一种基于局部 Lipschitz 下界估计支撑面的差分进化算法

周晓根 张贵军 郝小虎 俞 立
(浙江工业大学信息工程学院 杭州 310023)

摘 要 为了减少智能优化算法求解复杂问题时所需的目标函数评价次数,降低算法计算代价,在差分进化算法框架下,结合 Lipschitz 估计理论,提出一种基于局部 Lipschitz 下界估计支撑面的差分进化算法. 首先,对新个体的 N 邻近个体构建 Lipschitz 下界估计支撑面,进而通过支撑面获取新个体的下界估计值;然后,根据下界估计值设计 Lipschitz 估计选择策略来指导种群更新;其次,利用下界估计区域的极值信息排除部分无效区域,逐步缩小搜索区域;最后,根据 N 邻近个体下降方向和主导支撑面下降方向设计广义下降方向做局部增强. 数值实验结果表明,所提算法与文中给出的主流算法相比,能够以较少的目标函数评价次数获得高质量的最优解.

关键词 差分进化;智能优化算法;Lipschitz 下界估计;全局优化;支撑面
中图法分类号 TP18 **DOI 号** 10.11897/SP.J.1016.2016.02631

Differential Evolution Algorithm Based on Local Lipschitz Underestimate Supporting Hyperplanes

ZHOU Xiao-Gen ZHANG Gui-Jun HAO Xiao-Hu YU Li
(*College of Information Engineering, Zhejiang University of Technology, Hangzhou* 310023)

Abstract To reduce the number of function evaluations required to find optimal solutions of computationally expensive problems for intelligent optimization algorithms, a new algorithm which introduces the Lipschitz underestimate theory into basic differential evolution algorithm is proposed, referred to as differential evolution algorithm based on local Lipschitz underestimate supporting hyperplanes. Firstly, the Lipschitz underestimate supporting hyperplanes are constructed for the N neighboring individuals of the trial individual to obtain the underestimate value of the trial individual. Secondly, the Lipschitz underestimate selection strategy that designed according to the underestimate value of the trial individual is used to guide the population updating process. Thirdly, by excluding the invalid regions where the global optimum solution cannot be found according to the extremum information of the underestimate region, the search domain narrows gradually. Finally, the generalized descent direction based on the descent directions of the N neighboring individuals and the dominant supporting hyperplane is designed for local enhancement. Experiments results show that the proposed algorithm can obtain better quality optimal solutions with less number of function evaluations compare with the main-stream algorithms given in this paper.

Keywords differential evolution; intelligent optimization algorithms; Lipschitz underestimate; global optimization; supporting hyperplanes

收稿日期:2015-05-28;在线出版日期:2015-11-05. 本课题得到国家自然科学基金(61075062,61573317)、浙江省自然科学基金(LY13F030008)、浙江省科技厅公益项目(2014C33088)、浙江省重中之重学科开放基金资助项目(20151008,20151015)资助.**周晓根**,男,1987 年生,博士研究生,中国计算机学会(CCF)学生会员,主要研究方向为智能信息处理、优化理论及算法设计. E-mail: zxg@zjut.edu.cn. **张贵军**(通信作者),男,1974 年生,博士,教授,博士生导师,中国计算机学会(CCF)会员,主要研究领域为智能信息处理、优化理论及算法设计、生物信息学. E-mail: zgj@zjut.edu.cn. **郝小虎**,男,1990 年生,硕士研究生,主要研究方向为智能信息处理、生物信息学. **俞 立**,男,1961 年生,博士,教授,博士生导师,主要研究领域为智能控制、分布式控制和网络控制系统.

1 引 言

近年来,智能优化算法被广泛用于各种实际应用问题的求解.典型的智能优化算法包括遗传算法(Genetic Algorithm,GA)[1]、差分进化算法(Differential Evolution,DE)[2]、粒子群算法(Particle Swarm Optimization,PSO)[3]以及蚁群算法(Ant Colony Optimization,ACO)[4],这些算法不仅鲁棒性强,而且实用性强,被成功应用于电力、化工、通信以及网络等领域[5-7].然而,绝大多数智能优化算法面临的一个主要问题就是求解时需要大量的目标函数评价次数,从而导致计算代价极高.尤其对于一些实际优化问题,由于仿真模型运行时间的限制,在求解过程中,对新产生的候选解进行目标函数评价极其费时.例如,对于一个二维粗水动力仿真模型,运行一次可能耗时一分钟左右,对于一个完整的三维水动力模型运行一次可能耗时几分钟,甚至达到几小时[8].又如蛋白质结构预测问题中,对能量函数评价时有时需要调用第三方能量包,从而导致评价一次需要几秒,甚至达到几分钟.因此,对于这些目标函数计算复杂的实际优化问题,如何减少求解过程中所需的目标函数评价次数,降低算法计算代价是一个极其重要的问题,也是计算机科学领域面临的一个亟需解决的问题.

针对上述问题,国内外学者相继提出了各种解决方法,其中代理模型(Surrogate Model)[9]和 k-近邻预测(k-Nearest Neighbor Predictor)[8]为两种最常用的方法.代理模型方法利用代理模型来代替计算代价昂贵的目标函数评价.基于此方法,Liu 等人[10]提出了一种基于高斯代理模型的进化算法(Gaussian Process Surrogate Model Assisted Evolutionary Algorithm for Medium-scale Computationally Expensive Optimization Problems,GPEME),利用一种降维技术将原优化模型映射到一个低维空间,并通过一种代理意识模型搜索机制使得算法集中搜索希望较大的子区域.Zhou 等人[11]提出了一种基于多代理的文化基因算法,在进化过程中,通过精确插值代理模型来提高算法的局部搜索能力,以降低算法的计算代价.Lim 等人[12]提出了一种广义代理进化算法,利用各种不同代理模型预测值的加权和来指导种群进化,并通过各代理模型的不确定性预测来自适应调整加权值.上述算法最大的优点就是代理模型的计算复杂度远低于原目标问题,因此可以有效地降低计算代价.然而,没有一个通用的代理模型可以适用于所有问题,代理模型的选择直接影响着算法的性能[9].

k-近邻预测方法利用与个体最近的训练样本来预测目标函数值.基于此方法,Liu 等人[8]提出了一种基于 k-近邻预测的差分进化算法(Differential Evolution using k-Nearest Neighbour Predictor,DE-kNN),在进化前期,将所有个体作为训练样本进行目标函数评价,而在进化后期,利用 k-近邻预测个体的目标函数值,同时从新种群中选取部分较优个体更新训练样本.Park 等人[13]提出了一种基于加速 k-近邻预测的差分进化算法(Differential Evolution using Speed-up k-Nearest Neighbour Estimator,DE-EkNN),该算法与 Liu 等人[8]提出的算法最大的区别就是通过训练样本的加权平均值来预测个体的目标函数值,并有选择性的保存样本.Pham[14]提出了一种基于近邻比较的差分进化算法,在进化过程中,根据新个体的近邻预测值与目标个体的函数值比较来判断是否需要对新个体评价.上述算法的最大优点就是结构简单,但是其面临的主要问题就是需要内存来保存大量的训练样本,随着问题维数的增大,所需内存也急剧上升,因此,k-近邻预测算法并不适用于高维问题[14].

Lipchistz 估计方法[15]作为一种确定性方法,不需要进行模型选择和样本训练,而是利用一系列支撑函数建立原目标函数的下界估计,并通过不断增加支撑函数的数量使得下界估计不断逼近原目标函数,从而使得支撑函数的全局最优解不断向原目标函数的全局最优解收敛.因此,可以通过高效枚举下界估计的局部最优解得到原目标函数的全局最优解.Lipchistz 估计方法现已被应用于各种问题,如分子结构预测[16]、拟凸规划[17]和半无限规划[18]等.然而,为了得到更加精确的最优解,Lipchistz 估计方法需要使用大量的支撑函数来逼近原目标函数,从而导致极高的空间复杂度,因此,Lipchistz 估计方法仅适用于维数小于 10 的问题[19-20].

为了减少智能优化算法的函数评价次数,论文作者在群体算法框架下,结合抽象凸理论[21-22],提出了一种基于抽象凸下界估计的群体全局优化算法(Abstract Convex Underestimate Population-based Algorithm,ACUP)[23],该算法首先对整个初始种群构建抽象凸下界支撑面,然后利用不断收紧的下界信息来指导种群更新,最后根据进化信息更新下界支撑面.然而,对于一些复杂的高维优化问题,空间复杂度问题需要改进.因此,在此基础上,进一步引入局部抽象凸估计策略,提出了一种基于抽象凸

下界估计选择策略的差分进化算法(Differential Evolution Algorithm Based on Abstract Convex Underestimate Selection Strategy, DEUS)[24]，该算法通过提取新个体的邻近个体建立局部抽象凸下界松弛模型，进而利用下界松弛模型估计目标函数值来指导种群更新，并根据下界极值信息排除部分无效区域，同时借助支撑面的下降方向实现局部增强. 实验结果表明，ACUP 和 DEUS 算法能够有效地降低算法计算代价，提高算法性能. 然而，在使用 ACUP 和 DEUS 算法求解时，需要将原目标函数转化到单位单纯形约束条件下，并且需要对原目标函数加上常数 C 转化为正齐次递增函数(Increasing Positively Homogeneous Function of Degree One, IPH)[25]，由于目标函数曲面的粗糙复杂，不同的区域需要的常数 C 不同，所以 C 的值有时需要设置得异常的大，导致算法获得的下界估计值与实际目标函数值的误差较大，从而影响算法的性能.

本文在 DE 算法框架下，结合 Lipschitz 估计理论，对种群中的局部个体(新个体的 N 邻近个体)构建 Lipschitz 下界估计支撑面，进而通过支撑面获取目标函数的下界估计信息来指导种群进化，从而提出一种基于局部 Lipschitz 下界估计支撑面的差分进化算法(Differential Evolution Algorithm Based on Local Lipschitz Underestimate Supporting Hyperplanes, DELLU). 本文工作与文献[24]的主要区别在于：(1) 采用的 Lipschitz 估计支撑函数不需要通过模型转换将原目标函数转化为单位单纯形约束条件下的 IPH 函数，因此能够获得更加精确的下界估计值；(2) 设计了一种局部 Lipschitz 估计选择策略，在种群更新环节，根据新个体的下界估计值分 3 种情形来判断是否需要对新个体进行函数评价，有效地减少不必要的函数评价次数；(3) 根据邻近个体下降方向和主导支撑面下降方向提出了一种局部增强广义下降方向，利用此方向对新个体作局部增强，从而加快算法的收敛速度.

2　预备知识

2.1　基本定义

考虑如下全局优化问题

$$\min f(x),\ x\in D \tag{1}$$

其中，$x=(x_1,x_2,\cdots,x_n)$ 为 n 维优化变量，D 为可行域空间，$f(x)$ 为定义在可行域 D 上的目标函数，在可行域空间 D 中可能存在着多个全局最优解和大量的局部最优解.

定义 1.　若函数 $f:x\to R$ 对于任意的 $x^1,x^2\in D$ 都满足

$$|f(x^1)-f(x^2)|\le M\|x^1-x^2\| \tag{2}$$

则称函数 f 关于可行域 D Lipschitz 连续，其中，M 称为 Lipschitz 常数.

定义 2.　若存在函数族 $U\subseteq H$，使得函数 f 满足

$$f(x)=\sup\{h(x):h\subset U\},\ \forall x\in D \tag{3}$$

其中，H 为定义在可行域 D 上的函数族，则称函数 f 是关于函数族 H 的抽象凸函数(H-convex).

定义 3.　设 H 为定义在可行域 D 上的函数族，函数 f 为 H-convex 函数，则称

$$\partial_H f(x)=\{h\in H:h(y)\le f(y),f(x)=h(x)\},\ \forall y\in D \tag{4}$$

为函数 f 在点 x 处的 H-次微分(H-subgradient).

定义 4.　考虑一个由 $r=n+1$ 个半空间相交形成的有限凸多边形 $P=\bigcap_{j=1}^{r}\{x:\langle x,h_j\rangle\le 1\}$，其中

$$\begin{aligned}h_1&=(-1,0,0,\cdots)\\h_2&=(0,-1,0,\cdots)\\&\vdots\\h_n&=(0,\cdots,0,-1)\\h_{n+1}&=(1,1,\cdots,1)\end{aligned},$$

则称

$$d_P(x^1,x^2)=\max\left\{\max_{j=1,\cdots,r}(x_j^1-x_j^2),\sum_{j=1}^{n}(x_j^2-x_j^1)\right\} \tag{5}$$

为点 x^1 和 x^2 之间关于凸多边形 P 的单纯形距离.

2.2　DEUS 算法

定义 5.　设种群 $P=\{x_1,x_2,\cdots,x_{NP}\}$，$\boldsymbol{x}_{\text{trial}}$ 为新生成个体，则称与 $\boldsymbol{x}_{\text{trial}}$ 之间欧氏距离最近的 N 个种群个体为 $\boldsymbol{x}_{\text{trial}}$ 的 N 邻近个体，其中 NP 为种群规模，欧氏距离为

$$d(x_i,\boldsymbol{x}_{\text{trial}})=\sqrt{\sum_{j=1}^{n}(x_{i,j}-x_{\text{trial},j})^2} \tag{6}$$

假设式(1)的目标函数 $f(x)$ 满足定义 1，为了建立 $f(x)$ 的抽象凸下界估计松弛模型，首先需要根据线性变换式(7)将原目标函数 $f(x)$ 转化到单位单纯形空间 $S(S\equiv\{x'\in R_+^{n+1},x_i'\ge 0,\sum_{i=1}^{n+1}x_i'=1\})$ 中.

$$\begin{cases}x_i'\equiv(x_i-a_i)\Big/\sum_{i=1}^{n}(b_i-a_i)\\x_{n+1}'\equiv 1-\sum_{i=1}^{n}x_i'\end{cases},\ i=1,2,\cdots,n \tag{7}$$

其中，a_i 和 b_i 分别为 x_i 的下界和上界. 通过上述线性变换，原目标函数被转换为

$$\min \bar{f}(x'),\ x' \in S \subset R_+^{n+1} \tag{8}$$

然后，进一步对式(8)的目标函数加上常数 $C(C \geq 2M, M$ 为 Lipshitz 常数)将其转化为 IPH 函数，即 $\bar{g}=\bar{f}+C$.

经过上述模型转换过程，当新个体 $\boldsymbol{x}_{\text{trial}}$ 产生后，根据式(6)提取新个体的 N 邻近个体 $(x'^k, \bar{g}(x'^k))$，$k=1,2,\cdots,N$ 可以建立式(9)所示的局部抽象凸下界估计松弛模型

$$H^N(x')=\max_{k\leq N}\min_{i=1,\cdots,n+1} l_i^k x_i' \tag{9}$$

第 k 个支撑函数为

$$h^k(x')=\min_{i=1,\cdots n+1} l_i^k x_i'=\bar{g}(x'^k)\min\left\{\frac{x_1'}{x_1'^k},\cdots,\frac{x_{n+1}'}{x_{n+1}'^k}\right\} \tag{10}$$

其中

$$\boldsymbol{l}^k=\left(\frac{\bar{g}(x'^k)}{x_1'^k},\frac{\bar{g}(x'^k)}{x_2'^k},\cdots,\frac{\bar{g}(x'^k)}{x_{n+1}'^k}\right) \tag{11}$$

为点 $(x'^k, \bar{g}(x'^k))$ 的抽象凸下界估计支撑向量.

建立了局部抽象凸下界估计松弛模型以后，可以获取目标函数的下界信息来指导种群进化. 如图 1 所示，假设 A 为新个体，B 为目标个体，C 和 D 为 A 的 N 邻近个体，根据式(11)对个体 C 和 D 建立抽象凸下界支撑面，从而可以根据式(9)计算出新个体 A 的下界估计值 $\bar{y}_A$，通过 $\bar{y}_A$ 与目标个体 B 的函数值比较来判断是否需要对新个体 A 进行目标函数评价，有效减少目标函数评价次数；其次，通过比较下界估计区域的极值 $d_{\min}$ 与当前种群的最优值来判断下界估计区域所对应的优化区域(即 C 和 D 之间的区域)是否为无效区域；最后，根据抽象凸下界估计支撑面的下降方向获取下界估计区域的极值解作局部增强，即判断下界估计区域的极值解在目标函数上对应的点 E 是否优于新个体 A.

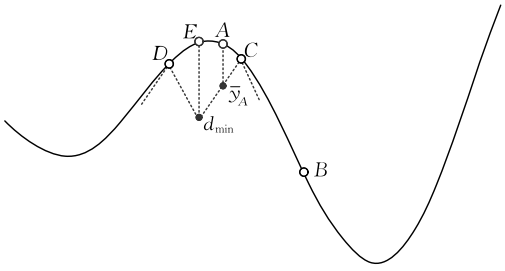


图 1 DEUS 算法的局部抽象凸估计选择策略示意图

3 DELLU 算法

在 DE 算法中，为了判断产生的新个体是否优于目标个体，算法需要对每次迭代产生的新个体进行目标函数评价，然而，对于一些实际优化问题，由于其仿真模型的复杂性，导致其目标函数评价极其费时. 为了减少 DE 算法的函数评价次数，降低算法计算代价，同时提高算法性能，本文结合 Lipschitz 估计理论，通过对新个体的 N 邻近个体构建 Lipschitz 下界估计支撑面来指导种群进化.

3.1 局部 Lipschitz 估计选择策略

假设式(1)的目标函数 $f(x)$ 满足定义 1，则根据 Lipschitz 估计理论可知[16]，函数 f 是关于以下支撑函数的 H-convex 函数

$$h^k(x)=f(x^k)-M\|x-x^k\| \tag{12}$$

其中，M 为 Lipschitz 常数. 由于任一 Lipschitz 函数的 H-次微分不为空[20]，则式(12)的支撑函数可进一步转化为

$$h^k(x)=f(x^k)-Md_P(x,x^k) \tag{13}$$

其中，$d_P(x,x^k)$ 为点 x 和 x^k 之间的单纯形距离.

通过引入松弛变量 $x_{n+1}=1-\sum_{j=1}^{n}x_j$，并令 $I=\{1,2,\cdots,n+1\}$，可将单纯形距离 $d_P(x,x^k)$ 简化为

$$d_P(x,x^k)=\max_{j\in I}(x_j^k-x_j) \tag{14}$$

则式(13)的支撑函数可简化为

$$\begin{aligned} h^k(x)&=\min_{j\in I}(f(x^k)-M(x_j^k-x_j))\\ &=\min_{j\in I}M(l_j^k+x) \end{aligned} \tag{15}$$

其中

$$\boldsymbol{l}^k=\left(\frac{f(x^k)}{M}-x_1^k,\frac{f(x^k)}{M}-x_2^k,\cdots,\frac{f(x^k)}{M}-x_{n+1}^k\right) \tag{16}$$

称为点 x^k 处的 Lipschitz 下界估计支撑向量.

经过变异、交叉操作生成新个体 $\boldsymbol{x}_{\text{trial}}$ 后，提取 $\boldsymbol{x}_{\text{trial}}$ 的 N 邻近个体 $x_{\text{nb}}^t(t=1,\cdots,N)$ 建立 Lipschitz 下界估计支撑向量 $\boldsymbol{l}_{\text{nb}}^t(t=1,\cdots,N)$，则根据 N 邻近个体的支撑向量可以计算出 $\boldsymbol{x}_{\text{trial}}$ 的下界估计值 $\bar{y}_{\text{trial}}$，即

$$\begin{aligned} \bar{y}_{\text{trial}}&=H^N(\boldsymbol{x}_{\text{trial}})=\max_{t\leq N}h^t(\boldsymbol{x}_{\text{trial}})\\ &=\max_{t\leq N}\min_{j\in I}M(l_{\text{nb},j}^t+x_{\text{trial},j}) \end{aligned} \tag{17}$$

其中，$l_{\text{nb},j}^t=f(x_{\text{nb}}^t)/M-x_{\text{nb},j}^t$.

定理 1. 设 $x_{\text{nb}}^t(t=1,\cdots,N)$ 为新个体 $\boldsymbol{x}_{\text{trial}}$ 的 N 邻近个体，则 $\boldsymbol{x}_{\text{trial}}$ 的下界估计值 $\bar{y}_{\text{trial}}$ 满足

$$\bar{y}_{\text{trial}}<f(\boldsymbol{x}_{\text{trial}}) \tag{18}$$

其中，$f(\boldsymbol{x}_{\text{trial}})$ 为新个体 $\boldsymbol{x}_{\text{trial}}$ 的实际目标函数值.

证明. 由于目标函数 f 满足定义 1，则

(1) 若 $f(x_{\text{nb}}^t)>f(\boldsymbol{x}_{\text{trial}})$，根据式(2)有

$$\begin{aligned} &f(x_{\text{nb}}^t)-f(\boldsymbol{x}_{\text{trial}})\leq M\|x_{\text{nb}}^t-\boldsymbol{x}_{\text{trial}}\|\Rightarrow\\ &f(x_{\text{nb}}^t)-M\|x_{\text{nb}}^t-\boldsymbol{x}_{\text{trial}}\|\leq f(\boldsymbol{x}_{\text{trial}}) \end{aligned} \tag{19}$$

由式(12)可知

$$h^t(\boldsymbol{x}_{\text{trial}})=f(x^t_{\text{nb}})-M\parallel x^t_{\text{nb}}-\boldsymbol{x}_{\text{trial}}\parallel\leq f(\boldsymbol{x}_{\text{trial}}) \quad (20)$$

根据式(13)和式(14)，式(20)可简化为

$$h^t(\boldsymbol{x}_{\text{trial}})=\min_{j\in I}(f(x^t_{\text{nb}})-M(x^t_{\text{nb},j}-x_{\text{trial},j})\leq f(\boldsymbol{x}_{\text{trial}}) \quad (21)$$

则由式(17)有

$$\bar{y}_{\text{trial}}=\max_{t\leq N}h^t(\boldsymbol{x}_{\text{trial}})\leq f(\boldsymbol{x}_{\text{trial}}) \quad (22)$$

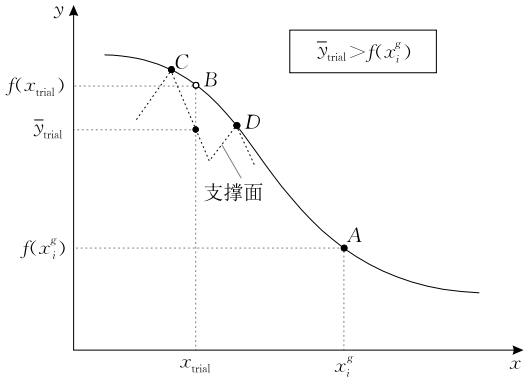
(2) 若 $f(x^t_{\text{nb}})<f(\boldsymbol{x}_{\text{trial}})$，根据式(2)有

$$-f(\boldsymbol{x}_{\text{trial}})+f(x^t_{\text{nb}})\leq M\parallel \boldsymbol{x}_{\text{trial}}-x^t_{\text{nb}}\parallel\Rightarrow$$

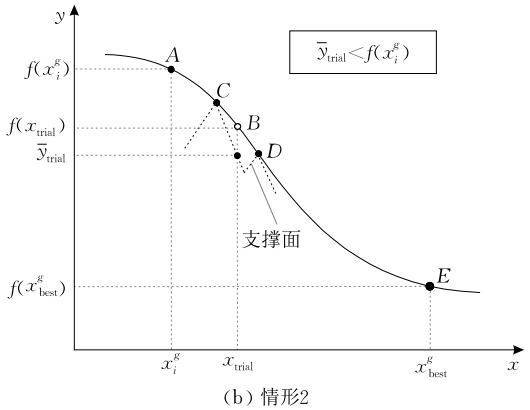
$$f(x^t_{\text{nb}})-M\parallel \boldsymbol{x}_{\text{trial}}-x^t_{\text{nb}}\parallel\leq f(\boldsymbol{x}_{\text{trial}}) \quad (23)$$

由于 $\parallel \boldsymbol{x}_{\text{trial}}-x^t_{\text{nb}}\parallel=\parallel x^t_{\text{nb}}-\boldsymbol{x}_{\text{trial}}\parallel$，则根据情形(1)同理可得 $\bar{y}_{\text{trial}}\leq f(\boldsymbol{x}_{\text{trial}})$，由于对于 $\forall t\in\{1,2,\cdots,N\}$有 $x^t_{\text{nb}}\neq\boldsymbol{x}_{\text{trial}}$，则 $\bar{y}_{\text{trial}}<f(\boldsymbol{x}_{\text{trial}})$. 证毕.

定理 1 表明，新个体 $\boldsymbol{x}_{\text{trial}}$ 根据其 N 邻近个体的 Lipschitz 支撑向量得到的下界估计值 $\bar{y}_{\text{trial}}$ 小于其实际目标函数值 $f(\boldsymbol{x}_{\text{trial}})$. 进而在更新过程中，可以根据新个体的下界估计值来指导种群更新，即根据下界估计值判断是否有必要对新个体进行目标函数评价. 以图 2 所示的一维问题为例说明更新过程中可能出现的情形.



(a) 情形1



(b) 情形2

图 2　局部 Lipschitz 估计选择策略中的两种情形

情形(1). 如图 2(a)所示，A 为目标个体 x^g_i（g 表示进化代数），B 为新个体 $\boldsymbol{x}_{\text{trial}}$，个体 C 和 D 为 B 的邻近个体，则根据 C 和 D 的 L 下界估计支撑面可以计算得到 B 的下界估计值 $\bar{y}_{\text{trial}}$，由于 $\bar{y}_{\text{trial}}$ 大于目标个体的函数值 $f(x^g_i)$，且由定理 1 可知 $f(\boldsymbol{x}_{\text{trial}})>\bar{y}_{\text{trial}}$，则 $f(\boldsymbol{x}_{\text{trial}})>\bar{y}_{\text{trial}}>f(x^g_i)$. 因此，无需对新个体 $\boldsymbol{x}_{\text{trial}}$ 进行目标函数评价（即计算 $f(\boldsymbol{x}_{\text{trial}})$），即可判定新个体 $\boldsymbol{x}_{\text{trial}}$ 比目标个体 x^g_i 差，而保持 x^g_i 不变.

情形(2). 如图 2(b)所示，A 为目标个体 x^g_i，B 为新个体 $\boldsymbol{x}_{\text{trial}}$，$E$ 为当前种群中的最优个体 x^g_{best}，同样根据 B 的邻近个体 C 和 D 的下界估计支撑面可以计算出 $\boldsymbol{x}_{\text{trial}}$ 的下界估计值 $\bar{y}_{\text{trial}}$，由于根据 $\bar{y}_{\text{trial}}$ 小于目标个体的函数值 $f(x^g_i)$ 无法判断新个体 $\boldsymbol{x}_{\text{trial}}$ 是否优于目标个体 x^g_i，则进一步将 $\bar{y}_{\text{trial}}$ 与当前种群中最优个体 x^g_{best} 的目标函数值 $f(x^g_{\text{best}})$ 进行比较，由于 $\bar{y}_{\text{trial}}>f(x^g_{\text{best}})$，则根据定理 1 可知 $f(\boldsymbol{x}_{\text{trial}})>\bar{y}_{\text{trial}}>f(x^g_{\text{best}})$. 因此，也无需对新个体 $\boldsymbol{x}_{\text{trial}}$ 进行目标函数评价，即可判定抛弃 $\boldsymbol{x}_{\text{trial}}$，而保持目标个体 x^g_i 不变.

情形(3). 除了情形(1)和情形(2)以外的情形，则需要对新个体 $\boldsymbol{x}_{\text{trial}}$ 进行目标函数评价，通过比较 $f(\boldsymbol{x}_{\text{trial}})$ 和目标个体的函数值 $f(x^g_i)$ 来判断新个体 $\boldsymbol{x}_{\text{trial}}$ 是否能够替换目标个体 x^g_i.

根据上述 3 种情形，局部 Lipschitz 估计选择策略可进一步概括为

$$x^{g+1}_i=\begin{cases}x^g_i, & \text{若 } \bar{y}_{\text{trial}}\geq f(x^g_i)\text{ 或 }\bar{y}_{\text{trial}}\geq f(x^g_{\text{best}})\\ \boldsymbol{x}_{\text{trial}}, & \text{若 } f(\boldsymbol{x}_{\text{trial}})\leq f(x^g_i)\\ x^g_i, & \text{其他}\end{cases} \quad (24)$$

由此可以看出，通过对新个体的 N 邻近个体建立 Lipschitz 下界估计支撑面获取新个体的下界估计值来指导种群更新，只在情形(3)中需要对新个体进行目标函数评价，因此相对于传统 DE 算法，可以有效地减少目标函数评价次数.

与 DEUS 算法相比，DELLU 算法在建立下界估计的过程中，采用的 Lipschitz 估计支撑函数不需要通过模型转换过程将原目标函数转化为单位单纯形中的 IPH 函数，避免了在转化过程中的信息丢失，从而能够求得更加精确的目标函数下界估计值；另外，DELLU 算法在选择策略中增加了情形(2)，只有在情形(3)中才需要对新个体进行目标函数评价. 因此 DELLU 算法能够节省更多的函数评价次数.

3.2　无效区域排除策略

为了进一步提高算法的搜索效率，减少函数评价次数，根据 Lipschitz 下界估计区域的极值信息排除部分无效子区域（即此子区域不可能包含全局最优解），逐步缩小搜索区域，从而使得算法集中搜索希望较大的区域.

定义 6. 定义新个体 $\boldsymbol{x}_{\text{trial}}$ 的 N 邻近个体的 Lipschitz 下界估计支撑面 $\boldsymbol{l}_{\text{nb}}^{t}(t=1,\cdots,N)$ 相交形成的区域

$$A(L)=\{l_{\text{nb}}^{1},l_{\text{nb}}^{2},\cdots,l_{\text{nb}}^{N}\}^{\text{T}} \tag{25}$$

为下界估计区域.

由式(17)可知,与下界估计区域 $A(L)$ 对应的搜索子区域中的任一个体 x_i 均可根据式(17)计算得到其下界估计值 $\bar{y}_i=H^N(x_i)$. 又由定理 1 可知,$\bar{y}_i$ 均小于 x_i 的实际目标函数值 $f(x_i)$,因此,当下界估计区域 $A(L)$ 的极小值(即与 $A(L)$ 对应的搜索子区域的最小下界估计值)大于当前种群(即第 g 代种群)的最优值 $f(x_{\text{best}}^{g})$ 时,即可判定 $A(L)$ 对应的搜索子区域不可能包含全局最优解,从而可以可靠排除.

由 Lipschitz 估计理论可知[16],任一下界估计区域 $A(L)$ 均对应着一个由 $n+1$ 个支撑向量组成的支撑矩阵 $\boldsymbol{L}$:

$$\boldsymbol{L}=\begin{pmatrix} l_1^{k_1} & l_2^{k_1} & \cdots & l_{n+1}^{k_1} \\ l_1^{k_2} & l_2^{k_2} & \cdots & l_{n+1}^{k_2} \\ \vdots & \vdots & \vdots & \vdots \\ l_1^{k_{n+1}} & l_2^{k_{n+1}} & \cdots & l_{n+1}^{k_{n+1}} \end{pmatrix} \tag{26}$$

其中,$L_{i,j}=\dfrac{f(x^{k_i})}{M}-x_j^{k_i}$.

定理 2. 设 $x_{\min}$ 为下界估计区域 $A(L)$ 的局部极小解,$d_{\min}=H^N(x_{\min})$ 为其对应的值,则 $A(L)$ 所对应的支撑矩阵 $\boldsymbol{L}$ 满足以下性质:

(1) $x_{\min,i}=\dfrac{d_{\min}}{M}-l_i^{k_i}$;

(2) $d_{\min}=\dfrac{\text{Trace}(L)+1}{\sum\limits_{i=1}^{n+1}\dfrac{1}{M}}=\dfrac{M(\text{Trace}(L)+1)}{n+1}$;

(3) $\forall i,j\in I,i\neq j:l_i^{k_j}>l_i^{k_i}$;

(4) $\forall r\notin\{k_1,\cdots,k_{n+1}\},\exists i\in I:L_{i,i}=l_i^{k_i}\geq l_i^{r}$.

证明. 设

$$R(x)=\{k\in\{1,\cdots,N\}:H^N(x)=h^k(x)\} \tag{27}$$

$$Q_k(x)=\{i\in I:h^k(x)=M(l_i^k+x_i)\} \tag{28}$$

则由文献[26]可知,对于 $\forall u\in U(x_{\min},D)=\{u\in R^{n+1}:\exists\alpha_0>0:x_{\min}+\alpha u\in D,\forall\alpha\in(0,\alpha_0)\}$,存在 $k\in R(x_{\min})$ 使得

$$(h^k)'(x_{\min},u)=\min_{i\in Q_k(x)}M\cdot u_i\geq 0 \tag{29}$$

设 $m\in I$,对于 $\forall i\neq m$ 存在 $\lambda_i>0$,使得 $\sum\limits_{i=1}^{n+1}\lambda_i=1$,考虑向量 $\boldsymbol{u}^*$

$$u_i^*=\begin{cases}1, & \text{若 } i=m\\ -\lambda_i, & \text{若 } i\neq m\end{cases},\quad i\in I \tag{30}$$

显然 $\boldsymbol{u}^*\in U(x_{\min},D)$. 因此存在 $k\in R(x_{\min})$,使得 $(h^k)'(x_{\min},\boldsymbol{u}^*)\geq 0$. 如果 $Q_k(x_{\min})\neq\{m\}$,则

$$(h^k)'(x_{\min},\boldsymbol{u}^*)=\min_{i\in Q_k(x_{\min})}Mu_i^*<0 \tag{31}$$

则与式(29)矛盾. 因此,对于任意的 $m\in I$,存在 $k^m\in R(x_{\min})$ 使得 $Q_k(x_{\min})=\{m\}$. 由此可知,设 k_{m_i} 为某一索引下标值使得 $Q_{k_{m_i}}(x_{\min})=\{m_i\}$,$i=1,2$,若 $m_1\neq m_2$,则 $k_{m_1}\neq k_{m_2}$.

(1) 设 $k_m\in R(x_{\min})$ 使得 $Q_{k_m}(x_{\min})=\{m\}$. 因为 $k_m\in R(x_{\min})$ 且 $m\in Q_{k_m}(x_{\min})$,则

$$H^N(x_{\min})=h_{k_m}(x_{\min})=M(l_m^{k_m}+x_m) \tag{32}$$

因此,

$$x_m=\frac{H^N(x_{\min})}{M}-l_m^{k_m},\quad\forall m\in I \tag{33}$$

则由式(33)有

$$\begin{aligned}x_{\min}&=\left(\frac{H^N(x_{\min})}{M}-l_1^{k_1},\cdots,\frac{H^N(x_{\min})}{M}-l_{n+1}^{k_{n+1}}\right)\\&=\left(\frac{d_{\min}}{M}-l_1^{k_1},\cdots,\frac{d_{\min}}{M}-l_{n+1}^{k_{n+1}}\right)\end{aligned} \tag{34}$$

(2) 由 $x_{n+1}=1-\sum\limits_{i=1}^{n}x_i$ 可知 $\sum\limits_{i=1}^{n+1}x_{\min,i}=1$,则

$$\begin{aligned}1&=\sum_{i=1}^{n+1}x_{\min,i}=\sum_{i=1}^{n+1}\left(\frac{d_{\min}}{M}-l_i^{k_i}\right)=\frac{d_{\min}(n+1)}{M}-\sum_{i=1}^{n+1}l_i^{k_i}\\&=\frac{d_{\min}(n+1)}{M}-\text{Trace}(L)\end{aligned} \tag{35}$$

因此,根据式(35)有

$$d_{\min}=\frac{M(\text{Trace}(L)+1)}{n+1}=\frac{\text{Trace}(L)+1}{\sum\limits_{i=1}^{n+1}\dfrac{1}{M}} \tag{36}$$

(3) 因为 $k_j\in R(x_{\min})$,$Q_{k_j}(x_{\min})=\{j\}$,则

$$M(l_i^{k_j}+x_{\min,i})>h_{k_j}(x_{\min})=M(l_j^{k_j}+x_{\min,j})=H^K(x_{\min}),\quad i\neq j \tag{37}$$

由式(37)有

$$\begin{aligned}&M(l_i^{k_j}+x_{\min,i})>M(l_j^{k_j}+x_{\min,j})\Rightarrow\\&l_i^{k_j}+x_{\min,i}>l_j^{k_j}+x_{\min,j}\end{aligned} \tag{38}$$

又因为 $x_{\min,i}=\dfrac{d_{\min}}{M}-l_i^{k_i}$,$x_{\min,j}=\dfrac{d_{\min}}{M}-l_j^{k_j}$,因此,当 $i\neq j$ 时

$$\begin{aligned}&l_i^{k_j}+\frac{d_{\min}}{M}-l_i^{k_i}>l_j^{k_j}+\frac{d_{\min}}{M}-l_j^{k_j}\Rightarrow\\&l_i^{k_j}-l_i^{k_i}>0\Rightarrow l_i^{k_j}>l_i^{k_i}\end{aligned} \tag{39}$$

(4) 由式(17)有

$$\begin{aligned}H^N(x_{\min})&=\max_{k\leq N}\min_{i\in I}M(l_i^{k}+x_{\min,i})\\&=\max_{k\leq N}\min_{i\in I}M\left(l_i^{k}+\frac{d_{\min}}{M}-l_i^{k_i}\right)=d_{\min}\end{aligned} \tag{40}$$

因此,

$$\max_{k\le N}\min_{i\in I}M(l_i^k-l_i^{k_i})=0\Rightarrow\max_{k\le N}\min_{i\in I}(l_i^k-l_i^{k_i})=0 \tag{41}$$

设 $\forall r\notin\{k_1,k_2,\cdots,k_{n+1}\}$，则

$$\min_{i\in I}(l_i^r-l_i^{k_i})\le 0 \tag{42}$$

因此，$\forall r\notin\{k_1,\cdots,k_{n+1}\}$，$\exists i\in I:l_i^{k_i}\ge l_i^r$. 证毕.

在选择过程中，当新个体的下界估计值 y_{trial} 大于目标个体 x_i^g 的目标函数值 $f(x_i^g)$，或大于当前种群中最优个体的目标函数值 $f(x_{\text{best}}^g)$ 时，即 $\bar{y}_{\text{trial}}\ge f(x_i^g)$ 或 $\bar{y}_{\text{trial}}\ge f(x_{\text{best}}^g)$ 时，继续根据定理 2 的性质(1)计算出下界估计区域的极小值 $d_{\min}$. 如图 3 所示，由于 $d_{\min}\ge f(x_{\text{best}}^g)$，即可判断此下界估计区域对应的搜索子区域(即 A 和 B 之间的区域)中不可能包含全局最优解，从而可将此区域看作无效区域，并建立数组 IR 记录其对应的支撑矩阵. 与 DEUS 算法相比，由于 DELLU 算法能够求得更加精确的下界估计值，而无效区域的排除与 $d_{\min}$ 的精确度有很大关系，因此 DELLU 算法能够找出更多的无效区域.

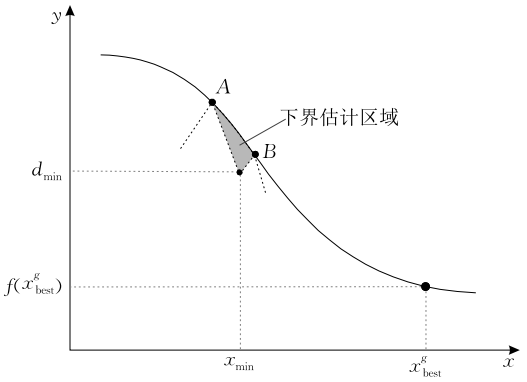


图 3　无效区域排除过程

定理 3.　设 $\boldsymbol{x}_{\text{trial}}$ 为新个体，L^u 为某一无效区域对应的支撑矩阵，若 $\boldsymbol{x}_{\text{trial}}$ 不在此无效区域中，则

$$\exists i,j\in I,i\ne j:x_j^{k_j}-x_{\text{trial},j}<x_i^{k_j}-x_{\text{trial},i} \tag{43}$$

其中，$x_i^{k_j}$ 为 $L_{j,i}^u(L_{j,i}^u=f(x^{k_j})/M-x_i^{k_j})$ 中的元素.

证明.　由于 $\boldsymbol{x}_{\text{trial}}$ 不在 L^u 所对应的无效区域中，则根据定理 2 的性质(4)有

$$\exists i,j\in I,i\ne j:l_{\text{trial},j}\le l_j^{k_j},l_{\text{trial},i}>l_i^{k_j} \tag{44}$$

将式(16)代入上式有

$$\frac{f(\boldsymbol{x}_{\text{trial}})}{M}-x_{\text{trial},j}\le\frac{f(x^{k_j})}{M}-x_j^{k_j},$$
$$\frac{f(\boldsymbol{x}_{\text{trial}})}{M}-x_{\text{trial},i}>\frac{f(x^{k_j})}{M}-x_i^{k_j} \tag{45}$$

由式(45)有

$$M(x_j^{k_j}-x_{\text{trial},j})\le f(x^{k_j})-f(\boldsymbol{x}_{\text{trial}})<M(x_i^{k_j}-x_{\text{trial},i}) \tag{46}$$

则由式(46)有 $x_j^{k_j}-x_{\text{trial},j}<x_i^{k_j}-x_{\text{trial},i}$. 证毕.

当新个体 $\boldsymbol{x}_{\text{trial}}$ 生成后，若无效区域存在，即 IR 不为空时，根据定理 3 即可判断出新个体是否被包含在无效区域中，如果 $\boldsymbol{x}_{\text{trial}}$ 在无效区域中，则保持目标个体 x_i^g 不变，并直接进入下一次迭代.

由此可以看出，通过无效区域排除策略可以有效排除部分无效区域(即在此区域中不可能包含全局最优解)，逐步缩小搜索区域，使得算法集中搜索希望较大的子区域，不仅能够提高算法的搜索效率，降低算法的计算代价，而且在一定程度上能够防止算法陷入局部最优，提高算法的可靠性.

3.3　广义下降方向局部增强策略

传统 DE 算法全局搜索能力较强，但是局部搜索能力较弱，尤其在进化后期收敛速度较慢，因此后期需要进行大量的目标函数评价. 针对此问题，DEUS 算法通过直接获取抽象凸估计区域的极小解在目标函数曲面上对应的点与目标个体进行比较，以获得更优个体. 而在本文中则结合 N 邻近个体的下降方向和主导支撑面的下降方向提出一种局部增强广义下降方向，利用此方向引导个体作局部增强，提高算法的局部搜索能力，从而加快算法的收敛速度，进一步减少函数评价次数.

定义 7.　若新个体 $\boldsymbol{x}_{\text{trial}}$ 的下界估计值

$$\bar{y}_{\text{trial}}=H^N(\boldsymbol{x}_{\text{trial}})=\max_{t\le N}\min_{j=1,\cdots,n+1}M(l_{\text{nb},j}^t+x_{\text{trial},j})$$
$$=M(l_{\text{nb},j}^k+x_{\text{trial},j}),\ k\in\{1,\cdots,N\} \tag{47}$$

即下界估计值 $\bar{y}_{\text{trial}}$ 可以根据邻近个体 x_{nb}^k 的支撑面 l_{nb}^k 求出，则称支撑面 l_{nb}^k 为主导支撑面，且称支撑面 l_{nb}^k 的下降方向

$$\vec{d}_{\text{dom}}=(d_{\text{dom},1},\cdots,d_{\text{dom},n+1})$$
$$=(x_{\min,1}-x_{\text{nb},1}^k,\cdots,x_{\min,n+1}-x_{\text{nb},n+1}^k) \tag{48}$$

为主导支撑面下降方向，其中 $x_{\min}$ 为下界估计区域的极小解.

定义 8.　设 $x_{\text{nb}}^t(t=1,\cdots,N)$ 为新个体 $\boldsymbol{x}_{\text{trial}}$ 的 N 邻近个体，则称方向

$$\vec{d}_{\text{nb}}=(d_{\text{nb},1},\cdots,d_{\text{nb},n+1})$$
$$=(x_{\text{Bnb},1}-x_{\text{Wnb},1},\cdots,x_{\text{Bnb},n+1}-x_{\text{Wnb},n+1}) \tag{49}$$

为 N 邻近个体下降方向，其中，x_{Bnb} 和 x_{Wnb} 分别为 N 邻近个体中的最优个体和最差个体.

定义 9.　设 $\vec{d}_{\text{dom}}$ 和 $\vec{d}_{\text{nb}}$ 分别为新个体 $\boldsymbol{x}_{\text{trial}}$ 的主导支撑面下降方向和 N 邻近个体下降方向，则称方向

$$\vec{d}=\vec{d}_{\text{dom}}+\vec{d}_{\text{nb}} \tag{50}$$

为 $\boldsymbol{x}_{\text{trial}}$ 的广义下降方向.

如图 4 所示，设 B 为新个体 $\boldsymbol{x}_{\text{trial}}$，个体 C 和 D 为 B 的邻近个体，图中给出了 B 的主导支撑面下降方向、N 邻近个体下降方向和广义下降方向，则当新个体 $\boldsymbol{x}_{\text{trial}}$ 优于目标个体 x_i^g 时，为了进一步加快收

敛速度，根据 $\boldsymbol{x}_{\text{trial}}$ 的广义下降方向作局部增强，即

$$x_{\text{test}}=\boldsymbol{x}_{\text{trial}}+F\cdot\vec{d} \tag{51}$$

其中参数 F 与变异策略中的步长因子相同，如果 x_{test} 优于 $\boldsymbol{x}_{\text{trial}}$，则 x_{test} 替换目标个体 x_i^g.

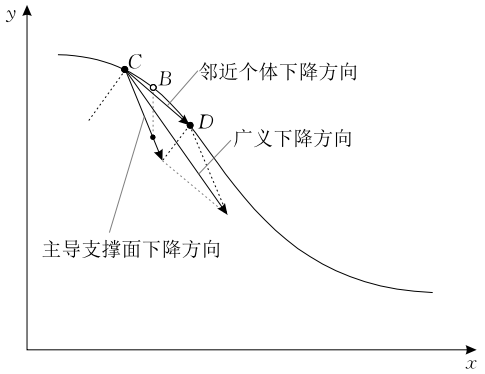


图 4 广义下降方向

3.4 算法设计

DELLU 算法的主要思想是：在基本 DE 算法框架下，提取新个体的 N 邻近个体构建基于 Lipschitz 估计理论的下界支撑面，通过支撑面获取新个体的下界估计信息指导种群更新，并根据下界估计区域的极值信息排除部分无效区域，同时根据基于 N 邻近个体下降方向和主导支撑面下降方向的广义下降方向作局部增强，从而有效提高算法的性能，减少进化过程中所需的目标函数评价次数，降低算法的计算代价.

以极小化问题为例，算法整体流程如下：

1. 初始化. 设置种群规模 NP，交叉概率 CR，增益常数 F，N 邻近个体数目 N，并随机生成初始种群 $P=\{x_1,x_2,\cdots,x_{NP}\}$，初始化无效区域 IR 为空，进化代数 $g=0$.

2. 对每个目标个体 $x_i^g\in P$ 执行变异、交叉过程生成新个体 x_{trial}^i.

3. 对每个目标个体 x_i^g 及其对应的新个体 x_{trial}^i 进行如下操作：

3.1 设置 $i=1$；

3.2 如果 IR 不为空，则根据式(43)判断 x_{trial}^i 是否在无效区域 IR 中，如果不在，则转至步骤 3.3，否则 $x_i^{g+1}=x_i^g$，并转至步骤 3.17；

3.3 根据式(6)找出 x_{trial}^i 的 N 邻近个体 x_{nb}^t；

3.4 根据式(16)计算 x_{nb}^t 的下界估计支撑向量 $\boldsymbol{l}_{\text{nb}}^t$，并建立支撑矩阵 $\boldsymbol{L}$；

3.5 根据式(17)计算 x_{trial}^i 的下界估计值 $\bar{y}_{\text{trial}}$，并根据式(47)记录主导支撑面；

3.6 如果 $\bar{y}_{\text{trial}}\geq f(x_i^g)$，则 $x_i^{g+1}=x_i^g$，并转至步骤 3.9，否则转至步骤 3.7；

3.7 计算当前种群的最优值 $f(x_{\text{best}}^g)$；

3.8 如果 $\bar{y}_{\text{trial}}\geq f(x_{\text{best}}^g)$，则 $x_i^{g+1}=x_i^g$，并转至步骤 3.9，否则转至步骤 3.11；

3.9 根据式定理 2 的性质(2)计算下界估计区域 L 的极小值 $d_{\min}$；

3.10 如果 $d_{\min}\geq f(x_{\text{best}}^g)$，则将 L 加入 IR，并转至步骤 3.17；

3.11 如果 $f(x_{\text{trial}}^i)<f(x_i^g)$，则转至步骤 3.12，否则 $x_i^{g+1}=x_i^g$，并转至步骤 3.17；

3.12 根据式(48)计算主导支撑面下降方向 $\vec{d}_{\text{dom}}$；

3.13 根据式(49)计算 N 邻近个体下降方向 $\vec{d}_{\text{nb}}$；

3.14 根据式(50)计算 x_{trial}^i 的广义下降方向 $\vec{d}$；

3.15 根据式(51)计算局部增强个体 x_{test}；

3.16 如果 $f(x_{\text{test}})<f(x_{\text{trial}}^i)$，则 $x_i^{g+1}=x_{\text{test}}$，否则 $x_i^{g+1}=x_{\text{trial}}^i$；

3.17 $i=i+1$，并删除所有支撑向量 $\boldsymbol{l}_{\text{nb}}^t$；

3.18 如果 $i\leq NP$，转至步骤 3.2.

4. $g=g+1$.

5. 如果不满足终止条件，则转至步骤 2.

6. 输出结果，退出.

注：步骤 2 中的变异交叉过程参见文献[2]；步骤 3.17 在每次迭代结束后删除所有支撑向量，只保留无效区域的支撑向量.

3.5 复杂度分析

根据 3.4 节的算法步骤分析 DELLU 算法相对于传统 DE 算法所增加的时间复杂度. 步骤 3.2 中判断新个体是否在无效区域中时，只需判断新个体是否满足式(43)，因为本文通过树的形式保存无效区域，因此时间复杂度为 $O((n+1)\log T)$，其中 T 为无效区域的数目；步骤 3.3 中找出新个体的 N 邻近个体时需要计算各个体与新个体的距离，时间复杂度为 $O(n\cdot NP)$，根据距离找出 N 个邻近个体的时间复杂度为 $O(N\cdot NP)$；步骤 3.4 中计算支撑向量的时间复杂度为 $O(N(n+1))$；步骤 3.5 中计算下界估计值的时间复杂度为 $O(N(n+1))$；步骤 3.6 的判断下界估计值是否大于目标个体的函数值的时间复杂度为 $O(1)$；步骤 3.7 中计算当前种群的最优值的时间复杂度为 $O(NP)$；步骤 3.8、步骤 3.9 和步骤 3.10 的时间复杂度均为 $O(1)$；步骤 3.11 中判断新个体是否优于目标个体的时间复杂度为 $O(2)$；步骤 3.12 中计算主导支撑面下降方向的时间复杂度为 $O(n+1)$；步骤 3.13 中计算 N 邻近个体下降方向时需要找出 N 邻近个体中的最优个体和最差个体，则时间复杂度为 $O(2N+1)$；步骤 3.14 中计算广义下降方向的时间复杂度为 $O(n+1)$；步骤 3.15 中计算局部增强个体的时间复杂度也为 $O(n+1)$；步骤 3.16 中判断局部增强个体是否优于

新个体时，需要计算局部增强个体的目标函数值，若增强个体优于新个体，则增强个体替换新个体，则时间复杂度为 $O(n+1)$；步骤 3.17 中删除所有支撑向量的时间复杂度为 $O(1)$. 上述步骤中，某些步骤在满足条件的情况下才会执行，当上述步骤都执行时，最大时间复杂度为 $O(2N\cdot n+n\cdot NP+N\cdot NP+NP+n\cdot \log T+\log T+4n+4N+12)$，忽略常量、低次幂和最高次幂的系数，则最终时间复杂度为 $O(\max\{N\cdot n, n\cdot NP, N\cdot NP\})$. 如果只对新个体附近的两个个体建立支撑向量(即 $N=2$)，则当问题维数大于 2 时，算法的时间复杂度为 $O(n\cdot NP)$，由此可以看出，DELLU 算法没有显著增加 DE 算法的时间复杂度，且 DELLU 算法主要针对目标函数计算复杂的优化问题，因此，建立下界估计所增加的计算代价小于实际目标函数评价所需的计算代价.

4　数值实验

4.1　测试函数及算法参数设置

应用 20 个典型的标准测试函数来验证所提算法的性能，表 1 给出了各测试函数基本参数和数学表达式. 其他特性见文献[27-28].

表 1　20 个标准测试函数

函数名	数学表达式	维数	搜索范围	最优值
Sphere	$f_1(x)=\sum_{i=1}^{n}x_i^2$	30	$(-100,100)$	0
Tablet	$f_2(x)=10^6x_1^2+\sum_{i=2}^{n}x_i^2$	30	$(-100,100)$	0
Schwefel 2.22	$f_3(x)=\sum_{i=1}^{n}\lvert x_i\rvert+\prod_{i=1}^{n}\lvert x_i\rvert$	30	$(-10,10)$	0
Schwefel 1.2	$f_4(x)=\sum_{i=1}^{n}(\sum_{j=1}^{i}x_j^2)^2$	30	$(-100,100)$	0
Step	$f_5(x)=\sum_{i=1}^{n}\lfloor x_i+0.5\rfloor^2$	30	$(-100,100)$	0
Zakharov	$f_6(x)=\sum_{i=1}^{n}x_i^2+(\sum_{i=1}^{n}0.5ix_i)^2+(\sum_{i=1}^{n}0.5ix_i)^4$	30	$(-5,10)$	0
Rosenbrock	$f_7(x)=\sum_{i=1}^{n-1}(100(x_{i+1}-x_i)^2+(x_i-1)^2)$	30	$(-2,2)$	0
Griewank	$f_8(x)=1+\frac{1}{4000}\sum_{i=1}^{n}x_i^2-\prod_{i=1}^{n}\cos\left(\frac{x_i}{\sqrt{i}}\right)$	30	$(-600,600)$	0
Schaffer 2	$f_9(x)=\sum_{i=1}^{n-1}(x_i^2+x_{i+1}^2)^{0.25}(\sin^2(50(x_i^2+x_{i+1}^2)^{0.1})+1)$	30	$(-100,100)$	0
Ackley	$f_{10}(x)=-20\exp\left(-0.2\sqrt{n^{-1}\sum_{i=1}^{n}x_i^2}\right)-\exp\left(n^{-1}\sum_{i=1}^{n}\cos(2\pi x_i)\right)+20+e$	30	$(-30,30)$	0
Schwefel 2.26	$f_{11}(x)=-\sum_{i=1}^{n}x_i\sin(\sqrt{\lvert x_i\rvert})$	30	$(-500,500)$	-12569.48
Himmelblau	$f_{12}(x)=n^{-1}\sum_{i=1}^{n}(x_i^4-16x_i^2+5x_i)$	30	$(-100,100)$	-78.3323
Levy and Montalvo 1	$f_{13}(x)=\pi(10\sin^2(\pi y_1)/n+\sum_{i=1}^{n-1}(y_i-1)^2(1+10\sin^2(\pi y_i+1))+(y_n-1)^2,\ y_i=1+0.25(x_i+1)$	30	$(-10,10)$	0
Levy and Montalvo 2	$f_{14}(x)=0.1(\sin^2(3\pi x_i)+\sum_{i=1}^{n-1}(x_i-1)^2(1+\sin^2(3\pi x_{i+1})+(x_n-1)^2(1+\sin^2(2\pi x_n)))$	30	$(-5,5)$	0
Rastrigin	$f_{15}(x)=10n+\sum_{i=1}^{n}(x_i^2-10\cos(2\pi x_i))$	30	$(-5,5)$	0
Cosine Mixture	$f_{16}(x)=0.1\sum_{i=1}^{n}\cos(5\pi x_i)-\sum_{i=1}^{n}x_i^2$	4	$(-1,1)$	-0.4
Kowalik	$f_{17}(x)=\sum_{i=1}^{11}(a_i-(x_1(b_i^2+b_ix_2))/(b_i^2+b_ix_3+x_4))^2$	4	$(-5,5)$	0.0003075
Six-hump Camel-back	$f_{18}(x)=4x_1^2-2.1x_1^4+x_1^6/3+x_1x_2-4x_2^2+4x_2^4$	2	$(-5,5)$	-1.0316285
Branin	$f_{19}(x)=(x_2-5.1x_1^2/4\pi+5x_1/\pi-6)^2+10(1-1/8\pi)\cos x_1+10$	2	$(-5,10)$	0.39789
Goldstein-Price	$f_{20}(x)=[1+(x_1+x_2+1)^2(19-14x_1+3x_1^2-14x_2+6x_1x_2+3x_2^2)]\cdot[30+(2x_1-3x_2)^2(18-32x_1+12x_1^2+48x_2-36x_1x_2+27x_2^2)]$	2	$(-2,2)$	3

20 个标准测试函数中包含了 7 个高维单模函数($f_1 \sim f_7$)、8 个高维多模函数($f_8 \sim f_{15}$)和 5 个低维多模函数($f_{16} \sim f_{20}$),其中,高维多模函数局部最优解的数量随着维数的增大呈现指数增加.上述所有测试函数均满足 Lipschitz 条件.DELLU 算法参数设置:用于建立 Lipschitz 下界估计支撑面的新个体的邻近个体数目 $N=2$,种群规模 $NP=50$.其次,为了公平比较,所有算法均设置相同的运行终止条件,且每种算法对于每个测试函数均独立运行 30 次.实验环境为 Intel(R) Core i5-2410M CPU@2.30 GHz with 8 GB RAM,Windows 7,算法实现代码采用 Visual Studio 2012 C++和 Matlab 8.2 编写.

4.2 DELLU 算法与基本 DE 算法比较

为了验证 DELLU 算法相对于基本 DE 算法的优势,选用 3 种使用不同变异策略的基本 DE 算法进行比较:DE/rand/1[2]、DE/best/1[2]和 DE/rand-to-best/1[32],上述所有比较算法的参数设置均为 $F=0.5$,$CR=0.5$,$NP=50$[29].

实验中,采用 3 种评价指标来比较分析各算法的性能:(1)在设定的最大函数评价次数 $MaxFE$(Maximum Number of Function Evaluations)内达到给定的函数误差值($f(x)-f(x^*)$)精度 δ 时所需的目标函数评价次数 FE(Function Evaluations);(2)成功率 SR(Success Rate);(3)在设定的最大函数评价次数内所求得的最小函数误差值的平均值(Mean Error)和标准偏差值(Standard Deviation,Std Dev).在函数误差值($f(x)-f(x^*)$)中,$f(x)$表示当前所找到的最优值,$f(x^*)$表示函数的全局最优值(见表 1).对于指标(2),规定算法在 $MaxFE$ 内能够达到设定的函数误差值精度 δ 时为求解成功,则 SR 为成功运行次数和总运行次数之比.在此实验中,对于指标(1)和(2),最大函数评价次数 $MaxFE$ 设置为 300 000,误差值精度 δ 设置为 1E−04,对于指标(3),$MaxFE$ 设置为 $1000\times n$,其中 n 为维数.另外,为了验证所提算法相对于其他比较算法是否有显著性优势,选用 Wilcoxon Signed Rank Test[30]对各算法 30 次优化得到的结果进行非参数假设检验,显著性水平为 5%,并且通过“+”表示所提算法显著优于所比算法,“−”表示所提算法显著差于所比算法,“≈”表示两种算法之间没有显著性差异.

表 2 给出了 DE/rand/1、DE/best/1、DE/rand-to-best/1 和 DELLU 4 种算法 30 次独立运行的平均目标函数评价次数和成功率,其中“NA”表示对应的算法在给定的 $MaxFE$ 内不能达到指定的函数误差值精度 δ.从表中可以看出,DE/rand/1 算法对 3 个函数无法求解成功(f_4、f_7 和 f_{15});对于 DE/best/1 和 DE/rand-to-best/1 算法,其变异策略具有

表 2 DE/rand/1、DE/best/1、DE/rand-to-best/1 和 DELLU 算法的平均函数评价次数和成功率

函数	维数	DE/rand/1		DE/best/1		DE/rand-to-best/1		DELLU	
		FE	*SR*/%	*FE*	*SR*/%	*FE*	*SR*/%	*FE*	*SR*/%
f_1	30	3.01E+04	**100**	1.09E+04	**100**	**9.40E+03**	60	1.20E+04	**100**
f_2	30	3.11E+04	**100**	1.17E+04	**100**	**9.47E+03**	50	2.56E+04	**100**
f_3	30	3.69E+04	**100**	1.38E+04	**100**	**1.11E+04**	**100**	1.64E+04	100
f_4	30	NA	0	NA	0	NA	0	**6.59E+04**	**100**
f_5	30	1.65E+04	**100**	6.40E+03	20	**5.14E+03**	53	5.66E+03	**100**
f_6	30	2.56E+05	**100**	**4.78E+04**	**100**	5.71E+04	60	8.09E+04	**100**
f_7	30	NA	0	1.78E+05	87	NA	0	**5.96E+04**	**93**
f_8	30	3.29E+04	97	**1.12E+04**	53	1.47E+04	40	2.00E+04	**100**
f_9	30	1.32E+05	**100**	NA	0	NA	0	7.98E+04	**100**
f_{10}	30	4.03E+04	**100**	1.49E+04	67	**1.41E+04**	33	2.84E+04	**100**
f_{11}	30	1.29E+05	100	NA	0	NA	0	**2.14E+04**	**100**
f_{12}	30	5.18E+04	90	NA	0	NA	0	**1.91E+04**	**100**
f_{13}	30	1.96E+04	**100**	6.38E+03	87	**5.34E+03**	**100**	9.09E+03	**100**
f_{14}	30	1.90E+04	**100**	6.82E+03	90	**5.16E+03**	93	8.53E+03	**100**
f_{15}	30	NA	0	NA	0	NA	0	**1.74E+04**	**100**
f_{16}	4	2.22E+03	**100**	1.22E+03	**100**	1.34E+03	**100**	**9.60E+02**	**100**
f_{17}	4	6.62E+03	**100**	**2.85E+03**	90	3.75E+03	90	4.78E+03	**100**
f_{18}	2	1.49E+03	**100**	**7.58E+02**	**100**	9.16E+02	**100**	1.20E+03	**100**
f_{19}	2	1.54E+03	**100**	**7.89E+02**	**100**	9.71E+02	**100**	1.39E+03	**100**
f_{20}	2	1.64E+03	**100**	**9.40E+02**	**100**	1.04E+03	**100**	1.30E+03	**100**
总平均值		8.55E+04	84.4	9.07E+04	64.7	9.70E+04	54.0	**2.40E+04**	**99.7**

贪婪性，即在迭代过程中促使每个个体向当前种群中的最优个体收敛，而在 DELLU 算法中仅采用 DE/rand/1 变异策略，因此在对部分测试函数求解时，DE/best/1 和 DE/rand-to-best/1 算法所需的函数评价次数少于 DELLU 算法，但是由于 DE/best/1 和 DE/rand-to-best/1 算法的贪婪性，使得算法很容易陷入局部最优，因此其成功率较低. DE/best/1 算法对 5 个函数无法求解，其中包括 1 个高维单模函数（f_4）和 4 个高维多模函数（f_9、f_{11}、f_{12} 和 f_{15}），DE/rand-to-best/1 算法对 6 个函数无法求解，其中包括 2 个高维单模函数（f_4 和 f_7）和 4 个高维多模函数（f_9、f_{11}、f_{12} 和 f_{15}）；除了函数 f_7，DELLU 算法对其他函数能够以 100% 的成功率求解. 尤其对于函数 f_7（Rosenbrock 问题），当其维数大于 3 时则变为多模函数[31]，虽然算法能够很快找到其局部最优解，但是由于其函数曲面极其复杂，使得算法极易陷入局部最优，而无法找到全局最优解，4 种比较算法在对函数 f_7 求解时，只有 DE/best/1 和 DELLU 算法能够成功求解，且相比于 DE/best/1 算法，DELLU 算法可以以较少的函数评价次数和较高的成功率求解. 另外，从表 2 最后一行的平均结果也可以看出，DELLU 算法对于 20 个测试函数的总平均函数评价次数为 2.40E＋04，DE/rand/1、DE/best/1 和DE/rand-to-best/1 算法分别为 8.55E＋04、9.07E＋04 和 9.70E＋04，也就是说，DELLU 算法相对于 DE/rand/1、DE/best/1 和 DE/rand-to-best/1 算法分别节省了 71.9%、73.5% 和 75.3% 的函数评价次数；其次，DELLU 的可靠性最高，成功率为 99.7%，由于 DE/rand/1、DE/best/1 和 DE/rand-to-best/1 算法对部分函数无法成功求解，所以成功率分别为 84.4%、64.7% 和 54.0%. 总之，相比于 3 种传统 DE 算法，DELLU 算法在保证成功率的同时，可以有效减少函数评价次数.

为了验证 DELLU 算法相对于 3 种传统 DE 算法在 FE 和 SR 方面的整体优势，首先根据表 2 中各算法对于各测试函数在最大函数评价次数 $MaxFE$ 内达到给定的函数误差值精度 δ 所需的函数评价次数及成功率计算各算法（a）对各测试函数（p）的成功性能（Success Performance，SP）

$$s_{a,p}=\frac{\text{Mean}(FE)}{SR} \tag{52}$$

然后通过各算法对于各测试函数的 SP 值除以最优算法的 SP 值进行归一化处理；最后绘制成功表现归一化经验分布图. 图中，SP 值最小而经验分布值最大的算法最好. 图 5 给出了 DE/rand/1、DE/best/1、DE/rand-to-best/1 和 DELLU 算法对 20 个测试函数的成功表现归一化经验分布图，可以看出，DELLU 算法的经验分布值最先到达 1，由此表明，DELLU 算法在 FE 和 SR 方面的整体性能优于其他 3 种传统 DE 算法.

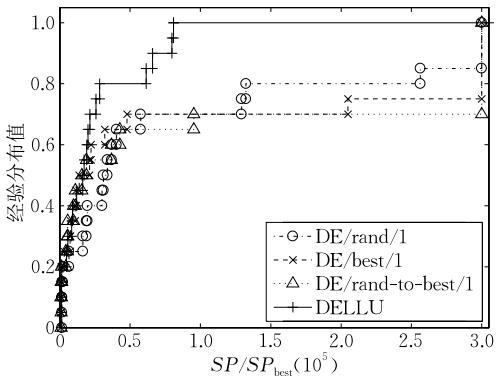


图 5　DE/rand/1、DE/best/1、DE/rand-to-best/1 和 DELLU 算法的成功表现归一化经验分布

表 3 给出了 DE/rand/1、DE/best/1、DE/rand-to-best/1 和 DELLU 4 种算法对各测试函数 30 次优化得到的最小函数误差的平均值和标准偏差值，最优结果通过加粗标出. 从表中可以看出，DELLU 算法对于大部分函数优于 3 种基本 DE 算法；其次，从表中最后一行列出的显著性检验结果可以看出，DE/rand/1 算法没有优于 DELLU 算法任何问题，而 DELLU 算法优于 DE/rand/1 算法 17 个问题，对于其余 3 个问题，两种算法相比没有显著性差异；虽然 DE/best/1 算法优于 DELLU 算法 4 个问题，其中包括 3 个高维单模问题（f_2、f_3 和 f_6）和 1 个低维多模问题（f_{19}），但是 DELLU 算法优于 DE/best/1 算法 13 个问题，对于其余 3 个问题，两种算法相比没有显著性差异，由此可以推断出，DE/best/1 算法仅适合求解高维单模和低维多模问题；DE/rand-to-best/1 算法仅优于 DELLU 算法 1 个单模问题（f_{19}），而 DELLU 算法优于 DE/rand-to-best/1 算法 16 个问题，对于其余 3 个问题，两种算法相比没有显著性差异；对于 DELLU 算法，除了函数 f_2、f_3、f_6 和 f_{19}，其余函数的结果均优于 3 种所比基本 DE 算法.

表 3 DE/rand/1、DE/best/1、DE/rand-to-best/1 和 DELLU 算法函数误差的平均值和标准偏差

函数	维数	DE/rand/1	DE/best/1	DE/rand-to-best/1	DELLU
		Mean Error±Std Dev	Mean Error±Std Dev	Mean Error±Std Dev	Mean Error±Std Dev
f_1	30	1.17E−04±5.27E−05+	9.66E−19±2.97E−18+	4.33E−02±1.64E−01+	**8.98E−21±5.23E−21**
f_2	30	1.95E−04±7.75E−05+	**1.63E−18±2.49E−18**−	4.21E−01±8.71E−01+	2.65E−12±1.71E−12
f_3	30	1.41E−03±3.20E−04+	**1.18E−11±2.03E−11**−	1.25E−07±3.62E−07+	8.12E−11±3.06E−11
f_4	30	6.02E+04±2.03E+04+	6.69E+03±3.20E+03+	3.62E+03±1.43E+03+	**1.07E+01±6.38E+00**
f_5	30	**0.00E+00±0.00E+00**≈	6.17E+00±9.81E+00+	1.60E+00±1.89E+00+	**0.00E+00±0.00E+00**
f_6	30	9.64E+01±2.11E+01+	**8.92E−02±9.90E−02**−	1.82E−01±2.31E−01+	1.74E+00±2.38E+00
f_7	30	2.56E+01±3.42E−01+	2.37E+01±1.04E+01+	2.81E+01±1.41E+00+	**3.89E−02±3.49E−02**
f_8	30	3.38E−03±9.06E−03+	7.95E−03±1.51E−02+	2.05E−02±5.98E−02+	**3.60E−08±2.13E−08**
f_9	30	1.30E+01±1.23E+00+	6.41E+00±5.44E+00+	1.03E+00±2.03E+00+	**8.66E−01±3.31E−01**
f_{10}	30	2.52E−03±5.14E−04+	7.48E−01±8.93E−01+	1.04E−01±3.08E−01+	**4.24E−05±1.91E−05**
f_{11}	30	5.51E+03±3.55E+02+	1.56E+03±5.09E+02+	1.78E+03±1.06E+03+	**0.00E+00±0.00E+00**
f_{12}	30	1.22E+01±3.75E+00+	1.06E+01±2.79E+00+	9.43E+00±2.71E+00+	**0.00E+00±0.00E+00**
f_{13}	30	1.25E−07±7.64E−08+	2.07E−02±4.22E−02+	2.75E−08±1.04E−07+	**6.50E−14±3.43E−14**
f_{14}	30	7.32E−08±3.97E−08+	1.46E−03±3.80E−03+	1.10E−03±3.35E−03+	**3.50E−17±1.60E−17**
f_{15}	30	1.46E+02±9.82E+00+	2.73E+01±1.20E+01+	7.65E+01±2.43E+01+	**4.06E−08±2.84E−08**
f_{16}	4	**0.00E+00±0.00E+00**≈	**0.00E+00±0.00E+00**≈	**0.00E+00±0.00E+00**≈	**0.00E+00±0.00E+00**
f_{17}	4	3.29E−04±8.44E−05+	1.46E−03±5.07E−03+	8.82E−04±3.63E−03+	**2.13E−04±8.34E−05**
f_{18}	2	2.32E−05±5.02E−05+	**0.00E+00±0.00E+00**≈	**0.00E+00±0.00E+00**≈	**0.00E+00±0.00E+00**
f_{19}	2	6.87E−05±7.87E−05≈	**7.00E−06±1.69E−16**−	7.07E−06±2.54E−07−	4.96E−05±6.29E−05
f_{20}	2	9.67E−06±1.47E−05+	**0.00E+00±0.00E+00**≈	**0.00E+00±0.00E+00**≈	**0.00E+00±0.00E+00**
+/≈/−		17/3/0	13/3/4	16/3/1	

4.3 DELLU 算法与 state-of-the-art DE 算法比较

4.3.1 与基于估计的 DE 算法比较

为了验证 DELLU 算法相对于其他基于估计的 DE 算法的优势，选取 2 种基于 k-近邻预测的 DE 算法 DE-EkNN[13]、DE-kNN[8] 和 1 种基于代理模型的 DE 算法 GPEME[10] 与 DELLU 算法进行比较. 实验中，DELLU 算法的参数设置均与上述所比算法的原文献相同. 为了客观比较，DE-EkNN、DE-kNN 和 GPEME 算法的实验数据均来自原文献. 由于原文献只对表 1 中的部分测试函数进行了测试，所以实验中只选取部分函数进行分析比较.

表 4 给出了 DE-EkNN、DE-kNN 和 DELLU 算法对部分测试函数 30 次求解所得到的结果，其中最优结果通过加粗标出. 从表中可以看出，对于函数 f_5，3 种算法均能够得到全局最优解；DE-EkNN 算法仅优于 DELLU 算法 1 个问题（f_8）；DE-kNN 算法对函数 f_5、f_{18} 和 f_{19} 求解时与 DELLU 获得了相同的结果，即找到了全局最优解，但对于其他问题均无法优于 DELLU 算法；DELLU 算法优于 DE-EkNN 和 DE-kNN 算法大多数问题，尤其对函数 f_1、f_{11} 和 f_{15}，DELLU 算法的优势更加明显. 对于函数 f_1，DELLU 算法优于其他算法 6 个数量级；对于函数 f_{15}，DELLU 算法优于其他算法 11 个数量级；对于函数 f_{15}，只有 DELLU 算法能够成功求得全局最优解. 此外，需要说明的是，由于所比算法在原文献中没有提供 30 次独立运行的结果，所以在比较中没有进行显著性检验.

表 5 为 GPEME 和 DELLU 算法对函数 f_7、f_8 和 f_{10} 求解所得结果的性能数据，其中，“Best”表示 30 次独立运行所得结果中的最优值，“Worst”表示最差值，各项性能指标的最优结果通过加粗标出. 从表中可以看出，对于函数 f_7，GPEME 算法仅在 20 维和 30 维的最优结果方面优于 DELLU 算法，对于其余性能，DELLU 算法均优于 GPEME 算法. 对于函数 f_8，在 20 维时，GPEME 算法在结果的最优值、平均值和标准偏差方面优于 DELLU 算法，但在最差值方面逊于 DELLU 算法；在 30 维和 50 维时，DELLU 算法除了对于 30 维的标准偏差不如 GPEME 算法外，其余性能均优于 GPEME 算法. 对于函数 f_{10}，GPEME 算法仅在 20 维的最优值和平均值方面及 30 维的标准偏差方面优于 DELLU 算法，其他性能均逊于 DELLU 算法；在 50 维时，DELLU 算法的所有性能均优于 GPEME 算法. 由此可以看出，DELLU 算法总体上优于 GPEME 算法.

表 4　DE-EkNN、DE-kNN 和 DELLU 算法函数误差的平均值和标准偏差

函数	维数	DE-EkNN	DE-kNN	DELLU
		Mean Error±Std Dev	Mean Error±Std Dev	Mean Error±Std Dev
f_1	10	1.83E−08±1.53E−08	3.52E−08±4.17E−08	**9.08E−15±1.02E−14**
f_3	10	6.33E−05±4.22E−05	8.79E−05±4.52E−05	**4.94E−08±2.17E−08**
f_4	10	1.08E−01±5.69E−01	3.84E−02±2.06E−01	**1.46E−05±3.84E−05**
f_5	10	**0.00E+00±0.00E+00**	**0.00E+00±0.00E+00**	**0.00E+00±0.00E+00**
f_7	10	8.04E+00±1.24E+00	9.15E+00±1.14E+00	**4.17E+00±1.62E+00**
f_8	10	**1.61E−01±2.26E−02**	8.17E−02±1.43E−01	2.33E−01±1.76E−01
f_{10}	10	7.35E−05±5.36E−05	9.28E−05±5.14E−05	**1.16E−07±9.01E−08**
f_{11}	10	1.53E+03±3.40E+02	1.63E+03±3.11E+02	**0.00E+00±0.00E+00**
f_{15}	10	4.33E+01±7.46E+00	4.24E+01±5.27E+00	**1.33E−10±1.79E−10**
f_{18}	2	2.85E−05±8.07E−11	**0.00E+00±0.00E+00**	**0.00E+00±0.00E+00**
f_{19}	2	2.00E−06±2.49E−08	**0.00E+00±0.00E+00**	**0.00E+00±0.00E+00**
f_{20}	2	0.00E+00±1.74E−14	0.00E+00±1.07E−14	**0.00E+00±0.00E+00**

表 5　GPEME 和 DELLU 算法的优化结果性能对比数据

函数	维数	GPEME				DELLU			
		Best	Worst	Mean Error	Std Dev	Best	Worst	Mean Error	Std Dev
f_7	20	**1.51E+01**	7.59E+01	2.24E+01	8.16E+01	1.91E+01	**2.80E+01**	**2.12E+01**	**2.87E+00**
f_7	30	**2.63E+01**	8.82E+01	4.62E+01	2.55E+01	2.94E+01	**7.07E+01**	**3.94E+01**	**1.26E+01**
f_7	50	1.72E+02	4.01E+02	2.58E+02	8.02E+01	**5.17E+01**	**2.46E+02**	**1.06E+02**	**6.46E+01**
f_8	20	**2.00E−04**	2.23E−01	**3.07E−02**	**6.82E−02**	1.03E−03	**1.86E−01**	3.57E−02	6.13E−02
f_8	30	7.37E−01	1.08E+00	9.97E−01	**1.08E−01**	**3.80E−02**	**1.01E+00**	**4.73E−01**	3.31E−01
f_8	50	2.25E+01	6.50E+01	3.66E+01	1.32E+01	**1.71E−03**	**1.14E+00**	**4.37E−01**	**5.10E−01**
f_{10}	20	**3.70E−03**	1.84E+00	**1.99E−01**	5.77E−01	1.37E−01	**1.67E+00**	5.60E−01	**4.61E−01**
f_{10}	30	1.95E+00	4.96E+00	3.01E+00	**9.25E−01**	**1.41E−02**	**3.13E+00**	**1.59E+00**	1.19E+00
f_{10}	50	9.25E+00	1.49E+01	1.32E+01	1.58E+00	**3.25E−02**	**2.51E+00**	**8.35E−01**	**9.18E−01**

4.3.2　与基于自适应机制的 DE 算法比较

为了验证 DELLU 算法相对于基于参数和策略自适应机制的 state-of-the-art DE 算法的优势，选取以下 3 种算法进行比较：

(1) Qin 等人[32]提出的策略自适应差分进化算法(Differential Evolution Algorithm with Strategy Adaptation，SaDE)：利用均匀布策略对变异率和交叉概率进行调整，并通过一种学习机制来自适应调整进化过程中不同阶段的变异策略及其参数.

(2) Mallipeddi 等人[33]提出的具系综变异策略及参数的差分进化算法(Differential Evolution Algorithm with Ensemble of Parameters and Mutation Strategies，EPSDE)：对种群中的每个个体随机分配不同的变异策略及参数，同时保存能够产生较优个体的变异策略及参数，并对产生较差个体的策略及参数重新初始化.

(3) Wang 等人[34]提出的具有混合新个体生成策略及参数的差分进化算法(Differential Evolution with Composite Trial Vector Generation Strategies and Control Parameters，CoDE)：设置一组变异策略池和参数池，从而通过随机组合策略池中的变异策略和参数池中的参数来竞争产生后代个体.

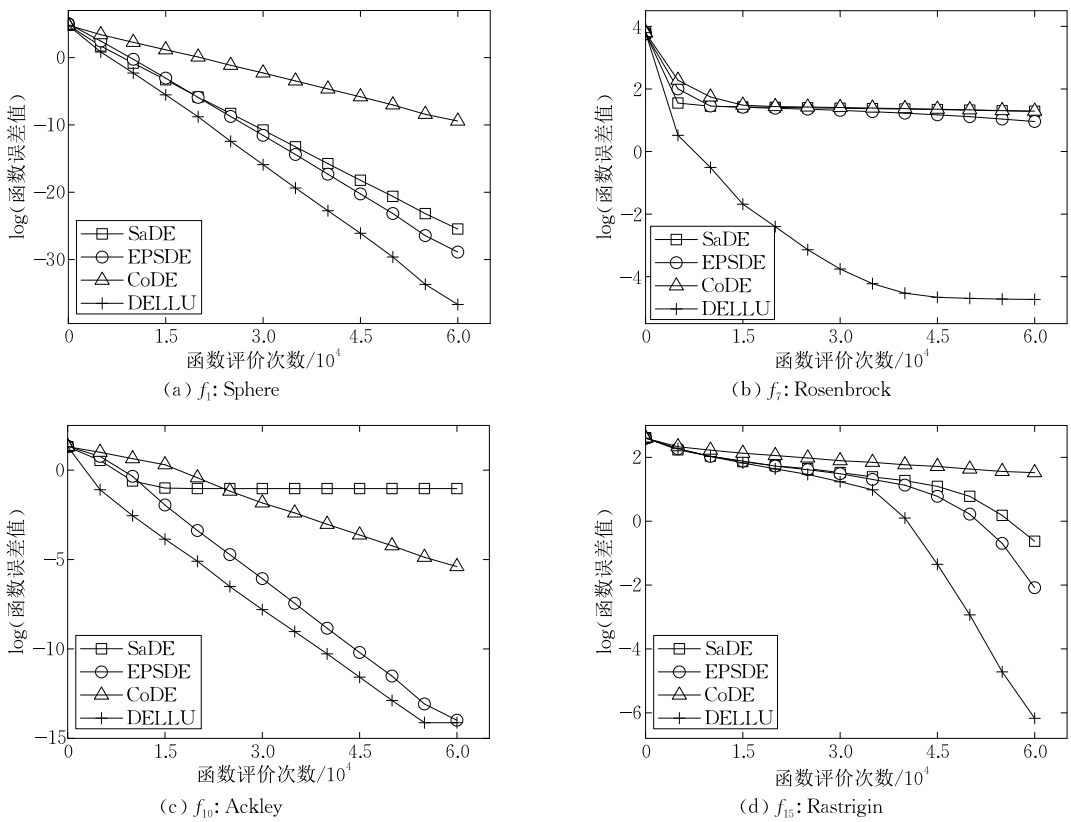
上述 3 种比较算法均采用原文献中的参数设置，其次，为了公平比较，DELLU 算法中的变异概率 CR 和增益常数 F 的设置采用 SaDE 算法中提出的参数自适应机制，具体参见文献[32]. 在此实验中，通过记录最大函数评价次数 $MaxFE$ 内 30 次独立运行所得的函数误差值的平均值和标准偏差来比较分析 4 种算法的性能，算法终止条件均为 $MaxFE=2000\times n$.

表 6 给出了 SaDE、EPSDE、CoDE 和 DELLU 算法对各测试函数 30 次独立运行得到的最小函数误差值的平均值和标准偏差，各函数的最优结果通过加粗标出. 从表中可以看出，除了函数 f_2、f_3、f_9 和 f_{10} 外，DELLU 算法对于其他函数的性能均优于其他 3 种对比算法，对于函数 f_5、f_{12}、f_{16}、f_{18} 和 f_{20}，4 种算法均达到了全局最优，而获得了相同的结果；其次，从表中最后一行的显著性检验结果可以看出，SaDE 算法优于 DELLU 算法 3 个问题，其中包括 2 个高维单模问题(f_2 和 f_3)和 1 个高维多模问题(f_{10})，DELLU 算法优于 SaDE 算法 10 个问题，对于其余 7 个问题，两者相比没有显著性差异；EPSDE 算法仅优于 DELLU 算法 2 个高维单模问题(f_2 和 f_3)，而 DELLU 算法优于 EPSDE 算法 10 个问题，对于其余 8 个问题，两者相比没有显著性差异；CoDE 算法没有优于 DELLU 算法任何问题，而 DELLU 算法优于 CoDE 算法 15 个问题，对于其余 5 个问题，两者之间没有显著性差异.

表 6 SaDE、EPSDE、CoDE 和 DELLU 算法函数误差的平均值和标准偏差

函数	维数	SaDE	EPSDE	CoDE	DELLU
		Mean Error±Std Dev	Mean Error±Std Dev	Mean Error±Std Dev	Mean Error±Std Dev
f_1	30	1.29E−23±3.07E−23+	3.87E−31±9.46E−31+	2.16E−10±1.73E−10+	**1.65E−37±2.25E−37**
f_2	30	8.85E−23±4.35E−22−	**8.85E−30±1.75E−29**−	4.04E−10±3.04E−10+	5.38E−21±8.12E−21
f_3	30	8.70E−16±5.17E−16−	**1.29E−16±2.15E−16**−	3.86E−06±1.32E−06+	2.91E−11±4.07E−11
f_4	30	2.83E−02±2.46E−02≈	3.82E+03±5.34E+03+	8.57E−01±1.06E+00+	**2.41E−02±1.98E−02**
f_5	30	**0.00E+00±0.00E+00**≈	**0.00E+00±0.00E+00**≈	**0.00E+00±0.00E+00**≈	**0.00E+00±0.00E+00**
f_6	30	9.27E−02±1.30E−01+	1.42E+01±2.25E+01+	5.69E−04±7.53E−04+	**1.53E−04±2.33E−04**
f_7	30	5.07E+01±3.36E+01+	8.61E+00±2.09E+00+	1.97E+01±5.80E−01+	**1.48E−05±1.62E−05**
f_8	30	2.22E−03±4.73E−03+	6.57E−04±2.50E−03+	2.47E−07±7.34E−07+	**0.00E+00±0.00E+00**
f_9	30	7.51E−04±7.06E−04−	2.35E−01±1.31E−01+	1.97E+00±4.24E−01+	1.83E−02±6.16E−03
f_{10}	30	2.40E−01±4.45E−01+	**6.04E−15±1.66E−15**≈	3.75E−06±1.88E−06+	7.55E−15±1.47E−15
f_{11}	30	**0.00E+00±0.00E+00**≈	**0.00E+00±0.00E+00**≈	1.66E−02±3.13E−02+	**0.00E+00±0.00E+00**
f_{12}	30	**0.00E+00±0.00E+00**≈	**0.00E+00±0.00E+00**≈	**0.00E+00±0.00E+00**≈	**0.00E+00±0.00E+00**
f_{13}	30	2.47E−27±4.90E−27+	2.91E−29±1.14E−28+	1.80E−13±3.36E−13+	**1.09E−30±8.88E−31**
f_{14}	30	1.46E−03±3.80E−03+	1.75E−32±9.04E−33+	1.31E−13±1.51E−13+	**1.36E−32±2.26E−34**
f_{15}	30	4.11E−02±1.82E−01+	5.48E−02±1.39E−01+	3.30E+01±5.81E+00+	**6.37E−07±5.39E−07**
f_{16}	4	**0.00E+00±0.00E+00**≈	**0.00E+00±0.00E+00**≈	**0.00E+00±0.00E+00**≈	**0.00E+00±0.00E+00**
f_{17}	4	9.38E−06±1.95E−05+	2.26E−05±5.70E−05+	9.60E−06±4.06E−05+	**2.52E−06±1.11E−05**
f_{18}	2	**0.00E+00±0.00E+00**≈	**0.00E+00±0.00E+00**≈	**0.00E+00±0.00E+00**≈	**0.00E+00±0.00E+00**
f_{19}	2	4.67E−07±1.17E−06+	**0.00E+00±0.00E+00**≈	1.33E−07±5.71E−07+	**0.00E+00±0.00E+00**
f_{20}	2	**0.00E+00±0.00E+00**≈	**0.00E+00±0.00E+00**≈	**0.00E+00±0.00E+00**≈	**0.00E+00±0.00E+00**
+/≈/−		10/7/3	10/8/2	15/5/0	

图 6 给出了测试函数 f_1、f_7、f_{10} 和 f_{15} 的平均收敛曲线图. 为了图形的清晰直观,图中纵坐标值为函数误差值的对数. 从图 4(a)可以看出,SaDE、EPSDE、CoDE 和 DELLU 这 4 种算法在对 Sphere 函数求解时,DELLU 算法收敛速度最快,EPSDE 算法次之. 从图 4(b)可以看出,虽然 4 种对比算法在对 Rosenbrock 这一经典复杂问题求解时均陷入了局部最优,但是 DELLU 算法的收敛速度仍显著优于 SaDE、



(a) f_1: Sphere (b) f_7: Rosenbrock

(c) f_{10}: Ackley (d) f_{15}: Rastrigin

图 6 SaDE、EPSDE、CoDE 和 DELLU 算法对部分函数的平均收敛曲线

EPSDE 和 CoDE 算法，且最终结果也显著优于 SaDE、EPSDE 和 CoDE 算法. 从图 4(c)可以看出，对于 Ackley 问题，SaDE 算法陷入了局部最优而无法成功求解，虽然 DELLU 算法和 EPSDE 算法最终结果相差无几，但是 DELLU 算法前期收敛速度仍优于 EPSDE 算法；从图 4(d)可以看出，在对 Rastrigin 函数求解时，虽然 DELLU 算法前期的收敛速度优势不明显，但是最终还是超过了其他算法，并非常稳健地向着全局最优收敛，且最终结果显著优于其他 3 种对比算法.

4.4　DELLU 算法与 ACUP 和 DEUS 算法比较

在 ACUP 和 DEUS 算法中，建立下界估计时均采用式(10)所示的抽象凸下界估计支撑函数，而 DELLU 算法采用的是式(15)所示的 Lipschitz 下界估计支撑函数. 因此，DELLU 算法不需要通过线性变换公式将原目标函数转化到单位单纯形空间中，而且也不需要通过增加足够大的常数 C 将原目标函数转化为 IPH 函数，而是引入单纯形距离来对 Lipschitz 下界估计支撑函数进行简化，避免了转化过程中的信息丢失，因此 DELLU 算法能够得到更加精确的下界估计值. 为了验证这一点，以 1 维 Shubert 函数为例，对 DEUS 算法和 DELLU 算法设置相同的参数，通过建立函数的全局下界估计进行比较.

2000 个采样点的平均估计误差显示(限于篇幅，文中没有列出)，DEUS 算法的平均估计误差为 3.03，DELLU 算法的平均估计误差为 1.60. 图 7 也给出了 DEUS 算法和 DELLU 算法对 Shubert 函数所建立的下界估计，可以看出，DELLU 算法所建立的下界估计明显比 DEUS 所建立的下界估计精确.

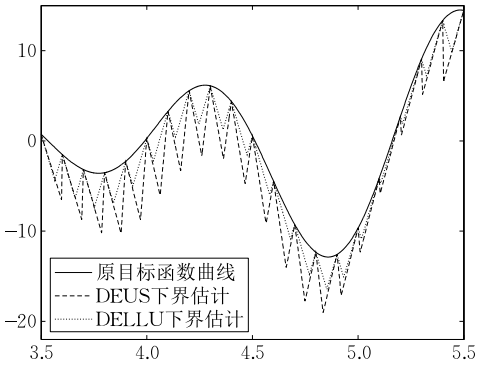


图 7　DEUS 和 DELLU 算法的下界估计对比

为了进一步验证 DELLU 算法相对于 ACUP[23] 和 DEUS[24] 算法在函数评价次数 FE 和成功率 SR 方面的优势，首先根据式(52)计算各算法(a)对各测试函数(p)的成功性能，然后根据式(53)计算对应的性能比 PR(Performance Ratio)[35]

$$r_{a,p}=\frac{s_{a,p}}{\min\{s_{a,p}:a\in A\}} \tag{53}$$

其中 A 表示对比算法集. 如果性能比 $r_{a,p}$ 等于 1，则表明算法(a)对测试函数(p)在 FE 和 SR 方面的综合性能最好，且 $r_{a,p}$ 越接近于 1，则表明对应算法的性能越接近于最优算法. 在此实验中，ACUP、DEUS 和 DELLU 算法均采用相同的参数设置：$F=0.5$，$CR=0.5$，$NP=50$，算法终止条件均为 $\delta=$1E-04.

表 7 给出了 ACUP、DEUS 和 DELLU 算法对各测试函数的函数评价次数 FE、成功率 SR 和性能

表 7　ACUP、DEUS 和 DELLU 算法的平均目标函数评价次数和成功率

函数	维数	ACUP			DEUS			DELLU		
		FE	SR/%	PR	FE	SR/%	PR	FE	SR/%	PR
f_1	30	1.58E+04	100	1.31	1.43E+04	100	1.19	**1.20E+04**	100	**1.00**
f_2	30	2.99E+04	100	1.17	2.82E+04	100	1.10	**2.56E+04**	100	**1.00**
f_3	30	1.69E+04	100	1.10	**1.62E+04**	100	**1.00**	1.64E+04	100	1.07
f_4	30	8.90E+04	100	1.35	7.30E+04	100	1.06	**6.59E+04**	100	**1.00**
f_5	30	6.10E+03	100	1.08	5.97E+03	100	1.04	**5.66E+03**	100	**1.00**
f_6	30	9.50E+04	100	1.17	8.91E+04	100	1.10	**8.09E+04**	100	**1.00**
f_7	30	6.52E+04	90	1.13	**5.89E+04**	90	1.02	5.96E+04	93	**1.00**
f_8	30	3.10E+04	100	1.55	2.50E+04	100	1.25	**2.00E+04**	100	**1.00**
f_9	30	8.90E+04	100	1.12	8.50E+04	100	1.07	**7.98E+04**	100	**1.00**
f_{10}	30	3.50E+04	100	1.23	3.32E+04	100	1.13	**2.84E+04**	100	**1.00**
f_{11}	30	2.50E+04	100	1.17	2.30E+04	100	1.03	**2.14E+04**	100	**1.00**
f_{12}	30	2.10E+04	100	1.17	**1.85E+04**	100	**1.00**	1.91E+04	100	1.06
f_{13}	30	1.13E+04	100	1.24	9.91E+03	100	1.09	**9.09E+03**	100	**1.00**
f_{14}	30	9.51E+03	100	1.15	**8.39E+03**	100	**1.00**	8.53E+03	100	1.03
f_{15}	30	2.10E+04	100	1.21	1.90E+04	100	1.09	**1.74E+04**	100	**1.00**
f_{16}	4	**9.50E+02**	100	**1.00**	9.79E+02	100	1.03	9.60E+02	100	1.01
f_{17}	4	5.21E+03	100	1.09	4.90E+03	100	1.03	**4.78E+03**	100	**1.00**
f_{18}	2	1.40E+03	100	1.16	1.31E+03	100	1.08	**1.20E+03**	100	**1.00**
f_{19}	2	**1.35E+03**	100	**1.00**	1.39E+03	100	1.03	1.39E+03	100	1.03
f_{20}	2	1.50E+03	100	1.15	1.40E+03	100	1.08	**1.30E+03**	100	**1.00**
总平均值		2.86E+04	99.5	1.18	2.59E+04	99.5	1.07	**2.40E+04**	**99.7**	**1.01**

比 PR,其中性能比为 1 的算法,即性能最优的算法通过加粗标出.可以看出,ACUP 算法仅在 2 个低维多模问题(f_{16} 和 f_{19})上获得了最优性能,DEUS 在 3 个问题上获得了最优性能,其中包括 1 个高维单模问题(f_3)和 2 个高维多模问题(f_{12} 和 f_{14}),而 DELLU 在其余 15 个问题上获得了最优性能.另外,从表中最后一行的总平均值可以看出,虽然 3 种比较算法的成功率相差无几,仅在对函数 f_7 求解时有所区别,其他函数的成功率均为 100%,但是由于 DELLU 算法所使用的 Lipschitz 下界估计支撑函数能够得到更精确的目标函数下界估计值,且对选择策略进行了改进,因此,DELLU 所需的函数评价次数最少,总平均值为 2.40E+04,ACUP 和 DEUS 算法分别为 2.86E+04 和 2.59E+04,也就是说,DELLU 算法相对于 ACUP 和 DEUS 算法分别节省了 16.1% 和 7.3%的函数评价次数.

图 8 给出了 ACUP、DEUS 和 DELLU 算法对 2 个单模函数(f_2 和 f_6)和 2 个多模函数(f_8 和 f_9)求解时的平均收敛曲线图.从图中可以看出,由于 DELLU 算法根据提出的广义下降方向做局部增强比 DEUS 算法中仅根据支撑面的下降方向获取下界估计区域的极值解作局部增强更合理,因此 DELLU 算法对于上述 4 个函数的收敛速度总是优于 ACUP 算法和 DEUS 算法.

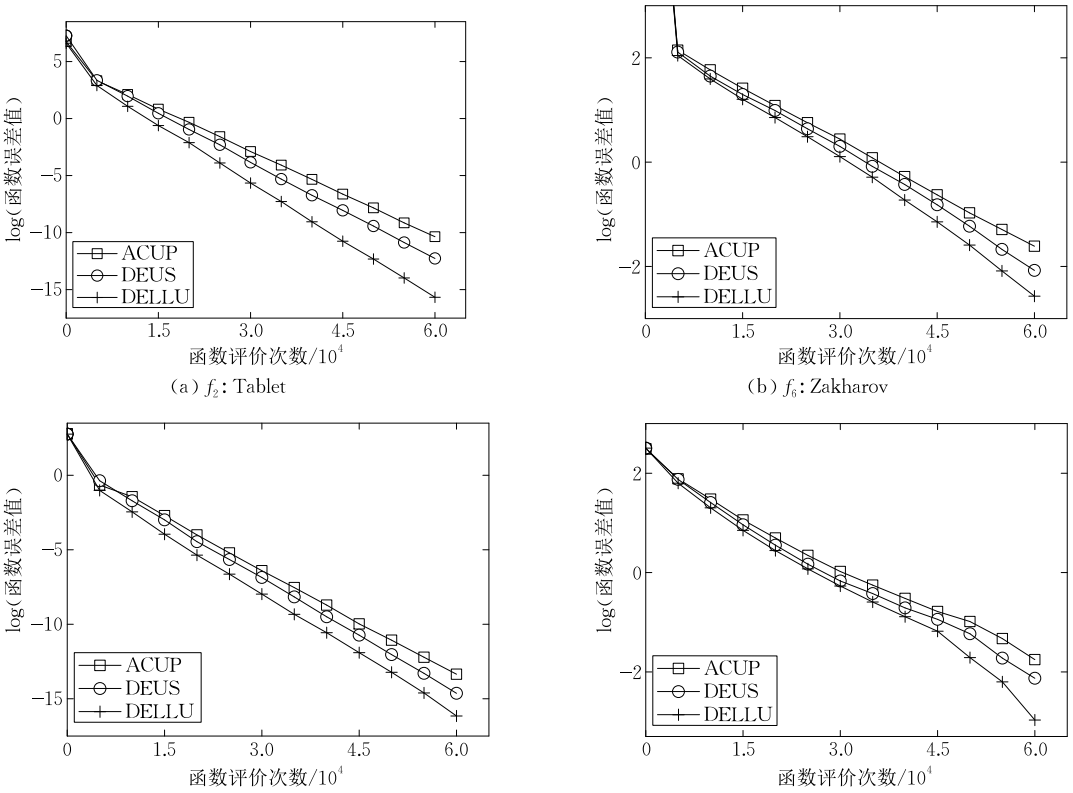


图 8 ACUP、DEUS 和 DELLU 算法对部分函数的平均收敛曲线

4.5 CEC 2013 偏移和旋转问题测试

为了验证 DELLU 算法求解复杂问题的性能,应用 CEC 2013 的偏移和旋转(Shifted and Rotated)问题进行测试,表 8 列出了所用测试问题的基本参数,具体数学表达式见文献[36].上述 15 个测试函数中,$F_1 \sim F_5$ 为单模函数,$F_6 \sim F_{15}$ 为多模函数.

实验中,DELLU 算法与以下 3 种先进的非 DE 算法进行比较:

(1) Liang 等人[37]提出的综合学习粒子群算法(Comprehensive Learning Particle Swarm Optimizer, CLPSO).该算法中,每个粒子均利用其他所有粒子的个人历史最优信息来更新速度;

(2) Hansen 等人[38]提出的基于协方差矩阵的自适应进化策略算法(Evolution Strategy with Covariance Matrix Adaptation, CMA-ES).该算法通过一种多元正态分布来枚举新的候选解,并利用协方差矩阵更新正态分布;

表 8　实验中使用的 CEC 2013 测试函数

函数名	最优值
F_1：Sphere Function	−1400
F_2：Rotated High Conditioned Elliptic Function	−1300
F_3：Rotated Bent Cigar Function	−1200
F_4：Rotated Discus Function	−1100
F_5：Different Powers Function	−1000
F_6：Rotated Rosenbrock's Function	−900
F_7：Rotated Schaffers F7 Function	−800
F_8：Rotated Ackley's Function	−700
F_9：Rotated Weierstrass Function	−600
F_{10}：Rotated Griewank's Function	−500
F_{11}：Rastrigin's Function	−400
F_{12}：Rotated Rastrigin's Function	−300
F_{13}：Non-Continuous Rotated Rastrigin's Function	−200
F_{14}：Schwefel's Function	−100
F_{15}：Rotated Schwefel's Function	100

(3) Garcia-Martinez 等人[39]提出的基于父代中心交叉操作的遗传算法(Genetic Algorithm Based on Parent-centric Crossover Operators，GL-25). 该算法利用父代中心实参交叉操作来提高算法的全局和局部搜索能力.

CLPSO、CMA-ES 和 GL-25 算法均采用原文献中的参数设置. 实验中，通过记录最大函数平次数 $MaxFE$ 内 30 次独立运行所得的最小函数误差值 $(f(x)-f(x^o))$ 的平均值和标准偏差来比较分析 4 种算法的性能，其中 x^o 为测试函数的全局最优解，所有算法终止条件均为 $MaxFE=10000\times n$.

表 9 给出了 CLPSO、CMA-ES、GL-25 和 DELLU 算法对 CEC 2013 偏移和旋转测试函数优化得到的结果，其中，最优结果通过加粗标出. 可以看出，4 种算法在对函数 F_1 优化时均达到了最优，因此得到了相同的结果. DELLU 算法虽然仅在 2 个单模问题上达到了最优，但是对于多模问题，除了函数 F_{10} 外，DELLU 算法对于其余 9 个多模函数均优于 CLPSO、CMA-ES 和 GL-25 算法. 其次，从表中最后一行的显著性检验对比结果可以看出，CLPSO 算法没有优于 DELLU 算法任何问题，而 DELLU 算法优于 CLPSO 算法 12 个问题，对于其余 2 个问题，两者之间没有显著性差异；与 CMA-ES 算法相比，虽然 CMA-ES 算法在 4 个问题上优于 DELLU 算法，其中包括 3 个单模函数($F_2\sim F_4$)和 1 个多模函数(F_{10})，但是 DELLU 算法在其余 10 个问题上优于 CMA-ES 算法；GL-25 算法同样没有优于 DELLU 算法任何问题，而 DELLU 算法优于 GL-25 算法 14 个问题.

表 9　CLPSO、CMA-ES、GL-25 和 DELLU 算法函数误差的平均值和标准偏差

函数	维数	CLPSO	CMA-ES	GL-25	DELLU
		Mean Error±Std Dev	Mean Error±Std Dev	Mean Error±Std Dev	Mean Error±Std Dev
F_1	10	**0.00E+00±0.00E+00**≈	**0.00E+00±0.00E+00**≈	**0.00E+00±0.00E+00**≈	**0.00E+00±0.00E+00**
F_2	10	1.06E+06±5.92E+05+	**0.00E+00±0.00E+00**−	4.27E+04±4.26E+04+	5.52E−12±4.13E−12
F_3	10	1.54E+06±3.01E+06+	**0.00E+00±0.00E+00**−	8.92E+01±1.95E+02+	1.21E−02±1.54E−02
F_4	10	6.22E+03±1.22E+03+	**0.00E+00±0.00E+00**−	1.12E+03±7.58E+02+	6.35E−08±2.11E−07
F_5	10	**0.00E+00±0.00E+00**≈	1.14E−13±1.31E−13+	5.68E−14±5.99E−14+	**0.00E+00±0.00E+00**
F_6	10	3.01E+00±3.65E+00+	7.85E+00±4.14E+00+	7.92E+00±3.98E+00+	**2.32E+00±3.13E+00**
F_7	10	1.26E+01±8.03E+00+	2.87E+01±1.88E+01+	4.87E−01±3.91E−01+	**2.76E−04±2.05E−04**
F_8	10	2.03E+01±8.12E−02+	2.03E+01±8.11E−02+	2.04E+01±7.08E−02+	**2.02E+01±1.07E−01**
F_9	10	3.91E+00±6.50E−01+	1.45E+01±5.16E+00+	2.97E+00±1.36E+00+	**2.59E+00±2.05E+00**
F_{10}	10	7.28E−01±2.42E−01+	**1.48E−02±1.65E−02**−	1.69E−01±9.27E−02+	3.63E−02±1.75E−02
F_{11}	10	**0.00E+00±0.00E+00**≈	2.04E+02±2.27E+02+	3.50E+00±1.72E+00+	**0.00E+00±0.00E+00**
F_{12}	10	1.14E+01±3.94E+00+	3.25E+02±2.45E+02+	1.09E+01±9.76E+00+	**6.45E+00±2.14E+00**
F_{13}	10	1.56E+01±4.49E+00+	1.66E+02±2.85E+02+	1.44E+01±7.78E+00+	**6.07E+00±2.02E+00**
F_{14}	10	6.64E−02±6.33E−02+	1.69E+03±4.48E+02+	2.02E+02±1.17E+02+	**1.56E−02±2.84E−02**
F_{15}	10	7.74E+02±1.52E+02+	1.74E+03±3.16E+02+	1.19E+03±1.89E+02+	**4.12E+02±1.36E+02**
+/≈/−		12/3/0	10/1/4	14/1/0	

4.6　参数选择分析

在 DELLU 算法中，当新个体生成后，为了获取其下界估计信息来指导种群进化，需要选择新个体的 N 个邻近个体建立 Lipschitz 下界估计支撑面，因此，需要对参数 N 的选择进行分析. 实验中，首先对参数 N 设置不同的值，即 $N=2,3,4,5$；然后通过记录对函数 $f_1\sim f_{15}$ 求解所需的函数评价次数和成功率来分析参数 N 对算法性能的影响，算法终止条件为函数误差值精度 $\delta=$1E-04.

表 10 给出了 DELLU 算法在参数 N 不同设置下对于各测试函数的函数评价次数 FE 和成功率 SR. 从表中最后一行可以看出，在参数 N 设置不同的情况下，DELLU 算法可以对所有函数成功求解，且所需函数评价次数相差无几. 此外，为了进一步分析参数 N 的设置对不同问题结果的影响，分别对单模和多模函数的平均函数评价次数进行分析. 图 9

给出了单模函数和多模函数在 N 不同取值下的函数评价次数箱型对比图，可以看出，无论是单模函数还是多模函数，在 N 取值不同的情况下，所需函数评价次数几乎没有差异. 由此可以推断出，DELLU 算法的性能对参数 N 的选择并不敏感. 然而，随着 N 的增大，即需要建立 Lipschitz 下界估计支撑面的个体的数目增多，算法的空间复杂度也会相应的变大，因此，文中只对与新个体邻近的 2(即 $N=2$)个种群个体构建下界估计支撑面，且在每次更新环节完成后删除所有下界估计支撑面.

表 10 DELLU 算法中参数 N 不同设置下的平均函数评价次数和成功率

函数	维数	$N=2$		$N=3$		$N=4$		$N=5$	
		FE	*SR*/%	*FE*	*SR*/%	*FE*	*SR*/%	*FE*	*SR*/%
f_1	20	9.45E+03	100	9.42E+03	100	9.32E+03	100	9.51E+03	100
f_2	20	2.03E+04	100	2.16E+04	100	2.15E+04	100	2.03E+04	100
f_3	20	1.39E+04	100	1.39E+04	100	1.33E+04	100	1.40E+04	100
f_4	20	5.69E+04	100	5.73E+04	100	5.65E+04	100	5.62E+04	100
f_5	20	2.23E+03	100	2.31E+03	100	2.21E+03	100	2.33E+03	100
f_6	20	4.51E+04	100	4.53E+04	100	4.45E+04	100	4.57E+04	100
f_7	20	2.43E+04	100	2.53E+04	100	2.51E+04	100	2.55E+04	100
f_8	20	1.29E+04	100	1.26E+04	100	1.32E+04	100	1.28E+04	100
f_9	20	5.52E+04	100	5.67E+04	100	5.44E+04	100	5.51E+04	100
f_{10}	20	2.21E+04	100	2.18E+04	100	2.12E+04	100	2.28E+04	100
f_{11}	20	1.02E+04	100	1.15E+04	100	1.12E+04	100	1.09E+04	100
f_{12}	20	1.65E+04	100	1.73E+04	100	1.70E+04	100	1.74E+04	100
f_{13}	20	6.93E+03	100	6.81E+03	100	6.82E+03	100	6.71E+03	100
f_{14}	20	8.37E+03	100	8.50E+03	100	8.45E+03	100	8.31E+03	100
f_{15}	20	9.76E+03	100	9.52E+03	100	9.91E+03	100	9.81E+03	100
总平均值		2.09E+04	100	2.13E+04	100	2.10E+04	100	2.12E+04	100

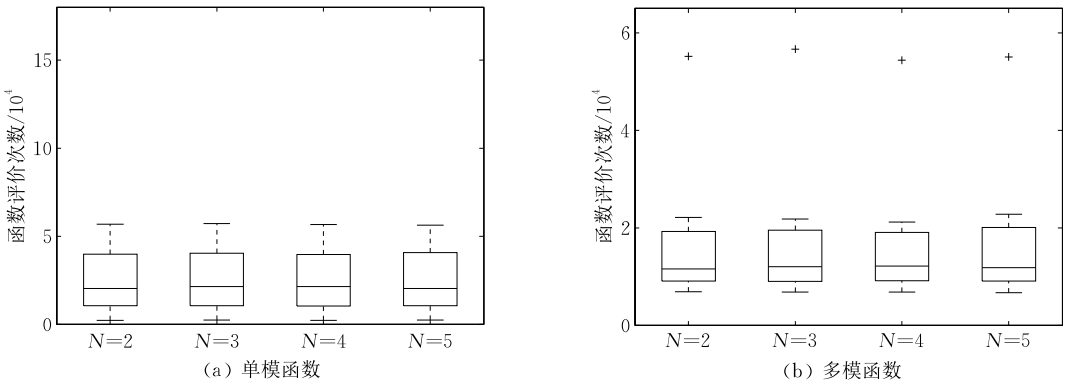


图 9 参数 N 不同设置下的函数评价次数箱型对比

5 总 结

本文提出了一种基于局部 Lipschitz 下界估计支撑面的差分进化算法来减少智能优化算法的函数评价次数. 所提算法通过提取新个体的 N 邻近个体构建基于 Lipschitz 估计理论的下界估计支撑面，从而利用下界估计支撑面获取信息设计 3 种策略来指导种群进化：局部 Lipschitz 估计选择策略，无效区域排除策略和广义下降方向局部增强策略. 局部 Lipschitz 估计选择策略根据下界支撑面获取新个体的下界估计值设计选择策略指导种群更新；无效区域排除策略利用 Lipschitz 下界估计分段线性特性，根据下界估计区域的极值信息排除部分无效区域，逐步缩小搜索区域；广义下降方向局部增强策略则结合邻近个体下降方向和主导支撑面下降方向的组合方向来指导算法作局部增强，提高算法的局部搜索能力. 数值实验结果表明，与传统 DE 算法、基于估计的改进 DE 算法、基于自适应机制的改进 DE 算法及一些先进的非 DE 算法相比，DELLU 算法能够对大部分测试函数以较少的目标函数评价次数求得高质量的解. 另外，所提出的 3 种策略不仅限于 DE 算法，而且可以应用到其他智能优化算法中. 下一步的工作将集中在局部 Lipschitz 下界估计智能优化算法在实际优化问题中的应用.

致　谢　感谢中南大学王勇教授在文献[34]中提供的 CoDE、SaDE 和 EPSDE 算法代码，同时感谢对本文工作给予支持和建议的所有评审专家和同行！

参　考　文　献

[1] Khoo K G, Suganthan P N. Structural pattern recognition using genetic algorithms with specialized operators. IEEE Transactions on Systems, Man, and Cybernetics, 2003, 33(1): 156-165

[2] Storn R, Price K. Differential evolution—A simple and efficient heuristic for global optimization over continuous spaces. Journal of Global Optimization, 1997, 11(4): 341-359

[3] Kennedy J, Eberhart, R. Particle swarm optimization// Proceedings of the IEEE International Conference on Neural Networks. New York, USA, 1995: 1942-1948

[4] Dorigo M, Birattari M. Ant Colony Optimization. Berlin, Heidelberg: Springer-Verlag, 2010

[5] Glotić A, Glotić A, Kitak P, et al. Parallel self-adaptive differential evolution algorithm for solving short-term hydro scheduling problem. IEEE Transactions on Power Systems, 2014, 29(5): 2347-2357

[6] Xia Ya-Mei, Cheng Bo, Chen Jun-Liang, Meng Xiang-Wu, Liu Dong. Optimizing services composition based on improved ant colony algorithm. Chinese Journal of Computers, 2012, 35(2): 270-281(in Chinese)
(夏亚梅，程渤，陈俊亮，孟祥武，刘栋. 基于改进蚁群算法的服务组合优化. 计算机学报，2012，35(2)：270-281)

[7] Wen Tao, Sheng Guo-Jun, Guo Quan, Li Ying-Qiu. Web service composition based on modified particle swarm optimization. Chinese Journal of Computers, 2013, 36(5): 1031-1046(in Chinese)
(温涛，盛国军，郭权，李迎秋. 基于改进粒子群算法的 Web 服务组合. 计算机学报，2013，36(5)：1031-1046)

[8] Liu Y, Sun Fang. A fast differential evolution algorithm using k-Nearest Neighbour predictor. Expert Systems with Applications, 2011, 38(4): 4354-4258

[9] Müller J, Piché R. Mixture surrogate models based on Dempster-Shafer theory for global optimization problems. Journal of Global Optimization, 2011, 51(1): 79-104

[10] Liu B, Zhang Q, Gielen G. A Gaussian process surrogate model assisted evolutionary algorithm for medium scale expensive optimization problems. IEEE Transactions on Evolutionary Computation, 2014, 18(2): 180-192

[11] Zhou Z, Ong Y S, Lim M H, et al. Memetic algorithm using multi-surrogates for computationally expensive optimization problems. Soft Computing, 2007, 11(10): 957-971

[12] Lim D, Jin Y, Ong Y S, Sendhoff B. Generalizing surrogate-assisted evolutionary computation. IEEE Transactions on Evolutionary Computation, 2010, 14(3): 329-355

[13] Park S Y, Lee J J. An efficient differential evolution using speeded-up k-Nearest Neighbor estimator. Soft Computing, 2014, 18(1): 35-49

[14] Pham H A. Reduction of function evaluation in differential evolution using nearest neighbor comparison. Vietnam Journal of Computer Science, 2015, 2(2): 121-131

[15] Rubinov A, Andramonov M. Lipschitz programming via increasing convex-along-rays functions. Optimization Methods and Software, 1999, 10(6): 763-781

[16] Beliakov G, Lim K F. Challenges of continuous global optimization in molecular structure prediction. European Journal of Operational Research, 2007, 181(3): 1198-1213

[17] Beliakov G, Ferrer A. Bounded lower subdifferentiability optimization techniques: Applications. Journal of Global Optimization, 2010, 47(2): 211-231

[18] Auslender A, Ferrer A, Goberna M A, et al. Comparative study of RPSALG algorithm for convex semi-infinite programming. Computational Optimization and Applications, 2015, 60(1): 59-87

[19] Bagirov A M, Rubinov A M. Global minimization of increasing positively homogeneous functions over the unit simplex. Annals of Operations Research, 2000, 98(1-4): 171-187

[20] Belikov G. Extended cutting angle method of global optimization. Pacific Journal of Optimization, 2008, 4(1): 153-175

[21] Rubinov A M. Abstract Convexity and Global Optimization. Nonconvex Optimization and Its Applications. Dordrecht: Kluwer Academic Publishers, 2000

[22] Zhang Gui-Jun, He Yang-Jun, Guo Hai-Feng, Feng Yuan-Jing, Xu Jian-Ming. Differential evolution algorithm for multimodal optimization based on abstract convex underestimation. Journal of Software, 2013, 24(6): 1177-1195(in Chinese)
(张贵军，何洋军，郭海锋，冯远静，徐建明. 基于广义凸下界估计的多模态差分进化算法. 软件学报，2013，4(6)：1177-1195)

[23] Zhang Gui-Jun, Zhou Xiao-Gen. Population-based global optimization algorithm using abstract convex underestimate. Control and Decision, 2015, 30(6): 1116-1120(in Chinese)
(张贵军，周晓根. 基于抽象凸下界估计的群体全局优化算法. 控制与决策，2015，30(6)：1116-1120)

[24] Zhou Xiao-Gen, Zhang Gui-Jun, Mei Shan, Ming Jie. Differential evolution algorithm based on abstract convex underestimate selection strategy. Control Theory & Applications, 2015, 32(3): 388-397(in Chinese)
(周晓根，张贵军，梅珊，明洁. 基于抽象凸估计选择策略的差分进化算法. 控制理论与应用，2015，32(3)：388-397)

[25] Batten L, Beliakov G. Fast algorithm for the cutting angle method of global optimization. Journal of Global Optimization, 2002, 24(10): 149-161

[26] Floudas C A. Deterministic Global Optimization: Theory, Methods, and Applications. Dordrecht: Kluwer Academic Publishers, 2000

[27] Wang H, Rahnamayan S, Sun H, et al. Gaussian bare-bones differential evolution. IEEE Transactions on Cybernetics, 2013, 43(2): 634-647

[28] Ali M M, Khompatraporn C, Zelda B Z. A numerical evaluation of several stochastic algorithms on selected continuous global optimization test problems. Journal of Global Optimization, 2005, 31(4): 635-672

[29] Sarker R A, Elsayed S M, Ray T. Differential evolution with dynamic parameters selection for optimization problems. IEEE Transactions on Evolutionary Computation, 2014, 18(5): 689-707

[30] Corder G W, Foreman D I. Nonparametric Statistics for Non-Statisticians: A Step-By-Step Approach. Hoboken: John Wiley & Sons, 2009

[31] Shang Y W, Qiu Y H. A note on the extended Rosenbrock function. Evolutionary Computation, 2006, 14(1): 119-126

[32] Qin A K, Huang V L, Suganthan P N. Differential evolution algorithm with strategy adaptation for global numerical optimization. IEEE Transactions on Evolutionary Computation, 2009, 13(2): 398-471

[33] Mallipeddi R, Suganthan P N, Pan Q K, et al. Differential evolution algorithm with ensemble of parameters and mutation strategies. Applied Soft Computing, 2011, 11(2): 1679-1696

[34] Wang Y, Cai Z X, Zhang Q F. Differential evolution with composite trial vector generation strategies and control parameters. IEEE Transactions on Evolutionary Computation, 2011, 15(1): 55-64

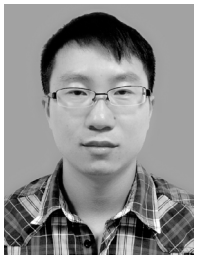
[35] Barbosa H J C, Bernardino H S, Barreto A M S. Using performance profiles to analyze the results of the 2006 CEC constrained optimization competition//Proceedings of the IEEE Congress on Evolutionary Computation (CEC) 2010. New York, USA, 2010: 1-8

[36] Liang J J, Qu B Y, Suganthan P N, et al. Problem definitions and evaluation criteria for the CEC 2013 special session on real-parameter optimization. Nanyang Technological University, Singapore: Technical Report 201212, 2013

[37] Liang J J, Qin A K, Suganthan P N, et al. Comprehensive learning particle swarm optimizer for global optimization of multimodal functions. IEEE Transactions on Evolutionary Computation, 2006, 10(3): 281-295

[38] Hansen N, Ostermeier A. Completely derandomized self-adaptation in evolution strategies. Evolutionary Computation, 2001, 9(2): 159-195

[39] Garcia-Martinez C, Lozano M, Herrera F, et al. Global and local real-coded genetic algorithms based on parent-centric crossover operators. European Journal of Operational Research, 2008, 185(3): 1088-1113



ZHOU Xiao-Gen, born in 1987, Ph. D. candidate. His main research interests include intelligent information processing, optimization theory and algorithm design.

ZHANG Gui-Jun, born in 1974, Ph. D., professor, Ph. D. supervisor. His main research interests include intelligent information processing, optimization theory and algorithm design and bioinformatics.

HAO Xiao-Hu, born in 1990, M. S. candidate. His main research interests include intelligent information processing and bioinformatics.

YU Li, born in 1961, Ph. D., professor, Ph. D. supervisor. His main research interests include intelligent control, decentralized control and networked control systems.

Background

Intelligent optimization algorithms have been successfully applied to solve numerous optimization problems in various fields. The intelligent optimization algorithms include Differential Evolution (DE), Genetic Algorithm (GA), Particle Swarm Optimization (PSO), Ant Colony Optimization (ACO) and so on. The properties like derivative-free and strong robustness make these algorithms attractive for applications to various real-world optimization problems. However, one of the main drawbacks of using these algorithms is that they require a large number of function evaluations to find optimal solutions. Especially, for the expensive-to-evaluate optimization problems in real-world applications, the function evaluation is very time consuming, generally constrained by the time needed to run simulation models. For example, each simulation of a coarse hydrodynamic 2D model may take up to 1 minute to run on a high performance computer. As the spatial and temporal resolution increases, the simulation time increases. One can expect the simulation time to increase to several minutes or even hours for full hydrodynamic 3D models. Thus it is important to reduce the number of real function evaluations for intelligent optimization algorithms.

Very recently, many time-efficient function approximate techniques were developed to reduce the number of function evaluations. Tenne proposed a Memetic Algorithm (MA)

using variable global and local surrogate-models. Paschalidis proposed a semidefinite programming-based underestimation method (SDU) using a class of general convex quadratic functions for underestimation. Liu proposed a function approximate model using the cheap k-nearest neighbour predictor constructed through dynamic learning. Jin combined a Multi-Layer Perceptron (MLP), a kind of Neural Network (NN), with Covariance Matrix Adaptation Evolution Strategy (CMA-ES) to build an efficient evolutionary optimization with function approximation. Zhang developed a predictive distribution model combining Gaussian stochastic model with fuzzy clustering based model to measure the expected improvement of each individual. All the above algorithms show approximate models can be particularly beneficial to reducing the number of function evaluations.

To reduce the number of function evaluations of intelligent optimization algorithms, we proposed a population-based global optimization algorithm using abstract convex underestimate, a differential evolution algorithm based on abstract convex underestimate selection strategy and a differential evolution algorithm with local abstract convex region partition. Numerical experiment results show these proposed algorithms can reduce the number of function evaluations effectively. In this paper, an efficient differential evolution algorithm based on local Lipschitz underestimate supporting hyperplanes is proposed, named DELLU. In DELLU, the supporting hyperplanes of the neighboring individuals of the trial individual are constructed. Then the underestimate of the trial individual is calculated through the supporting hyperplanes of its neighboring individuals to guide the evolutionary process. The results obtained from the unconstrained benchmark functions indicate that the proposed algorithm significantly better than or at least comparable to the traditional DE, some state-of-the-art DE variants and other compared non-DE intelligent optimization algorithms.

This work is supported by the National Natural Science Foundation of China under Grant Nos. 61075062, 61573317, the Natural Science Foundation of Zhejiang Province under Grant No. LY13F030008, the Public Welfare Project of Science Technology Department of Zhejiang Province under Grant No. 2014C33088, and the Open Fund for Key-Key Discipline of Zhejiang Province under Grant Nos. 20151008, 20151015.