计算机科学 COMPUTER SCIENCE
http://www.jsjkx.com
DOI:10.11896/jsjkx.181202356

基于群体分布的自适应差分进化算法

李章维　　王柳静
浙江工业大学信息工程学院　杭州 310023

摘　要　差分进化算法是一种简单有效的启发式全局优化算法,但是其优化性能受差分进化策略及控制参数取值的影响较大,不合适的策略和参数容易导致算法早熟收敛。因此,针对差分进化算法搜索过程中变异策略和控制参数的选择问题,文中提出了一种基于群体分布的自适应差分进化算法(Population Distribution-based Self-adaptive Differential Evolution,PDSDE)。首先,设计适应因子以衡量当前种群的分布情况,进而实现算法所处进化阶段的自适应判断;然后,根据不同进化阶段的特点,设计阶段特定的变异策略和控制参数,并设计自适应机制以实现算法策略和参数的动态调整,从而平衡算法的全局探测和局部搜索能力,以达到提高算法搜索效率的目的;最后,将所提算法与6种主流改进算法进行比较。15个典型测试函数的数值实验表明,所提算法在平均函数评价次数、求解精度、收敛速度等指标的评价优于文中给出的6种主流改进算法,因此可以证明所提算法的计算代价、优化性能和收敛性能更具优势。

关键词:差分进化;群体分布;全局优化;阶段划分;自适应
中图法分类号　TP301.6

Population Distribution-based Self-adaptive Differential Evolution Algorithm

LI Zhang-wei and WANG Liu-jing
College of Information Engineering,Zhejiang University of Technology,Hangzhou 310023,China

Abstract　Differential evolution is a simple and powerful heuristic global optimization algorithm. However,its performance is strongly influenced by the differential evolution strategies and the value of control parameters. Inappropriate strategies and parameters may lead the algorithm fall into premature convergence. Aiming at the problem about selection of strategies and parameters in search process of differential evolution,a population distribution-based self-adaptive differential evolution algorithm was proposed. Firstly,the adaptive factor is established for measuring the distribution of the current population,and the evolution stage of the algorithm can be further determined adaptively. Then,according to the characteristics of different evolution stages,the stage-specific mutation strategies and control parameters are designed,the self-adaptive mechanism is also designed in order to realize dynamic adjustment of strategies and parameters,to balance the global detection and local search capabilityof the algorithm,and improve the search efficiency of the algorithm. Finally,the proposed algorithm is compared with six main-stream differential evolution variants. The numerical experiments of fifteen typical test functions show that the proposed algorithm is superior to six main-stream differential evolution variants in terms of the measures of the average function evaluation times,solution accuracy and converge velocity. Therefore,the computational cost,optimization performance and convergence performance of the proposed algorithm can be proved to be more advantageous.

Keywords　Differential evolution,Population distribution,Global optimization,Stage division,Self-adaptive

1　引言

差分进化算法(Differential Evolution,DE)是一种基于群体智能的全局优化算法,模拟生物进化机制,通过反复迭代保留适应环境的优良个体,从而达到种群寻优的目的[1]。DE算法在优化求解过程中不依赖于问题的特征信息,而是采用个体之间的差分信息指导种群对最优解的搜索,使得算法能够根据当前的搜索情况来动态调整搜索策略。由于其全局收敛能力和鲁棒性较好[2],具有原理简单、控制参数少、易于实现的优点,DE算法更适于求解一些在工程技术和科学研究等诸多领域中利用常规的数学规划方法所无法求解的复杂优化问题[3],诸如经济学[4]、生产管理[5]、电力系统[6]、机器人[7]、化工[8]、生物信息[9]等。

DE算法从随机初始种群开始,通过变异(Mutation)、交

到稿日期:2018-12-19　返修日期:2019-06-21　本文已加入开放科学计划(OSID),请扫描上方二维码获取补充信息。
基金项目:国家自然科学基金(61573317)
This work was supported by the National Natural Science Foundation of China (61573317).
通信作者:李章维(lizhangwei18@163.com)

叉(Crossover)、选择(Selection)操作生成新一代的种群，反复迭代计算，引导搜索过程向全局最优解逼近[10]。DE 算法的特点是全局探测能力强，能够快速定位局部最优解所在的区域，但是进入局部搜索阶段后，由于多样性的减小会出现收敛速度变慢、搜索效率下降等问题。影响 DE 算法性能的主要因素是策略和控制参数的合理选择[11]，因此，平衡 DE 算法的全局探测和局部搜索的关键在于如何在搜索过程中实现策略和控制参数的自适应调整。

国内外学者在策略和参数的适应性方面进行了研究，以提高算法的性能。Qin 等[12]提出的自适应差分进化算法(SaDE)，设置了包含 4 个变异策略的策略池，通过学习机制借鉴产生优质解的历史记录构建概率模型，来进行变异策略和参数的自适应调整，有效改进了 DE 算法的优化性能。Mallipeddi 等[13]提出的变异策略及参数的差分进化算法(EPSDE)，设置了包含 3 个变异策略的策略池，能够保留产生优质实验个体的变异策略和参数，对产生较差实验个体的策略和参数以相同的概率保留或丢弃。Zhang 等[14]提出了具有外部存档的自适应差分进化算法，基于成功的历史经验设计了变异策略 DE/current-to-pbest 以及参数自适应机制，指导种群收敛方向，提高了算法效率。Wang 等[15]提出了变异策略和参数混合的差分进化算法(CoDE)，通过竞争方式完成了对不同策略与参数随机组合的选择，以确定每一代种群采用的变异策略。以上改进实际上通过设计概率模型、组合择优、采用历史经验等方法来实现对策略和参数的选择，但还存在着需要先验知识、竞争盲目等问题，因此选择更为合适的策略和控制参数仍是改进 DE 算法的重要研究内容。

为了提高 DE 算法的搜索效率，本文提出了基于群体分布的自适应差分进化算法。首先，考虑到搜索过程中各个体间距离的变化，设计了适应因子来衡量当前种群的分布情况，进而自适应判断算法所处阶段；其次，针对不同进化阶段的特点，设计了一种策略和参数的自适应机制，使得算法处于不同进化阶段时能够采用阶段特定的变异策略和控制参数，来实现变异策略和控制参数的动态调整，以平衡算法的全局探测和局部搜索能力，提高算法的优化性能；最后，通过与 6 种 DE 改进算法的对比实验，验证了所提 PDSDE 算法在计算代价、可靠性、解的质量及收敛速度方面的优越性。

2 PDSDE 算法

2.1 建立适应因子

DE 算法的搜索过程分为全局探测和局部搜索两部分，从随机初始种群开始，通过变异、交叉、选择操作的反复迭代，引导种群向全局最优解逐渐收敛。因此，可以通过搜索过程中种群的分布情况来估计当前所处的进化阶段，进而分配合适的变异策略和控制参数。

由于种群会随着搜索过程的进行最终收敛至全局最优解附近，可以推断出种群初始时最为分散，而算法终止时其收敛程度较高，因此种群分布情况可采用种群个体之间的欧氏距离进行衡量，如式(1)所示：

$$d^g=\frac{1}{NP}\sum_{i=1}^{NP}\sum_{k=i+1}^{NP}\sqrt{\sum_{j=1}^{N}(x_{i,j}^g-x_{k,j}^g)^2} \tag{1}$$

其中，NP 是种群个体数，N 为种群个体的维数，d^g 为第 g 代种群的分散程度，$x_{i,j}^g$ 和 $x_{k,j}^g$ 分别是第 g 代种群中第 i 个、第 k 个个体的第 j 维。

为了消除不同目标函数或同一目标不同维数之间的差异，需要建立统一的指标作为进化阶段的判定依据，对式(1)进行归一化处理得到该指标，即式(2)适应因子 AF：

$$AF=\frac{d^g-d_{\min}}{d_{\max}-d_{\min}} \tag{2}$$

其中，$d_{\max}$为当前种群的最大分散程度，表明各个体之间的相互距离最大，分析算法搜索过程的特点，将初代种群距离赋给 $d_{\max}$，并且随着搜索的进行不断更新 $d_{\max}$；而 $d_{\min}$ 为当前种群的最小分散程度，当种群收敛至全局最优解时记录为 $d_{\min}$，理论上 $d_{\min}=0$。

2.2 阶段划分

适应因子包含种群分布信息，通过建立基于该指标的判定模型，来实现搜索过程中进化阶段的自适应判定，最终合理选择变异策略和控制参数，以提高算法效率。

基于适应因子 AF，文中设计了一种策略和参数自适应机制，首先通过式(3)实现对当前种群所处进化阶段的判定，从而根据不同进化阶段的特点对控制参数进行动态调整。

$$\Phi=\begin{cases}stage1, & rand(0,1)<AF\\ stage2, & \text{otherwise}\end{cases} \tag{3}$$

其中，Φ 为当前种群所处的进化阶段，stage1 为全局探测阶段，stage2 为局部搜索阶段。而基于式(3)划分进化阶段，出于以下考虑。

1)适应因子 AF 反映了种群分布情况，随着搜索过程的进行，不同阶段可以共存于同一代中，因此更新每一代中每个个体的阶段判定是合理的。

2)当处于全局探测阶段时，种群分散于解空间进行大范围的探测，即种群个体之间的距离较大，因此适应因子 AF 的值偏大；当处于局部搜索阶段时，种群聚集于某些局部最优解附近，导致适应因子 AF 的值较小。因此，适应因子 AF 能够描述种群当前的分布情况。

3)由于 DE 算法的搜索特性，种群在早期分散于解空间进行大范围的探测，随着搜索过程的进行，种群逐渐收敛至某些局部最优解附近进行局部搜索，而全局探测和局部搜索阶段在整个搜索过程中可能会反复出现，因此以随机数方式判定进化阶段更为合理。

2.3 策略和参数的自适应机制

处于 stage1 时，当前种群比上一代种群更加分散，种群个体应分布于搜索空间进行大范围寻优，因此变异策略采用全局探测能力强的 DE/rand/1 策略，其中，差分向量的变异因子 F 值应适当增大以引导种群跳出局部极值区，并减小交叉概率 CR 以增加种群多样性，向更有可能存在优质解的区域进行搜索，具体操作如式(4)、式(5)所示：

$$\boldsymbol{v}_i^g=\boldsymbol{x}_{r_1}^g+F\cdot(\boldsymbol{x}_{r_2}^g-x_{r_3}^g) \tag{4}$$

$$\begin{cases}F=F+rand(0,1)\cdot\Delta\\ CR=CR\text{-}rand(0,1)\cdot\Delta\end{cases} \tag{5}$$

其中，$\boldsymbol{v}_i^g$ 为 $\boldsymbol{x}_i^g$ 的变异个体，$\boldsymbol{x}_{r_1}^g$，$\boldsymbol{x}_{r_2}^g$，$\boldsymbol{x}_{r_3}^g$ 为从第 g 代种群中随机挑选的第 r_1，r_2，r_3 个体，且 $r_1 \neq r_2 \neq r_3 \neq i$，$rand(0,1)$ 为 0 至 1 的随机数，将控制参数变异因子和交叉概率初始化为 $F=0.5$，$CR=0.5$，$\Delta=AF$。

而处于阶段 stage2 时，与上述情况相反，当前种群比上一代种群更加聚集，种群个体定位于最优解所在区域，逐渐收敛以进行局部搜索，因此变异策略采用局部搜索能力强的 DE/best/1 策略，其中，差分向量的变异因子 F 应适当减小以实现对该局部区域的充分搜索，并将交叉概率 CR 增大，具体操作如式(6)、式(7)所示：

$$\boldsymbol{v}_i^g = \boldsymbol{x}_{\text{best}}^g + F \cdot (\boldsymbol{x}_{r_1}^g - \boldsymbol{x}_{r_2}^g) \tag{6}$$

$$\begin{cases} F = F - rand(0,1) \cdot \Delta \\ CR = CR + rand(0,1) \cdot \Delta \end{cases} \tag{7}$$

其中，$\boldsymbol{x}_{\text{best}}^g$ 为从第 g 代种群中挑选的最优个体。

变异操作之后，对种群个体进行交叉、选择操作，其计算如式(7)、式(8)所示：

$$\boldsymbol{u}_{i,j}^g = \begin{cases} \boldsymbol{v}_{i,j}^g, & \text{if } rand(0,1) \leqslant CR \text{ or } j = j_{\text{rand}} \\ \boldsymbol{x}_{i,j}^g, & \text{otherwise} \end{cases} \tag{8}$$

其中，$\boldsymbol{u}_{i,j}^g$ 为经交叉操作生成的实验个体 $\boldsymbol{u}_i^g$ 的第 j 维，j_{rand} 为 0 到 N 之间的随机整数，N 为问题的维数。

$$\boldsymbol{x}_i^{g+1} = \begin{cases} \boldsymbol{u}_i^g, & f(\boldsymbol{u}_i^g) < f(\boldsymbol{x}_i^g) \\ \boldsymbol{x}_i^g, & \text{otherwise} \end{cases} \tag{9}$$

其中，$\boldsymbol{x}_i^{g+1}$ 表示第 $g+1$ 代种群的第 i 个个体，$f(\boldsymbol{u}_i^g)$ 和 $f(\boldsymbol{x}_i^g)$ 分别为 $\boldsymbol{u}_i^g$ 和 $\boldsymbol{x}_i^g$ 的适应值。

3 数值仿真

为了对本文所提算法的优化性能进行合理评估，数值仿真实验中将采用如下 6 种算法与本文所提算法进行对比：Tanabe 等[16]提出的 CEC2013 中性能最好的 DE 改进算法(SHADE)，该算法利用成功参数设置的历史经验指导子代控制参数的选择，实现控制参数的自适应调整；Zhou 等[17]提出的基于抽象凸估计策略的差分进化算法(DELU)；以及引言中提到的 EPSDE，SaDE，JADE 和 CODE。这 6 种主流改进 DE 算法将与本文所提算法 PDSDE 在计算代价、可靠性、优化性能和收敛性能方面进行比较。在典型标准测试函数方面，本文选取了 15 个问题[18]来保证实验结果的客观性，其中包含 7 个高维单模态问题 f_1-f_7 和 8 个高维多模态问题 f_8-f_{15}，表 1 列出了各测试函数的数学表达式及对应的参数。

表 1 标准测试函数

Table 1 Benchmark functions

函数名	数学表达式	搜索范围	全局最优值
Sphere	$f_1(x)=\sum_{i=1}^{N} x_i^2$	$[-100,100]$	0
SumSquares	$f_2(x)=\sum_{i=1}^{N} i x_i^2$	$[-10,10]$	0
Schwefel2.22	$f_3(x)=\sum_{i=1}^{n} \lvert x_i \rvert + \prod_{i=1}^{n} \lvert x_i \rvert$	$[-10,10]$	0
Exponential	$f_4(x)=-\exp(-0.5\sum_{i=1}^{N} x_i^2)$	$[-1,1]$	0
Table	$f_5(x)=10^6 x_1^2+\sum_{i=2}^{n} x_i^2$	$[-100,100]$	0
Step	$f_6(x)=\sum_{i=1}^{n} \lfloor x_i+0.5 \rfloor^2$	$[-100,100]$	0
Zakharov	$f_7(x)=\sum_{i=1}^{N} x_i^2+\sum_{i=1}^{N}(0.5ix_i)^2+\sum_{i=1}^{N}(0.5ix_i)^4$	$[-5,10]$	0
Griewank	$f_8(x)=1+\frac{1}{4000}\sum_{i=1}^{N} x_i^2-\prod_{i=1}^{N}\cos(\frac{x_i}{\sqrt{i}})$	$[-600,600]$	0
Levy and Montalvo 1	$f_9(x)=\pi(10\sin^2(\pi y_1)/n+\sum_{i=1}^{n-1}(y_i-1)^2(1+10\sin^2(\pi y_i+1)))+(y_n-1)^2$，$y_i=1+0.25(x_i+1)$	$[-10,10]$	0
Levy and Montalvo 2	$f_{10}(x)=0.1\{\sin^2(3\pi x_i)+\sum_{i=1}^{n-1}(x_i-1)^2(1+\sin^2(3\pi x_i+1))+(x_n-1)^2(1+\sin^2(2\pi x_n))\}$	$[-2,2]$	0
Ackley	$f_{11}(x)=-20\exp(-0.2\sqrt{\frac{1}{N}\sum_{I=1}^{N} x_i^2})-\exp(\frac{1}{N}\sum_{i=1}^{N}\cos(2\pi x_i))+20+e$	$[-30,30]$	0
Penalized 1	$f_{12}(x)=\frac{\pi}{N}\{10\sin^2(\pi y_i)+10\sin^2(\pi y_i)+\sum_{i=1}^{N-1}(y_i-1)^2[1+10\sin^2(\pi y_{i+1})]+(y_N-1)^2\}+\sum_{i=1}^{N} u(x_i,10,100,4)$	$[-50,50]$	0
Penalized 2	$f_{13}(x)=0.1\{\sin^2(3\pi x_1)+\sum_{i=1}^{N-1}(x_i-1)^2[1+\sin^2(3\pi x_{i+1})]+(x_N-1)^2[1+\sin^2(2\pi x_N)]\}+\sum_{i=1}^{N} u(x_i,5,100,4)$	$[-50,50]$	0
Neumaier3	$f_{14}(x)=\sum_{i=1}^{N}(x_i-1)^2-\sum_{i=2}^{N} x_i x_{i-1}+\frac{N(N+4)(N-1)}{6}$	$[-900,900]$	0
Alpine	$f_{15}(x)=\sum_{i=1}^{N} \lvert x_i \sin x_i+0.1x_i \rvert$	$[-10,10]$	0

表 2 函数评价次数和成功率

Table 2 Function evaluations and success rates

函数	维数	SHADE		DELU		EPSDE		SaDE		JADE		CoDE		PDSDE	
		FES	*SR*	*FES*	*SR*	*FES*	*SR*	*FES*	*SR*	*FES*	*SR*	*FES*	*SR*	*FES*	*SR*
f_1	30	2.35×10^4	1.00	1.22×10^4	1.00	1.67×10^4	1.00	2.03×10^4	1.00	2.36×10^4	1.00	4.02×10^4	1.00	$\mathbf{1.10\times10^4}$	1.00
f_2	30	2.16×10^4	1.00	1.07×10^4	1.00	1.51×10^4	1.00	1.84×10^4	1.00	2.16×10^4	1.00	3.63×10^4	1.00	$\mathbf{9.80\times10^3}$	1.00
f_3	30	3.42×10^4	1.00	3.15×10^4	1.00	2.32×10^4	1.00	2.44×10^4	1.00	3.58×10^4	1.00	5.64×10^4	1.00	$\mathbf{1.53\times10^4}$	1.00
f_4	30	1.67×10^4	1.00	7.98×10^3	1.00	9.36×10^3	1.00	1.10×10^4	1.00	1.34×10^4	1.00	2.24×10^4	1.00	$\mathbf{6.30\times10^3}$	1.00
f_5	30	2.67×10^4	1.00	2.55×10^4	1.00	1.83×10^4	1.00	2.11×10^4	1.00	2.70×10^4	1.00	4.13×10^4	1.00	$\mathbf{1.21\times10^4}$	1.00
f_6	30	1.19×10^4	1.00	1.16×10^4	1.00	8.33×10^3	1.00	1.07×10^4	1.00	1.18×10^4	1.00	2.01×10^4	1.00	$\mathbf{5.25\times10^3}$	1.00
f_7	30	5.98×10^4	1.00	7.71×10^4	1.00	1.33×10^5	1.00	1.37×10^5	1.00	5.61×10^4	0.97	7.80×10^4	1.00	$\mathbf{4.90\times10^4}$	1.00
f_8	30	$\mathbf{2.56\times10^4}$	1.00	$\mathbf{1.54\times10^4}$	1.00	$\mathbf{1.76\times10^4}$	0.83	$\mathbf{2.17\times10^4}$	0.73	$\mathbf{2.69\times10^4}$	0.97	$\mathbf{4.39\times10^4}$	1.00	1.38×10^4	1.00
f_9	30	1.67×10^4	1.00	1.05×10^4	1.00	1.30×10^4	1.00	1.23×10^4	1.00	1.73×10^4	1.00	2.63×10^4	1.00	$\mathbf{8.70\times10^3}$	1.00
f_{10}	30	1.71×10^4	1.00	8.71×10^3	1.00	1.15×10^4	1.00	1.29×10^4	1.00	1.74×10^4	1.00	2.70×10^4	1.00	$\mathbf{8.90\times10^3}$	1.00
f_{11}	30	3.21×10^4	1.00	2.65×10^4	1.00	2.32×10^4	1.00	3.05×10^4	0.93	3.31×10^4	1.00	5.68×10^4	1.00	1.58×10^4	1.00
f_{12}	30	1.92×10^4	1.00	$\mathbf{9.17\times10^3}$	1.00	1.29×10^4	1.00	1.49×10^4	1.00	1.97×10^4	1.00	3.03×10^4	1.00	1.08×10^4	1.00
f_{13}	30	1.50×10^4	1.00	1.52×10^4	1.00	1.59×10^4	0.97	1.74×10^4	0.87	2.24×10^4	1.00	3.54×10^4	1.00	$\mathbf{1.26\times10^4}$	1.00
f_{14}	30	1.09×10^5	1.00	1.68×10^5	0.97	NA	0.00	NA	0.00	$\mathbf{8.07\times10^4}$	1.00	2.69×10^5	1.00	1.21×10^5	1.00
f_{15}	30	NA	0.00	NA	0.00	1.00×10^5	1.00	7.26×10^4	1.00	1.60×10^5	1.00	NA	0.00	$\mathbf{1.54\times10^4}$	1.00
平均值		4.86×10^4	0.933	4.87×10^4	0.931	4.79×10^4	0.92	4.84×10^4	0.902	3.78×10^4	0.996	7.22×10^4	0.933	2.29×10^4	1.000

另外，本文所提算法设置种群规模 $NP=50$，变异因子 F 和交叉概率 CR 的初始值均设置为 0.5。而 SHADE，DELU，EPSDE，SaDE，JADE 和 CoDE 算法的参数设置均参见其各自的原文献。另外，每种算法对于每个测试函数均独立运行 30 次，取平均值作为最终结果。

3.1 计算代价与可靠性

为了验证所提算法的计算代价和可靠性，采用函数评价次数(FES)和成功率(SR)两个指标来进行评价。此实验的终止条件为算法在最大函数评价次数(MAXFES)内所得解的精度达到预设精度(δ)。FES 为算法求得的解达到预设精度时所需的目标函数的评价次数；SR 为算法的成功求解次数与总运行次数之比，其中，成功求解意为算法所得解的精度在 MAXFE 内达到设定的 δ。实验中，$MAXFES=3\times10^5$，$\delta=1\times10^5$。

表 2 列出了各函数的函数评价次数 FES 和成功率 SR，其中，最优结果加粗表示，“NA”表示算法在 MAXFE 内没有求解到达到预设精度 δ 的结果，即未成功求解，为便于计算，此类结果的值在计算时视为 MAXFE。从表 2 中可以看出，除了函数 f_{12} 和 f_{14} 以外，所提算法的计算代价与可靠性均优于其他算法。从表 2 最后一行的总平均结果可以看出，所提算法的 FES 最小，为 2.29×10^4，而 SHADE，DELU，EPSDE，SaDE，JADE 和 CoDE 算法的 FES 分别为 4.86×10^4，4.87×10^4，4.79×10^4，4.84×10^4，3.78×10^4 和 7.22×10^4，即所提算法相对于 SHADE，DELU，EPSDE，SaDE，JADE 和 CoDE 算法分别节省了 52.87%，52.93%，52.17%，52.64%，39.34% 和 68.27% 的 FES。在成功率方面，所提算法对所有测试函数均以 100% 的成功率进行求解，而 SHADE，DELU，EPSDE，SaDE，JADE 和 CoDE 算法的成功率评价值分别为 0.933，0.931，0.920，0.902，0.996 和 0.933，其中，EPSDE 和 SaDE 算法未能对函数 f_{14} 成功求解，SHADE，DELU 和 CoDE 未能对函数 f_{15} 成功求解。

3.2 优化性能

为了验证所提算法的优化性能，采用 30 次运行的平均函数误差值(Mean error)和标准偏差值(Std)来进行衡量算法所得解的质量。此实验的终止条件为算法的 FES 达到给定的函数评价次数(GFES)。函数误差为算法所得解与全局最优解之间的目标函数差值。实验中，$GFES=2000\times D$，D 为问题的维数。

表 3 列出了各测试函数 30 次运行的平均函数误差和标准偏差，最优结果加粗表示。综合看来，在 15 个测试函数中，所提算法的结果有 10 个优于其他算法。对于函数 f_1，f_8 和 f_{10}，DELU 算法在所有算法中取得了最优结果。相比于其他算法，EPSDE 算法在函数 f_{11} 上的求解结果为最优。对于函数 f_{14}，JADE 算法的结果显著优于其他算法。

表 3 优化结果的性能

Table 3 Performance of optimization results

函数	维数	SHADE	DELU	EPSDE	SaDE	JADE	CoDE	PDSDE
		Mean error (Std)	Mean error (Std)	Mean error (Std)	Mean error (Std)	Mean error (Std)	Mean error (Std)	Mean error (Std)
f_1	30	2.87×10^{-19} (2.31×10^{-19})	$\mathbf{9.25\times10^{-37}}$ ($\mathbf{2.99\times10^{-36}}$)	3.87×10^{-31} (9.46×10^{-31})	1.29×10^{-23} (3.07×10^{-23})	3.01×10^{-20} (8.74×10^{-20})	2.16×10^{-10} (1.73×10^{-10})	1.73×10^{-32} (3.73×10^{-32})
f_2	30	4.40×10^{-20} (3.25×10^{-20})	2.56×10^{-32} (4.00×10^{-32})	5.52×10^{-31} (1.94×10^{-30})	6.59×10^{-25} (8.39×10^{-25})	1.16×10^{-21} (1.59×10^{-21})	2.83×10^{-11} (2.61×10^{-11})	$\mathbf{1.98\times10^{-34}}$ ($\mathbf{2.34\times10^{-34}}$)
f_3	30	4.98×10^{-10} (2.39×10^{-10})	1.46×10^{-11} (1.23×10^{-11})	1.29×10^{-16} (2.15×10^{-16})	8.70×10^{-16} (5.17×10^{-16})	3.11×10^{-10} (3.36×10^{-10})	3.86×10^{-06} (1.32×10^{-06})	$\mathbf{2.45\times10^{-18}}$ ($\mathbf{2.51\times10^{-18}}$)
f_4	30	4.07×10^{-17} (5.44×10^{-17})	$\mathbf{0.00\times10^{+00}}$ ($\mathbf{0.00\times10^{+00}}$)	$\mathbf{0.00\times10^{+00}}$ ($\mathbf{0.00\times10^{+00}}$)	$\mathbf{0.00\times10^{+00}}$ ($\mathbf{0.00\times10^{+00}}$)	$\mathbf{0.00\times10^{+00}}$ ($\mathbf{0.00\times10^{+00}}$)	1.30×10^{-14} (1.13×10^{-14})	$\mathbf{0.00\times10^{+00}}$ ($\mathbf{0.00\times10^{+00}}$)
f_5	30	1.97×10^{-18} (2.27×10^{-18})	1.65×10^{-20} (4.70×10^{-20})	8.85×10^{-30} (1.75×10^{-29})	8.85×10^{-23} (4.35×10^{-22})	1.83×10^{-18} (6.07×10^{-18})	4.04×10^{-10} (3.04×10^{-10})	$\mathbf{3.24\times10^{-32}}$ ($\mathbf{4.23\times10^{-32}}$)
f_6	30	$0.00\times10^{+00}$ ($0.00\times10^{+00}$)	$\mathbf{0.00\times10^{+00}}$ ($\mathbf{0.00\times10^{+00}}$)	$\mathbf{0.00\times10^{+00}}$ ($\mathbf{0.00\times10^{+00}}$)	$\mathbf{0.00\times10^{+00}}$ ($\mathbf{0.00\times10^{+00}}$)	$\mathbf{0.00\times10^{+00}}$ ($\mathbf{0.00\times10^{+00}}$)	$\mathbf{0.00\times10^{+00}}$ ($\mathbf{0.00\times10^{+00}}$)	$\mathbf{0.00\times10^{+00}}$ ($\mathbf{0.00\times10^{+00}}$)

（续表）

函数	维数	SHADE	DELU	EPSDE	SaDE	JADE	CoDE	PDSDE
		Mean error (Std)	Mean error (Std)	Mean error (Std)	Mean error (Std)	Mean error (Std)	Mean error (Std)	Mean error (Std)
f_7	30	3.57×10^{-04} (1.00×10^{-03})	1.82×10^{-04} (3.39×10^{-04})	$1.42\times10^{+01}$ ($2.25\times10^{+01}$)	9.27×10^{-02} (1.30×10^{-01})	6.16×10^{-03} (3.13×10^{-02})	5.69×10^{-04} (7.53×10^{-04})	**3.31×10^{-06}** (**4.37×10^{-06}**)
f_8	30	2.47×10^{-04} (1.35×10^{-03})	**$0.00\times10^{+00}$** (**$0.00\times10^{+00}$**)	6.57×10^{-04} (2.50×10^{-03})	2.22×10^{-03} (4.73×10^{-03})	9.70×10^{-15} (5.25×10^{-14})	2.47×10^{-07} (7.34×10^{-07})	6.34×10^{-17} (1.55×10^{-16})
f_9	30	5.53×10^{-21} (5.00×10^{-21})	2.19×10^{-30} (2.11×10^{-30})	2.91×10^{-29} (1.14×10^{-28})	2.47×10^{-27} (4.90×10^{-27})	3.69×10^{-21} (1.55×10^{-20})	1.80×10^{-13} (3.36×10^{-13})	**1.83×10^{-32}** (**6.22×10^{-33}**)
f_{10}	30	4.07×10^{-21} (6.03×10^{-21})	**1.36×10^{-32}** (**3.13×10^{-34}**)	1.75×10^{-32} (9.04×10^{-33})	1.46×10^{-03} (3.80×10^{-03})	5.07×10^{-22} (2.22×10^{-21})	1.31×10^{-13} (1.51×10^{-13})	1.89×10^{-32} (7.89×10^{-34})
f_{11}	30	1.19×10^{-10} (6.88×10^{-11})	2.42×10^{-10} (1.35×10^{-10})	**6.04×10^{-15}** (**1.66×10^{-15}**)	2.40×10^{-01} (4.45×10^{-01})	4.19×10^{-11} (4.49×10^{-11})	3.75×10^{-06} (1.88×10^{-06})	8.97×10^{-15} (2.84×10^{-15})
f_{12}	30	2.85×10^{-20} (4.26×10^{-20})	4.57×10^{-26} (7.30×10^{-26})	6.02×10^{-32} (1.48×10^{-31})	2.75×10^{-19} (2.53×10^{-19})	5.65×10^{-21} (1.54×10^{-20})	1.44×10^{-12} (1.26×10^{-12})	**1.62×10^{-32}** (**1.55×10^{-33}**)
f_{13}	30	3.51×10^{-19} (3.01×10^{-19})	3.80×10^{-21} (4.03×10^{-21})	1.566×10^{-30} (3.18×10^{-30})	7.75×10^{-18} (8.46×10^{-18})	3.60×10^{-20} (7.81×10^{-20})	2.45×10^{-11} (2.36×10^{-11})	**1.04×10^{-31}** (**1.78×10^{-31}**)
f_{14}	30	$6.53\times10^{+02}$ ($6.50\times10^{+02}$)	$3.67\times10^{+02}$ ($4.24\times10^{+02}$)	$9.64\times10^{+02}$ ($4.81\times10^{+02}$)	$2.40\times10^{+03}$ ($8.58\times10^{+02}$)	**$7.94\times10^{+00}$** (**$1.99\times10^{+01}$**)	$6.92\times10^{+02}$ ($8.00\times10^{+02}$)	$3.77\times10^{+02}$ ($1.46\times10^{+02}$)
f_{15}	30	1.60×10^{-02} (1.92×10^{-03})	2.20×10^{-01} (1.67×10^{-05})	1.38×10^{-03} (1.06×10^{-03})	7.45×10^{-05} (4.86×10^{-05})	9.94×10^{-03} (3.50×10^{-03})	$1.75\times10^{+00}$ ($1.47\times10^{+00}$)	**1.74×10^{-15}** (**1.10×10^{-15}**)

为了进一步验证所提算法的优越性，选用了非参数假设检验——Wilcoxon 检验[19]，来对表 3 所得优化结果进行统计分析，以验证所提算法相对于其他比较算法是否有显著性优势。其中，显著性水平设置为 5%，并且通过“+”表示所提算法显著优于对比算法，“−”表示所提算法显著劣于对比算法，“≈”表示两种算法之间没有显著性差异。如表 4 所列，所提算法相对于 SHADE，DELU，EPSDE，SaDE，JADE 和 CoDE 算法，分别有 12，9，10，13，11，13 个函数的优势。DELU 算法在 4 个函数上显著优于所提算法，SHADE，EPSDE，JADE 算法则在 2 个函数上显著优于所提算法，CoDE 仅在 1 个函数上显著优于所提算法，而 SaDE 并没有在任何测试函数上优于所提算法。

表 4 PDSDE 与其他算法的 Wilcoxon 检验结果

Table 4 Results of Wilcoxon test between PDSDE and other algorithms

PDSDE v. s.	SHADE	DELU	EPSDE	SaDE	JADE	CoDE
+	12	9	10	13	11	13
≈	2	4	2	0	2	1
−	1	2	3	2	2	1

另外，文中继续采用 Friedman 检验[20]对所有算法的优化性能进行统计分析。如表 5 所列，所提算法的排名最高，即所提算法的优化性能优于其他算法，之后的排名分别是 DELU，EPSDE，JADE，SHADE，SaDE 和 CoDE 算法。

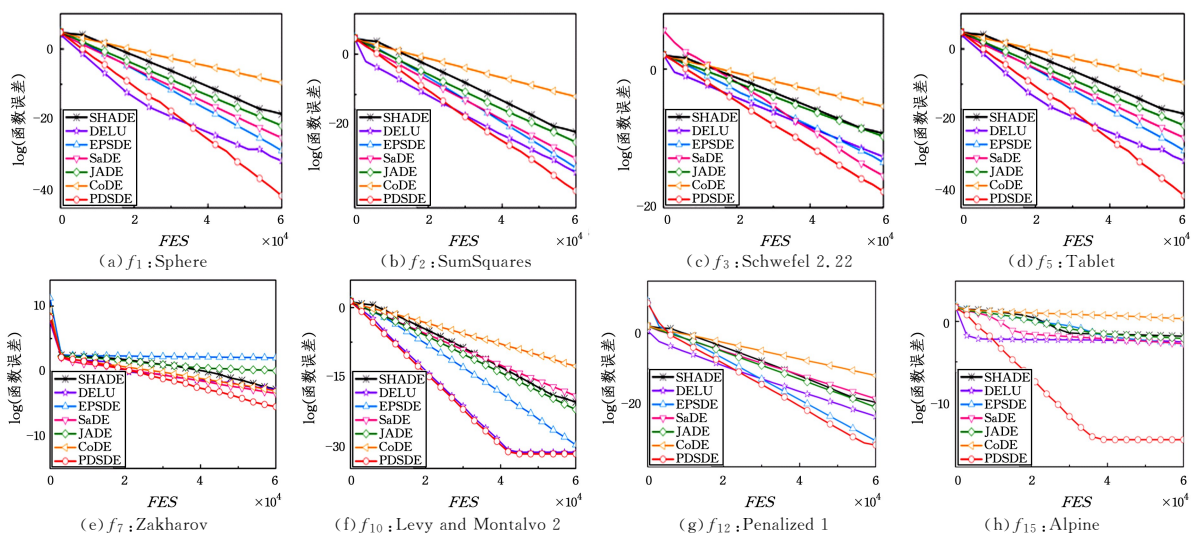
表 5 PDSDE 与其他算法的排名（Friedman 检验）

Table 5 Rankings of PDSDE and other algorithms (Friedman test)

	Ranking		Ranking
SHADE	4.93	JADE	3.93
DELU	2.80	CoDE	5.87
EPSDE	3.53	PDSDE	1.93
SaDE	5.00		

3.3 收敛性能

所提算法的收敛性能采用评价收敛速度来进行衡量，对表 1 中的 8 个测试函数 f_1，f_2，f_3，f_5，f_7，f_{10}，f_{12}，f_{15} 绘制平均收敛曲线图并对其进行分析说明。图 1 给出了这 8 个测试函数的收敛曲线图，图中数据是基于各算法 30 次独立优化的平均结果，为方便制图，横坐标均取函数评价次数，纵坐标均取平均函数误差的对数。



(a) f_1: Sphere (b) f_2: SumSquares (c) f_3: Schwefel 2.22 (d) f_5: Tablet

(e) f_7: Zakharov (f) f_{10}: Levy and Montalvo 2 (g) f_{12}: Penalized 1 (h) f_{15}: Alpine

图 1 PDSDE 和其他算法对 8 个函数的收敛曲线

Fig. 1 Convergence curves of PDSDE and other algorithms on 8 functions

从图 1 可以看出，所提算法的收敛速度优于其他算法，表明根据进化阶段自适应的调整控制参数，能够保证算法快速收敛且得到稳定的高质量解。所提算法由于正确判定了进化阶段，并采用合适的控制参数使得算法自搜索初期就表现出较好的收敛性能，即表现为图 1 中对 8 个测试函数的收敛曲线大部分自初期开始就位于其他算法之下。图 1(a)中的测试函数 f_1 为单模函数，各算法均能对其进行快速求解，其中，DELU 在前期收敛较快，而本文所提算法在适应因子的指导下动态调整策略和参数，在算法后期显示了较强的搜索能力，从而以最快的收敛速度搜索到最优解。图 1(b)－图 1(d)、图 1(f)、图 1(g)也呈现相似特性，所提算法与 DELU 均快速收敛，而所提算法最终解的精度优于 DELU 算法。图 1(e)中，其他对比算法对函数 f_7 均求解缓慢，EPSDE 算法更是陷入了局部最优，图 1(h)也是如此。在对函数 f_{15} 的求解过程中，SHADE，EPSDE，JADE，SaDE 算法收敛缓慢，且 DELU 和 CoDE 算法陷入了局部最优，只有所提算法能够根据进化阶段不断调整相应的变异策略和控制参数，避免算法陷入局部最优，收敛速度始终较快，从而能够找到精度较高的解。因此，所提算法的收敛性能好且能够找到满意解。综合而言，测试结果表明本文所提算法收敛速度快、成功率高且解的质量较高。

结束语 为了平衡算法的全局探测能力和局部搜索能力，本文提出了基于群体分布的自适应差分进化算法，通过种群之间的距离建立适应因子以判断进化阶段，针对不同进化阶段的特点切换变异策略和控制参数，从而实现算法的自适应调整。将本文算法与 6 种主流改进算法基于 15 个基本测试函数进行实验，结果表明，PDSDE 算法在计算代价、可靠性、收敛性能、解的质量方面具有优越性，是一种可靠有效的全局优化算法。进一步的工作将集中于特定阶段的变异、交叉、选择策略的研究，控制参数自适应机制的完善和复杂实际问题的应用研究。

参 考 文 献

[1] STORN R，PRICE K V. Differential evolution：a simple and efficient heuristic for global optimization over continuous spaces [J]. Journal of Global Optimization，1997，11(4)：341-359.

[2] DAS S，SUGANTHAN P N. Differential evolution：a survey of the state-of-the-art [J]. IEEE Transactions on Evolutionary Computation，2011，15 (1)：4-31.

[3] DAS S，MULLICK S S，SUGANTHAN P N. Recent advances in differential evolution－An updated survey [J]. Swarm & Evolutionary Computation，2016，27(6)：1-30.

[4] PANDIT M，SRIVASTAVA L，SHARMA M. Environmental economic dispatch in multi-area power system employing improved differential evolution with fuzzy selection [J]. Applied Soft Computing，2015，28(3)：498-510.

[5] ZHANG G J，DING Q，WANG L J，et al. Optimization method of production scheduling in flexible job [J]. Computer Science，2018，45(2)：269-275.

[6] ZHANG G J，XIA H D，ZHOU X G，et al. Hybrid differential evolution based on tabu search algorithm for distribution network line planning [J]. Computer Science，2016，43(10)：248-255.

[7] PIOTROWSKI A P. Differential evolution algorithms applied to neural network training suffer from stagnation [J]. Applied Soft Computing，2014，21(5)：382-406.

[8] ZHU T，WANG J C，XIONG Z H. DE-based nonlinear model predictive control of a pH neutralization process [J]. Acta Automatica Sinica，2010，36(1)：159-163.

[9] HAO X H，ZHANG G J，ZHOU X G，et al. Protein conformational space optimization algorithm based on fragment-assembly [J]. Computer Science，2015，42(3)：237-240.

[10] ZHOU X G，ZHANG G J，HAO X H，et al. Enhanced differential evolution using local Lipschitz underestimate strategy for computationally expensive optimization problems [J]. Applied Soft Computing，2016，48：169-181.

[11] DRAGOI E N，DAFINESCU V. Parameter control and hybridization techniques in differential evolution：a survey [J]. Artificial Intelligence Review，2016，45(4)：447-470.

[12] QIN A K，HUANG V L，SUGANTHAN P N. Differential evolution algorithm with strategy adaptation for global numerical optimization [J]. IEEE Transactions on Evolutionary Computation，2009，13(2)：398-417.

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

[14] ZHANG J，SANDERSON A C. JADE：Adaptive differential evolution with optional external archive [J]. IEEE Transactions on Evolutionary Computation，2009，13(5)：945-958.

[15] WANG Y，CAI Z，ZHANG Q. Differential evolution with composite trial vector generation strategies and control parameters [J]. IEEE Transactions on Evolutionary Computation，2011，15(1)：55-66.

[16] TANABE R，FUKUNAGA A. Success-history based parameter adaptation for Differential Evolution [C]// 2013 IEEE Congress on Evolutionary Computation. Cancun：IEEE，2013：71-78.

[17] ZHOU X G，ZHANG G J，HAO X H，et al. A novel differential evolution algorithm using local abstract convex underestimate strategy for global optimization [J]. Computers & Operations Research，2016，75(11)：132-149.

[18] ALI M M，KHOMPATRAPORN C，ZABINSKY Z B. A numerical evaluation of several stochastic algorithms on selected continuous global optimization test problems [J]. Journal of Global Optimization，2005，31(4)：635-672.

[19] CORDER G W，FOREMAN D I. Nonparametric statistics for non-statisticians：a step-by-step approach [M]. Hoboken，New Jersey：John Wiley & Sons，2009.

[20] GARCÍA S，FERNÁNDEZ A，LUENGO J，et al. Advanced nonparametric tests for multiple comparisons in the design of experiments in computational intelligence and data mining：Experimental analysis of power [J]. Information Sciences，2010，180(10)：2044-2064.

LI Zhang-wei，born in 1967，Ph.D，is the member of China Computer Federation. His main research interests include intelligent information processing and so on.