第47卷 第4期 计 算 机 工 程 2021年4月
Vol.47 No.4 Computer Engineering April 2021

·人工智能与模式识别· 文章编号:1000-3428(2021)04-0084-08 文献标志码:A 中图分类号:TP301.6

一种求解动态优化问题的改进自适应差分进化算法

刘树强,秦 进

(贵州大学 计算机科学与技术学院,贵阳 550025)

摘 要: 针对原始动态自适应差分进化(SADE)算法局部搜索能力弱和寻优精度低的问题,提出一种求解动态优化问题的邻域搜索差分进化(NSDE)算法。通过引入邻域搜索机制,在划分种群最优个体的邻域空间范围内产生候选解,选取候选解集合中的最优解并对种群最优个体进行迭代,增强算法局部搜索能力。在传统基于距离的排斥方案中,引入hill-valley函数追踪邻近峰,提高算法寻优精度。实验结果表明,与SADE、人工免疫网络动态优化、多种群竞争差分进化和改进差分进化算法相比,NSDE算法在49个测试问题中分别有28、38、29和38个测试问题的平均误差更小,综合性能表现更好。

关键词: 自适应差分进化;动态优化问题;邻域搜索;排斥方案;平均误差

开放科学(资源服务)标志码(OSID):

中文引用格式: 刘树强,秦进.一种求解动态优化问题的改进自适应差分进化算法[J].计算机工程,2021,47(4):84-91,99.

英文引用格式: LIU Shuqiang,QIN Jin.An improved self-adaptive differential evolution algorithm for solving dynamic optimization problem[J].Computer Engineering,2021,47(4):84-91,99.

An Improved Self-Adaptive Differential Evolution Algorithm for Solving Dynamic Optimization Problem

LIU Shuqiang,QIN Jin

(College of Computer Science and Technology,Guizhou University,Guiyang 550025,China)

【Abstract】 To address the weak local search ability and low optimization accuracy of the original dynamic Self-Adaptive Differential Evolution(SADE) algorithm,this paper proposes a Neighborhood Search Differential Evolution(NSDE) algorithm for solving Dynamic Optimization Problem(DOP).By introducing the neighborhood search mechanism,the neighborhood space of the best individual of the population is divided properly,and a set of candidate solutions are generated within the divided space.The optimal solution in this set is selected to iterate the best individual of the population,which enhances the local search ability of the algorithm.At the same time,the hill-valley function is introduced into the traditional distance-based exclusion scheme to track the adjacent peaks to improve the optimization accuracy of the algorithm.Experimental results show that in terms of the average error, NSDE algorithm outperforms the original dynamic SADE on 28 test problems(49 test problems in total), Dynamic Optimization of Artificial Immune Network(dopt-aiNet) on 38 problems, Differential Evolution with Competitive Strategy Based on Multi-Population(DECS) on 29 problems, and Modified Differential Evolution(MDE) on 38 problems, which demonstrates the NSDE algorithm has better overall performance.

【Key words】 Self-Adaptive Differential Evolution(SADE); Dynamic Optimization Problem(DOP); neighborhood search;exclusion scheme;average error

DOI: 10. 19678/j. issn. 1000-3428. 0057567

0 概述

在现实生活中很多优化问题均具有动态变化特性[1],例如在调度问题中,随着新任务的到达,机器可能随时发生故障,原材料质量也会随时间产生变化,而静态优化方法无法有效解决上述问题,因此动态优化方法应运而生[2]。当前研究通常将动态优化问题(Dynamic Optimization Problem,DOP)视为一系列静态优化问题(Static Optimization Problem,SOP)的组合,由于目标函数随时改变,因此会导致

基金项目: 国家自然科学基金(61562009)。

作者简介: 刘树强(1994—),男,硕士研究生,主研方向为智能计算、机器学习;秦 进,副教授、博士。

收稿日期: 2020-03-03 **修回日期:** 2020-04-13 **E-mail:** flyhacker666@gmail.com

最优解位置和搜索空间特征的不断变化[3]，而解决动态优化问题的关键是控制解的多样性[4]。传统演化算法在解决静态优化问题方面取得了较好的效果，但不能很好地解决动态优化问题，因为其在运行一段时间后会收敛到固定值，失去探索区域所需的多样性，导致不能跟踪到新环境下已变化的最优解[5]。在动态优化问题中，对不断变化的最优解的快速追踪与找到最优解本身同等重要[6]，这就要求动态优化问题的求解方法同时具备快速收敛和保持多样性的能力。国内外许多研究人员对传统演化算法进行了相关改进，使其能够更好地解决动态优化问题。在过去十几年中，研究人员提出了许多求解动态优化问题的算法及提升算法性能的策略[7]。

差分进化算法与多数演化算法相比，实现过程更简单和直观，具备良好的全局搜索能力，适用于处理大规模问题[8]。差分进化算法不仅在求解静态问题方面表现出较好的性能，而且被证明可用于解决动态优化问题[9]。自适应差分进化（Self-Adaptive Differential Evolution，SADE）是差分进化的分支，通过适应性地改变算法参数值有效处理动态优化问题[10]。文献[11]在动态自适应差分进化算法中引入种群竞争机制。文献[12]认为动态环境下的自适应分为元启发级别、DOP机制级别以及两者组合3个主要的应用级别。文献[13]提出一种动态环境下的自适应差分进化算法，不仅使用自适应控制机制改变其中的控制参数，而且采用带老化机制的多种群方法处理停滞问题，并利用存档的个体重新初始化得到新个体。文献[14]提出一种改进的动态自适应差分进化算法DDEBQ，该算法使用布朗个体和自适应量子个体与差分进化个体共同维持种群的多样性和探索能力，还利用邻域驱动的双变异策略控制摄动以防止过快收敛，并通过排斥规则将子种群分布到更大的搜索空间以加强最优值追踪能力，同时使用老化机制防止算法陷入局部最优。

求解动态优化问题的进化算法需要探索与开发之间的平衡，探索对应算法的全局搜索能力，开发对应算法的局部搜索能力。进化算法既不能陷于局部最优，过分开发某一局部区域，丧失探索其他区域的能力，也不能一直在探索其他区域而不收敛或过慢收敛，忽视开发某一局部区域。差分进化算法是全局搜索能力较好的进化算法，但在演化后期通常收敛慢，局部搜索能力有待提升。自适应差分进化算法虽然通过算子的适应性变化提高了寻优精度，但是参数值的变化仍不能为种群提供足够的多样性。本文提出一种邻域搜索差分进化（Neighborhood Search Differential Evolution，NSDE）算法，运用邻域搜索机制增强差分进化算法中每个子种群的局部搜索能力，在传统基于距离的排斥方案基础上引入hill-valley函数追踪邻近峰以提高寻优精度。

1　邻域搜索差分进化算法

1.1　动态自适应差分进化算法

SADE使用rand/1/bin策略的自适应控制机制，在算法运行过程中改变控制参数F和CR，而种群规模NP在演化过程中保持不变。自适应控制参数F_i^{G+1}和CR_i^{G+1}的计算公式如下：

$$F_i^{G+1}=\begin{cases}F_l+\mathrm{rand}_1\cdot F_u, & \mathrm{rand}_2<\tau_1\\ F_i^G, & \text{其他}\end{cases}$$

$$\mathrm{CR}_i^{G+1}=\begin{cases}\mathrm{rand}_3, & \mathrm{rand}_4<\tau_2\\ \mathrm{CR}_i^G, & \text{其他}\end{cases}$$

其中：$\mathrm{rand}_j, j\in\{1,2,3,4\}$表示取值为[0，1]的均匀随机数；$\tau_1$和$\tau_2$分别为控制参数$F$和CR的调整概率；$\tau_1$、$\tau_2$、$F_l$、$F_u$分别取固定值0.1、0.1、0.1、0.9；$F$取值为[0.1，1.0]的随机数；CR取值为[0，1]的随机数；F_i^{G+1}和CR_i^{G+1}通常在变异前计算得到，从而影响新向量$\boldsymbol{x}_i^{G+1}$的变异、交叉和选择。

在原始动态SADE算法[13]中，老化机制的第i个个体的老化算法以及改进和老化算法如算法1和算法2所示，利用存档的个体重新初始化算法如算法3所示，其中，subSize是子种群大小，ARC是关于个体的存档，arcNum是存档大小，mRand()是产生随机数的函数，age是个体年龄。

算法1　第i个个体的老化算法

1.if 第i个个体是全局最优个体 then 不使用老化
2.else if 第i个个体是局部最优个体（子种群中的最优个体）and age>30 and mRand()<0.1 then
3.重新初始化包含第i个个体的子种群
4.else if age>25 and mRand()<0.1 then
5.重新初始化第i个个体

算法2　改进和老化算法

1.if Distance(x，y)<0.01 or |x.fitness−y.fitness|<0.1 then
2.age=min{age，20}
3.else
4.age=min{age，5}

算法3　重新初始化算法

1.if mRand()<0.5 and i<subSize and arcNum>0 and i<arcNum then
2.产生取值为[0，arcNum]的随机数t
3.y=ARC[t]
4.$w=\dfrac{0.1}{1+(i \bmod subSize)}$
5.if (i mod 2)==1 then
6.y = y +w×N(0，1)

7.else
8.y = y +w×U(−0.5,0.5)
9.end if
10.else
11.随机产生y
12.end if

原始动态SADE算法的具体内容详见文献[13]，本文在原始动态SADE算法的基础上提出NSDE算法（如算法4所示），主要改进为邻域搜索和排斥算法。

算法4 NSDE算法

1.初始化算法参数
2.随机初始化并评估所有的子种群
3.while 不满足终止条件 do
4.for 子种群i do
5.for 个体j do
6.产生新的F值和CR值
7.个体j的老化（算法1）
8.变异
9.交叉
10.排斥（算法6）
11.选择
12.改进和老化（算法2）
13.个体j的age加1
14.end
15.得到当前子种群i的最优个体x
16.邻域搜索（算法5）
17.end
18.end

1.2 邻域搜索机制

种群最优个体邻域范围内的解会有较高的可能性靠近最优值点，该可能性随着维度的增加而增加。本文提出一种种群最优个体的邻域解生成方式，首先利用自适应差分进化算法产生初步的种群最优个体，然后对种群最优个体的邻域空间进行适当划分，分别在不同范围内产生候选解，构成候选解集合，最后选取集合中的最优解，对种群最优个体进行迭代，以期逼近最优值点。

组成动态优化问题的静态优化问题可表示为：

$$\min f(\boldsymbol{X}),\ \text{s.t. } \boldsymbol{L}\leqslant\boldsymbol{X}\leqslant\boldsymbol{U} \tag{1}$$

其中，$\boldsymbol{X}$、$\boldsymbol{L}$、$\boldsymbol{U}$均为n维向量，$\boldsymbol{X}=(x_1,x_2,\cdots,x_n)$，$\boldsymbol{L}=(l,l,\cdots,l)$，$\boldsymbol{U}=(u,u,\cdots,u)$，$\boldsymbol{X}$各分变量的取值均为$[l,u]$。

NSDE算法利用矩形定义点的邻域，通过同心超矩形划分当前解分量$x_j(j=1,2,\cdots,n)$的邻域空间[15]，如图1所示，假设分成k个空间，按式(2)划分每个空间大小：

$$\begin{aligned}&R_i(x,\ r_{i-1},\ r_i)=\{x'\mid r_{i-1,j}\leqslant|x'_j-x_j|<r_{i,j}\}\\&l<x'_j<u,\ j=1,\ 2,\ \cdots,\ n,\ i=1,\ 2,\ \cdots,\ k\\&r_i=r_{i-1}\times 2,\ i=1,\ 2,\ \cdots,\ k\end{aligned} \tag{2}$$

其中：

$$\begin{aligned}&R_0(x,\ r_0)=\{x'\mid|x'_j-x_j|<r_0\}\\&l<x'_j<u,\ j=1,\ 2,\ \cdots,\ n\end{aligned} \tag{3}$$

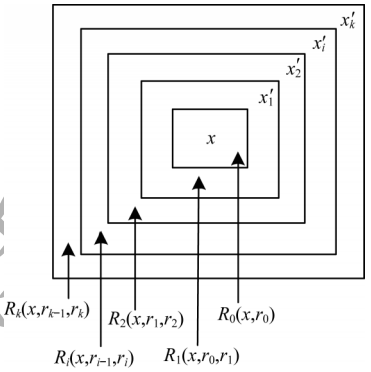


图1 邻域空间划分

Fig.1 Division of neighborhood space

在$R_i(i=1,2,\cdots,k)$的每个同心矩形内随机选取1个点，这k个点构成当前解分量x_j的候选值集合$\{x'_1,x'_2,\cdots,x'_k\}$（如算法5所示），整个候选解的产生如图2所示，其中，MaxIter是迭代次数，mRand()是产生随机数的函数，candiNum是邻域候选解个数，r是邻域半径r_0，x是某个子种群中的最优个体，x_i是x的第i维的值，candiSol′是得到的候选解集合，BestSol是候选解集合中的最优解。

算法5 邻域搜索算法

1.iter←0
2.while iter<MaxIter do
3.seed=mRand()
4.for i←1 to D do
5.for j←1 to candiNum do
6.if seed<0.5 then
7.candiSol(i,j)=r×2^{j-1}+x_i+mRand()×r×2^{j-1}
8.else
9.candiSol(i,j)=x_i−r×2^j+mRand()×r×2^{j-1}
10.end
11.end
12.end
13.将candiSol视为矩阵进行转置，得到candiSol′
14.BestSol = $\begin{cases}\text{argmax}_{\text{candiSol}'(i,j)}(\text{candiSol}'(i,j).\text{fitness}),\\ i\in[1,\text{candiNum}],j\in[1,D],\text{F1}\\ \text{argmin}_{\text{candiSol}'(i,j)}(\text{candiSol}'(i,j).\text{fitness}),\\ i\in[1,\text{candiNum}],j\in[1,D],\text{F2~F6}\end{cases}$
15.x = $\begin{cases}\text{BestSol, F1下BestSol.fitness > x.fitness,}\\ \text{F2~F6下BestSol.fitness < x.fitness}\\ \text{x, 其他}\end{cases}$
16.iter←iter+1
17.end

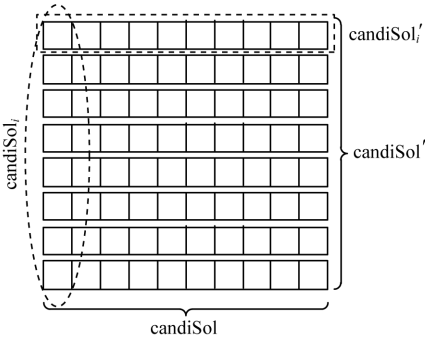


图2　候选解的产生

Fig.2　Generation of candidate solutions

1.3　排斥方案

排斥方案的关键为将每个子种群维持在适应度（fitness）范围内的不同峰上，子种群可以由其中的最优个体代表。每个子种群实施一种排斥方案，如果与其他子种群在同一个峰上，那么较好的子种群保持不变并重新初始化较坏的子种群。文献[16]提出一种基于距离的排斥方案，该方案假设所有的峰均匀分布在整个搜索空间中，如果两个子种群间的距离小于阈值 $r_{\text{excl}}=A/2m^{\frac{1}{n}}$，那么较坏的子种群被重新初始化，其中，$n$是维数，$A$是解的范围，$m$是峰数。

基于距离的排斥方案中的两个峰可能很接近，以至于它们之间的距离比排斥方案的阈值还小，在此情况下很难同时找到这两个峰。本文在基于距离的排斥方案的基础上增添一个hill-valley函数[9]，如图3所示。当满足基于距离的排斥方案条件时，判断是否存在满足$f(\boldsymbol{z})<\min\{f(\boldsymbol{x}),f(\boldsymbol{y})\}$，如果不满足，则最终进行排斥操作（如算法6所示），其中，$\boldsymbol{x}$和$\boldsymbol{y}$分别是两个子种群中的最优个体，$\boldsymbol{z}=c\cdot\boldsymbol{x}+(1-c)\cdot\boldsymbol{y}$, $c\in\{0.05, 0.50, 0.95\}$。

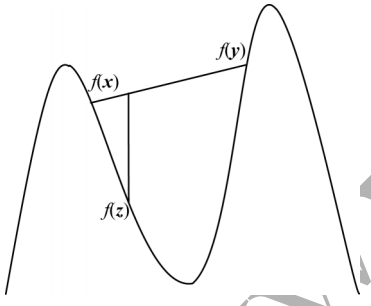


图3　基于hill-valley函数的排斥方案

Fig.3　Exclusion scheme based on hill-valley function

算法6　排斥算法

1.z_1=0.05×x+（1−0.05）×y

2.z_2=0.5×x+（1−0.5）×y

3.z_3=0.95×x+（1−0.95）×y

4.if Distance(x,y)<r_{excl} and min {z_1.fitness, z_2.fitness, z_3.fitness } ≥ min{x.fitness, y.fitness} then

5.r = {x, F1下x.fitness < y.fitness, F2~F6下y.fitness < x.fitness；y, 其他

6.重新初始化包含r的子种群（算法3）

7.end

2　实验与结果分析

2.1　对比算法

在求解动态优化问题的演化算法中，本文采用人工免疫网络动态优化（Dynamic Optimization of Artificial Immune Network，dopt-aiNet）算法[17]、原始动态SADE算法（简称SADE）[13]、多种群竞争差分进化（Differential Evolution with Competitive Strategy Based on Multi-Population，DECS）算法[18]和改进差分进化（Modified Differential Evolution，MDE）算法[19]与NSDE算法进行对比，具体为：1）dopt-aiNet算法对opt-aiNet算法[20]进行扩展，增加了分离的记忆种群、新的变异算子和种群规模最大值及控制衰退的搜索过程和亲近度测量过程，可避免opt-aiNet算法的细胞数盲目扩增问题；2）SADE算法使用自适应控制机制、多种群机制与老化机制改变控制参数；3）DECS算法将竞争机制融入多种群差分进化算法中，增强算法寻优能力；4）MDE算法将种群分为跟踪和搜索两个子种群，对两个子种群采用不同的变异策略，利用跟踪种群判断环境变化，采用搜索种群扩大搜索范围。为避免算法结果的复现问题，SADE算法的实验数据由实际仿真得到，而dopt-aiNet、DECS和MDE算法的实验数据来自文献[17-18]。

2.2　测试函数与算法参数设置

本文使用GDBG生成器[21]验证NSDE算法的有效性。GDBG是一种广义动态benchmark生成器，构造二元空间、实空间与组合空间的动态问题，这些问题具有大量的旋转操作和局部最优解以及更高的维度。GDBG包含6种测试函数和7种变化情况，总计49个测试问题，通用参数设置如表1所示，其中高度范围、起始高度和采样频率的单位为1。

表1　GDBG通用参数设置

Table 1　General parameter setting of GDBG

参数名称	参数取值
维数D	固定维数为10、变化维数范围为[5,15]
搜索范围	$[-5,5]^D$
函数或峰的数目	10
变化频率	每10 000×D次适应度评估发生一次变化
变化数目	60
周期数目	12
噪声程度	0.8
混沌常数	3.67
阶路程度α	0.04
α的最大值	0.1
高度范围	[10,100]
起始高度	50
高度变化程度	5.0
采样频率	100

测试函数由表2所示的Sphere、Rastrigin、Weierstrass、Griewank、Ackley这5种函数通过旋转、组合产生，F1测试函数求解最大值优化问题，F2~F6测试函数求解最小值优化问题，其中F1分为带有10个峰和带有50个峰两种情况，具体定义为：

F1：旋转峰函数（10 and 50 peaks）；

F2：Sphere函数的组合函数；

F3：Rastrigin函数的组合函数；

F4：Griewank函数的组合函数；

F5：Ackley函数的组合函数；

F6：混合组合函数。

7种变化类型具体为小步变化（T1）、大步变化（T2）、随机变化（T3）、混沌变化（T4）、周期变化（T5）、带噪声的周期变化（T6）和维度改变的随机变化（T7）。

表2 GDBG基本测试函数

Table 2 Basic test function of GDBG

函数名称	运算公式	取值范围
Sphere	$f(x)=\sum_{i=1}^{n}x_i^2$	[−100,100]
Rastrigin	$f(x)=\sum_{i=1}^{n}(x_i^2-10\cos(2\pi x_i)+10)$	[−5,5]
Weierstrass	$f(x)=\sum_{i=1}^{n}\left(\sum_{k=0}^{k_{\max}}[a^k\cos(2\pi b^k(x_i+0.5))]\right)-n\sum_{k=0}^{k_{\max}}[a^k\cos(\pi b^k)],\ a=0.5,\ b=3,\ k_{\max}=20$	[−0.5,0.5]
Griewank	$f(x)=\frac{1}{4\,000}\sum_{i=1}^{n}(x_i)^2-\prod_{i=1}^{n}\cos\left(\frac{x_i}{\sqrt{i}}\right)+1$	[−100,100]
Ackley	$f(x)=-20\exp\left(-0.2\sqrt{\frac{1}{n}\sum_{i=1}^{n}x_i^2}\right)-\exp\left(\frac{1}{n}\sum_{i=1}^{n}\cos(2\pi x_i)\right)+20+\mathrm{e}$	[−32,32]

算法参数设置如表3所示，种群规模、子种群数、子种群规模、F初始值、CR初始值的取值参考文献[13]，邻域半径、邻域候选解个数、迭代次数的取值通过简单实验计算得到。

表3 算法参数设置

Table 3 Algorithm parameter setting

参数名称	参数取值
种群规模NP	50
子种群数subNum	5
子种群规模subSize	10
F初始值F_{init}	0.5
CR初始值CR_{init}	0.9
邻域半径r	0.005
邻域候选解个数candiNum	8
迭代次数MaxIter	5

2.3 评价指标

本文选用平均误差和标准差作为评价算法性能的指标，平均误差和标准差的计算公式如下：

$$A_{\text{Avg\_mean}}=\frac{\sum_{i=1}^{r_{\text{runs}}}\sum_{j=1}^{n_{\text{num\_change}}}E_{i,j}^{\text{last}}(t)}{r_{\text{runs}}\times n_{\text{num\_change}}} \tag{4}$$

$$S_{\text{STD}}=\sqrt{\frac{\sum_{i=1}^{r_{\text{runs}}}\sum_{j=1}^{n_{\text{num\_change}}}(E_{i,j}^{\text{last}}(t)-A_{\text{Avg\_mean}})^2}{r_{\text{runs}}\times n_{\text{num\_change}}-1}} \tag{5}$$

其中，$n_{\text{num_change}}$表示测试函数的变化次数，r_{runs}表示算法独立运行的次数，$E_{i,j}^{\text{last}}(t)$表示算法在第i次独立运行时第j次变化的适应度绝对误差值，计算公式如下：

$$E^{\text{last}}(t)=|f(\boldsymbol{x}_{\text{best}}(t))-f(\boldsymbol{x}^*(t))| \tag{6}$$

其中，$f(\boldsymbol{x}_{\text{best}}(t))$和$f(\boldsymbol{x}^*(t))$分别表示算法寻得的最优值和理论最优值。

2.4 结果分析

本文在Windows 7 x86_64操作系统、Intel® Core™ i3-3220 CPU、3.30 GHz主频、8 GB RAM、C/C++编程语言环境下进行实验。对6个测试函数进行20次独立测试，实验结果如表4、表5所示，并将5种算法在每个测试函数下得到的平均误差的最小值加粗显示。根据对比实验结果，从测试函数角度分析可知，NSDE算法在F1（50 peaks）、F2、F3、F4和F5测试问题上的精度相比SADE算法提升最为明显，能够更好地应对F1（50 peaks）的峰数增加导致的环境复杂性变化以及F2、F3、F4和F5特征明显不同的环境动态变化。从变化类型角度分析可知，NSDE算法在T1、T2、T5、T6和T7变化类型下的平均误差比SADE算法小，即在小步变化、大步变化、周期变化、带噪声周期变化、维度改变随机变化情况下的寻优能力相较SADE算法更好。可以看出：NSDE算法相比SADE算法在49个测试问题中有28个测试问题的平均误差更小，表明其具有更优的复杂问题求解能力；仅在F2、T1、T7上劣于dopt-aiNet算法，在49个测试问题中有38个测试问题的平均误差更小；仅在F4、T3上劣于MDE算法，在49个测试问题中有38个测试问题的平均误差更小；仅在F2、F4、T2、T7上劣于DECS，在49个测试问题中有29个测试问题的平均误差更小。可见，NSDE算法相比DECS、dopt-aiNet和MDE算法具有更优的复杂问题求解能力。

根据仿真实验结果可以得出NSDE算法在各类测试问题及各种变化类型下的收敛趋势，如图4所示。相对误差（E）的计算如式（7）所示：

$$E=\begin{cases}f(\boldsymbol{x})/f(\boldsymbol{x}^*), & \text{F1}\\ f(\boldsymbol{x}^*)/f(\boldsymbol{x}), & \text{F2~F6}\end{cases} \tag{7}$$

其中，$f(\boldsymbol{x})$是算法寻得的最优值，$f(\boldsymbol{x}^*)$是理论最优值。

NSDE算法收敛曲线反映了收敛速度及陷入局部最优值的次数，进而体现算法响应环境变化的能力。由于F3测试函数的基础组成函数Rastrigin是典型的非线性多模态函数，峰与峰之间的起伏较大，并且具有大量局部极值点，因此算法在F3测试函数中的表现较差，相对而言更容易陷入局部最优而出现停滞状态。但除此之外，NSDE算法在其他测试函数中均表现良好，具有较快的收敛速度，能及时响应环境变化并快速搜索新的全局最优值并保持种群多样性。

表4　NSDE与SADE算法的实验结果

Table 4　Experimental results of NSDE and SADE algorithms

测试函数	算法名称	评价指标	T1	T2	T3	T4	T5	T6	T7
F1(10 peaks)	NSDE	平均误差	**3.093e−01**	4.015e+00	**3.485e+00**	1.095e+00	2.133e+00	**1.926e+00**	4.379e+00
		标准差	1.503e+00	8.134e+00	8.123e+00	3.490e+00	4.491e+00	6.816e+00	8.999e+00
	SADE	平均误差	4.777e−01	**4.004e+00**	3.732e+00	**9.180e−01**	**1.866e+00**	1.974e+00	**4.177e+00**
		标准差	1.953e+00	8.493e+00	7.932e+00	2.781e+00	4.071e+00	6.448e+00	8.900e+00
F1(50 peaks)	NSDE	平均误差	9.007e−01	**4.361e+00**	**4.434e+00**	9.408e−01	**1.115e+00**	**2.637e+00**	**5.034e+00**
		标准差	1.958e+00	6.208e+00	6.513e+00	2.227e+00	2.290e+00	7.647e+00	7.191e+00
	SADE	平均误差	**8.976e−01**	4.929e+00	4.758e+00	**8.203e−01**	1.172e+00	2.690e+00	5.786e+00
		标准差	1.976e+00	7.043e+00	6.817e+00	1.873e+00	2.361e+00	7.400e+00	7.843e+00
F2	NSDE	平均误差	**3.548e+00**	**4.145e+01**	**4.570e+01**	**2.811e+00**	**7.916e+01**	4.897e+00	**3.943e+01**
		标准差	7.107e+00	1.067e+02	1.166e+02	6.082e+00	1.446e+02	2.046e+01	7.955e+01
	SADE	平均误差	4.333e+00	5.350e+01	5.526e+01	3.291e+00	8.146e+01	**4.173e+00**	3.969e+01
		标准差	8.625e+00	1.301e+02	1.307e+02	7.438e+00	1.467e+02	2.160e+01	7.629e+01
F3	NSDE	平均误差	2.125e+01	**5.935e+02**	5.928e+02	**7.811e+01**	4.784e+02	**2.610e+02**	**7.387e+02**
		标准差	9.177e+01	3.745e+02	3.724e+02	2.258e+02	3.819e+02	3.873e+02	3.556e+02
	SADE	平均误差	**1.968e+01**	5.939e+02	**5.739e+02**	1.087e+02	**4.560e+02**	2.693e+02	7.408e+02
		标准差	8.234e+01	3.736e+02	3.814e+02	2.668e+02	3.858e+02	3.949e+02	3.465e+02
F4	NSDE	平均误差	**3.324e+00**	6.578e+01	6.713e+01	**3.902e+00**	**1.025e+02**	**4.868e+00**	7.836e+01
		标准差	6.700e+00	1.546e+02	1.553e+02	7.367e+00	1.758e+02	2.091e+01	1.448e+02
	SADE	平均误差	3.656e+00	**5.373e+01**	**6.132e+01**	4.431e+00	1.327e+02	5.051e+00	**7.806e+01**
		标准差	8.828e+00	1.385e+02	1.458e+02	8.122e+00	1.962e+02	1.114e+01	1.383e+02
F5	NSDE	平均误差	**1.533e+00**	**2.289e+00**	2.046e+00	1.738e+00	**1.462e+00**	1.526e+00	**2.406e+01**
		标准差	4.010e+00	6.242e+00	5.989e+00	5.625e+00	5.348e+00	2.177e+01	8.049e+01
	SADE	平均误差	1.701e+00	3.421e+00	**1.644e+00**	**1.277e+00**	1.481e+00	**1.003e+00**	3.158e+01
		标准差	5.546e+00	3.768e+01	4.893e+00	3.756e+00	6.670e+00	3.958e+00	8.237e+01
F6	NSDE	平均误差	1.073e+01	**1.493e+01**	1.472e+01	1.153e+01	2.025e+01	9.487e+00	**5.129e+01**
		标准差	1.734e+01	2.147e+01	1.769e+01	2.259e+01	7.177e+01	1.312e+01	1.084e+02
	SADE	平均误差	**9.371e+00**	1.785e+01	**1.398e+01**	**1.022e+01**	**1.330e+01**	**9.109e+00**	6.302e+01
		标准差	1.979e+01	4.292e+01	3.069e+01	1.389e+01	3.124e+01	1.215e+01	1.331e+02

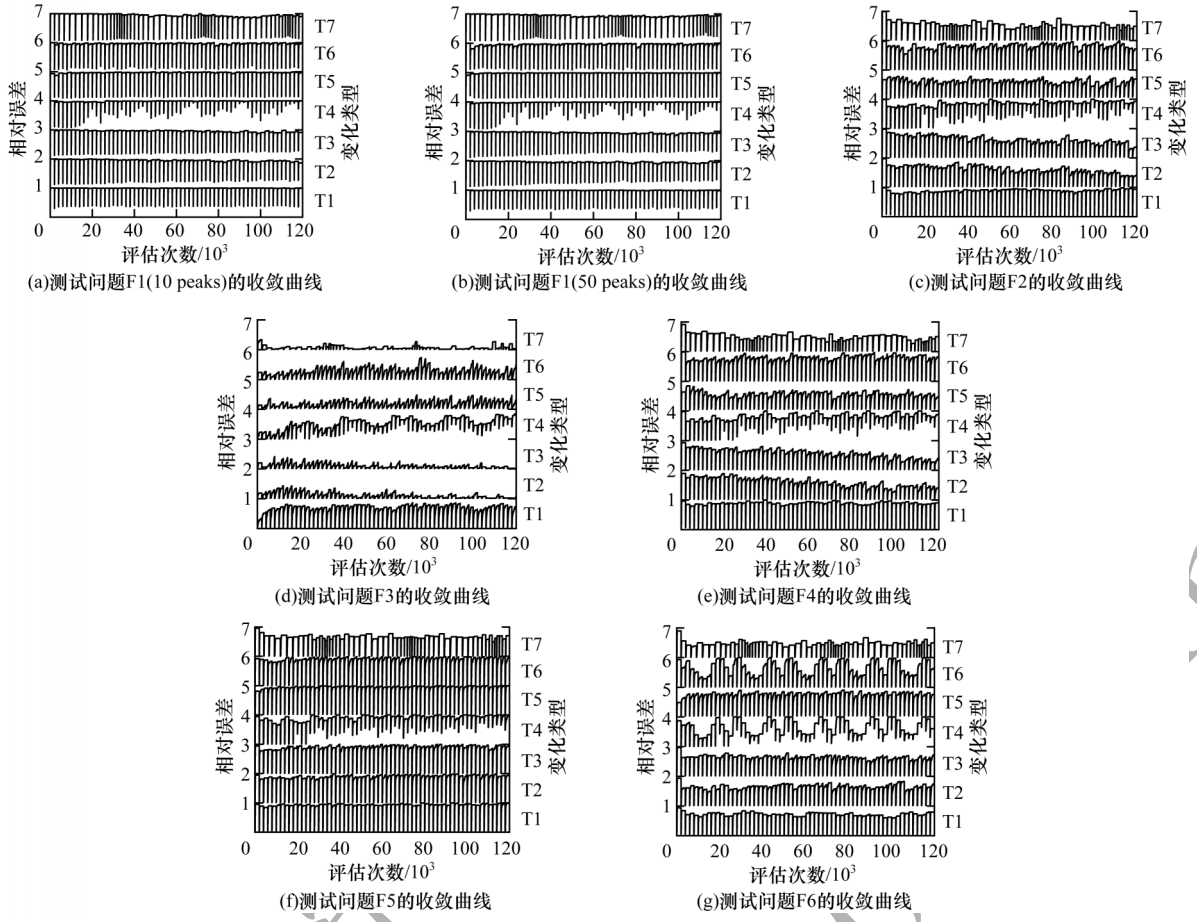
表5　NSDE、DECS、dopt-aiNet和MDE算法的实验结果

Table 5　Experimental results of NSDE，DECS，dopt-aiNet and MDE algorithms

测试函数	算法名称	评价指标	T1	T2	T3	T4	T5	T6	T7
F1(10 peaks)	NSDE	平均误差	3.093e−01	4.015e+00	**3.485e+00**	**1.095e+00**	**2.133e+00**	**1.926e+00**	4.379e+00
		标准差	1.503e+00	8.134e+00	8.123e+00	3.490e+00	4.491e+00	6.816e+00	8.999e+00
	DECS	平均误差	**3.724e−04**	**6.896e−01**	3.864e+00	5.085e+00	9.389e+00	1.536e+01	**1.465e+00**
		标准差	1.139e−03	2.179e+00	6.985e+00	4.644e+00	1.531e+01	1.880e+01	4.614e+00
	dopt-aiNet	平均误差	1.353e−01	5.867e+00	4.255e+00	5.356e+00	4.436e+00	9.941e+00	4.211e+00
		标准差	1.006e+00	1.028e+01	8.183e+00	8.941e+00	5.555e+00	1.582e+01	8.687e+00
	MDE	平均误差	6.931e+00	1.242e+01	1.132e+01	6.514e+00	2.788e+01	2.815e+01	5.090e+00
		标准差	5.216e+00	1.016e+01	9.456e+00	4.908e+00	2.618e+01	1.915e+01	4.393e+00

续表

测试函数	算法名称	评价指标	T1	T2	T3	T4	T5	T6	T7
F1(50 peaks)	NSDE	平均误差	9.007e−01	4.361e+00	**4.434e+00**	**9.408e−01**	**1.115e+00**	**2.637e+00**	5.034e+00
		标准差	1.958e+00	6.208e+00	6.513e+00	2.227e+00	2.290e+00	7.647e+00	7.191e+00
	DECS	平均误差	1.660e+00	**3.580e−01**	4.802e+00	6.491e+00	1.319e+01	2.670e+01	**2.658e+00**
		标准差	1.906e+00	1.006e+00	4.844e+00	7.322e+00	1.275e+01	3.178e+01	7.942e+00
	dopt-aiNet	平均误差	**3.644e−01**	4.749e+00	5.253e+00	2.657e+00	2.864e+00	6.833e+00	4.417e+00
		标准差	9.275e−01	6.758e+00	6.683e+00	5.977e+00	4.158e+00	1.188e+01	6.453e+00
	MDE	平均误差	1.518e+01	1.242e+01	6.522e+00	6.463e+00	1.427e+01	3.558e+01	2.310e+01
		标准差	1.593e+01	1.016e+01	4.149e+00	5.920e+00	1.639e+01	3.175e+01	1.248e+01
F2	NSDE	平均误差	3.548e+00	4.145e+01	4.570e+01	2.811e+00	7.916e+01	**4.897e+00**	3.943e+01
		标准差	7.107e+00	1.067e+02	1.166e+02	6.082e+00	1.446e+02	2.046e+01	7.955e+01
	DECS	平均误差	5.022e−01	1.790e+01	**2.623e+00**	2.436e+02	**4.103e+01**	4.104e+01	2.721e+01
		标准差	1.112e+00	2.627e+01	3.038e+00	1.249e+02	3.630e+01	3.649e+01	1.517e+01
	dopt-aiNet	平均误差	**9.840e−02**	**8.121e+00**	1.800e+01	**1.065e+00**	1.014e+02	6.519e+00	**3.739e+00**
		标准差	2.910e−02	1.438e+01	6.223e+01	2.827e+00	1.345e+02	1.382e+01	7.954e+00
	MDE	平均误差	2.814e+01	3.929e+01	2.801e+01	2.419e+02	4.110e+01	4.070e+01	1.745e+02
		标准差	1.115e+01	2.979e+01	1.017e+01	1.218e+02	3.585e+01	3.578e+01	2.047e+02
F3	NSDE	平均误差	**2.125e+01**	5.935e+02	5.928e+02	**7.811e+01**	**4.784e+02**	2.610e+02	7.387e+02
		标准差	9.177e+01	3.745e+02	3.724e+02	2.258e+02	3.819e+02	3.873e+02	3.556e+02
	DECS	平均误差	6.275e+02	5.817e+02	6.014e+02	6.571e+02	5.662e+02	**4.104e+01**	7.336e+02
		标准差	3.684e+01	1.182e+02	1.081e+01	3.829e+01	1.624e+02	3.649e+01	7.934e+01
	dopt-aiNet	平均误差	8.108e+02	1.079e+03	1.073e+03	1.032e+03	1.024e+03	1.187e+03	1.062e+03
		标准差	6.611e+01	6.412e+01	6.500e+01	2.747e+02	5.787e+01	2.923e+02	1.101e+02
	MDE	平均误差	5.433e+02	**5.760e+02**	**5.162e+02**	6.600e+02	5.592e+02	6.235e+02	**7.263e+02**
		标准差	1.659e+02	2.127e+02	2.480e+02	5.219e+01	1.932e+02	1.140e+02	6.702e+01
F4	NSDE	平均误差	3.324e+00	6.578e+01	6.713e+01	**3.902e+00**	1.025e+02	**4.868e+00**	7.836e+01
		标准差	6.700e+00	1.546e+02	1.553e+02	7.367e+00	1.758e+02	2.091e+01	1.448e+02
	DECS	平均误差	1.377e+01	**3.620e+01**	**2.225e+00**	2.882e+02	**4.103e+01**	4.089e+01	1.450e+02
		标准差	9.994e+00	3.671e+01	2.803e+00	1.471e+02	3.630e+01	3.622e+01	2.117e+02
	dopt-aiNet	平均误差	**1.423e+00**	1.224e+02	9.867e+01	4.263e+00	3.046e+02	1.263e+01	**5.290e+01**
		标准差	4.546e+00	2.016e+02	1.967e+02	9.726e+00	2.032e+02	5.584e+01	1.306e+02
	MDE	平均误差	2.814e+01	3.929e+01	2.801e+01	2.871e+02	4.110e+01	4.108e+01	2.700e+02
		标准差	1.115e+01	2.979e+01	1.017e+01	1.444e+02	3.585e+01	3.577e+01	2.232e+02
F5	NSDE	平均误差	**1.533e+00**	**2.289e+00**	**2.046e+00**	**1.738e+00**	**1.462e+00**	**1.526e+00**	2.406e+01
		标准差	4.010e+00	6.242e+00	5.989e+00	5.625e+00	5.348e+00	2.177e+01	8.049e+01
	DECS	平均误差	1.714e+01	2.173e+01	1.084e+01	1.684e+01	1.119e+01	9.568e+00	**2.117e+01**
		标准差	1.625e+01	3.418e+01	8.155e+00	1.922e+01	1.568e+01	1.446e+01	1.482e+01
	dopt-aiNet	平均误差	4.089e+01	3.445e+01	3.494e+01	1.206e+02	9.432e+02	4.803e+02	2.195e+02
		标准差	2.212e+02	1.199e+02	1.150e+02	2.935e+02	6.333e+02	6.108e+02	4.278e+02
	MDE	平均误差	1.913e+02	1.566e+02	9.475e+01	3.781e+02	3.761e+01	3.744e+01	3.479e+02
		标准差	3.226e+02	2.551e+02	1.441e+02	4.512e+02	2.937e+01	3.192e+01	4.107e+02
F6	NSDE	平均误差	1.073e+01	**1.493e+01**	1.472e+01	**1.153e+01**	**2.025e+01**	**9.487e+00**	5.129e+01
		标准差	1.734e+01	2.147e+01	1.769e+01	2.259e+01	7.177e+01	1.312e+01	1.084e+02
	DECS	平均误差	**8.172e+00**	3.528e+01	**8.376e−01**	2.187e+01	2.434e+01	3.374e+01	1.730e+01
		标准差	8.278e+00	3.080e+01	1.923e+00	2.280e+01	2.759e+01	3.037e+01	1.460e+01
	dopt-aiNet	平均误差	2.044e+01	3.912e+02	4.564e+02	8.397e+01	8.459e+02	4.822e+02	3.725e+02
		标准差	7.932e+01	3.954e+02	4.050e+02	2.202e+02	2.512e+02	4.344e+02	3.947e+02
	MDE	平均误差	2.050e+01	4.792e+01	9.782e+00	8.904e+01	3.716e+01	3.310e+01	**1.492e+01**
		标准差	1.895e+01	2.720e+01	9.931e+00	1.760e+02	2.702e+01	2.834e+01	1.664e+01



(a)测试问题F1(10 peaks)的收敛曲线　(b)测试问题F1(50 peaks)的收敛曲线　(c)测试问题F2的收敛曲线

(d)测试问题F3的收敛曲线　(e)测试问题F4的收敛曲线

(f)测试问题F5的收敛曲线　(g)测试问题F6的收敛曲线

图4　NSDE算法在7个测试问题上的收敛曲线

Fig.4　Convergence curves of NSDE algorithm on seven test questions

3　结束语

本文在原始动态SADE算法的基础上提出一种邻域搜索差分进化(NSDE)算法，产生初步的种群最优个体，对种群最优个体的邻域空间进行划分，并在不同的领域空间范围内产生候选解构成候选解集合，选取集合中的最优解对种群最优个体进行迭代，增强算法局部搜索能力。在基于距离的排斥方案中，引入hill-valley函数提高算法寻优精度。实验结果表明，NSDE算法相比SADE、dopt-aiNet、DECS和MDE算法整体性能更优。后续将对NSDE算法做进一步优化，提升其在处理复杂动态问题时的适用性与稳定性。

参考文献

[1] YANG Zhou, YUAN Yichuan, LUO Tingxing, et al. A hybrid immune algorithm for solving dynamic optimization problems[C]//Proceedings of Chinese Control and Decision Conference. Washington D. C., USA: IEEE Press, 2018: 5326-5332.

[2] FU H B, LEWIS P R, SENDHOFF B, et al. What are dynamic optimization problems?[C]//Proceedings of IEEE Congress on Evolutionary Computation. Washington D. C., USA: IEEE Press, 2014: 1550-1557.

[3] SHI Xuhua, QIAN Feng. An optimization algorithm based on multi-population artificial immune network [C]// Proceedings of International Conference on Natural Computation. Washington D. C., USA: IEEE Press, 2009: 379-383.

[4] RANGINKAMAN A E, KORDESTANI J K. A note on the paper "a multi-population harmony search algorithm with external archive for dynamic optimization problems" by Turky and Abdullah [J]. Information Sciences, 2014, 288(1): 12-14.

[5] CHEN Li, DING Lixin. Survey on dynamic optimization algorithms [J]. Journal of Wuhan University (Natural Science Edition), 2011, 57(3): 255-264. (in Chinese)
陈莉，丁立新. 动态优化算法综述[J]. 武汉大学学报(理学版)，2011，57(3)：255-264.

[6] NGUYEN T T, YANG S X, BRANKE J. Evolutionary dynamic optimization: a survey of the state of the art[J]. Swarm and Evolutionary Computation, 2012, 6: 1-24.

[7] MAVROVOUNIOTIS M, LI C H, YANG S X. A survey of swarm intelligence for dynamic optimization: algorithms and applications[J]. Swarm & Evolutionary Computation, 2017, 33: 1-17.

[8] KORDESTANI J K, RANGINKAMAN A E. A novel framework for improving multi-population algorithms for dynamic optimization problems: a scheduling approach[J]. Swarm and Evolutionary Computation, 2019, 44: 788-805.

[9] ZUO Xingquan, XIAO Li. A DE and PSO based hybrid algorithm for dynamic optimization problems [J]. Soft Computing, 2014, 18(7): 1405-1424.

（下转第99页）

（上接第91页）

[10] TURKY A, ABDULLAH S, DAWOD A. A dual-population multi operators harmony search algorithm for dynamic optimization problems[J]. Computers & Industrial Engineering,2018,117:19-28.
[11] PLESSIS M C D, ENGELBRECHT A P. Self-adaptive competitive differential evolution for dynamic environments[C]//Proceedings of IEEE Symposium on Differential Evolution. Washington D. C. ,USA:IEEE Press, 2011:1-8.
[12] NOVOA-HERNANDEZ P,CORONA C C,PELTA D A. Self-adaptation in dynamic environments—a survey and open issues[J]. International Journal of Bio-Inspired Computation,2016,8(1):1-13.
[13] BREST J,ZAMUDA A,BOSKOVIC B,et al. Dynamic optimization using self-adaptive differential evolution[C]// Proceedings of IEEE Congress on Evolutionary Computation. Washington D. C. ,USA:IEEE Press,2009:415-422.
[14] DAS S, MANDAL A, MUKHERJEE R. An adaptive differential evolution algorithm for global optimization in dynamic environments[J]. IEEE Transactions on Cybernetics,2014,44(6):966-978.
[15] ZHANG Xiaofei,ZHANG Huoming. Improved tabu search algorithm for continuous problems[J]. Journal of China Jiliang University,2010,21(3):69-74. (in Chinese)
张晓菲,张火明. 基于连续函数优化的禁忌搜索算法[J]. 中国计量学院学报,2010,21(3):69-74.
[16] BLACKWELL T,BRANKE J. Multi-swarm optimization in dynamic environments[C]//Proceedings of Workshops on Applications of Evolutionary Computation. Berlin, Germany:Springer,2004:489-500.
[17] DE F F O,VON Z F J. A dynamic artificial immune algorithm applied to challenging benchmarking problems[C]//Proceedings of IEEE Congress on Evolutionary Computation. Washington D. C. ,USA:IEEE Press,2009:423-430.
[18] YUAN Yichuan,YANG Zhou,LUO Tingxing,et al. Multi-population-based competitive differential evolution algorithm for dynamic optimization problem[J]. Journal of Computer Applications,2018,38(5):1254-1260. (in Chinese)
袁亦川,杨洲,罗廷兴,等. 求解动态优化问题的多种群竞争差分进化算法[J]. 计算机应用,2018,38(5):1254-1260.
[19] JIANG Liqiang,QIANG Hongfu,LIU Guangbin. Modified differential evolution for dynamic optimization problems[J]. Journal of Chinese Computer System,2013,34(12):2837-2840. (in Chinese)
姜立强,强洪夫,刘光斌. 求解动态优化问题的改进差分进化算法[J]. 小型微型计算机系统,2013,34(12):2837-2840.
[20] CASTRO L N, TIMMIS J. An artificial immune network for multimodal function optimization[C]//Proceedings of IEEE Congress on Evolutionary Computation. Washington D. C. , USA: IEEE Press, 2002:699-704.
[21] LI C H, YANG S X, NGUYEN T T, et al. Benchmark generator for CEC'2009 competition on dynamic optimization[EB/OL]. [2020-01-18]. https://www. cs. le. ac. uk/people/sy11/Papers/TR-CEC09-DBG. pdf.

编辑　陆燕菲