doi:10.3969/j.issn.1001-4616.2019.04.001

基于新约束集成的差分进化算法

孙越泓[1,2],王　丹[1]

(1.南京师范大学数学科学学院,江苏 南京 210023)
(2.江苏省大规模复杂系统数值模拟重点实验室,江苏 南京 210023)

[**摘要**]　提出基于新约束集成的差分进化算法用于求解带约束的优化问题. 在产生新个体的阶段,算法采用 3 种不同的突变策略. 利用不同的约束处理技术对新个体进行选择,并通过引入局部搜索,增强算法局部寻优能力,避免算法陷入局部最优. 该算法在 CEC 2017 的 28 个基准函数上进行数值实验,并且与其他较为先进的算法进行比较,实验结果显示,新算法在求解精度上表现较好.

[**关键词**]　约束优化,差分进化算法,约束处理技术集成
[**中图分类号**]TP391;TN911.7　[**文献标志码**]A　[**文章编号**]1001-4616(2019)04-0001-11

Differential Evolutionary Algorithm Based on New Ensemble of Constraint Handing Techniques

Sun Yuehong[1,2],Wang Dan[1]

(1. School of Mathematical Sciences,Nanjing Normal University,Nanjing 210023,China)
(2.Jiangsu Key Laboratory for Numerical Simulation of Large Scale Complex Systems,Nanjing 210023,China)

Abstract:In this paper,a differential evolutionary algorithm based on new ensemble of constraint handing techniques is proposed to solve optimization problems with constraints. At the stage of generating new individuals,the algorithm adopts three different mutation strategies. Different constraint handing techniques are used to select new individuals,and local search is introduced to enhance the local optimization ability and avoid the algorithm falling into local optimum. Numerical experiments are carried out on 28 benchmark functions from CEC 2017 and compared with other advanced algorithms. The results show that the new algorithm performs better in solution accuracy.

Key words:constrained optimization,differential evolutionary algorithm,ensemble of constraint handling techniques

优化问题涉及面广,伴随着人类社会和各个行业的发展. 尤其在科学和工程领域,许多的优化问题都涉及到约束,如 0-1 背包问题、路径规划问题和超大集成电路设计(VLSI)问题等. 约束的存在使问题的可行域减少,最优解的搜索过程复杂化. 因此,求解带约束的优化问题是一个较为复杂的课题,它吸引了较多研究者的关注.

对于约束优化问题有两大类解决方法:一类是传统的优化方法,如增广拉格朗日法、可行方向法和牛顿法等. 这类方法仅能求出优化问题的局部最优解,求解的结果通常依赖于初始值. 另一类是智能优化算法,是模拟自然选择、变异和遗传进化过程的搜索算法. 常见的智能优化算法有遗传算法(genetic algorithm,GA)、差分进化算法(differential evolution,DE)、人工蜂群算法(artificial bee colony,ABC)、进化策略(evolution strategy,ES)、进化规划(evolutionary programming,EP)和遗传规划(genetic programming,GP)等. 这些算法是将种群中的每个个体看做优化问题中的一个解,在种群的更新迭代中逐步找到最优解.

在求解约束优化问题时,一个主要困难是如何在搜索过程中处理不可行的个体. 最早的一种处理方式是完全无视不可行的个体,只在可行的个体中搜索最优解. 忽略不可行个体,不可行的个体中存在的潜在有用

收稿日期:2019-05-12.
基金项目:国家自然科学基金(11871279、61971234)、教育部人文社会科学青年基金(12YJCZH179)、江苏省教育厅高校自然科学研究重大项目(16KJA110001).
通讯联系人:孙越泓,博士,副教授,研究方向:智能优化研究. E-mail:05234@njnu.edu.cn

信息可能会得不到有效利用,从而导致搜索最优解的过程变得缓慢. 因此,许多学者采用不同的技术来利用不可行的个体. 1996 年 Michalewicz 和 Schoenauer[1]将进化算法中处理约束的方法分为 4 类:保留个体的可行性[2]、罚函数、在可行的和不可行的个体之间进行分离以及混合方法. 近年流行的约束处理技术包括随机排名(stochastic ranking,SR)[3],惩罚函数(self-adaptive penalty,SP)[4],ε-约束方法(ε-constraint,EC)[5],特殊算子,多目标约束处理技术[6]和约束集成(ensemble of constraint handling techniques,ECHT)[7]等.

一般来说,约束处理技术的性能取决于它利用进化过程中产生的不可行个体信息的有效性. 然而,根据"无免费午餐"(NFL)定理[8],没有一种最先进的约束处理技术能在每一个问题上胜过其他所有技术. 不同的约束处理方法可以在不同的问题和搜索过程的不同阶段发挥不同作用. 因此,解决特定的约束优化问题需要进行大量的试错运行,以选择合适的约束处理技术并对相关参数进行微调. 在目标函数计算昂贵或实时需要解决方案的应用问题中,试错方法是费时且不能满足实时性的要求.

为了优化算法的性能,有不少专家学者对约束集成作研究,主要是指约束的集成方式和集成所用的策略. 2009 年,Mallipeddi 和 Suganthan 等[7]提出基于约束集成的差分进化算法(ECHDE),通过对 4 个种群分别使用 4 种不同的约束处理技术来提高优化性能. 算法在每一次迭代中都使用可行解的优越性(superiority of feasible solution,SF)[9],自适应罚函数,随机排名,ε-约束这 4 种约束处理技术对解进行选择. 约束集成有利于函数值较小且约束违背值较小的不可行解被选择. 2017 年,Trivedi[10]等人提出新的集成方式,在算法的前期使用罚函数方法,保持种群的多样性;在算法后期使用可行解的优越性使得算法快速趋于收敛.

ECHTDE[7]算法是 Suganthan 等人在 2010 年提出的关于约束集成的 DE 算法,本文在 ECHDE 算法的基础上,提出一种基于新约束集成的差分进化算法(NECHDE). 首先,ECHTDE 算法在产生子代的过程中,采用的是基本差分进化算法中的随机产生子代的方法;NECHTDE 则使用 3 种突变策略产生突变向量,增加了种群的多样性. 其次,ECHTDE 算法处理约束时采用 4 种约束处理技术对 4 个子种群进行处理,而 NECHTDE 是依次选择不同的约束处理技术对个体进行选择,提高了算法的收敛速度. 最后,NECHTDE 算法还采用了局部搜索,将种群中最差的个体用最好的个体替换,有利于提高算法的求解精度.

本文内容安排如下:第 1 节介绍预备知识,包括约束优化问题和常见约束处理技术;第 2 节给出基于新约束集成的差分进化算法的基本思想和算法流程;第 3 节是数值实验与实验结果分析比较;第 4 节对本文工作进行总结和展望.

1 预备知识

1.1 约束优化问题

最优化问题按照自变量是否有约束条件,分为无约束优化问题和约束优化问题. 约束优化问题[11]的一般形式是:

$$\begin{aligned} &\min f(\boldsymbol{x})\\ &\text{s.t. } g_j(\boldsymbol{x})\leqslant 0, j=1,2,\cdots,q\\ &\quad h_j(\boldsymbol{x})=0, j=q+1,\cdots,m \end{aligned} \tag{1}$$

式中,$\boldsymbol{x}$ 表示一个 D 维的解向量,$f(\boldsymbol{x})$是目标函数. $g_j(\boldsymbol{x})$是不等式约束,$h_j(\boldsymbol{x})$是等式约束. q 是不等式约束的个数,$m-q$ 是等式约束的个数. 目标函数和约束可以是线性的,也可以是非线性的. 每个分量由 $l_i\leqslant x_i\leqslant u_i(i=1,2,\cdots,D)$来控制,由此构成了搜索空间 $S\subseteq R^D$. F 是由所有满足约束的解构成的可行域. 满足所有约束的解是可行解,至少有一个约束不满足的解是不可行解. 不可行解的质量不仅依赖于其目标函数值,还有约束违背值. 定义不可行解的平均约束违背值为:

$$CV(\boldsymbol{x})=\frac{\sum_{l=1}^{q}\max(0,g_l(\boldsymbol{x}))+\sum_{l=1}^{q}|H_l(\boldsymbol{x})|}{m}, \tag{2}$$

式中

$$H(\boldsymbol{x})=\begin{cases}|h_j(\boldsymbol{x})|, & \text{if } |h_j(\boldsymbol{x})|-\delta>0,\\ 0, & \text{otherwise},\end{cases} \tag{3}$$

参数 δ 是一个容许值,是对(1)中的等式约束作了松弛.

1.2 常见约束处理技术

求解带约束优化问题时,根据迭代过程中可行解的数量,将可行全局最优解的搜索过程分为 3 个阶段[12]:无可行解;至少一个可行解;都是可行解. 在这 3 个阶段中,不同的约束处理技术执行的方式各不相同. 下面介绍另外几种常用的约束处理技术.

1.2.1 可行解的优越性

用可行解的优越性(SF)[9]比较 $\boldsymbol{x}$ 和 $\boldsymbol{y}$ 两个解,$\boldsymbol{x}$ 比 $\boldsymbol{y}$ 好需满足以下 3 个条件之一:

(1)$\boldsymbol{x}$ 是可行解而 $\boldsymbol{y}$ 是不可行解;

(2)$\boldsymbol{x}$ 和 $\boldsymbol{y}$ 都是可行解,$\boldsymbol{x}$ 的目标函数值比 $\boldsymbol{y}$ 的目标函数值要小(在最小化问题中);

(3)$\boldsymbol{x}$ 和 $\boldsymbol{y}$ 都是不可行解,$\boldsymbol{x}$ 的约束违背值比 $\boldsymbol{y}$ 的约束违背值小.

在 SF 中,可行解总是被认为比不可行解更好. 对两个不可行解的选择是基于约束违背值,而对两个可行解的选择是基于目标函数值. 通过比较不可行解的约束违背值的情况,保留约束违背值小的个体,并将其推入可行域. 因此,在搜索过程中的第一阶段,选择约束违背值较小的不可行解;在第二阶段,首先选择所有可行解,其次,选择约束违背值较小的不可行解;在第三阶段,选择目标函数值最小的可行解.

1.2.2 自适应罚函数

针对约束优化问题,Tessema 等[4]提出了一种自适应罚函数法(SP). 对每个不可行个体添加两种类型的惩罚,以确定当前种群中最不可行的个体. 增加的惩罚值由目前在种群中可行个体的个数来控制. 如果存在少数可行个体,则对约束违反值较大的不可行个体加较高的惩罚. 另一方面,如果存在多个可行个体,那么对具有较高适应度值的不可行个体,在它们的适应度值上加较小的惩罚. 这两个惩罚使得算法在搜索过程中可以随时在寻找更可行的解和寻找最优解之间进行切换. 该算法不需要参数调优,个体最终的适应度值为

$$F(\boldsymbol{x})=d(\boldsymbol{x})+p(\boldsymbol{x}), \tag{4}$$

式中,$d(\boldsymbol{x})$为距离函数,$p(\boldsymbol{x})$为惩罚函数. 距离函数计算公式如下

$$d(\boldsymbol{x})=\begin{cases} CV(\boldsymbol{x}) & \text{if } r_f=0, \\ \sqrt{f''(\boldsymbol{x})^2+CV(\boldsymbol{x})^2} & \text{otherwise}, \end{cases} \tag{5}$$

$$f''(\boldsymbol{x})=(f(\boldsymbol{x})-f_{\min})/(f_{\max}-f_{\min}). \tag{6}$$

式中,可行率 r_f=可行解的个数/NP,NP 是种群中个体的数. $f_{\min}$ 和 $f_{\max}$ 是当前种群中目标函数值的最小值和最大值. 罚函数计算公式如下

$$p(\boldsymbol{x})=(1-r_f)M(\boldsymbol{x})+r_f N(\boldsymbol{x}), \tag{7}$$

$$M(\boldsymbol{x})=\begin{cases} 0 & \text{if } r_f=0 \\ CV(\boldsymbol{x}) & \text{otherwise} \end{cases}, \tag{8}$$

$$N(\boldsymbol{x})=\begin{cases} 0 & \text{if } \boldsymbol{x} \text{ is feasible} \\ f''(\boldsymbol{x}) & \text{otherwise} \end{cases}. \tag{9}$$

1.2.3 随机排名

Runarsson 和 Yao[3]提出随机排名(SR)方法,实现目标函数值和约束违背值的随机平衡. 根据概率因子 p_f 判断是用目标函数值还是约束违背值决定每个解的排名. 如果 $r_f=1$ 或 $rand<p_f$,只按照目标函数值排名;否则只按照约束违背值排名. 其中,p_f 的值不是常量,文献中它的取值是从初始的 $p_f=0.475$,一直线性递减到最后一次迭代 $p_f=0.025$.

1.2.4 ε-约束

ε-约束(EC)是 2006 年 Takahama 等[5]提出的通过参数 ε 控制的一种约束松弛方法. 当约束优化问题含等式约束且存在积极约束条件时,取适当的控制参数 ε 是获得高质量解的关键. 如式(10)和(11)所示,参数 ε 由迭代次数 k 控制. 当 k 到达控制迭代次数 T_c 时,ε 就停止更新. 当 k 超过 T_c 时,将 ε 设置为 0,以获得可行解.

$$\varepsilon(0)=CV(x_\theta), \tag{10}$$

$$\varepsilon(k)=\begin{cases}\varepsilon(0)\left(1-\dfrac{k}{T_c}\right)^{cp}, & 0<k<T_c,\\ 0, & k\geqslant T_c,\end{cases} \tag{11}$$

x_θ 是第 θ 个个体(将初始种群中个体的约束违背值从小到大排序),$\theta=0.05*NP$. 其余相关的参数[5]的取值为:$T_C\in[0.1T_{\max},0.8T_{\max}]$,$cp\in[2,10]$.

ε-约束与可行解的优越性比较两个解的方式是相似的. 但在 ε-约束中,当解的约束违背值小于 ε 时,就认为这个解可行.

2 基于新约束集成的差分进化算法

差分进化算法是在 1995 年由 Storn 和 Price[13]首次提出的,其优点是结构简单、速度快、稳健性好. 近年来,微分进化算法在工程优化设计、数字滤波器设计、图像处理、数据挖掘和多传感器融合等多个领域获得了成功的应用.

受基于约束集成的差分算法(ECHTDE)[7]的启发,本文提出基于新的约束集成的差分进化算法(NECHTDE),用于求解带约束优化问题. 算法包括差分进化算法的两个部分:突变和交叉. 在该算法的突变过程中,采用了 3 个不同的突变策略,产生了 3 个不同的突变向量. 对突变向量进行交叉,产生 3 个不同的试验向量. 选出 3 个试验向量中最好的一个,并将其与目标向量比较,如果试验向量比目标向量好,那么在接下来的迭代中,试验向量将取代目标向量. 下面介绍该算法的基本思路和算法流程.

2.1 初始化过程

在 DE 算法中,个体的编码方式是 $\boldsymbol{x}_i=\{x_{i1},x_{i2},\cdots x_{iD}\}$,$i=1,2,\cdots NP$,$D$ 是维数. 个体的上界、下界定义为 $\boldsymbol{x}_{\min}=\{x^1_{\min},x^2_{\min},\cdots,x^D_{\min}\}$,$x_{\max}=\{x^1_{\max},x^2_{\max},\cdots,x^D_{\max}\}$. 差分进化是一种直接搜索的算法,它利用 NP 个 D 维的参数向量 x_i 作为每一次迭代的种群,初始种群随机均匀分布在整个搜索区域内. 根据式(12)产生 NP 个初始向量,构成初始种群 pop

$$x_{ij}=x^j_{\min}+rand(x^j_{\max}-x^i_{\min}), \tag{12}$$

式中,$j=1,2,\cdots,D$,rand 是[0,1]之间的一个随机数.

2.2 突变、交叉和选择

在产生初始种群之后算法就进入了突变,交叉和选择阶段. NECHTDE 中采用 3 种突变策略,分别是:DE/rand/1,DE/rand/2 和 DE/current-to-best/1.

$$\boldsymbol{v}_i=\boldsymbol{x}_{r1}+F(\boldsymbol{x}_{r2}-\boldsymbol{x}_{r3}), \tag{13}$$

$$\boldsymbol{v}_i=\boldsymbol{x}_{r_1}+F(\boldsymbol{x}_{r_2}-\boldsymbol{x}_{r_3})+F(\boldsymbol{x}_{r_4}-\boldsymbol{x}_{r_5}), \tag{14}$$

$$\boldsymbol{v}_i=\boldsymbol{x}_i+F(\boldsymbol{x}_b-\boldsymbol{x}_i)+F(\boldsymbol{x}_{r1}-\boldsymbol{x}_{r2}). \tag{15}$$

在突变之后,每个目标向量 $\boldsymbol{x}_i(i=1,2,\cdots,NP)$ 就对应 3 个突变向量 $\boldsymbol{v}^k_i(k=1,2,3)$,对这 3 个突变向量进行交叉,产生了 3 个试验向量 $\boldsymbol{u}^k_i(k=1,2,3)$

$$u^k_{ij}=\begin{cases}v^k_{ij} & \text{if } rand<CR\\ x_{ij} & \text{otherwise}\end{cases}. \tag{16}$$

建立外部档案 A,比较 3 个试验向量 $\boldsymbol{u}^k_i(k=1,2,3)$. 首先,用自适应罚函数(SP)方法计算 3 个试验向量的适应度值,将最大适应度值所对应的个体放入外部档案 A 中;对剩余的两个试验向量用 ε-约束(EC)选择,将较差的个体放入外部 档案 A 中;最后剩下的个体记为 $\boldsymbol{u}_i$,将其放入新种群 $newpop$ 中.

对于新种群 $newpop$ 中的每个个体 $new\boldsymbol{x}_i(i=1,2,\cdots,NP)$,在外部档案 A 中找到与其欧氏距离最近的个体,记为 $\boldsymbol{x}_{Ai}$. 用随机排名(SR)方法从 $new\boldsymbol{x}_i$ 和 $\boldsymbol{x}_{Ai}$ 这两个个体中选择较好的个体,并更新种群 $newpop$.

在种群 $newpop$ 更新结束之后,将父代种群 pop 中的个体和新种群 $newpop$ 中的个体用可行解的优越性(SF)两两比较. 选择较好的个体,并更新父代种群 pop. 此时,从更新后的父代种群 pop 中选出最好的一个个体,记为本次迭代中的最好的个体 $\boldsymbol{x}_b$,pop 成为下一代种群.

2.3 局部搜索

引入一个基于 DE 的局部搜索算子[10]. 每隔一代,对当前种群中最好的前 NL 个个体(不包含最好个

体)采用突变策略,进行局部搜索

$$\boldsymbol{x}_L^k=\boldsymbol{x}_k+FL(\boldsymbol{x}_b-\boldsymbol{x}_{r_1})+FL(\boldsymbol{x}_{r_2}-\boldsymbol{x}_{r_3}),\tag{18}$$

式中,$\boldsymbol{x}_L^k$ 是第 k 个个体 $\boldsymbol{x}_k(k=1,2,\cdots NL)$ 通过局部搜索产生的,$\boldsymbol{x}_b$ 是当前种群中最好的个体,FL 是局部搜索中的突变因子,$\boldsymbol{x}_{r_1}$,$\boldsymbol{x}_{r_3}$是从当前整个种群中随机选取的个体,$\boldsymbol{x}_{r_2}$是从当代种群排名后的前 $NP/2$ 个个体中随机选取的. 在局部搜索生成的 NL 个体中,采用可行解的优越性,从中选出最好的个体并记为 $\boldsymbol{x}_L$. 将其与当前种群中最佳个体 $\boldsymbol{x}_b$ 进行比较. 如果 $\boldsymbol{x}_L$ 优于 $\boldsymbol{x}_b$,那么当前种群中最差的个体 $\boldsymbol{x}_w$就会被 $\boldsymbol{x}_L$ 取代,否则 $\boldsymbol{x}_w$就会被 $\boldsymbol{x}_b$ 取代.

基于 2.1~2.3 节对初始化过程、突变、交叉和选择操作,以及局部搜索的描述,ECHTDE 算法的流程如图 1 所示.

基于新约束集成的差分进化算法(NECHTDE)

1:输入　种群数 NP,维数 D,局部搜索频率 f_L,局部搜索系数 FL,最大函数计算次数 $MaxFES$,$G=1$,$FES=0$,外部档案 $A=\phi$,NL;
2:初始化　利用式(12)初始化 pop,得到目标向量 $\boldsymbol{x}_i$,计算其目标函数值 $f(\boldsymbol{x}_i)(i=1,2,\cdots,NP)$;
3:通过式(2)计算种群中每个个体的约束违背值 $CV(\boldsymbol{x}_i)(i=1,2,\cdots,NP)$;
4:$FES=FES+NP$;
5:while $FES\leqslant MaxFES$ do
6:$newpop=\phi$;
6:for $i=1:NP$ do
7:对于目标向量 $\boldsymbol{x}_i$,使用 3 种不同的突变策略(DE/rand/1,DE/rand/2 和 DE/current-to-best/1),产生 3 个突变向量,交叉之后,产生了 3 个试验向量 $\boldsymbol{u}_i^k(k=1,2,3)$;
8:计算 3 个试验向量 $\boldsymbol{u}_i^k(k=1,2,3)$ 的函数值和约束违背值;
9:用自适应罚函数(SP)方法对 $\boldsymbol{u}_i^k(k=1,2,3)$ 进行评估,将 3 个个体中最差的个体放入外部档案 A 中;
10:对剩余的两个个体用 ε-约束方法进行比较,将两个个体中较差的个体放入外部档案 A 中;
11:将剩下的个体记为 $\boldsymbol{u}_i$;并放入新种群 $newpop$ 中;
12:$FES=FES+3$;
13:end for
14:for $i=1:NP$ do
15:从外部档案 A 中找一个与 $\boldsymbol{u}_i$ 欧式距离最近的个体记为 $\boldsymbol{x}_{Ai}$,用随机排名(SR)从两个个体中选择较好的一个记为 $new\boldsymbol{x}_i$,更新 $newpop$;
16:end for
17:for $i=1:NP$ do
18:将 new$\boldsymbol{x}_i$ 与 $\boldsymbol{x}_i$,用可行解的优越性(SF)进行比较,选出较好的个体,作为下一代种群中的个体;
19:end for
20:if(rem(G,f_L)= =0)
21:对当前种群中排名前 NL 个个体(除最佳个体外)进行局部搜索(18),用可行解的优越性(SF)找到局部搜索中产生的最好个体,并记为 $\boldsymbol{x}_L$;
22:如果 $\boldsymbol{x}_L$ 优于 $\boldsymbol{x}_b$,那么用 $\boldsymbol{x}_L$ 代替当前种群中最差的个体 $\boldsymbol{x}_w$,否则用 $\boldsymbol{x}_b$ 代替 $\boldsymbol{x}_w$;
23:对当前种群排名,更新 $\boldsymbol{x}_b$;
24:$FES=FES+NL$;
25:end if
26:$G=G+1$;
27:end while

图 1　NECHTDE 算法流程

Fig. 1　The flowchart of NECHTDE algorithm

3　数值实验

3.1　比较算法

为了验证 NECHTDE 算法的性能,在 CEC 2017[14] 的 28 个基准测试函数上进行测试. 函数 C01、C02、C03、C05、C13、C14、C17 与 C20 是不可分函数,C04、C06、C07、C08、C09、C10、C11、C12、C13、C16、C18 与 C19 是可分函数,C21~C28 是旋转函数.

为了展现新算法的优越性,将其和统一差分进化(UDE)[10]、多策略的 LSHADE 算法(LSHADE44)[15]、LSHADE44 和 IDE 相结合的算法(LSHADE44-IDE)[16]、SHADE[17] 和基于约束集成的差分进化算法(ECHTDE)[7] 等进行比较. UDE[10] 算法是由 Trivedi 等人在 2017 年提出的,在 DE 算法的基础上采用了一种新的约束集成,在算法前期采用可行解的优越性,在算法后期采用罚函数法. LSHADE44[15] 算法是在 LSHADE 的基础上添加了另外的 3 种突变策略并引入策略之间的竞争机制. LSHADE44-IDE[16] 算法分为两个阶段:第一阶段采用的是 LSHADE44 算法;第二阶段使用的是具有个体依赖机制的自适应 DE 变体. SHADE[17] 算法包含了种群规模线性缩减,增强了对约束违背的处理. 在约束处理方法中,可行解优先于不可行解. 然而,当不可行解的约束违背值在阈值以下时,就认为是可行解. 表 1 给出各个算法的参数设置,各算法的最大函数计算次数在 30 维时设为 60 000,所有的测试结果都是各算法单独运行 25 次的结果.

表 1　比较算法的参数设置

Table 1　Parameter settings of comparative algorithms

算法	参数设置
UDE[10]	参数池 $F=0.9, CR=0.9, F=0.5, CR=0.5, p=0.11, L=25, f_L=2, N_L=10$,罚参数 $penalty=100, F_L=0.9$,种群数目 $NP=10*D$
LSHADE44[15]	$p=0.11, n_0=2, K=4, \delta=1/(5*K), H=6, A=2.6*N$,种群数目 $N_{\text{init}}=18*D, N_{\max}=4$
LSHADE44-IDE[16]	$p=0.11, A=2.6*N, H=6$,种群数目 $N_{\text{init}}=18*D, N_{\max}=50$
SHADE[17]	$p=0.11, g_c=500, H=5$,种群数目 $N_{\text{init}}=2*D$
ECHTDE[7]	$T_c=0.2*T_{\max}, cp=5, p_f$ 是从 0.475~0.025 的线性递减,种群数目 $NP=50$
NECHTDE	$T_c=0.2*T_{\max}, cp=5, p_f$ 是从 0.475~0.025 的线性递减,种群数目 $NP=50, f_L=2, NL=10, FL=0.9$

容许值 δ 是对约束优化问题中等式约束的松弛:

$$\delta(t+1)=\frac{\delta(t)}{\hat{\delta}}. \tag{19}$$

初始的 $\delta(0)$ 是初始种群中等式约束违背值的中值. $\hat{\delta}$ 是用来控制 δ,可以确保 δ 在最大迭代次数的一半迭代次数附近减小到 10^{-4},之后将 δ 的值固定为 10^{-4}.

3.2　实验结果与分析

表 2 是 28 个测试函数在 30 维时在不同算法下的测试结果.

表 2　基准函数在 30 维上的测试结果

Table 2　The test results of benchmark functions on 30D

F	Criteria	UDE	LSHADE44	LSHADE44-IDE	SHADE	ECHTDE	NECHTDE
C01	Md	0(1)	0(1)	0(1)	0(1)	2.60e-04(6)	0(1)
	CV	0	0	0	0	0	0
	Mn	0(1)	0(1)	0(1)	0(1)	2.61e-04(6)	0(1)
	FR	100	100	100	100	100	100
	vio	0	0	0	0	0	0
C02	Md	0(1)	0(1)	0(1)	0(1)	2.55e-04(6)	0(1)
	CV	0	0	0	0	0	0
	Mn	0(1)	0(1)	0(1)	0(1)	2.55e-04(6)	0(1)
	FR	100	100	100	100	100	100
	vio	0	0	0	0	0	0
C03	Md	7.47e+01(1)	2.06e+05(3)	6.58e+06(5)	7.36e+05(6)	3.96e+05(4)	6.55e+04(2)
	CV	0	0	0	1.44e-03	0	0
	Mn	7.33e+01(1)	3.55e+05(3)	6.70e+06(5)	1.29e+06(6)	3.96e+05(4)	6.61e+04(2)
	FR	100	100	100	32	100	100
	vio	0	0	0	2.43e-02	0	0

续表 2　Table 2 continued

F	Criteria	UDE	LSHADE44	LSHADE44-IDE	SHADE	ECHTDE	NECHTDE
C04	Md	8.36e+01(3)	1.36e+01(2)	1.35e+01(1)	1.13e+02(6)	1.10e+02(5)	9.95e+01(4)
	CV	0	0	0	0	0	0
	Mn	8.24e+01(3)	1.36e+01(1)	1.39e+01(2)	1.15e+02(6)	1.11e+02(5)	9.87e+01(4)
	FR	100	100	100	100	100	100
	vio	0	0	0	0	0	0
C05	Md	0(1)	0(1)	0(1)	0(1)	2.81e-06(6)	8.74e-07(5)
	CV	0	0	0	0	0	0
	Mean	0(1)	0(1)	0(1)	7.97e-01(6)	5.19e-06(5)	7.42e-07(4)
	FR	100	100	100	100	100	100
	vio	0	0	0	0	0	0
C06	Md	2.55e+02(1)	5.80e+03(4)	4.37e+03(5)	3.82e+03(3)	3.23e+03(6)	1.94e+03(2)
	CV	0	1.27e-02	2.55e-02	0	8.53e-01	0
	Mn	3.03e+02(1)	4.07e+03(4)	5.53e+03(5)	3.74e+03(3)	3.43e+03(6)	2.29e+03(2)
	FR	100	0	0	100	0	100
	vio	0	1.50e-02	2.57e-02	0	1.4082	0
C07	Md	-6.22e+02(1)	-1.34e+02(3)	-8.07e+01(4)	-3.26e+01(5)	-8.95e+01(6)	-2.68e+02(2)
	CV	0	0	0	0	2.41e+01	0
	Mn	-5.98e+02(1)	-1.09e+02(3)	-8.11e+01(4)	-2.41e+01(5)	-9.33e+01(6)	-2.73e+02(2)
	FR	100	96	96	52	0	100
	vio	0	0	0	3.56e-03	1.24e+02	0
C08	Md	-2.80e-04(4)	-2.80e-04(4)	-2.70e-04(6)	-2.84e-04(3)	-8.82e-03(1)	-3.78e-03(2)
	CV	0	0	0	0	0	0
	Mn	-2.80e-04(3)	-2.80e-04(3)	-2.60e-04(5)	-2.84e-04(2)	-9.14e-03(6)	-3.53e-03(1)
	FR	100	100	100	100	92	100
	vio	0	0	0	0	0	0
C09	Md	-2.67e-03(1)	-2.67e-03(1)	-2.67e-03(1)	-2.666e-03(4)	-2.665e-03(5)	-2.04e-03(6)
	CV	0	0	0	0	0	0
	Mn	-2.67e-03(1)	-2.67e-03(1)	-2.67e-03(1)	2.33e-02(6)	-2.665e-03(4)	-1.18e-03(5)
	FR	100	100	100	96	100	100
	vio	0	0	0	1.07e+06	0	0
C10	Md	-1.00e-04(4)	-1.00e-04(4)	-9.90e-05(6)	-1.03e-04(3)	-1.02e-02(1)	-8.35e-04(2)
	CV	0	0	0	0	0	0
	Mn	-1.00e-04(4)	-1.00e-04(4)	-9.80e-05(6)	-1.03e-04(3)	-1.01e-02(1)	-8.23e-04(2)
	FR	100	100	100	100	100	100
	vio	0	0	0	0	0	0
C11	Md	-2.56e+01(5)	-9.249e-01(2)	-9.247e-01(3)	-9.19e-01(4)	-1.69e+01(6)	-7.57e+02(1)
	CV	4.26e-02	0	0	0	5.15e-02	0
	Mn	-2.83e+01(5)	-8.74e-01(2)	-8.65e-01(3)	7.81e-01(4)	-1.60e+01(6)	-7.86e+02(1)
	FR	0	100	100	100	0	100
	vio	2.07e-02	0	0	0	1.05e-01	0
C12	Md	9.77516(4)	3.9853(3)	3.9825(1)	9.7752(6)	3.9831(2)	9.77517(5)
	CV	0	0	0	0	0	0
	Mn	1.87e+01(6)	3.98(1)	6.07(2)	1.42e+01(5)	6.6341(3)	1.08e+01(4)
	FR	100	100	100	100	100	100
	vio	0	0	0	0	0	0
C13	Md	8.06e+01(4)	5.91(2)	0(1)	1.45e+02(5)	2.69e+02(6)	7.79e+01(3)
	CV	0	0	0	0	0	0
	Mn	8.15e+01(4)	6.15(1)	2.60e+01(2)	3.53e+03(5)	1.39e+03(6)	7.59e+01(3)
	FR	100	100	100	100	96	100
	vio	0	0	0	0	9.94e-02	0
C14	Md	1.4456(1)	1.8394(4)	1.8962(5)	1.49544(2)	2.1719(6)	1.49544(2)
	CV	0	0	0	0	0	0
	Mn	1.5258(2)	1.8296(4)	1.9086(5)	1.5484(3)	2.1657(6)	1.4781(1)
	FR	100	100	100	100	100	100
	vio	0	0	0	0	0	0

续表 2 Table 2 continued

F	Criteria	UDE	LSHADE44	LSHADE44-IDE	SHADE	ECHTDE	NECHTDE
C15	Md	8.64(1)	1.81e+01(4)	1.18e+01(2)	2.36e+02(6)	2.12e+01(5)	1.46e+01(3)
	CV	0	0	0	0	0	0
	Mn	9.14(4)	1.92e+01(5)	1.29e+01(1)	2.41 e+01(6)	2.12e+01(3)	1.42e+01(2)
	FR	88	84	100	48	100	100
	vio	0	0	0	2.01e+02	0	0
C16	Md	6.2831(1)	1.51e+02(4)	1.44e+02(3)	2.21e+02(6)	1.52e+02(5)	1.38e+02(2)
	CV	0	0	0	2.821e-03	0	0
	Mn	8.4193(1)	1.46e+02(4)	1.43e+02(3)	2.11e+02(6)	1.53e+02(5)	1.38e+02(2)
	FR	100	100	100	16	100	100
	vio	0	0	0	8.83e-03	0	0
C17	Md	1.0109(3)	1.0010(1)	1.0167(4)	1.0299(6)	1.0296(5)	1.0096(2)
	CV	1.55e+01	1.55e+01	1.55e+01	1.5501e+01	1.55e+01	1.55e+01
	Mn	1.0214(4)	9.97e-01(2)	1.0128(3)	1.0312(6)	1.0297(5)	1.0132(1)
	FR	0	0	0	0	0	0
	vio	1.55e+01	1.55e+01	1.55e+01	1.73e+01	1.55e+01	1.546e+01
C18	Md	3.11e+03(5)	3.05e+03(1)	7.89e+02(4)	1.50e+03(3)	3.15e+04(6)	6.19e+04(2)
	CV	1.59e+07	2.08e+03	1.01e+07	7.68e+06	2.47e+09	1.15e+04
	Mn	3.57e+03(4)	2.79e+03(2)	4.01e+03(3)	6.71e+03(5)	3.16e+04(6)	4.88e+04(1)
	FR	0	0	0	0	0	0
	vio	3.36e+07	2.84e+05	1.33e+06	8.76e+08	2.47e+09	1.74e+04
C19	Md	4.16(6)	6.16e-06(4)	0(5)	1e-05(1)	7.06e-05(3)	1.08e-05(2)
	CV	2.1379e+04	2.13749e+04	2.137491e+04	1.42493e+04	1.424994e+04	1.424994e+04
	Mn	3.65(6)	6.20e-06(4)	0(5)	3.09e-01(3)	7.07e-05(2)	1.07e-05(1)
	FR	0	0	0	0	0	0
	vio	2.13789e+04	2.13749e+04	2.137491e+04	1.425e+05	1.424994e+04	1.424994e+04
C20	Md	4.4524(5)	1.4509(1)	2.2894(3)	2.0772(2)	6.8016(6)	2.6524(4)
	CV	0	0	0	0	0	0
	Mn	4.5816(5)	1.4171(1)	2.2475(3)	2.1134(2)	6.8016(6)	2.8356(4)
	FR	100	100	100	100	100	100
	vio	0	0	0	0	0	0
C21	Md	9.7752(2)	2.83261e+01(4)	2.83262e+01(5)	9.7752(2)	2.84e+01(6)	3.98(1)
	CV	0	0	0	0	0	0
	Mn	1.19e+01(6)	2.25e+01(3)	2.74e+01(4)	1.32e+01(2)	2.85e+01(5)	7.41(1)
	FR	92	100	100	100	100	100
	vio	0	0	0	0	0	0
C22	Md	8.06e+01(1)	7.96e+02(4)	4.78e+02(3)	1.79e+05(5)	2.58e+04(6)	2.73e+02(2)
	CV	0	0	0	0	1.84e+01	0
	Mn	9.41e+01(1)	1.27e+03(4)	8.44e+02(3)	3.42e+05(5)	2.76e+04(6)	3.05e+02(2)
	FR	100	100	100	72	4	100
	vio	0	0	0	2.76e-01	2.37e+01	0
C23	Md	1.4085(1)	1.7407(4)	1.8483(5)	1.4923(3)	2.3061(6)	1.4590(2)
	CV	0	0	0	0	0	0
	Mn	1.4349(1)	1.7151(3)	1.8564(4)	1.5818(6)	2.3061(5)	1.4761(2)
	FR	100	100	100	84	100	100
	vio	0	0	0	1.60e-04	0	0
C24	Md	8.6393(1)	1.18e+01(2)	1.492e+01(4)	1.88e+01(6)	2.3505(5)	1.491e+01(3)
	CV	0	0	0	4.34e-02	6.53e-03	0
	Mn	8.3879(1)	1.22e+01(2)	1.39e+01(3)	2.09e+01(5)	2.3061(6)	1.53e+01(4)
	FR	100	100	100	20	8	100
	vio	0	0	0	1.47e+01	7.97e-03	0
C25	Md	1.41e+01(1)	1.46e+02(3)	1.39e+02(2)	2.14e+02(6)	2.34e+02(5)	1.77e+02(4)
	CV	0	0	0	9.66e-04	0	0
	Mn	1.59e+01(1)	1.42e+02(3)	1.41e+02(2)	2.07e+02(6)	2.34e+02(5)	1.79e+02(4)
	FR	100	100	100	32	100	100
	vio	0	0	0	2.34e-02	0	0

续表 2　Table 2 continued

F	Criteria	UDE	LSHADE44	LSHADE44-IDE	SHADE	ECHTDE	NECHTDE
C26	Md	1.0227(4)	0.9990(3)	0.9837(2)	1.0299(6)	1.0304(5)	0.9527(1)
	CV	1.55e+01	1.55e+01	1.55e+01	1.5503e+01	1.55e+01	1.55e+01
	Mn	1.0194(4)	0.9972(2)	1.0123(3)	1.0353(6)	1.0304(5)	0.9543(1)
	FR	0	0	0	0	0	0
	vio	1.55e+01	1.55e+01	1.55e+01	2.15e+01	1.55e+01	1.55e+01
C27	Md	9.04e+03(5)	1.24e+05(2)	1.15e+06(4)	2.33e+03(1)	1.22e+05(6)	2.02e+05(3)
	CV	2.12e+08	5.10e+06	6.83e+07	4.09e+06	2.61e+10	9.36e+06
	Mn	1.18e+04(5)	9.31e+03(3)	5.22e+04(2)	3.14e+03(4)	1.22e+05(6)	1.90e+05(1)
	FR	0	0	0	0	0	0
	vio	3.41e+08	6.88e+06	4.68e+05	1.52e+07	2.61e+10	6.25e+04
C28	Md	6.67e+01(4)	1.44e+02(6)	1.91e+02(5)	1.29e+02(3)	1.41e+02(1)	1.36e+02(2)
	CV	2.1444e+05	2.1485e+05	2.1484e+05	1.43e+05	1.43e+04	2.14e+04
	Mean	6.43e+01(4)	1.43e+02(6)	1.53e+02(5)	1.22e+02(3)	1.41e+02(2)	1.31e+02(1)
	FR	0	0	0	0	0	0
	vio	2.1446e+05	2.1483e+05	2.1481e+05	1.43e+05	1.4334e+04	1.4315e+04

表 2 中“Mn”表示各基准函数在每个算法独立运行 25 次得到的 25 个解的目标函数值的平均值,“Md”表示 25 次运行结果的中值,“CV”是中值解的约束违背值. 可行运行是指在一次运行过程中,至少出现一个可行解,则该次运行就是可行运行,“FR”表示的是可行率,即在 25 次独立运行中可行运行所占的比率. “vio”表示所有运行过程中所有解的约束违背的平均值[17]. 表格中圆括号里的数字表示该算法的两种排名. 对于带约束的优化问题有两种算法排名方法:

第一种:根据中值排名

(1)对于同一个函数,先根据中值所对应的解的约束违背值排名;

(2)如果约束违背值相同,根据目标函数值排名;

第二种:根据平均值排名

(1)先根据可行率排名;

(2)如果可行率相同,根据平均违背值(vio)对算法进行排序;

(3)如果平均违背值(vio)相同,根据平均目标函数值对算法进行排序.

对 NECHTDE 算法的比较实验结果分析如下:

(1)不可分问题:对于 C01 和 C02 ECHTDE 算法的表现较差,其他所有算法的性能都一样收敛到 0. 在求解 C03 时 NECHTDE 仅次于 UDE,居于第二位. NECHTDE 在 C05 上表现较差,但在 C14、C17 这两个函数上,根据均值排名,NECHTDE 是表现最好的算法,根据中值排名是排名第二的算法. 对于其他不可分函数,NECHTDE 并没有表现出最好的优势,是属于排名居中的算法.

(2)可分问题:NECHTDE 在 C04,C09 和 C12 这 3 个函数上无论是根据中值排名还是根据均值排名都较差. 在 C06、C07、C08、C10 和 C16 这 5 个函数上,NECHTDE 是排名第二的算法. C11 是算法 NECHTDE 表现最好的一个函数. C18 和 C19 上 NECHTDE 算法根据中值排名是第二名;根据均值排名,位于第一名. 在函数 C15 上,NECHTDE 算法表现较为一般,排在第三位.

(3)旋转问题:NECHTDE 在 C21、C26 上表现最好. NECHTDE 在 C24 和 C25 上表现一般,排在第三、第四位. 在 C22 和 C23 上,NECHTDE 的表现仅次于 UDE,是排名第二的算法. NECHTDE 在剩余的 2 个旋转函数上,根据均值排名,是位于第一的算法.

从表 2 知,UDE,LSHADE44,LSHADE44-IDE,SHADE 和 ECHTDE 算法在 28 个 30 维测试函数均值的排名分别为:81,74,87,121 和 137,其平均排名为 2.89,2.64,3.10,4.32 和 4.89. 这 5 个算法根据中值的排名分别为:72,78,92,106 和 136,其平均排名为 2.57,2.78,3.28,3.78 和 4.85. 而 NECHTDE 根据均值和中值的排名为(60,71),其平均排名为(2.17,2.53). 图 2~图 3 是函数 C14、C21 的收敛曲线图. 由此可以看出 NECHTDE 算法的性能要好于其他 5 种比较算法.

表 3　30 维时 6 种算法排名结果

Table 3　Ranking results of 6 algorithms on 30D

Algorithm	Mean Rank	Median Rank	Total Rank	Rank	Algorithm	Mean Rank	Median Rank	Total Rank	Rank
UDE	81	72	153	3	SHADE	121	106	227	5
LSHADE44	74	78	152	2	ECHTDE	137	136	276	6
LSHADE44-IDE	87	92	179	4	NECHTDE	60	71	131	1

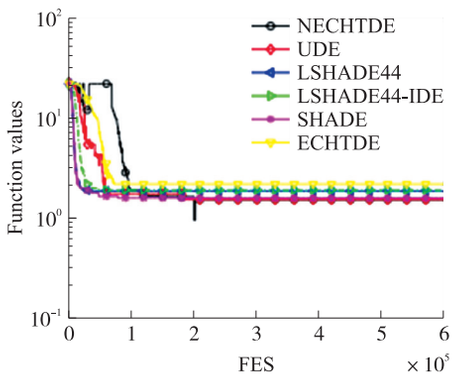

图 2　30 维函数 C14 收敛曲线

Fig. 2　Convergence curve of C14 on 30D

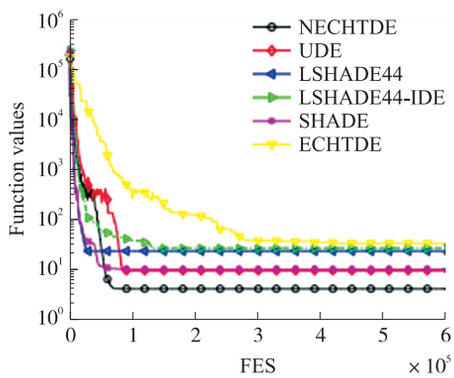

图 3　30 维函数 C21 收敛曲线

Fig. 3　Convergence curve of C21 on 30D

4　结语

由于带约束的优化问题比较复杂,常用约束处理技术处理约束. 近年来一些学者提出将约束处理集成技术用于处理约束优化问题. 本文提出一种基于差分进化算法的新约束集成技术(NECHTDE). 新算法使用 3 个突变策略产生突变向量,使得个体在整个空间中的分布更加均匀;依次采用多种约束处理技术进行选择,可以充分地利用较好的不可行解,提高了下一代种群中个体的质量;通过引入局部搜索,增强算法局部寻优能力,避免算法陷入局部最优. 实验结果表明,该算法的性能优于其他几种比较算法,问题的求解精度和收敛速度得到了提高.

新算法仅在基准函数上进行了测试,还可以将其应用到更多实际生活中,解决复杂的优化问题,如工程设计、数据挖掘、通信系统的设计和优化等. 另一方面,如何将改进算法扩展到处理组合优化问题,如设施选址问题、分配问题、排序问题等,也是我们需要进一步考虑的研究方向.

[参考文献]

[1] MICHALEWICZ Z,SCHOENAUER M. Evolutionary algorithms for constrained parameter optimization problems[J]. Evolutionary computation,1996,4(1):1-32.

[2] KOZIEL S,MICHALEWICZ Z. Evolutionary algorithms,homomorphous mappings,and constrained parameter optimization[J]. Evolutionary computation,1999,7(1):19-44.

[3] RUNARSSON T P,YAO X. Stochastic ranking for constrained evolutionary optimization[J]. IEEE transactions evolutionary computation,2000,4(3):284-294.

[4] TESSEMA B,YEN G G. A self adaptive penalty function based algorithm for constrained optimization[C]//2006 IEEE Congress on Evolutionary Computation(CEC),Vancouver,BC,Canada,2006:246-253.

[5] TAKAHAMA T,SAKAI S. Constrained optimization by the constrained differential evolution with gradient-based mutation and feasible elites[C]//2006 IEEE Congress on Evolutionary Computation(CEC),Vancouver,BC,Canda,2006:246-253,1-8.

[6] WANG Y,CAI Z,GUO G,et al. Multiobjective optimization and hybrid evolutionary algorithm to solve constrained optimization problems[J]. IEEE transactions on systems man & cybernetics part B,2007,37(3):560-575.

[7] MALLIPEDDI R,SUGANTHAN P N. Ensemble of constraint handling techniques[J]. IEEE transactions on evolutionary computation,2010,14(4):561-579.

[8] WOLPERT D H,MACREADY W G. No free lunch theorems for optimization[J]. IEEE transactions evolutionary computation,

1997,1(1):67-82.

[9] DEB K. An efficient constraint handling method for genetic algorithms[J]. Computer methods in applied mechanics and engineering,2000,186(2):311-338.

[10] TRIVEDI A,SANYAL K,VERMA P,et al. A unified differential evolution algorithm for constrained optimization problems[C]//2017 IEEE Congress on Evolutionary Computation(CEC),San Sebastian,Spain,2017:1231-1238.

[11] 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.

[12] WANG Y,CAI Z,ZHOU Y,et al. An adaptive trade-off model for constrained evolutionary optimization[J]. IEEE transactions on evolutionary computation,2008,12(1):80-92.

[13] STORN R,PRICE K V. Differential evolution—a simple and efficient heuristic for global optimization over continuous spaces[J]. Global optimization,1997,11:41-359.

[14] WU G,MALLIPEDI R,SUGANTHAN P N. Problem definitions and evaluation criteria for the CEC 2017 competition on constrained real-parameter optimization[R]. Singapore:Nanyang Technological University,2016.

[15] R.Pol6kov6. L-SHADE with competing strategies applied to constrained optimization[C]//2017 IEEE Congress on Evolutionary Computation(CEC),San Sebastian,Spain,2017:1683-1689.

[16] TVRDIK J,POLAKOVA R. A simple framework for constrained problems with application of L-SHADE44 and IDE[C]//2017 IEEE Congress on Evolutionary Computation(CEC),San Sebastian,Spain,2017:1436-1443.

[17] ALE Š,ZAMUDA. Adaptive constraint handling and success history differential evolution for CEC 2017 constrained real-parameter optimization[C]//2017 IEEE Congress on Evolutionary Computation(CEC),San Sebastian,Spain,2017:2443-2450.

[责任编辑:陆炳新]