求解大规模优化问题的正交反向混合差分进化算法*

董小刚[1]，邓长寿[1]，谭毓澄[1]，彭 虎[1,2]
(1. 九江学院 信息科学与技术学院，江西 九江 332005；2. 武汉大学 软件工程国家重点实验室，武汉 430072)

摘 要： 差分进化算法简单高效，然而在求解大规模优化问题时，其求解性能迅速降低。针对该问题，提出一种正交反向差分进化算法。首先，该算法利用正交交叉算子，加强了算法的局部搜索能力。其次，为防止过强的局部搜索使算法陷入早熟收敛，利用反向学习策略调节种群多样性，从而有效地平衡算法的全局和局部搜索能力。利用 11 个标准测试函数进行实验，并和差分进化算法的四种优秀改进版本进行比较，实验结果表明提出的算法求解精度高、收敛速率快，是一种求解大规模优化问题的有效算法。

关键词： 大规模优化问题；差分进化；正交交叉；反向学习
中图分类号： TP301.6 **文献标志码：** A **文章编号：** 1001-3695(2016)06-1656-06
doi:10.3969/j.issn.1001-3695.2016.06.013

Hybridization differential evolution algorithm of orthogonal crossover and opposition-based learning for large-scale optimization problem

Dong Xiaogang[1], Deng Changshou[1], Tan Yucheng[1], Peng Hu[1,2]
(1. *School of Information Science & Technology, Jiujiang University, Jiujiang Jiangxi* 332005, *China*; 2. *State Key Lab of Software Engineering, Wuhan University, Wuhan* 430072, *China*)

Abstract: Differential evolution is simple and efficient. However, when solving the large-scale optimization problems, the performance decreases rapidly. To overcome this problem, this paper proposed a hybridization differential evolution algorithm of orthogonal crossover and opposition-based learning. In the hybrid algorithm, it used orthogonal crossover to enhance the exploitation ability and adopted opposition-based learning to adjust the diversity of population. Thus it could balance the local and global search ability efficiently. It tested the new algorithm on 11 standard benchmark problems and compared with other four famous variants of differential evolution. The results show that performance of the algorithm is better than those of the compared algorithms in terms of accuracy and speed. Thus, it can be an efficient algorithm for large scale optimization problems.
Key words: large-scale optimization problems; differential evolution; orthogonal crossover; opposition-based learning

0 引言

随着社会的发展，大规模数据优化问题不断涌现。例如，机器学习领域中深度学习的多层次神经网络权值训练问题、工程设计领域中器件形状优化设计问题以及大规模生物计算等。由于问题规模的增加使得求解复杂度呈指数级增长，先前成功的寻优策略往往无法取得理想的求解结果。

进化算法的优秀分支差分进化(differential evolution，DE)[1]算法，是一种具备很强竞争力的优化方法。在过去的近二十年中，DE 算法的各种改进版本在 30～100 维的小规模优化问题上取得了巨大的成功[2]。例如，2006 年 Brest 等人[3]提出控制参数自适应的 jDE 算法，很好地解决了 DE 算法参数设置困难的问题；2011 年 Wang 等人[4]提出基于变异策略和参数组合的差分进化算法 CoDE，该算法通过对候选池中策略和参数的随机组合来调节算法的进化过程，提高了 DE 算法的优化性能；2012 年 Wang 等人[5]为增强 DE 算法的局部寻优能力，提出了 OXDE 算法。该算法利用正交实验改进 DE 算法的交叉算子，使得算法在局部空间上进行相对充分的搜索，加速了算法的收敛速率。这些 DE 的改进算法在低维的数值优化问题上均取得优于标准 DE 算法的求解效果。然而，当面对大规模(1 000 维以上)的优化问题时，其求解性能迅速下降，无法取得理想的优化结果。为此，研究者们不断创新，探索 DE 算法求解大规模优化问题的新方法。2008 年，Yang 等人[6]在协同进化的框架之下提出一种新的协同差分进化算法 DECCG，该算法采用了随机分组，协同进化的方法，在 1 000 维大规模优化问题上取得了较好的求解结果。但是，为了使相关的变量能够高概率地进入同一分组，DECCG 需要进行多轮的协同进化，这给算法的执行带来一定的消耗，影响了算法的整体性能，且 DECCG 中优化器的选择较为敏感。

本文提出一种正交反向混合差分进化(hybridization differential evolution algorithm of orthogonal crossover and opposition-based learning，HDEOO)算法，该算法利用正交交叉算子提升算法的局部搜索能力，加快算法的收敛速率。同时在每一代进化中，采用反向学习算子对种群进行调节，防止进化过程

收稿日期：2015-02-09；修回日期：2015-04-02 基金项目：国家自然科学基金资助项目(61364025)；武汉大学软件工程国家重点实验室开放基金资助项目(SKLSE 2012-09-39)；江西省教育厅科学技术资助项目(GJJ13729，GJJ14742)；九江学院科研资助项目(2013KJ27，2014KJYB034)

作者简介：董小刚(1979-)，男，陕西宝鸡人，讲师，硕士，主要研究方向为智能计算(dxg110@jju.edu.cn)；邓长寿(1972-)，男，教授，博士，主要研究方向为智能计算；谭毓澄(1964-)，男，副教授，主要研究方向为应用数学；彭虎(1981-)，男，讲师，博士，主要研究方向为智能计算.

过早停滞，实验结果表明该算法能够有效地求解大规模优化问题。

1　标准 DE 算法

DE 算法是一种基于实数编码的群体优化算法，其主要进化过程包括种群初始化、变异、交叉以及选择四个步骤。

a) 初始种群。DE 算法首先在一个 D 维空间上依据式(1)随机地产生 NP 个受上下界约束的个体 $x_{i,0}, i \in [1, NP]$。

$$x_{ij,0} = x_{\min j} + \text{rand} \times (x_{\max j} - x_{\min j}) \tag{1}$$

其中：$j \in [1, D]$，$x_{\min j}$和 $x_{\max j}$分别为第 j 维空间的上下边界。

b) 变异算子。变异算子是 DE 算法的核心的算子。变异算子是利用当前种群中多个随机选择的个体之间的差向量产生变异向量(记为 $v_{i,g+1}$)。DE 算法的变异算子有多种形式，其效率各有不同，其中最常采用的变异算子是 DE/rand/1/bin，如式(2)所示：

$$v_{i,g+1} = x_{r_1,g} + F \times (x_{r_2,g} - x_{r_3,g}) \tag{2}$$

其中：$r_1 \neq r_2 \neq r_3 \neq i$；$r_1, r_2, r_3 \in [1, NP]$；$F$ 为缩放因子。

c) 交叉算子。它是依据交叉概率对变异向量 $v_{i,g+1}$ 与 $x_{i,g}$ 进行重组，从而获得实验向量 $u_{i,g+1}$。二项式交叉是 DE 算法最常采用的交叉算子，如式(3)所示。

$$u_{ij,g+1} = \begin{cases} v_{ij,g+1} & \text{if rand} < CR \text{ or } j = R(i) \\ x_{ij,g} & \text{otherwise} \end{cases} \tag{3}$$

其中：$i = 1, 2, \cdots, NP$；$j = 1, 2, \cdots, D$；$R(i) \in [1, D]$；rand 是一个均匀分布的随机数；$CR \in [0, 1]$ 为交叉概率。

d) 选择。DE 算法的选择采用贪婪策略，即让实验向量 $u_{i,g+1}$和目标向量 $x_{i,g}$ 按照目标函数的适应度值进行竞争，获胜者进入下一代种群。其具体定义如式(4)所示：

$$x_{i,g+1} = \begin{cases} u_{i,g+1} & \text{if } f(u_{i,g+1}) \leqslant f(x_{i,g}) \\ x_{i,g} & \text{otherwise} \end{cases} \tag{4}$$

2　正交反向混合差分进化算法

2.1　正交交叉算子

DE 算法是一种全局搜索能力非常强的进化算法，提高其优化性能的有效途径是设法增强在局部空间上的搜索能力。为此，本文采用正交交叉算子来增强 DE 算法的局部搜索性能。

2.1.1　正交实验设计

在实验中，多因素、多水平的实验非常常见。如果对实验系统中的因素和水平进行组合实验，则总的实验规模将迅速增大，这样的充分实验在实际工程环境中往往难以实现[7]。正交实验设计是解决这种工程实验的有效方法，它是利用正交表来安排实验，以寻找实验的优化方案，其优势是能以较少的实验次数和较低的实验消耗，获取较为满意的实验结果。

本质上，正交实验设计是通过选择每个因素的最优水平进行组合来安排实验，从而减少实验次数和实验消耗。例如，一个有 F 个因素、Q 个水平的实验中，充分的组合实验需要 Q^F 次实验，假设 $F = 3$、$Q = 2$，则总共需要 8 次实验。然而，若通过构造一个 3 因素、2 水平的正交表 $L_4(2^3)$ 来进行实验方案的安排，则只需要安排 4 次实验即可获得较为满意的实验效果。相比于 8 次的全面实验，这种实验方案的安排大大减少了实验次数。当所考查的因素数和水平数越大，该方法优势更明显。

2.1.2　正交交叉算子

DE 算法的进化过程中，交叉算子重要性尤为突出。而交叉算子最常采用的方法是二项式交叉(如式(3)所示)。以 2 维空间为例，假设 $x_i(x_1, y_1)$ 和 $v_i(x_2, y_2)$ 是待交叉的两个向量，则通过二项式交叉，获得的结果向量是 $q_i(x_1, y_2)$ 和 $p_i(x_2, y_1)$ 两者的其中之一(图 1)。显然，在由这四个点所构成的矩形空间中，仅仅对两个顶点进行搜索是远远不够的，在矩形空间内部存在更优秀解的可能是存在的。然而，二项式交叉并没有对矩形空间进行充分的搜索，这就限制了算法的整体寻优性能。同理，在维数较大的空间中，二项式交叉也存在这种限制。

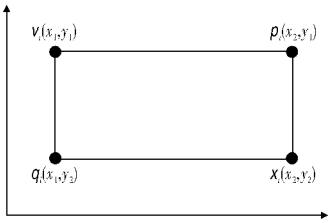

图 1　二维二项式交叉

一个理所当然的改进途径是设法对上述矩形区间(多维数时为超矩型空间)进行充分搜索。然而，逐一实验该空间中的每个点(即全面实验)是不可能实现的。为此，本文引入正交实验设计方法，采用正交交叉算子来取代原有的二项式交叉。该算子根据正交性，选取空间中若干个“均匀分散，整齐可比”的代表点进行实验，达到减少实验次数的情况下获得满意的空间搜索结果。

本文算法的正交交叉采用了 2001 年 Leung 等人[8]提出的量化正交交叉方法(QOX)。假设有两个待交叉父个体 $p = (p_1, p_2, \cdots, p_D)$ 和 $q = (q_1, q_2, \cdots, q_D)$，其中 D 为维数，那么 p 和 q 所构成解空间在第 i 维上的取值区间可描述为 $[\min(p_i, q_i), \max(p_i, q_i)]$，则依据 QOX 方法进行正交交叉的具体步骤可描述为：

a) 离散化搜索空间(由待交叉的两个父个体和构成)，并定义每一维空间的 Q 个水平值。

b) 随机产生 $[1, D]$ 之间的 $F-1$ 个整数 $k_1, k_1, \cdots, k_{F-1}$，依据此将 $[1, D]$ 分成 F 段，构成正交交叉的 F 个因素。

c) 使用正交表 $L_M(Q^F)$，进行正交交叉，产生 M 个后代个体。

其中，对第 i 维的 Q 个水平值的计算依据式(5)进行：

$$l_{i,j} = \begin{cases} \min(p_i, q_i) & j = 1 \\ \min(p_i, q_i) + (j-1) \times \left(\dfrac{|p_i - q_i|}{Q-1}\right) & \text{otherwise} \\ \max(p_i, q_i) & j = Q \end{cases} \tag{5}$$

例如，在一个 7 维空间上两个待交叉的父个体分别为 $p = (8, 2, 10, 9, 20, 7, 3)$ 和 $q = (1, 9, 6, 2, 13, 8, 5)$，选取标准的 3 水平、4 因素正交表 $L_9(3^4)$ 进行交叉(其中 $L_9(3^4)$ 是本文算法中将要采用的正交表)，则：

a) 离散空间所得每一维水平值 l 为

$$l = \begin{cases} l_1 = [1.0, 4.5, 8.0] \\ l_2 = [2.0, 5.5, 9.0] \\ l_3 = [6.0, 8.0, 10.0] \\ l_4 = [2.0, 5.5, 9.0] \\ l_5 = [13.0, 16.5, 20.0] \\ l_6 = [7.0, 7.5, 8.0] \\ l_7 = [3.0, 4.0, 5.0] \end{cases}$$

b) 随机产生 [1,7] 之间的三个随机数，设分别为 2、4、6。则将 7 维空间分为 4 段，分别为 $\text{factor}_1=(x_1,x_2)$、$\text{factor}_2=(x_3,x_4)$、$\text{factor}_3=(x_5,x_6)$ 及 $\text{factor}4=(x_7)$。

c) 使用正交表 $L_9(3^4)$ 进行正交交叉，所得 9 个后代个体为

$$u=\begin{cases}u_1=[1.0,2.0,6.0,2.0,13.0,7.0,3.0]\\u_2=[1.0,2.0,8.0,5.5,16.5,7.5,4.0]\\u_3=[1.0,2.0,10.0,9.0,20.0,8.0,5.0]\\u_4=[4.5,5.5,6.0,2.0,16.5,7.5,5.0]\\u_5=[4.5,5.5,8.0,5.5,20.0,8.0,3.0]\\u_6=[4.5,5.5,10.0,9.0,13.0,7.0,4.0]\\u_7=[8.0,9.0,6.0,2.0,20.0,8.0,4.0]\\u_8=[8.0,9.0,8.0,5.5,13.0,7.0,5.0]\\u_9=[8.0,9.0,10.0,9.0,16.5,7.5,3.0]\end{cases}$$

QOX 方法利用正交性选取搜索空间中均匀分散、整齐可比的代表点进行实验安排，有效地减少了实验的次数，节省了实验资源和时间。

2.2 反向学习策略

反向学习是近年来计算智能领域新出现的概念，已被证明能够显著提高优化方法性能[9~11]。反向学习策略一经提出，便在计算智能领域引起了广泛关注。

2.2.1 反向学习

中国古代经典《易经》有“太极生两仪，两仪生四象，四象生八卦”之说，其所表达的哲学理念是相反相成是事物发生、发展、演化的规律和根源。从这些对立又互补的哲学理念及自然事物所表现出来的各种相反相成的规律性获得灵感，Tizhoosh 等人于 2005 年首次提出反向学习的概念(opposition-based learning，OBL)[12]，并证明了与随机产生的近似解相比较，反向候选解能以更大的概率接近全局最优解。Tizhoosh 等人的研究成果推动了反向学习策略在优化问题上的应用。反向学习相关定义如下：

定义 1 令 $P=(x_1,x_2\cdots,x_D)$ 为 D 维空间中的一个点，其中 $x_i\in[a_i,b_j]$，且 $i=1,2,\cdots,D$，则点 P 的反向点定义为 $\breve{P}=(\breve{x}_1,\breve{x}_2,\cdots,\breve{x}_D)$，其中：

$$\breve{x}_i=a_i+b_i-x_i \tag{6}$$

定义 2 令 $P=(x_1,x_2\cdots,x_D)$ 是 D 维空间中的一个点(候选解)，$\breve{P}=(\breve{x}_1,\breve{x}_2,\cdots,\breve{x}_D)$ 是其反向点，$f(\cdot)$ 是衡量候选解优劣的适应度函数。如果 $f(\breve{P})\geqslant f(P)$，那么用点 $\breve{P}$ 来代替点 P；否则继续使用点 P。通过同时评价点 P 和反向点 $\breve{P}$ 的适应度，得到相对优秀的候选解。

2.2.2 一般反向学习

在反向学习理论的基础之上，王晖等人为进一步提高优化算法找到最优解的概率，提出一般反向学习(generalized OBL，GOBL)的概念。一般反向学习理论具体描述如下：

设 x 是当前空间 s 上的一个解，$x\in[a,b]$。则对应反向空间上 x 的反向解 $\bar{x}$ 定义为

$$\breve{x}=\Delta-x \tag{7}$$

其中：Δ 是一个可计算值，根据式(7)可知，$\bar{x}\in[\Delta-b,\Delta-a]$。使 $\Delta=k(a+b)$，其中 k 为实数，则 GOBL 的定义为

$$\breve{x}=k(a+b)-x \tag{8}$$

将 GOBL 理论，一般化到 D 维空间上如式(9)所示：

$$\breve{x}_j=k(a+b)-x_j \tag{9}$$

其中，$j=1,2,\cdots,D$。

GOBL 依据 k 的取值不同，有多种不同的方案，其中搜索效率最高的是 k 取值为 [0,1] 之间的随机数。为了能适应单模、多模等各类优化问题，避免远离全局最优的反向解进入到下一代种群中，GOBL 采用了一种精英选择机制，即将所有当前解和反向解合并成一个解集，然后依据适应度从低到高排序，选取前面的 NP(NP 为种群大小)个解作为新的解集。

2.3 正交反向混合差分进化算法

正交交叉和反向学习策略分别从两个不同方面对标准 DE 算法进行改进。正交交叉通过对局部空间的充分搜索，较好地提高了算法的局部寻优能力；而反向学习策略本质上是变换了算法的搜索空间，在搜索当前空间的同时，对反向空间也进行搜索，这在一定程度上避免了陷入局部最优，提高了算法找到最优解的概率。

为了进一步提高 DE 算法的整体性能，使其在求解大规模优化问题时获得较好的结果，充分结合正交交叉和反向学习策略的优势，本文提出正交反向混合差分进化(HDEOO)算法。HDEOO 算法是将 DE 算法的每一次迭代过程划分为两个阶段。第一阶段，在标准 DE 算法的每一次迭代中利用正交交叉算子，对一个随机选取的个体进行交叉操作。同时，为提高搜索效率，该个体变异操作采用随机数作为缩放因子。仅对一个个体进行正交交叉，目的是增强算法局部寻优能力的同时避免评价次数的急剧增加。第二阶段，为了避免过强的局部寻优使算法陷入局部优化而导致早熟收敛，从种群(大小为 NP)中选取 20% 的个体，利用一般反向学习策略 GOBL 求得对应的反向解，然后将原有种群和所有反向解合并，依据适应度值从低到高排序，选取前 NP 个作为下一代种群。HDEOO 算法的具体步骤如下：

a) 初始化参数(包括缩放因子 F、交叉概率 CR、种群规模 NP 以及正交表 $L_9(3^4)$)，设置代数 $G=0$，产生搜索空间内的 NP 个个体构成初始种群 P_0。

b) 对种群 P_0 进行评价，$FES=NP$。

c) 判断结束条件是否满足。若满足，转步骤 p)；否则，转步骤 d)。

Stage 1：

d) 随机产生 [1,NP] 内的一个整数 K；

e) 设置变量 $i=1$；

f) 判断 $i==K$ 是否满足。若满足转步骤 g)；否则转步骤 j)；

g) 对个体 $P_{i,G}$ 进行变异产生个体 $V_{i,G}$；

h) 对 $P_{i,G}$ 和 $V_{i,G}$ 按照正交表 $L_9(3^4)$ 进行正交交叉操作，产生 9 个实验个体。

i) 评价这 9 个个体，并选择最优个体作为 $P_{i,G}$ 的后代个体 $U_{i,G}$，同时 $FES=FES+9$，转步骤 k)。

j) 直接对 P_i 进行变异和二项式交叉操作产生后代个体 $U_{i,G}$，评价 $U_{i,G}$，同时 $FES=FES+1$，转步骤 k)。

k) 判断，若 $i==NP$，转步骤 c)；否则 $i=i+1$，转步骤 f)；

l) 选择，判断若 $f(U_{i,G})<f(P_{i,G})$，使 $P_{i,G+1}=U_{i,G}$；否则，$P_{i,G+1}=P_{i,G}$；

Stage 2：

m) 选取种群 P_{G+1} 中 20% 的个体，依据式(7)(其中参数 k 取值为 [0,1] 之间的随机数)，求其反向个体 $\breve{P}_{G+1}$($\breve{P}_{G+1}$ 大小为 $NP\times0.2$)。

n) 评价 $\breve{P}_{G+1}$，同时 $FES = FES + NP \times 0.2$。

o) $P_{G+1} \cup \breve{P}_{G+1}$，并依据适应度值，选取前 NP 个重新作为 P_{G+1}。转步骤 c)。

p) 输出最优解，结束算法。

3　实验与分析

3.1　测试函数

本文实验选择了 11 个广泛使用的标准测试函数[13]，进行仿真，函数维度取 1 000。这 11 个函数的特点各不相同，其中 $f_1 \sim f_5$ 是单峰函数，$f_6 \sim f_{11}$ 是多峰函数。各函数特征描述如表 1 所示。

表 1　测试函数

类型	函数	名称	区间	最优解
单峰	f_1	Sphere Model	[-100,100]	0
	f_2	Schwefel's 1.2	[-100,100]	0
	f_3	Rosenbrock	[-30,30]	0
	f_4	Step	[-100,100]	0
	f_5	Quartic With Noise	[-1.28,1.28]	0
多峰	f_6	Schwefel's 2.26	[-500,500]	$-418.98 \times D$
	f_7	Rastrigin	[-5.12,5.12]	0
	f_8	Ackley	[-32,32]	0
	f_9	Griewank	[-600,60]	0
	f_{10}	Penalized 1	[-50,50]	0
	f_{11}	Penalized 2	[-50,50]	0

3.2　实验设置和结果

为充分验证 HDEOO 算法的性能，分别进行了两组实验。

实验 1　将 HDEOO 算法与三种经典的 DE 算法(jDE，CoDE，OXDE) 进行比较，该实验中所有四种算法的种群规模设置为 $NP = 100$，OXDE 和本文算法 HDEOO 的缩放因子和交叉概率设置为 $F = 0.9$，$CR = 0.9$，其他两种算法采用算法原有设置。算法结束条件均设置为 $D \times 10\ 000$ 次评价。各算法对每个问题独立执行 30 次，记录最差解、最优解、均值和方差。实验结果见表 2。为使仿真结果更加公平客观，采用 MATLAB 所提供的 Wilcoxon 秩和检验方法进行统计分析，显著水平值取 0.05，在表 2 中均值之后分别用“+”“-”“≈”表示优、劣和相似(与 HDEOO 比较)。检验结果的统计数据见表 3。图 2 ~ 6 给出了四种算法 30 次运行的平均收敛曲线图，为节省篇幅，仅选取五个问题绘制收敛曲线图。将四种算法对 11 个问题的求解均值进行 Friedman 检验排名，检验结果如表 4 所示。

表 2　CoDE、jDE、OXDE 及 OXOPDE 结果比较和 WilCoxon 检验结果

fun	算法	最好值	最差值	均值	方差
f_1	CoDE	7.43e-016	4.71e-014	9.62e-015 -	1.03e-014
	jDE	1.26e-035	1.27e-019	5.45e-021 -	2.37e-020
	OXDE	1.11e+000	1.85e+002	2.81e+001 -	4.46e+001
	HDEOO	0.00e+000	0.00e+000	0.00e+000	0.00e+000
f_2	CoDE	1.97e+004	3.72e+004	2.68e+004 -	4.09e+003
	jDE	4.40e+004	8.80e+004	5.98e+004 -	1.05e+004
	OXDE	4.54e+005	7.81e+005	5.83e+005 -	8.08e+004
	HDEOO	0.00e+000	0.00e+000	0.00e+000	0.00e+000
f_3	CoDE	1.47e+003	2.36e+003	1.85e+003 -	1.95e+002
	jDE	1.54e+003	1.97e+003	1.76e+003 -	1.18e+002
	OXDE	3.28e+003	8.43e+003	5.06e+003 -	1.12e+003
	HDEOO	9.13e+002	9.57e+002	9.21e+002	7.85e+000

续表 2

fun	算法	最好值	最差值	均值	方差
f_4	Code	0.00e+000	5.00e+000	4.40e+000 -	1.05e+001
	jDE	4.64e+003	2.74e+004	1.33e+004 -	5.52e+003
	OXDE	3.36e+003	6.80e+003	5.09e+003 -	8.69e+002
	HDEOO	0.00e+000	0.00e+000	0.00e+000	0.00e+000
f_5	CoDE	3.83e-001	5.65e-001	4.88e-001 -	4.82e-002
	jDE	2.31e+000	1.52e+002	3.84e+001 -	3.49e+001
	OXDE	2.42e+000	4.75e+000	3.35e+000 -	6.17e-001
	HDEOO	3.31e-006	5.46e-005	2.88e-005	1.22e-005
f_6	CoDE	-2.88e+005	-2.70e+005	-2.78e+005 -	4.18e+003
	jDE	-4.19e+005	-4.18e+005	-4.19e+005 -	2.02e+002
	OXDE	-4.19e+005	-4.17e+005	-4.18e+005 ≈	5.37e+002
	OXOPDE	-4.19e+005	-4.16e+005	-4.18e+005	6.02e+002
f_7	Code	2.08e+003	2.73e+003	2.41e+003 -	1.63e+002
	jDE	4.618e-011	2.39e+001	7.26e+000 -	7.18e+000
	OXDE	3.49e+002	5.64e+002	4.27e+002 -	4.63e+001
	HDEOO	0.00e+000	0.00e+000	0.00e+000	0.00e+000
f_8	CoDE	1.90e+000	2.68e+000	2.24e+000 -	1.79e-001
	jDE	4.06e+000	7.56e+000	5.75e+000 -	8.33e-001
	OXDE	5.86e+000	6.80e+000	6.36e+000 -	2.23e-001
	HDEOO	8.88e-016	4.44e-015	4.09e-015	1.08e-015
f_9	CoDE	6.66e-016	5.47e-001	3.87e-002 -	1.14e-001
	jDE	1.25e-010	4.03e+000	1.32e+000 -	1.18e+000
	OXDE	1.73e-001	1.73e+000	7.48e-001 -	4.16e-001
	HDEOO	0.00e+000	0.00e+000	0.00e+000	0.00e+000
f_{10}	CoDE	7.47e-013	3.75e-002	7.80e-003 ≈	1.01e-002
	jDE	2.39e-010	6.12e+006	7.46e+005 -	1.42e+006
	OXDE	6.95e-001	2.27e+000	1.48e+000 -	4.14e-001
	HDEOO	2.93e-005	3.80e-003	6.22e-004	1.10e-003
f_{11}	CoDE	7.47e-013	3.75e-002	3.13e-001 +	9.04e-001
	jDE	4.82e-007	3.70e+007	3.79e+006 ≈	9.47e+006
	OXDE	6.36e+002	1.22e+003	9.08e+002 -	1.28e+002
	HDEOO	5.40e-001	3.74e+000	2.00e+000	8.91e-001

表 3　CoDE、jDE、OXDE 与 HDEOO 的 Wilcoson 检验结果统计

比较	CoDE	jDE	OXDE
-	9	10	10
+	1	0	0
≈	1	1	1

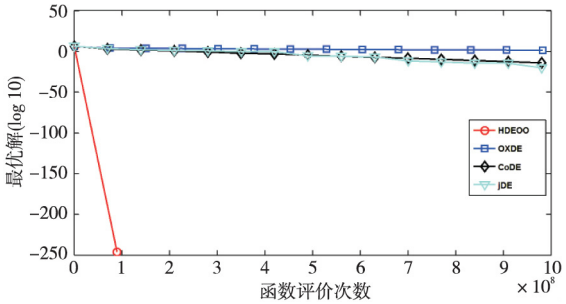

图 2　HDEOO、CoDE、jDE、OXDE 四种算法 f_1 收敛曲线图

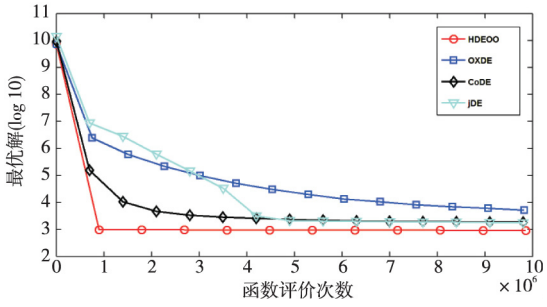

图 3　HDEOO、CoDE、jDE、OXDE 四种算法 f_3 收敛曲线图

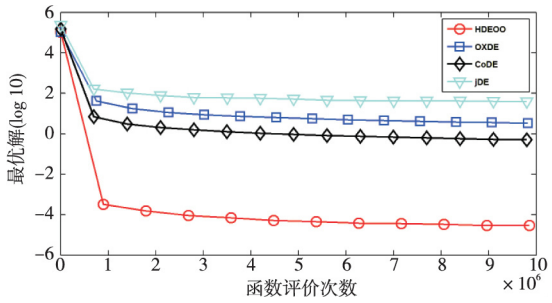

图 4 HDEOO、CoDE、jDE、OXDE 四种算法 f_5 收敛曲线图

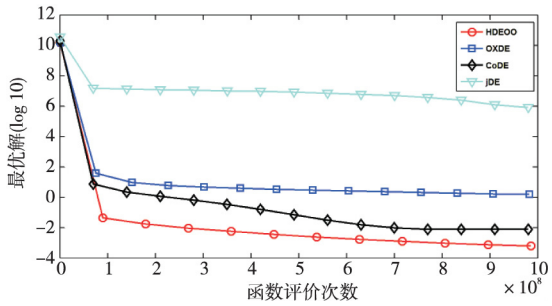

图 5 HDEOO、CoDE、jDE、OXDE 四种算法 f_{10} 收敛曲线图

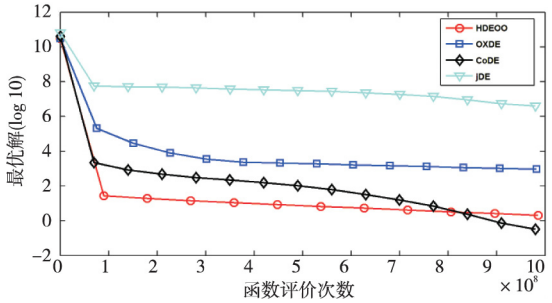

图 6 HDEOO、CoDE、jDE、OXDE 四种算法 f_{11} 收敛曲线图

表 4 CoDE、jDE、OXDE、HDEOO 四种算法 Friedman 检验排名

算法	HDEOO	CoDE	jDE	OXDE
排名	1.27	2.45	2.95	3.32

实验 2 将 HDEOO 算法和基于协同框架的知名算法 DECCG 进行比较，该实验中两种算法的种群设置均为 NP = 100，HDEOO 算法的缩放因子和交叉概率设置为 F = 0.9，CR = 0.9，DECCG 算法的其他参数采用其原有设置，两种算法的结束条件均设置为 $D \times 5\,000$ 次评价。两种算法对每个问题独立执行 25 次，记录最差解、最优解、均值和方差。具体实验结果如表 5 所示。同样，采用 0.05 显著水平的 Wilcoxon 检验方法对实验结果进行了检验，表 5 中“+”“-”“≈”三种符号与实验一种所表示含义相同，表 6 对检验进行了统计，图 7 ~ 11 给出了两种算法对五个问题的 25 次平均收敛曲线图(为与原 DECCG 算法保持一致，收敛图横轴坐标为进化代数)。

3.3 结果分析

表 2 的实验结果表明 HDEOO 算法所取得的结果多数优于 CoDE、jDE 和 OXDE 三种算法。f_1、f_2、f_3、f_4、f_5 五个问题的数据结果表明，在单峰优化问题上，HDEOO 算法的求解性能全面优于其他三种算法。尤其是 f_1、f_2、f_4 三个问题 HDEOO 算法迅速收敛于问题的最优解，而 CoDE、jDE、OXDE 均未能求得最优解，且出现了不同程度的进化停滞，HDEOO 算法优势非常明显。在 f_3 和 f_5 两个问题上，HDEOO 算法虽没有取得最优解，但其所获得的最优解、最差解及均值都优于其他三种算法，且方差值明显较小，说明 HDEOO 算法稳定性较好。从 f_6、f_7、f_8、f_9、f_{10}、f_{11} 六个问题的结果来看，对多峰问题的求解，HDEOO 算法总体优于其他三种算法，在 f_7、f_8、f_9 和 f_{10} 四个问题上优势较大。在问题 f_6 上，HDEOO 算法和 OXDE 算法均收敛于最优解，性能相当，jDE 算法结果稍弱，CoDE 结果最差。在问题 f_{11} 上，HDEOO 算法所得结果的均值略差于 CoDE 算法，但强于其他两种算法。以上结果表明，HDEOO 算法在多峰问题的求解上总体优于其他三种算法。从表 3 的 Wilcoxon 检验统计来看，在所有优化问题上，HDEOO 算法 9 个优于 CoDE、10 个优于 jDE 和 OXDE，只有一个问题上差于 CoDE 算法。图 2 ~ 6 的收敛曲线图表明，对于单峰问题，HDEOO 算法收敛能力显著增强。对多峰问题，HDEOO 算法虽然没有收敛于最优解，但多数问题的收敛能力均强于其他三种算法，只是 f_{11} 在进化后期弱于 CoDE 算法。表 4 的 Friedman 检验排名结果，HDEOO 算法的值最小，排名第一，说明在四种算法中，HDEOO 算法的优化性能整体优于 CoDE、jDE、OXDE 三种算法。综上所述，实验 1 的结果，表明 HDEOO 算法在对大规模优化问题的求解上精度好，收敛快，整体优于其他三种知名 DE 算法。

表 5 DECCG、HDEOO 结果比较和 WilCoxon 检验结果

fun	算法	最优值	最差值	均值	方差
f_1	DECCG	4.45e-029	1.58e-028	9.61E-029 -	3.11e-029
	HDEOO	0.00e+000	0.00e+000	0.00e+000	0.00e+000
f_2	DECCG	2.25e-004	2.80e-003	1.20E-003 -	6.70e-004
	HDEOO	0.00e+000	0.00e+000	0.00e+000	0.00e+000
f_3	DECCG	9.85e+002	9.87e+002	9.86E+002 -	4.11e-001
	HDEOO	9.45e+002	9.89e+002	9.52e+002	8.35e+000
f_4	DECCG	0.00e+000	0.00e+000	0.00e+000 ≈	0.00e+000
	HDEOO	0.00e+000	0.00e+000	0.00e+000	0.00e+000
f_5	DECCG	1.50e-003	3.70e-003	2.62e-003 -	6.68e-004
	HDEOO	3.92e-006	1.06e-007	5.25e-005	2.71e-005
f_6	DECCG	-4.19e+005	-4.19e+005	-4.19e+005 +	9.28e-011
	HDEOO	-4.18e+005	-4.15e+005	-4.17e+005	5.59e+002
f_7	DECCG	0.00e+000	3.55e-014	1.25e-014 -	7.49e-015
	HDEOO	0.00e+000	0.00e+000	0.00e+000	0.00e+000
f_8	DECCG	1.15e-013	1.39e-013	1.27-013 -	7.11e-015
	HDEOO	4.44e-015	7.99e-015	4.58e-015	7.11e-016
f_9	DECCG	6.66e-016	1.33e-015	9.50e-016 -	1.44e-016
	HDEOO	0.00e+000	0.00e+000	0.00e+000	0.00e+000
f_{10}	DECCG	6.20e-028	1.95e-027	1.24e-027 +	3.30e-028
	HDEOO	7.59e-004	6.80e-003	2.74e-003	1.31e-003
f_{11}	DECCG	5.27e-024	1.10e-002	2.64e-003 +	4.79e-003
	HDEOO	4.10e+000	3.78e+001	8.50e+000	6.48e+000

表 6 DECCG 与 HDEOO 的 Wilcoson 检验结果统计

比较	DECCG
-	7
+	3
≈	1

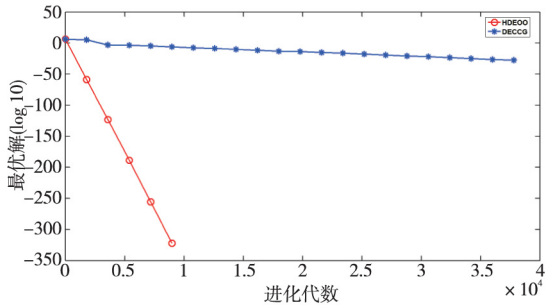

图 7 HDEOO、DECCG 两种算法 f_1 收敛曲线图

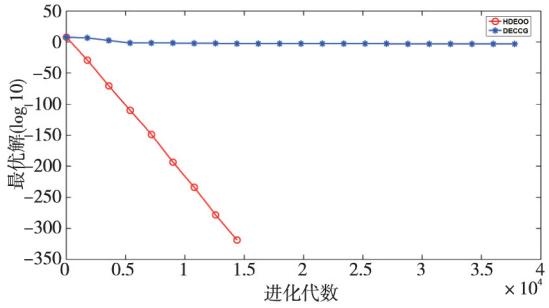

图 8　HDEOO、DECCG 两种算法 f_2 收敛曲线图

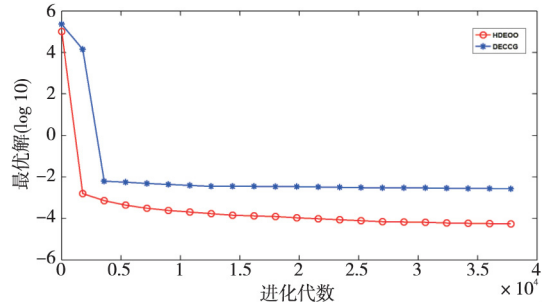

图 9　HDEOO、DECCG 两种算法 f_5 收敛曲线图

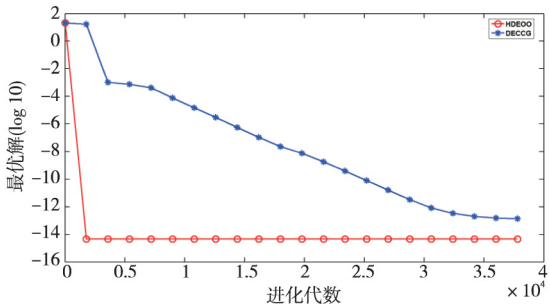

图 10　HDEOO、DECCG 两种算法 f_8 收敛曲线图

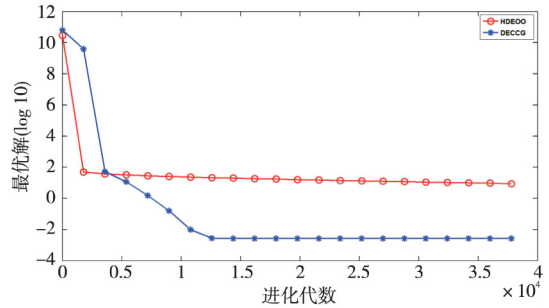

图 11　HDEOO、DECCG 两种算法 f_{11} 收敛曲线图

比较表 5 的实验结果，HDEOO 算法在五个单峰问题上的表现依然明显胜出，其中对 f_1、f_2、f_4 三个问题，HDEOO 算法均最终收敛于全局最优解，而 DECCG 算法，只在 f_4 上收敛于全局最优。在 f_5 问题上，HDEOO 算法的结果均值比 DECCG 高出两个数量级，其中最优解和最差解优势更大，且方差进一步缩小，说明 HDEOO 算法优势显著。然而，问题 f_3 的两种算法所取得结果接近，说明两种方法对该问题的求解性能相似。在另外的六个多峰问题上，HDEOO 算法在 f_7、f_8、f_9 三个问题上胜出，且优势比较明显；在 f_6、f_{10}、f_{11} 三个问题上 HDEOO 算法的表现差于 DECCG。对 f_6 问题，HDEOO 算法的结果在均值上相差不大，主要是稳定性上弱于 DECCG。对 f_{10}、f_{11} 两个问题 HDEOO 算法明显弱于 DECCG 算法。表 6 的 Wilcoxon 检验数据的统计结果显示出，在所有 11 个问题上 HDEOO 算法取得的结果 7 优、3 劣、1 相当，因此总体上优于 DECCG 算法。

4　结束语

针对大规模优化问题，提出了一种新的正交反向混合差分进化算法 HDEOO，HDEOO 算法充分结合了正交交叉和反向学习两种策略的优势，使得其能在整个进化过程中既能保持较强的局部寻优能力，又能及时对种群进行调节，有效防止了进化停滞。实验结果和统计分析表明 HDEOO 算法在对大规模优化问题的求解上收敛精度高，收敛速率快，是一种有效求解大规模优化问题的方法。后期，将尝试把 HDEOO 算法以异构优化器的方式嵌入到协同进化框架中，进一步提升其优化能力。

参考文献：

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

[2] Tang K, Yao X, Suganthan P N, *et al*. Benchmark functions for the CEC'2008 special session and competition on large scale global optimization [R]. Hefei: Nature Inspired Computation and Applications Laboratory, USTC, 2007.

[3] Brest J, Greiner S, Boskovic B, *et al*. Self-adapting control parameters in differential evolution: a comparative study on numerical benchmark problems [J]. IEEE Trans on Evolutionary Computation, 2006, 10(6): 646-657.

[4] Wang Yong, Cai Zixing, Zhang Qingfu. Differential evolution with composite trial vector generation strategies and control parameters [J]. IEEE Trans on Evolutionary Computation, 2011, 15(1): 55-66.

[5] Wang Yong, Cai Zixing, Zhang Qingfu. Enhancing the search ability of differential evolution through orthogonal crossover [J]. Information Sciences, 2012, 18(1): 153-177.

[6] Yang Zhenyu, Tang Ke, Yao Xin. Large scale evolutionary optimization using cooperative coevolution [J]. Information Sciences, 2008, 178(15): 2986-2999.

[7] 吴浩扬，常炳国，朱长纯. 遗传算法的一种特例——正交试验设计法 [J]. 软件学报，2001，12(1)：148-153.

[8] Leung Y W, Wang Yuping. An orthogonal genetic algorithm with quantization for global numerical optimization [J]. IEEE Trans on Evolutionary Computation, 2001, 5(1): 41-53.

[9] Rahnamayan S, Tizhoosh H R, Salama M M A. Opposition based differential evolution algorithms [C]//Proc of IEEE Congress on Evolutionary Computation. 2006: 2010-2017.

[10] Rahnamayan S, Tizhoosh H R, Salama M M A. Opposition based differential evolution for optimization of noisy problems [C]//Proc of IEEE Congress on Evolutionary Computation. 2006: 1865-1872.

[11] Wang Hui, Wu Zhijian, Rahnamayan S. Enhanced opposition-based differential evolution for solving high-dimensional continuous optimization problems [J]. Soft Computing, 2011, 15(11): 2127-2140.

[12] 向万里，马寿峰. 一种高效率收敛的反向差分进化算法 [J]. 小型微型计算机系统，2014，35(2)：343-347.

[13] Yao Xin, Liu Yong, Lin Guangming. Evolutionary programming made-faster [J]. IEEE Trans on Evolutionary Computation, 1999, 3(2): 82-102.