文章编号: 1001-0920 (2016) 08-1379-08 DOI: 10.13195/j.kzyjc.2015.0779

基于最优高斯随机游走和个体筛选策略的差分进化算法

李牧东, 赵 辉, 翁兴伟, 韩 统

(空军工程大学 航空航天工程学院，西安 710038)

摘 要: 针对差分进化算法开发能力较差的问题, 提出一种具有快速收敛的新型差分进化算法. 首先, 利用最优高斯随机游走策略提高算法的开发能力; 然后, 采用基于个体优化性能的简化交叉变异策略实现种群的进化操作以加强其局部搜索能力; 最后, 通过个体筛选策略进一步提高算法的探索能力以避免陷入局部最优. 12 个标准测试函数和两种带约束的工程优化问题的实验结果表明, 所提出的算法在收敛速度、算法可靠性及收敛精度方面均优于 EPSDE、SaDE、JADE、BSA、CoBiDE、GSA 和 ABC 等算法, 在加强算法探索能力的同时能够有效地提高算法的开发能力.

关键词: 差分进化；无约束优化；约束优化；高斯随机游走；个体筛选

中图分类号: TP391 **文献标志码:** A

Differential evolution based on optimal Gaussian random walk and individual selection strategies

LI Mu-dong, *ZHAO Hui*, *WENG Xing-wei*, *HAN Tong*

(Department of Aeronautics and Astronautics Engineering，Air Force Engineering University，Xi'an 710038，China. Correspondent: LI Mu-dong，E-mail：modern_lee@163.com)

Abstract: To solve the problems of poor performance in exploitation of the differential evolution(DE) algorithm, a new DE algorithm with fast convergence rate is proposed. Firstly, the optimal Gaussian random walk strategy is used to improve the exploitation ability of the algorithm. Then, the simplified crossover and mutation strategy based on the individuals' optimization performance is employed to realize the evolution operation so as to improve the performance of local search. Finally, the individual selection strategy is proposed to avoid local optimum and enhance the exploration performance. Experimental results of 12 unconstrained benchmark functions and two constrained engineering design optimization problems show that the proposed algorithm is superior to the algorithm of EPSDE, SaDE, JADE, BSA, CoBiDE, GSA and ABC in terms of convergence rate, stability and convergence accuracy. The proposed algorithm can effectively enhance the exploration performance and improve the exploitation ability.

Keywords: differential evolution; unconstrained optimization; constrained optimization; Gaussian random walk; individual selection

0 引 言

差分进化算法 (DE)[1]是 Storn 等提出的一种结构简单且高效的进化算法 (EA). DE 算法通过变异、重组和选择操作模拟了自然界中种群进化的过程. 由于其具有较好的收敛性能和易于操作等特点, 已经成为了进化算法中求解全局优化问题最常用的算法之一[2]. 目前, 国内外学者针对 DE 算法的研究主要集中于两个方面: 针对不同类型的优化问题, 对 DE 算法实验向量的生成策略进行改进; 针对 DE 算法中的两个控制参数, 增益常数 F 和交叉概率 CR, 提出不同的改进策略. 虽然 DE 算法在模式识别、工程设计优化、电力系统及经济等多个领域的成功应用验证了其有效性, 并展现了其独特的优势[3], 但是也暴露了诸多不足, 如易于陷入局部最优、收敛速度较慢且收敛精度不高等[4].

为了提高 DE 算法的性能, 国内外学者相继提出了一些改进算法. 在对 DE 算法交叉变异策略的研究方面, Fan 等[5]提出了一种基于三角函数的变异策略,

收稿日期: 2015-06-17；**修回日期:** 2015-10-28.

基金项目: 航空科学基金项目(20105196016)；中国博士后科学基金项目(2012M521807).

作者简介: 李牧东(1987–), 男, 博士生, 从事无人飞行器武器系统总体技术、智能优化算法的研究；赵辉(1973–), 男, 教授, 博士生导师, 从事武器系统运用与工程、进化计算等研究.

提高了算法在收敛速度与鲁棒性之间的平衡能力; Mallipeddi 等[6]提出了一种基于系统的变异和交叉策略以及相关控制参数的改进算法 (EPSDE), 通过设置不同的变异、交叉策略池和与之相对应的控制参数池实现对进化过程中不同的策略选择, 以提高算法的优化性能. Civicioglu[7]通过设置历史种群和 0、1 整数矩阵的方式简化了差分进化算法的结构, 对于一些特定的优化问题表现出了较快的收敛性能.

针对 DE 算法的两个控制参数, Qin 等[8]提出了一种自适应差分进化算法 (SaDE), 通过设计的选择学习策略自适应地调整算法的控制参数, 从而加快算法的收敛速度. 为了提高算法针对不同优化问题控制参数之间的协调性, 文献 [9] 提出了一种新型自适应 DE 算法 (JADE), 通过记录历史中 p 个最优个体的方式产生新的实验向量, 同时调整控制参数的生成方式以提高其优化性能. 文献 [10] 提出了 DE 算法的改进算法 (CoBiDE), 利用双峰分布策略针对每代种群产生新的控制参数, 同时利用协方差矩阵学习策略产生新的种群, 从而提高算法的收敛速度.

上述改进算法通过对变异、交叉操作或控制参数进行改进, 虽然取得了一定的效果, 但是在如何平衡算法的探索和开发能力这一方面仍需要进一步的研究. 因此, 本文在对 DE 算法研究的基础上, 通过设计的最优高斯随机游走策略、改进的交叉变异操作以及个体筛选策略, 提出了基于最优高斯随机游走和个体筛选策略的 DE 算法 (GSDE), 以期在加强 DE 算法探索性能的同时, 进一步提高其开发能力.

1 GSDE 算法

由于标准 DE 算法的交叉、变异操作在一定程度上存在盲目性, 导致算法具有较好的探索能力, 但同时, 其开发能力较差. 如何较好地平衡这两种能力, 已成为提高 DE 算法优化性能的关键问题. 本文针对这一问题, 首先利用最优高斯随机游走策略, 在巩固算法探索能力的同时, 提高其开发能力; 其次利用个体筛选策略进一步加强算法的局部搜索能力; 最后根据种群个体优化性能的优劣选择是否进行变异操作, 以提高算法操作的针对性, 并加强算法的探索能力.

1.1 最优高斯随机游走策略

高斯随机游走模型是随机游走模型中较为典型的一种, 具有较强的开发能力[8]. 因此, 本文利用高斯随机游走的特点, 通过采用当前最优个体进行引导, 产生新的随机种群, 以平衡其探索与开发能力. 下式为基于当前最优个体的高斯随机游走策略:

$$\mathrm{Rw}_i^G = \mathrm{Gaussian}(x_{\mathrm{best}}^G, \tau) + (r_1 \cdot x_{\mathrm{best}}^G - r_2 \cdot x_i^G). \quad (1)$$

其中: x_i^G 是种群中第 i 个个体, x_{best}^G 是第 G 代种群中的最优个体; r_1 和 r_2 是服从 [0, 1] 间均匀分布的随机数; 步长 τ 表示为

$$\tau = \frac{\log(G)}{G}(x_i^G - x_{\mathrm{best}}^G). \quad (2)$$

从式 (2) 可以看出, 通过当前最优个体的引导, 使种群在最优个体附近产生新的个体, 有利于算法的快速收敛; 同时, 通过对高斯分布的方差 τ 进行自适应控制, 使得算法在迭代初期, 具有较大的 τ 值, 因此具有较强的探索能力; 随着迭代次数的增加, τ 逐渐减小, 从而提高了算法的局部开发能力.

1.2 改进的交叉变异操作

为了进一步提高 DE 算法的开发能力, 首先利用种群个体的适应度函数值来判断个体的优化性能, 然后根据个体优化性能的好坏进行变异操作, 即优化性能较差的个体进行变异操作, 反之, 优化性能较优的个体保持不变, 如下式所示:

$$x_{i,j}^{G+1} = \begin{cases} x_{i,j}^{G+1}, \ r < P_{\mathrm{rank}(i)}; \\ x_{t_1,j}^G - \mathrm{rand} \cdot (x_{t_2,j}^G - x_{t_3,j}^G), \ \mathrm{else}. \end{cases} \quad (3)$$

其中: t_1、t_2 和 t_3 为 $\{1, 2, \cdots, N\}$ 中随机选择的个体, 且互不相等, N 为种群规模; rand 和 r 为 $[0, 1]$ 之间的随机数; $P_{\mathrm{rank}(i)}$ 为当前种群第 i 个个体对应的优化性能级别, 按下式计算:

$$P_{\mathrm{rank}(i)} = \frac{N - i + 1}{N}, \quad (4)$$

$\mathrm{rank}(i)$ 为个体 x_i^G 在种群 $\boldsymbol{X}$ 中根据适应度值从小到大的顺序.

从式 (3) 和 (4) 可以看出, 具有较好优化性能的个体, 保持不变的概率较大; 相反, 具有较差优化性能的个体, 进行变异操作的概率也较大. 这一策略使得变异操作更具有针对性, 从而提高了种群进化的效率, 简化了标准 DE 算法的交叉变异操作, 并提高了个体的局部探索能力.

1.3 个体筛选策略

为了避免种群陷入局部最优, 加强算法的探索能力, 利用种群个体的适应度值按照概率对个体进行筛选操作, 以提高种群的多样性. 首先, 根据种群个体的适应度值对其进行排序; 其次, 当随机数 rand 小于 $\mathrm{rank}(i)/N$ 时, 根据下式对个体 x_i^{G+1} 进行操作:

$$x_i'^{G+1} = \begin{cases} x_i^{G+1} - \eta(x_{t_1}^{G+1} - x_{\mathrm{best}}^G), \ r > 0.5; \\ x_i^{G+1} + \eta(x_{t_1}^{G+1} - x_{t_2}^{G+1}), \ \mathrm{else}. \end{cases} \quad (5)$$

其中: t_1 和 t_2 是 $\{1, 2, \cdots, N\}$ 中随机选择的个体, 且互不相等; η 和 r 是 $[0, 1]$ 之间的随机数. 不同于上一操作, 式 (5) 是基于种群个体间的计算, 而式 (3) 是基于个体中元素的操作. 可以看出, 通过个体筛选策略, 适应度值较差的个体具有较大概率被淘汰, 并根据式 (5) 产生新的个体; 同时, 式 (5) 分别利用了两种不同类型的变异方程, 即随机搜索方程和基于最优个体的搜索方程, 以在加强算法探索能力的同时提高其开发能力.

1.4 改进算法分析及步骤

在标准DE算法的基础上, 从提高算法的开发能力并加强算法的探索能力的角度出发, 首先利用基于当前最优个体的高斯随机游走策略产生新的种群, 并利用 $\log(G)/G$ 控制步长以平衡算法的探索能力和开发能力; 然后采用改进的交叉变异策略对更新后的种群 $\boldsymbol{X}$ 进行操作, 通过计算优化性能级别 $P_{\mathrm{rank}(i)}$ 对优化性能较差的个体, 有针对性地进行个体中元素的变异操作, 以提高算法的局部搜索能力; 最后利用个体筛选策略, 根据个体适应度值的大小进行排序, 针对较差个体, 利用不同的个体变异搜索方程对其进行变异操作, 在加强算法探索能力的同时, 考虑了算法的开发能力. 根据上述3个操作过程, 所提出的改进算法的步骤总结如下.

Step 1: 在搜索空间 [low, up] 中随机产生种群规模为 N, 维数为 D 的初始种群;

Step 2: 计算种群适应度函数值, 函数评价次数 $\mathrm{FEs} = N$;

Step 3: While 算法终止条件不满足

　　for 最优高斯随机游走策略

　　根据式 (1) 产生新的个体 Rw_i^G

　　end for;

Step 4: 计算种群适应度函数值, $\mathrm{FEs} = \mathrm{FEs} + N$;

Step 5: 保留适应度值较优的个体并更新种群 $\boldsymbol{X}$;

Step 6: 由式 (4) 计算个体优化性能级别 $P_{\mathrm{rank}(i)}$;

Step 7: for 改进的交叉变异操作

　　根据式 (3) 产生新的种群个体 x_i^{G+1}

　　end for;

Step 8: 计算种群适应度函数值, $\mathrm{FEs} = \mathrm{FEs} + N$;

Step 9: 保留适应度值较优的个体并更新种群 $\boldsymbol{X}$;

Step 10: 对种群个体进行排序并计算 $\mathrm{rank}(i)/N$;

Step 11: for 个体筛选策略

　　if $\mathrm{rand} < \mathrm{rank}(i)/N$

　　根据式 (5) 产生新的个体

　　保留较优个体并更新种群 $\boldsymbol{X}$

　　$\mathrm{FEs} = \mathrm{FEs} + 1$;

　　end if

　　end for;

Step 12: end While;

Step 13: 输出最优解及最优目标函数值.

2 数值实验与分析

通过分别对非约束优化问题和约束优化问题进行测试, 并与DE算法、其他DE改进算法及其他进化算法进行比较, 验证改进算法的优化性能. 选择12个经典测试函数[12]作为非约束优化问题; 将压力容器设计问题和张力/压缩弹簧设计问题作为约束优化问题[12]. 另外, 本文采用Matlab R2013a进行仿真, 运行环境为Intel(R) Core(TM) i5-3470处理器, 4 G内存.

2.1 非约束优化问题

为了验证本文所提算法对于求解非约束优化问题的性能, 选取了12个经典测试函数[12]. 同时, 选取了5种目前较为流行的差分进化算法的改进算法 (EPSDE[6]、BSA[7]、SaDE[8]、JADE[9]、CoBiDE[10]) 及2种经典智能优化算法作为比较算法 (人工蜂群算法ABC[13]和引力搜索算法GSA[14]).

实验以函数的评价次数作为衡量标准, 表1给出了12个测试函数的函数评价次数, 同时鉴于比较的合理性和公平性, 确保每种算法针对不同的测试函数在每一迭代次数中拥有相同的函数评价次数. 对于本文算法 (简称GSDE), 设置种群规模为50, 其余算法种群规模为100. 所有实验独立运行30次. 不同于其他DE算法, 本文算法并没有特殊的控制参数, 而其他算法的控制参数按照对应文献进行设置, 这里不再赘述.

表 1　12个测试函数的函数评价次数

函数	评价次数	函数	评价次数
f_{01}	50 000	f_{07}	150 000
f_{02}	100 000	f_{08}	4 000
f_{03}	100 000	f_{09}	8 000
f_{04}	400 000	f_{10}	5 000
f_{05}	100 000	f_{11}	150 000
f_{06}	150 000	f_{12}	10 000

表2为各测试函数的基本参数, U 表示单模态函数, M 表示多模态函数. 目前, 对于单模态函数已经存在许多其他数学方法进行求解, 因此它主要用来测试算法的收敛速度和收敛精度; 而多模态函数具有较多的局部最优解, 因此常用来评价算法跳出局部最优的能力.

表3和表4分别给出了8种算法对于单模态测试函数和多模态测试函数独立运行30次所得的结果, Mean为平均值, Std为标准偏差, NA表示该算法不适用某一测试函数, 其中最优结果通过加粗标出. 同时, 本文根据各个算法优化得出的平均值按照从小到大的顺序进行排序 (Rank), 并在表中最后两行计算出每个算法的平均顺序, 同时对这一结果按照上述同样的方法进行排序, 以直观地比较不同算法的优化性能.

从表3可以看出, GSDE算法在对6个单模态函数进行优化求解时, 在给定的函数评价次数内, 30次独立运行所求得的最优结果的平均值及标准差, 除了函数 f_{05} 排名第3外, 其余函数的优化结果均显著优于其他7种算法, 特别是对于函数 $f_{01} \sim f_{03}$, GSDE算法在限定的函数评价次数内可以较快地寻找到全局最优解, 且明显优于其他7种算法. 另外, 从表3中最后一行对各算法针对6个单模态函数排序的平均值进行排序的结果可以看出, 整体上, GSDE对上述6种单模态函数的优化性能排名第一.

表 2 测试函数

基准测试函数	类别	维数	搜索范围	最优值
$f_{01}(x)=\sum_{i=1}^{D}x_i^2$	U	30	$[-100,100]$	0
$f_{02}(x)=\sum_{i=1}^{D}\lvert x_i\rvert+\prod_{i=1-1}^{D}\lvert x_i\rvert$	U	30	$[-10,10]$	0
$f_{03}(x)=\max_i\{\lvert x_i\rvert,1\leqslant i\leqslant D\}$	U	30	$[-100,100]$	0
$f_{04}(x)=\sum_{i=1}^{D}[100(x_{i+1}-x_i^2)^2+(x_i-1)^2]$	U	30	$[-30,30]$	0
$f_{05}(x)=\sum_{i=1}^{D}(\lfloor x_i+0.5\rfloor)^2$	U	30	$[-100,100]$	0
$f_{06}(x)=\sum_{i=1}^{D}ix_i^4+\text{random}[0,1)$	U	30	$[-1.28,1.28]$	0
$f_{07}(x)=-\sum_{i=1}^{D}x_i\sin\sqrt{\lvert x_i\rvert}$	M	30	$[-500,500]$	−1.256 9e-04
$f_{08}(x)=\sum_{i=1}^{D}(x_i^2-10\cos(2\pi x_i)+10)$	M	30	$[-5.12,5.12]$	0
$f_{09}(x)=20+e-20\exp\Big(-0.2\sqrt{\frac{1}{D}\sum_{i=1}^{D}x_i^2}\Big)-\exp\Big(\frac{1}{D}\sum_{i=1}^{D}\cos(2\pi x_i)_i^2\Big)$	M	30	$[-32,32]$	8.881 8e-16
$f_{10}(x)=\frac{1}{4\,000}\sum_{i=1}^{D}x_i^2-\prod_{i=1}^{D}\cos(\frac{x_i}{\sqrt{i}})+1$	M	30	$[-600,600]$	0
$f_{11}(x)=\frac{\pi}{D}\Big\{10\sin^2(\pi y_i)+\sum_{i-1}^{D-1}(y_i-1)^2[1+10\sin^2(\pi y_{i+1})]+(y_D-1)\Big\}+\sum_{i-1}^{D}u(x_i,10,100,4)$，$y_i=1+\frac{x_i+1}{4}$，$u(x_i,a,k,m)=\begin{cases}k(x_i-a)^m,\ x_i>a\\0,\ -a<x_i<a\\k(-x_i-a)^m,\ x_i<a\end{cases}$	M	30	$[-50,50]$	0
$f_{12}(x)=[1+(x_1+x_2+1)^2(19-14x_1+3x_1^2-14x_2+6x_1x_2+3x_2^2)]\times[30+(2x_1-3x_2)^2(18-32x_1+12x_1^2+48x_2-36x_1x_2+27x_2^2)]$	M	2	$[-5,5]$	3

表 3 单模态函数的优化结果

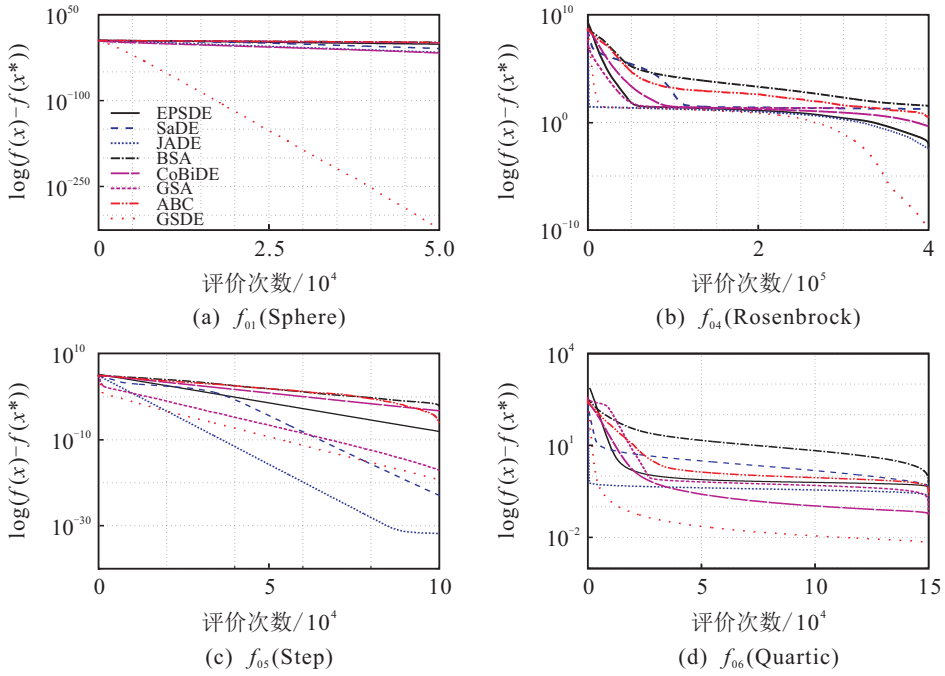
函数	指标	EPSDE	SaDE	JADE	BSA	CoBiDE	ABC	GSA	GSDE
f_{01}	Mean	1.206 7e-02	4.021 9e-10	1.113 4e-16	5.743 9e+00	3.771 0e+00	1.485 7e-02	2.464 5e-18	**0**
	Std	3.916 4e-03	4.480 9e-10	3.919 2e-17	6.764 8e+00	1.257 5e+00	2.364 1e-02	6.230 5e-19	**0**
	Rank	5	4	3	7	8	6	2	**1**
f_{02}	Mean	1.823 9e-04	7.331 6e-14	1.542 0e-18	1.292 3e-02	3.408 9e-02	2.454 5e-05	7.074 0e-09	**0**
	Std	5.451 1e-05	4.340 9e-14	4.234 6e-19	7.590 3e-03	9.207 0e-03	7.794 3e-06	7.146 9e-10	**0**
	Rank	6	3	2	7	8	5	4	**1**
f_{03}	Mean	2.659 1e-01	3.124 2e-02	4.697 9e-08	2.718 0e+00	4.745 1e+00	2.707 6e+01	6.571 1e-10	**0**
	Std	5.599 4e-02	3.867 6e-02	2.656 2e-08	5.790 2e-01	8.490 0e-01	3.265 4e+00	7.355 9e-11	**0**
	Rank	5	4	3	6	7	8	2	**1**
f_{04}	Mean	7.050 4e-04	1.773 9e+01	2.798 6e-03	2.940 8e+01	3.4793e-01	5.567 3e-02	2.264 0e+01	**6.901 0e-11**
	Std	3.754 1e-04	4.364 7e+00	5.653 8e-03	2.285 8e+01	2.428 4e-01	5.153 6e-02	1.1932e-01	**1.508 0e-10**
	Rank	2	6	3	8	5	4	7	**1**
f_{05}	Mean	2.824 4e-09	5.068 1e-24	**4.005 9e-33**	1.911 1e-03	2.655 7e-04	1.922 8e-08	1.798 8e-18	2.8259 e-20
	Std	1.270 0e-09	5.491 8e-24	**2.702 2e-33**	2.422 5e-03	9.994 2e-05	1.357 8e-08	5.123 6e-19	2.861 6e-20
	Rank	5	2	**1**	8	7	6	4	3
f_{06}	Mean	8.719 9e-03	3.606 3e-03	2.122 3e-03	9.817 6e-03	1.987 9e-02	7.777 0e-02	2.320 0e-03	**7.746 3e-04**
	Std	2.207 9e-03	1.064 5e-03	5.778 9e-04	2.957 7e-03	5.991 7e-03	1.558 5e-02	7.878 9e-04	**4.140 7e-04**
	Rank	5	4	2	6	7	8	3	**1**
平均顺序		4.666 7	3.833 3	2.333 3	7.000 0	7.000 0	6.166 7	3.666 7	**1.333 3**
整体排序		5	4	2	7	7	6	3	**1**

从表 4 可以看出, 除了函数 f_{11}, GSDE 算法在限定的函数评价次数内, 30 次独立运行的优化结果均优于其他 7 种算法, 且均寻找到全局最优值, 表现出了较强的跳出局部最优的能力. 同时, 从表 4 的最后一行也可以看出, GSDE 对于 6 种多模态函数优化结果平均排名第一, 也验证了本文算法具有较强的局部开发能力.

为了验证 GSDE 算法在收敛速度方面的优势, 图 1 给出了 8 种算法基于函数评价次数 (FEs) 的收敛特性对比.

表 4　多模态函数的优化结果

函数	指标	EPSDE	SaDE	JADE	BSA	CoBiDE	ABC	GSA	GSDE
f_{07}	Mean	NA	−1.256 9e+04	−1.183 1e+04	−1.208 8e+04	NA	−1.220 0e+04	−3.169 5e+03	**−1.256 9e+04**
	Std	NA	4.338 8e-12	1.005 5e+03	1.647 3e+02	NA	7.771 1e+01	3.420 8e+02	**1.850 1e-12**
	Rank	NA	2	5	4	NA	3	6	**1**
f_{08}	Mean	3.028 9e+02	2.235 2e+02	2.036 0e+02	2.877 8e+02	3.213 1e+02	2.408 1e+02	1.801 0e+02	**0**
	Std	2.089 2e+01	1.250 4e+01	1.185 7e+01	2.781 0e+01	1.745 1e+01	1.616 4e+01	1.975 7e+01	**0**
	Rank	7	4	3	6	8	5	2	**1**
f_{09}	Mean	1.348 1e+01	4.904 9e+00	2.845 6e+00	1.626 6e+01	1.701 5e+01	1.615 2e+01	4.852 3e+00	**8.881 8e-16**
	Std	6.978 7e-01	4.579 3e-01	1.606 0e-01	1.802 9e+00	6.993 8e-01	5.820 7e-01	4.809 6e-01	**0**
	Rank	5	4	2	7	8	6	3	**1**
f_{10}	Mean	1.136 9e+02	1.448 2e+01	4.491 7e+00	1.612 9e+02	1.923 6e+02	2.293 0e+02	4.605 4e+02	**0**
	Std	1.555 4e+01	2.790 3e+00	5.749 4e-01	6.639 9e+01	3.316 2e+01	3.307 5e+01	4.637 7e+01	**0**
	Rank	4	3	2	5	6	7	8	**1**
f_{11}	Mean	3.105 5e-16	**1.570 5e-32**	**1.570 5e-32**	5.360 6e-09	4.596 9e-08	1.212 6e-15	1.536 3e-03	7.663 3e-32
	Std	3.182 1e-16	**5.567 4e-48**	**5.567 4e-48**	7.708 7e-09	3.746 9e-08	2.991 3e-16	8.414 7e-03	7.401 8e-32
	Rank	4	**1**	**1**	6	7	5	8	3
f_{12}	Mean	3.000 0e+00	3.000 0e+00	3.000 0e+00	3.000 7e+00	3.000 0e+00	3.000 0e+00	3.000 0e+00	**3.000 0e+00**
	Std	1.680 7e-14	8.909 7e-15	5.857 4e-15	1.202 1e-03	6.501 7e-14	4.883 1e-05	4.062 6e-15	**2.516 2e-15**
	Rank	5	4	3	8	6	7	2	**1**
平均顺序		5.000 0	3.000 0	2.666 7	6.000 0	7.000 0	5.500 0	4.833 3	**1.500 0**
整体排序		4	3	2	7	8	6	4	**1**



(a) f_{01}(Sphere)　(b) f_{04}(Rosenbrock)

(c) f_{05}(Step)　(d) f_{06}(Quartic)

图 1　4 个单模态函数的平均收敛曲线

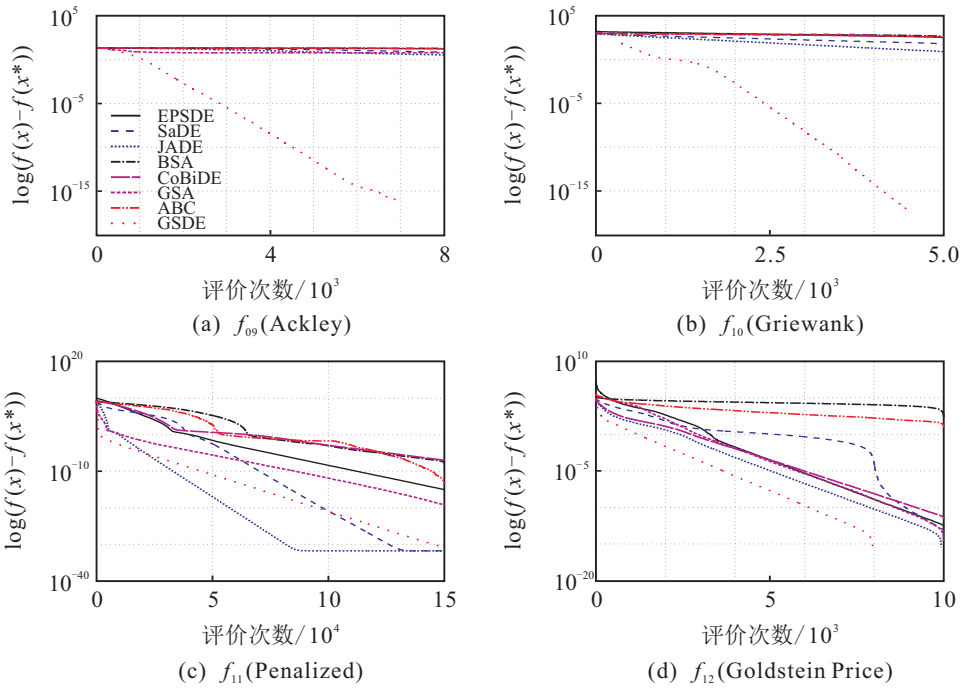
从图 1 可以看出, 由于 GSDE 算法采用了具有较强寻优能力的最优高斯随机游走策略, 有效地加快了算法的收敛速度, 对于除了 f_{05} 以外的其余 5 个单模态函数, 相比于其他 7 种算法, 均表现出了优越的收敛性能. 对于 f_{05} 函数, JADE 算法收敛速度较快.

为了直观分析算法对于多模态测试函数的收敛速度, 图 2 给出了 8 种算法基于 FEs 的收敛特性对比.

从图 2 可以看出: 对于 f_{09} 和 f_{10}, 在基于最优高斯随机游走策略的基础上, 结合个体筛选策略, GSDE 算法能够快速地跳出局部最优, 并收敛至全局最优解; 对于 f_{12}, 虽然 8 种算法均搜索到全局最优值, 但是 GSDE 算法以较快的收敛速度收敛至全局最优解; 对于 f_{11}, JADE 算法表现出了较好的收敛速度.

为了进一步对实验结果进行分析, 本文采用文献 [15] 用到的 Friedman 检验, 其中显著性水平 $\alpha = 0.05$. 该方法能够有效地检验出 GSDE 算法与其他多个算法之间的显著性差异. 表 5 给出了 Friedman 检验结果.

从表 5 可以看出, GSDE 算法与 SaDE、BSA、CoBiDE、GSA、EPSDE 和 ABC 算法在收敛速度和稳定性方面具有明显的差异. 虽然与 JADE 算法相比没有明显差异, 但从表 5 中的秩的排序可以看出, GSDE 算法分别以 1.75 和 1.934 8 在 Mean 和 SD 方面排名第一, 因此可以得出[15], GSDE 算法优于 JADE 算法及其他相比较的算法.



(a) f_{09}(Ackley) (b) f_{10}(Griewank)

(c) f_{11}(Penalized) (d) f_{12}(Goldstein Price)

图 2 4 个多模态函数的平均收敛曲线

表 5 Friedman 检验结果

算法	P 值	秩 (Mean)	P 值	秩 (Std)
EPSDE	0.002 353	4.791 7	0.001 335	4.833 3
SaDE	0.040 793	3.625	0.080 118	3.375
JADE	0.260 589	2.875	0.243 345	2.791 7
BSA	0.000 006	6.291 7	0	7.083 3
CoBiDE	0.000 001	6.708 3	0	6.666 7
GSA	0.005 96	5.458 3	0.007 661	5.333 3
ABC	0.000 209	4.5	0.000 209	4.291 7
GSDE	1	1.75	1	1.934 8

2.2 约束优化问题

2.2.1 压力容器设计问题

压力容器设计优化问题是一个复杂类型的优化问题, 其目的是最小化总成本, 包括成型代价、材料代价和焊接代价. 图 3 为圆柱形压力容器和半球形封头的基本参数. 其中: $T_s(x_1)$ 为半球形封头厚度, $T_h(x_2)$ 为圆柱形压力容器厚度, $R(x_3)$ 为内径, $L(x_4)$ 为圆柱段长度.

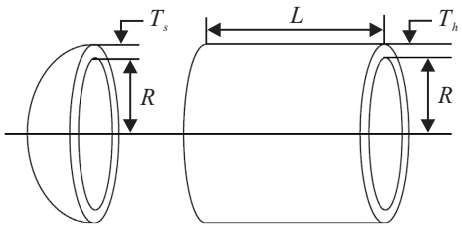


图 3 压力容器设计问题

这一问题可以概括为如下目标函数:

$$\min f(x_1,x_2,x_3,x_4)=$$
$$0.622\,4x_1x_3x_4+1.778\,1x_2x_3^2+$$
$$3.166\,1x_1^2x_4+19.84x_1^2x_3.$$

约束条件为

$$\begin{cases} g_1(X)=-x_1+0.019\,3x_3\leqslant 0,\\ g_2(X)=-x_2+0.009\,54x_3\leqslant 0,\\ g_3(X)=-\pi x_3^2x_4-\dfrac{4}{3}\pi x_3^2+1\,296\,000\leqslant 0,\\ g_4(X)=x_4-240\leqslant 0. \end{cases}$$

其中: $1\times 0.062\,5\leqslant x_1、x_2\leqslant 99\times 0.062\,5$; $10\leqslant x_3、x_4\leqslant 200$.

国内外学者将这一优化问题作为结构优化问题的一个测试函数, 并利用不同的算法来验证其求解实际优化问题的能力. 如文献 [12] 提出的灰狼优化算法 (GWO), 文献 [16] 提出的基于人工免疫系统的混合遗传算法 (GA-AIS), 文献 [17] 提出的广泛学习粒子群算法 (CLPSO), 文献 [18] 设计的协同差分进化算法 (CEDE), 以及文献 [19] 提出的蝙蝠算法 (BA) 等.

本文针对这一优化问题, 采用文献 [19] 的约束条件处理方法对其进行优化计算, 表 6 和表 7 给出了 GSDE 在种群规模为 50 的条件下独立运行 30 次的最优结果, 并与上述文献中 4 种算法得出的优化结果进行比较, 其中 NA 表示原文中没有相应的参数.

从表 6 可以看出, GSDE 在 90 000 次函数评价次数的条件下优化出的总代价明显小于其他 5 种算法. 从表 7 可以看出, 在利用 GSDE 算法得出的优化结果中, g_3 约束函数达到 0 值, g_1 和 g_2 约束函数值分别达到 10^{-12} 数量级. 相比于其他算法, 3 个约束函数的函数值更接近 0, 这也说明了该算法求解这一问题的有效性.

表 6　不同算法对于压力容器设计问题的最优结果对比

算法	$T_s(x_1)$	$T_h(x_2)$	$R(x_3)$	$L(x_4)$	总代价	评价次数
BA	0.812 5	0.437 5	42.098 445 6	176.636 595 8	6 059.714 335	375 000
GA-AIS	0.812 5	0.437 5	42.094 967	176.679 72	6 060.138	150 000
CEDE	0.812 5	0.437 5	42.098 4	176.746 5	6 059.734 0	27 500
CLPSO	0.812 5	0.437 5	42.098 4	176.636 6	6 059.714 3	60 000
GWO	0.812 500	0.434 500	42.089 181	176.758 731	6 051.563 9	NA
GSDE	0.778 168 64	0.384 649 16	40.319 618 72	199.999 999 99	**5 885.332 774**	90 000

表 7　压力容器设计问题的约束函数值对比

算法	g_1	g_2	g_3	g_4
BA	NA	NA	NA	NA
GA-AIS	0.000 007	0.035 914	0.062 5	−63.320 282
CEDE	−6.67e-07	−3.58e-02	−3.705 123	−63.362 3
CLPSO	−8.800 0e-07	−0.035 9	3.122 7	−63.363 4
GWO	NA	NA	NA	NA
GSDE	−3.624 323e-12	−1.434 519e-12	**0**	−40.000 000

2.2.2　张力/压缩弹簧设计问题

张力/压缩弹簧设计问题是一个著名的用来验证算法优越性的工程优化设计问题. 它的主要目标是在满足挠度、剪应力和振动频率等约束条件下使其质量最小化. 这一问题主要包括 4 个约束条件和 3 个变量: 弹簧线圈直径 $w(x_1)$, 弹簧的平均直径 $d(x_2)$ 和弹簧的有效线圈数 $L(x_3)$, 如图 4 所示.

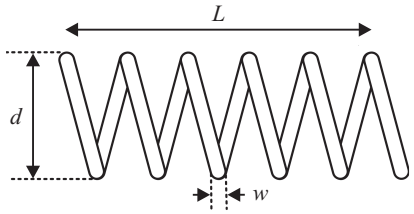


图 4　张力/压缩弹簧设计问题

目标函数和约束条件如下所示:

$$\min\ f(x_1,x_2,x_3)=(x_3+2)x_1{}^2x_2;$$

$$\text{s.t. } g_1(X)=1-\frac{x_2^3x_3}{71\,785x_1^4}\leqslant 0,$$

$$g_2(X)=\frac{x_2(4x_2-x_1)}{12\,566x_1^3(x_2-x_1)}+\frac{1}{5\,108x_1^2}-1\leqslant 0,$$

$$g_3(X)=1-\frac{140.45x_1}{x_2^2x_3}\leqslant 0,$$

$$g_4(X)=\frac{2(x_1+x_2)}{3}-1\leqslant 0.$$

设计变量的取值范围为: $0.05\leqslant x_1\leqslant 2$, $0.25\leqslant x_2\leqslant 1.3$, $2.0\leqslant x_3\leqslant 15.0$.

同样, 国内外学者提出了不同的方法来优化这一问题, 如文献 [20] 提出的定向搜索优化算法 (DSO), 文献 [21] 提出的细菌遗传重组算法 (BGRA), 以及上述 GA-AIS、CEDE 和 GWO 算法等.

表 8 和表 9 给出了 GSDE 在种群规模为 50 的条件下独立运行 30 次的最优结果, 并与上述文献中 4 种算法得出的优化结果进行比较.

表 8　不同算法对于弹簧设计问题的最优结果对比

算法	$w(x_1)$	$d(x_2)$	$L(x_3)$	总代价	评价次数
BGRA	0.051 674 711	0.356 372 600	11.309 229 42	0.012 665 237 3	200 000
GA-AIS	0.051 660 806	0.356 032 34	11.329 555	0.012 666 6	36 000
CEDE	0.051 609	0.354 714	11.410 831	0.012 670 2	204 800
DSO	0.051 711 791	0.357 264 808	11.256 964 83	0.012 665	NA
GWO	0.051 7	0.356 7	11.289	0.012 67	NA
GSDE	0.051 689 083	0.356 718 262	11.288 935 111	**0.012 665 232 7**	90 000

表 9　弹簧设计问题的约束函数值对比

算法	g_1	g_2	g_3	g_4
BGRA	−1.170 95e-09	−2.863 08e-08	−4.053 1	−0.727 968
GA-AIS	−0.000 006 437	−0.000 013 709	−4.052 324 300	−0.728 204 600
CEDE	−3.90e-05	−1.83e-04	−4.048 627	−0.729 118
DSO	−5.156 3e-09	−2.808 7e-10	−4.054 9	−0.727 3
GWO	NA	NA	NA	NA
GSDE	−1.150 191e-13	−1.143 529e-14	−4.053 786 662 5	−0.727 728 436 5

从表 8 和表 9 可以看出, 相比于其他 5 种算法, 本文算法展现出了较优的优化性能. 例如, 对于表现较好的 BGRA 算法, 在明显较小评价次数的条件下, GSDE 计算出的总代价小于 BGRA, 说明了 GSDE 算法在确保收敛速度的前提下具有较高的收敛精度. 同时, 相比于其他算法, 计算出的 g_1 和 g_2 约束函数值至少提高了 4 个数量级.

3　结　论

本文在标准差分进化算法的基础上, 提出了一种基于最优高斯随机游走策略和个体筛选策略的差分进化算法. 在 DE 算法中, 基于最优引导提出了最优高斯随机游走策略, 在加强探索能力的同时, 提高了算法的局部开发能力; 利用个体筛选策略, 对较差的个体进行进化操作, 以提高其跳出局部最优的能力; 为

了进一步加强算法的局部开发能力, 对DE算法中的交叉变异操作进行了改进, 提出了基于个体优化性能的变异策略, 同时仅对性能较差的个体中的元素进行变异操作. 通过对12个典型测试函数的实验, 验证了本文所提算法在求解无约束优化问题时, 相比于SaDE、JADE、BSA、CoBiDE、GSA、ABC以及EPSDE算法, 整体上具有明显较快的收敛速度和收敛精度, 同时表现出了较好的稳定性. 通过对2个带约束的工程设计优化问题, 验证了本文所提算法在求解约束优化问题上的优越性. 但算法的结构有待进一步简化, 以提高其求解优化问题的效率, 同时, 将该算法应用于组合优化问题及其扩展问题上, 如无人机任务分配问题和航迹规划问题等, 也是下一步需要解决的问题.

参考文献(References)

[1] Price K, Storn R, Lampinen J. Differential evolution: A practical approach to global optimization[M]. Berlin: Springer-Verlag, 2005: 1-24.

[2] 李牧东, 赵辉, 翁兴伟. 具有广泛学习策略的回溯搜索优化算法[J]. 系统工程与电子技术, 2015, 37(4): 958-963.
(Li M D, Zhao H, Weng X W. Backtracking search optimization algorithm with comprehensive learning strategy[J]. Systems Engineering and Electronics, 2015, 37(4): 958-963.)

[3] Yang M, Li C H, Cai Z H, et al. Differential evolution with auto-enhanced population diversity[J]. IEEE Trans on Cybernetics, 2015, 45(2): 302-315.

[4] 周晓根, 张贵军, 梅珊, 等. 基于抽象凸估计选择策略的差分进化算法[J]. 控制理论与应用, 2015, 32(3): 389-397.
(Zhou X G, Zhang G J, Mei S, et al. Differential evolution algorithm based on abstract convex underestimate selection strategy[J]. Control Theory & Applications, 2015, 32(3): 389-397.)

[5] Fan H Y, Lampinen J. A trigonometric mutation operation to differential evolution[J]. J of Global Optimization, 2003, 27(1): 105-129.

[6] Mallipeddia R, Suganthana P N, Panb Q K, et al. Differential evolution algorithm with ensemble of parameters and mutation strategies[J]. Applied Soft Computing, 2011, 11(2): 1679-1696.

[7] Civicioglu P. Backtracking search optimization algorithm for numerical optimization problems[J]. Applied Mathematics and Computation, 2013, 219(15): 8121-8144.

[8] Qin A, Suganthan P. Self-adaptive differential evolution algorithm for numerical optimization[C]. IEEE Congress of Evolution on Computation(CEC 2005). Edinburgh: IEEE, 2005: 1785-1791.

[9] Zhang J, Arthur C. JADE: Adaptive differential evolution with optional external archive[J]. IEEE Trans on Evolution Computation, 2009, 13(5): 945-958.

[10] Wang Y, Li H X, Huang T W, et al. Differential evolution based on covariance matrix learning and bimodal distribution parameter setting[J]. Applied Soft Computing, 2014, 18: 232-247.

[11] Kaplan D T, Peacock L G. Coarse-grained embeddings of time series: Random walks, Gaussian random processes, and deterministic chaos[J]. Physica D, 1993, 64(4): 431-454.

[12] Mirjalili S, Mirjalili S M A. Grey wolf optimizer[J]. Advances in Engineering Software, 2014, 69: 46-61.

[13] Mernik M, Liu S H, Karaboga M D, et al. On clarifying misconceptions when comparing variants of the Artificial Bee Colony Algorithm by offering a new implementation[J]. Information Sciences, 2015, 291(10): 115-127.

[14] Rashedi E, Nezamabadi-pour H, Saryazdi S. GSA: A gravitational search algorithm[J]. Information Sciences, 2009, 179(13): 2232-2248.

[15] Derrac J, Garcia S, Molina D, et al. A practical tutorial on the use of nonparametric statistical tests as a methodology for comparing evolutionary and swarm intelligence algorithms[J]. Swarm Evolution Computation, 2011, 1(1): 3-18.

[16] Bernardino H, Barbosa I, Lemonge A. A hybrid genetic algorithm for constrained optimization problems in mechanical engineering[C]. IEEE Congress on Evolutionary Computation(CEC 2007). Singapore: IEEE, 2007: 646-653.

[17] Gao L, Hai L A. Comprehensive learning particle swarm optimizer for constrained mixed-variable optimization problem[J]. Int J of Intelligent Systems, 2010, 3(6): 832-842.

[18] Huang F Z, Wang L, He Q. An effective co-evolutionary differential evolution for constrained optimization [J]. Applied Mathematics and Computation, 2007, 186(1): 340-356.

[19] Gandomi A, Yang X S, Alavi A, et al. Bat algorithm for constrained optimization tasks[J]. Neural Computing & Applications, 2013, 22(6): 1239-1255.

[20] Zou D, Liu H, Gao L, et al. Directed searching optimization algorithm for constrained optimization problems[J]. Expert Systems and Applications, 2011, 38(7): 8716-8723.

[21] Hsieh T J. A bacterial gene recombination algorithm for solving constrained optimization problems[J]. Applied Mathematics and Computation, 2014, 231(15): 187-204.

(责任编辑：齐 霁)