https://doi.org/10.1051/jnwpu/20193751000

基于广义反向学习的自适应约束差分进化算法

吴文海[1], 郭晓峰[1], 周思羽[1], 刘锦涛[2]

(1.海军航空大学(青岛校区) 航空仪电控制工程与指挥系, 山东 青岛 266041;
2.陆军工程大学 指挥控制工程学院, 江苏 南京 210002)

摘 要:差分进化算法是一种基于“贪婪竞争”机制的全局寻优算法,其控制参量少、结构简单,具有较高的可靠性和收敛性,将约束处理机制引入到差分进化算法可以高效解决约束优化问题。提出一种基于广义反向学习的自适应约束差分进化算法,利用广义反向学习机制生成初始种群并执行种群"代跳"操作,采用自适应权衡模型将约束区分状态处理以及改进自适应变异操作对个体进行排序变异。通过与 CDE、DDE、A-DDE、εDE 以及 DPDE 算法进行试验比较以及对广义反向学习和改进自适应排序操作性能分析证明该算法具有较好的寻优精度及收敛速度。

关 键 词:约束优化;差分进化算法;广义反向学习;自适应;权衡模型;排序变异操作
中图分类号:TP18 **文献标志码**:A **文章编号**:1000-2758(2019)05-1000-11

在实际问题及工程优化方面存在着诸多约束限制,这类问题被称为约束优化问题(constrained optimization problems,COPs),处理这类问题的主要矛盾为最终解须满足全部约束限制,但由于约束的限制使得可行域变小,因而增加了搜索难度。一般进化算法(evolution algorithm,EA)在解决无约束优化问题中取得了优异表现,因此,近几年以来众多学者开始研究不同的约束处理机制,以期改进 EA 在约束优化问题中的不足,解决约束优化带来的难题[1]。

差分进化(differential evolution,DE)算法是基于“贪婪竞争”机制的全局寻优算法,具有结构简单、控制参量少,可靠性和鲁棒性强的特点[2]。将约束处理机制引入到 DE(constrained differential evolution,CDE)算法中来解决约束优化问题成为广阔的研究领域。Takahama 等人[3]提出了一种 算法,该算法将 约束处理机制引入到 DE 算法中实现对约束优化问题的求解,然而约束缩放程度 的选取对于算法寻优性能影响较大;Mallipeddi 等人[4]提出一种整体约束处理机制来解决约束优化问题,该方法为每一约束处理机制分配属于自己的种群进行运算,但是 4 种不同的约束处理机制大大增加了算法的复杂程度;Wang 等人[5]提出一种(μ+λ)-CDE 算法,为了丰富种群多样性,该算法对父代种群采取 3 种不同的变异策略生成 3 种不同后代种群共同参与进化;为了克服(μ+λ)-CDE 的不足,Jia 等人[6]对该算法进行了改进,并提出了一种基于存档的新型自适应权衡模型(ArATM)来处理约束条件,以上 2 种算法同时执行 3 种变异策略,大大增加了算法评价次数,其次,变异策略无法根据种群状态充分利用“精英”个体信息,算法灵活性不足,最终导致算法收敛速度下降;Gong 等人[7]将种群排序嵌入到变异操作中,并对工程优化问题进行分析,证明了算法有效性,但该算法在选择概率计算方面存在不足:不可行状态下的选择概率采用线性计算,无法保证多样性要求,半可行和可行状态下选择概率计算方式无法充分体现个体间的支配优势,无法在利用“精英”个体信息的前提下兼顾种群多样性;Wang 等人[8]提出了一种基于广义反向学习的种群初始化方法,该方法通过在种群初始化阶段生成初始和反向 2 个种群增加了算法在初始阶段的多样性和效率,然而算法仅考虑在初始阶段采用广义反向学习策略,并未将其应用于全局之中,降低了广义反向学习策略的功效性。本文受文献[5-8]的启发,提出一种基于广义反向学习的自适应约束差分进化(GOBL-

收稿日期:2018-10-08 **基金项目**:国家重点研发计划(2018YFC0806900,2016YFC0800606,2016YFC0800310)资助
作者简介:吴文海(1962—),海军航空大学教授、博士生导师,主要从事精确制导与控制研究。

ACDE)算法。初始阶段利用广义反向学习生成初始化种群,在每一代进化结束后利用广义反向学习执行种群"代跳"操作,提高算法多样性,避免算法陷入局部最优;引用自适应权衡模型将种群分为 3 种状态,分别对约束限制进行处理;采用改进自适应变异操作,根据个体排序值,分别选取余弦和反余弦模型计算选择概率,充分利用"精英"个体支配优势,针对前述算法变异策略固定的缺点,依据可行个体比率自适应选择"improved rand-to-best and current/2"策略完成变异,提高算法效率及动态性。通过与几种 CDE 算法在寻优精度的比较以及对广义反向学习和改进自适应排序操作在收敛性能的分析,证明本算法在处理约束优化问题方面具有较好的表现。

1　基本知识

1.1　约束优化问题

约束优化问题中,限制条件通常包含等式、不等式及边界约束等不同形式,本文对其做出如下定义:

$$\begin{aligned}&\text{minimize: } f(\boldsymbol{x}),\boldsymbol{x}\in D\\&\qquad g_j(\boldsymbol{x})\leqslant 0,j=1,\cdots,l\\&\text{subject to: } h_j(\boldsymbol{x})=0,j=l+1,\cdots,m\\&\qquad l_i\leqslant x_i\leqslant u_i,i=1,\cdots,n\end{aligned}\tag{1}$$

式中,$\boldsymbol{x}=(x_1,x_2,\cdots,x_n)\in \boldsymbol{D}\subseteq \mathbf{R}^n$ 为 n 维决策向量,$f:\boldsymbol{D}\rightarrow \boldsymbol{R}$ 为目标函数,$g_j(\boldsymbol{x})\leqslant 0$ 和 $h_j(x)=0$ 分别为 l 个不等式约束和 $m-l$ 个等式约束,u_i 和 l_i 分别为 x_i 的上、下界。

处理等式约束时,通常将其转化为相应的不等式约束

$$|h_j(\boldsymbol{x})|-\delta\leqslant 0,j\in\{l+1,\cdots,m\}\tag{2}$$

式中,δ 为正容忍因子,取值为 0.000 1。

解 $\boldsymbol{x}$ 到第 j 个约束的距离定义为

$$G_j(\boldsymbol{x})=\begin{cases}\max\{0,g_j(\boldsymbol{x})\}, & 1\leqslant j\leqslant l\\ \max\{0,|h_j(\boldsymbol{x})|-\delta\}, & l+1\leqslant j\leqslant m\end{cases}\tag{3}$$

则解 $\boldsymbol{x}$ 到可行域边界的距离宇航义为约束违反程度,可表示为

$$G(\boldsymbol{x})=\sum_{j=1}^{p}G_j(\boldsymbol{x})\tag{4}$$

1.2　差分进化算法

差分进化算法是一种基于种群变异的智能算法,种群个体通过差分变异、交叉、"贪婪"选择等操作产生后代个体,实现种群进化[2]。

差分变异过程:父代向量 $\boldsymbol{x}_i$ 经变异因子 F 缩放后产生新的变异向量 $\boldsymbol{v}_i$,根据其生成方式差异,常用策略有:

1)"DE/rand/1"

$$\boldsymbol{v}_{i,t}=\boldsymbol{x}_{r1,t}+F(x_{r2,t}-x_{r3,t})\tag{5}$$

2)"DE/best/1"

$$\boldsymbol{v}_{i,t}=\boldsymbol{x}_{\text{best},t}+F(x_{r2,t}-x_{r3,t})\tag{6}$$

3)"DE/rand/2"

$$\boldsymbol{v}_{i,t}=\boldsymbol{x}_{r1,t}+F(x_{r2,t}-x_{r3,t})+F(x_{r4,t}-x_{r5,t})\tag{7}$$

4)"DE/rand-to-best/1"

$$\boldsymbol{v}_{i,t}=\boldsymbol{x}_{i,t}+F(x_{best,t}-x_{i,t})+F(x_{r1,t}-x_{r2,t})\tag{8}$$

5)"DE/current-to-rand/1"

$$\boldsymbol{v}_{i,t}=\boldsymbol{x}_{i,t}+F(x_{r1,t}-x_{i,t})+F(x_{r2,t}-x_{r3,t})\tag{9}$$

式中,$x_{r1,t},x_{r2,t},x_{r3,t},x_{r4,t},x_{r5,t}$ 为种群第 t 代中随机选取的不同个体,$x_{\text{best},t}$ 为种群第 t 代中适应值最优个体,rand 为[0,1] 均匀分布随机数。

交叉过程:父代向量 $\boldsymbol{x}_{i,t}$ 与变异向量 $\boldsymbol{v}_{i,t}$ 通过交叉因子 C_R 杂交,产生试验向量 $\boldsymbol{u}_{i,t}$,具体操作如下

$$\boldsymbol{u}_{ji,t}=\begin{cases}\boldsymbol{v}_{ji,t}, & \text{if}\quad \text{rand}_j\leqslant C_R\quad\text{or}\quad j=j_{\text{rand}}\\ \boldsymbol{x}_{ji,t}, & \text{otherwise}\end{cases}\tag{10}$$

式中,$j=1,2,\cdots,n$,j_{rand} 是为了确保试验向量 $\boldsymbol{u}_{i,t}$ 异于目标向量 $\boldsymbol{x}_{i,t}$ 而选取参数,交叉因子 $C_R\in(0,1)$。

选择过程:差分进化算法对父代向量 $\boldsymbol{x}_{i,t}$ 和试验向量 $\boldsymbol{u}_{i,t}$ 的适应值进行比较,选取较优个体进入下代种群:

$$\boldsymbol{x}_{i,t+1}=\begin{cases}\boldsymbol{u}_{i,t}, & \text{if}\quad f(u_{i,t})\leqslant f(x_{i,t})\\ \boldsymbol{x}_{i,t}, & \text{otherwise}\end{cases}\tag{11}$$

1.3　广义反向学习策略

广义反向学习[9-10]概念定义如下:

n 维空间中,设 $\tilde{\boldsymbol{x}}_i=(\tilde{x}_{i,1},\tilde{x}_{i,2},\cdots,\tilde{x}_{i,n})$ 是个体 $\boldsymbol{x}_i=(x_{i,1},x_{i,2},\cdots,x_{i,n})$ 的反向解,则 $\tilde{\boldsymbol{x}}_{i,j}(j=1,2,\cdots,n)$ 定义为

$$\tilde{\boldsymbol{x}}_{i,j}=k(a_j+b_j)-\boldsymbol{x}_{i,j}\tag{12}$$

式中,k 为(0,1) 之间的均匀分布随机数,个体 $\boldsymbol{x}_i$ 在第 j 维的动态搜索范围为[a_j,b_j]。若 $\tilde{\boldsymbol{x}}_{i,j}$ 取值超出搜索范围,则在[a_j,b_j] 内对其进行随机生成:

$$\tilde{\boldsymbol{x}}_{i,j}=\text{rand}(a_j,b_j)\tag{13}$$

由概率论可知,反向解 $\tilde{\boldsymbol{x}}_i$ 的适应值等概率优于原个体解 $\boldsymbol{x}_i$ 适应值[11],将广义反向学习移植到 DE 算法中,充分利用反向种群信息,在不增加原有搜索

空间前提下,极大地改进算法寻优效率,增加获得最终解的可能性。

基于广义反向学习的种群初始化伪代码如下:

算法 1 GOBL 种群初始化

随机产生初始化种群 $P_0 = (x_1, x_2, \cdots, x_{N_P})$;
for $i = 1$ to N_P do
  for $j = 1$ to n do
    $P_{\mathrm{GOP}_{ji,0}} = k * (a_j + b_j) - P_{ji,0}$;
    if $P_{\mathrm{GOP}_{ji,0}} < a_j \| P_{\mathrm{GOP}_{ji,0}} > b_j$ then
      $P_{\mathrm{GOP}_{ji,0}} = \mathrm{rand}(a_j, b_j)$;
    end if
  end for
end for
选取$\{P_{\mathrm{GOP}_0} \cup P_0\}$ 中前 N_P 个最优个体组成初始种群

基于广义反向学习的种群"代跳"伪代码如下:

算法 2 GOBL 种群"代跳"

if rand(0,1) < J_r then // J_r 为代跳率
  for $i = 1$ to N_P do
    for $j = 1$ to n do
    $P_{\mathrm{GOP}_{ji,t}} = k * (\min(P_{ji,t}) + \max(P_{ji,t})) - P_{ji,t}$;
  if $P_{\mathrm{GOP}_{ji,t}} < \min(P_{ji,t}) \| P_{\mathrm{GOP}_{ji,t}} > \max(P_{ji,t})$ then
      $P_{\mathrm{GOP}_{ji,t}} = \mathrm{rand}(\min(P_{ji,t}), \max(P_{ji,t}))$;
    end if
    end for
  end for
end
选取$\{P_{\mathrm{GOP}_t} \cup P_t\}$ 中前 N_P 个最优个体组成新种群

2 自适应约束差分进化算法

2.1 自适应权衡模型

约束优化问题中不能简单的将适应值等同于目标函数值,需要充分考虑约束条件对适应值的影响,为了使 CDE 算法更加合理,本文引用自适应权衡模型(adaptive trade-off model, ATM)机制[5]将种群划分为 3 种状态,分别计算其适应值。

2.1.1 不可行状态约束处理机制

在不可行状态下,约束处理机制不考虑目标函数值,仅计算约束违反程度,采用违约值代替适应值

$$f_{\mathrm{fitness}}(\boldsymbol{x}_i) = G(\boldsymbol{x}_i) \tag{14}$$

2.1.2 半可行状态约束处理机制

半可行状态约束处理机制目的在于将可行与不可行个体维持在合理的状态,平衡目标函数值与约束违反程度,本文采用适应值转换策略[12](adaptive fitness transformation, AFT)来实现该目标。

该策略将种群 P 划分为可行个体集合 $\boldsymbol{Z}_1$ 和不可行个体集合 $\boldsymbol{Z}_2$,分别表示为

$$\begin{aligned} \boldsymbol{Z}_1 &= \{i \mid G(x_i) = 0, i = 1, \cdots, N_P\} \\ \boldsymbol{Z}_2 &= \{i \mid G(x_i) > 0, i = 1, \cdots, N_P\} \end{aligned} \tag{15}$$

个体 $\boldsymbol{x}_i$ 的目标函数值 $f(\boldsymbol{x}_i)$ 根据下式进行转换

$$f'(\boldsymbol{x}_i) = \begin{cases} f(\boldsymbol{x}_i) & i \in \boldsymbol{Z}_1 \\ \max\{\phi \cdot f(\boldsymbol{x}_{\mathrm{best}}) + (1-\phi) \cdot f(\boldsymbol{x}_{\mathrm{worst}}), f(x_i)\} & i \in \boldsymbol{Z}_2 \end{cases} \tag{16}$$

式中,ϕ 为种群中可行个体所占比率,$f(\boldsymbol{x}_{\mathrm{best}})$,$f(\boldsymbol{x}_{\mathrm{worst}})$ 分别为 $\boldsymbol{Z}_1$ 中最优和最差目标函数值。

将目标函数值 $f(\boldsymbol{x}_i)$ 标准化

$$f_{\mathrm{nor}}(\boldsymbol{x}_i) = \frac{f'(\boldsymbol{x}_i) - \min\limits_{j \in Z_1 \cup Z_2} f'(\boldsymbol{x}_j)}{\max\limits_{j \in Z_1 \cup Z_2} f'(\boldsymbol{x}_j) - \min\limits_{j \in Z_1 \cup Z_2} f'(\boldsymbol{x}_j)}, i \in (1, \cdots, N_P) \tag{17}$$

将违约值 $G(x_i)$ 标准化

$$G_{\mathrm{nor}}(x_i) = \begin{cases} 0 & i \in \boldsymbol{Z}_1 \\ \dfrac{G(x_i) - \min\limits_{j \in Z_2} G(x_j)}{\max\limits_{j \in Z_2} G(x_j) - \min\limits_{j \in Z_2} G(x_j)} & i \in \boldsymbol{Z}_2 \end{cases} \tag{18}$$

则个体最终适应值可表示为

$$f_{\mathrm{final}}(\boldsymbol{x}_i) = f_{\mathrm{nor}}(\boldsymbol{x}_i) + G_{\mathrm{nor}}(\boldsymbol{x}_i),\ i \in \{1, \cdots, N_P\} \tag{19}$$

2.1.3 可行状态约束处理机制

该状态下的选择标准只取决目标函数值,即个体适应值由目标函数值确定:

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

2.2 改进自适应排序变异操作

2.2.1 排序值分配

为了充分利用种群中"精英"个体所携带的信息,对种群按适应值从最优到最差进行排序[13],个体 $\boldsymbol{x}_i$ 的排序值由下式表示:

$$R_i = N_P + 1 - i, \quad i = 1, \cdots, N_P \tag{21}$$

式中,N_P 为种群规模,i 为第 i 个个体在排序中的序号。

2.2.2 选择概率计算

约束优化问题中,个体选择概率需根据种群当前所处状态分别计算,不同状态下选择概率计算方

法有所差异。

不可行状态下选择概率计算为:

$$p_i = 0.5 \cdot \left[1 - \cos\left(\frac{R_i}{N_P} \cdot \pi\right)\right] \tag{22}$$

半可行状态下选择概率计算为:

$$p_i = 0.5 \cdot \left[1 - \cos\left(\frac{R_i}{N_P} \cdot \pi\right)\right] \tag{23}$$

可行状态下选择概率计算为:

$$p_i = \frac{\arccos\left(1 - 2 \cdot \frac{R_i}{N_P}\right)}{\pi} \tag{24}$$

式中, $i = 1, \cdots, N_P$。

在不可行状态和半可行状态下采用余弦模型计算选择概率,可行状态下采用反余弦模型计算选择概率,2 种模型选择概率与个体排序值之间的关系如图 1 所示(种群规模 $N_P = 50$)。

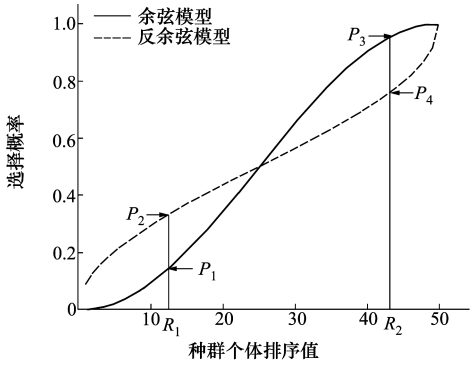

图 1　选择概率与个体排序值关系

如图 1 所示, R_1, R_2 为个体 x_1, x_2 的排序值且 $R_2 > R_1$,可以得到 $p_3 - p_1 > p_4 - p_2$,2 个体选择概率差在不同模型下具有明显差异。这意味着在余弦模型下较优个体比较差个体更占支配优势,在反余弦模型下,由于概率差别微小使得这种优势并不明显。

在不可行状态下,为了使种群更迅速到达可行域,具有较小违约值的占优个体应分配较大的选择概率,所以采用余弦模型计算选择概率。

在半可行状态下,重要的可行个体和不可行个体携带大量重要信息,因此获得较高的排序值。目标函数值小的可行个体能够引导算法找到全局最优解,而目标函数值小、约束违反程度轻的不可行个体能够加速算法搜寻到可行域,这些个体应更加注意,因此,同样采用余弦模型计算选择概率。

在可行状态下,为了避免算法出现早熟情况而陷入局部最优,采用反余弦模型进行计算,以使减小较差个体被支配性,增加其被选中几率,保持种群多样性。

2.2.3　变异操作

本文采用“improved rand-to-best and current/2”变异策略执行变异操作,该策略依据上代种群中可行个体比率 φ 将变异策略分为两部分:“rand-to-best and current/2”前 2 项向量基于选择概率选取,剩余向量依照随机规则选取;“rand-to-current/2”前 3 项向量基于选择概率选取,剩余向量依照随机规则选取。

对每个个体随机生成服从 $N(0,5,0.15)$ 的变异因子 F_i 和交叉因子 Cr_i,进化过程中自适应变异因子与交叉因子通过下式计算[14]

$$F_Z = F_{r1,G} + N(0,0.5) \cdot (F_{r2,G} - F_{r3,G}) \tag{25}$$

$$Cr_Z = Cr_{r1,G} + N(0,0.5) \cdot (Cr_{r2,G} - Cr_{r3,G}) \tag{26}$$

自适应排序变异伪代码如下:

算法 3　自适应排序变异伪代码

根据个体适应值对种群排序 R_i;

计算种群中每各个体的选择概率 p_i;

计算变异因子 F_i 和交叉因子 Cr_i;

if rand(0,1) $< \phi$ then

依据选择概率选取 r_1, r_2, r_3, r_4 且 $r_1 \neq r_2 \neq r_3 \neq r_4 \neq i$

$v_{i,t} = x_{1,t} + F_Z \cdot ((x_{r2,t} - x_{i,t}) + (x_{r3,t} - x_{r4,t}))$;

else

依据选择概率选取 r_1, r_2, r_3 且 $r_1 \neq r_2 \neq r_3 \neq i$

$v_{i,t} = x_{1,t} + Fz \cdot ((x_{rbest,t} - x_{r2,t}) + (x_{r3,t} - x_{i,t}))$;

end

在约束优化问题初期,进化种群中可能仅包含不可行个体,此时算法主要目标应尽快使种群接近到达可行域内,因此有必要利用种群中“精英”个体(适应值较低)信息进行变异操作,因此采用“rand-to-best and current/2”变异策略。随着进化过程的发展,种群中可行个体数量增多,若继续向“精英”个体学习,会导致种群多样性急剧下降,陷入局部最优状态,导致算法早熟,所以此时采用“rand-to-current/2”变异策略,在保证种群多样性的前提下提高种群的开发能力。

3　GOBL-ACDE 算法流程

基于广义反向学习的自适应约束差分进化算法

的伪代码如下：

算法 4 GOBL-ACDE 算法

输入：搜索空间 R，目标函数 f，违约函数 G，最大函数评价次数 $NFE_{\max}$

```
t = 1; // t 为进化代数
执行算法 1 生成初始种群;
while 不满足终止条件 do
    for i = 1 to N_P do
      for j = 1 to n do
       执行算法 3 生成 v_{i,j};
      if rand(0,1) < Cr_{i,j} or then
        u_{i,j} = v_{i,j}
      else        u_{i,j} = x_{i,j}
        end if
      end for
    for i = 1 to N_P do
      if f(u_i) ≤ f(x_i) then
        x_{i,t+1} = u_{i,t};
      else
        x_{i,t+1} = x_{i,t}
      end if
    end for
    执行算法 2 进行种群代跳;
    t = t + 1;
end while
```

输出：最优解

4 试验测试及结果分析

为测试 GOBL-ACDE 算法性能，验证其寻优能力，本文选取 CEC2006 中 13 个约束优化测试函数[15]进行试验，并与 CDE[16]、DDE[17]、A-DDE[18]、εDE[19]以及 DPDE[20]5 种算法进行性能对比，各算法均与原文献相同。

4.1 GOBL-ACDE 与其他算法性能对比

将 GOBL-ACDE 算法分别与上述 5 种算法进行比较，其中 GOBL-ACDE、CDE、DDE 以及 DPDE 分别独立运行 30 次，εDE 独立运行 50 次，A-DDE 独立运行 100 次，最大函数评价次数均为 200 000 次，各算法运行结果如表 1 所示，其中黑体表示为较优算法值。

根据表 1 各算法寻优精度指标[15]可知，GOBL-ACDE 算法在除函数 g06 之外的 12 个测试函数中，最优值均优于或等于其余 5 种算法。平均值或最差值结果中，GOBL-ACDE 算法在函数 g02 的最差值略差于 εDE 和 DPDE 算法，在函数 g03 的最差值略差于 DDE、A-DDE、εDE 和 DPDE 算法，在函数 g06 的平均值和最差值略差于 DDE、A-DDE 和 DPDE 算法。

函数 g02 由于可行域大，函数结构复杂，搜索难度大，试验主要考察算法寻优能力，在 GOBL-ACDE 算法寻得最优值的前提下，仅最差值的表现略差于 εDE 和 DPDE 算法，因此综合考虑算法成功率以及评价次数，我们完全可以认为 GOBL-ACDE 算法的表现不弱于 和 DPDE 算法，具有较强的搜索能力。对于函数 g06，GOBL-ACDE 算法已获得测试函数给出的标准值，虽然其结果略差于 DDE、A-DDE 和 DPDE 算法，但由于其最优值与标准值一致，我们仍可以认为 GOBL-ACDE 取得满意表现。

函数 g03，g05，g11 和 g13 为含有等式约束的测试函数，运行结果表明 GOBL-ACDE 算法综合表现完全优于其余 5 种算法，在所有 4 个测试函数中均寻得最优结果。其中，函数 g03 只有 GOBL-ACDE 算法寻得最优值，平均值和最差值方面同样优于其余 5 种算法；函数 g05 只有 GOBL-ACDE 和 DPDE 算法寻得最优结果，函数 g11 只有 GOBL-ACDE 和算法寻得最优结果，2 组测试中 GOBL-ACDE 算法的平均值和最差值均优于其余 4 种算法；GOBL-ACDE 和 DDE 算法在函数 g13 测试试验中均寻得最优结果，GOBL-ACDE 算法在平均值和最差值表现方面同样取得较好表现，且明显优于其余 5 种算法，DDE 算法虽寻得最优值，但平均值和最差值表现却并不令人满意。因此从综合表现来看，GOBL-ACDE 算法在处理含有等式约束的约束优化问题时，相较于其余 5 种算法在精确寻优方面具有更好的性能。

函数 g02 和 g03 为非线性，根据表 1 运算结果可知 GOBL-ACDE 算法在最优值、平均值和最差值三方面均优于 A-ADE 算法，表明 GOBL-ACDE 算法在处理非线性约束优化问题时较 A-ADE 算法更为出色。

函数 g05，g07，g10，g11 和 g13 不存在可行域，即在取得最优值时并不满足所有约束限制，在该条件下 CDE 算法仅在函数 g07 获得最优值，其余各指标性能均弱于 GOBL-ACDE 算法，这表明在处理无可行域约束优化问题时，GOBL-ACDE 算法较 CDE 算法具有明显优势。

算法结果稳定性性方面，在函数 g01，g02，g04，

g10,g11,g12,g13 测试中,GOBL-ACDE 算法的标准差均优于或等于其余 5 种算法。其中函数 g01 和 g02 为高维函数,函数 g01,g04,g11,g12 为二次函数,这表明 GOBL-ACDE 算法在处理高维和二次约束优化方面表现出较强的鲁棒性,算法寻优稳定性优于其余 5 种算法。

在含有等式约束的测试结果中,GOBL-ACDE 算法的标准差较 CDE 和 εDE 算法具有明显优势,其中函数 g03,g05 和 g13 较 CDE 算法分别相差 6,13 和 17 个数量级,较 εDE 算法分别相差 2,7 和 16 个数量级,这表明在等式约束下,GOBL-ACDE 算法较 CDE 和 εDE 算法鲁棒性更强。

表 1　算法运行结果

测试函数	指标	GOBL-ACDE	CDE	DDE	A-DDE	εDE	DPDE
g01	最优值	**-15.000 000**	**-15.000 000**	**-15.000 000**	**-15.000 000**	**-15.000 000**	**-15.000 000**
	平均值	**-15.000 000**	**-14.999 999**	**-15.000 000**	**-15.000 000**	**-15.000 000**	**-15.000 000**
	最差值	**-15.000 000**	-14.999999	**-15.000000**	**-15.000 000**	**-15.000 000**	**-15.000 000**
	标准差	**0.00**	5.1×10^{-7}	4.13×10^{-9}	9.54×10^{-7}	1.97×10^{-13}	**0.00**
g02	最优值	**-0.803 619 0**	**-0.803 619**	**-0.803 619**	-0.803 599	**-0.803 619**	-0.803 619
	平均值	**-0.803 612 1**	-0.731 136	-0.798 132	-0.783 149	-0.803 004	-0.802 350
	最差值	-0.785 939 2	-0.601 107	-0.761 511	-0.700 312	-0.786 215	**-0.795 326**
	标准差	**2.35×10^{-4}**	7.15×10^{-2}	9.9×10^{-3}	3.2×10^{-2}	4.96×10^{-3}	1.18×10^{-3}
g03	最优值	**-1.000 5**	-0.993 685	-1.000	-1.000	-1.000	-1.000
	平均值	**-1.000**	-0.801 376	**-1.000**	**-1.000**	**-1.000**	**-1.000**
	最差值	-0.999 999 94	-0.699 587	**-1.000**	**-1.000**	**-1.000**	**-1.000**
	标准差	3.026×10^{-8}	8.71×10^{-2}	**0.00**	8.74×10^{-12}	2.76×10^{-6}	**0.00**
g04	最优值	**-30 665.539**	**-30 665.539**	**-30 665.539**	**-30 665.539**	**-30 665.539**	-3 066.538 7
	平均值	**-30 665.539**	**-30 665.539**	**-30 665.539**	**-30 665.539**	**-30 665.539**	-3 066.538 7
	最差值	**-30 665.539**	**-30 665.539**	**-30 665.539**	**-30 665.539**	**-30 665.539**	**-3 066.538 7**
	标准差	**0.00**	**0.00**	**0.00**	4.68×10^{-12}	**0.00**	**0.00**
g05	最优值	**5 216.496 7**	5 126.570 92	5 126.497	5 126.497	5 126.498	**5 216.496 7**
	平均值	**5 216.4967**	5 197.335 98	5 126.497	5 126.497	5 126.498	**5 216.496 7**
	最差值	**5 216.4967**	5 327.390 49	5 126.497	5 126.497	5 126.498	**5 216.496 7**
	标准差	8.6×10^{-12}	9.352 5×10	**0.00**	4.47×10^{-11}	3.13×10^{-5}	**0.00**
g06	最优值	-6 961.813 8	-6 961.813 87	**-6 961.814**	**-6 961.814**	-6 961.813 87	**-6 961.814**
	平均值	-6 961.813 8	-6 961.813 87	**-6 961.814**	**-6 961.814**	-6 961.81387	**-6 961.814**
	最差值	-6 961.813 8	-6 961.813 87	**-6 961.814**	**-6 961.814**	-6 961.813 87	**-6 961.814**
	标准差	5.46×10^{-12}	**0.00**	**0.00**	1.98×10^{-12}	**0.00**	**0.00**
g07	最优值	**24.306 2**	**24.306 20**	**24.306**	**24.306**	**24.306 209**	**24.306**
	平均值	**24.306 2**	24.306 21	**24.306**	**24.306**	**24.306 209**	**24.306**
	最差值	**24.306 2**	24.306 22	**24.306**	**24.306**	**24.306 209**	**24.306**
	标准差	5.3×10^{-7}	4.13×10^{-6}	**9.59×10^{-9}**	7.38×10^{-5}	1.43×10^{-7}	6.25×10^{-6}

续表 1

测试函数	指标	GOBL-ACDE	CDE	DDE	A-DDE	εDE	DPDE
g08	最优值	**−0.095 825**	**−0.095 825**	**−0.095 825**	**−0.095 825**	**−0.095 825**	**−0.095 825**
	平均值	**−0.095 825**	**−0.095 825**	**−0.095 825**	**−0.095 825**	**−0.095 825**	**−0.095 825**
	最差值	**−0.095 825**	**−0.095 825**	**−0.095 825**	**−0.095 825**	**−0.095 825**	**−0.095 825**
	标准差	6.72×10^{-17}	**0.00**	**0.00**	3.88×10^{-12}	3.13×10^{-17}	**0.00**
g09	最优值	**680.630 0**	**680.630 057**	**680.630**	**680.630**	**680.630 057**	680.630 0
	平均值	**680.630 0**	**680.630 057**	**680.630**	**680.630**	**680.630 057**	680.630 0
	最差值	**680.630 0**	**680.630 057**	**680.630**	**680.630**	**680.630 057**	680.630 0
	标准差	2.49×10^{-14}	**0.00**	**0.00**	7.17×10^{-10}	1.32×10^{-12}	3.65×10^{-14}
g10	最优值	**7 049.248**	**7 049.248**	**7 049.248**	**7 049.248**	**7 049.248**	**7 049.248**
	平均值	**7 049.248**	7 049.249	7 049.316	**7 049.248**	**7 049.248**	**7 049.248**
	最差值	**7 049.248**	7 049.251	7 049.932	**7 049.248**	7 049.349	**7 049.248**
	标准差	$\mathbf{3.96\times10^{-10}}$	7.01×10^{-3}	6.15×10^{-2}	4.69×10^{-4}	7.31×10^{-5}	8.36×10^{-8}
g11	最优值	**0.749 999 9**	0.75	0.75	0.75	**0.749 9999 9**	0.75
	平均值	**0.749 999 9**	0.757 912	0.75	0.75	**0.749 999 99**	0.75
	最差值	**0.749 999 9**	0.796 386	0.75	0.75	**0.749 999 99**	0.75
	标准差	**0.00**	2.06×10^{-2}	**0.00**	4.17×10^{-12}	**0.00**	**0.00**
g12	最优值	**−1.000 000**	**−1.000 000**	**−1.000**	**−1.000**	**−1.000 000**	**−1.000 000**
	平均值	**−1.000 000**	**−1.000 000**	**−1.000**	−1.000	**−1.000 000**	**−1.000 000**
	最差值	**−1.000 000**	**−1.000 000**	**−1.000**	−1.000	**−1.000 000**	**−1.000 000**
	标准差	**0.00**	**0.00**	**0.00**	**0.00**	**0.00**	**0.00**
g13	最优值	**0.053 941**	0.059 003	0.053 941	0.539 42	0.053 941 5	0.053 941 5
	平均值	**0.053 941**	0.291 429	0.071 437	0.796 27	0.060 156 3	0.053 941 5
	最差值	**0.053 941**	0.397 278	0.491 253	0.438 80	0.176 562 1	0.053 941 5
	标准差	1.14×10^{-18}	1.73×10^{-1}	8.21×10^{-2}	8.06×10^{-2}	4.14×10^{-2}	1.16×10^{-17}

4.2 广义反向学习性能分析

为了对广义反向学习性能进行分析,本文分别对 GOBL-ACDE 和 ACDE 算法进行 13 个标准函数测试试验,2 种算法分别独立运行 30 次,最大函数评价次数为 200 000 次。

试验运行结果如表 2 所示,算法平均函数评价次数对比如表 3 所示,其次图 2 给出 2 种算法在函数 g01 和 g06 测试中的收敛图。

表 2　GOBL-ACDE 和 ACDE 运行结果

函数	GOBL-ACDE			ACDE		
	最优值	平均值	标准差	最优值	平均值	标准差
g01	**−15.000 000**	**−15.000 000**	**0.00**	**−15.000 000**	**−15.000 000**	1.54×10^{-11}
g02	**−0.803 619 0**	**−0.803 612 1**	$\mathbf{2.35\times10^{-3}}$	**−0.803 619 0**	−0.803 522 9	1.12×10^{-2}
g03	**−1.000**	**−1.000**	3.026×10^{-8}	**−1.000**	**−1.000**	**0.00**
g04	**−30 665.539**	**−30 665.539**	**0.00**	**−30 665.539**	**−30 665.539**	**0.00**
g05	**5 216.496 7**	**5 216.496 7**	$\mathbf{8.6\times10^{-12}}$	**5 216.496 7**	**5 216.496 7**	3.21×10^{-8}
g06	**−6 961.813 8**	**−6961.8138**	7.13×10^{-12}	**−6961.8138**	**−6961.8138**	$\mathbf{5.46\times10^{-12}}$
g07	**24.306 2**	**24.306 2**	$\mathbf{5.3\times10^{-7}}$	**24.306 2**	**24.306 2**	$\mathbf{5.3\times10^{-7}}$
g08	**−0.095 825**	**−0.095 825**	$\mathbf{6.72\times10^{-17}}$	**−0.095 825**	**−0.095 825**	$\mathbf{5.42\times10^{-16}}$
g09	**680.630 0**	**680.630 0**	$\mathbf{2.49\times10^{-14}}$	**680.6300**	**680.6300**	$\mathbf{3.76\times10^{-12}}$
g10	**7 049.248**	**7 049.248**	$\mathbf{3.96\times10^{-10}}$	**7 049.248**	**7 049.248**	4.37×10^{-8}
g11	**0.749 999 9**	**0.749 999 9**	**0.00**	**0.749 999 9**	**0.749 999 9**	**0.00**
g12	**−1.000 000**	**−1.000 000**	**0.00**	**−1.000 000**	**−1.000 000**	**0.00**
g13	**0.053 941**	**0.053 941**	$\mathbf{1.14\times10^{-8}}$	**0.053 941**	**0.053 941**	6.72×10^{-7}

表 3　GOBL-ACDE 和 ACDE 平均函数评价次数对比

函数	g01	g02	g03	g04	g05	g06	g07	g08	g09	g10	g11	g12	g13
GOBL-ACDE	**50 060**	**96 770**	**35 120**	**19 040**	**36 400**	10 230	**49 060**	**1 950**	**18 050**	**76 390**	**19 700**	**3 140**	**46 680**
ACDE	73 470	113 640	43 660	35 730	42 960	**9 060**	61 170	2 800	19 500	89 880	29 040	3 720	51 040

由表 2 可知,GOBL-ACDE 与 ACDE 算法在 13 个测试函数中均获得最优结果,在寻优性方面 2 种算法表现相近,其中 ACDE 算法仅在函数 g02 平均值略差于 GOBL-ACDE 算法。除 g03 和 g06,GOBL-ACDE 算法在其余 11 个函数测试中标准差均较小,显示了较好的稳定性、鲁棒性。

由表 3 可以看出 GOBL-ACDE 算法在 13 个测试函数的 12 个中,平均函数评价次数均小于 ACDE 算法,从整体来看 GOBL-ACDE 算法在收敛速度方面优于 ACDE 算法。

图 2 中 a)、b)分别 GOBL-ACDE 与 ACDE 算法在函数 g01,g06 测试中的收敛图,其中实线代表 GOBL-ACDE 算法,虚线代表 ACDE 算法。

如图 2a)所示,GOBL-ACDE 算法在函数 g01 测试中,种群从不可行状态过渡到半可行状态、从半可行状态过渡到可行性状态以及获得全局最优解的函数评价次数均小于 ACDE 算法,收敛速度明显优于 ACDE 算法。

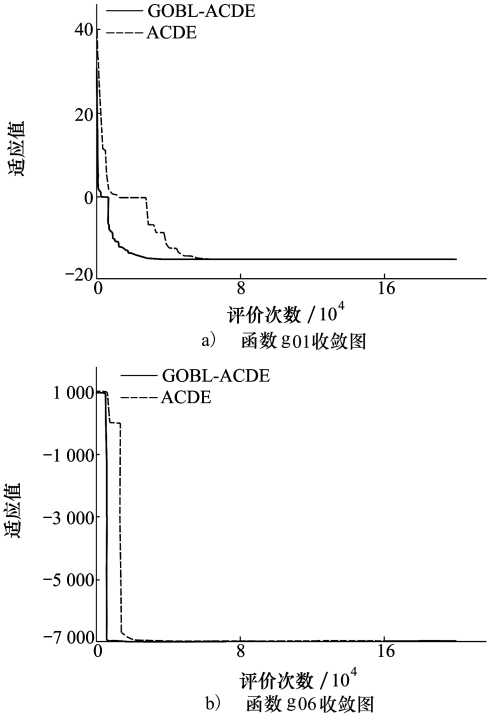

图 2　GOBL-ACDE 与 ACDE 收敛性比较图

由图 2b)可知,GOBL-ACDE 算法在函数 g06 测试中,种群跳过半可行状态,直接从不可行状态过渡至可行状态,这是由于在进化过程中,上代种群完成交叉、变异、选择后执行“代跳”操作使得种群个体发生突变,促使后代个体全部满足约束条件转至可行状态,加速收敛速度直至 GOBL-ACDE 算法寻得全局最优解。ACDE 算法在进化过程不存在种群"代跳",因此种群从不可行状态过渡到半可行状态、从半可行状态过渡到可行性状态以及获得全局最优解的收敛速度慢于 GOBL-ACDE 算法。

上述分析表明 GOBL-ACDE 算法的收敛性明显优于 ACDE 算法,广义方向学习机制对算法收敛速度具有明显的提升。

4.3 改进自适应排序变异操作性能分析

为了分析改进自适应排序变异操作性能,本文对 GOBL-ACDE 和 GOBL-CDE 算法进行对比,其中 GOBL-CDE 算法采用“DE/rand/2”变异策略,随机选取 $r_i(i=1,2,\cdots,5)$,选取 $F=0.8$,$Cr=0.9$,2 种算法分别独立运行 30 次,最大函数评价次数为 200 000次。算法平均函数评价次数对比如表 4 所示,图 3 给出 2 种算法在函数 g04 和 g11 测试中的收敛图。

表 4 GOBL-ACDE 和 GOBL-CDE 平均函数评价次数对比

函数	g01	g02	g03	g04	g05	g06	g07	g08	g09	g10	g11	g12	g13
GOBL-ACDE	**50 060**	**96 770**	**35 120**	**19 040**	**36 400**	**10 230**	**49 060**	**1 950**	**18 050**	**76 390**	**19 700**	**3 140**	**46 680**
GOBL-CDE	86 130	104 840	39 570	41 680	37 090	19 620	83 120	3 020	33 840	144 090	49 300	5 270	47 500

由表 4 可以看出,GOBL-ACDE 算法在 13 个测试函数中,平均函数评价次数均优于 GOBL-CDE 算法,这说明相较于采用单一“DE/rand/2”变异策略,改进自适应排序变异操作在寻优过程中具有更加出色的收敛性,提高了算法的收敛速度。

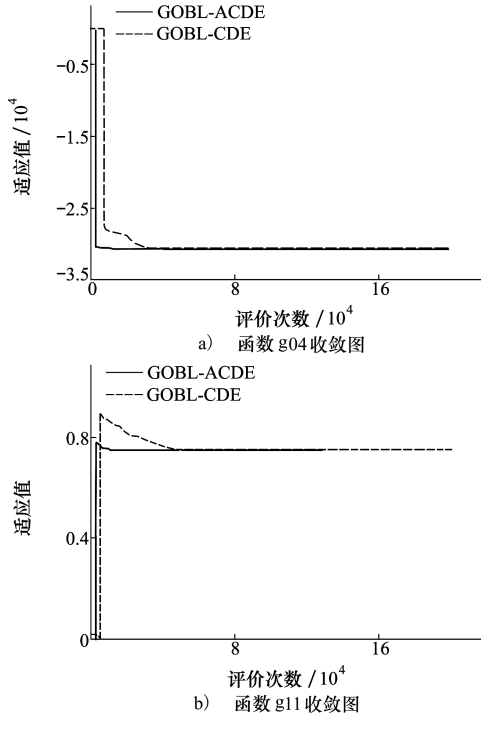

图 3 GOBL-ACDE 与 ACDE 收敛性比较图

图 3 中 a)、b)分别为 GOBL-ACDE 与 GOBL-CDE 算法在函数 g04、g11 测试中的收敛图,其中实线代表 GOBL-ACDE 算法,虚线代表 GOBL-CDE 算法。

如图 3 所示,在函数 g04、g11 测试中 GOBL-ACDE 算法均以较小评价次数到达可行状态并获得全局最优解,而 GOBL-CDE 算法在评价次数方面表现较差,收敛速度缓慢。函数 g11 测试中 GOBL-CDE 与 GOBL-ACDE 算法在收敛速度上有明显差距,这说明 GOBL-ACDE 算法在含有等式约束的优化问题中具有更好的处理能力。GOBL-ACDE 算法在变异操作中充分利用“精英”个体携带的信息,根据种群可行个体所占比率自适应调节变异策略,在保证种群多样性的前提下提高了种群收敛速度,验证了改进自适应排序变异操作的有效性

5 结 论

本文提出了一种基于广义反向学习的自适应约束差分进化算法,本算法通过生成反向种群完成种群初始化,进化过程中,通过执行"代跳"操作引导算法跳离局部最优状态,避免算法早熟,提高种群多样性。其次,采用自适应权衡模型将进化过程中的种群状态分为 3 类,并分别计算其相应适应值。最后,根据改进自适应排序变异操作对种群内个体进行排序,依据可行个体比率调节变异策略,提高算法

动态性能,完成种群进化。本文通过对 GOBL-ACDE 与 CDE、DDE、A-DDE、和 DPDE 5 种算法性能进行测试对比,结果显示 GOBL-ACDE 在寻优精确能力及稳定性方面具有较好表现,最后通过对广义反向学习机制和改进自适应排序变异操作分析,验证了其对算法收敛性的改进。

参考文献:

[1] MOHAMED A W, SABRY H Z. Constrained Optimization Based on Modified Differential Evolution Algorithm[J]. Information Sciences, 2012, 194(5): 171-208

[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] TAKAHAMA T, SAKAI S. Constrained Optimization by the ε Constrained Differential Evolution with Gradient-Based Mutation and Feasible Elites[C]//IEEE Congress on Evolutionary Computation, 2006:1-8

[4] MALLIPEDDI R, SUGANTHAN P N. Ensemble of Constraint Handling Techniques[J]. IEEE Trans on Evolutionary Computation, 2010, 14(4): 561-579

[5] WANG Y, CAI Z. Constrained Evolutionary Optimization by Means of $(\mu+\lambda)$-Differential Evolution and Improved Adaptive Trade-Off Model[J]. Evolutionary Computation, 2014, 19(2): 249-285

[6] JIA G, WANG Y, CAI Z, et al. An Improved $(\mu+\lambda)$-Constrained Differential Evolution for Constrained Optimization[J]. Information Sciences, 2013, 222(4): 302-322

[7] GONG W, CAI Z, LIANG D. Engineering Optimization by Means of an Improved Constrained Differential Evolution[J]. Comput Methods Appl Mech Engrg, 2014, 268: 884-904

[8] WANG H, RAHNAMAYAN S, WU Z J. Parallel Differential Evolution with Self Adapting Control Parameters and Generalized Opposition-Based Learning for Solving High-Dlimensional Optimization Problems[J]. Journal of Parallel and Distributed Computing, 2013,73(1): 62-73

[9] TIZHOOSH H R. Opposition-Based Learning: a New Scheme for Machine Intelligence[C]//The IEEE International Conference of Intelligent for Modeling, Control and Automation, 2005: 695-701

[10] WANG H, WU Z J, RAHNAMAYAN S, et al. Enhancing Particle Swarm Optimization Using Generalized Opposition-Based Learning[J]. Information Sciences, 2011,181: 4699-4714

[11] RAHNAMAYAN S, TIZHOOSH H R, SALAMA M M A. Opposition Versus Randomness in Soft Computing Techniques[J]. Soft Comput, 2008, 8(2): 906-918

[12] WANG Yong, CAI Zixing, ZHOU Yuren. An Adaptive Trade-Off Model for Constrained Evolutionary Optimization[J]. IEEE Trans on Evolutionary Computation, 2008, 12(1): 80-92

[13] GONG W, CAI Z, LIANG D. Adaptive Ranking Mutation Operator Based Differential Evolution for Constrained Optimization[J]. IEEE Trans on Cybernetics, 2015, 45(4): 716-727

[14] ELSAYED S M, SARKER R A, ESSAM D L. Multi-Operator Based Evolutionary Algorithms for Solving Constrained Optimization Problems[M]. Computers and Operations Research, 2011, 38(2): 1877-1896

[15] 合大海,李元香,龚文,等. 一种求解约束优化问题的自适应差分进化算法[J]. 电子学报, 2016, 44(10): 2535-2542
HE Dahai, LI Yuanxiang, GONG Wen, et al. An Adaptive DifferentiaI EvoIution AIgorithm for Constrained Optimization ProbIems[J]. Acta Electronica Sinica, 2016, 44(10): 2535-2542 (in Chinese)

[16] BECERRA R L, COELLO C A C. Cultured Differential Evolution for Constrained Optimization[J]. Computer Methods in Applied Mechanics & Engineering, 2006, 195(33/34/35/36): 4303-4322

[17] EFRÉN MEZURA-MONTES, COELLO C A C. Promising Infeasibility and Multiple Offspring Incorporated to Differential Evolution for Constrained Optimization[C]//Proceedings of Genetic and Evolutionary Computation, Washington DC, USA, 2005: 225-232

[18] MEZURAMONTES E, PALOMEQUEORTIZ A G. Self-Adaptive and Deterministic Parameter Control in Differential Evolution for Constrained Optimization[J]. 2009, 198: 95-120

[19] 郑建国,王翔,刘荣辉,等. 求解约束优化问题的 εDE 算法[J]. 软件学报, 2012,23(9): 2374-2387

ZHENG Jianguo, WANG Xiang, LIU Ronghui, et al. Dlif Terential Evolution Algorithm for Constrained Optimization Problems [J]. Journal of Software, 2012, 23(9): 2374-2387 (in Chinese)

[20] GAO W F, YEN G G, LIU S Y. A Dual-Population Differential Evolution with Coevolution for Constrained Optimization[J]. IEEE Trans on Cybernetics, 2015, 45(5): 1094-1107

Adaptive Constrained Differential Evolution Algorithm by Using Generalized Opposition-Based Learning

WU Wenhai[1], GUO Xiaofeng[1], ZHOU Siyu[1], LIU Jintao[2]

(1.Department of Aviation Control and Command, Qingdao Campus, Naval Aviation University, Qingdao 266041, China;
2.College of Command and Control Engineering, Army Engineering University, Nanjing 210002, China)

Abstract: Differential evolution is a global optimization algorithm based on greedy competition mechanism, which has the advantages of simple structure, less control parameters, higher reliability and convergence. Combining with the constraint-handling techniques, the constraint optimization problem can be efficiently solved. An adaptive differential evolution algorithm is proposed by using generalized opposition-based learning (GOBL-ACDE), in which the generalized opposition-based learning is used to generate initial population and executes the generation jumping. And the adaptive trade-off model is utilized to handle the constraints as the improved adaptive ranking mutation operator is adopted to generate new population. The experimental results show that the algorithm has better performance in accuracy and convergence speed comparing with CDE, DDE, A-DDE and . And the effect of the generalized opposition-based learning and improved adaptive ranking mutation operator of the GOBL-ACDE have been analyzed and evaluated as well.

Keywords: constrained optimization; differential evolution; generalized opposition-based learning; adaptation; trade-off model; ranking mutation;