DOI: 10.7641/CTA.2013.11238

链式多种群多智能体进化算法

吴亚丽†, 靳笑一, 刘 格

(西安理工大学 自动化与信息工程学院, 陕西 西安 710048)

摘要: 将多种群的进化方式和链式结构的动态邻域引入到多智能体进化算法中, 提出了一种链式多种群多智能体进化算法. 算法设置了多种群交互的演化结构. 各种群中的智能体通过与其动态邻域智能体的竞争、合作及自学习操作来增加自身的能量; 动态邻域的链式结构提高了算法的效率、降低了计算复杂度; 多个种群之间的信息定期以一定的方式进行交互, 增强了种群的多样性, 减小了算法陷入局部最优的机率. 理论分析和多个测试函数的仿真结果均表明: 链式多种群多智能体进化算法在求解高维优化问题上具有很好的性能.

关键词: 多种群; 链式结构; 多智能体进化算法

中图分类号: TP273 **文献标识码**: A

Chainlike multi-population multi-agent evolutionary algorithm

WU Ya-li†, JIN Xiao-yi, LIU Ge

(Automation and Information Engineering School, Xi'an University of Technology, Xi'an Shaanxi 710048, China)

Abstract: We propose a novel chainlike multi-population multi-agent evolutionary algorithm which combines the dynamic neighborhood environment chainlike structure with the evolutionary framework of multi-population. This algorithm provides the evolution structure for multi-populations interaction. Agents in the population increase their own energy by competition, cooperation and self-study with its dynamic neighborhood agents. The chainlike structure improves the efficiency of algorithms and reduces the computational complexity. The interaction of information among various populations in a regular period of time improves the diversity of the population and decreases the possibility of sticking at local optima. Theoretical analysis and simulation of multiple test functions show that the new algorithm is very good for handling high-dimension optimization problems.

Key words: multi-population; chainlike structure; multi-agent evolutionary algorithm

1 引言(Introduction)

进化算法由于具有广泛的应用性、高度的非线性、易修改性和可并行性等特点, 现已被广泛应用于计算科学、人工智能等领域. 遗传算法作为进化算法的代表, 不但具有很强的鲁棒性和适应性, 而且对优化目标本身既不要求连续也不要求可微, 因此在函数优化、参数辨识、机器人控制等各个领域得到了广泛的应用. 但遗传算法本身还存在着早熟收敛和全局收敛速度慢等问题.

近年来, 基于分布式结构的智能体(agent)计算已应用于计算机学科的各个领域. 多智能体系统的分布式并行结构有利于避免进化算法进入早熟, 而智能体本身的邻域结构有利于提高种群的局部探索能力. 钟伟才等将智能体结构与遗传算法相结合[1–3], 提出了多智能体进化算法, 提高了遗传算法的求解精度, 并分别用于无约束优化和约束优化问题的求解, 得到了更高质量的解; 但由于多智能体结构中邻域数目较大, 在提高算法精度的同时增加了计算时间, 为此文献[4]在遗传算法中加入动态链式智能体网络结构, 减少了次优个体过早取得“顶端优势”而导致过早收敛的发生, 可以更好的保持种群多样性; 文献[5]将链式多智能体进化算法用于求解多产品批调度问题, 取得了很好的效果; 为保持种群的多样性, 文献[6]将均匀设计方法引入智能体遗传算法中, 在一定程度上避免了算法陷入局部最优; 文献[7]将差分进化算子引入到多智能体进化算法中, 提高智能体的更新速度和保持种群多样性; 为提高进化算法的收敛性能, 文献[8]将交互式引入多智能进化算法, 提出交互式多智能体进化算法, 以提高算法的全局收敛能力和局部搜索能力; 文献[9]将多种群的思想引入到遗传算法中, 研究了种群数目对收敛速度的影响, 提高了遗传算法的收敛速度和寻优能力. 上述各种改进算法均在一定的程度上提高了算法的寻优能力和收敛速度.

收稿日期: 2011−11−02; 收修改稿日期: 2012−10−05.

†通信作者. Tel.: +86 13119121204.

基金项目: 陕西省自然科学基金资助项目(2010JQ8006); 陕西省教育厅科学研究计划资助项目(2010JK711); 国家基金资助项目(61172123).

为提高多智能体进化算法的全局收敛能力, 本文将多种群交互演化的思想引入到多智能体进化算法之中, 同时为了实现算法局部探索能力和计算时间的均衡, 针对多智能体的动态邻域改为链式结构, 提出了一种链式多种群多智能体进化算法(chainlike multi-population multi-agent evolutionary algorithm, CMPMA). 算法采用多种群交互的演化结构, 各群体分别采用链式结构的多智能体进化规则, 通过与其动态邻域智能体的竞争、合作及自学习来实现自身能量的增加, 向着群体最优的方向移动; 各种群定期通过交互操作进行信息的同步交互. 理论分析结果表明了算法的收敛性和快速性. 对多个测试函数的仿真结果表明, 该算法对于求解多维函数优化问题具有很好的性能.

2 CMPMA的原理(Principle of CMPMA)

CMPMA的演化结构如图1所示. 图1中多个种群分别独立进化, 定期通过一定的方式进行信息交互.

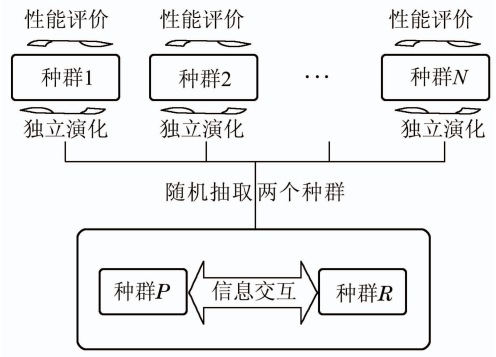

图 1 CMPMA的演化结构图

Fig. 1 The evolution structure of CMPMA

2.1 各种群的演化规则(Evolution rules of each population)

CMPMA由N个种群交互演化构成. 各种群采用链式结构的多智能体进化算法独立演化.

与智能体(agent)相关定义:

定义 1 agent结构及能量. 设agent是问题搜索空间S的一个候选解a, 其能量定义为

$$a \in S,\ E(a) = 1/f(a), \tag{1}$$

其中适应度函数$f(a)$表示目标函数的适应值. 可以看出, 能量越大, 适应值越小, 目标值越接近最优值.

每个agent数据信息结构表示为

$$a = (\mathrm{body}, \mathrm{place}, \mathrm{local}),$$

其中: body为一候选解以及其被优化问题所决定的适应值; place为智能体a在环境中的位置; local为智能体a的邻域, 表示为$\mathrm{local}(a) = (z_1, z_2, \cdots, z_m)$, 其中$z_i = (\mathrm{place}(z_i), \mathrm{trust}(z_i))$表示智能体$a$邻域的第$i$个智能体, 其中$\mathrm{place}(z_i)$和$\mathrm{trust}(z_i)$分别表示agent z_i所处的位置以及信任度.

定义 2 agent的链式环境. agent链式环境定义如图2所示, 其中圆圈代表agent, 圆圈中的数据表示agent在环境中的位置. 每个agent都生存在链式环境L中, 称L为智能体链表, 其规模为L_{size}. 由于智能体只有局部感知能力, 故它只能与邻域内的两个智能体相互作用.

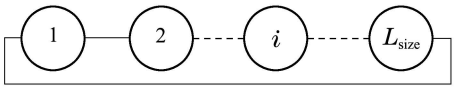

图 2 智能体链式环境

Fig. 2 Chainlike environment of agent

定义 3 agent邻域. agent邻域定义是基于其环境而言的, 针对本文算法采用链式结构环境, 那么agent邻域做如下定义: 假设L_i是链表上坐标为i的agent, $i = 1, 2, \cdots, L_{\mathrm{size}}$, L_i的初始邻域$\mathrm{local}(L_i) = [L_{i_1}, L_{i_2}]$, 其中:

$$i_1 = \begin{cases} i-1, & i \neq 1, \\ L_{\mathrm{size}}, & i = 1, \end{cases} \tag{2}$$

$$i_2 = \begin{cases} i+1, & i \neq L_{\mathrm{size}}, \\ 1, & i = L_{\mathrm{size}}. \end{cases} \tag{3}$$

CMPMA中各种群中的智能体主要通过与其邻域中智能体的竞争、交叉、变异和自学习4种操作完成能量的增加.

1) 竞争. 假设智能体$H_i = (h_1, h_2, \cdots, h_n)$ (n为变量的维数)是链表位置为i的智能体的邻域内能量最大的智能体. 若L_i满足$\mathrm{energy}(L_i) > \mathrm{energy}(H_i)$, 则继续存活在链表中; 否则产生新智能体new取代原有智能体.

2) 交叉. 交叉采用正交交叉的规则, 正交交叉操作参考文献[10]所述, 利用正交矩阵在搜索空间产生均匀分布的个体, 将正交交叉算子作用在$L_{i,j}$和$L\max_{i,j}$上产生新的智能体替代$L_{i,j}$.

3) 变异. 设智能体$c = (c_1, c_2, \cdots, c_n)$($n$为变量的维数)的每一维以概率$P_{\mathrm{m}}$进行变异, 产生新的智能体$c'$.

4) 自学习. 自学习操作的实质为一局部链式智能体进化算法, 具体的操作过程见算法4.

2.2 种群间的信息交互体系(Information interaction system of populations)

为了增加种群的多样性, 提高种群中优秀个体信息的利用效率, 本算法对多个种群进行以下的交互操作. 在演化进行一定代数的时候, 随机抽取整个演化空间的两个种群, 种群$P(P = 1, 2, \cdots, N)$和种群$R(R = 1, 2, \cdots, N; R \neq P)$进行交互操作, 种

群P就通过交互操作1接受种群R中较优的个体经验知识进行种群更新, 种群R就通过交互操作2接受种群P的种群经验指导其种群进化. 下面是两个交互行为的具体操作:

1) 交互操作1. 在种群P的种群进化过程中, 每运行k_1代, 用种群R的当前最好的适应值个体来替换种群P中适应值较差的个体. 其中

$$k_1 = A_{\mathrm{mun}} + k/\mathrm{Gen} \times B_{\mathrm{mum}}, \tag{4}$$

k是种群当前运行的代数; Gen为算法预先设定的最大演化代数; A_{mun}和B_{mun}用于调节交互操作1和交互操作2的完成次数, 根据问题的特点选择, 它们的大小随着函数峰值的增多而递减.

2) 交互操作2. 在种群R的种群进化过程中, 每运行k_2代时, 用种群P的历代种群所产生的最优个体来替换群体空间中适应值较差的个体. 其中

$$k_2 = A_{\mathrm{mun}} + (\mathrm{Gen} - k)/\mathrm{Gen} \times B_{\mathrm{mum}}. \tag{5}$$

从算法的结构可以看出, 算法利用种群P与种群R的信息交互, 一方面互相吸收个体的有用信息指导个体进化的搜索的方向, 降低算法陷入局部最优的概率, 进而改善算法的全局收敛性能; 一方面增加了种群的多样性, 防止了算法的局部收敛. 多智能体进化算法本身强大的搜索能力加快了算法的收敛速度; 而基于链式结构的动态邻域, 在一定程度上降低了算法的搜索能力, 但降低了算法陷入局部极值的概率, 同时计算量明显减少.

3　实现CMPMA(Implementation of CMPMA)

3.1　CMPMA中各操作的具体算法(Specific algorithm of each operation for CMPMA)

CMPMA算法中涉及4类操作: 竞争、交叉、变异和自学习. 各操作的具体实施算法如下:

算法 1　竞争操作.

Step 1　初始化占据概率P_0和new $= (e_1, e_2, \cdots, e_n)$, 令$k = 0$;

Step 2　若$U(0,1) < P_0$, 设$\bar{X}_k$为第k维的变量的上限, $\underline{X}_k$为第k维的变量的下限, $U(-1,1)$表示产生一个$(-1,1)$之间均匀分布的实数, $P_{\mathrm{split}} = \mathrm{rand}/2$表示拆分概率, 则

$$e_k = \begin{cases} \underline{X}_k, h_k \times U(-1,1) \times (h_k - l_k) < \underline{X}_k, \\ \bar{X}_k, X_k \times U(-1,1) \times (X_k - l_k) > \bar{X}_k, \\ h_k \times U(-1,1) \times (h_k - l_k), \text{ 其他}, \end{cases}$$

$k = 1, 2, \cdots, n$; 否则$e_k = e_k^1 + e_k^2$, 其中$e_k^1 = P_{\mathrm{split}} \times h_k$, $e_k^2 = 1 - P_{\mathrm{split}} \times h_k$, 令$k = k + 1$;

Step 3　若$k < n$, 则转步骤2, 否则转Step 4;

Step 4　更新智能体链表及能量, 输出new.

算法 2　交叉操作.

该算法的交叉采用正交交叉的方法, 具体操作见参考文献[10].

算法 3　变异操作.

设$G(0, 1/t)$为高斯分布的随机数产生器, t为当前演化的代数.

Step 1　初始化变异概率P_{m}和c', 令$k = 0$;

Step 2　若$U(0,1) < P_{\mathrm{m}}$, $c'(k) = c(k)$, 否则$c'(k) = c(k) + G(0, 1/t)$, $k = k + 1$;

Step 3　若$k < n$, 则转步骤2, 否则转Step 4;

Step 4　更新智能体链表及能量, 输出c'.

算法 4　自学习操作.

假设自学习作用在智能体$L_i = (l_1, l_2, \cdots, l_n)$上, 产生的新智能体为news, 设$s$Gen表示最大代数, sR表示相对搜索半径, $sL_{\mathrm{size}} < L_{\mathrm{size}}$, sP_{m}为预先设定的参数, energy$(\cdot)$表示智能体的能量, sL_k表示第k代的智能体链表, $s\mathrm{local}_k$为第k代智能体链表的局部生存环境, sL'_k为介于sL_k和sL_{k+1}之间的临时链表. $s\mathrm{opt}_k$为$sL_0, sL_1, \cdots, sL_k$中最好的智能体, $s\mathrm{best}_k$为链表sL_k中最好的智能体.

Step 1　根据

$$\mathrm{s}L_{i'} = \begin{cases} L_j, & i' = 1, \\ \mathrm{new}_{i'}, & \text{其他}. \end{cases}$$

产生智能体链表$sL_i^0 (i = 1, 2, \cdots, sL_{\mathrm{size}})$; 初始化局部环境$s\mathrm{local}_0$, 并更新$s\mathrm{opt}_0$, 令$k = 0$, 其中$\mathrm{new}_{i'} = (Ne_1, Ne_2, \cdots, Ne_n)$由式

$$Ne_k = \begin{cases} \underline{X}, l_k \times U(1 - sR, 1 + sR) < \underline{X}, \\ \bar{X}, l_k \times U(1 - sR, 1 + sR) > \bar{X}, \\ l_k \times U\left(1 - sR, 1 + sR\right), \text{ 其他} \end{cases}$$

确定, 其中$k = 1, 2, \cdots, n$;

Step 2　对sL_k上每个智能体执行领域竞争操作, 得到sL'_k;

Step 3　对sL'_k上的每个智能体, 如果$U(0,1) < sP_{\mathrm{m}}$, 则执行变异操作, 得到sL_{k+1};

Step 4　在sL_{k+1}中找到该空间能量最高的智能体$s\mathrm{best}_{k+1}$; 若energy$(s\mathrm{best}_{k+1}) >$ energy$(s\mathrm{best}_k)$, 则令$s\mathrm{best}_{k+1} \to s\mathrm{opt}_{k+1}$; 否则令$s\mathrm{opt}_k \to s\mathrm{opt}_{k+1}$, $s\mathrm{opt}_k \to s\mathrm{best}_{k+1}$;

Step 5　若$k =$ Gen, 则停止并输出$s\mathrm{opt}_k \to$ news 否则令$k = k + 1$, 转Step 2.

3.2　CMPMA实现步骤(Implementing steps of CMPMA)

CMPMA采用十进制的编码方式, 假设L_k表示第k代的智能体链表, local_k为第k代智能体链表的局部生存环境, L'_k, L''_k分别为介于L_k和L_{k+1}之间的临时链表, opt_k为$L_0, L_1, \cdots, L_k$中最好的智能体,

best_k为链表L_k中最好的智能体, $\text{energy}(\cdot)$表示智能体的能量. CMPMA算法的具体实现步骤如下:

Step 1 种群初始化: 多种群个数N, 链表大小L_{size}, 最大迭代次数Gen, 各空间的智能体链表L_0, 局部生存环境local_0, 更新opt_0令$k=0$;

Step 2 根据式(4)判断k/k_1为整数时, 随机选取两个种群进行交互操作1;

Step 3 N个种群按如下规则独立演化:

1) 对种群中的个体进行竞争操作, 得到L'_k, 更新智能体的局部生存环境local_k;

2) 对种群个体进行交叉操作, 得到L''_k, 更新智能体的局部生存环境local_k;

3) 对种群个体进行变异操作, 得到L_{k+1}, 更新智能体的局部生存环境local_k;

4) 在L_{k+1}中找到该空间能量最高的智能体best_{k+1}, 对其执行自学习操作;

5) 如果$\text{energy}(\text{best}_{k+1}) > \text{energy}(\text{best}_k)$, 则令$\text{best}_{k+1}\rightarrow\text{opt}_{k+1}$; 否则$\text{opt}_k\rightarrow\text{opt}_{+1}$, $\text{opt}_k\rightarrow\text{best}_{k+1}$;

Step 4 根据式(5)判断k/k_2为整数时, 进行交互操作2;

Step 5 判断终止条件, 若满足则停止并输出opt_k, 否则令$k=k+1$, 转Step 2.

4 算法的收敛性及复杂度的分析(Convergence and complexity analysis of the algorithm)

4.1 算法的收敛性分析(Convergence analysis of the algorithm)

链式多种群多智能体进化算法的收敛性取决于每一个种群的独立进化规则的收敛性, 即链式多智能体进化算法的收敛性.

引理 1 多智能体进化算法的种群进化以概率1收敛, 具有全局收敛性.

由于链式多智能体进化算法与普通的多智能体进化算法只是在邻域的个数上有区别, 因此链式多智能体进化算法也具有全局收敛性.

由于算法的多种群演化结构, 因此设N个种群的第i个种群的解及对应的能量为$X_i=(x_{i1},x_{i2},\cdots,x_{iL_{\text{size}}})$, $\text{energy}_i=(E_{i1},E_{i2},\cdots,E_{iL_{\text{size}}})$; 其中: $i=1,2,\cdots,N$; L_{size}为每个种群的规模. 假设第i个种群的最优解及其对应的能量为x_{ij}, $E_{ij}(j=1,2,\cdots,L_{\text{size}})$, 由引理1知, 每个种群的演化都是收敛的, 即下式成立:

$$\lim_{k\rightarrow\infty}\Pr\{\text{energy}_i=E_{ij}\}=1, \tag{6}$$

其中k为种群的演化代数.

设随机抽取的种群P的最优解及对应的能量为X_{Ps}, energy_{Ps}; 种群R的解以及对应的能量为X_{Rt}, $\text{energy}_{Rt}(P,R=1,2,\cdots,N$, 且$P\neq R; s,t=1,2,\cdots,L_{\text{size}})$.

由种群交互机制知, 交互只是种群中优秀个体的替代, 即最优解一定会被交互到另外的种群中. 交互后, 若$E_{Rt}>E_{Ps}$, 对于种群P中最优解将变成X_{Rt}, 此时对于种群P式(6)依然成立; 对于种群R的最优解保持不变, 式(6)依然成立. 若$E_{\text{Rs}}<E_{\text{Pt}}$, 同上, 收敛式(6)依然成立. 因此多种群进行交互后, 不改变各个种群的收敛性.

由上分析可得, 链式多种群多智能体进化算法具有全局收敛性.

4.2 算法的复杂度分析(Complexity analysis of the algorithm)

算法的复杂性分析主要从时间复杂度和空间复杂度来考虑.

1) 算法的时间复杂度, 即度量算法的执行时间. 可以从时间频度衡量, 即语句执行次数成正比例.

传统的智能体邻域环境中每一个智能体的邻域为8个, 而链式结构的智能体邻域个数为2个. 假设传统网格结构与链式所含智能体个数相同, 即$L_{\text{size}}=l^2_{\text{size}}, sL_{\text{size}}=sl^2_{\text{size}}$. 从邻域个数上而言链式结构的邻域个数明显少了很多. 而动态邻域结构的改变, 影响了算法的邻域竞争和自学习的比较次数:

MPCMA的比较次数: $\text{comparison}=\text{Gen}\times N\times 2\times(L_{\text{size}}+s\text{Gen}\times sL_{\text{size}})$.

网络结构的多种群多智能体进化算法的比较次数(multi-population multi-agent evolutionary algorithm, MPMA): $\text{comparison}=\text{Gen}\times N\times 8\times(l^2_{\text{size}}+s\text{Gen}\times sl^2_{\text{size}})$.

因此链式多种群多智能体进化算法的时间复杂度有了明显的改进.

2) 算法的空间复杂度, 即算法的存储空间. 从网络结构跟链式结构的特点看: 一个链表的存储空间明显小于一个多维矩阵的存储空间. 同时在寻址时, 一个链表的寻址明显比一个多维矩阵的寻址简单.

综上所述, 链式多种群多智能体算法的复杂度明显降低.

5 实例仿真与结果分析(Simulation and analysis)

为验证CMPMA的性能, 本文选用文献[11]中的6个测试函数对CMPMA的性能进行仿真分析. 本文从以下4个方面对算法进行性能分析:

1) MPMA及CMPMA两种算法的对比分析. 首先分别采用MPMA与本文算法分别对上述6个测试函数进行仿真, 算法参数设置为: $Q=3, F=4, s\text{Gen}=10, P_0=0.5, sP_{\text{m}}=0.08, P_{\text{c}}=0.48,$

$P_{\rm m}=0.08, L_{\rm size}=64, sL_{\rm size}=9, l_{\rm size}=8, sl_{\rm size}=3, N=2$. 表1统计了各算法均执行30次的平均最好值、最差值、平均值、方差、均方差、成功运行次数(Succ)以及CPU时间等性能指标.

表 1 两种算法在6种测试函数上的仿真结果

Table 1 The simulation results of two algorithms with six test functions

函数	算法	最好值	最差值	平均值	方差	均方差	Succ	CPU / s
Camel	$\mathrm{Gen}=50; X_{\max}=10; \varepsilon=0.00001; sR=0.8$							
	MPMA	−1.03163	−1.03163	−1.03163	−3.7E−16	0	30	0.5593
	CMPMA	−1.03163	−1.03163	−1.03163	−2.5E−16	0	30	0.5255
Shubert	$\mathrm{Gen}=200; X_{\max}=10; \varepsilon=0.0001; sR=0.8$							
	MPMA	−186.731	−79.4109	−178.943	609.485	24.6878	30	0.2339
	CMPMA	−186.731	−123.577	−123.577	132.948	11.5303	30	0.2244
Sphere	$\mathrm{dimension}=10; \mathrm{Gen}=150; X_{\max}=10; \varepsilon=0.00001; sR=0.8$							
	MPMA	1.02E−71	6.57E−61	1.1554	1.4E−122	1.2E−61	30	5.1359
	CMPMA	9.72E−73	2.74E−64	2.54E−62	1.3809	1.1751	30	4.9397
	$\mathrm{dimension}=20; \mathrm{Gen}=250; X_{\max}=10; \varepsilon=0.0001; sR=0.8$							
	MPMA	4.71E−69	106.5210	9.22804	−11.1864	0	30	15.7560
	CMPMA	6.68E−73	81.57964	3.83147	−15.1864	0	30	10.5598
	$\mathrm{dimension}=30; \mathrm{Gen}=350; X_{\max}=10; \varepsilon=0.001; sR=0.8$							
	MPMA	2.5E−84	4.6E−72	2.04E−73	8.8E−142	2.9E−71	30	18.6875
	CMPMA	7.2E−86	1.02E−72	1.67E−76	7.1E−145	8.3E−73	30	18.4538
Ackley	$\mathrm{dimension}=10; \mathrm{Gen}=150; X_{\max}=10; \varepsilon=0.00001; sR=0.8$							
	MPMA	2.66E−15	2.66E−15	0.97641	−0.9863	0	30	6.0672
	CMPMA	8.9E−16	2.66E−15	2.55E−15	4.2E−31	6.5E−16	30	5.9767
	$\mathrm{dimension}=20; \mathrm{Gen}=250; X_{\max}=10; \varepsilon=0.0001; sR=0.8$							
	MPMA	2.66E−15	13.81915	2.109879	−4.6051	0	30	13.1931
	CMPMA	2.66E−15	2.66E−15	2.66E−15	0	0	30	13.0275
	$\mathrm{dimension}=30; \mathrm{Gen}=350; X_{\max}=10; \varepsilon=0.001; sR=0.8$							
	MPMA	2.66E−15	6.22E−15	1.8987	−3.7296	0	30	22.6522
	CMPMA	2.66E−15	2.66E−15	2.66E−15	0	0	30	22.3990
Griewank	$\mathrm{dimension}=10; \mathrm{Gen}=300; X_{\max}=600; \varepsilon=0.05; sR=0.8$							
	MPMA	0	0.236124	0.07048	0.00408	0.06385	22	3.7748
	CMPMA	0	0.152988	0.064401	0.00173	0.04164	25	3.6986
	$\mathrm{dimension}=20; \mathrm{Gen}=250; X_{\max}=600; \varepsilon=0.01; sR=0.8$							
	MPMA	0	0.156212	0.016026	0.00121	0.03484	21	4.0741
	CMPMA	0	0.049405	0.009383	0.00024	0.01536	27	4.0088
	$\mathrm{dimension}=30; \mathrm{Gen}=250; X_{\max}=600; \varepsilon=0.01; sR=0.8$							
	MPMA	0	0.192702	0.01257	0.00132	0.03639	19	3.9891
	CMPMA	0	0.041382	0.009709	0.00020	0.01428	22	2.9769
Rastrigrin	$\mathrm{dimension}=10; \mathrm{Gen}=500; X_{\max}=5.12; \varepsilon=0.05; sR=1$							
	MPMA	0	10.76541	0.607754	5.30782	2.30387	27	5.56505
	CMPMA	0	7.202518	0.591015	3.52181	1.87665	28	5.41027
	$\mathrm{dimension}=20; \mathrm{Gen}=500; X_{\max}=5.12; \varepsilon=0.1; sR=1$							
	MPMA	0	40.11609	13.55657	159.921	12.646	11	7.2372
	CMPMA	0	24.87817	6.20122	67.3089	8.20420	17	7.0529
	$\mathrm{dimension}=30; \mathrm{Gen}=600; X_{\max}=5.12; \varepsilon=0.1; sR=1$							
	MPMA	0	73.17613	26.79501	687.908	26.228	19	10.7647
	CMPMA	0	72.6667	23.46832	367.183	19.1620	22	10.3680

从表1可见, 无论对二维函数还是多维函数, 在相同的参数设置下, 与MPMA算法相比, CMPMA平均值、方差和均方差、平均CPU时间小于MPMA; 成功运行次数也高; 尤其对于多维函数, 随着维数的增加, CMPMA算法的时间性能越优, CMPMA算法寻优效率和寻优能力均优于MPMA算法.

2) MPMA及CMPMA在求解高维函数优化问题的对比分析. 为测试CMPMA算法求解高维函数的性能, 对上述4个多维多模态函数优化问题, 在参数设置不变时, 分别将维数设置为100, 200和300时, 表2统计了30次独立实验后各性能指标的平均值.

表 2 两种算法在求解4类高维函数上的对比结果

Table 2 The comparison results of two algorithms in high dimension function

函数	算法	最好值	最差值	平均值	方差	均方差	Succ	CPU / s
	$\text{dimension}=100;\text{Gen}=500;X_{\max}=10;\varepsilon=0.1;sR=0.8$							
	MPMA	7.48E−54	2.66E−43	9.05E−45	2.36E−87	4.8E−44	30	62.8987
	CMPMA	2.55E−54	8.34E−46	3.66E−47	2.35E−92	1.5E−46	30	62.3793
	$\text{dimension}=200;\text{Gen}=500;X_{\max}=10;\varepsilon=0.1;sR=0.8$							
Sphere	MPMA	1.85E−55	4.65E−42	1.6E−43	7.19E−85	8.4E−43	30	64.3258
	CMPMA	9.55E−56	3.19E−46	1.31E−47	3.39E−93	5.8E−47	30	63.9511
	$\text{dimension}=300;\text{Gen}=500;X_{\max}=10;\varepsilon=0.1;sR=0.8$							
	MPMA	2.8E−06	0.198363	0.021702	0.001619	0.0402	29	351.4634
	CMPMA	7.46E−27	7.63E−05	8.63E−06	4.64E−10	2.2E−05	30	168.1365
	$\text{dimension}=100;\text{Gen}=500;X_{\max}=10;\varepsilon=0.1;sR=0.8$							
	MPMA	6.22E−15	5.366134	0.315967	1.47297	1.21366	28	76.5439
	CMPMA	6.22E−15	4.920451	0.274541	1.13600	1.06584	28	75.5762
	$\text{dimension}=200;\text{Gen}=500;X_{\max}=10;\varepsilon=0.1;sR=0.8$							
Ackley	MPMA	2.04E−14	7.168016	4.180882	7.28197	2.69851	19	145.662
	CMPMA	1.33E−14	6.528281	4.209059	4.06226	2.01551	22	145.446
	$\text{dimension}=300;\text{Gen}=500;X_{\max}=10;\varepsilon=0.1;sR=0.8$							
	MPMA	0.001794	7.4450	5.9723	0.34293	0.58560	16	329.895
	CMPMA	4.39E−12	7.1464	4.9370	0.33365	0.57762	21	214.485
	$\text{dimension}=100;\text{Gen}=300;X_{\max}=600;\varepsilon=0.01;sR=0.8;$							
	MPMA	4.77E−15	0.21088	0.023893	0.00319	0.05644	25	7.2030
	CMPMA	0	0.01724	0.000350	4.01E−04	0.02001	27	6.1310
	$\text{dimension}=200;\text{Gen}=300;X_{\max}=600;\varepsilon=0.01;sR=0.8$							
Griewank	MPMA	4.49E−12	0.527848	0.031803	0.01252	0.11188	27	26.3999
	CMPMA	2.46E−14	0.114724	0.003825	0.00044	0.02095	29	22.5935
	$\text{dimension}=300;\text{Gen}=300;X_{\max}=600;\varepsilon=0.01;sR=0.8$							
	MPMA	1.11E−16	0.416435	0.017284	0.00603	0.07766	27	129.506
	CMPMA	1.11E−16	0.17742	0.009145	0.00121	0.03478	28	118.815
	$\text{dimension}=100;\text{Gen}=1000;X_{\max}=5.12;\varepsilon=0.1;sR=1$							
	MPMA	0	297.8157	65.25396	8945.27	94.5795	15	39.7760
	CMPMA	0	271.1449	65.25396	8425.6	91.7911	20	39.6060
	$\text{dimension}=200;\text{Gen}=1000;X_{\max}=5.12;\varepsilon=0.1;sR=1$							
Rastrigrin	MPMA	0	934.1465	98.0931	69425.8	263.488	18	71.5440
	CMPMA	0	793.7921	84.94364	48507.1	220.243	22	70.3139
	$\text{dimension}=300;\text{Gen}=1000;X_{\max}=5.12;\varepsilon=0.1;sR=1$							
	MPMA	0	1572.137	75.21578	81843.5	286.083	19	185.024
	CMPMA	0	568.9059	47.62901	20237.4	142.258	21	181.648

从表2可以看出, 对同一个函数, 在相同的参数设置下, 随着问题规模的增大, 运算时间随着问题的规模急剧增大, 维数越高, CMPMA的时间优化性能越明显; 无论维数高低, CMPMA算法的平均值、最优值、最差值和方差及成功收敛次数均优于MPMA算法. CMPMA无论从寻优能力和寻优效率上均优于MPMA.

3) 不同种群个数对算法性能的影响分析. 在以上两部分的分析中, 种群数$N = 2$. 当种群数成倍增加时, 相当于算法中每次迭代的个体成倍增加, 导致算法计算复杂度增加, 算法的其他性能也有了相应的改变. 附表1和附表2给出了其他参数不变时, 种群数分别为4和6时两种算法在求解各测试函数的性能指标. 附表3统计了种群个数分别为4和6时各高维函数的性能指标.

将附表1和附表2与表1中的数据进行对比分析, 可以看出, 对于同一问题, 采用同一种算法, 当种群个数增大时, 算法的运算时间明显增加, 对简单的Camel函数, 算法其他性能差别不大; 对其他稍复杂的低维问题, 当$N = 4$时, 大部分函数的优化性能要好一些. 对低维问题, 若选择过多的种群个数, 在增大计算量的同时算法的其他性能反而降低, 因此对低维问题$N = 4$是比较合理的选择.

通过表2和附表3的对比分析, 可以得出, 对于规模较大的函数优化问题, 当增加种群个数时, 算法的成功收敛次数、平均值、方差和均方差等性能有一定的提高, 时间性能急剧下降. 综合时间性能和其他性能$N = 4$是个合理的选择.

综合表1, 表2, 附表1–3, 可以看出, 无论在求解高维问题还是低维问题, 在相同的参数设置下, CMPMA算法的各个性能指标均优于MPMA算法.

4) 不同种群规模对算法性能的影响分析. 在以上的分析中, 种群个数$N = 4$无论在求解低维还是高维问题, 都是较好的选择. 本节在此基础上进一步分析种群规模大小对两种算法性能的影响. 当种群规模为$L_{\text{size}} = 36, 64, 100$, 附表4统计了MPMA及CMPMA两种算法对Camel和Shubert函数的优化结果; 附表5–8分别统计了两种算法在不同种群规模下对Sphere、Ackley、Griewank和Rastrigrin4个多维函数从10到300维变化时的优化结果, 表中结果均为算法执行30次的平均值.

从附表4–8可以看出, 在不同种群规模下, 相同种群规模时, 无论维数高低, CMPMA算法的平均值、最优值、最差值和方差、均方差、成功收敛次数及平均CPU时间均优于MPMA算法, 随着函数维数的增加, CMPMA算法表现出的更好的寻优效率和寻优能力; 两种算法对Camel函数和Shubert函数的优化能力相当, 但是CMPMA的方差和均方差均小于MPMA; 随着种群规模的增加, CMPMA算法寻优效率和寻优能力比MPMA算法表现出更好的性能. 尤其对于多维数Griewank函数和Rastrigrin函数, 在维数为100、200和300时, CMPMA得到的结果无论从最优值、平均值、最差值、方差、均方差还是成功次数上都比网格结构的好, 而且CPU运行时间短, 即在短时间内达到了更好的效果. 随着规模的变化, CMPMA算法的寻优能力越来越强.

综上, CMPMA在进化初期能实现在较大空间搜索; 而在进化后期使得求解的精度提高. 算法不仅具有较高的求解精度和求解稳定性, 能有效抑制早熟收敛, 跳出局部较优解, 而且具有较快的收敛速度. 经过多种群的信息交互, 增加了种群的多样性, 使种群不容易进入局部收敛,提高了算法的寻优能力.

6 结论(Conclusions)

本文在多智能体进化算法的结构中引入多种群交互的进化理念, 并对多智能体的动态邻域环境的结构做了链式处理, 提出了基于链式结构的多种群多智能体进化算法. 多个种群独立进行演化, 并通过与其动态邻域智能体的竞争、合作及自学习操作来完成自身的演化; 采用一种链式邻域结构, 使得智能体在邻域竞争与自学习的过程中邻域个数呈指数递减, 加快了种群进化速度, 有效地提高了算法效率. 整个演化空间的种群定期交互优秀个体信息, 一方面加快了各种群的收敛速度, 另一方面利用交互的多样化信息增加了种群的多样性, 有效防止了种群的过早收敛. 通过对典型的多维测试函数优化问题的多次仿真结果表明: 基于链式结构的多种群多智能体进化算法搜索速度快, 且可以获得高质量的解, 是求解高维复杂优化问题的有效算法. 如何运用该算法解决实际优化问题是下一步要研究的问题.

参考文献(References):

[1] 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 Weicai, XUE Mingzhi, LIU Jing, et al. Multi-agent genetic algorithm for super high- dimensional function optimization [J]. *Progress in Natural Science*, 2003, 13(10): 1078 – 1083.)

[2] ZHONG W C, LIU J, XUE M Z, et al. A multi-agent genetic algorithm for global numerical optimization [J]. *IEEE Transactions on Systems, Man, and Cybernetics, Part B: Cybernetics*, 2004, 34(2): 1128 – 1141.

[3] LIU J, ZHONG W C, JIAO L C. A multiagent evolutionary algorithm for constraint satisfaction problems [J]. *IEEE Transactions on Systems, Man, and Cybernetics, Part B: Cybernetics*, 2006, 36(1): 1128 – 1141.

[4] ZENG X P, LI Y M, JIAN Q. A dynamic chain-like agent genetic algorithm for global numerical optimization and feature selection [J]. *Neurocomputing*, 2009, 72(4–6): 1214 – 1228.

[5] 吴亚丽, 张万良, 张立香. 多阶段多产品调度问题的链式智能体遗传算法 [J]. 控制理论与应用, 2011, 28(2): 215 – 222.
(WU Yali, ZHANG Wanliang, ZHANG Lixiang. Chain-like agent genetic algorithm for multi-stage multi-product scheduling problem [J]. *Control Theory & Applications*, 2011, 28(2): 215 – 222.)

[6] 梁昌勇, 陆青, 张恩桥, 等. 基于均匀设计的多智能体遗传算法研究 [J]. 系统工程学报, 2009, 24(1): 109 – 113.
(LIANG Changyong, LU Qing, ZHANG Enqiao, et al. Research on multi-agent genetic algorithm based on uniform design [J]. *Journal of Systems Engineering*, 2009, 24(1): 109 – 113.)

[7] 何大阔, 高广宇, 王福利, 等. 多智能体差分进化算法 [J]. 控制与决策, 2011, 26(7), 962 – 972.
(HE Dakuo, GAO Guangyu, WANG Fuli, et al. Multi-agent differential evolution algorithm [J]. *Control and Decision*, 2011, 26(7): 962 – 972.)

[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 – 2055.
(HUANG Yongqing, LU Qing, LIANG Changyong. Interation multi-agent evolutionary algorithm and application [J]. *Journal of System Simulation*, 2006, 18(7): 2030 – 2005.)

[9] 姚志红, 赵国文. 多种群变换遗传算法及其在优化调度中的应用 [J]. 控制理论与应用, 2001, 18(7): 882 – 886.
(YAO Zhihong, ZHAO Guowen. The comparison of multi-reproduction groups of genetic algorithms and its application in the optimization schedule [J]. *Control Theory & Applications*, 2001, 18(7): 882 – 886.)

[10] LEUNG Y W, WANG Y. An orthogonal genetic algorithm with quantization for global numerical optimization [J]. *IEEE Transactions on Evolutionary Computation*, 2001, 5 (1): 41 – 53.

[11] JIE J, ZENG J C, HAN C Z. Self-organization particle swarm optimization based on information feedback [C] //*Advances in Natural Computation-Second International Conference, Xi'an Lecture Notes in Computer Science*. Heidelberg: Springerlink, 2006, 4221: 913 – 922.

作者简介:

吴亚丽 (1975–), 女, 博士, 副教授, 主要研究方向为复杂系统建模、优化与仿真, E-mail: yliwu@xaut.edu.cn;

靳笑一 (1986–), 女, 硕士研究生, 主要研究方向为大系统理论与优化, E-mail: jinxiaoyijxy@163.com;

刘 格 (1988–), 女, 硕士研究生, 主要研究方向为大系统理论与优化, E-mail: liuge198872@163.com.

附录 表格(Appendix tables)

表格如下. 见附表1–8.

附表 1 种群数为4时两种算法在求解各测试函数的性能指标

Appended Table 1 The comparison results of two algorithms in functions with 4 population

函数	算法	最好值	最差值	平均值	方差	均方差	Succ	CPU / s
Camel	$\mathrm{Gen}=50; X_{\max}=10; \varepsilon=0.00001; sR=0.8$							
	MPMA	−1.03163	−1.03163	−1.03163	2.5E−16	1.6E−08	30	1.1393
	CMPMA	−1.03163	−1.03163	−1.03163	1.2E−16	1.1E−08	30	0.8850
Shubert	$\mathrm{Gen}=200; X_{\max}=10; \varepsilon=0.0001; sR=0.8$							
	MPMA	−186.731	−186.731	−186.731	6.0E−11	7.8E−06	30	1.7560
	CMPMA	−186.731	−186.731	−186.731	1.2E−11	3.5E−06	30	1.4033
Sphere	$\mathrm{dimension}=10; \mathrm{Gen}=150; X_{\max}=10; \varepsilon=0.00001; sR=0.8$							
	MPMA	9.15E−68	9.01E−54	1.27E−60	1.6886	1.2995	30	9.8164
	CMPMA	2.03E−76	4.22E−68	2.54E−70	0.0012	0.0346	30	8.1033
	$\mathrm{dimension}=20; \mathrm{Gen}=250; X_{\max}=10; \varepsilon=0.0001; sR=0.8$							
	MPMA	5.69E−69	283.0899	15.95688	−263.402	0	30	38.6395
	CMPMA	3.44E−73	1.96E−63	6.59E−65	1.3E−127	3.6E−64	30	37.4448
	$\mathrm{dimension}=30; \mathrm{Gen}=350; X_{\max}=10; \varepsilon=0.001; sR=0.8$							
	MPMA	1.24E−83	1.26E−73	6.26E−75	6.2E−148	2.5E−74	30	65.9791
	CMPMA	2.2E−86	1.14E−75	11.2803	−131.632	0	30	64.3153
Ackley	$\mathrm{dimension}=10; \mathrm{Gen}=150; X_{\max}=10; \varepsilon=0.00001; sR=0.8$							
	MPMA	2.66E−15	12.1977	1.3313	1.8335	0	30	12.6520
	CMPMA	8.90E−16	2.66E−15	2.31E−15	1.18E−30	1.08E−15	30	12.5920
	$\mathrm{dimension}=20; \mathrm{Gen}=250; X_{\max}=10; \varepsilon=0.0001; sR=0.8$							
	MPMA	2.66E−15	13.1083	2.8641	−8.48615	0	30	27.0210
	CMPMA	2.66E−15	2.66E−15	2.66E−15	0	0	30	26.8580
	$\mathrm{dimension}=30; \mathrm{Gen}=350; X_{\max}=10; \varepsilon=0.001; sR=0.8$							
	MPMA	2.66E−15	3.67E−15	2.66E−15	1.84E−15	0	30	45.7170
	CMPMA	2.66E−15	2.66E−15	2.66E−15	0	0	30	44.8200

续附表1

函数	算法	最好值	最差值	平均值	方差	均方差	Succ	CPU / s
Griewank	$\text{dimension}=10;\text{Gen}=300;X_{\max}=600;\varepsilon=0.05;sR=0.8$							
	MPMA	0	0.14282	0.03084	0.00127	0.03559	26	7.6350
	CMPMA	0	0.09825	0.02321	0.00082	0.02858	27	7.4930
	$\text{dimension}=20;\text{Gen}=250;X_{\max}=600;\varepsilon=0.01;sR=0.8$							
	MPMA	0	0.05726	0.00476	1.51E−04	0.01225	26	8.8940
	CMPMA	0	0.04009	0.00296	8.31E−05	0.00912	27	8.6604
	$\text{dimension}=30;\text{Gen}=250;X_{\max}=600;\varepsilon=0.01;sR=0.8$							
	MPMA	0	0.02917	0.00237	5.42E−05	0.00736	27	6.0360
	CMPMA	0	0.02857	0.00095	2.72E−05	0.00522	29	5.7750
Rastrigrin	$\text{dimension}=10;\text{Gen}=500;X_{\max}=5.12;\varepsilon=0.05;sR=1$							
	MPMA	0	0	0	0	0	30	12.1770
	CMPMA	0	0	0	0	0	30	10.7890
	$\text{dimension}=20;\text{Gen}=500;X_{\max}=5.12;\varepsilon=0.1;sR=1$							
	MPMA	0	20.8728	4.996776	46.4089	6.8124	18	15.6250
	CMPMA	0	26.8719	2.841445	40.8914	6.3946	24	13.8880
	$\text{dimension}=30;\text{Gen}=600;X_{\max}=5.12;\varepsilon=0.1;sR=1$							
	MPMA	0	50.8446	16.80369	370.493	19.2481	26	23.9780
	CMPMA	0	48.7532	9.63878	204.535	14.3015	28	20.5660

附表 2 种数为6时两种算法在求解各测试函数的性能指标

Appended Table 2 The comparison results of two algorithms in functions with 6 population

函数	算法	最好值	最差值	平均值	方差	均方差	Succ	CPU / s
Camel	$\text{Gen}=50;X_{\max}=10;\varepsilon=0.00001;sR=0.8$							
	MPMA	−1.03163	−1.03163	−1.03163	−3.7E−16	0	30	1.6895
	CMPMA	−1.03163	−1.03163	−1.03163	−2.5E−16	0	30	1.5727
Shubert	$\text{Gen}=200;X_{\max}=10;\varepsilon=0.0001;sR=0.8$							
	MPMA	−186.731	−186.731	−186.731	2.2E−08	0.00015	30	3.6807
	CMPMA	−186.731	−186.731	−186.731	7.6E−11	8.7E−06	30	1.3019
Sphere	$\text{dimension}=10;\text{Gen}=150;X_{\max}=10;\varepsilon=0.00001;sR=0.8$							
	MPMA	4.24E−66	1.61E−44	1.4994	−2.3257	0	30	19.1416
	CMPMA	1.81E−67	4.98E−47	1.66E−48	8.28E−95	9.1E−48	30	18.6106
	$\text{dimension}=20;\text{Gen}=250;X_{\max}=10;\varepsilon=0.0001;sR=0.8$							
	MPMA	2.47E−66	104.8363	5.421801	−30.4096	0	30	44.5851
	CMPMA	1.81E−71	3.38E−60	1.2E−61	3.8E−121	6.2E−61	30	43.3406
	$\text{dimension}=30;\text{Gen}=350;X_{\max}=10;\varepsilon=0.001;sR=0.8$							
	MPMA	8.28E−85	1.19E−71	28.4233	−897.65	0	30	79.4937
	CMPMA	2E−86	6.05E−78	6.07E−79	3.7E−156	1.9E−78	30	77.3608
Ackley	$\text{dimension}=10;\text{Gen}=150;X_{\max}=10;\varepsilon=0.00001;sR=0.8$							
	MPMA	8.9E−16	2.66E−15	1.3665	1.9318	1.4079	30	33.7294
	CMPMA	8.9E−16	2.66E−15	2.3E−15	1.2E−30	1.1E−15	30	33.2640
	$\text{dimension}=20;\text{Gen}=250;X_{\max}=10;\varepsilon=0.0001;sR=0.8$							
	MPMA	2.66E−15	12.8413	2.0355	−4.2861	0	30	103.536
	CMPMA	8.9E−16	2.66E−15	2.43E−15	8.1E−31	9.0E−16	30	76.3476
	$\text{dimension}=30;\text{Gen}=350;X_{\max}=10;\varepsilon=0.001;sR=0.8$							
	MPMA	2.66E−15	2.66E−15	2.8337	−8.3069	0	30	148.926
	CMPMA	2.66E−15	2.66E−15	2.66E−15	0	0	30	132.052

续附表2

函数	算法	最好值	最差值	平均值	方差	均方差	Succ	CPU/s
Griewank	$\text{dimension}=10;\text{Gen}=300;X_{\max}=600;\varepsilon=0.05;sR=0.8$							
	MPMA	0	0.2140	0.0512	0.0050	0.0710	27	13.6047
	CMPMA	0	0.0893	0.0173	0.0008	0.0286	29	12.9075
	$\text{dimension}=20;\text{Gen}=250;X_{\max}=600;\varepsilon=0.01;sR=0.8$							
	MPMA	0	0.0413	0.0091	0.0003	0.01641	27	15.2041
	CMPMA	0	0.0200	0.0020	4.04E−05	0.0064	29	14.9933
	$\text{dimension}=30;\text{Gen}=250;X_{\max}=600;\varepsilon=0.01;sR=0.8$							
	MPMA	0	0.063605	0.008481	0.0004	0.0205	28	11.6461
	CMPMA	0	0.010004	0.001	1.0E−05	0.0032	29	11.3481
Rastrigrin	$\text{dimension}=10;\text{Gen}=500;X_{\max}=5.12;\varepsilon=0.05;sR=1$							
	MPMA	0	0	0	0	0	30	30.2815
	CMPMA	0	0	0	0	0	30	28.6724
	$\text{dimension}=20;\text{Gen}=500;X_{\max}=5.12;\varepsilon=0.1;sR=1$							
	MPMA	0	21.89673	5.075773	52.3771	7.2372	26	40.6812
	CMPMA	0	17.91757	3.787991	51.1689	7.1533	27	39.2994
	$\text{dimension}=30;\text{Gen}=600;X_{\max}=5.12;\varepsilon=0.1;sR=1$							
	MPMA	0	47.7627	14.57379	249.655	15.8005	24	63.8608
	CMPMA	0	41.8025	6.867199	220.931	14.8638	28	59.3057

附表 3 不同种群个数时两种算法求解高维函数的性能指标

Appended Table 3 The comparison results of two algorithms in high dimension functions with different population

函数	算法	最好值	最差值	平均值	方差	均方差	Succ	CPU/s
Sphere	$\text{dimension}=100;\text{Gen}=500;X_{\max}=10;\varepsilon=0.1;sR=0.8$							
	$N=4$							
	MPMA	4.35E−64	5.15E−46	1.73E−47	8.83E−93	9.4E−47	30	136.9684
	CMPMA	6.30E−75	6.62E−47	3.26E−48	1.63E−94	1.3E−47	30	127.1526
	$N=6$							
	MPMA	6.93E−86	3.74E−63	3.74E−64	1.4E−126	1.2E−63	30	301.6794
	CMPMA	1.18E−91	6.17E−76	6.24E−77	3.8E−152	1.9E−76	30	290.4099
	$\text{dimension}=200;\text{Gen}=500;X_{\max}=10;\varepsilon=0.1;sR=0.8$							
	$N=4$							
	MPMA	2.27E−65	9.01E−09	5.66E−10	4.62E−18	2.2E−09	30	288.2266
	CMPMA	1.19E−73	1.44E−14	5.93E−16	6.96E−30	2.6E−15	30	235.7977
	$N=6$							
	MPMA	3.1E−79	1.6E−52	3.2E−60	2.4E−14	1.5E−7	30	327.0201
	CMPMA	1.0E−99	2.0E−72	2.0E−73	4.0E−145	6.3E−73	30	274.7423
	$\text{dimension}=300\text{Gen}=500;X_{\max}=10;\varepsilon=0.1;sR=0.8$							
	$N=4$							
	MPMA	2.27E−11	2.578779	0.169968	0.229099	0.47864	21	669.8335
	CMPMA	6.06E−21	0.114071	0.003829	0.000434	0.02082	29	336.8166
	$N=6$							
	MPMA	1.7E−92	0.0033	3.5E−04	2.06E−08	1.4E−04	30	917.722
	CMPMA	2.6E−123	0.0001	1.5E−05	1.16E−09	3.4E−05	30	592.754
Ackley	$\text{dimension}=100;\text{Gen}=500;X_{\max}=10;\varepsilon=0.1;sR=0.8$							
	$N=4$							
	MPMA	2.66E−15	1.34E−13	8.7E−15	5.64E−28	2.38E−14	30	152.536
	CMPMA	2.66E−15	6.22E−15	5.27E−15	2.55E−30	1.60E−15	30	145.127

续附表3

函数	算法	最好值	最差值	平均值	方差	均方差	Succ	CPU / s
Ackley	dimension = 100; Gen = 500; $X_{\max}=10$; $\varepsilon=0.1$; $sR=0.8$							
	$N=6$							
	MPMA	2.66E−15	6.22E−15	5.51E−15	2.7E−30	1.7E−15	30	341.531
	CMPMA	2.66E−15	6.22E−15	5.15E−15	2.2E−30	1.5E−15	30	327.338
	dimension = 200; Gen = 500; $X_{\max}=10$; $\varepsilon=0.1$; $sR=0.8$							
	$N=4$							
	MPMA	6.22E−15	4.81746	2.19425	4.0253	2.00633	23	297.738
	CMPMA	6.22E−15	5.43037	1.89246	5.1333	2.26568	29	275.536
	$N=6$							
	MPMA	1.33E−14	5.4402	3.5593	5.1505	2.2695	22	678.904
	CMPMA	6.22E−15	4.7749	2.1371	3.9005	1.9750	25	634.470
	dimension = 300; Gen = 500; $X_{\max}=10$; $\varepsilon=0.1$; $sR=0.8$							
	$N=4$							
	MPMA	3.64E−05	6.38458	5.2575	1.2447	1.11568	22	461.682
	CMPMA	1.52E−13	5.4142	4.0814	2.8638	1.50228	26	426.415
	$N=6$							
	MPMA	1.41E−15	2.0315	1.3009	0.0286	0.1691	26	945.267
	CMPMA	2.66E−15	1.24E−8	3.4E−14	2.0E−20	4.5E−10	29	857.109
Griewank	dimension = 100; Gen = 300; $X_{\max}=600$; $\varepsilon=0.01$; $sR=0.8$							
	$N=4$							
	MPMA	0	0.04593	0.00153	7.03E−05	0.00839	29	15.7380
	CMPMA	0	0.09190	0.00306	2.82E−04	0.01678	29	13.4000
	$N=6$							
	MPMA	1.11E−16	0.1544	0.017054	0.0024	0.0485	28	29.3318
	CMPMA	0	9.55E−10	1.16E−10	8.84E−20	2.97E−10	30	28.8340
	dimension = 200; Gen = 300; $X_{\max}=600$; $\varepsilon=0.01$; $sR=0.8$							
	$N=4$							
	MPMA	5.71E−14	0.27355	0.01894	0.00294	0.05419	24	52.7560
	CMPMA	0	1.65E−06	5.51E−08	9.11E−14	3.02E−07	30	49.7220
	$N=6$							
	MPMA	0	0.0064	0.0006	4.12E−06	0.0020	30	110.475
	CMPMA	0	0	0	0	0	30	108.052
	dimension = 300; Gen = 300; $X_{\max}=600$; $\varepsilon=0.01$; $sR=0.8$							
	$N=4$							
	MPMA	0	0.045981	0.00153	7.05E−05	0.00839	29	281.763
	CMPMA	0	6.85E−07	3.44E−08	1.64E−14	1.28E−07	30	276.927
	$N=6$							
	MPMA	0	0	0	0	0	30	544.206
	CMPMA	0	0	0	0	0	30	529.441
Rastrigrin	dimension = 100; Gen = 1000; $X_{\max}=5.12$; $\varepsilon=0.1$; $sR=1$							
	$N=4$							
	MPMA	0	251.4463	28.64123	4279.85	65.4205	22	79.393
	CMPMA	0	171.6871	27.11921	3511.09	59.2544	24	70.151
	$N=6$							
	MPMA	0	0	0	0	0	30	171.393
	CMPMA	0	0	0	0	0	30	145.270
	dimension = 200; Gen = 1000; $X_{\max}=5.12$; $\varepsilon=0.1$; $sR=1$							
	$N=4$							
	MPMA	5.71E−14	0.273554	0.018941	0.00293	0.0541	24	122.756
	CMPMA	0	1.65E−06	5.51E−08	9.11E−14	3.02E−07	30	109.722

续附表3

函数	算法	最好值	最差值	平均值	方差	均方差	Succ	CPU/s
Rastrigrin	$\text{dimension}=200;\text{Gen}=1000;X_{\max}=5.12;\varepsilon=0.1;sR=1$							
	$N=6$							
	MPMA	0	0	0	0	0	30	325.477
	CMPMA	0	0	0	0	0	30	285.113
	$\text{dimension}=300;\text{Gen}=1000;X_{\max}=5.12;\varepsilon=0.1;sR=1$							
	$N=4$							
	MPMA	0	0.045981	0.00153	7.05E−05	0.00839	29	281.763
	CMPMA	0	6.85E−07	3.44E−08	1.64E−14	1.28E−07	30	276.927
	$N=6$							
	MPMA	0	0	0	0	0	30	682.598
	CMPMA	0	0	0	0	0	30	665.436

附表 4 两种算法对Camel和Shubert函数的优化结果

Appended Table 4 The optimization results of two algorithms for Camel and Shubert functions

函数	算法	最好值	最差值	平均值	方差	均方差	Succ	CPU/s
Camel	$\text{Gen}=50;X_{\max}=10;\varepsilon=0.00001;sR=0.8$							
	$L_{\text{size}}=36$							
	MPMA	−1.03163	−1.03163	−1.03163	−1.2E−16	0	30	0.2906
	CMPMA	−1.03163	−1.03163	−1.03163	1.2E−15	3.5E−08	30	0.2767
	$L_{\text{size}}=64$							
	MPMA	−1.03163	−1.03163	−1.03163	2.5E−16	1.6E−08	30	1.1393
	CMPMA	−1.03163	−1.03163	−1.03163	1.2E−16	1.1E−08	30	0.8850
	$L_{\text{size}}=100$							
	MPMA	−1.03163	−1.03163	−1.03163	6.1E−16	2.4E−08	30	1.4633
	CMPMA	−1.03163	−1.03163	−1.03163	0	0	30	1.3609
Shubert	$\text{Gen}=200;X_{\max}=10;\varepsilon=0.0001;sR=0.8$							
	$L_{\text{size}}=36$							
	MPMA	−186.731	−169.587	−186.141	9.78509	3.12811	30	0.4675
	CMPMA	−186.731	−186.728	−186.731	2.6E−07	0.00052	30	0.4469
	$L_{\text{size}}=64$							
	MPMA	−186.731	−186.731	−186.731	6.0E−11	7.8E−06	30	1.7560
	CMPMA	−186.731	−186.731	−186.731	1.2E−11	3.5E−06	30	1.4033
	$L_{\text{size}}=100$							
	MPMA	−186.731	−186.731	−186.731	4.4E−11	4.0E−12	30	2.8392
	CMPMA	−186.731	−186.731	−186.731	6.7E−06	2.0E−06	30	2.1572

附表5 不同种群规模对不同维数的Sphere函数的优化结果

Appended Table 5 The optimization results of different population size for Sphere function with different dimensions

维数	算法	最好值	最差值	平均值	方差	均方差	Succ	CPU/s
10	$\text{Gen}=150;X_{\max}=10;\varepsilon=0.00001;sR=0.8$							
	$L_{\text{size}}=36$							
	MPMA	1.59E−64	5.11E−55	1.611365	−2.68603	0	30	7.0173
	CMPMA	6.46E−73	9.56E−56	3.29E−57	3.0E−112	1.7E−56	30	6.6950
	$L_{\text{size}}=64$							
	MPMA	9.15E−68	9.01E−54	1.27E−60	1.6886	1.2995	30	9.8164
	CMPMA	2.03E−76	4.22E−68	2.54E−70	0.0012	0.0346	30	8.1033
	$L_{\text{size}}=100$							
	MPMA	5.65E−68	1.12E−49	1.550901	−2.48823	0	30	32.6059
	CMPMA	3.48E−68	3.6E−56	1.41E−57	4.4E−113	6.6E−57	30	32.4832

续附表5

维数	算法	最好值	最差值	平均值	方差	均方差	Succ	CPU/s
20	Gen = 250; $X_{\max}=10$; $\varepsilon=0.0001$; $sR=0.8$							
	$L_{\text{size}}=36$							
	MPMA	1.57E−63	54.14182	2.853024	−8.42043	0	30	23.7501
	CMPMA	3.98E−73	2.46E−65	1.59E−66	2.7E−131	5.2E−66	30	22.0294
	$L_{\text{size}}=64$							
	MPMA	5.69E−69	283.0899	15.95688	−263.402	0	30	38.6395
	CMPMA	3.44E−73	1.96E−63	6.59E−65	1.3E−127	3.6E−64	30	37.4448
	$L_{\text{size}}=100$							
	MPMA	2.33E−69	104.3296	6.433757	−42.8206	0	30	74.8285
	CMPMA	2.4E−76	6.97E−65	4.35E−66	1.8E−130	1.3E−65	30	67.9848
30	Gen = 350; $X_{\max}=10$; $\varepsilon=0.001$; $sR=0.8$							
	$L_{\text{size}}=36$							
	MPMA	6.18E−83	1.87E−73	12.35291	−157.856	0	30	38.62182
	CMPMA	1.57E−84	1.41E−74	5.74E−76	6.6E−150	2.6E−75	30	38.22658
	$L_{\text{size}}=64$							
	MPMA	1.24E−83	1.26E−73	6.26E−75	6.2E−148	2.5E−74	30	65.9791
	CMPMA	2.2E−86	1.14E−75	11.2803	−131.632	0	30	64.3153
	$L_{\text{size}}=100$							
	MPMA	2.3E−84	1.4E−72	10.29419	−109.625	0	30	57.64844
	CMPMA	3.77E−88	5.68E−74	8.73E−74	1.1E−145	3.2E−73	30	57.30487
100	Gen = 500; $X_{\max}=10$; $\varepsilon=0.1$; $sR=0.8$							
	$L_{\text{size}}=36$							
	MPMA	2.87E−75	1.47E−49	4.92E−51	7.2E−100	2.7E−50	30	122.5775
	CMPMA	8.69E−96	1.75E−51	8.08E−53	1.1E−10	3.3E−52	30	117.9484
	$L_{\text{size}}=64$							
	MPMA	4.35E−64	5.15E−46	1.73E−47	8.83E−93	9.4E−47	30	136.9684
	CMPMA	6.30E−75	6.62E−47	3.26E−48	1.63E−94	1.3E−47	30	127.1526
	$L_{\text{size}}=100$							
	MPMA	1.12E−78	1.56E−45	7.43E−47	9.38E−92	3.06E−46	30	203.9249
	CMPMA	2.69E−81	1.69E−48	1.28E−49	1.54E−97	3.92E−49	30	130.4855
200	Gen = 500; $X_{\max}=10$; $\varepsilon=0.1$; $sR=0.8$							
	$L_{\text{size}}=36$							
	MPMA	9.13E−79	1.96E−45	6.68E−47	1.28E−91	3.6E−46	30	143.4301
	CMPMA	7.23E−90	6.14E−48	2.17E−49	1.26E−96	1.1E−48	30	140.1902
	$L_{\text{size}}=64$							
	MPMA	2.27E−65	9.01E−09	5.66E−10	4.62E−18	2.2E−09	30	288.2266
	CMPMA	1.19E−73	1.44E−14	5.93E−16	6.96E−30	2.6E−15	30	235.7977
	$L_{\text{size}}=100$							
	MPMA	1.12E−78	1.56E−45	7.43E−47	9.38E−92	3.06E−46	30	202.7920
	CMPMA	2.69E−81	1.69E−48	1.28E−49	1.54E−97	3.92E−49	30	130.9253
300	Gen = 500; $X_{\max}=10$; $\varepsilon=0.1$; $sR=0.8$							
	$L_{\text{size}}=36$							
	MPMA	7.54E−20	1.738528	0.102594	0.106389	0.32617	24	539.0687
	CMPMA	4.01E−25	0.018899	0.00064	1.19E−05	0.00345	30	441.0779
	$L_{\text{size}}=64$							
	MPMA	2.27E−11	2.578779	0.169968	0.229099	0.47864	21	669.8335
	CMPMA	6.06E−21	0.114071	0.003829	0.000434	0.02082	29	336.8166
	$L_{\text{size}}=100$							
	MPMA	2.17E−08	0.911426	0.056726	0.031879	0.17855	27	784.793
	CMPMA	1.22E−18	0.001095	6.36E−05	4.48E−08	0.00021	30	518.5459

附表 6 不同种群规模对不同维数的Ackley函数的优化结果

Appended Table 6 The optimization results of different population size for Ackley function with different dimensions

维数	算法	最好值	最差值	平均值	方差	均方差	Succ	CPU / s
10	$\mathrm{Gen}=150; X_{\max}=10; \varepsilon=0.00001; sR=0.8$							
	$L_{\mathrm{size}}=36$							
	MPMA	8.9E−16	2.66E−15	1.132675	−1.3272	0	30	12.1312
	CMPMA	8.9E−16	2.66E−15	2.31E−15	1.2E−30	1.1E−15	30	11.4839
	$L_{\mathrm{size}}=64$							
	MPMA	2.66E−15	12.1977	1.3313	1.8335	0	30	12.6520
	CMPMA	8.90E−16	2.66E−15	2.31E−15	1.18E−30	1.08E−15	30	12.5920
	$L_{\mathrm{size}}=100$							
	MPMA	2.665E−15	2.665E−15	0.9254149	0.88592	0	30	19.3011
	CMPMA	8.9E−16	2.99E−15	2.99E−15	0	0	30	18.1703
20	$\mathrm{Gen}=250; X_{\max}=10; \varepsilon=0.0001; sR=0.8$							
	$L_{\mathrm{size}}=36$							
	MPMA	2.66E−15	14.2605	1.82923	−3.4615	0	30	18.7744
	CMPMA	2.66E−15	2.66E−15	2.66E−15	0	0	30	16.7039
	$L_{\mathrm{size}}=64$							
	MPMA	2.66E−15	13.1083	2.8641	−8.48615	0	30	27.0210
	CMPMA	2.66E−15	2.66E−15	2.66E−15	0	0	30	26.8580
	$L_{\mathrm{size}}=100$							
	MPMA	2.66E−15	12.78738	1.652394	−2.8246	0	30	30.2687
	CMPMA	2.66E−15	2.66E−15	2.66E−15	0	0	30	30.1101
30	$\mathrm{Gen}=350; X_{\max}=10; \varepsilon=0.001; sR=0.8$							
	$L_{\mathrm{size}}=36$							
	MPMA	2.66E−15	2.66E−15	2.47427	−6.3331	0	30	37.6024
	CMPMA	2.66E−15	2.66E−15	2.66E−15	0	0	30	32.3696
	$L_{\mathrm{size}}=64$							
	MPMA	2.66E−15	3.67E−15	2.66E−15	1.84E−15	0	30	45.7170
	CMPMA	2.66E−15	2.66E−15	2.66E−15	0	0	30	44.8200
	$L_{\mathrm{size}}=100$							
	MPMA	2.66E−15	6.22E−15	2.65498	−7.2920	0	30	55.0722
	CMPMA	2.66E−15	6.22E−15	2.78E−15	4.2E−31	6.5E−16	30	54.6926
100	$\mathrm{Gen}=500; X_{\max}=10; \varepsilon=0.1; sR=0.8$							
	$L_{\mathrm{size}}=36$							
	MPMA	2.66E−15	2.404153	0.080138	0.19267	0.43895	29	146.238
	CMPMA	2.66E−15	6.22E−15	4.8E−15	3.1E−30	1.8E−15	30	144.797
	$L_{\mathrm{size}}=64$							
	MPMA	2.66E−15	1.34E−13	8.7E−15	5.64E−28	2.38E−14	30	152.536
	CMPMA	2.66E−15	6.22E−15	5.27E−15	2.55E−30	1.60E−15	30	145.127
	$L_{\mathrm{size}}=100$							
	MPMA	2.66E−15	1.33E−14	4.56E−15	5.9E−30	2.4E−15	30	233.855
	CMPMA	2.66E−15	6.22E−15	5.27E−15	2.6E−30	1.6E−15	30	229.810
200	$\mathrm{Gen}=500; X_{\max}=10; \varepsilon=0.1; sR=0.8$							
	$L_{\mathrm{size}}=36$							
	MPMA	6.22E−15	5.886163	1.981127	5.03829	2.24461	19	159.205
	CMPMA	2.66E−15	5.450582	1.633357	5.06203	2.24988	25	112.586
	$L_{\mathrm{size}}=64$							
	MPMA	6.22E−15	4.81746	2.19425	4.0253	2.00633	23	297.738
	CMPMA	6.22E−15	5.43037	1.89246	5.1333	2.26568	29	275.536

续附表6

维数	算法	最好值	最差值	平均值	方差	均方差	Succ	CPU / s
200	$\mathrm{Gen}=500; X_{\max}=10; \varepsilon=0.1; sR=0.8$							
	$L_{\mathrm{size}}=100$							
	MPMA	6.22E−15	5.45368	1.645546	3.86868	1.96689	22	461.213
	CMPMA	2.66E−15	4.636408	2.348008	4.27216	2.06692	27	456.669
300	$\mathrm{Gen}=500; X_{\max}=10; \varepsilon=0.1; sR=0.8$							
	$L_{\mathrm{size}}=36$							
	MPMA	1.69E−14	6.051794	5.139738	1.16962	1.08149	16	334.853
	CMPMA	6.69E−15	5.751865	5.09629	0.14889	0.38586	18	312.558
	$L_{\mathrm{size}}=64$							
	MPMA	3.64E−05	6.38458	5.2575	1.2447	1.11568	22	461.682
	CMPMA	1.52E−13	5.4142	4.0814	2.8638	1.50228	26	426.415
	$L_{\mathrm{size}}=100$							
	MPMA	1.32E−14	5.603522	4.713897	0.83933	0.91615	17	1385.54
	CMPMA	3.37E−15	5.157194	4.299704	0.25178	0.50177	25	692.646

附表7 不同种群规模对不同维数的Griewank函数的优化结果

Appended Table 7 The optimization results of different population size for Griewank function with different dimensions

维数	算法	最好值	最差值	平均值	方差	均方差	Succ	CPU / s
10	$\mathrm{Gen}=300; X_{\max}=600; \varepsilon=0.05; sR=0.8$							
	$L_{\mathrm{size}}=36$							
	MPMA	0	0.132263	0.026066	0.00096	0.0311	25	4.2715
	CMPMA	0	0.078797	0.018497	0.00053	0.0231	26	3.9302
	$L_{\mathrm{size}}=64$							
	MPMA	0	0.14282	0.03084	0.00127	0.03559	26	7.6350
	CMPMA	0	0.09825	0.02321	0.00082	0.02858	27	7.4930
	$L_{\mathrm{size}}=100$							
	MPMA	0	0.072225	0.024315	0.0005	0.0229	24	36.7880
	CMPMA	0	0.066427	0.018871	0.0004	0.0191	26	36.3184
20	$\mathrm{Gen}=250; X_{\max}=600; \varepsilon=0.01; sR=0.8$							
	$L_{\mathrm{size}}=36$							
	MPMA	0	0.029559	0.002552	5.2E−05	0.00723	27	5.5022
	CMPMA	0	0.019701	0.00115	1.9E−05	0.00442	28	5.2148
	$L_{\mathrm{size}}=64$							
	MPMA	0	0.05726	0.00476	1.51E−04	0.01225	26	8.8940
	CMPMA	0	0.04009	0.00296	8.31E−05	0.00912	27	8.6604
	$L_{\mathrm{size}}=100$							
	MPMA	0	0.0418	0.0032	7.9E−05	0.0089	26	40.1938
	CMPMA	0	0.0221	0.0021	2.9E−05	0.0054	28	39.6486
30	$\mathrm{Gen}=250; X_{\max}=600; \varepsilon=0.01; sR=0.8$							
	$L_{\mathrm{size}}=36$							
	MPMA	0	0.016329	0.00109	1.3E−05	0.00359	28	4.4998
	CMPMA	0	0.013811	0.00046	6.4E−06	0.00252	29	4.4022
	$L_{\mathrm{size}}=64$							
	MPMA	0	0.02917	0.00237	5.42E−05	0.00736	27	6.0360
	CMPMA	0	0.02857	0.00095	2.72E−05	0.00522	29	5.7750
	$L_{\mathrm{size}}=100$							
	MPMA	0	0.0345	0.0027	5.73E−05	0.0076	26	31.3586
	CMPMA	0	0	0	0	0	30	12.2145

续附表7

维数	算法	最好值	最差值	平均值	方差	均方差	Succ	CPU/s
100	$\mathrm{Gen}=300; X_{\max}=600; \varepsilon=0.01; sR=0.8$							
	$L_{\mathrm{size}}=36$							
	MPMA	8.63E−14	0.1278	0.0043	0.0006	0.0233	29	12.7904
	CMPMA	0	0.1091	0.0036	0.0004	0.0199	29	12.4408
	$L_{\mathrm{size}}=64$							
	MPMA	0	0.04593	0.00153	7.03E−05	0.00839	29	15.7380
	CMPMA	0	0.09190	0.00306	2.82E−04	0.01678	29	13.4000
	$L_{\mathrm{size}}=100$							
	MPMA	2.78E−12	0.2470	0.0125	0.0025	0.0499	28	80.9986
	CMPMA	2.11E−15	0.0662	0.0022	0.0002	0.0121	29	75.3931
200	$\mathrm{Gen}=300; X_{\max}=600; \varepsilon=0.01; sR=0.8$							
	$L_{\mathrm{size}}=36$							
	MPMA	5.09E−11	0.317197	0.025132	0.00620	0.07875	27	41.641
	CMPMA	0	0.051335	0.001747	8.8E−05	0.00937	29	40.667
	$L_{\mathrm{size}}=64$							
	MPMA	5.71E−14	0.27355	0.01894	0.00294	0.05419	24	52.7560
	CMPMA	0	1.65E−06	5.51E−08	9.11E−14	3.02E−07	30	49.7220
	$L_{\mathrm{size}}=100$							
	MPMA	3.82E−12	0.184946	0.01205	0.00149	0.03864	24	301.820
	CMPMA	0	0.133168	0.014415	0.00119	0.03451	26	299.149
300	$\mathrm{Gen}=300; X_{\max}=600; \varepsilon=0.01; sR=0.8$							
	$L_{\mathrm{size}}=36$							
	MPMA	2.22E−16	0.0376	0.0019	5.3E−05	0.0073	28	99.740
	CMPMA	0	3.34E−09	3.32E−10	7.4E−19	8.6E−10	30	96.037
	$L_{\mathrm{size}}=64$							
	MPMA	0	0.045981	0.00153	7.05E−05	0.00839	29	281.763
	CMPMA	0	6.85E−07	3.44E−08	1.64E−14	1.28E−07	30	276.927
	$L_{\mathrm{size}}=100$							
	MPMA	9.66E−15	0.192635	0.006434	0.00124	0.03517	29	466.65
	CMPMA	0	0.180855	0.006293	0.00109	0.03299	29	454.51

附表 8 不同种群规模对不同维数的Rastrigrin函数的优化结果

Appended Table 8 The optimization results of different population size for Rastrigrin function with different dimensions

维数	算法	最好值	最差值	平均值	方差	均方差	Succ	CPU/s
10	$\mathrm{Gen}=500; X_{\max}=5.12; \varepsilon=0.05; sR=1$							
	$L_{\mathrm{size}}=36$							
	MPMA	0	0	0	0	0	30	10.5890
	CMPMA	0	0	0	0	0	30	9.2640
	$L_{\mathrm{size}}=64$							
	MPMA	0	0	0	0	0	30	12.1770
	CMPMA	0	0	0	0	0	30	10.7890
	$L_{\mathrm{size}}=100$							
	MPMA	0	0	0	0	0	30	112.020
	CMPMA	0	0	0	0	0	30	108.951

续附表8

维数	算法	最好值	最差值	平均值	方差	均方差	Succ	CPU / s
20	$\text{Gen}=500; X_{\max}=5.12; \varepsilon=0.1; sR=1$							
	$L_{\text{size}}=36$							
	MPMA	0	18.9042	4.2936	32.5263	5.7032	17	10.748
	CMPMA	0	13.9296	1.9934	15.2605	3.9065	23	9.483
	$L_{\text{size}}=64$							
	MPMA	0	20.8728	4.996776	46.4089	6.8124	18	15.6250
	CMPMA	0	26.8719	2.841445	40.8914	6.3946	24	13.8880
	$L_{\text{size}}=100$							
	MPMA	0	10.9445	2.3547	12.1854	3.4908	20	142.499
	CMPMA	0	10.9446	1.0613	8.1881	2.8615	26	141.264
30	$\text{Gen}=600; X_{\max}=5.12; \varepsilon=0.1; sR=1$							
	$L_{\text{size}}=36$							
	MPMA	0	35.8185	14.69361	167.241	12.9322	21	18.4000
	CMPMA	0	30.8440	8.421624	71.9191	8.48052	23	17.2053
	$L_{\text{size}}=64$							
	MPMA	0	50.8446	16.80369	370.493	19.2481	26	23.9780
	CMPMA	0	48.7532	9.63878	204.535	14.3015	28	20.5660
	$L_{\text{size}}=100$							
	MPMA	0	18.90421	6.08224	39.4192	6.2785	14	220.120
	CMPMA	0	17.90925	4.519752	37.3571	6.1120	18	213.133
100	$\text{Gen}=1000; X_{\max}=5.12; \varepsilon=0.1; sR=1$							
	$L_{\text{size}}=36$							
	MPMA	0	207.2757	51.2518	4253.75	65.2208	17	69.613
	CMPMA	0	176.0411	22.5364	3638.49	60.3199	26	58.824
	$L_{\text{size}}=64$							
	MPMA	0	251.4463	28.64123	4279.85	65.4205	22	79.393
	CMPMA	0	171.6871	27.11921	3511.09	59.2544	24	70.151
	$L_{\text{size}}=100$							
	MPMA	0	163.6368	48.2142	2681.29	51.7811	22	471.079
	CMPMA	0	116.4104	47.6378	2309.36	48.0558	26	457.886
200	$\text{Gen}=1000; X_{\max}=5.12; \varepsilon=0.1; sR=1$							
	$L_{\text{size}}=36$							
	MPMA	0	464.5793	19.8561	7286.30	85.3598	2697.549	
	CMPMA	0	24.77633	1.716227	33.8671	5.81955	27	88.340
	$L_{\text{size}}=64$							
	MPMA	5.71E−14	0.2736	0.0189	0.0029	0.0541	24	122.756
	CMPMA	0	1.65E−06	5.5E−08	9.1E−14	3.0E−07	30	109.722
	$L_{\text{size}}=100$							
	MPMA	0	0.0554	0.0411	0.0003	0.0273	21	860.412
	CMPMA	0	1.07E−08	3.1E−10	6.8E−16	2.6E−08	30	834.674
300	$\text{Gen}=1000; X_{\max}=5.12; \varepsilon=0.1; sR=1$							
	$L_{\text{size}}=36$							
	MPMA	0	630.5808	57.16358	26521.4	162.854	20	238.19
	CMPMA	0	482.9236	23.19314	7961.33	89.2263	23	232.78
	$L_{\text{size}}=64$							
	MPMA	0	0.045981	0.00153	7.05E−05	0.00839	29	281.763
	CMPMA	0	6.85E−07	3.44E−08	1.64E−14	1.28E−07	30	276.927
	$L_{\text{size}}=100$							
	MPMA	0	1.20772	1.33905	4.11E−06	0.00233	20	541.210
	CMPMA	0	2.14E−06	1.50E−09	2.51E−12	1.40E−06	26	430.660