一种基于反向学习的约束差分进化算法

魏文红[1],周建龙[2],陶 铭[1],袁华强[1]
(1. 东莞理工学院计算机学院,广东东莞 523808; 2. 西安交通大学城市学院计算机系,陕西西安,710018)

摘 要: 差分进化算法是一种结构简单、易用且鲁棒性强的全局搜索启发式优化算法,它可以结合约束处理技术来解决约束优化问题.机器学习在进化算法中,经常可以引导种群的进化,而且被广泛地应用于无约束的差分进化算法中,但对于约束差分进化算法却很少有应用.针对这一情况,提出了一种基于反向学习的约束差分进化算法框架.该算法框架采用基于反向学习的机器学习方法,提高约束差分进化算法的多样性和加速全局收敛速度.最后把该算法框架植入了两个著名的约束差分进化算法:$(\mu+\lambda)$-CDE 和 ECHT,并采用 CEC 2010 的 18 个 Benchmark 函数进行了实验评估,实验结果表明:与$(\mu+\lambda)$-CDE 和 ECHT 相比,植入后的算法具有更强的全局搜索能力、更快的收敛速度和更高的收敛精度.

关键词: 反向学习; 差分进化; 约束优化; 收敛性
中图分类号: TP38 **文献标识码:** A **文章编号:** 0372-2112 (2016)02-0426-11
电子学报 URL: http://www.ejournal.org.cn **DOI:** 10.3969/j.issn.0372-2112.2016.02.026

Constrained Differential Evolution Using Opposition-Based Learning

WEI Wen-hong[1], ZHOU Jian-long[2], TAO Ming[1], YUAN Hua-qiang[1]
(1. *School of Computer, Dongguan University of Technology, Dongguan, Guangdong* 523808, *China*;
2. *Department of Computer, Xi'an Jiaotong University City College, Xi'an, Shaanxi* 710018)

Abstract: Differential evolution is a global heuristic algorithm, which is simple, easy-to-use and robust in practice. Combining with the constraint-handling techniques, it can solve constrained optimization problems. Machine learning often guides population to evolve in the evolution computation, and is widely applied to unconstrained differential evolution algorithm. However, machine learning is rarely applied to constrained differential evolution algorithm, so this paper proposed a constrained differential evolution algorithm framework using opposition-based learning. The algorithm can improve the diversity and convergence of differential evolution. At last, the proposed algorithm framework is applied to two popular constrained differential evolution variants, that is $(\mu+\lambda)$-CDE and ECHT-DE. And 18 benchmark functions presented in CEC 2010 are chosen as the test suite, experimental results show that comparing with $(\mu+\lambda)$-CDE and ECHT-DE, our algorithms are able to improve global search ability, convergence speed and accuracy in the majority of test cases.

Key words: opposition-based learning; differential evolution; constrained optimization; convergence

1 引言

优化问题(Optimization Problem, OP)一直都是人工智能领域研究的热点,起初人们一直研究着无约束的优化算法.但实际上许多的科学和工程问题都存在着各种各样的约束条件,这就导致了人们加强了对约束优化问题(Constrained Optimization Problem, COP)的研究[1].在一般的进化算法(Evolution Algorithm, EA)中,约束条件的出现,会导致可行解区域减小并使得搜索解的过程变得更为复杂.虽然约束优化问题复杂,但人们还是成功地在无约束优化算法中加入约束处理机制来解决约束优化问题.

差分进化算法(Differential Evolution, DE)是由 Storn 和 Price 提出的一种功能强大、结构简单、易用且鲁棒性强的全局优化算法[2],该算法已经被成功地用于各种不同的应用领域.加入约束处理机制的差分进化算法可以解决约束优化问题,如 Storn[3] 提出的约束适应性差分进化算法(CADE),它是一种多成员的差分进化算法;Lampinen[4] 提出了一种基于帕雷托占优约束

收稿日期:2015-04-10;修回日期:2015-07-08;责任编辑:马兰英
基金项目:国家自然科学基金(No. 61103037, No. 61300198);广东省自然科学基金(No. S2013010011858);广东省高校科技创新项目(No. 2013KJCX0178);陕西省工业科技攻关项目(No. 2015GY012);陕西省自然科学基础研究计划项目(No. 2015JM6331);西安交通大学城市学院科研项目(No. 2015KZ01, 2015KZ02)

处理技术的差分进化算法;Mezura-Montes 等人[5]提出了一种基于多成员多样性的差分进化算法(MMDE),该算法允许每一个父辈可以产生多个子代;Wang 等人[6]提出了($\mu+\lambda$)-CDE 算法,该算法通过 μ 个种群产生 λ 个子代,然后把($\mu+\lambda$)个种群一起进行迭代;Mallipeddi 等人[7]提出了一种合并多个约束处理技术的 ECHT-DE 算法;在 CEC 2006[8]竞赛中,基于 ε 约束处理技术的 εDE 算法获得了第一名[9]. 大多数的约束差分进化算法虽然在一定程度上解决了约束优化问题,但它们却存在容易过早收敛而陷入局部最优且收敛速度较慢的缺点[10].

机器学习(Machine Learning)在进化算法中起着重要的作用[11],由于进化算法在迭代搜索过程中需要存储搜索空间、问题特征和种群信息等大量的数据,因此机器学习方法可以分析这些数据的特征来提高搜索性能. 即在全局优化过程中,提取有用的信息来理解搜索行为,用以指导下一步的搜索方向. 在许多应用领域,引入机器学习方法的进化算法在收敛速度和问题求解精度方面都有所提高[11].

基于反向学习(Opposition-based Learning,OBL)的机器学习方法是由 Tizhoosh[12]首次提出的,该方法为了获得更优的解来进行下一代迭代,在每次迭代过程中,不仅要评价本次搜索到的最优解,而且还要评价与该最优解处于相反方向的解,然后得出最终的最优解,用来进行下一次迭代. 最近基于反向学习的机器学习方法被广泛用于一些启发式进化算法如差分进化、粒子群、人工神经网络、蚁群和人工蜜蜂算法等[13]. Rahnamayan 等人[14]首次提出了基于反向学习的差分进化算法,在该算法中,采用基于反向学习的机制来初始化种群. 接下来的几年里,包括 Rahnamayan 自己在内的许多专家学者,分别提出了改进的基于反向学习的差分进化算法[15~19]. Ahandani 等人[15]利用基于反向学习策略提出了交换的差分进化算法(SDE);Omran[16]结合混沌搜索、基于反向学习、差分进化和量子机制提出了 CODEQ 算法;Miao 等人[17]提出了一种自适应的基于反向学习的差分进化算法(SAODE);最近 Wang 等人[18]推广了基于反向学习的机制,提出了一种基于推广反向学习的差分进化算法(GODE). 所有这些基于反向学习的差分进化算法都是处理无约束优化问题的,然而对于基于反向学习的约束差分进化算法却少见报道,目前 Omran 提出的 CODEQ 算法可以处理一些简单的约束优化问题[19],然而该算法针对 CEC 2006[8]中的 24 个 Benchmark 函数只测试了 5 个约束函数,不能够说明该算法具有很强的优势. 在这种背景下,本文提出了一种基于反向学习的约束差分进化算法框架(OBL-CDE),并且把 OBL-CDE 植入到两个著名的约束差分进化算法($\mu+\lambda$)-CDE 和 ECHT-DE 中,针对 CEC 2010[20]中的 18 个 Benchmark 函数全部进行了实验测试. 测试结果显示,与($\mu+\lambda$)-CDE 和 ECHT-DE 算法相比,我们的算法具有更强的全局搜索能力、更快的收敛速度和更高的收敛精度.

虽然 OBL-CDE 与 CODEQ 都是基于反向学习的约束差分进化算法,但 OBL-CDE 与 CODEQ 仍然存在本质的区别:(1)CODEQ 是一种具体的算法,OBL-CDE 是一种算法框架. OBL-CDE 可以植入其它的一些具体的约束差分进化算法中,而 CODEQ 却不能;(2)CODEQ 将混沌搜索、基于反向学习、差分进化和量子机制集于一体,使得算法结构变得复杂;而 OBL-CDE 只包含了基于反向学习的机制,并不改变差分进化算法的主体结构,因此算法结构简单,运算速度快;(3)CODEQ 只测试了 CEC 2006 中 24 个 Benchmark 函数的 5 个,显然不能证明该算法具有很强的优势,而 OBL-CDE 测试了 CEC 2010 中全部的 Benchmark 函数,并通过各种实验比较说明了 OBL-CDE 的优势.

2　相关背景知识

2.1　约束优化问题

在约束优化问题中,不但包括等式约束条件和不等式约束条件,还包括向量 $\boldsymbol{x}$ 的上界和下界,具体如下:

$$\min f(\boldsymbol{x}) \tag{1}$$

$$\text{s.t.}\quad g_j(\boldsymbol{x})\leqslant 0, j=1,\cdots,q \tag{2}$$

$$h_j(\boldsymbol{x})=0, j=q+1,\cdots,m \tag{3}$$

$$l_i\leqslant x_i\leqslant u_i, i=1,\cdots,n \tag{4}$$

其中 $\boldsymbol{x}=(x_1,x_2,\cdots,x_n)$ 是一个 n 维决策向量,$f(\boldsymbol{x})$ 为目标函数,$g_j(\boldsymbol{x})\leqslant 0$ 和 $h_j(\boldsymbol{x})=0$ 分别表示 q 个不等式约束和 $m-q$ 个等式约束. 函数 f,g_j 和 h_j 为线性或非线性实数函数. u_i 和 l_i 分别为变量 x_i 的上界和下界. 另外,假定可行解空间中满足所有约束的点集合用 U 表示,搜索空间中满足上界和下界约束的点集合用 S 表示,其中 $S\subset U$.

一般地,在约束进化算法中,等式约束经常需要转化成不等式约束,具体如下:

$$|h_j(\boldsymbol{x})|-\delta\leqslant 0 \tag{5}$$

其中 $j\in\{q+1,\cdots,m\}$,δ 为等式约束违反的容忍因子,一般取正整数,解 $\boldsymbol{x}$ 到第 j 个约束的距离可以表示为:

$$G_j(\boldsymbol{x})=\begin{cases}\max\{0,g_j(\boldsymbol{x})\}, & 1\leqslant j\leqslant q\\ \max\{0,|h_j(\boldsymbol{x})|-\delta\}, & q+1\leqslant j\leqslant m\end{cases} \tag{6}$$

那么解 $\boldsymbol{x}$ 到可行解区域的边界距离,即约束违反程度可以表示为:

$$G(\boldsymbol{x})=\sum_{j=1}^{m}G_j(\boldsymbol{x}) \tag{7}$$

2.2 差分进化算法

差分进化算法是一类比较流行的进化算法,并且具有良好的性能,广泛地应用于各种应用领域. 在差分进化算法中,一般包括 NP 个种群,每个个体向量的维度为 n. 一般地,个体表示为:$\boldsymbol{x}_{ri,t}=(x_{1i,t},x_{2i,t},\cdots,x_{ni,t})$,其中 $i=1,2,\cdots,NP$,NP 为种群大小,n 为个体向量的维度,t 为当前种群的代数. 差分进化算法包括三个主要的操作,分别是:变异,交叉和选择[21].

变异 在变异操作中,对于每个种群产生目标向量 $\boldsymbol{v}_{i,t}$,变异策略主要如下:

$$\text{rand/1}:\boldsymbol{v}_{i,t}=\boldsymbol{x}_{r1,t}+F\cdot(\boldsymbol{x}_{r2,t}-\boldsymbol{x}_{r3,t}) \tag{8}$$

$$\text{best/1}:\boldsymbol{v}_{i,t}=\boldsymbol{x}_{best,t}+F\cdot(\boldsymbol{x}_{r1,t}-\boldsymbol{x}_{r2,t}) \tag{9}$$

$$\text{current-to-best/1}:\boldsymbol{v}_{i,t}=\boldsymbol{x}_{i,t}+F\cdot(\boldsymbol{x}_{best,t}-\boldsymbol{x}_{i,t})+F\cdot(\boldsymbol{x}_{r1,t}-\boldsymbol{x}_{r2,t}) \tag{10}$$

$$\text{best/2}:\boldsymbol{v}_{i,t}=\boldsymbol{x}_{best,t}+F\cdot(\boldsymbol{x}_{r1,t}-\boldsymbol{x}_{r2,t})+F\cdot(\boldsymbol{x}_{r3,t}-\boldsymbol{x}_{r4,t}) \tag{11}$$

$$\text{rand/2}:\boldsymbol{v}_{i,t}=\boldsymbol{x}_{r1,t}+F\cdot(\boldsymbol{x}_{r2,t}-\boldsymbol{x}_{r3,t})+F\cdot(\boldsymbol{x}_{r4,t}-\boldsymbol{x}_{r5,t}) \tag{12}$$

其中下标 $r1$、$r2$、$r3$、$r4$ 和 $r5$ 都是随机从集合 $\{1,2,\cdots,NP\}\backslash\{i\}$ 中选取的,另外 $\boldsymbol{x}_{best,t}$ 为种群在第 t 代最好的个体. 缩放因子 F 为实数,$F\in[0,1]$.

交叉 交叉操作主要产生试验向量 $\boldsymbol{u}_{i,t}=(u_{1i,t},u_{2i,t},\cdots,u_{ni,t})$,具体如下:

$$u_{ji,t}=\begin{cases}\boldsymbol{v}_{ji,t}, & \text{if } rand_j(0,1)\leqslant CR \text{ or } j=j_{rand}\\ \boldsymbol{x}_{ji,t}, & \text{otherwise}\end{cases} \tag{13}$$

其中 $i=1,2,\cdots,NP$,$j=1,2,\cdots,n$,j_{rand} 是属于从 1 到 n 之间的一个随机整数,$rand_j(0,1)$ 为对于每个 j 产生 $[0,1]$ 均匀分布的随机数. 使用参数 j_{rand} 是为了保证试验向量 $\boldsymbol{u}_{i,t}$ 不同于目标向量 $\boldsymbol{x}_{i,t}$. 交叉概率因子 CR 的取值通常在 0 - 1 之间.

选择 在选择操作中,目标向量 $\boldsymbol{x}_{i,t}$ 和试验向量 $\boldsymbol{u}_{i,t}$ 根据它们的适应值进行比较,选择适应值更优的进入下一代种群:

$$\boldsymbol{x}_{i,t+1}=\begin{cases}\boldsymbol{u}_{i,t}, & \text{if } f(\boldsymbol{u}_{i,t})\leqslant f(\boldsymbol{x}_{i,t})\\ \boldsymbol{x}_{i,t}, & \text{othterwise}\end{cases} \tag{14}$$

2.3 基于反向学习

为了更形象地、更清楚地解释基于反向学习的概念,必须先理解反向点的概念. 图 1 显示了区间 $[a,b]$ 之间 x 的反向点 $\bar{x}$ 的例子.

定义 1 假设 $x\in[a,b]$,x 为实数,x 的反向点 $\bar{x}$ 表示为:$\bar{x}=a+b-x$.

如果对于一个 n 维向量的反向点,则用如下定义表示.

定义 2 假设 $\boldsymbol{P}=(x_1,x_2,\cdots,x_n)$ 为一个 n 维向量空间的点,其中 $x_1,x_2,\cdots,x_n\in R$ 且 $x_i\in[a_i,b_i]$,$\forall i\in[1,2,\cdots,n]$. 则 $\boldsymbol{P}$ 的反向点 $\bar{\boldsymbol{P}}=(\bar{x}_1,\bar{x}_2,\cdots,\bar{x}_n)$,其中 $\bar{x}_i=a_i+b_i-x_i$. 有了反向点的定义之后,那么基于反向学习的优化可以定义如下.

定义 3 假设 $\boldsymbol{P}=(x_1,x_2,\cdots,x_n)$ 为一个 n 维向量空间的点(比如 $\boldsymbol{P}$ 是候选解),$f(\cdot)$ 是候选解的目标函数适应值,另外根据反向点的定义可知 $\boldsymbol{P}$ 的反向点为 $\bar{\boldsymbol{P}}=(\bar{x}_1,\bar{x}_2,\cdots,\bar{x}_n)$. 如果 $f(\bar{\boldsymbol{P}})<f(\boldsymbol{P})$,则表示 $\bar{\boldsymbol{P}}$ 比 $\boldsymbol{P}$ 具有更好的适应值,此时选择 $\bar{\boldsymbol{P}}$ 代替 $\boldsymbol{P}$,否则保持 $\boldsymbol{P}$ 不变.

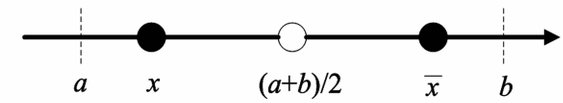


图1 反向点的例子

3 算法框架

本文提出的算法并不是一种新的约束差分进化算法,而是基于反向学习机制的一种通用算法框架以改善约束差分进化算法的搜索性能,特别是搜索求解的精度和收敛速度. 由定义 3 可知,基于反向学习的机制需要比较种群与反向种群的适应值,而约束优化问题的适应值不能简单地由目标函数值决定,因此我们就采用一种适应值变换的方法来处理基于反向学习机制中的适应值比较问题.

3.1 适应值变换

一般情况下,在无约束差分进化算法中,适应值都等于目标函数值,但是在约束差分进化算法中,由于约束的存在,不能简单地考虑适应值等于目标函数值,必须要考虑约束条件的存在. 比如,对于最小值优化问题,解向量 $\boldsymbol{X}$ 的目标函数值比 $\boldsymbol{Y}$ 小,但 $\boldsymbol{X}$ 却违反了约束条件,而 $\boldsymbol{Y}$ 没有违反约束条件. 此时,应该考虑可能解向量 $\boldsymbol{Y}$ 比 $\boldsymbol{X}$ 更优. 为了更好地处理约束差分进化算法中的适应值,采用适应值变换(Adaptive Fitness Transformation,简称 AFT)[22]方法把种群分成三种状态:不可行状态、半可行状态和可行状态.

不可行状态 在不可行状态下,种群只包含了不可行解,因此不用考虑目标函数值,只需要考虑约束违反程度. 约束违反程度可以通过式(7)计算,此时适应值计算公式如下:

$$f_{\text{fitness}}(\boldsymbol{x}_i)=G(\boldsymbol{x}_i) \tag{15}$$

半可行状态 在半可行状态下,种群既包含了部分可行解又包含了部分不可行解,因此就必须在目标函数值和约束违反程度之间找到一个平衡点. 从解的层面分析,此时把种群细分为可行解组(Z_1)和不可行解组(Z_2). 因此解 $\boldsymbol{x}_i$ 的目标函数值 $f'(\boldsymbol{x}_i)$ 就可以转换成:

$f'(\boldsymbol{x}_i)=$

$$\begin{cases} f(\boldsymbol{x}_i), & i \in Z_1 \\ \max\{\varphi \cdot f(\boldsymbol{x}_{\text{best}}) + (1-\varphi) \cdot f(\boldsymbol{x}_{\text{worst}}), f(\boldsymbol{x}_i)\}, & i \in Z_2 \end{cases} \tag{16}$$

其中 φ 是上一代种群的可行解比率,$\boldsymbol{x}_{\text{best}}$ 和 $\boldsymbol{x}_{\text{worst}}$ 分别是可行解组 Z_1 最优和最差的解. 进一步把式(16)归一化为:

$$f_{\text{nor}}(\boldsymbol{x}_i) = \frac{f'(\boldsymbol{x}_i) - \min\limits_{j \in Z_1 \cup Z_2} f'(\boldsymbol{x}_j)}{\max\limits_{j \in Z_1 \cup Z_2} f'(\boldsymbol{x}_j) - \min\limits_{j \in Z_1 \cup Z_2} f'(\boldsymbol{x}_j)} \tag{17}$$

式(7)可以计算约束违反程度,在此状态下,式(7)归一化为:

$$G_{\text{nor}}(\boldsymbol{x}_i) = \begin{cases} 0, & i \in Z_1 \\ \dfrac{G(\boldsymbol{x}_i) - \min\limits_{j \in Z_2} G(\boldsymbol{x}_j)}{\max\limits_{j \in Z_2} G(\boldsymbol{x}_j) - \min\limits_{j \in Z_2} G(\boldsymbol{x}_j)}, & i \in Z_2 \end{cases} \tag{18}$$

因此最终的适应值可以表示为:

$$f_{\text{fitness}}(\boldsymbol{x}_i) = f_{\text{nor}}(\boldsymbol{x}_i) + G_{\text{nor}}(\boldsymbol{x}_i) \tag{19}$$

可行状态　在可行状态下,种群中所有个体都是可行解,因此就可以看作是无约束优化问题来处理. 此时适应值就等于目标函数值 $f(\boldsymbol{x}_i)$,具体如下:

$$f_{\text{fitness}}(\boldsymbol{x}_i) = f(\boldsymbol{x}_i) \tag{20}$$

3.2　算法描述

在进化算法中,基于反向学习的机制基本上都是用于种群初始化和代跳跃阶段. 在种群初始化阶段,对于随机生成的初始种群 P_0(其中种群大小为 NP,个体向量的维度为 n),通过公式 $OP_{ji,0} = a_j + b_j - P_{ji,0}$ 计算出相应的反向种群 OP_0,式中 $i = 1,2,\cdots,NP$,$j = 1,2,\cdots,n$. 然后再从种群 P_0 和 OP_0 中选出适应值更优的个体组成新的初始种群 P_0. 在种群代跳跃阶段,与初始化阶段不同的是,首先,该阶段的执行必须满足一个代跳跃概率 J_r,在绝大部分情况下 $J_r = 0.3$[14]. 其次,反向种群的计算公式为 $OP_{ji,t} = MIN(P_{ji,t}) + MAX(P_{ji,t}) - P_{ji,t}$,其中 $MIN(P_{ji,t})$ 和 $MAX(P_{ji,t})$ 分别表示第 t 代种群中最差和最优的个体. OBL-CDE 算法伪代码如下:

算法 1　OBL-CDE 伪代码

```
t = 0;  //t 表示种群代数
产生初始种群 P0 = (x1, x2, ..., xNP);
for (i = 0; i < NP; i++)
    for (j = 0; j < n; j++)
        OPji,0 = aj + bj - Pji,0;
P0 = Select_Fitbest_Individual(P0, OP0);//从种群 P0 和反向种群 OP0 中选取最优的个体
repeat
    t = t + 1;
    Pt = Pt-1;
    for 种群 Pt 中的每个个体 do
        执行变异、交叉和选择策略获得新种群 Pt';
    end
    Pt = Pt'
    if (rand(0,1) < Jr)  //Jr 是代跳跃概率
        for (i = 0; i < NP; i++)
            for (j = 0; j < n; j++)
                OPji,t = MIN(Pji,t) + MAX(Pji,t) - Pji,t;
    end
    Pt = Select_Fitbest_Individual(Pt, OPt);//从种群 Pt 和反向种群 OPt 中选取最优的个体
until 停止条件满足;
output:种群 Pt 的个体
```

算法 1 中的函数 Select_Fitbest_Individual(P,OP) 表示从种群 P 和反向种群 OP 中选取最优的个体组成新的种群,函数 Select_Fitbest_Individual(P,OP) 详细描述如下:

算法 2　Select_Fitbest_Individual(P,OP)

```
计算种群 P 中每个个体的目标函数值和约束违反程度;
φ1 = num_1 / NP;//num_1 表示种群 P 中可行解个数,0 ≤ num_1 ≤ NP.
根据 φ1 判断种群 P 的位置并通过 AFT 方法求出其变换后的适应值 Fitp;
//种群 P 的位置是指种群属于不可行状态、半可行状态或可行状态中的某一种状态
计算种群 OP 中每个个体的目标函数值和约束违反程度;
φ2 = num_2 / NP;  //num_2 表示种群 OP 中可行解个数,0 ≤ num_2 ≤ NP.
根据 φ2 判断种群 OP 的位置并通过 AFT 方法求出其变换后的适应值 Fitop;
if (Fitop < Fitp)
    return (OP);
else
    return (P);
end
```

从算法描述中可以看出,OBL-CDE 算法简单且容易操作,它可以植入到所有的约束差分进化算法中. 另外,OBL-CDE 的时间复杂度为 $O(G_{\max} \cdot NP \cdot n)$,其中 $G_{\max}$ 表示迭代的最大代数. 如果植入到其它约束差分进化中,并不会增加它们的时间复杂度.

3.3　OBL-CDE 算法移植讨论

根据 OBL-CDE 算法的描述和伪代码,可以看出 OBL-CDE 算法核心有两部分:(1)在种群初始阶化段采用 OBL 机制求出该初始种群的反向种群,然后使用适应值变换方法从原始种群和其反向种群中选择最优的种群作为新的初始种群;(2)在子种群的生存选择之后,采用代跳跃方法使当前的子种群跳入到新的候选解中以加速收敛速度. 也就是说,求出当前子种群的反向种群,然后也使用适应值变换方法从当前子种群和其反向种群中选

择最优的种群作为新的子种群. 由于所有的 CDE 算法都有种群初始化和生存选择两部分,所以我们只要把 OBL-CDE 算法中求初始反向种群和代跳跃两部分的代码植入到其它 CDE 算法中种群初始化和生存选择的位置便可完成植入过程. 由于只需要修改两个位置的代码就可完成 OBL-CDE 框架到其它 CDE 算法的植入,所以该植入过程简单可靠,而且通用性强.

4 实验测试

我们对$(\mu+\lambda)$-CDE 和 OBL-$(\mu+\lambda)$-CDE、ECHT-DE 和 OBL-ECHT-DE 四个算法选取 CEC 2010 中 18 个 Benchmark 函数的 10 维(10D)测试函数和 30 维(30D)测试函数分别进行了对比实验测试. 实验环境为 64 位的 Windows 7 系统,其中 CPU 为 Intel Core TM 2.83GHz,内存为 4GB. 我们没有对 50 维(50D)或更高维的测试函数进行实验,是因为目前的进化算法求解效果都很差,难以分出胜负. 与此同时,我们也没有选取 CEC 2006 中的 Benchmark 函数做实验测试,因为目前绝大多数的约束差分进化算法对于 CEC 2006 中的 24 个 Benchmark 函数都能求解到最优解. 另外,由于目前大多数的约束差分进化算法都是针对 CEC 2006 进行实验测试的,而 OBL-$(\mu+\lambda)$-CDE 和 OBL-ECHT-DE 算法对于 CEC 2006 中的 24 个 Benchmark 函数也同样能求解到最优解,因此无法区分出优劣. 所以我们的算法只和 CEC 2010 竞赛中冠军算法 εDEag[23] 进行了对比测试.

4.1 实验参数设置

为了比较公平,OBL-$(\mu+\lambda)$-CDE 与$(\mu+\lambda)$-CDE 设置相同的参数,具体如下[6]:(1)种群大小:$\mu=70$ ($NP=\mu$),$\lambda=210$;(2)交叉概率:$CR=0.9$;(3)缩放因子:$F=0.8$;(4)代跳跃概率:$J_r=0.3$.

其它参数的设置也与文献[6]相同,根据文献[6],变异策略概率 $p_m=0.05$,用户自定义参数 $g=200$,阈值代因子 $k=0.6$ 和容忍因子 $\delta=0.0001$.

与前面类似,OBL-ECHT-DE 与 ECHT-DE 设置相同的参数[7]:(1)种群大小:$\mu=50$($NP=\mu$);(2)交叉概率:$CR=0.7$;(3)缩放因子:$F=0.9$;(4)代跳跃概率:$J_r=0.3$.

4.2 性能评价指标

(1)可行解比率:在算法所有次数的运行过程中,成功找到可行解的个数占所有种群个数的比率平均值. 该可行解比率的最大值等于 1,最小值等于 0. 值越大,表示算法能成功找到可行解的性能越好.

(2)收敛图:用来表示算法在求解过程中,向最优解靠近的一种渐近走向. 该走向的趋势下降的越快,表示收敛性越好.

(3)Wilcoxon 符号秩检验[24]:该方法是在成对观测数据的符号检验基础上发展起来的,比传统的单独用正负号的检验更加有效,是非参数的统计测试方法. 例如,A 算法和 B 算法进行 Wilcoxon 符号秩检验测试,如果在测试结果中,A 算法对于 B 算法的 R^+ 比 R^- 的值要大,则表明 A 算法优于 B 算法.

(4)Friedman 测试[25]:与 Wilcoxon 符号秩检验不同的是,Friedman 测试可以针对三个及以上算法进行 Rank 测试,Rank 值越小,表明算法性能越好.

4.3 实验结果分析

根据 CEC 2010 的要求,实验中的每一个算法对于每一个测试函数都运行 50 次,在每一次运行过程中,对于 10D 测试函数设置 Max_FEs $=2\times10^5$,对于 30D 测试函数设置 Max_FEs $=6\times10^5$.

(1)OBL-$(\mu+\lambda)$-CDE 与$(\mu+\lambda)$-CDE 比较

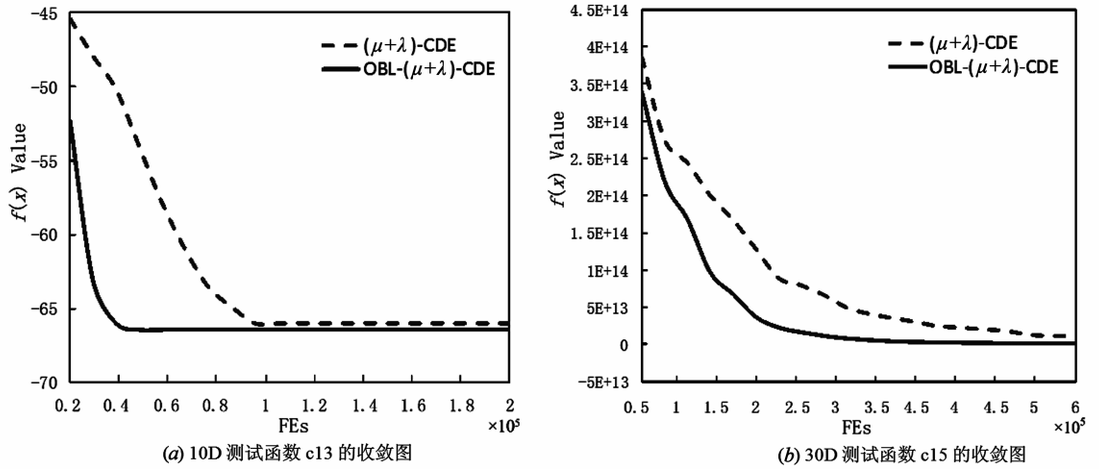
我们对 OBL-$(\mu+\lambda)$-CDE 和$(\mu+\lambda)$-CDE 算法针对 CEC 2010 的 Benchmark 函数进行了实验测试,结果显示在表 1 中,其中粗体表示该算法的值更优. 从表中的结果来看,在 10D 测试函数方面,对于平均值,OBL-$(\mu+\lambda)$-CDE 比$(\mu+\lambda)$-CDE 胜出 16 个函数,有 2 个函数处于劣势;对于可行解比率,OBL-$(\mu+\lambda)$-CDE 比$(\mu+\lambda)$-CDE 也是胜出 16 个函数,有 2 个函数处于劣势. 而在 30D 测试函数方面,对于平均值,OBL-$(\mu+\lambda)$-CDE 比$(\mu+\lambda)$-CDE 胜出 17 个函数,只有 1 个函数处于劣势;对于可行解比率,OBL-$(\mu+\lambda)$-CDE 比$(\mu+\lambda)$-CDE 胜出 15 个函数,有 3 个函数相等. 比较结果说明了对于高维测试函数,OBL-$(\mu+\lambda)$-CDE 更有优势. 另外,我们对 10D 和 30D 测试函数各随机选取了一个测试函数进行了收敛性测试,并画出了它们的收敛图,如图 2 所示. 其中图 2(a)为 10D 测试函数 c13 的收敛图,图 2(b)为 30D 测试函数 c15 的收敛图. 从图 2 中很明显可以看出,无论是 10D 测试函数还是 30D 测试函数,OBL-$(\mu+\lambda)$-CDE 的收敛性都好于$(\mu+\lambda)$-CDE.

(2)OBL-ECHT-CDE 与 ECHT-CDE 比较

我们也对 OBL-ECHT-CDE 和 ECHT-CDE 算法针对 CEC 2010 的 Benchmark 函数进行了实验测试,表 2 显示了比较结果,其中粗体表示该算法的值更优. 从表 2 中可以看到,对于 10D 测试函数,OBL-ECHT-DE 在平均值方面比 ECHT-DE 胜出 14 个函数,有 4 个函数处于劣势;可行解比率方面的情况与平均值差不多,OBL-ECHT-DE 比 ECHT-DE 胜出 13 个函数,有 1 个函数相等,4 个函数处于劣势. 而对于 30D 测试函数,在平均值方面,OBL-ECHT-DE 比 ECHT-DE 胜出 16 个函数,只有 2 个函数处于劣势;在可行解比率方面,OBL-ECHT-DE 比 ECHT-DE 胜出 14 个函数,有 3 个函数相等,只有 1 个函数处于劣势. 比较结果也说明了对于高维测试函数,OBL-ECHT-DE 优势更明显. 图3给出了 OBL-ECHT-CDE

表 1　OBL-$(\mu+\lambda)$-CDE 与$(\mu+\lambda)$-CDE 实验结果比较

prob	Criteria	10D		30D		prob	Criteria	10D		30D	
		OBL-$(\mu+\lambda)$-CDE	$(\mu+\lambda)$-CDE	OBL-$(\mu+\lambda)$-CDE	$(\mu+\lambda)$-CDE			OBL-$(\mu+\lambda)$-CDE	$(\mu+\lambda)$-CDE	OBL-$(\mu+\lambda)$-CDE	$(\mu+\lambda)$-CDE
c01	Mean	−7.4549E-01	**−8.2188E-01**	−8.0601E-01	**−8.1716E-01**	c10	Mean	**1.8657E+12**	3.7288E+13	**2.3529E+13**	3.7677E+13
	Std	3.3623E-03	**0.0000E+00**	1.0616E-02	**6.0624E-03**		Std	**3.2634E+12**	1.2615E+13	**1.2606E+13**	1.0865E+13
	FeasiPro	0.9689	**0.9714**	0.9874	0.9874		FeasiPro	**0.3006**	0.0000	**0.1463**	0.0000
c02	Mean	**−2.0603E+00**	3.1200E+00	**−1.4729E+00**	2.1445E+00	c11	Mean	**−1.5227E-03**	2.6786E-03	**1.1329E-03**	1.9212E-03
	Std	**5.3876E-01**	1.9965E+00	**9.7917E-01**	2.1061E+00		Std	**8.8922E-12**	7.1236E-03	**5.2784E-03**	6.3129E-03
	FeasiPro	**0.7029**	0.0177	**0.4194**	0.0126		FeasiPro	**0.9783**	0.5331	**0.8463**	0.5680
c03	Mean	**6.5781E+13**	8.1744E+13	**2.0076E+13**	2.6255E+14	c12	Mean	**−1.9925E-01**	6.3430E-01	**−1.3901E-01**	4.1880E-01
	Std	**1.6908E+14**	3.1943E+14	**3.3944E+13**	5.8661E+14		Std	**2.7056E-11**	1.8388E+00	**1.7707E-01**	1.5654E+00
	FeasiPro	**0.8011**	0.3314	**0.3229**	0.2971		FeasiPro	**0.7143**	0.6674	**0.7497**	0.7451
c04	Mean	**−1.0000E-05**	4.0771E+00	**5.0883E+00**	7.1406E+00	c13	Mean	**−6.6434E+01**	−6.3255E+01	**−6.4085E+01**	−6.3599E+01
	Std	**0.0000E+00**	8.4561E+00	**8.6441E+00**	1.0681E+01		Std	**2.3596E+00**	1.3345E+00	**2.4785E+00**	1.8567E+00
	FeasiPro	**1.0000**	0.4057	**0.3920**	0.3851		FeasiPro	**0.9966**	0.9863	**0.9914**	0.9874
c05	Mean	**−2.7939E+02**	5.1851E+02	**2.9259E+02**	5.1470E+02	c14	Mean	**0.0000E+00**	2.1426E+11	**6.5084E+00**	3.2504E+11
	Std	**3.0228E+02**	7.6027E+01	**2.4929E+02**	6.4649E+01		Std	**0.0000E+00**	2.2387E+11	**3.9895E+00**	3.9037E+11
	FeasiPro	**0.7120**	*0.000	**0.2549**	0.0000		FeasiPro	**1.0000**	0.9440	**1.0000**	0.9354
c06	Mean	**−3.2562E+02**	4.6348E+02	**−2.0828E+02**	4.8184E+02	c15	Mean	**2.6571E+12**	8.7592E+12	**9.7949E+11**	1.0699E+13
	Std	**2.7853E+02**	1.5593E+02	**1.6534E+02**	1.2534E+02		Std	**5.5500E+12**	1.0604E+13	**1.2258E+12**	1.7855E+13
	FeasiPro	**0.6480**	0.0063	**0.6371**	0.0097		FeasiPro	**0.2120**	0.0714	**0.1069**	0.0749
c07	Mean	**0.0000E+00**	4.9249E+00	**3.2509E-09**	5.3901E+00	c16	Mean	4.9360E-02	**3.4785E-02**	**7.7145E-03**	3.6296E-02
	Std	**0.0000E+00**	2.3972E+00	**1.3221E-08**	2.6630E+00		Std	6.3299E-02	**1.2154E-01**	**2.5502E-02**	1.1131E-01
	FeasiPro	1.0000	1.0000	1.0000	1.0000		FeasiPro	**0.9983**	0.3931	**0.9943**	0.2623
c08	Mean	**9.4073E+00**	2.3955E+01	**2.2745E+01**	2.7651E+01	c17	Mean	**0.0000E+00**	1.3957E+03	**7.5525E-01**	1.4164E+03
	Std	**3.5493E+00**	1.6544E+01	**6.9742E+01**	3.1823E+01		Std	**0.0000E+00**	7.4757E+02	**8.7731E-01**	6.2554E+02
	FeasiPro	0.9486	**1.0000**	1.0000	1.0000		FeasiPro	**1.0000**	0.0149	**0.6949**	0.0269
c09	Mean	**1.9465E+12**	3.6946E+13	**2.2492E+13**	3.6413E+13	c18	Mean	**0.0000E+00**	1.1682E+04	**1.8427E+00**	1.6337E+04
	Std	**3.6767E+12**	1.4113E+13	**1.7902E+13**	1.0236E+13		Std	**0.0000E+00**	1.5125E+04	**8.0174E+00**	1.6477E+04
	FeasiPro	**0.3029**	0.0000	**0.2251**	0.0006		FeasiPro	**1.0000**	0.2526	**0.9286**	0.1440

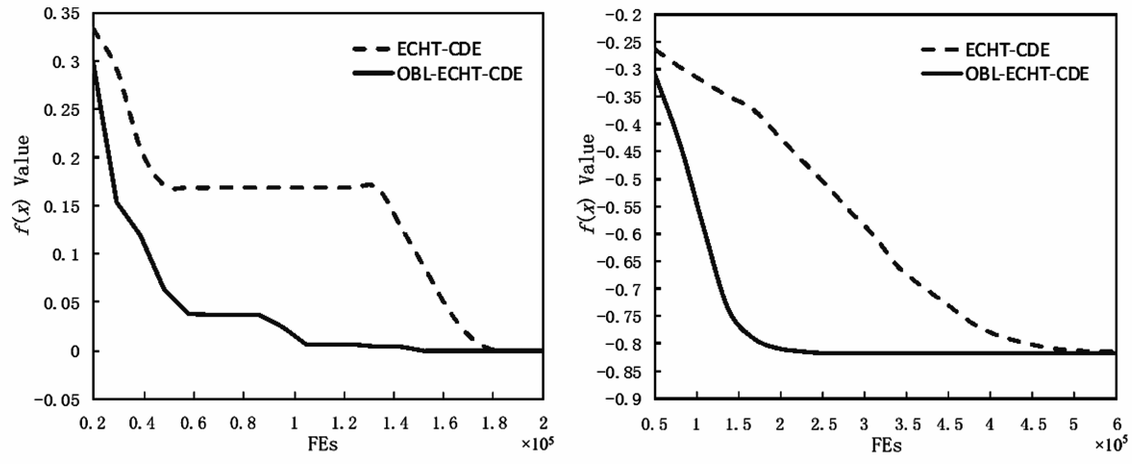


(*a*) 10D 测试函数 c13 的收敛图　　(*b*) 30D 测试函数 c15 的收敛图

图 2 OBL-$(\mu+\lambda)$-CDE 和$(\mu+\lambda)$-CDE 收敛性比较

表 2 OBL-ECHT-DE 与 ECHT-DE 实验结果比较

prob	Criteria	10D		30D		prob	Criteria	10D		30D	
		OBL-ECHT-CDE	ECHT-CDE	OBL-ECHT-CDE	ECHT-CDE			OBL-ECHT-CDE	ECHT-CDE	OBL-ECHT-CDE	ECHT-CDE
c01	Mean	−7.4667E-01	**−7.4730E-01**	**−8.1755E-01**	−8.1441E-01	c10	Mean	**3.3381E+00**	4.1726E+01	**2.1063E+13**	3.0964E+13
	Std	1.8642E-03	**5.0677E-13**	**1.8349E-02**	9.4937E-03		Std	**1.1553E+01**	6.0910E-11	**1.0350E+13**	1.1331E+13
	FeasiPro	0.9177	**0.9643**	0.9299	0.9725		FeasiPro	0.6873	0.5010	0.0003	0.0000
c02	Mean	**−2.0121E+00**	−3.5760E-01	**3.6940E+00**	4.5511E+00	c11	Mean	2.9098E-03	**−1.5000E-03**	6.7341E-02	2.2626E-03
	Std	**2.1277E-01**	9.4850E-01	**1.0642E+00**	6.4532E-01		Std	2.7908E-01	**5.5479E-09**	1.0801E-01	2.4085E-03
	FeasiPro	**0.1471**	0.0241	**0.0104**	0.0074		FeasiPro	0.0398	**0.2339**	0.0000	0.0000
c03	Mean	7.1004E-01	**4.0551E-27**	**3.1395E+09**	5.1552E+11	c12	Mean	**−4.2928E+02**	−2.8368E+01	**−1.1003E+02**	−1.0523E+01
	Std	2.4575E+00	**2.0276E-26**	**1.2163E+10**	1.9406E+12		Std	**1.9700E+02**	4.5084E+01	**9.2827E+01**	4.7256E+01
	FeasiPro	**0.6940**	0.6215	**0.2045**	0.0012		FeasiPro	0.0000	**0.2625**	**0.2081**	0.1790
c04	Mean	**−1.0000E-05**	−9.7277E-06	**1.3011E-02**	9.9666E-02	c13	Mean	**−6.7726E+01**	−6.7570E+01	**−6.4214E+01**	−6.2403E+01
	Std	**1.2769E-14**	5.9495E-08	**1.3262E-01**	7.5941E-02		Std	**1.6849E+00**	3.0313E+00	**1.5026E+00**	6.9631E+00
	FeasiPro	**0.4956**	0.4881	**0.0098**	0.0000		FeasiPro	**0.8528**	0.7454	**0.8594**	0.7149
c05	Mean	**3.4745E+02**	3.6185E+02	**4.8124E+02**	4.8434E+02	c14	Mean	**0.0000E+00**	6.5516E+03	**1.6347E-01**	3.8755E+11
	Std	**1.3317E+02**	1.6429E+02	**7.1626E+01**	7.3198E+01		Std	**0.0000E+00**	3.2561E+04	**7.9655E-01**	4.0016E+11
	FeasiPro	0.0000	0.0000	**0.0054**	0.0000		FeasiPro	**0.8347**	0.7393	**0.8304**	0.7258
c06	Mean	**−3.8191E+02**	2.4417E+02	**5.2111E+02**	5.2665E+02	c15	Mean	**7.8144E+12**	1.8015E+13	**1.2812E+14**	1.2610E+14
	Std	**2.8059E+02**	2.2873E+02	**6.8990E+01**	6.5901E+01		Std	**1.2005E+13**	1.4057E+13	**3.1777E+13**	4.5022E+13
	FeasiPro	**0.2970**	0.0000	0.0000	0.0000		FeasiPro	**0.1968**	0.1539	**0.3261**	0.3243
c07	Mean	**0.0000E+00**	5.1381E-28	**1.8493E-27**	2.8303E-03	c16	Mean	**6.2476E-01**	1.0392E+00	**1.1257E+00**	1.1492E+00
	Std	**0.0000E+00**	1.0210E-27	**4.6483E-27**	3.5297E-03		Std	**4.9141E-01**	5.3800E-02	**3.6539E-02**	4.0019E-02
	FeasiPro	**0.9992**	0.9990	**0.9990**	0.9985		FeasiPro	**0.2156**	0.0182	0.0000	0.0000
c08	Mean	7.8829E+00	**7.0858E+00**	**1.4362E+00**	4.7799E+00	c17	Mean	**0.0000E+00**	5.2983E+01	**1.2171E+03**	1.4008E+03
	Std	4.8694E+00	**4.8747E+00**	**3.5377E+01**	1.9173E+00		Std	**0.0000E+00**	9.4550E+01	**3.7730E+02**	4.7815E+02
	FeasiPro	0.9701	0.9894	**0.9955**	0.9953		FeasiPro	**0.7125**	0.0366	**0.0129**	0.0052
c09	Mean	**0.0000E+00**	9.8500E-28	**2.2037E+13**	2.8925E+13	c18	Mean	**1.2326E-34**	7.3820E+00	**7.2932E+03**	2.4402E+04
	Std	**0.0000E+00**	4.6169E-27	**1.4021E+13**	9.4248E+12		Std	**6.1630E-34**	2.0877E+01	**4.9927E+03**	5.1869E+03
	FeasiPro	**0.7185**	0.6188	**0.0005**	0.0000		FeasiPro	**0.5469**	0.3137	**0.0455**	0.0371



(*a*) 10D 测试函数 c16 的收敛图 (*b*) 30D 测试函数 c01 的收敛图

图 3 OBL-ECHT-CDE 与 ECHT-CDE 收敛性比较

表 3　OBL-(μ+λ)-CDE、OBL-ECHT-DE 与其 εDEag 算法比较结果

prob	algorithms	10D			30D		
		Best	Mean	Worst	Best	Mean	Worst
c01	OBL-(μ+λ)-CDE	**−7.4731E-01**	−7.4549E-01	**−7.3804E-01**	−8.1993E-01	−8.0601E-01	**−7.8550E-01**
	OBL-ECHT-DE	**−7.4731E-01**	**−7.4667E-01**	**−7.3804E-01**	**−8.2188E-01**	**−8.1755E-01**	−7.3737E-01
	εDEag	**−7.4731E-01**	**−7.4704E-01**	**−7.4056E-01**	**−8.2183E-01**	**−8.2087E-01**	**−8.1955E-01**
c02	OBL-(μ+λ)-CDE	**−2.2777E+00**	−2.0603E+00	2.5001E-01	**−2.0940E+00**	**−1.4729E+00**	**3.0516E+00**
	OBL-ECHT-DE	**−2.2777E+00**	**−2.0121E+00**	**−1.3943E+00**	9.6126E-01	3.6940E+00	5.1008E+00
	εDEag	**−2.2777E+00**	**−2.2695E+00**	**−2.1745E+00**	**−2.1692E+00**	**−2.1514E+00**	**−2.1171E+00**
c03	OBL-(μ+λ)-CDE	**0.0000E+00**	6.5781E+13	6.5562E+14	4.9621E+09	2.0076E+13	1.5531E+14
	OBL-ECHT-DE	**0.0000E+00**	**7.1004E-01**	**8.8756E+00**	**1.7052E+03**	**3.1395E+09**	**6.0561E+10**
	εDEag	**0.0000E+00**	**0.0000E+00**	**0.0000E+00**	**2.8673E+01**	**2.8838E+01**	**3.2780E+01**
c04	OBL-(μ+λ)-CDE	**−1.0000E-05**	**−1.0000E-05**	**−1.0000E-05**	9.9232E-03	5.0883E+00	2.5638E+01
	OBL-ECHT-DE	**−1.0000E-05**	**−1.0000E-05**	**−1.0000E-05**	**1.8869E-03**	**1.3011E-02**	**4.7953E-01**
	εDEag	−9.9923E-06	−9.9185E-06	−9.2823E-06	**4.6981E-03**	**8.1630E-03**	**1.7779E-02**
c05	OBL-(μ+λ)-CDE	**−4.8361E+02**	**−2.7939E+02**	**4.9621E+02**	**−1.6819E+02**	**2.9259E+02**	**5.7003E+02**
	OBL-ECHT-DE	4.7059E+01	3.4745E+02	5.8583E+02	2.9232E+02	4.8124E+02	5.9065E+02
	εDEag	**−4.8361E+02**	**−4.8361E+02**	**−4.8361E+02**	**−4.5313E+02**	**−4.4955E+02**	**−4.4216E+02**
c06	OBL-(μ+λ)-CDE	**−5.7866E+02**	−3.2562E+02	**2.1126E+02**	**−5.0017E+02**	**−2.0828E+02**	**2.4000E+02**
	OBL-ECHT-DE	**−5.7866E+02**	**−3.8191E+02**	5.9796E+02	3.2227E+02	5.2111E+02	5.9923E+02
	εDEag	**−5.7866E+02**	**−5.7865E+02**	**−5.7864E+02**	**−5.2858E+02**	**−5.2791E+02**	**−5.2645E+02**
c07	OBL-(μ+λ)-CDE	**0.0000E+00**	**0.0000E+00**	**0.0000E+00**	**5.7873E-19**	3.2509E-09	6.6399E-08
	OBL-ECHT-DE	**0.0000E+00**	**0.0000E+00**	**0.0000E+00**	**0.0000E+00**	**1.8493E-27**	**1.4405E-26**
	εDEag	**0.0000E+00**	**0.0000E+00**	**0.0000E+00**	1.1471E-15	**2.6036E-15**	**5.4819E-15**
c08	OBL-(μ+λ)-CDE	**0.0000E+00**	9.4073E+00	**1.0942E+01**	4.3852E-08	2.2745E+01	3.3024E+02
	OBL-ECHT-DE	**0.0000E+00**	**7.8829E+00**	1.5375E+01	**3.2158E-26**	**1.4362E+00**	**1.1792E+02**
	εDEag	**0.0000E+00**	**6.7275E+00**	**1.5375E+01**	**2.5187E-14**	**7.8315E-14**	**2.5781E-13**
c09	OBL-(μ+λ)-CDE	1.5252E+08	1.9465E+12	1.2220E+13	**9.0025E+11**	2.2492E+13	**5.7278E+13**
	OBL-ECHT-DE	**0.0000E+00**	**0.0000E+00**	**0.0000E+00**	2.6582E+12	**2.2037E+13**	5.9667E+13
	εDEag	**0.0000E+00**	**0.0000E+00**	**0.0000E+00**	**2.7707E-16**	**1.0721E+01**	**1.0528E+02**

与 ECHT-CDE 的收敛图,从图中可以看出,无论是 10D 测试函数还是 30D 测试函数,OBL-ECHT-CDE 的收敛性都明显优于 ECHT-CDE.

(3)OBL-(μ+λ)-CDE、OBL-ECHT-DE 与 εDEag 算法比较

εDEag 算法是 CEC 2010 竞赛中的冠军算法,即在目前求解 CEC 2010 的 18 个 Benchmark 函数的差分进化算法中,εDEag 算法是最优的. 在表 3 和表 4 中,具有灰色底纹加粗的数据表示最优,仅仅加粗的数据表示次优.

从 OBL-(μ+λ)-CDE 与 εDEag 的比较可以看出,对于 10D 函数,在最优值方面,OBL-(μ+λ)-CDE 胜出 3 个函数,有 11 个函数相等,只有 4 个函数处于劣势;在平均值方面,OBL-(μ+λ)-CDE 胜出 4 个函数,有 3 个函数相等,有 11 个函数处于劣势;在最差值方面,基本情况与平均值差不多,只是多胜出 1 个函数,即共胜出 5 个函数. 对于 30D 函数,在最优值方面,OBL-(μ+λ)-CDE 胜出 5 个函数,有 1 个函数相等,有 13 个函数接近或略差于 εDEag 算法;在平均值方面,OBL-(μ+λ)-CDE 也胜出了 3 个函数,有 15 个函数接近或略差于 εDEag 算法;在最差值方面,与平均值的情况相同. 从 OBL-ECHT-CDE 与 εDEag 的比较可以看出,对于 10D 函数,在最优值方面,OBL-ECHT-CDE 胜出 4 个函数,有 11 个函数相等,只有 3 个函数处于劣势;在平均值方面,OBL-ECHT-CDE 胜出 4 个函数,有 3 个函数相

等,其它函数是接近或略差于 εDEag 算法,在最差值方面,与平均值的情况相同;对于 30D 函数,在最优值方面,OBL-ECHT-CDE 胜出 6 个函数,有 1 个函数相等,其它函数是接近或略差于 εDEag 算法. 在平均值方面,OBL-ECHT-CDE 胜出 2 个函数,其它函数是接近或略差于 εDEag 算法,在最差值方面,与平均值的情况相同.

表 4 OBL-($\mu+\lambda$)-CDE、OBL-ECHT-DE 与其 εDEag 算法比较结果(续)

prob	algorithms	10D			30D		
		Best	Mean	Worst	Best	Mean	Worst
c10	OBL-($\mu+\lambda$)-CDE	1.1745E+09	1.8657E+12	1.0320E+13	**1.4145E+12**	2.3529E+13	**4.0329E+13**
	OBL-ECHT-DE	**0.0000E+00**	**3.3381E+00**	**4.1726E+01**	2.2367E+12	**2.1063E+13**	4.0857E+13
	εDEag	**0.0000E+00**	**0.0000E+00**	**0.0000E+00**	**3.2520E+01**	**3.3262E+01**	**3.4632E+01**
c11	OBL-($\mu+\lambda$)-CDE	**-1.5227E-03**	**-1.5227E-03**	**-1.5227E-03**	**-3.9233E-04**	**1.1329E-03**	**1.8671E-02**
	OBL-ECHT-DE	**-2.9263E-01**	2.9098E-03	1.2555E+00	**-4.7432E-02**	6.7341E-02	3.1802E-01
	εDEag	**-1.5227E-03**	**-1.5227E-03**	**-1.5227E-03**	-3.2685E-04	**-2.8639E-04**	**-2.2363E-04**
c12	OBL-($\mu+\lambda$)-CDE	-1.9925E-01	-1.9925E-01	**-1.9925E-01**	**-1.9926E-01**	**-1.3901E-01**	**6.1557E-01**
	OBL-ECHT-DE	**-6.3752E+02**	**-4.2928E+02**	**-1.9925E-01**	**-3.1912E+02**	**-1.1003E+02**	**-1.0365E+01**
	εDEag	**-5.7009E+02**	**-3.3673E+02**	-1.9891E-01	-1.9915E-01	3.5623E+02	5.4617E+02
c13	OBL-($\mu+\lambda$)-CDE	**-6.8429E+01**	-6.6434E+01	**-6.3518E+01**	-6.5334E+01	-6.4085E+01	-5.9783E+01
	OBL-ECHT-DE	**-6.8429E+01**	**-6.7726E+01**	**-6.3518E+01**	**-6.7232E+01**	**-6.4214E+01**	**-6.1166E+01**
	εDEag	**-6.8429E+01**	**-6.8429E+01**	**-6.8429E+01**	**-6.6425E+01**	**-6.5353E+01**	**-6.4297E+01**
c14	OBL-($\mu+\lambda$)-CDE	**0.0000E+00**	**0.0000E+00**	**0.0000E+00**	3.6824E-06	6.5084E+00	1.4970E+01
	OBL-ECHT-DE	**0.0000E+00**	**0.0000E+00**	**0.0000E+00**	**1.4691E-11**	**1.6347E-01**	**3.9866E+00**
	εDEag	**0.0000E+00**	**0.0000E+00**	**0.0000E+00**	**5.0159E-14**	**3.0894E-13**	**2.9235E-12**
c15	OBL-($\mu+\lambda$)-CDE	**3.6732E+00**	**2.6571E+12**	**1.9350E+13**	**3.6042E+10**	**9.7949E+11**	**5.1809E+12**
	OBL-ECHT-DE	**3.6732E+00**	7.8144E+12	5.2415E+13	6.5681E+13	1.2812E+14	1.8238E+14
	εDEag	**0.0000E+00**	**1.7990E-01**	**4.4974E+00**	**2.1603E+01**	**2.1604E+01**	**2.1604E+01**
c16	OBL-($\mu+\lambda$)-CDE	**0.0000E+00**	**4.9360E-02**	**2.3426E-01**	**0.0000E+00**	**7.7145E-03**	**1.2744E-01**
	OBL-ECHT-DE	**0.0000E+00**	6.2476E-01	1.1270E+00	1.0530E+00	1.1257E+00	1.1916E+00
	εDEag	**0.0000E+00**	**3.7021E-01**	**1.0183E+00**	**0.0000E+00**	**2.1684E-21**	**5.4210E-20**
c17	OBL-($\mu+\lambda$)-CDE	**0.0000E+00**	**0.0000E+00**	**0.0000E+00**	**1.1766E-08**	**7.5525E-01**	**3.3691E+00**
	OBL-ECHT-DE	**0.0000E+00**	**0.0000E+00**	**0.0000E+00**	4.0436E+02	1.2171E+03	2.4955E+03
	εDEag	1.4632E-17	1.2496E-01	7.3018E-01	**2.1657E-01**	**6.3265E+00**	**1.8891E+01**
c18	OBL-($\mu+\lambda$)-CDE	**0.0000E+00**	**0.0000E+00**	**0.0000E+00**	**6.0964E-21**	**1.8427E+00**	**4.0163E+01**
	OBL-ECHT-DE	**0.0000E+00**	**1.2326E-34**	**3.0815E-33**	4.2997E+02	7.2932E+03	1.8349E+04
	εDEag	3.7314E-20	9.6788E-19	9.2270E-18	**1.2261E+00**	**8.7546E+01**	**7.3754E+02**

我们对于 OBL-($\mu+\lambda$)-CDE、($\mu+\lambda$)-CDE、OBL-ECHT-DE、ECHT-DE 四个算法进行了 Friedman 测试,测试结果如表 5 所示. 从测试数据很明显可以看出,无论是对于 10D 测试函数还是 30D 测试函数,OBL-($\mu+\lambda$)-CDE 和 OBL-ECHT-DE 的排名都分别排在($\mu+\lambda$)-CDE 和 ECHT-DE 的前面. 另外还可以看出,对于 10D 测试函数,OBL-($\mu+\lambda$)-DE 稍优于 ECHT-DE;而对于 30D 测试函数,OBL-($\mu+\lambda$)-DE 则明显优于 ECHT-DE.

最后我们对于 OBL-($\mu+\lambda$)-CDE、($\mu+\lambda$)-CDE、OBL-ECHT-DE、ECHT-DE 和 εDEag 算法进行了 Wilcoxon 符号秩检验测试,表 6 是这个几个算法的 Wilcoxon 符号秩检验测试结果,从结果中可以知道,OBL-($\mu+\lambda$)-CDE 和 OBL-ECHT-DE 分别明显优于 ($\mu+\lambda$)-CDE 和 ECHT-DE,稍差于 εDEag,但 OBL-ECHT-DE 在 10D 函数的最优值方面要优于 εDEag.

通过与 εDEag 算法的比较,我们发现 OBL-($\mu+$

λ)-CDE 和 OBL-ECHT-DE 算法并没有很大的优势,甚至稍弱于 εDEag 算法. 这是因为我们提出的是一种算法框架,而不是某一个具体的算法,因此在与 εDEag 算法比较时,能否超越 εDEag 算法,还要取决于 OBL-CDE 植入的母体算法(如($\mu+\lambda$)-CDE、ECHT-DE 等). 当然,我们将来的工作,可以考虑将 OBL-CDE 植入到 εDEag 算法中,以证明我们所提出的算法框架的优越性.

表 5　四个算法针对 CEC 2010 的 Friedman 测试结果

Algorithms	Ranking	
	10D	30D
($\mu+\lambda$)-CDE	3.5278 (4)	3.1944 (4)
OBL-($\mu+\lambda$)-CDE	**2.1667** (**2**)	**1.8611** (**1**)
ECHT-DE	2.5 (3)	2.9722 (3)
OBL-ECHT-DE	**1.8056** (**1**)	**1.9722** (**2**)

表 6　Wilcoxon 符号秩检验结果

Algorithms	Dim.	Criteria	R^+	R^-
OBL-($\mu+\lambda$)-CDE-to-$\mu+\lambda$)-CDE	10D	Best Fitness	165	6
		Mean Fitness	166	5
		Worst Fitness	169	2
	30D	Best Fitness	138	33
		Mean Fitness	169	2
		Worst Fitness	125	28
OBL-ECHT-DE-to-ECHT-DE	10D	Best Fitness	134	19
		Mean Fitness	134	19
		Worst Fitness	129.5	41.5
	30D	Best Fitness	119	34
		Mean Fitness	151	20
		Worst Fitness	124.5	46.5
OBL-($\mu+\lambda$)-CDE-to-εDEag	10D	Best Fitness	51	102
		Mean Fitness	29.5	141.5
		Worst Fitness	38.5	114.5
	30D	Best Fitness	30	123
		Mean Fitness	32	139
		Worst Fitness	31	140
OBL-ECHT-DE-to-εDEag	10D	Best Fitness	93.5	59.5
		Mean Fitness	39	132
		Worst Fitness	30.5	122.5
	30D	Best Fitness	27	126
		Mean Fitness	11	160
		Worst Fitness	11	160

5　结束语

在过去的几年中,加入约束处理技术的差分进化算法可以用于解决约束优化问题,但大数的约束差分进化算法容易陷入局部最优. 基于反向学习的机制是机器学习中一种常用的方法,它用于约束差分进化算法中,通过设置反向种群引导算法跳出局部最优的限制. 本文提出了一种基于反向学习的约束差分进化算法框架,它采用基于反向学习的机制引导种群进化,以提高求解精度和收敛速度. 最后我们把该算法框架应用于($\mu+\lambda$) - CDE 和 ECHT-DE 算法,并针对 CEC 2010 中的 18 个 Benchmark 函数进行了实验测试,实验结果表明,我们的算法植入($\mu+\lambda$)-CDE 和 ECHT-DE 后,对($\mu+\lambda$)-CDE 和 ECHT-DE 算法都有相当明显的改进. 将来的工作就是如何把我们的算法框架植入到更多的约束差分进化算法或其它具有约束的进化算法中,以改进它们的性能.

参考文献

[1] Runarsson T P, Yao X. Search biases in constrained evolutionary optimization[J]. IEEE Transactions on Systems, Man, and Cybernetics-Part C: Applications and Reviews, 2005, 35(2): 233 - 243.

[2] 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.

[3] Storn R. System design by constraint adaptation and differential evolution[J]. IEEE Transactions on Evolutionary Computation, 1999, 3(1): 22 - 34.

[4] Lampinen J. A constraint handling approach for the differential evolution algorithm[A]. IEEE Congress on Evolutionary Computation[C]. Honolulu, HI: IEEE Computational Intelligence Society, 2002. 1468 - 1473.

[5] Mezura-Montes E, Velazquez-Reyes J, Coello Coello C A. Promising infeasibility and multiple offspring incorporated to differentialevolution for constrained optimization[A]. Proceedings of the Conference on Genetic and Evolutionary Computation[C]. Washington DC: ACM Press, 2005. 225 - 232.

[6] Wang Y, Cai Z. Constrained evolutionary optimization by means of ($\mu+\lambda$)-differential evolution and improved adaptive trade-off model[J]. Evolutionary Computation, 2011, 19(2): 249 - 285.

[7] Mallipeddi R, Ponnuthurai P, Suganthan N. Ensemble of constraint handling techniques[J]. IEEE Transactions on Evolutionary Computation, 2010, 14(4): 561 - 579.

[8] Liang J J, et al. Problem definitions and evaluation criteria for the CEC 2006 special session on constrained realparameter optimization[R]. Singapore: Nanyang Technological University, Technical Report, 2006. 1 - 24.

[9] Takahama T, Sakai S. Constrained optimization by the ε constrained differential evolution with gradient-based mutation and feasible elites[A]. IEEE Congress on Evolutionary Computation[C]. Vancouver, BC: IEEE Computational Intelligence Society, 2006. 1 - 8.

[10] 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.
[11] Zhang J,Zhan Z,Lin Y,et al. Evolutionary computation meets machine learning:A survey[J]. IEEE Computational Intelligence Magazine,2011,6(4):68 - 75.
[12] Tizhoosh H R. Opposition-based learning:A new scheme for machine intelligence[A]. Proceedings of the International Conferenceon Computational Intelligence for Modelling,Control and Automation,and International Conference on Intelligent Agents,Web Technologies and Internet Commerce[C]. Vienna:IEEE Computer Society,2005. 695 - 701.
[13] Xu Q Z,Wang L,Wang N,et al. A review of opposition-based learning from 2005 to 2012[J]. Engineering Applications of Artificial Intelligence,2014,29(1):1 - 12.
[14] Rahnamayan S,Tizhoosh H R,Salama M M A. Opposition-based differential evolution algorithms[A]. IEEE Congress on Evolutionary Computation[C]. Vancouver,BC:IEEE Computational Intelligence Society,2006. 2010 - 2017.
[15] Ahandani M A,Alavi-Rad H. Opposition-based learning in the shuffled differential evolution algorithm[J]. Soft Computing,2012,16(8):1303 - 1337.
[16] Omran M G H. CODEQ:An effective metaheuristic for continuous global optimization[J]. International Journal of Metaheuristics,2010,1(2):108 - 131.
[17] Miao X F,Mu D J,Han X W,et al. A hybrid differential evolution for numerical optimization[A]. International Conference on Biomedical Engineering and Informatics[C]. Tianjin:IEEE Engineering in Medicine and Biology Society,2009. 1 - 5.
[18] Wang H,Rahnamayan S,Wu Z J. Parallel differential evolution with self adapting control parameters and generalized opposition-based learning for solving high-dimensional optimization problems[J]. Journal of Parallel and Distributed Computing,2013,73(1):62 - 73.
[19] Omran M G H,Salman A. Constrained optimization using CODEQ[J]. Chaos,Solitons Fractals,2009,42(2):662 - 668.
[20] Mallipeddi R,Suganthan P N. Problem definitions and evaluation criteria for the CEC 2010 competition on constrained real-parameter optimization[R]. Singapore:Nanyang Technological University,Technical Report,2010. 1 - 16.
[21] 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.
[22] Gong W,Cai Z,Liang D. Engineering optimization by means of an improved constrained differential evolution [J]. Computer Methods in Applied Mechanics and Engineering,2014,268(1):884 - 904.
[23] Takahama T,Sakai S. Constrained Optimization by the ε constrained differential evolution with an archive and gradient-based mutation[A]. IEEE Congress on Evolutionary Computation[C]. Barcelona:IEEE Computational Intelligence Society,2010. 1 - 9.
[24] Derrac J,Garc』a 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 and Evolutionary Computation,2011,1(1):3 - 18.
[25] Corder G,Foreman D. Nonparametric Statistics for Non-Statisticians:A Step-by-Step Approach[M]. Hoboken,NJ:John Wiley,2009. 38 - 56.

作者简介

魏文红 男,1977 年 9 月生于江西南昌,东莞理工学院副教授,博士,主要研究方向:高性能计算、进化算法、多目标优化处理.
E-mail:weiwh@dgut.edu.cn

周建龙 男,1974 年 6 月出生于甘肃临洮,西安交通大学城市学院特聘教授,博士,研究兴趣包括:人机交互、体视化、增强现实、认知计算以及机器学习.
E-mail:zhou_jianlong@hotmail.com

陶 铭 男,1986 年 6 月生于安徽马鞍山,东莞理工学院副研究员,博士,主要研究方向:移动 IP 技术.
E-mail:taom@dgut.edu.cn

袁华强(通信作者) 男,1966 年 12 月生于湖南衡东,东莞理工学院教授,博士,主要研究方向:智能计算.
E-mail:hyuan66@163.com