搜索速度上仍有提高的空间.本文提出了一种新的微分进化改进算法 MPDE,MPDE 通过引入局部增强算子,大大加快了算法的收敛速度.

## 2 DE 基本算法

DE 算法是求解有 $n$ 个连续变量全局优化问题的算法.全局优化问题都可以转化为求解如下的最小化问题:

$$\min f(X) \tag{1}$$

其中 $X=(x_1, x_2, \cdots, x_n) \in D \subset R^n$ 是连续变量,目标函数 $f: D \to R$ 可能是不可微的.

DE 基本算法的流程图如图 1 所示.

收稿日期:2006-06-15;修回日期:2006-09-06

假设 DE 算法种群规模为 $NP$,每个个体有 $D$ 维变量,则第 $G$ 代的个体可表示为 $X_{i,G}$, $i=1,2,\cdots,NP$. DE 算法的主要算子包括:变异、交叉、选择.

DE 算法和其它进化算法的主要区别是变异方式. 变异操作后得到中间个体记为 $v_{i,G+1}$,即

$$v_{i,G+1}=X_{r1,G}+F\cdot(X_{r2,G}-X_{r3,G}) \quad (2)$$

其中 $r_1,r_2,r_3\in\{1,2,\cdots,NP\}$ 且 $r_1\neq r_2\neq r_3\neq i$, $F$ 为一常数. 这种利用随机偏差扰动产生新个体的方式可以获得一个具有非常好收敛性质的自适应程序[10].

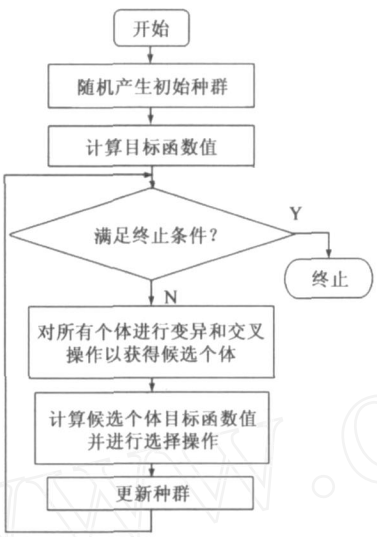


图 1 DE 基本算法流程图

交叉操作如下:将变异得到的中间个体 $v_{i,G+1}=(v_{1i,G+1},v_{2i,G+1},\cdots,v_{Di,G+1})$ 和当前个体 $X_{i,G}=(X_{1i,G},X_{2i,G},\cdots,X_{Di,G})$ 进行杂交,如式(3)所示. 经过杂交后得到当前个体的候选个体 $u_{i,G+1}=(u_{1i,G+1},u_{2i,G+1},\cdots,u_{Di,G+1})$.

$$u_{ji,G+1}=\begin{cases}v_{ji,G+1}, & \text{if } (randb(j)\le CR) \text{ 或 } j=rnbr(i)\\ x_{ji,G}, & \text{else}\end{cases} \quad (3)$$

其中 $i=1,\cdots,NP$. $j=1,\cdots,D$. $rnbr(i)$ 是一个随机参数保证 $u_{i,G+1}$ 至少从 $v_{i,G+1}$ 中取到一个分量值, $randb(j)\in[0,1]$ 是一个均匀分布的随机数,杂交因子 $CR\in[0,1]$ 是 DE 算法的一个重要参数,它决定了中间个体分量值代替当前个体分量值的概率.

选择操作如下:对候选个体 $u_{i,G+1}$ 进行适应度评价,然后根据式(4)决定是否选取新产生的个体.

$$X_{i,G+1}=\begin{cases}u_{i,G+1}, & \text{if } f(u_{i,G+1})\le f(X_{i,G})\\ X_{i,G}, & \text{else}\end{cases} \quad (4)$$

DE 算法有三个控制参数: $F$、$CR$、$NP$. 种群规模 $NP$ 取值一般为变量维数 $D$ 的 2 倍~20 倍,且至少应大于 4. $F$ 取值范围一般为 0.4~1.0,建议初始值选为 0.5. CR 初始值建议选为 0.5.

DE 算法有几种变形,为了区分不同的 DE 算法,通常将 DE 表示为:DE/ $x$/ $y$/ $z$. 其中 $x$ 表示式(2)中的基点向量是随机选取的还是选择当前种群中的最优个体, $y$ 表示差分向量的个数, $z$ 表示交叉方式,有指数交叉和二项式交叉两种. 本文中所描述的 DE 基本算法可表示为 DE/ rand/ 1/ bin,即基点向量为随机选取,差分向量一个,二项式交叉方式.

## 3 改进算法 MPDE

### 3.1 MPDE 算法原理

DE 算法是模拟自然进化的随机搜索算法,其核心思想是利用随机偏差扰动产生新的中间个体. 其产生中间个体的方式决定了 DE 算法在许多问题上都有很好的收敛表现,但搜索速度则相对缓慢,局部搜索能力不强,尤其是逼近全局最优解时仍需要经过很多代才能获得最优值. 本文在对 DE 基本算法特性进行分析的基础上,提出了一种加速收敛的微分进化改进算法 MPDE. 其基本思想如下:在按 DE 基本算法得到新种群后,以 $MP(0<MP<1)$ 概率对新种群中的部分个体(不含当前最优个体)重新赋值,并使这部分个体分布在当前种群中的最优个体附近,以增强这部分个体的贪婪性,使算法加速收敛. 为此,引入局部增强算子,如式(5)所示:

$$C_{i,G+1}=X_{best,G+1}+(X_{r1,G+1}-X_{r2,G+1})/gen \quad (5)$$

其中 $X_{best,G+1}$ 为新种群中的当前最优个体, $C_{i,G+1}$ 为增强后的新个体,用于替换旧个体 $X_{i,G+1}$, $X_{r1,G+1}$、$X_{r2,G+1}$ 从新种群中随机选取且满足 $r_1\neq r_2\neq i$, $gen$ 为算法迭代次数. 当一定数量的个体被选中并按局部增强算子更新时,其效果相当于使这部分个体在当前种群中的最优个体附近随机扰动. 为了控制进行局部增强个体的数量,MPDE 算法引入了新参数 $MP$,当 $MP=0$ 时,没有个体被选中进行局部增强,算法与 DE 基本算法一致,当 $MP=1$ 时,算法每迭代一次,种群中除最优个体外其它个体都按式(5)更新一次.

MPDE 算法的实质是使种群中的部分个体在当前最优个体附近寻优,其目标是在保证种群多样性的同时,增加部分个体的贪婪性,以保证算法又快又好地找到全局最优解. 引入局部增强算子以后,算法每迭代一次,就会使部分个体变得更贪婪,并在当前最优解附近寻优,同时,并不是所有个体都变得更贪婪,从而使得算法的贪婪性又受到一定限制,以避免影响算法的良好收敛性. 需要指出的是,随着收敛过程的进行,这部分个体的随机扰动范围也动态缩小,这种设计

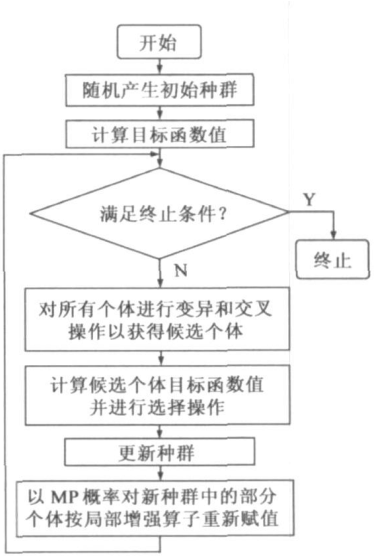


图 2 MPDE 算法流程图

可以增强算法的局部搜索能力,加快算法的收敛速度,尤其是在逼近全局最优解时,可以减少收敛所需迭代次数,当然也有陷入局部最优的风险.

### 3.2 MPDE 算法流程

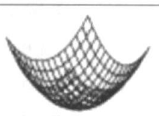
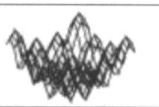
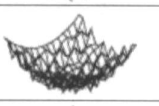
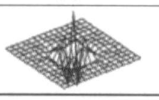
MPDE 算法的流程图如图 2 所示. MPDE 算法仅在 DE 基本算法流程的基础上增加一个步骤:按 DE 基本算法得到新种群后以一定概率 $MP$ 对新种群中除最优个体以外的其它个体按局部增强算子重新计算并替换这些个体. 显然,在每次的迭代过程中,MPDE 算法并没有增加对目标函数的评价次数,且对算法整体计算量影响很小.

## 4 MPDE 算法仿真实验

### 4.1 测试函数

为了检验 MPDE 算法的性能,选择了 5 个常用的优化算法测试函数对算法进行测试,它们分别是 Sphere 函数 ($f_1$)、Rosenbrock 函数 ($f_2$)、Rastrigin 函数 ($f_3$)、Griewank 函数($f_4$)和 Schaffer 函数($f_5$). 这 5 个函数的表达式分别见式(6)～式(10),测试函数描述如表 1 所示. 除了 Schaffer 是 2 维函数外,其它函数都用 30 维变量进行测试. 5 个函数中前两个为单峰函数,只有一个极小值,其中 Rosenbrock 函数较难找到全局最优解. 后三个为多峰函数,有多个局部极小值,其中 Rastrigin 和 Griewank 函数有很大的曲率,可以引导搜索到全局极小,而 Schaffer 函数在全局极小附近剧烈震荡,一般算法难以得到最优解.

表 1　测试函数描述

| 函数 | 维数 $n$ | 范围 $[x_{\min}, x_{\max}]$ | 最优值 $f^*$ | 停止条件 | 2 维图形 |
|---|---|---|---|---|---|
| $f_1$ | 30 | $[-100,100]^n$ | 0 | 0.01 |  |
| $f_2$ | 30 | $[-30,30]^n$ | 0 | 100 |  |
| $f_3$ | 30 | $[-5.12,5.12]^n$ | 0 | 100 |  |
| $f_4$ | 30 | $[-600,600]^n$ | 0 | 0.1 |  |
| $f_5$ | 2 | $[-100,100]^n$ | 0 | $10^{-5}$ |  |

$$f_1(\vec{x}) = \sum_{i=1}^{n} x_i^2 \tag{6}$$

$$f_2(\vec{x}) = \sum_{i=1}^{n-1} (100(x_{i+1} - x_i^2)^2 + (x_i - 1)^2) \tag{7}$$

$$f_3(\vec{x}) = \sum_{i=1}^{n} (x_i^2 - 10\cos(2\pi x_i) + 10) \tag{8}$$

$$f_4(\vec{x}) = \frac{1}{4000}\sum_{i=1}^{n} x_i^2 - \prod_{i=1}^{n} \cos\left(\frac{x_i}{\sqrt{i}}\right) + 1 \tag{9}$$

$$f_5(\vec{x}) = 0.5 + \frac{\sin^2\sqrt{x_1^2 + x_2^2} - 0.5}{(1 + 0.001(x_1^2 + x_2^2))^2} \tag{10}$$

### 4.2 实验结果及分析

为了使 MPDE 算法的测试结果更具对比性,我们将 MPDE 与 DE 基本算法,以及最近提出的 DE 改进算法 DDE 进行了比较. 此外,当 DE 和 PSO 算法所采用的种群规模一致时,每次迭代过程中这两种算法对目标函数的评价次数一样,而 PSO 算法以算法简洁、收敛速度快等优点著称,因此仿真结果中还列出了 PSO 算法对 5 个测试函数的仿真结果,PSO 算法的仿真结果引自参考文献[11].

各算法实验结果如表 2～表 6 所示. 每种算法针对 5 个测试函数都运行 20 次,取其最大迭代次数(max)、最小迭代次数(min)、搜索成功率($ps$),并计算搜索成功情况下的平均迭代次数(avg)和平均收敛时间($t$),由于文献[11]中未给出 PSO 算法的平均收敛时间,因此此栏空缺. 在表 2～表 6 中,DE7 代表 DE 基本算法 DE/rand/1/bin,DDE7 为基于 DE/rand/1/bin 策略的动态微分进化算法,DDE1 则为基于 DE/best/1/exp 策略的动态微分进化算法. 为了便于比较,表中给出了 MPDE 算法在不同 $MP$ 取值时的实验结果,MPDE(0.01)表示 $MP$ 取值为 1%的实验结果,依此类推.

本文中的所有算法对 5 个测试函数设置的种群数量都为 60 个,最大迭代次数为 10000 次,算法迭代 10000 次还没有达到表 1 所列的停止条件则认为本次搜索失败. DE 算法其它参数设置为: $F = 0.5$, $CR = 0.5$; PSO 算法参数设置如下: $w = 0.729$; $c_1 = c_2 = 1.494$.

本文所列 DE 相关算法均采用 C 语言进行编写,开发环境为 LabWindows/CVI 6.0,这是一个 C 语言编程环境,所用计算机为 IBM-T30 笔记本电脑,主频 2.0GHz,内存 256M,操作系统为 Windows 2000.

表 2　$f_1$ 函数仿真结果

| 算法 | $ps$ | min | max | avg | $t$(s) |
|---|---|---|---|---|---|
| DE7 | 100% | 474 | 500 | 487 | 0.5157 |
| DDE7 | 100% | 438 | 476 | 453 | 0.4360 |
| DDE1 | 100% | 366 | 402 | 383 | 0.1955 |
| PSO | 100% | 269 | 368 | 314 | — |
| MPDE(0.01) | 100% | 398 | 450 | 417 | 0.4196 |
| MPDE(0.05) | 100% | 248 | 275 | 260 | 0.2656 |
| MPDE(0.1) | 100% | 187 | 219 | 201 | 0.2081 |
| MPDE(0.2) | 100% | 248 | 471 | 341 | 0.3588 |

表 3 $f_2$ 函数仿真结果

| 算法 | ps | min | max | avg | t(s) |
|---|---|---|---|---|---|
| DE7 | 100 % | 474 | 987 | 571 | 0.6016 |
| DDE7 | 100 % | 424 | 889 | 536 | 0.5108 |
| DDE1 | 100 % | 268 | 456 | 330 | 0.1715 |
| PSO | 100 % | 219 | 4450 | 611 | — |
| MPDE(0.001) | 100 % | 460 | 681 | 528 | 0.5316 |
| MPDE(0.01) | 100 % | 347 | 580 | 395 | 0.3998 |
| MPDE(0.05) | 100 % | 194 | 598 | 293 | 0.3165 |
| MPDE(0.1) | 100 % | 177 | 682 | 336 | 0.3691 |

表 4 $f_3$ 函数仿真结果

| 算法 | ps | min | max | avg | t(s) |
|---|---|---|---|---|---|
| DE7 | 100 % | 988 | 6446 | 3961 | 5.5351 |
| DDE7 | 100 % | 1140 | 5000 | 3952 | 5.1002 |
| DDE1 | 100 % | 120 | 164 | 142 | 0.1217 |
| PSO | 100 % | 119 | 214 | 166 | — |
| MPDE(0.001) | 100 % | 688 | 1276 | 919 | 1.2265 |
| MPDE(0.01) | 100 % | 250 | 330 | 286 | 0.3868 |
| MPDE(0.1) | 100 % | 59 | 81 | 69 | 0.0952 |
| MPDE(0.3) | 90 % | 30 | 189 | 56 | 0.0816 |

表 5 $f_4$ 函数仿真结果

| 算法 | ps | min | max | avg | t(s) |
|---|---|---|---|---|---|
| DE7 | 100 % | 427 | 487 | 455 | 0.7043 |
| DDE7 | 100 % | 413 | 543 | 442 | 0.6385 |
| DDE1 | 100 % | 335 | 410 | 377 | 0.3782 |
| PSO | 100 % | 266 | 328 | 287 | — |
| MPDE(0.005) | 100 % | 396 | 473 | 429 | 0.6396 |
| MPDE(0.01) | 100 % | 366 | 443 | 395 | 0.5897 |
| MPDE(0.1) | 100 % | 164 | 208 | 181 | 0.2748 |
| MPDE(0.15) | 90 % | 160 | 278 | 201 | 0.3096 |

表 6 $f_5$ 函数仿真结果

| 算法 | ps | min | max | avg | t(s) |
|---|---|---|---|---|---|
| DE7 | 100 % | 156 | 306 | 207 | 0.0787 |
| DDE7 | 100 % | 166 | 294 | 203 | 0.0835 |
| DDE1 | 50 % | 39 | 63 | 51 | 0.0290 |
| PSO | 95 % | 83 | 2361 | 319 | — |
| MPDE(0.001) | 100 % | 148 | 247 | 185 | 0.0697 |
| MPDE(0.005) | 100 % | 109 | 172 | 144 | 0.0545 |
| MPDE(0.008) | 100 % | 90 | 164 | 119 | 0.0458 |
| MPDE(0.01) | 90 % | 75 | 154 | 111 | 0.0423 |

从仿真数据可以看出,通过调整参数 $MP$,MPDE 算法针对 5 个测试函数都取得了很好的效果. MPDE 算法在保证搜索成功率的基础上,大大降低了收敛所需迭代次数和时间. MPDE 算法的迭代次数和收敛时间与 DE 基本算法相比减少了 45 %以上. 显而易见,MPDE 算法不仅大大优于 DE 基本算法,总体上也优于 DDE 算法和 PSO 算法. 本文所提的改进算法当然也有其局限性,当目标函数在全局极小处剧烈震荡时,改进算法的全局收敛性可能会变差,并使搜索成功率有所下降.

从仿真结果不难发现,在一定范围内,随着 $MP$ 取值的增大,MPDE 算法收敛速度随之变快,同时,当 $MP$ 达到一定数值时,算法会产生"早熟"的情况. 5 个测试函数中,对于较简单的 $f_1$、$f_3$、$f_4$ 函数而言,当 $MP = 0.1$ 时 MPDE 算法效果最佳,对于相对复杂的 $f_2$ 函数,当 $MP = 0.05$ 时 MPDE 算法效果最佳,对于最复杂的 $f_5$ 函数,当 $MP = 0.008$ 时 MPDE 算法效果最佳. 因此,在使用该算法进行优化应用时,需根据实际情况,使 $MP$ 从较小的数值(例如 0.001)开始,再逐渐加大其数值以获得最佳效果,对于较简单的优化问题,建议 $MP$ 取值为 0.1 左右,对于相对复杂的优化问题,$MP$ 取值则不宜过大,应控制在 0.05 以内.

## 5 结论

本文提出了一种微分进化改进算法—MPDE 算法,该算法引入了局部增强算子,增强了算法的局部搜索能力,在保证算法良好收敛性的同时,大大加快了算法收敛速度. 对 5 个测试函数的仿真结果表明 MPDE 算法相对于 DE 基本算法、DDE 改进算法等的性能有了较大提高,算法收敛所需迭代次数和收敛时间比 DE 基本算法减少了 45 %以上.

参考文献:

[1] Rainer Storn, Kenneth Price. Differential evolution —A simple and efficient heuristic for global optimization over continuous spaces [J]. Journal of Global Optimization, 1997, 11(4): 341 - 359.

[2] Rainer Storn. Designing nonstandard filters with differential evolution[J]. IEEE Signal Processing Magazine, 2005, 22(1): 103 - 106.

[3] Chong-wei Chen, De-zhao Chen, Guang-zhi Cao. An improved differential evolution algorithm in training and encoding prior knowledge into feedforward networks with application in chemistry[J]. Chemometrics and Intelligent Laboratory Systems, 2002, 64(1): 27 - 43.

[4] Sandra Paterlinia, Thiemo Krinkb. Differential evolution and particle swarm optimisation in partitional clustering[J]. Computational Statistics & Data Analysis, 2006, 50(5): 1220 - 1247.

[5] 冯琦,周德云. 基于微分进化算法的时间最优路径规划[J]. 计算机工程与应用, 2005, 41(12): 74 - 75.

[6] Anyong Qing. Dynamic differential evolution strategy and applications in electromagnetic inverse scattering problems[J]. IEEE Transactions on Geoscience and Remote Sensing, 2006,

44(1) :116 - 125.

[7] Paul K Bergey, Cliff Ragsdale. Modified differential evolution: A greedy random strategy for genetic recombination[J]. The International Journal of Management Science, 2005, 33(3) :255 - 265.

[8] Swagatam Das, Amit Konar, Uday K Chakraborty. Two improved differential evolution schemes for faster global search [A]. Genetic and Evolutionary Computation Conference (GECCO '05) [C]. Washington, DC, USA, 2005. 991 - 998.

[9] Jianyong Sun, Qingfu Zhang, Edward P K Tsang. DE/ EDA: A new evolutionary algorithm for global optimization[J]. Information Sciences, 2005, 169(3 - 4) :249 - 262.

[10] 李颖,徐桂芝,等. 微分进化算法在头部电阻抗成像中的应用[J]. 中国生物医学工程学报, 2005, 24(6) :672 - 675.

[11] Ioan Cristian Trelea. The particle swarm optimization algorithm: Convergence analysis and parameter selection[J]. Information Processing Letters, 2003, 85(6) :317 - 325.

**作者简介**:

**赵光权**　男,1978 年出生于浙江省东阳市,在读博士研究生,哈尔滨工业大学自动化测试与控制系讲师. 研究方向为计算智能、故障诊断等. E-mail: hit53zhao @hit. edu. cn

**彭喜元**　男,1961 年生于内蒙古四子王旗,博士,哈尔滨工业大学自动化测试与控制系教授,博士生导师. 研究方向为自动测试技术及系统、计算智能、故障诊断技术及应用.