文章编号: 1001-0920 (2011) 07-0961-06

多 智 能 体 差 分 进 化 算 法

何大阔, 高广宇, 王福利, 刘 阳

(东北大学 a. 教育部暨辽宁省流程工业综合自动化重点实验室, b. 信息科学与工程学院, 沈阳 110819)

摘 要: 基于多智能体与差分进化算法的各自优势, 充分地将对多智能体环境的感知和反作用于环境的能力与差分进化速度和全局寻优能力有机结合, 提出一种多智能体差分进化算法. 引入差分进化算子以提高智能体更新速度并保持群体多样性, 同时应用正交交叉算子以改善智能体协作特性确保有效竞争, 并通过局部寻优算子提高算法的寻优精度. 对几种典型测试函数进行了测试, 实验结果表明所提出的算法具有较强的全局寻优能力.

关键词: 多智能体; 差分进化; 正交交叉算子; 差分进化算子; 局部寻优算子

中图分类号: TP273 **文献标识码:** A

Multi-agent differential evolution algorithm

HE Da-kuo, *GAO Guang-yu*, *WANG Fu-li*, *LIU Yang*

(a. Key Laboratory of Process Industry Automation of Ministry of Education, b. College of Information Science and Engineering, Northeastern University, Shenyang 110819, China. Correspondent: HE Da-kuo, E-mail: hedakuo@mail.sy.ln.cn)

Abstract: Based on the respective strengths of multi-agent and differential evolution algorithm, multi-agent's ability of sensing the environment and reacting to the environment and differential evolution's capacity of the speed and good global optimization are fully combined, and multi-agent differential evolution algorithm is proposed. The proposed differential evolution operator is introduced to improve the update speed of agent, and the diversity of the population is kept. The orthogonal crossover operator is imported to ameliorate cooperation characteristic and compete with their neighbors effectively. The local optimization operator is applied to improve searching precision. Several classic test functions are tested, and the results show that the proposed algorithm can improve the global convergence ability.

Key words: multi-agent; differential evolution; orthogonal crossover operator; differential evolution operator; local searching operator

1 引 言

函数优化问题具有广泛的应用背景, 几乎在科学、商业和工程等领域的各个分支都存在这类问题. 差分进化算法作为一种智能随机搜索算法, 以其快速性和对搜索空间形状、函数类型等无任何限制等优势, 已成为求解函数优化问题的有效方法[1-5]. 但是, 当函数的维数增高时, 搜索空间变大, 变量间耦合度增强, 从而会导致差分进化算法的全局搜索能力下降. 多智能体进化计算以其感知环境并反作用于环境的特点表现出较大的智能特性, 在对复杂问题尤其在高维问题的求解中表现出色, 已经广泛应用于各个学科领域[6-9].

本文针对高维函数优化问题, 有机地将智能体对环境的感知和反作用的能力与差分进化算法的快速性和全局寻优能力相结合, 提出一种多智能体差分进化算法(MADE), 设计了差分进化算子、局部寻优算子等进化算子来实现智能体间的竞争、合作、自学习等行为, 其收敛速度远远高于传统算法. 仿真实验结果显示, 本文方法对高维函数优化问题具有较好的寻优性能.

2 多智能体差分进化算法

用于函数优化的智能体定义如下: 一个智能体 a 表示待优化函数的一个候选解, 其能量为其目标函数值的相反数 (极小问题), 即

$$a \in S,\ \mathrm{Energy}(a) = -f(a). \tag{1}$$

智能体的目的是尽可能地增大自身能量. 所有智

收稿日期: 2010-06-03; **修回日期:** 2010-08-23.

基金项目: 国家自然科学基金项目(60774068, 61004083); 中央高校基本科研业务费专项资金项目.

作者简介: 何大阔(1975−), 男, 副教授, 从事复杂工业过程智能建模、控制、优化的研究; 王福利(1957−), 男, 教授, 博士生导师, 从事复杂工业过程建模与优化、故障诊断等研究.

能体均生存在一个网格环境中, 称为智能体网格, 记为 L. 网格的大小为 $L_{\rm size}\times L_{\rm size}$, 其中 $L_{\rm size}$ 为非零正整数. 每个智能体固定在一个格点上, 记处于第 i 行、第 j 列的智能体为 $L_{i,j}$, $i,j=1,2,\cdots,L_{\rm size}$, 则智能体 $L_{i,j}$ 的邻域为

$$L_{i,j}^{\rm neighbors}\{L_{i',j},L_{i,j'},L_{i'',j},L_{i,j''}\}. \tag{2}$$

其中

$$i'=\begin{cases} i-1,\ i\neq 1;\\ L_{\rm size},\ i=1; \end{cases}\quad j'=\begin{cases} j-1,\ j\neq 1;\\ L_{\rm size},\ j=1; \end{cases}$$
$$i''=\begin{cases} i+1,\ i\neq L_{\rm size};\\ 1,\ i=L_{\rm size}; \end{cases}\quad j''=\begin{cases} j+1,\ j\neq L_{\rm size};\\ 1,\ j=L_{\rm size}. \end{cases} \tag{3}$$

每个智能体不能移动, 只能与其邻域发生相互作用. 智能体网格可表示成图 1 所示的形式, 每个圆圈表示一个智能体, 圈中的数字表示该智能体在网格中所处的位置, 只有存在连线的两个智能体才能发生相互作用.

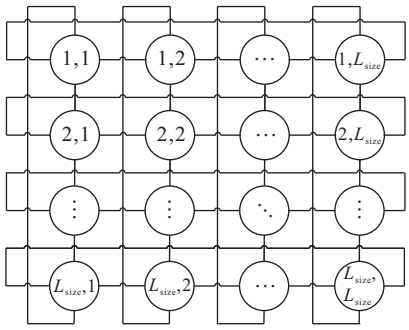

图 1 智能体网格

每个智能体一般通过竞争与合作行为达到增加自身能量的目的. 除了竞争、合作行为外, 智能体还可以利用知识增加自身能量. 根据这些行为, 除邻域竞争算子外, 本文还设计了其他相应算子, 其中邻域竞争算子和正交交叉算子实现智能体的竞争与合作行为, 差分进化算子和局部寻优算子实现智能体利用知识的行为. 邻域竞争算子为多智能体算法常规算子, 在此不再累述. 下面主要介绍正交交叉算子、差分进化算子和局部寻优算子.

2.1 正交交叉算子

正交交叉算子是利用正交设计产生新的个体, 即利用正交设计信息对解空间的代表性, 通过正交设计矩阵确定较小数量并在搜索空间中均匀分布的个体, 从而产生具有代表性的个体. 因此, 将正交交叉算子作用在 $L_{i,j}$ 与 $L_{i,j}^{\max}$ 上, 以形成正交交叉个体.

正交表一般有矩阵直积、特征函数构造哈达马矩阵两种产生方式, 当求解函数的维数增加时, 搜索空间的大小与局部极值的个数也随着问题维数的增高而增长, 能量函数求取时间增加, 函数各维数的耦合性增大, 这些都给高维问题的求解带来困难. 当然, 要应用正交交叉算子必须解决高维正交表的构造问题, 为此, 本文设计了一种近似正交表构造方法:

假设待优化的高维函数为 n 维, 现在有一个 m 维的正交表, 求取 S, 这里

$$S={\rm EVEN}\Big(\frac{n}{m}\Big), \tag{4}$$

其中 EVEN 表示向上取整. 由 $L_q(2^m)$ 组成如下矩阵:

$$H_{SL}=[L_q(2^m)_1,\cdots,L_q(2^m)_S]_{q\times(S\times m)}. \tag{5}$$

在进行正交交叉之前, 随机从高维正交表中选取前 n 列组成正交交叉表.

为了减少计算量, 正交交叉只取正交交叉表中的 $P(2<P\leqslant q)$ 行进行交叉, 即按交叉率 P_c 在所有智能体中选择进行交叉操作的智能体 $L_{i,j}$, 将 $L_{i,j}$ 和 $L_{i,j}^{\max}$ 依据正交交叉表产生 P 个智能体, 并计算所产生的 P 个智能体能量, 利用包括 $L_{i,j}$ 在内的 $P+1$ 个智能体中能量最大的智能体取代 $L_{i,j}$.

正交交叉算子的作用: 多智能体在竞争和合作时, 正交交叉算子可以以最小的代价, 获得在多智能体竞争中产生的有利于增加多智能体能量的所有因素, 增加多智能体自身的优良因素, 从而保证多智能体之间的竞争全部为有效竞争, 使得多智能体的能量不断增大.

2.2 差分进化算子

差分进化是利用父代个体的差异性产生新个体, 保留优良个体, 淘汰劣质的个体, 从而引导搜索过程向最优解逼近[10]. 其优点是鲁棒性好、速度快、在实数域上的搜索能力强. 为了提高多智能体算法的性能, 引入差分进化算法以利用其快速性等特点, 同时加速智能体交流以实现能量增加. 构建差分进化算子, 即以智能体为种群进行一代差分进化. 差分进化算子可以实现快速更新多智能体的作用, 且鲁棒性好、多样性较强, 从而提高了算法的寻优性能.

2.3 局部寻优算子

智能体具有与所求解问题相关的知识, 因此它可以利用这些知识进行局部寻优来提高其性能, 而对智能体进行局部搜索便可实现其局部寻优的行为. 本文根据当前最优个体 ${\rm Agent\_max}\_c^G$ 的信息构建小规模多智能体网络实现局部搜索, 即局部寻优算子.

在局部寻优算子中, 首先产生一个智能体子网格 sL, 其大小为 $sL_{\rm size}\times sL_{\rm size}$($sL_{\rm size}$ 为非零正整数), 其上的所有智能体 $sL_{i',j'}(i',j'=1,2,\cdots,sL_{\rm size})$ 均根据下式产生:

$$sL_{i',j'}=\begin{cases} {\rm Agent\_max}\_c^G,\ i'=1,j'=1;\\ N_{i',j'},\ {\rm otherwise}. \end{cases} \tag{6}$$

式中 $N_{i',j'}=(s_{i',j',1},s_{i',j',2},\cdots,s_{i',j',k},s_{i',j',n})$ 由下

式产生:

$$S_{i',j',k}=\begin{cases}\underline{x}_k,\ A_k+\dfrac{1}{G}\times\text{randn}(1,n)<\underline{x}_k;\\ \overline{x}_k,\ A_k+\dfrac{1}{G}\times\text{randn}(1,n)>\overline{x}_k;\\ A_k+\dfrac{1}{G}\times\text{randn}(1,n),\ \text{otherwise.}\end{cases}\tag{7}$$

其中: $\overline{x}_k$ 和 x_k 为第 k 维变量的上下限, $\text{randn}(1,n)$ 为高斯分布的随机数, G 为进化代数. 变量 $s_{i',j',k}$ 的均值为 A_k, 即 Agent_max_c^G 的第 k 维变量值, 标准差为 $1/G$. 由于局部寻优算子的作用是在最优解附近搜索更优的解, 而高斯变异有利于在最优点附近快速向最优点无限靠近, 同时随着进化代数 G 的增加, 高斯变异的标准差变小, 从而实现由粗到细的搜索过程. 局部寻优算子流程如图 2 所示.

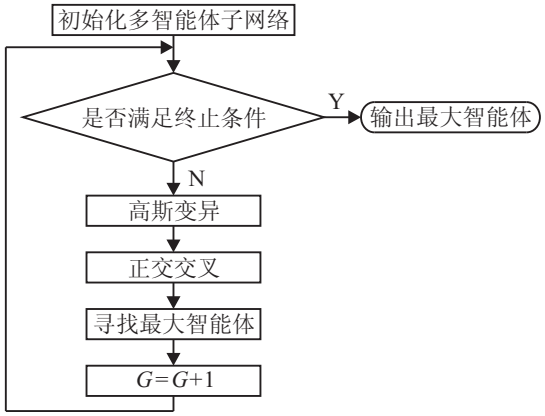

图 2　局部寻优算子流程

2.4　多智能体差分进化算法步骤与实现

在多智能体差分进化算法中, 邻域竞争算子起优胜劣汰的作用; 正交交叉算子通过交叉保留优势信息; 差分进化算子和变异算子则将产生和更新优势多智能体; 局部寻优算子进一步提高算法精度. 当主网格迭代一定代数后加入局部寻优算子操作, 可降低计算时间, 更有效地发挥局部寻优算子的局部搜索能力. 而当主网格运行到一定代数时, 它所找到的最优点是全局极值点的可能性增大, 这时再运用局部寻优算子可以达到事半功倍的效果. 为防止早敛现象, 由随机更新率 P_0 控制并进行智能体随机更新, 保持多智能体网格的多样性, 即依据 P_0, 利用高斯变异更新多智能体主网格. 多智能体差分进化算法的详细流程如下:

L^G 表示第 G 代的智能体网格, $L^{G+1/3}$ 和 $L^{G+2/3}$ 是 L^G 与 L^{G+1} 之间的中间代智能体网格.

Agent_max^G 是 $L^0, L^1, \cdots, L^G$ 中最优的智能体, Agent_max_c^G 是 L^G 中最优的智能体.

Step 1: 初始化 L^0, 更新 Agent_max^0, Energy_max^0, $G=0$.

Step 2: 对 L^G 中的每个智能体执行邻域竞争算子, 得到 $L^{G+1/3}$.

Step 3: 对于 $L^{G+1/3}$ 中的每个智能体, 将正交交叉算子作用其上, 得到 $L^{G+2/3}$.

Step 4: 依据 P_0 对 $L^{G+2/3}$ 中的每个智能体进行差分进化和更新操作, 得到 L^{G+1}.

Step 5: 从 L^{G+1} 中找出 Agent_max_c^G, 并将局部寻优算子作用其上.

Step 6: 如果

$$\text{Energy}(\text{Agent\_max}\_c^{G+1})>\text{Energy}(\text{Agent\_max}^G),$$

则有

$$\text{Agent\_max}^{G+1}=\text{Agent\_max}\_c^G,$$

$$\text{Energy\_max}^{G+1}=\text{Energy}(\text{Agent\_max}^{G+1});$$

否则

$$\text{Agent\_max}^{G+1}=\text{Agent\_max},$$

$$\text{Energy\_max}^{G+1}=\text{Energy}(\text{Agent\_max}^G).$$

Step 7: 如果满足终止条件, 则输出 Agent_max^G, 并停止; 否则, 令 $G=G+1$, 转 Step 2.

多智能体差分进化算法流程如图 3 所示.

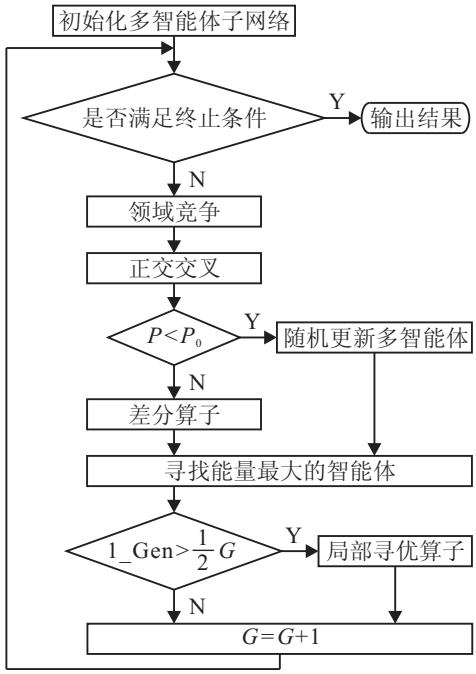

图 3　多智能体差分进化算法流程

3　测试函数及仿真结果与分析

3.1　测试函数及仿真结果

为了测试本文提出的 MADE 算法的性能, 本文针对 8 个标准测试函数进行了测试, 并将测试结果与以往文献算法[11-20]进行了比较. 测试函数中多为多峰函数, 特别是函数 F_{08} 拥有 $9.33\times10\,157$ 个局部极值点, 搜索过程容易陷入局部最优, 以检验算法的多峰搜索能力. 测试函数如下:

$$F_{01}:\min[f(x)]=\sum_{i=1}^{n}x_i^2,$$

$$x^*=(0,0,\cdots,0),\ f(x^*)=0.$$

$$F_{02}:\min[f(x)]=\sum_{i=1}^{n}(-x_i\sin(\sqrt{|x_i|})),$$

$$S=[-500,500]^n,\ f(x^*)=-12\,569.5.$$

$$F_{03}:\min[f(x)]=\sum_{i=1}^{n}[x_i^2-10*\cos(2\pi x_i)+10],$$

$$S=[-5.12,5.12]^n,\ x^*(0,\cdots,0),\ f(x^*)=0.$$

$$F_{04}:\min[f(x)]=\frac{1}{4\,000}\sum_{i=1}^{n}x_i^2-\prod_{i=1}^{n}\cos\left(\frac{x_i}{\sqrt{i}}\right)+1,$$

$$S=[-600,600]^n,\ x^*=(0,\cdots,0),\ f(x^*)=0.$$

$$F_{05}:\min[f(x)]=$$

$$\frac{\pi}{n}\{10\sin^2(\pi y_i)+(y_n-1)^2+$$

$$\sum_{i=1}^{n}(y_i-1)^2[1+10\sin^2(\pi y_{i+1})]\}+$$

$$\sum_{i=1}^{n}u(x_i,10,100,4),$$

$$S=[-50,50]^n,\ x^*=(1,\cdots,1),\ f(x^*)=0.$$

其中

$$u(x_i,a,k,m)=$$

$$\begin{cases}k(x_i-a)^m,\ x_i>a;\\ 0,\ -a\leqslant x_i\leqslant a,\ y_i=1+\dfrac{1}{4}(x_i+1);\\ k(-x_i-a)^m,\ x_i<-a.\end{cases}$$

$$F_{06}:\min[f(x)]=$$

$$\frac{1}{10}\Big\{\sin^2(3\pi x_i)+\sum_{i=1}^{n}(x_i-1)^2[1+\sin^2(3\pi x_{i+1})]+$$

$$(x_n-1)^2[1+\sin^2(2\pi x_n)]\Big\}+\sum_{i=1}^{n}u(x_i,5,100,4),$$

$$S=[-50,50]^n,\ x^*=(1,\cdots,1),\ f(x^*)=0.$$

$$F_{07}:\min[f(x)]=\frac{1}{n}\sum_{i=1}^{n}(x_i^4-16x_i^2+5x_i),$$

$$S=[5,5]^n,$$

$$x^*=(-2.903\,53,-2.903\,53,\cdots,-2.903\,53),$$

$$f(x^*)=-78.332\,36.$$

$$F_{08}:\min[f(x)]=-\sum_{i=1}^{n}\sin(x_i)\sin^{20}\left(\frac{i\times x_i^2}{\pi}\right),$$

$S=[0,\pi]^n$, 最优解未知.

F_{07} 和 F_{08} 为 100 维测试函数, 其余测试函数为 30 维. 本文 MADE 算法利用 MATLAB 7.1 实现 (定义小于 1.0×10^{-308} 为 0), 在奔腾 Ⅳ 电脑独立运行 50 次 (以往文献算法均在奔腾 Ⅲ 电脑独立运行 50 次). MADE 参数设置如下: $L_{\text{size}}=9$, CR $=0.4$, $P_0=0.3$, $P=4$, $P_c=0.3$, $F=2.5-2\times\text{rand}(1,0)$, $sL_{\text{size}}=3$, 局部寻优算子最大迭代代数为 10.

测试结果如表 1 所示. 其中 NA 表示文献未给出该信息. 由表 1 可知, MADE 算法无论是最优值还是标准差均优于其他算法, 从而验证了 MADE 算法的精确性和收敛性, 尤其对测试函数 F_{07} 和 F_{08} 的寻优性能较好. 另外, 在对 100 维测试函数 F_{08} 的优化计算中, MADE 算法得到了更优的优化结果 (−99.620 194, 见表 2), 验证了 MADE 算法对于解决多峰问题的优越性能.

表 1 各种算法结果比较

Function	Algorithms	Generations	M-best	Std	Opt-F
F_{01}	ALEP	1 500	6.32×10^{-4}	7.6×10^{-5}	0
	FEP	1 500	5.7×10^{-4}	1.3×10^{-4}	
	CEP/best	5 000	$3.09@10^{-7}$	NA	
	OGA/Q	1 000+	0	0	
	HTGA	NA	0	0	
	HPSO-TVAC	3 000	0.01	NA	
	M-L	NA	3.191 23	0.294 63	
	LEA	1 000+	4.727×10^{-16}	6.218×10^{-17}	
	MADE	1 000	2.67×10^{-18}	3.5×10^{-18}	
F_{02}	ALEP	1 500	−11 469.2	58.2	−12 569.5
	FEP	9 000	−12 554.5	52.6	
	OGA/Q	1 000+	−12 569.453 7	6.447×10^{-4}	
	HTGA	NA	−12 569.46	0	
	EDA/L	30+	−12 569.48	NA	
	M-L	NA	−5 461.826	275.15	
	LEA	1 000+	−12 569.454 2	$4\,831\times10^{-4}$	
	MADE	1 000	−12 569.486 618	0	

续表 1　各种算法结果比较

Function	Algorithms	Generations	M-best	Std	Opt-F
F_{03}	ALEP	1 500	5.85	2.07	0
	FEP	5 000	0.046	0.012	
	CEP/best	5 000	4.73	NA	
	OGA/Q	1 000+	0	0	
	HTGA	NA	0	0	
	HPSO-TVAC	5 000	0.044	0.196	
	CPSO-H6	10 000	0.778	NA	
	EDA/L	30+	0	NA	
	M-L	NA	121.757 5	7.757 3	
	LEA	1 000+	2.103×10^{-18}	3.359×10^{-18}	
	MADE	1 000	0	3.221×10^{-16}	
F_{04}	ALEP	1 500	0.024	0.028	0
	FEP	2 000	0.016	0.022	
	CEP/best	5 000	2.52×10^{-7}	NA	
	OGA/Q	1 000+	0	0	
	HTGA	NA	0	0	
	HPSO-TVAC	5000	0.01	0.001	
	CPSO-H6	10 000	0.052 4	NA	
	EDA/L	30+	0	NA	
	M-L	NA	0.118 94	0.010 404	
	LEA	1 000+	6.104×10^{-16}	2.513×10^{-17}	
	MADE	1 000	4.44×10^{-16}	1.02×10^{-16}	
F_{05}	ALEP	1 500	0.024	0.028	0
	FEP	2 000	0.016	0.022	
	OGA/Q	1 000+	0	0	
	HTGA	NA	0	0	
	EDA/L	30+	0	NA	
	M-L	NA	0.118 94	0.010 404	
	LEA	1 000+	6.104×10^{-16}	2.513×10^{-17}	
	MADE	1 000	$4.602\,2 \times 10^{-22}$	3.821×10^{-21}	
F_{06}	ALEP	1 500	9.8×10^{-5}	1.2×10^{-5}	0
	FEP	2 000	1.6×10^{-4}	7.3×10^{-5}	
	OGA/Q	1 000+	1.869×10^{-4}	2.615×10^{-5}	
	HTGA	NA	1.000×10^{-4}	0	
	EDA/L	30+	3.485×10^{-21}	NA	
	M-L	NA	1.505 34	2.255 64	
	LEA	1 000+	1.734×10^{-4}	1.205×10^{-4}	
	MADE	1 000	$6.164\,4 \times 10^{-19}$	4.1362×10^{-19}	
F_{07}	OGA/Q	1 000+	−78.300 029 6	1.293×10^{-3}	−78.332 36
	HTGA	NA	−78.303 0	0	
	EDA/L	30+	−78.310 77	NA	
	M-L	NA	−35.809 95	0.891 46	
	LEA	1 000+	−78.310	6.127×10^{-3}	
	MADE	1 000	−78.332 331	0	
F_{08}	OGA/Q	1 000+	−92.83	0.026 26	−99.278 4
	HTGA	NA	−92.83	0	
	EDA/L	30+	−94.375 7	NA	
	M-L	NA	−23.975 44	0.628 75	
	LEA	1 000+	−93.01	0.023 14	
	MADE	1 000	−96.018 802	0.261 8	

表 2　各种算法对 F_{08} 的求解结果

	算 法	最小值	维数	最优解	峰值个数
F_{08}	MADE	未知	100	−99.620 194 0	9.33×10^{159}
	文献 [11,12]			−99.278 4	
	文献[9]			−99.616 303 65	
	HSOGA[20]			−99.618 0	

3.2　计算复杂度分析

MADE算法的寻优操作主要由邻域竞争算子、正交交叉算子、差分进化算子和局部寻优算子构成. 在一个进化代内, 邻域竞争算子、正交交叉算子、差分进化算子和局部寻优算子目标函数计算数量分别为L_{size}^2, L_{size}^2, $\frac{1}{2}P \times P_c \times L_{\text{size}}^2$, $sL_{\text{size}}^2 + P \times P_c \times sL_{\text{size}}^2$, 则算法在一个进化代内目标函数计算数量为

$$F_{\text{MADE}} = L_{\text{size}}^2 + L_{\text{size}}^2 + \frac{1}{2}P \times P_c \times L_{\text{size}}^2 + (sL_{\text{size}}^2 + P \times P_c \times sL_{\text{size}}^2).$$

由 $\beta = sL_{\text{size}}/L_{\text{size}}$ 得

$$F_{\text{MADE}} = L_{\text{size}}^2 + L_{\text{size}}^2 + \frac{1}{2}P \times P_c \times L_{\text{size}}^2 + (sL_{\text{size}}^2 + P \times P_c \times sL_{\text{size}}^2) = \left(2 + \frac{1}{2}PP_c + \beta + \beta PP_c\right) \times L_{\text{size}}^2.$$

可见, MADE算法的计算复杂度与 β, P, L_{size} 相关, β, P, L_{size} 和 P_c 的数值越大, MADE算法的计算复杂度越高.

另外, 以只有邻域竞争算子和正交交叉算子的基本多智能体遗传算法为例, 基本多智能体遗传算法在一个进化代内目标函数计算数量为

$$F_{\text{MAGA}} = L_{\text{size}}^2 + 2L_{\text{size}}^2 = 3L_{\text{size}}^2.$$

可见, MADE算法的计算复杂度与基本多智能体遗传算法呈线性关系, 因为多智能体遗传算法呈现多项式的渐进计算复杂度[9], 所以MADE算法也呈现多项式的渐进计算复杂度.

4 结 论

针对复杂高维函数优化问题, 本文将多智能体算法与差分进化算法相结合, 充分利用多智能体模拟自然环境、类似小生境逐渐进行扩散的优点, 以及差分进化算法在快速性和鲁棒性方面的优势, 提出了一种多智能体差分进化算法. 该方法在提出高维正交表产生方法的基础上引入了正交交叉算子, 同时利用局部竞争算子等寻优策略改进了多智能体算法性能. 最后通过函数优化仿真测试, 验证了所提出方法的有效性.

参考文献(References)

[1] Brest J, Greiner S, Bokovi B, et al. Self-adapting control parameters in differential evolution: A comparative study on numerical benchmark problems[J]. IEEE Trans on Evolutionary Computation, 2006, 10(6): 646-657.

[2] Kaelo P, Ali M M. Differential evolution algorithms using hybrid mutation[J]. Computational Optimization and Applications, 2007, 37(2): 231-246.

[3] Brest J, Boskovic B, Greiner S, et al. Performance comparison of self-adaptive and adaptive differential evolution algorithms[J]. Soft Computing, 2007, 11(7): 617-629.

[4] Noman N, Iba H. Accelerating differential evolution using an adaptive local search[J]. IEEE Trans on Evolutionary Computation, 2008, 12(1): 107-125.

[5] Qin A K, Huang V L, Suganthan P N. Differential evolution algorithm with strategy adaptation for global numerical optimization[J]. IEEE Trans on Evolutionary Computation, 2009, 13(2): 398-417.

[6] 潘晓英, 刘芳, 焦李成. 基于智能体的多目标社会进化算法[J]. 软件学报, 2009, 20(7): 1703-1713.
(Pan X Y, Liu F, Jiao L C. Multiobjective social evolutionary algorithm based on multi-agent[J]. J of Software, 2009, 20(7): 1703-1713.)

[7] 闫杨, 汪定伟, 王大志, 等. 求解动态背包问题的多智能体进化算法[J]. 东北大学学报: 自然科学版, 2009, 30(7): 948-951.
(Yan Y, Wang D W, Wang D Z, et al. Multiagent-based evolutionary algorithm for dynamic knapsack problem[J]. J of Northeastern University: Natural Science, 2009, 30(7): 948-951.)

[8] 黄永青, 陆青, 梁昌勇, 等. 交互式多智能体进化算法及其应用[J]. 系统仿真学报, 2006, 18(7): 2030-3032.
(Huang Y Q, Lu Q, Liang C Y, et al. Interactive multi-agent evolutionary algorithm and its application[J]. J of System Simulation, 2006, 18(7): 2030-3032.)

[9] 钟伟才, 薛明志, 刘静, 等. 多智能体遗传算法用于超高维函数优化[J]. 自然科学进展, 2003, 13(10): 1078-1083.
(Zhong W C, Xue M Z, Liu J, et al. Multi-agent genetic algorithm for high-dimensional optimization[J]. Progress in Natural Science, 2003, 13(10): 1078-1083.)

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

[11] Yiu-Wing Leung, Wang Yuping. An orthogonal genetic algorithm with quantization for global numerical optimization[J]. IEEE Trans on Evolutionary Computation, 2001, 5(1): 41-53.

[12] Wang Yuping, Dang Chuangyin. An evolutionary algorithm for global optimization based on level-set evolution and latin squares[J]. IEEE Trans on Evolutionary Computation, 2007, 11(5): 579-595.

[13] ChangYong Lee, Yao Xin. Evolutionary programming using mutations based on the levy probability distribution[J]. IEEE Trans on Evolutionary Computation, 2004, 8(1): 1-13.

[14] Asanga Ratnaweera, Saman K Halgamuge, Harry C Watson. Self-organizing hierarchical particle swarm optimizer with time-varying acceleration coefficients[J]. IEEE Trans on Evolutionary Computation, 2004, 8(3): 240-255.

[15] Frans van den Bergh, Andries P Engelbrecht. A cooperative approach to particle swarm optimization[J]. IEEE Trans on Evolutionary Computation, 2004, 8(3): 225-239.

[16] Yao Xin, Liu Yong Liu, Lin Guangming. Evolutionary programming made faster[J]. IEEE Trans on Evolutionary Computation, 1999, 3(2): 82-102.

[17] Jinn-Tsong Tsai, Tung-Kuan Liu, Jyh-Horng Chou. Hybrid Taguchi-genetic algorithm for global numerical optimization[J]. IEEE Trans on Evolutionary Computation, 2004, 8(4): 365-377.

(下转第972页)