第 41 卷 第 7 期 计 算 机 工 程 2015 年 7 月
Vol. 41 No. 7 Computer Engineering July 2015

·专栏· 文章编号：1000-3428(2015)07-0017-08 文献标识码：A 中图分类号：TP18

求解高维优化问题的正交动态差分进化算法

董小刚[a]，邓长寿[a]，谢 清[a]，柯 林[b]，刘 妍[a]

（九江学院 a. 信息科学与技术学院；b. 理学院，江西 九江 332005）

摘 要： 为解决差分进化算法求解高维优化问题时效率低下的问题，提出一种正交动态差分进化算法（ODDE），通过动态差分进化框架增强全局搜索能力，利用基于正交实验设计的正交交叉算子加强局部空间搜索能力。基于 9 个标准测试函数，选择 30，100，300，500 这 4 种维度对差分进化算法、动态差分进化算法、正交差分进化算法和 ODDE 算法进行对比实验，结果表明，ODDE 算法的求解精度和收敛速率均优于对比算法，可广泛用于求解工程应用中的高维优化问题。

关键词： 高维优化问题；动态差分进化；正交设计；局部搜索；正交交叉

中文引用格式： 董小刚，邓长寿，谢 清，等. 求解高维优化问题的正交动态差分进化算法[J]. 计算机工程，2015，41(7)：17-24.

英文引用格式： Dong Xiaogang，Deng Changshou，Xie Qing，et al. Orthogonal Dynamic Differential Evolution Algorithm for Solving High-dimensional Optimization Problem[J]. Computer Engineering，2015，41(7)：17-24.

Orthogonal Dynamic Differential Evolution Algorithm for Solving High-dimensional Optimization Problem

DONG Xiaogang[a]，DENG Changshou[a]，XIE Qing[a]，KE Lin[b]，LIU Yan[a]

（a. School of Information Science and Technology；b. School of Science，Jiujiang University，Jiujiang 332005，China）

【Abstract】 In order to enhance the performance of Dynamic Differential Evolution（DDE）for solving high dimensional optimization problems，an Orthogonal Dynamic Differential Evolution（ODDE）algorithm is proposed. ODDE is based on the framework of DDE algorithm，so it has very powerful global search ability. At the same time，orthogonal crossover operator based on orthogonal experiment design method is used to enhance the local search ability of algorithm. Nine commonly used benchmark problems with different dimensional size 30，100，300 and 500 are used to evaluate the performance of ODDE which is compared with Differential Evolution（DE），DDE，Orthogonal Crossover Differential Evolution（OXDE）algorithm. Numerical results show that the performance of solving accuracy and convergence rate of ODDE is superior to other algorithms，it can be widely used to solve high-dimensional optimization problems in engineering application.

【Key words】 high-dimensional optimization problem；Dynamic Differential Evolution（DDE）；orthogonal design；local search；orthogonal crossover

DOI： 10.3969/j.issn.1000-3428.2015.07.004

1 概述

大数据时代的高维优化问题在科学研究和工程应用中不断出现，比如航天器设计、大规模无线传感器布局优化等。这些优化问题一般具有非线性、不可微、多峰、复杂度高等特性，依赖工程经验的传统优化方法通常难以求解[1]。因此，探索并研究简化高维问题的方法十分必要[2]。进化计算技术可以不受相关条件限制，能够有效地求解高维优化问题。高维优化问题的进化计算求解一直是学术界的热门话题之一，文献[3]提出一种改进型萤火虫群算法，在高维优化问题的求解精度和速度上取得了较好的效果。文献[4]提出一种混合型 PSO 算法求解30 维的复杂优化问题，取得了理想结果。文献[5]通过改进随机克隆算子和超变异算子，提出一种新型的免疫进化算法，提高了算法求解高维优化问题的整体

基金项目： 国家自然科学基金资助项目（61364025）；江西省教育厅科学技术基金资助项目（GJJ13729，GJJ14742）；武汉大学软件工程国家重点实验室开放基金资助项目（SKLSE2012-09-39）；九江学院科研基金资助项目（2013KJ27）。

作者简介： 董小刚（1979－），男，讲师、硕士，主研方向：智能计算；邓长寿，教授、博士；谢 清、柯 林，讲师、硕士；刘 妍，副教授、硕士。

收稿日期： 2014-07-28 **修回日期：** 2014-08-29 **E-mail：** dxg110@jju.edu.cn

性能。文献[6]通过结合多父体杂交算法、差分进化算法以及蜂群算法的优势,提出一种混合蜂群算法,该算法对 30 维的优化问题求解具有一定优势。差分进化[7](Differential Evolution,DE)算法是一种求解连续优化问题的新型算法,对低维度的函数优化问题(100 维以内)可以获得满意结果。为提高 DE 算法的求解速度,文献[8]提出一种混沌差分进化算法,该算法在每一代进化中,在最优个体附近进行 k 次混沌搜索,得到 k 个个体,然后选择其中的最优个体与当前最优解进行贪梦选择,从而增强算法整体搜索能力。文献[9]提出一种自适应局部搜索交叉算子,通过引入该算子显著增强 DE 算法的整体性能。文献[10]将单纯型确定算法与差分进化算法的 DE/rand/1/bin 方案相结合,提出一种混合差分进化算法,在求解高维多模态函数优化问题上取得了理想效果。文献[11]提出一种动态差分进化(Dynamic Differential Evolution, DDE)算法。文献[12]提出一种单种群二次变异的改进差分进化算法,该算法利用单种群动态进化策略和二次变异的局部搜索技术,有效平衡了算法的全局搜索和局部搜索,从而提高求解速度和精度。文献[13]为提高算法的局部搜索能力,将正交设计思想融入交叉算子,提出一种正交差分进化(OXDE)算法。现有基于 DE 的算法大多是求解低维(100 维以内)的函数优化问题。对于高维优化问题(100 维以上),由于搜索空间随着问题维数的增加呈指数增长,算法性能(速度和精度)会急剧下降,无法求得满意解。

本文在文献[11-12]的基础上,利用正交交叉算子局部寻优能力强的优势,提出一种新的正交动态差分进化(Orthogonal Dynamic Differential Evolution, ODDE)算法求解高维优化问题。ODDE 结合 DDE 的全局搜索能力和正交交叉算子的局部搜索优势,能够有效求解高维优化问题。

2 DE 算法

DE 算法是一种基于实数编码的群体优化算法,主要过程包括初始种群、个体变异、交叉以及选择 4 个步骤,具体描述如下:

(1)初始种群:DE 算法首先在一个 D 维空间上随机产生 NP 个受上下界约束的个体 $X_{i,0}$, $i \in [1,NP]$。

(2)变异算子:DE 算法的变异算子是利用当前种群中 2 个随机选择的个体之间的差向量产生新的中间个体(记为 $V_{i,g+1}$)。DE 算法的变异算子有多种形式,其中最常用的变异算子是 DE/rand/1/bin,具体描述如式(1)所示:

$$v_{i,g+1} = x_{r1,g} + F \cdot (x_{r2,g} - x_{r3,g}) \tag{1}$$

其中,$r1 \neq r2 \neq r3 \neq i$, $r1, r2, r3 \in [1, NP]$; F 为缩放因子。

(3)交叉算子:交叉是将变异得到的中间个体 $V_{i,g+1}$ 与 $X_{i,g}$ 进行杂交从而得到候选个体 $U_{i,g+1}$。基本 DE 算法通常采用二项式交叉算子产生候选个体,在交叉过程中保证候选个体中至少有一维来自 $V_{i,g+1}$,其具体描述如式(2)所示:

$$u_{ij,g+1} = \begin{cases} V_{ij,g+1} & \text{if} \quad rand < CR \quad \text{or} \quad j = R(i) \\ x_{ij,g} & \text{otherwise} \end{cases} \tag{2}$$

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

(4)选择算子:DE 算法的选择采用贪梦策略,即使候选个体 $U_{i,g+1}$ 和当前种群的 $X_{i,g}$ 按照目标函数的适应度值进行竞争,从而选择更优秀的个体进入下一代种群,其具体描述如式(3)所示:

$$x_{i,g+1} = \begin{cases} u_{i,g+1} & \text{if} \quad f(u_{i,g+1}) \leqslant f(x_{i,g}) \\ x_{i,g} & \text{otherwise} \end{cases} \tag{3}$$

DE 算法在迭代寻优的过程中,当前种群中的个体经过上述变异、交叉和选择算子得到优秀个体的基因,在本次进化过程中不能发挥作用,而只有进入下一代才有可能参与进化。DE 算法的这种机制不仅影响了其收敛速度,而且也一定程度限制了其搜索范围。

3 正交动态差分进化算法

正交动态差分进化(ODDE)算法充分利用 DDE 算法的全局搜索能力强的优势,并基于正交交叉算子提高其局部搜索能力,从而提高整体搜索性能。

3.1 动态差分进化算法

经典 DE 算法在进化过程中,当前种群一直保持不变,直到一次迭代完成。因此,在整个进化过程中,当前种群中经变异、交叉操作所产生的优秀个体和基因在本次迭代过程中不能发挥作用,而只有其进入下一代群体后才能够发挥优势,即 DE 算法的进化中不允许不同代的多个个体进行变异和交叉操作。这种局限性不仅在一定程度上影响了 DE 算法的收敛速度,也限制了 DE 算法的搜索范围。此外,从自然界的生物进化来看,不同代的个体进行杂交和变异也会时常发生。

DDE 算法采用动态进化策略,动态进化策略的核心是改变经典 DE 算法进化时保持当前种群不变的模式。在每一代的进化中,当某个个体经过变异和交叉所产生的候选个体(既后代个体)适应度更优时,及时地用候选个体将当前个体替换掉,

使当前种群获得实时更新,促使优秀候选个体在当代进化中发挥作用。相对于 DE 算法的静态进化策略,动态进化策略在一定程度上加快了算法的全局寻优速率。动态差分进化算法得进化步骤具体描述为:

Step1　初始化参数,产生搜索空间内的 NP 个体构成初始种群 P。

Step2　对种群 P 进行评价。

Step3　判断终止条件是否满足,若满足,则转 Step8;否则转 Step4。

Step4　设置变量 $i=1$。

Step5　对 P_i 进行变异和交叉,产生后代个体 U_i。

Step6　判断如果 $f(U_i)<f(P_i)$,更新当前种群 P(用 U_i 替换当前种群 P 中的个体 P_i),否则保持当前种群 P 不变。

Step7　如果 $i==NP$,则转 Step3;如果 $i<NP$,则使 $i=i+1$,并转 Step5。

Step8　输出最优解,结束算法。

上述动态差分进化算法的步骤,与经典 DE 算法的关键差别是 Step6,即当产生更优秀的后代个体时,及时地用后代个体替换种群中父代个体。这样就能使该后代个体的优秀基因及时地参与到种群中其他个体的进化过程中,从而加速算法的整体寻优速率。

3.2　基于正交实验设计的交叉算子

3.2.1　正交实验设计

在工程实验中,当需要考虑 H 个因素,且每个因素有 Q 个水平时,若要进行充分的实验,则必须对 Q^H 个组合进行逐次实验。如果实验中需要考虑的因素和水平数较大时,则需要进行的实验次数将非常大,在实际的工程中受资源和环境的限制,这样的逐次实验显然是不可能的。

正交实验设计是一种研究多因素、多水平的设计方法,是从全面实验中挑选出部分有代表性的点进行实验,这些有代表性的点具备均匀分散、齐整可比的特点,正交实验设计是分析因式设计的主要方法,是一种高效率、快速、经济的实验设计方法。如上文所述,对于 H 个因素、Q 个水平的实验系统,可以利用一个 $\boldsymbol{L}_M(Q^H)$ 的正交表进行实验,其中,M 是安排要进行的实验组合数,这样的实验安排中所要进行的实验次数 M 是远少于原先充分实验的次数 Q^H,这就大幅减少了实验工作量。例如对于一个四因素、三水平的实验,可以按照一个 $\boldsymbol{L}_9(3^4)$(如式(4)所示)的正交表进行安排,这样原本需要进行的 3^4(81)次的实验,则只需安排 9 次实验即可达到效果,减少了 72 次实验,从而大幅提高效率。

$$\boldsymbol{L}_9(3^4)=\begin{bmatrix}1&1&1&1\\1&2&2&2\\1&3&3&3\\2&1&2&3\\2&2&3&1\\2&3&1&2\\3&1&3&2\\3&2&1&3\\3&3&2&1\end{bmatrix} \tag{4}$$

DE 算法交叉算子本质上是从每个因素的不同水平进行组合,因此,可以借用正交实验方法来提高交叉算子的效率。

3.2.2　正交交叉算子

DE 算法最常采用的交叉算子是二项式交叉。以二维搜索空间为例,交叉算子如图 1 所示。二项式交叉算子对目标个体 P_i 和变异个体 V_i 进行交叉,得到候选个体为 U_i 或 U'_i。显然,U_i 与 U'_i 2 个点并不能代表 P_i 和 V_i 所构成的矩形空间中的所有点,在此矩形空间中很可能存在优于 P_i,U_i 和 U'_i 的点。因此,二项式交叉算子并不能对矩形空间进行充分搜索,这影响了算法整体寻优能力。在维数较大的空间中,二项式交叉同样存在以上不足,而且其影响会更加明显。

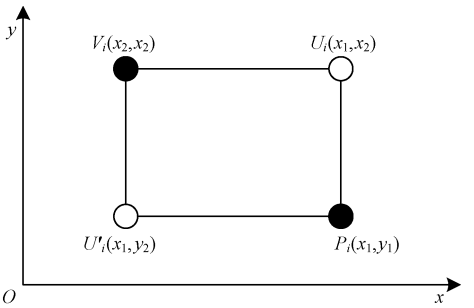

图 1　二项式交叉示意图

如何对 P_i 和 V_i 构成的矩形区间(多维时为超矩形空间)进行充分搜索是提高算法性能的关键之一。显然,不可能穷举实验空间中每个点。因此,借鉴正交实验的优势,在 ODDE 算法中设计一种正交交叉算子来增强其局部搜索能力。正交交叉算子具体操作如下:

Step1　首先对参与交叉的目标个体 P_i 和变异个体 V_i 所形成的矩形空间进行分割,将其分割成 Q 个水平。

Step2　对参与交叉的 2 个个体分别按维数进行分段,将其分成 H 个分段,即构成正交实验的 H 个因素。

Step3　使用 $\boldsymbol{L}_M(Q^H)$ 的正交表对 2 个个体进行正交交叉,产生 M 个候选个体(在 ODDE 算法中,取

$Q=3$、$H=4$、$M=9$,既使用式(4)所示的正交表进行正交交叉),从这 M 个候选个体中选出最优者与目标个体 P_i 进行选择操作。

3.2.3 正交动态差分进化算法流程

本文提出的正交动态差分进化算法的具体步骤如下:

Step1 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 $\boldsymbol{L}_9(3^4)$),产生搜索空间内的 NP 个个体构成初始种群 P。

Step2 对种群 P 进行评价。

Step3 判断结束条件是否满足,若满足,转 Step13;否则转 Step4。

Step4 随机产生[1,NP]范围内的一个整数 K。

Step5 设置变量 $i=1$。

Step6 判断 $i==K$ 是否满足,若满足,转 Step7;否则转 Step10。

Step7 对个体 P_i 进行变异产生个体 V_i。

Step8 对 P_i 和 V_i 按照正交表 $\boldsymbol{L}_9(3^4)$ 进行正交交叉操作,产生 9 个实验个体。

Step9 从这 9 个个体中选择最优个体作为 P_i 的后代个体 U_i,转 Step11。

Step10 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,转 Step11。

Step11 若 $f(U_i)<f(P_i)$,则更新当前种群 P(用 U_i 替换当前种群 P 中的个体 P_i);否则保持种群 P 不变。

Step12 若 $i==NP$,转 Step3;否则 $i=i+1$,转 Step6。

Step13 输出最优解,结束算法。

如上所述,ODDE 进化过程中随机选取一个个体(Step4 随机产生[1,NP]范围内的整数 K)进行正交实验,这种策略可以有效降低目标函数的评价次数。同时为了能够更加充分地对变异个体和目标个体所构成的局部空间进行搜索,被选中进行正交交叉的个体,在变异操作时缩放因子 F 设置为[0,1]之间的一个均匀分布的随机数,而其他个体的变异操作仍采用固定缩放因子 F。

4 实验结果与分析

4.1 测试问题

为验证 ODDE 算法的寻优性能,选择文献[9]中的 9 个常用的测试函数作为测试集,对算法性能进行测试。这 9 个函数的特征各不相同,其中大多都是多峰函数,拥有多个局部极值点,复杂度比较高。

9 个测试函数及特征具体如下:

(1)$f_1(x)=\sum_{i=1}^{N}x_i^2$ 函数,单峰,变量不相关,定义域为[-100,100],最优值为 0。

(2)$f_2(x)=\sum_{i=1}^{N-1}(100(x_{i+1}-x_i^2)^2+(1-x_i)^2)$ 函数,多峰,变量相关,定义域为[-100,100],最优值为 0。

(3)$f_3(x)=20-20\exp\left(-0.2\sqrt{\frac{1}{N}\sum_{i=1}^{N}x_i^2}\right)-\exp\left(\frac{1}{N}\sum_{i=1}^{N}\cos(2\pi x_i)\right)+\exp(1)$ 函数,多峰,变量不相关,定义域为[-32,32],最优值为 0。

(4)$f_4(x)=\frac{1}{4\,000}\sum_{i=1}^{N}x_i^2-\prod_{i=1}^{N}\cos(\frac{x_i}{\sqrt{i}})+1$ 函数,多峰,变量相关,定义域为[-600,600],最优值为 0。

(5)$f_5(x)=\sum_{i=1}^{N}(x_i^2-10\cos(2\pi x_i))+10)$ 函数,多峰,变量不相关,定义域为[-5.12,5.12],最优值为 0。

(6)$f_6(x)=418.982\,9N+\sum_{i=1}^{N}x_i\sin(\sqrt{|x_i|})$ 函数,多峰,变量不相关,定义域为[-500,500],最优值为 0。

(7)$f_7(x_i)=-\cos(2\pi\sqrt{\sum_{i=1}^{N}x_i^2})+0.1\sqrt{\sum_{i=1}^{N}x_i^2}+1$ 函数,多峰,变量不相关,定义域为[-100,100],最优值为 0。

(8)$f_8(x_i)=\frac{\pi}{N}\{10\sin^2(\pi y_1)+\sum_{i=1}^{N-1}(y_i-1)^2[1+10\sin^2(\pi y_{i+1})]+(y_N-1)^2\}+\sum_{i=1}^{N}u(x_i,10,100,4)$,

where $y_i=1+\frac{1}{4}(x_i+1)$, $u(x_i,a,k,m)=\begin{cases}k(x_i-a)^m & x_i>a\\ 0 & -a\leqslant x_i\leqslant -a\\ k(-x_i-a)^m & x_i<a\end{cases}$ 函数,多峰,变量相关,定义域为[-50,50],最优值为 0。

(9)$f_9(x_i)=0.1\{\sin^2(3\pi x_1+\sum_{i=1}^{N-1}(x_i-1)^2[1+\sin^2(3\pi x_{i+1})]+(x_N-1)^2\}+\sum_{i=1}^{N}u(x_i,5,100,4)$ 函数,多峰,变量相关,定义域为[-50,50],最优值为 0。

4.2 实验结果

在实验中,DE 算法、DDE 算法、OXDE 算法、ODDE 算法的缩放因子 F 为 0.9,交叉概率 CR 为 0.9,种群规模 NP 和问题维数 D 设置为 $NP=D$,且分别取值 30,100,300,500 4 种情况(为能更充分的地反映算法性能,仍对 30 维的低维度问题进行测试,以便比较维度的增加对各算法性能的影响),函数评价次数 $FES=D\times 10\,000$。

本文分别对每个问题(30 维、100 维、300 维和

500 维),利用 DE 算法、DDE 算法、OXDE 算法和本文的 ODDE 算法进行 30 次独立实验,记录最优解、最差解、平均值以及标准方差,最优结果以黑体表示。在 Matlab7.0 环境下实现上述 4 种算法,实验所使用的硬件环境为 Intel I5 处理器、4 GB 内存。表 1 的实验结果表明,对于 9 个标准测试问题,ODDE 算法对其中的 8 个优化问题(f_2 除外)所求出的最优解、最差解、均值和方差均优于其他 3 种算法。

对于 f_2 问题,ODDE 和 OXDE 算法能够发现最优解,而 DE 算法、DDE 算法未能找到最优解,ODDE 算法的最差解、均值和方差稍逊于 OXDE 算法。表 2 ~ 表 4 的实验结果表明,对于 9 个高维测试问题(100 维、300 维和 500 维),ODDE 算法求出的最优解、最差解、均值和方差等 4 个方面均优于 DE 算法、DDE 算法、OXDE 算法,而且只有 ODDE 算法能够发现大部分的最优解,其他方法无法找到最优解。

表 1　算法 30 维仿真结果比较

函数	算法	最优解	最差解	均值	方差
f_1	DE	2.39e-020	5.85e-017	9.28e-018	1.48e-017
	DDE	1.18e-020	2.62e-017	2.37e-018	5.21e-018
	OXDE	1.22e-061	7.18e-058	4.00e-059	1.30e-058
	ODDE	**0.00**	**3.31e-322**	**9.88e-324**	**0.00**
f_2	DE	2.60e-003	7.28e+001	2.15e+001	2.25e+001
	DDE	1.76e-001	7.14e+001	2.15e+001	2.47e+001
	OXDE	**0.00**	**3.99e+000**	**3.99e-001**	**1.21e+000**
	ODDE	**0.00**	2.85e+001	1.80e+001	1.39e+001
f_3	DE	2.60e-011	2.52e-009	9.01e-010	7.98e-010
	DDE	5.55e-011	9.83e-010	3.17e-010	2.36e-010
	OXDE	3.55e-015	3.55e-017	3.55e-015	**0.00**
	ODDE	**0.00**	**0.00**	**0.00**	**0.00**
f_4	DE	**0.00**	1.72e-002	3.61e-003	5.17e-003
	DDE	**0.00**	2.71e-002	5.01e-003	7.93e-003
	OXDE	**0.00**	2.46e-002	1.72e-003	5.17e-003
	ODDE	**0.00**	**0.00**	**0.00**	**0.00**
f_5	DE	1.12e+001	4.43e+001	2.51e+001	8.22e+000
	DDE	1.42e+001	4.72e+001	2.50e+001	7.00e+000
	OXDE	2.00e+000	1.69e+001	9.49e+000	3.37e+000
	ODDE	**0.00**	**0.00**	**0.00**	**0.00**
f_6	DE	**0.00**	1.63e+003	5.07e+002	3.27e+002
	DDE	1.18e+002	1.26e+003	6.07e+002	3.05e+002
	OXDE	**0.00**	2.17e+002	7.24e+000	3.96e+001
	ODDE	**0.00**	0.00	**0.00**	**0.00**
f_7	DE	2.00e-001	3.00e-001	2.41e-001	4.93e-002
	DDE	2.00e-001	3.00e-001	2.34e-001	4.59e-002
	OXDE	2.00e-001	3.00e-001	2.03e-001	1.83e-002
	ODDE	**0.00**	**0.00**	**0.00**	**0.00**
f_8	DE	5.71e-022	4.15e-001	2.07e-002	7.89e-002
	DDE	7.78e-023	1.04e-001	1.04e-002	3.16e-002
	OXDE	**1.57e-032**	1.04e-001	3.46e-003	1.89e-002
	ODDE	**1.57e-032**	**1.57e-032**	**1.57e-032**	**1.57e-048**
f_9	DE	8.44e-020	1.60e+000	5.36e-002	2.92e-001
	DDE	2.62e-020	1.10e-002	1.10e-002	3.35e-002
	OXDE	1.84e-032	**1.35e-032**	1.38e-032	1.25e-033
	ODDE	**1.35e-032**	**1.35e-032**	**1.35e-032**	**5.57e-048**

表 2　算法 100 维仿真结果比较

函数	算法	最优解	最差解	均值	方差
f_1	DE	2.22e+003	5.69e+003	3.95e+003	9.62e+002
	DDE	1.60e+003	5.41e+003	3.69e+003	9.91e+002
	OXDE	2.70e-005	1.34e-007	7.29e-008	2.87e-008
	ODDE	**3.57e-132**	**1.69e-124**	**8.86e-126**	**3.13e-125**
f_2	DE	1.20e+008	6.44e+008	3.10e+008	1.24e+008
	DDE	1.20e+008	5.79e+008	2.91e+008	1.14e+008
	OXDE	8.82+001	1.54e+002	9.56e+001	**1.46e+001**
	ODDE	**0.00**	**9.78e+001**	**6.51e+001**	4.68e+001
f_3	DE	7.67e+000	1.05e+001	8.80e+000	7.04e-001
	DDE	7.51e+000	1.10e+001	9.14e+000	8.02e-001
	OXDE	2.90e-005	8.80-001	2.94e-002	1.60e-001
	ODDE	**0.00**	**0.00**	**0.00**	**0.00**
f_4	DE	2.47e+001	7.25e+001	3.86e+001	9.60e+000
	DDE	2.15e+001	5.53e+001	3.65e+001	9.21e+000
	OXDE	1.40e-008	9.54e-008	3.95e-008	1.94e-008
	ODDE	**0.00**	**0.00**	**0.00**	**0.00**
f_5	DE	7.51e+002	1.01e+003	8.56e+002	5.90e+001
	DDE	6.50e+002	9.26e+002	8.42e+002	7.03e+001
	OXDE	2.40e+001	2.28e+002	1.09e+002	5.10e+001
	ODDE	**0.00**	**0.00**	**0.00**	**0.00**
f_6	DE	1.82e+004	2.94e+004	2.51e+004	2.20e+003
	DDE	2.21e+004	3.08e+004	2.61e+004	2.18e+003
	OXDE	6.93e-008	2.17e+002	7.24e+000	3.96e+001
	ODDE	**1.09e-010**	**1.09e-010**	**1.09e-010**	**0.00**
f_7	DE	7.32e+000	1.16e+001	9.59e+000	1.00e+000
	DDE	7.60e+000	1.21e+001	1.00e+001	1.04e+000
	OXDE	4.00e-001	6.00e-001	4.57e-001	6.07e-002
	ODDE	**0.00**	**2.96e-004**	**4.84e-005**	**8.17e-005**
f_8	DE	5.12e+004	3.17e+006	6.12e+005	6.33e+005
	DDE	2.35e+004	2.30e+006	6.66e+005	6.17e+005
	OXDE	3.14e-010	1.56e-001	1.56e-002	4.23e-002
	ODDE	**4.71e-033**	**4.71e-033**	**4.71e-033**	**1.39e-048**
f_9	DE	8.81e+005	9.20e+006	3.42e+006	1.93e+006
	DDE	1.03e+006	1.36e+007	4.35e+006	3.01e+006
	OXDE	2.03e-008	1.10e-002	7.49e-004	2.78e-003
	ODDE	**1.35e-032**	**1.35e-032**	**1.35e-032**	**5.57e-048**

表 3 算法 300 维仿真结果比较

函数	算法	最优解	最差解	均值	方差
f_1	DE	2.59e+005	3.70e+005	3.11e+005	2.72e+004
	DDE	2.76e+005	3.48e+008	3.06e+005	1.81e+004
	OXDE	1.66e+002	2.75e+002	2.25e+002	2.58e+001
	ODDE	**2.71e-051**	**9.92e-042**	**3.90e-043**	**1.82e-042**
f_2	DE	6.59e+010	1.21e+011	9.14e+010	1.14e+010
	DDE	7.40e+010	1.25e+011	9.90e+010	9.19e+009
	OXDE	2.47e+005	6.32e+005	4.13e+005	9.53e+004
	ODDE	**1.07e-006**	**2.89e+002**	**6.31e+001**	**9.00e+000**
f_3	DE	1.89e+001	1.98e+001	1.93e+000	2.56e-001
	DDE	1.85e+001	1.97e+001	1.92e+001	2.69e-001
	OXDE	2.38e+000	2.87e+000	2.65e+000	1.22e-001
	ODDE	**0.00**	**0.00**	**0.00**	**0.00**
f_4	DE	2.19e+003	3.10e+003	2.82e+003	2.06e+002
	DDE	2.35e+003	3.08e+003	2.78e+003	1.81e+002
	OXDE	2.49e+00	3.44e+000	3.02e+000	2.34e-001
	ODDE	**0.00**	**0.00**	**0.00**	**0.00**
f_5	DE	3.67e+003	4.39e+003	3.96e+003	1.55e+002
	DDE	3.74e+003	4.29e+003	3.97e+003	1.29e+002
	OXDE	1.84e+003	2.14e+003	2.03e+000	8.06e+001
	ODDE	**0.00**	**0.00**	**0.00**	**0.00**
f_6	DE	1.04e+005	1.07e+005	1.06e+005	7.50e+002
	DDE	1.04e+005	1.08e+005	1.06e+005	1.07e+003
	OXDE	2.40e+002	5.02e+002	3.50e+000	5.81e+001
	ODDE	**4.80e-010**	**4.80e-010**	**4.80e-010**	**0.00**
f_7	DE	5.23e+001	6.10e+001	5.67e+001	2.11e+000
	DDE	5.26e+001	6.11e+001	5.71e+001	2.24e+000
	OXDE	4.10e+000	5.00e+000	4.46e+000	2.45e-001
	ODDE	**5.12e-005**	**5.70e-003**	**1.38e-003**	**1.50e-003**
f_8	DE	1.00e+009	1.92e+009	1.44e+009	2.35e+008
	DDE	1.15e+009	2.14e+009	1.52e+009	2.38e+008
	OXDE	4.35e-001	1.42e+000	7.49e-001	2.27e-001
	ODDE	**1.57e-033**	**1.57e-033**	**1.57e-033**	**1.04e-048**
f_9	DE	2.18e+009	3.95e+009	3.03e+009	4.91e+008
	DDE	2.23e+009	3.89e+009	3.01e+009	4.22e+008
	OXDE	5.15e+001	1.25e+002	7.77e+001	1.98e+001
	ODDE	**1.35e-032**	**1.35e-032**	**1.35e-032**	**5.57e-048**

表 4 算法 500 维仿真结果比较

函数	算法	最优解	最差解	均值	方差
f_1	DE	6.49e+005	8.21e+005	7.38e+005	4.54e+004
	DDE	6.37e+005	8.48e+005	7.40e+005	4.49e+004
	OXDE	3.95e+003	5.23e+003	4.56e+003	3.26e+002
	ODDE	**4.79e-034**	**1.49e-026**	**7.47e-028**	**2.87e-027**
f_2	DE	2.30e+011	3.43e+011	2.82e+011	2.34e+010
	DDE	1.96e+011	3.27e+011	2.71e+011	2.34e+010
	OXDE	1.52e+007	2.44e+007	1.88e+007	2.65e+006
	ODDE	**1.92e-004**	**4.91e+002**	**9.21e+001**	**1.38e+002**
f_3	DE	1.97e+001	2.03e+001	2.01e+001	1.50e-001
	DDE	1.99e+001	2.04e+001	2.01e+001	1.19e-001
	OXDE	5.10e+000	5.59e+000	5.27e+000	1.17e-001
	ODDE	**0.00**	**3.55e-015**	**1.42e-015**	**1.77e-015**
f_4	DE	5.58e+003	7.58e+003	6.57e+003	4.47e+002
	DDE	5.78e+003	7.15e+003	6.66e+003	3.54e+002
	OXDE	3.61e+001	4.88e+001	4.13e+001	2.77e+000
	ODDE	**0.00**	**0.00**	**0.00**	**0.00**
f_5	DE	6.73e+003	7.63e+003	7.30e+003	2.35e+002
	DDE	6.95e+003	7.77e+003	7.33e+003	2.15e+002
	OXDE	3.87e+003	4.12e+003	4.03e+003	6.26e+001
	ODDE	**0.00**	**0.00**	**0.00**	**0.00**
f_6	DE	1.81e+005	1.86e+005	1.84e+005	8.87e+002
	DDE	1.83e+005	1.86e+005	1.84e+005	7.12e+002
	OXDE	5.70e+003	7.35e+003	6.51e+003	4.69e+002
	ODDE	**8.15e-010**	**8.15e-010**	**8.15e-010**	**0.00**
f_7	DE	8.20e+001	9.15e+001	8.73e+001	2.19e+000
	DDE	8.08e+001	9.25e+001	8.66e+001	2.38e+000
	OXDE	8.43e+000	9.93e+000	9.20e+000	3.44e-001
	ODDE	**1.70e-004**	**9.30e-002**	**1.31e-003**	**1.69e-003**
f_8	DE	3.16e+009	5.79e+009	4.50e+009	5.60e+008
	DDE	2.79e+009	5.43e+009	4.39e+009	6.42e+008
	OXDE	3.82e+000	5.83e+000	4.69e+000	5.97e-001
	ODDE	**9.42e-034**	**9.42e-034**	**9.42e-034**	**1.74e-049**
f_9	DE	6.08e+009	1.09e+010	8.64e+009	1.16e+009
	DDE	5.88e+009	1.02e+010	8.85e+009	8.85e+008
	OXDE	3.06e+002	6.61e+002	4.52e+002	9.09e+001
	ODDE	**1.34e-032**	**1.35e-032**	**1.35e-032**	**5.57e-048**

此外实验结果还表明,DE 算法、DDE 算法和 OXDE 算法的求解性能随着维数(100 维以上)的增加,性能急剧下降,几乎不能提供满意的求解结果。从表 2 ~ 表 4 实验结果还可以看出,尽管 ODDE 算法性能也受到维数增加的影响,但是 ODDE 算法对于9 个500 维的问题,仍然可以发现 3 个问题(f_3,f_4 和 f_5)的最优解,对于其他 6 个问题,可以找到满意的结果。

为进一步说明 ODDE 算法的收敛性能,图 2 ~ 图 10 给出了上述 9 个问题 500 维下的收敛过程(30 次独立求解的平均收敛过程)。从表 1 ~ 表 4 的仿真数据来看,本文实验环境下 DE 算法和 DDE 算法的30 次独立求解结果不存在数量级上的差别,所以这 2 种算法的收敛曲线接近重合。因此,图 2 ~ 图 10 的收敛曲线略去了 DDE 算法的曲线。

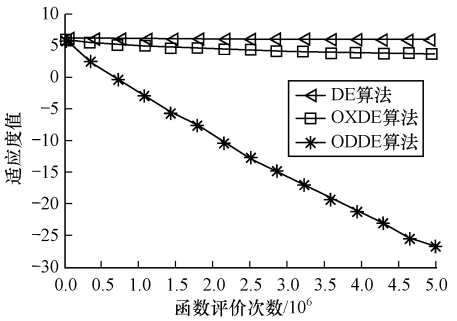

图 2　f_1 收敛图

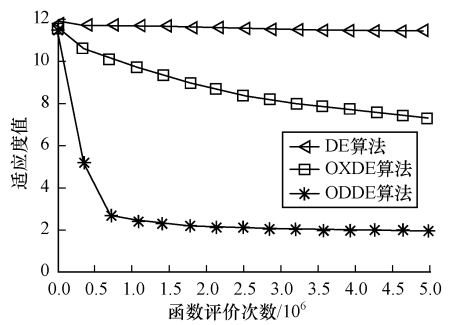

图 3　f_2 收敛图

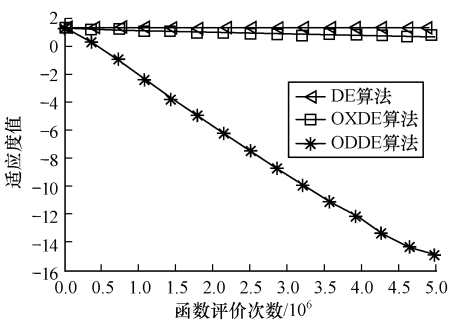

图 4　f_3 收敛图

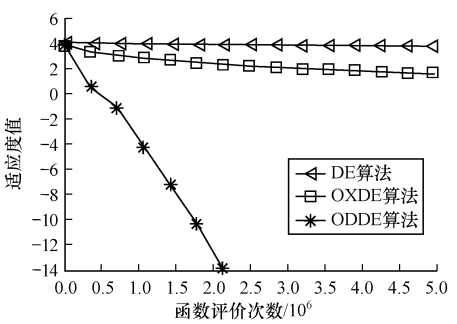

图 5　f_4 收敛图

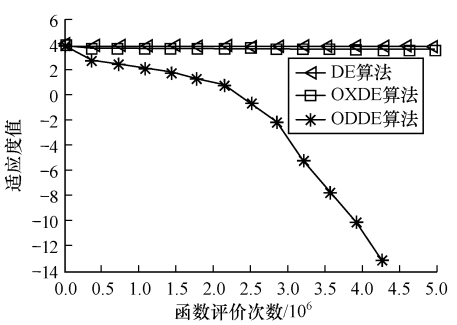

图 6　f_5 收敛图

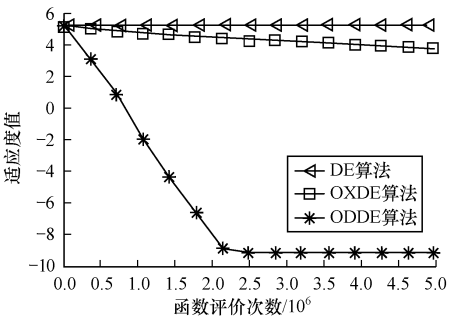

图 7　f_6 收敛图

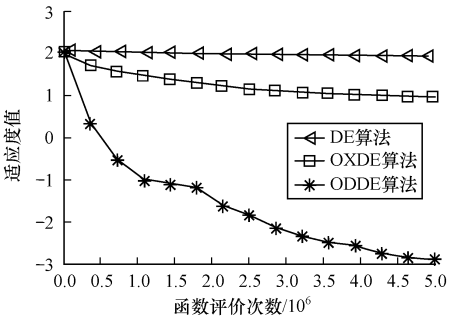

图 8　f_7 收敛图

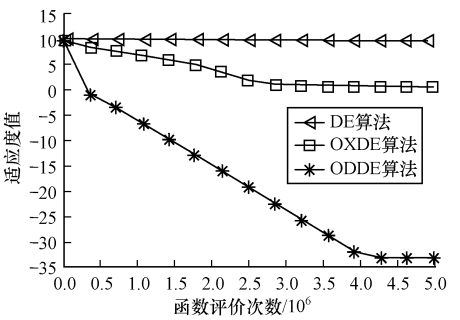

图 9　f_8 收敛图

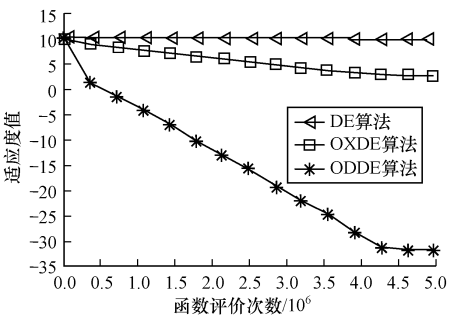

图 10　f_9 收敛图

从 9 个问题求解的收敛图可以看出,求解高维优化问题时,DE 算法在所有 9 个问题上均出现了早熟收敛现象。此外,尽管 OXDE 算法在所有9 个问题上的进化一定程度上均优于DE 算法。然而,在求解 f_1, f_3,f_5 函数时,OXDE 同样会陷入局部最优,出现明显的早熟收敛现象。在其他6 个问题上,虽未陷入局部最优,但是进化后期最优解更新非常缓慢,因此,导致最终的求解结果不理想。ODDE 算法在所有 9 个函

数的进化过程中,收敛速度明显快于 OXDE 算法和标准 DE 算法,而且对于大部分问题都能迅速地收敛,尤其是对 f_4,f_5 函数,能够迅速地收敛于实际最优解 0,优势更明显。图 2 ~ 图 10 的收敛对比图表明,对于高维优化问题,本文提出的 ODDE 算法的求解速度明显优于 DE 算法和 OXDE 算法。

5 结束语

针对求解高维优化问题,本文提出一种高效的正交动态差分进化(ODDE)算法。ODDE 算法利用动态进化策略来提高其全局搜索能力,并使用正交交叉算子提高其局部搜索能力。采用 9 个标准测试问题,分别选择 4 种不同维度对 ODDE 算法的性能进行测试,并与 DE,DDE 和 OXDE 算法进行比较。结果表明,ODDE 算法在求解精度和收敛速度上具有明显优势。今后将把 ODDE 算法引入到协同进化框架下,进一步提升其求解性能。

参考文献

[1] Li Xiaodong, Yao Xin. Cooperatively Conolving Particle Swarm for Large Scale Optimization[J]. IEEE Transactions on Evolutionary Computation,2012,16(2):210-224.

[2] Omidvar M N,Li Xiaodong, Mei Yi, et al. Cooperative Co-evolution with Differential Grouping for Large Scale Optimization[J]. IEEE Transactions on Evolutionary Computation,2014,18(3):378-393.

[3] 彭 硕,欧阳艾嘉,乐光学,等. 求解高维函数的改进萤火虫群优化算法[J]. 计算机应用,2013,33(8):2253-2256,2260.

[4] 李炳宇,萧蕴诗,汪 镭. 一种求解高维复杂函数优化问题的混合粒子群优化算法[J]. 信息与控制,2004,33(1):27-30.

[5] 刘星宝,蔡自兴,王 勇,等. 应用于高维优化问题的免疫进化算法[J]. 控制与决策,2011,26(1):59-64.

[6] 林志毅,王玲玲. 求解高维函数优化问题的混合蜂群算法[J]. 计算机科学,2013,40(3):279-282.

[7] 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.

[8] 谭 跃,谭冠政,涂 立. 一种新的混沌差分进化算法[J]. 计算机工程,2009,35(11):216-217,220.

[9] Noman N, Iba H. Accelerating Differential Evolution Using an Adaptive Local Search[J]. IEEE Transactions on Evolutionary Computation,2008,12(1):107-125.

[10] 刘 洁,吴亮红,刘建勋. 基于单纯形算子的混合差分进化算法[J]. 计算机工程,2009,35(13):179-182.

[11] Qing Anyong. Dynamic Differential Evolution Strategy and Applications in Electromagnetic Inverse Scattering Problems[J]. IEEE Transactions on Geoscience and Remote Sensing,2006,44(1):116-125.

[12] 邓长寿,赵秉岩,梁昌勇. 改进的差异演化算法[J]. 计算机工程,2009,35(24):194-195.

[13] Wang Yong, Cai Zixing, Zhang Qingfu. Enhancing the Search Ability of Differential Evolution Through Orthogonal Crossover[J]. Information Sciences, 2006, 185:153-177.

编辑 陆燕菲

(上接第 16 页)

[4] Yu Peijie, Xia Mingyuan, Li Qian, et al. Real-time Enhancement for Xen Hypervisor[C]//Proceedings of 2010 IEEE/IFIP International Conference on Embedded and Ubiquitous Computing. Washington D. C., USA: IEEE Computer Society,2010:23-30.

[5] Lee Min,Krishnakumar A S,Krishnan P,et al. Supporting Soft Real-time Tasks in the Xen Hypervisor[C]//Proceedings of the 6th ACM SIGPLAN/SIGOPS International Conference on Virtual Execution Environments. New York,USA:ACM Press,2010:97-108.

[6] Kim Hwanju,Lim Hyeontaek,Jeong Jinkyu,et al. Task-aware Virtual Machine Scheduling for I/O Performance[C]//Proceedings of 2009 ACM SIGPLAN/SIGOPS International Conference on Virtual Execution Environments. New York, USA: ACM Press, 2009: 101-110.

[7] 王 凯,侯紫峰. 自适应调整虚拟机权重参数的调度方法[J]. 计算机研究与发展,2011,48(11):2094-2102.

[8] Credit Scheduler[EB/OL]. (2012-05-14). http://wiki.Xenproject.org/wiki/Credit_Scheduler.

[9] 时 光,郭玉东,王晓睿,等. Xen 中的 VCPU 调度算法分析[J]. 计算机工程与设计,2010,31(18):4116-4119.

[10] Barham P, Dragovic B, Fraser K. Xen and the Art of Virtualization[C]//Proceedings of the 9th ACM Symposium on Operating Systems Principles. New York, USA:ACM Press,2003:164-177.

[11] Chisnall D. The Definitive Guide to the Xen Hypervisor[M]. Upper Saddle River, USA: Prentice Hall,2007.

[12] Xen 4.1.2[EB/OL]. (2011-10-21). http://www.Xen-project.org/downloads/Xen-archives/supported-Xen-41-series/Xen-412/21-Xen-412.

[13] Xu C,Gamage S,Rao P N,et al. vSlicer: Latency-aware Virtual Machine Scheduling via Differentiated-frequency CPU Slicing[C]//Proceedings of the 21st ACM Symposium on High-performance Parallel and Distributed Computing. New York,USA:ACM Press,2012:3-14.

[14] RT TEST UTILS[EB/OL]. [2014-03-10]. Http://git.kernel.org/cgit/linux/kernel/git/clkwllms/rt-test.git.

[15] Xi Sisu, Wilson J, Lu Chenyang, et al. RT-Xen: Towards Real-time Hypervisor Scheduling in Xen[C]//Proceedings of 2011 International Conference on Embedded Software. Washington D. C.,USA:IEEE Press,2011:39-48.

[16] Xi Sisu, Xu Meng, Lu Chenyang, et al. Real-time Multi-core Virtual Machine Scheduling in Xen[C]//Proceedings of 2014 International Conference on Embedded Software. Washington D. C.,USA:IEEE Press,2014:1-11.

编辑 陆燕菲