一种阶段性策略自适应差分进化算法

倪洪杰 彭春祥 周晓根 俞 立
(浙江工业大学信息工程学院 杭州 310023)

摘 要 针对差分进化算法的变异策略选择问题,提出一种阶段性策略自适应差分进化算法(SSADE)。首先,根据各个体与当前最优个体之间的平均距离衡量种群的拥挤度,进而估计种群的进化阶段;然后,将整个种群划分为多个子种群,并针对不同阶段的特性,设计子种群协同进化变异策略池;最后,根据各变异策略的历史成功信息,从对应的策略池中动态自适应地选择合适的变异策略,从而达到平衡全局探测和局部搜索的目的。在 12 个经典测试函数上的实验结果表明,所提 SSADE 算法在计算代价、可靠性、解的质量和扩展性方面优于现有主流算法。

关键词 差分进化,策略自适应,子种群,全局优化,协同进化

中图法分类号 TP391 **文献标识码** A

Differential Evolution Algorithm with Stage-based Strategy Adaption

NI Hong-jie PENG Chun-xiang ZHOU Xiao-gen YU Li
(College of Information Engineering, Zhejiang University of Technology, Hangzhou 310023, China)

Abstract To solve the problem of the selection for the mutation strategy in differential evolution algorithm, this paper proposed a differential evolution with stage-based strategy adaption. Firstly, the crowding degree of the population is measured by the average distance between each individual and the best individual, and the evolution stage is estimated. Then, the population is divided into multiple sub-populations, and a pool mutation strategy based on the coevolution of sub-populations is designed for different stages. Finally, according to the historical success information of each strategy, a suitable strategy is adaptively selected from the corresponding strategy pool to balance the exploration and exploitation. Experimental results on 12 classical test functions show that the proposed algorithm is superior to the mainstream algorithm in terms of the computational cost, success rate, quality of the solution, and scalability.

Keywords Differential evolution, Strategy self-adaption, Sub-population, Global optimization, Coevolution

1 引言

差分进化算法(DE)是由 Stron 和 Price[1] 提出的一种基于种群的随机性全局优化算法。DE 算法根据种群个体之间的竞争与合作产生的群体智能指导整个优化搜索过程。DE 算法实现简单、鲁棒性强、收敛速度快[2-3],因此,自提出以来被广泛应用于各领域中的优化求解问题[4],如蛋白质结构预测[5]、配电网络规划[6]、货位分配优化[7]和生产排产调度优化[8]等。

与其他基于种群的算法一样,DE 算法也包含变异、交叉和选择 3 个操作[1]。在这 3 个操作中,差分变异操作极其重要[9]。变异操作通过组合随机选择或给定的种群个体生成新个体,其主要目的是提高种群的多样性,从而防止算法陷入局部最优而出现早熟收敛现象[10]。为了提高 DE 算法的性能,近年来,DE/current-to-pbest/1[11]、DE/lbest/1[12]、质心变异策略[9]和高斯变异策略[13]等被相继提出。然而,不同的变异策略具有不同的特性,例如,DE/best/1 策略局部搜索能力强,但是全局探测能力弱,而 DE/rand/1 策略全局探测能力强,但是局部搜索能力弱[14]。因此,将不同的策略应用于不同的进化阶段可能比整个进化过程中仅采用一种变异策略更有效[15]。

为了对不同进化阶段选择合适的变异策略,Qin 等[14]提出一种自适应 DE 算法(SaDE),根据各变异策略的历史成功率选择当前的变异策略。Mallipeddi 等[16]提出一种具系综变异策略及参数的 DE 算法(EPSDE),对种群中的每个个体随机分配不同的变异策略及参数,并保存成功的策略及参数,对失败策略和参数对应的个体重新初始化。Wang 等[17]提出一种复合策略和参数的 DE 算法(CoDE),根据不同的策略和参数组合对每个个体生成多个测试个体,并保留最优的测试个体。Zhou 等[9]提出一种基于下界估计的多变异策略 DE 算法,通过对每个个体同时生成多个测试个体,并根据每个测试个体的抽象凸下界估计值进行筛选。Pan 等[18]也提出了一

本文受浙江省重点研发计划项目(2017C03060),国家自然科学基金(61573317)资助。

倪洪杰(1978-),男,博士生,主要研究方向为智能信息处理、优化理论及算法设计,E-mail:zdfynhj@zjut.edu.cn(通信作者);**彭春祥**(1995-),男,硕士生,主要研究方向为优化理论及算法设计、生物信息学;**周晓根**(1987-),男,博士,CCF 学生会员,主要研究方向为智能信息处理、优化理论及算法设计和生物信息学;**俞 立**(1961-),男,博士,教授,主要研究方向为无线传感器网络、鲁棒控制和网络化控制系统。

种变异策略自适应 DE 算法，将能够成功产生较优个体的变异策略保存到一个动态更新列表中，然后从此列表中选择适合每个个体的变异策略。数值实验表明，上述策略可以提高 DE 算法的性能。

本文提出一种阶段性策略自适应差分进化算法（SSADE）。在 SSADE 算法中，首先，通过各个体与当前最优个体之间的平均距离来衡量种群的拥挤度，从而估计种群所处的进化阶段；然后，将整个种群划分为多个子种群，从而针对不同阶段的特性，基于子种群协同合作思想，设计不同的变异策略，并建立阶段策略池；最后，对于每个个体，根据各策略的历史成功信息，从对应的策略池中动态自适应地选择合适的变异策略。

2　多策略自适应差分进化算法

针对 DE 算法的变异策略选择问题，基于分阶段思想，本文提出一种阶段性策略自适应差分进化算法（SSADE）。所提 SSADE 算法包括基于种群拥挤度的进化阶段估计、多子种群协同变异策略和基于历史成功信息的子种群变异策略 3 个部分，下面对各部分进行详细介绍，并对整个算法流程进行详细描述。

2.1　进化阶段估计

DE 算法在进化初期，种群相对分散，此时，整个种群致力于搜索尽可能多的有前途区域；随着进化过程的进行，整个种群开始逐渐收敛，并在最后收敛到一个全局最优或者局部最优区域[9]。因此，可以根据种群的拥挤度变化来估计算法的进化阶段。

为了衡量种群的拥挤度，首先，计算当前种群中各个体与最优个体（目标函数值最小）之间的平均距离：

$$d_{ave}^{g}=\frac{\sum_{i=1}^{N_P}\sqrt{\sum_{j=1}^{D}(x_j^{i,g}-x_j^{\mathrm{best},g})^2}}{N_P} \tag{1}$$

其中，N_P 为种群规模，D 为问题维数，$x^{\mathrm{best},g}$ 为第 g 代种群中的最优个体。以初始种群的平均距离为其最大值 $d_{\max}$。在进化过程中，将每代的平均距离与 $d_{\max}$ 进行比较，如果当前代的平均距离大于 $d_{\max}$，则替换 $d_{\max}$。由于种群最终收敛到一个点，则平均距离的最小值 $d_{\min}=0$。

然后，根据平均距离的最大值和最小值对每一代的平均距离进行归一化处理，即：

$$\bar{d}_{ave}^{g}=\frac{d_{ave}^{g}-d_{\min}}{d_{\max}-d_{\min}}=\frac{d_{ave}^{g}}{d_{\max}} \tag{2}$$

最后，根据每一代的 $\bar{d}_{ave}^{g}$ 估计当前种群所处的阶段：

$$\Psi=\begin{cases}S_1, & \text{if } \bar{d}_{ave}^{g}>c\\ S_2, & \text{if } 1-c\leqslant\bar{d}_{ave}^{g}<c\\ S_3, & \text{otherwise}\end{cases} \tag{3}$$

其中，Ψ 表示种群的进化阶段，S_1，S_2 和 S_3 分别表示第一、第二和第三阶段，c 为阶段控制因子。

2.2　子种群协同变异策略

为了提高算法的搜索效率，同时维持种群多样性，避免算法陷入局部最优而出现早熟收敛，针对不同阶段的特性，设计不同的策略池进行变异。

首先，将整个种群随机划分成 $N(N\geqslant 4)$ 个大小相等的子种群，各子种群的大小和成员个体在整个进化过程中保持不变。在第一阶段 S_1，整个种群相对分散，所有个体均致力于搜索有希望的子区域，此时应该保持种群多样性，从而保证搜索到尽可能多的区域，因此，在第一阶段，采用以下变异策略进行变异：

1）DE/rand/1

$$v^{i,g}=x^{a,g}+F(x^{b,g}-x^{c,g}) \tag{4}$$

2）DE/current-to-rand/1

$$v^{i,g}=x^{i,g}+F(x^{a,g}-x^{i,g})+F(x^{b,g}-x^{c,g}) \tag{5}$$

其中，$v^{i,g}$ 和 $x^{i,g}$ 分别表示变异个体和目标个体，$a\neq b\neq c\in[1,N_P]$，$x^{a,g}$，$x^{b,g}$ 和 $x^{c,g}$ 为从不同子种群中随机选择的个体。通过不同子种群中的个体协同合作进行变异，从而保持种群的多样性。

在第二阶段 S_2，算法开始探测已经搜索到的有希望的区域，并进一步搜索更有希望的区域。此时算法既要进行全局探测又要进行局部搜索。因此，为了平衡种群多样性与收敛速度之间的关系，通过以下策略进行变异：

1）DE/sbest/1

$$v^{i,g}=x^{\mathrm{sbest},g}+F(x^{a,g}-x^{b,g}) \tag{6}$$

2）DE/current-to-sbest/1

$$v^{i,g}=x^{i,g}+F(x^{\mathrm{sbest},g}-x^{i,g})+F(x^{a,g}-x^{b,g}) \tag{7}$$

其中，$x^{\mathrm{sbest},g}$ 为目标个体 $x^{i,g}$ 所属子种群中的最优个体，$x^{a,g}$ 和 $x^{b,g}$ 为从两个不同的子种群中随机选择的个体。通过局部最优个体和其他子种群中的个体协同合作来指导变异，不仅可以保持种群多样性，保证算法全局探测的动力而搜索尽可能多的区域，还可以提高算法的局部搜索能力，从而加快收敛速度。

在第三阶段 S_3，所有个体可能都聚集到某一个区域中，此时算法致力于搜索该区域中的最优解，因此，为了加快收敛速度，采用以下策略进行变异：

1）DE/best/1

$$v^{i,g}=x^{\mathrm{best},g}+F(x^{a,g}-x^{b,g}) \tag{8}$$

2）DE/ current-to-best/1

$$v^{i,g}=x^{i,g}+F(x^{\mathrm{best},g}-x^{i,g})+F(x^{a,g}-x^{b,g}) \tag{9}$$

其中，$x^{\mathrm{best},g}$ 为整个种群中的最优个体，$x^{a,g}$ 和 $x^{b,g}$ 为从除了 $x^{\mathrm{best},g}$ 所属的子种群以外的两个不同的子种群中随机选择的个体。通过利用当前最优个体的信息指导变异，使得算法快速收敛。

2.3　策略自适应

在各阶段中，每个变异策略均设置有对应的选择概率。由于每个阶段采用两种不同的策略进行变异，因此，在每个阶段的初始代中，每个变异策略的选择概率均被初始化为 0.5。为了更新每个变异策略的选择概率，在每一代结束时，通过每个策略在当前代中的成功次数除以所有策略的成功次数之和更新其选择概率，即：

$$p_k=\frac{NS_k}{\sum_{s=1}^{2}NS_s} \tag{10}$$

其中，p_k 表示第 k 个变异策略的选择概率，NS_k 为第 k 个策

略在当前代中的成功次数，即产生的测试个体成功进入下一代的次数。根据文献[14]的建议，为了避免各变异策略在进化过程中由于表现较差而丢失，对每个变异策略的选择概率加上一个常数 $\xi=0.01$。在进化过程中，首先根据式(3)确定当前代所处的阶段，然后根据对应阶段中各策略的选择概率，利用轮盘赌的方法选择一个变异策略进行变异，从而达到变异策略动态自适应选择的效果。

2.4 算法描述

以 D 维最小化问题为例，所提 SSADE 算法的整体流程如下。

Step 1 初始化：设置种群规模 N_P，交叉概率 CR，增益常数 F，子种群数目 N，阶段控制因子 c，最大进化代数 $G_{\max}$，并随机生成初始种群 $P=\{x^{1,g}, x^{2,g},\cdots,x^{N_P,g}\}$，置进化代数 $g=0$，将各变异策略的选择概率 p_k 设置为 0.5。

Step 2 根据如下操作估计当前种群所处的进化阶段：

1)根据式(1)计算当前种群中各个体与最优个体之间的平均距离 d_{ave}^{g}；

2)根据式(2)对 d_{ave}^{g} 进行归一化处理得到 $\bar{d}_{ave}^{g}$；

3)根据式(3)判断当前种群所处的阶段。

Step 3 将整个种群随机划分成 N 个大小相等的子种群。

Step 4 对当前种群中的每个目标个体 $x^{i,g}$ 进行如下操作：

1)设置 $i=1$；

2)根据每个策略的选择概率，从对应的阶段变异策略池中，利用轮盘赌方法选择一个变异策略生成变异个体 $v^{i,g}$；

3)根据式(11)进行交叉生成测试个体 $x_{\text{trial}}^{i,g}$：

$$x_{\text{trial},j}^{i,g}=\begin{cases}v_j^{i,g}, & \text{if } R\leqslant CR \text{ or } j=r\\ x_j^{i,g}, & \text{otherwise}\end{cases} \tag{11}$$

其中，$j\in\{1,2,\cdots,D\}$，R 为 0 和 1 之间的均匀分布随机小数，r 为 1 和 D 之间的均匀分布随机整数；

4)分别计算测试个体和目标个体的目标函数值 $f(x_{\text{trial}}^{i,g})$ 和 $f(x^{i,g})$；

5)如果 $f(x_{\text{trial}}^{i,g})<f(x^{i,g})$，则 $x_{\text{trial}}^{i,g}=x^{i,g}$；

6)$i=i+1$，如果 $i\leqslant N_P$，转至 Step4 的步骤 2)；

Step 5 根据式(10)更新当前阶段对应的策略池中各变异策略的选择概率；

Step 6 $g=g+1$，如果 $g\leqslant G_{\max}$，则转至 Step4；否则，输出最优解并退出。

3 数值实验

3.1 测试函数及参数设置

为了验证所提 SSADE 算法的性能，选用 12 个文献中常用的经典测试函数进行实验。表 1 给出了各测试函数的名称、最优值和搜索域。函数 F_1-F_5 为单模函数，函数 F_6-F_{12} 为多模函数，且局部最优解的数量随着维数的增大呈现指数增加。上述所有函数均为最小化问题。各函数的具体数学表达式见文献[19-21]。

选用 SaDE[14]、EPSDE[16] 和 CoDE[17] 3 种主流的改进 DE 算法与所提 SSADE 算法进行比较。为了公平，上述 3 种算法均采用其原文献中的参数设置。SSADE 算法设置为：种群规模 $N_P=50$，子种群数目 $N=5$，阶段控制因子 $c=0.85$[9]，交叉概率 CR 和增益常数 F 的设置采用 SaDE[14] 算法中提出的参数自适应机制，具体参见文献[14]。算法对于每个测试函数均独立运行 30 次。实验环境为：Intel(R) Core i5-2410M CPU@2.30 GHz with 8 GB RAM，Windows 7。算法实现代码采用 Visual Studio 2012 C++编写。

表 1 测试函数

函数	最优值	搜索域
F_1：Sphere	0	(−100,100)
F_2：SumSquares	0	(−10,10)
F_3：Schwefel 2.22	0	(−10,10)
F_4：Exponential	0	(−1,1)
F_5：Tablet	0	(−100,100)
F_6：Griewank	0	(−600,600)
F_7：Schaffer 2	0	(−100,100)
F_8：Schwefel 2.26	$-418.983D$	(−500,500)
F_9：Ackley	0	(−30,30)
F_{10}：Rastrigin	0	(−5.12,5.12)
F_{11}：Penalized 1	0	[−50,50]
F_{12}：Penalized 2	0	[−50,50]

3.2 收敛速度与成功率

在此实验中，采用 3 种指标来衡量各算法的性能：1)在给定的最大目标函数评价次数(MaxFES)内，搜索得到的最优值与函数的全局最优值之间的误差达到给定的精度 δ 所需要的目标函数评价次数 FES；2)成功率(SR)，即成功运行次数与总运行次数之比，其中，规定在给定的 MaxFES 内函数误差值达到设定的精度 δ 时为运行成功；3)加速比(AR)：所比算法的目标函数评价次数与 SSADE 算法的目标函数评价次数之比，$AR>1$，$AR=1$ 和 $AR<1$ 分别表示所提算法优于、等价于和差于比较算法。在本实验室中，$\delta=0.000\,01$，MaxFES 设置为 $10\,000\times D$。

表 2 给出了 SaDE，EPSDE，CoDE 和 SSADE 算法对各测试函数独立运行 30 次得到的平均函数评价次数、成功率和加速比，其中，最优结果通过加粗标出。对比表中数据可以看出，在计算代价方面，除了函数 F_{10} 和 F_{11} 以外，所提 SSADE 算法对于其他问题求解时，函数误差值达到给定精度所需要的目标函数评价次数均低于 3 种比较算法。在成功率方面，SSADE 算法对所有函数问题均能以 100% 的成功率进行求解，而 SaDE，EPSDE 和 CoDE 算法分别对 8 个、10 个和 11 个函数以 100% 的成功率求解。在加速比方面，SSADE 算法分别在 11 个、10 个和 11 个问题上优于 SaDE，EPSDE 和 CoDE 算法。

表 2 的最后一行给出 12 个测试函数的平均结果，可以看出，SSADE 算法所需的平均目标函数评价次数最少，即 21 081，SaDE，EPSDE 和 CoDE 算法所需的目标函数评价次数分别为 25 090，23 048 和 49 072，也就是说，SSADE 算与 SaDE，EPSDE 和 CoDE 算法相比，分别节省了 16.0%，8.5% 和 57.0% 的目标函数评价次数；而且 SSADE 算法的平均成功率最高，即 1.00。CoDE 算法同样获得了 1.00 的成功率。由于 SaDE 和 EPSDE 算法对部分问题没有能够成功求解，平均成功率分别为 0.96 和 0.98。

表 2　函数评价次数、成功率和加速比

函数	维数	SaDE			EPSDE			CoDE			SSADE	
		FE	*SR*	*AR*	*FE*	*SR*	*AR*	*FE*	*SR*	*AR*	FE	SR
F_1	30	20297	1.00	1.61	16677	1.00	1.32	40155	1.00	3.18	**12637**	1.00
F_2	30	18410	1.00	1.43	15080	1.00	1.17	36270	1.00	2.82	**12843**	1.00
F_3	30	24368	1.00	1.18	23212	1.00	1.13	56421	1.00	2.74	**20585**	1.00
F_4	30	10993	1.00	1.28	9357	1.00	1.09	22362	1.00	2.60	**8609**	1.00
F_5	30	21117	1.00	1.31	18250	1.00	1.14	41286	1.00	2.57	**16060**	1.00
F_6	30	21740	0.73	1.48	17581	0.83	1.20	43878	1.00	2.98	**14705**	1.00
F_7	30	36050	1.00	1.01	37098	1.00	1.04	59982	1.00	1.68	**35801**	1.00
F_8	30	22873	1.00	1.17	23267	1.00	1.19	35565	1.00	1.82	**19514**	1.00
F_9	30	30453	0.93	1.46	23242	1.00	1.11	56826	1.00	2.72	**20858**	1.00
F_{10}	30	**62508**	0.93	0.96	64025	1.00	0.98	130409	0.97	2.00	65206	1.00
F_{11}	30	14852	1.00	1.14	**12944**	1.00	0.99	30324	1.00	2.33	13022	1.00
F_{12}	30	17421	0.87	1.33	15850	0.97	1.21	35385	1.00	2.70	**13126**	1.00
平均值		25090	0.96		23048	0.98		49072	1.00		**21081**	**1.00**

图 1 给出了 4 个具有代表性的函数（F_1，F_6，F_9 和 F_{10}）的平均收敛曲线图。

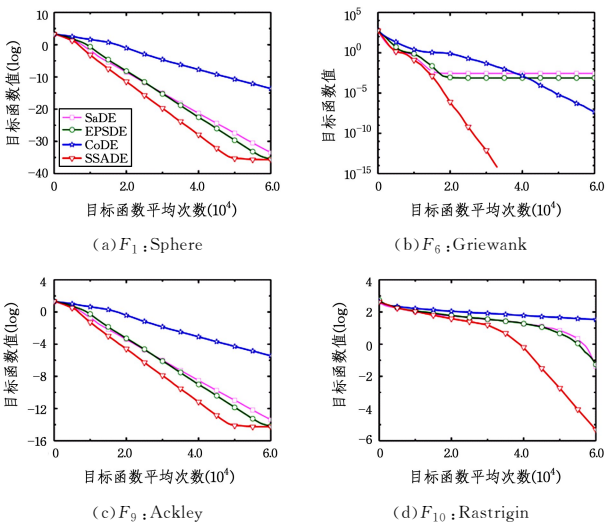

图 1　平均收敛曲线图

可以看出，所提 SSADE 算法的收敛速度明显优于其他 3 种比较算法。对于函数 F_6，SaDE 和 EPSDE 算法均陷入了局部最优，CoDE 算法虽然没有陷入局部最优，但是收敛速度较慢。对于函数 F_{10}，3 种比较算法的收敛速度缓慢或出现了早熟收敛，而 SSADE 算法可以稳健、快速地向全局最优解收敛。

3.3　最优解的质量

为了评价各算法得到的最优解的质量，记录各函数在给定的 MaxFES 内优化得到的结果。同时，根据 30 次独立运行的结果，采用 Wilcoxon signed rank test[22] 进行非参数检验，从而验证所提算法得到的结果与比较算法之间的差异性，通过＋、≈和－分别表示所提算法显著优于比较算法、所提算法与比较方法没有显著性差异、所提算法显著差于比较算法。在本实验中，MaxFES 设置为 $2000\times D$，非参数检验的显著性水平设置为 0.05。

表 3 给出了各算法对各函数独立运行 30 次得到的最优值的平均值、标准偏差和显著性结果。从表中数据可以看出，SSADE 算法对于大部分函数的最优结果优于 3 种对比算法。具体来说，SSADE 算法在 9 个问题上显著优于 SaDE 算法，而 SaDE 算法没有在任何问题上显著优于 SSADE 算法，仅在 3 个问题上取得了与 SaDE 算法相同的结果。与 EPSDE 算法相比，SSADE 算法在 8 个问题上表现出了显著优势，而 EPSDE 算法仅在其余 4 个问题上获得了与 SSADE 算法相同或相似的结果。SSADE 算法在 12 个函数中的 11 个函数上显著优于 CoDE 算法，而 CoDE 算法仅在 1 个函数上得到了与 SSADE 算法相同的结果。

表 3　30 维函数最优值的平均值、标准偏差和显著性

函数	维数	SaDE			EPSDE			CoDE			SSADE	
		Mean	*Std*	*Sig*	*Mean*	*Std*	*Sig*	*Mean*	*Std*	*Sig*	*Mean*	*Std*
F_1	30	1.29×10^{-23}	3.07×10^{-23}	＋	3.87×10^{-31}	9.46×10^{-31}	＋	2.16×10^{-10}	1.73×10^{-10}	＋	$\mathbf{7.04\times10^{-35}}$	$\mathbf{1.55\times10^{-34}}$
F_2	30	6.59×10^{-25}	8.39×10^{-25}	＋	5.52×10^{-31}	1.94×10^{-30}	＋	2.83×10^{-11}	2.61×10^{-11}	＋	$\mathbf{7.63\times10^{-36}}$	$\mathbf{1.71\times10^{-35}}$
F_3	30	8.70×10^{-16}	5.17×10^{-16}	＋	1.29×10^{-16}	2.15×10^{-16}	＋	3.86×10^{-6}	1.32×10^{-6}	＋	$\mathbf{6.18\times10^{-19}}$	$\mathbf{5.69\times10^{-19}}$
F_4	30	0.00	0.00	≈	0.00	0.00	≈	1.30×10^{-14}	1.13×10^{-14}	＋	0.00	0.00
F_5	30	8.85×10^{-23}	4.35×10^{-22}	＋	8.85×10^{-30}	1.75×10^{-29}	＋	4.04×10^{-10}	3.04×10^{-10}	＋	$\mathbf{5.06\times10^{-36}}$	$\mathbf{7.08\times10^{-36}}$
F_6	30	2.22×10^{-3}	4.73×10^{-3}	＋	6.57×10^{-4}	2.50×10^{-3}	＋	2.47×10^{-7}	7.34×10^{-7}	＋	0.00	0.00
F_7	30	0.00	0.00	≈	0.00	0.00	≈	1.66×10^{-2}	3.13×10^{-2}	＋	0.00	0.00
F_8	30	0.00	0.00	≈	0.00	0.00	≈	0.00	0.00	≈	0.00	0.00
F_9	30	2.40×10^{-1}	4.45×10^{-1}	＋	6.04×10^{-15}	1.66×10^{-15}	≈	3.75×10^{-6}	1.88×10^{-6}	＋	$\mathbf{5.42\times10^{-15}}$	$\mathbf{1.95\times10^{-15}}$
F_{10}	30	4.11×10^{-2}	1.82×10^{-1}	＋	5.48×10^{-2}	1.39×10^{-1}	＋	3.30×10^{1}	5.81	＋	$\mathbf{3.43\times10^{-6}}$	$\mathbf{1.64\times10^{-6}}$
F_{11}	30	3.46×10^{-3}	1.89×10^{-2}	＋	6.02×10^{-32}	1.48×10^{-31}	＋	1.44×10^{-12}	1.26×10^{-12}	＋	$\mathbf{1.57\times10^{-32}}$	**0.00**
F_{12}	30	1.83×10^{-3}	4.16×10^{-3}	＋	3.66×10^{-4}	2.01×10^{-3}	＋	2.45×10^{-11}	2.36×10^{-11}	＋	$\mathbf{1.35\times10^{-32}}$	**0.00**

3.4　扩展性分析

为了研究所提 SSADE 算法的性能与问题维数之间的关系，通过 50 维问题研究其扩展性。表 4 给出了各算法对各函数在 MaxFES＝$3000\times D$ 内优化得到的结果。从表中数据可以看出，SSADE 算法对于大部分函数的结果优于 3 种比较算法。具体地，从显著性检验的结果可知，SSADE 算法在 12 个函数中的 7 个函数上显著优于 SaDE 算法，而 SaDE 算法仅在 1 个函数上的结果优于 SSADE 算法，对于其余 4 个函数，SSADE 算法和 SaDE 算法得到了相同或相似的结果。与 EPSDE 算法相比，SSADE 算法在对 9 个函数优化时获得了显著优势，而 EPSDE 算法仅在 1 个函数上得到了显著优于 SSADE 算法的结果，对于其余 2 个函数，两者结果相同。CoDE 算法仅在 4 个函数上获得了与 SSADE 算法相同的结果，对于其余 8 个函数，SSADE 算法得到的结果显著优于

CoDE 算法。由此可以看出，SSADE 算法的性能没有随着问题维数的增加而受到影响。

表 4 50 维函数最优值的平均值、标准偏差和显著性

函数	维数	SaDE			EPSDE			CoDE			SSADE	
		Mean	*Std*	*Sig*	*Mean*	*Std*	*Sig*	*Mean*	*Std*	*Sig*	*Mean*	*Std*
F_1	50	2.40×10^{-40}	3.71×10^{-40}	+	2.52×10^{-50}	7.95×10^{-50}	+	4.94×10^{-21}	5.74×10^{-21}	+	$\mathbf{1.79\times10^{-68}}$	$\mathbf{3.58\times10^{-68}}$
F_2	50	2.42×10^{-40}	3.65×10^{-40}	+	1.80×10^{-51}	3.75×10^{-51}	+	4.46×10^{-22}	6.24×10^{-22}	+	$\mathbf{8.69\times10^{-86}}$	$\mathbf{1.12\times10^{-85}}$
F_3	50	3.67×10^{-25}	2.46×10^{-25}	+	5.63×10^{-29}	1.68×10^{-28}	+	6.05×10^{-12}	2.77×10^{-12}	+	$\mathbf{2.12\times10^{-40}}$	$\mathbf{3.85\times10^{-40}}$
F_4	50	0.00	0.00	≈	1.33×10^{-16}	4.68×10^{-17}	+	0.00	0.00	≈	0.00	0.00
F_5	50	2.03×10^{-39}	2.34×10^{-39}	+	3.04×10^{-50}	7.79×10^{-50}	+	1.01×10^{-20}	1.21×10^{-20}	+	$\mathbf{3.01\times10^{-77}}$	$\mathbf{5.78\times10^{-77}}$
F_6	50	3.04×10^{-2}	4.14×10^{-2}	+	2.46×10^{-3}	5.69×10^{-3}	+	0.00	0.00	≈	0.00	0.00
F_7	50	0.00	0.00	≈	0.00	0.00	≈	0.00	0.00	≈	0.00	0.00
F_8	50	0.00	0.00	≈	0.00	0.00	≈	0.00	0.00	≈	0.00	0.00
F_9	50	8.65×10^{-1}	6.24×10^{-1}	+	1.14×10^{-14}	4.97×10^{-15}	+	8.59×10^{-12}	4.34×10^{-12}	+	$\mathbf{7.55\times10^{-15}}$	**0.00**
F_{10}	50	8.95×10^{-1}	8.71×10^{-1}	−	4.42	1.09×10^{1}	−	4.27×10^{1}	7.39	+	1.54×10^{1}	2.12
F_{11}	50	1.19×10^{-2}	3.76×10^{-2}	+	1.00×10^{-32}	1.96×10^{-33}	+	9.53×10^{-24}	4.52×10^{-24}	+	$\mathbf{9.42\times10^{-33}}$	**0.00**
F_{12}	50	1.35×10^{-32}	2.88×10^{-48}	≈	1.10×10^{-3}	3.47×10^{-3}	+	1.29×10^{-22}	1.22×10^{-22}	+	$\mathbf{1.35\times10^{-32}}$	**0.00**

结束语 本文针对 DE 算法变异策略选择问题，提出一种阶段性策略自适应差分进化算法(SSADE)。在 SSADE 算法中，通过种群个体与当前最优个体之间的平均距离衡量种群的拥挤度，从而估计算法的进化阶段；进一步将整个种群划分为多个子种群，从而针对不同的阶段，结合子种群协同合作思想，设置适合各阶段的变异策略池；在进化过程中，根据各策略的历史成功信息计算对应的选择概率，从而根据各策略的选择概率从对应的阶段策略池中动态自适应地选取合适的策略，从而平衡全局探测和局部搜索的关系，提高了算法的性能。12 个经典测试函数的实验结果表明，所提 SSADE 算法在收敛速度、成功率、解的质量以及扩展性方面优于现有的主流 DE 改进算法。

参 考 文 献

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

[2] 周晓根，张贵军，郝小虎. 局部抽象凸区域剖分差分进化算法[J]. 自动化学报，2015，41(7)：1315-1327.

[3] 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)：388-397.

[4] DAS S, MULLICK S S, SUGANTHAN P N. Recent advances in differential evolution-An updated survey[J]. Swarm & Evolutionary Computation, 2016, 27(4): 1-30.

[5] ZHANG G J, ZHOU X G, YU X F, et al. Enhancing protein conformational space sampling using distance profile-guided differential evolution [J]. IEEE/ACM Transactions on Computational Biology and Bioinformatics, 2017, 14(6): 1288-1301

[6] 张贵军，夏华栋，周晓根，等. 一种配电网络差分禁忌线路规划方法[J]. 计算机科学，2016，43(10)：248-255.

[7] 张贵军，姚俊，周晓根，等. 基于精英多策略的货位分配优化方法[J]. 计算机科学，2018，45(1)：273-279.

[8] 张贵军，丁情，王柳静，等. 柔性车间生产排产调度优化方法[J]. 计算机科学，2018，45(2)：269-275.

[9] ZHOU X G, ZHANG G J. Abstract convex underestimation assisted multistage differential evolution[J]. IEEE Transactions on Cybernetics, 2017, 47(9): 2730-2741.

[10] GONG W, CAI Z. Differential evolution with ranking-based mutation operators[J]. IEEE Transactions on Cybernetics, 2013, 43(6): 2066-2081.

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

[12] YU W J, SHEN M, CHEN W N, et al. Differential evolution with two-level parameter adaptation[J]. IEEE Transactions on Cybernetics, 2014, 44(7): 1080-1099.

[13] WANG H, RAHNAMAYAN S, SUN H, et al. Gaussian bare-bones differential evolution[J]. IEEE Transactions on Cybernetics, 2013, 43(2): 634-647.

[14] 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.

[15] ZHOU X G, ZHANG G J. Differential evolution with underestimation-based multimutation strategy[J]. IEEE Transactions on Cybernetics, 2018, PP(99): 1-12.

[16] 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.

[17] 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.

[18] PAN Q K, SUGANTHAN P N, WANG L, et al. A differential evolution algorithm with self-adapting strategy and control parameters[J]. Computers & Operations Research, 2011, 38(1): 394-408.

[19] 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 & Operation Research, 2016, 75(11): 132-149.

[20] 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(11): 169-181.

[21] 周晓根，张贵军，郝小虎，等. 一种基于局部 Lipschitz 下界估计支撑面的差分进化算法[J]. 计算机学报，2016，39(12)：2631-2651.

[22] CORDER G W, FOREMAN D I. Nonparametric statistics for non-statisticians: A step-by-step approach[M]. Hobo- ken: John Wiley & Sons, 2009.