基于猴群算法和单纯法的混合优化算法

陈 信 周永权
(广西民族大学信息科学与工程学院 南宁 530006)

摘 要 针对猴群算法求解全局优化问题精度不高和花费大量的计算时间等问题,结合传统的单纯法的搜索思想,设计出一种基于猴群算法和单纯法的混合算法。该混合算法较大程度上提高了猴群算法求解精度,且加快了猴群算法的收敛速度。通过18个标准测试函数进行了测试,结果表明,与PSO、GA与MA比较,文中提出的猴群-单纯形混合算法在函数优化方面有较强的优势,其测试函数最优解更接近理论最优解。

关键词 猴群算法,伪梯度,反向学习,单纯法,测试函数
中图法分类号 TP183 **文献标识码** A

Hybrid Algorithm Based on Monkey Algorithm and Simple Method

CHEN Xin ZHOU Yong-quan
(College of Information Science and Engineering, Guangxi University for Nationalities, Nanning 530006, China)

Abstract In view of the problem that Monkey algorithm cannot acquire solutions exactly in solving global optimization and spend a lot of time in computation, this paper designed a hybrid algorithm based on monkey algorithm and simple method which combine with the searching idea of traditional simple method. The algorithm improves the calculation accuracy and speeds up monkey algorithm converge speed in a certain degree. The simulation results show that the improved monkey-simple hybrid algorithm has strong advantage in function testing. The results are more close to the theory optimal solution.

Keywords Monkey algorithm, Pseudo-gradient, Opposition-based, Simple method, Testing functions

1 引言

近年人们为求解复杂非线性优化问题,提出了许多群智能算法,如蚁群算法(ACO)[1]、粒子群算法(PSO)[2]等,相比较传统优化方法,群智能算法实现简单、不受搜索空间和目标函数形态的制约,因而得到广泛的应用。猴群算法(Monkey Algorithm, MA)是由Zhao和Tang[3]提出的一种用于求解大规模的、多峰优化问题的新型群智能算法,其思想是模拟自然界中猴群在爬山过程中表现出来的爬、望、跳、翻等动作而设计出3个过程:爬过程主要是通过多次爬来搜索当前位置的局部最优解;望-跳过程是在到达局部最优解后通过眺望寻找一个优于当前解的点,并跳离当前点,以加快算法搜索最优解的速度;翻过程主要目的在于由当前搜索区域转移到其他区域,以避免搜索过程陷入局部最优。MA突出优点是在求解高维优化问题时,花费的时间主要是爬过程中每次迭代时目标函数的伪梯度的计算,即只需要计算当前位置的两个临近位置的目标函数值而与决策向量的维数无关,不会陷入"维灾难",同时,MA有很强的开发能力,一般不会陷入局部最优。因此,猴群算法已成功应用于求解各优化问题,如输电线扩展规划问题[4]、结构健康监测传感器优化问题[5]、入侵检测问题[6]等。但猴群算法本身也存在着缺陷,即计算精度不高、花费大量的计算时间、翻过程容易跳离搜索区域等。

针对猴群算法求解全局优化问题精度不高和花费大量的计算时间等不足,结合传统的单纯法的搜索思想,设计出一种基于猴群算法和单纯法的混合算法。该算法较大程度上提高了猴群算法求解精度,加快了猴群算法的收敛速度。通过18个标准测试函数,与PSO、GA与MA比较,测试结果表明,提出的猴群-单纯形混合算法在函数优化方面有较强优势,测试函数最优解更接近理论最优解。

2 单纯法-猴群算法(SMMA)

为了解决MA计算速度慢、计算精度不高等问题,本文通过引入反向学习的初始化方法和惯性步长,结合传统的单纯法搜索策略,提出一种单纯法-猴群算法(SMMA)。SMMA主要由以下过程组成:解的表示、初始化、爬过程、望-跳过程、翻过程、引入单纯法搜索策略。

2.1 解的表示

首先用正整数N表示空间维度,用M代表种群大小,则第i只猴子的位置用下式表示:$X_i=(x_{i1},x_{i2},\cdots,x_{in}),i=1,2,\cdots,M$,式中的各分量$x_{ij}$代表每只猴子在每一维的实际位

到稿日期:2013-01-29 返修日期:2013-04-27 本文受国家自然科学基金(61165015),广西自然科学基金(2012GXNSFDA053028),广西高等学校重大科研项目(201201ZD008)资助。

陈 信(1987-),男,硕士生,主要研究方向为计算智能及其应用;周永权(1962-),男,博士,教授,主要研究方向为计算智能、神经网络及应用,E-mail:yongquangzhou@126.com。

置。每只猴子的实际位置实际上代表了优化问题的一个决策向量。

2.2 初始化

种群的初始化对算法的全局收敛和寻优效果有着重要的影响，原猴群算法是在解空间随机初始化种群，本文引入反向学习[7,8]的初始化方法产生初始猴群。过程如下：

```
{随机初始化阶段}
for i=1 to M do
  for j=1 to N do
    x_{i,j}=x_{min,j}+(x_{max,j}-x_{min,j})rand
  end for
end for
{反学习阶段}
  for i=1 to M do
    for j=1 to N do
      x_{i,j}'=x_{min,j}+x_{max,j}-x_{i,j}
  end for
end for
```

从$\{X(M)\cup X'(M)\}$中选取适应度值最好的M只猴子作为初始种群。

2.3 爬过程

爬过程是一个通过迭代逐步改善优化问题的目标函数值的过程。每次爬仅计算当前位置的两个临近位置的目标函数值，通过比较，逐步移动的过程。爬过程步骤如下：

1)随机生成向量$\Delta x_i=(\Delta x_{i1},\Delta x_{i2},\cdots,\Delta x_{in})$，其中$\Delta x_{ij}$以相同的概率0.5取值$a$或$-a$，$j=1,2,\cdots,n$。参数$a(a>0)$叫做爬过程步长，步长$a$的大小又对优化问题的最优解精度起着决定性的作用。$a$的值越小，问题的解的精度越高，但同时要花费更多的CPU时间去计算爬过程。例如步长$a=0.001$，爬次数设置为1000，即每次迭代时爬过程至少需要计2000次目标函数值；步长$a=0.0001$，爬次数设置为10000，即每次迭代时爬过程至少需要计算20000次目标函数值。因此，每次迭代时的计算时间主要是由爬次数决定的，爬次数越大，花费的时间越长；而爬次数大小又是由步长a的精度决定的，a的精度越高，爬次数越大。

为了在解的精度和CPU计算时间中取得平衡，本文引入惯性步长，即在每次迭代的过程中逐渐减小步长，当步长a的值较大时，有利于寻找全局最优值，当a的值较小时，有利于寻找局部最优值。设定其惯性步长为：

$$a^{iter+1}=\frac{MAXGEN-iter}{MAXGEN}a^{iter},a_{\min}\leqslant a^{iter}\leqslant a_{\max}$$

式中，$a_{\min}$为最小步长，$a_{\max}$为最大步长，$MAXGEN$为最大迭代次数，$iter$为当前迭代次数，a^{iter}为第$iter$代的步长。迭代开始时，初始步长a的值为$a_{\max}$，随着迭代次数的增加，$iter$的值逐渐增大，$\frac{MAXGEN-iter}{MAXGEN}$的值逐渐减小，这样，步长$a$的值逐渐减小，最终接近于$a_{\min}$。给定步长最大值$a_{\max}$和最小值$a_{\min}$是为了让步长在一定范围内变化，可以避免算法一直进行搜索而无法收敛。对于原始的猴群算法，由于步长a固定，步长太大会影响求解的精度，太小又会降低搜索速度，因此只有根据求解的情况惯性地减小步长，才能更好地提高算法的搜索速度和寻优精度。

2)计算

$$f_{ij}'(x_i)=\frac{f(x_i+\Delta x_i)-f(x_i-\Delta x_i)}{2\Delta x_{ij}},j=1,2,\cdots,n$$

向量$f_i'(x_i)=(f_{i1}'(x_i),f_{i2}'(x_i),\cdots,f_{in}'(x_i))$叫做目标函数在点$x_i$处的伪梯度。

3)令$y_j=x_{ij}+a\cdot sign(f_{ij}'(x_i))$，$j=1,2,\cdots,n$，且$y=(y_1,y_2,\cdots,y_n)$。

4)如果y在可行解区域内，则置$x_i=y$；否则保持x_i的值不变。

重复步骤1)到4)，直到达到设置的最大爬次数或者前后两次迭代过程中的目标函数值无变化。

2.4 望-跳过程

猴群经多次爬后，每只猴子达到当前位置的最高山峰，即达到局部最优值。此时，猴子通过望动作，在视野范围内寻找一个优于当前位置的点，然后逃离当前位置。其步骤如下：

1)在区间$(x_{ij}-b,x_{ij}+b)$，$j=1,2,\cdots,n$中随机产生一个实数y_j，且$y=(y_1,y_2,\cdots,y_n)$。

2)如果y满足约束条件且有$f(y)>f(x_i)$，则置$x_i=y$。否则，重复步骤1)直到一个满足条件的点y产生。在这里，只用大于或等于$f(x_i)$的点替换x_i。

3)将y作为初始位置，重复爬过程。

其中，b叫做猴子的视野，其大小主要根据目标函数的解空间来确定。一般最优化问题的可行域越大，b的取值也越大。

2.5 翻过程

翻过程的主要目的是迫使猴群从当前的搜索区域转移到一个新的区域，从而避免陷入局部最优。选取所有猴子的位置的中心作为支点，每只猴子沿着指向支点的方向或者相反的方向翻到一个新的区域。对于第i只猴子，其翻过程如下：

1)在区间$[c,d]$中随机产生一个实数α。其中$[c,d]$叫做翻区间，其大小的选择根据优化问题的解空间的大小决定，一般优化问题可行域越大，翻区间c和d的绝对值也越大。

2)令$y_j=x_{ij}+\alpha(p_j-x_{ij})$，且$p_j=\frac{1}{M}\sum_{i=1}^{M}x_{ij}$，$j=1,2,\cdots,n$即所有猴子的位置的中点。点$p=(p_1,p_2,\cdots,p_n)$叫做翻支点。

3)如果$y=(y_1,y_2,\cdots,y_n)$满足约束条件，则置$x_i=y$。否则重复步骤1)和2)直到产生一个可行点y为止。

在步骤2)中，如果$\alpha>0$，则猴子会沿着指向支点的方向翻越，否则会沿着相反的方向翻越；猴群在选择翻过程的支点时，也可以采用$p_j'=\frac{1}{M-1}(\sum_{i=1}^{M}x_{ij}-x_{ij})$作为翻区间的支点，则用公式$y_j=p_j'+\alpha(p_j'-x_{ij})$或$y_j=x_{ij}+\alpha|p_j'-x_{ij}|$，$j=1,2,\cdots,n$，来更新猴群在翻过程中的翻越后位置。

2.6 引入单纯法优化策略

由于引入惯性步长，导致猴群算法的收敛速度降低，为了提高收敛速度，本文引入传统的单纯法搜索策略[9]，在翻过程结束后，选择K只位置较差的猴子，利用单纯法[9]搜索策略，优化这些猴子的位置，这样可以加快猴群搜索最优解的速度。单纯法的步骤如下(见图1)：

Step 1　计算所有搜索点对应的目标函数值，找到最优点 xg，次优点 xb，以及若干个较差的猴子的位置，取其中一个位置记为 xs，对应的目标函数值分别记为 $f(xg)$，$f(xb)$，$f(xs)$。

Step 2　计算最优点 xg 和次优点 xb 的中心位置：

$$xc=\frac{xg+xb}{2}$$

Step 3　执行反射操作。将 xs 依据中心点 xc 进行反射，得到反射点 xr，其反射系数通 α 常取 1：

$$xr=xc+\alpha(xc-xs)$$

Step 4　如果反射点 xr 的目标函数值高于最优点 xg，即 $f(xr)>f(xg)$，说明该反射方向正确，可进行扩张操作，得到扩张点 xe。通常扩张系数 γ 取 2：

$$xe=xc+\gamma(xr-xc)$$

如果扩张点 xe 的目标函数值高于最优点 xg，即 $f(xe)>f(xg)$，则用扩张点 xe 取代最差点 xs，否则用反射点 xr 取代最差点 xs。

Step 5　如果 $f(xr)<f(xs)$ 说明反射失败，执行压缩操作得到压缩点 xt，其中压缩系数 β 一般取 0.5：

$$xt=xc+\beta(xs-xc)$$

如果 $f(xt)>f(xs)$，则用压缩点 xt 取代最差点 xs。

Step 6　如果 $f(xs)<f(xr)<f(xg)$，进行收缩操作得到收缩点 xw，收缩系数与压缩系数取值相同：

$$xw=xc-\beta(xs-xc)$$

如果 $f(xw)>f(xs)$，则用收缩点 xw 取代最差点 xs，否则用 xr 取代最差点 xs。

爬过程结束后，根据单纯法搜索策略每次都能找到优于当前较差猴子所在位置的点，有时能找到优于最优点的位置，这样可以避免猴群一直在边缘爬。

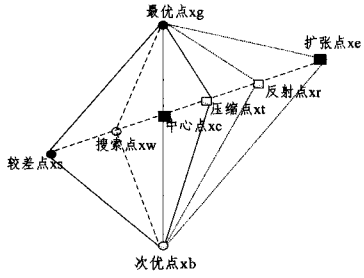

图 1　单纯法搜索的不同点

2.7　SMMA 实施过程和步骤如下：

Step 1　设定 SMMA 参数，种群大小 M、爬步长 a、视野 b 及其他参数，并为猴群生成初始位置。

Step 2　爬过程中根据伪梯度优化猴群的位置。

Step 3　在视野参数范围内搜索更优位置，并更新猴群位置到更优位置。

Step 4　望到更优解，则转到步骤 2；否则转到步骤 5。

Step 5　进行翻动作，并据此迫使猴群到新的搜索范围内。

Step 6　根据单纯法更新较差猴子的位置。

Step 7　检验是否满足结束条件，若满足，则算法结束；否则，转到步骤 2。

Step 8　输出最优目标函数解及对应的最优位置向量。

Step 9　程序结束。

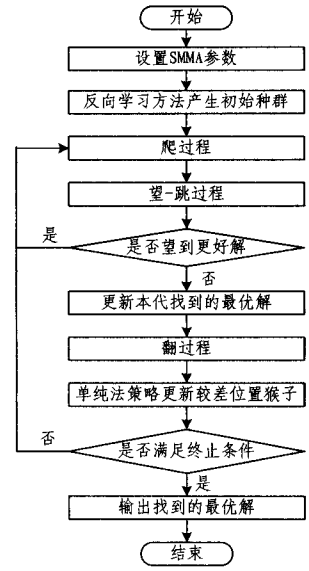

图 2　SMMA 流程图

3　仿真实验

3.1　实验操作平台

本实验的程序运行环境为：处理器：AMD Athlon(tm) II X4640，主频：3.01GHz，内存：3.00GB，操作系统：Windows XP，集成开发环境：Matlab 2012a。

3.2　实验结果

SMMA 算法参数设置如下：种群规模为 $M=5$，每次优化的猴子个数取 $K=2$，初始步长设置为 $a_{max}=0.1$，最终爬步长为 $a_{min}=10^{-13}$，视野大小为 $b=0.5$，翻区间为 $[c,d]=[-1,1]$，爬次数设置为 20，望次数为 2，最大迭代次数为 200。MA 参数的爬步长设置为 $a=1000$，爬次数设置为 1000，其他与 SMMA 相同。PSO 算法中设置的种群大小为 500，GA 算法利用的是 GATOOL 工具箱，设置的交概率为 0.95，变异概率为 0.05，种群大小为 1000。

对函数 $f_1(x)-f_{18}(x)$ 独立运行 20 次，表 1 列出了这些标准测试函数的搜索空间、维数以及最优值等参数。

表 2 列出了算法优化函数的最优值 Best、最差值 Worst、平均值 Mean、标准差 Std。对于每个函数，Best、Worst、Mean 反映了解的质量，而 Mean 反映在给定的测定次数下所能达到解的精度，Std 反映算法求解优化函数的稳定性和鲁棒性。从表 2 可看出，相比 GA、PSO、MA，SMMA 搜索最优值的能力更强，无论是在低维还是在高维，求解的最优值都更加接近理论值，PSO 算法在二维空间内有着较好的搜索能力，而 MA 在高维的搜索能力较强。对于函数 f_2，SMMA 和 PSO 的最优值都为 0，平均求解精度远远高于 MA 和 GA 算法。对于函数 f_2，PSO 求解精度最高，MA 的求解精度比 SMMA 高 2 个数量级。对于函数 f_3，SMMA 最优值精度为 e^{-23}，平均求解精度比 MA 高 21 个数量级。对于函数 f_4，SMMA 最优值精度为 e^{-11}，平均求解精度比 MA 高 9 个数量级。对于函数 f_5，SMMA 最优值精度为 e^{-21}，平均求解精度比 MA 高 19 个数量级，GA 和 PSO 在最大迭代次数内无法得到理想值。对于函数 f_6，SMMA 最优值精度为 e^{-12}，平均求解精度比 MA 高 9 个数量级。对于函数 f_7，MA 求得的平均最优值的结果最小，但 SMMA 的标准差最小，说明 SMMA 有着更强的稳

定性和鲁棒性。对于函数 f_8，SMMA 的求解精度达到 e^{-4}，平均求解精度比 MA 提高了 2 个数量级。对于函数 f_9、f_{11}，SMMA 的最优值为 0，平均求精精度远远高于 MA、PSO、GA 算法。对于函数 f_{10}，SMMA 最优值精度达到 e^{-16}，平均求解精度比 MA 提高了 12 个数量级。对于函数 f_{12}，SMMA 最优值精度达到 e^{-21}，平均求解精度比 MA 提高了 18 个数量级，PSO 算法在最大迭代次数内未能得到理想值。对于函数 f_{13}，SMMA 的最优值精度达到 e^{-12}，平均求解精度比 MA 提高了 10 个数量级。对于函数 f_{14}，SMMA 的最优值精度达到 e^{-3}，平均求解精度比 MA 提高了 1 个数量级。对于函数 f_{15}，PSO 和 MA 的平均求解精度比 SMMA 高 2 个数量级。对于函数 f_{16}，SMMA 最优值精度达到 e^{-47}，平均求解精度比 MA 提高了 33 个数量级。对于函数 f_{17}，SMMA 最优值为 -1，平均求解精度远远高于 MA 算法，而 GA 在最大迭代次数内没有找到理想值。对于函数 f_{18}，SMMA 最优值精度达到 e^{-8}，而 MA 和 PSO 在最大迭代次数内没有求解出来。

表 1　测试函数

函数	搜索空间	维数	最优值
$f_1(x)=0.5+(\sin\sqrt{x_1^2+x_2^2})^2-0.5)/(1+0.001(x_1^2+x_2^2))^2$	$[-10,10]$	2	0
$f_2(x)=4x_1^2-2.1x_1^4+(x_1^6/3)+x_1x_2-4x_2^2+4x_2^4$	$[-5,5]$	2	-1.0316285
$f_3(x)=x_i^2$	$[-100,100]$	30	0
$f_4(x)=\sum_{i=1}^{n}\lvert x_i\rvert+\prod_{i=1}^{n}\lvert x_i\rvert$	$[-100,100]$	30	0
$f_5(x)=\sum_{i=1}^{n}(\sum_{j=1}^{i}x_j)^2$	$[-100,100]$	30	0
$f_6(x)=\max\{\lvert x_i\rvert\}$	$[-10,10]$	30	0
$f_7(x)=\sum_{i=1}^{n-1}[100(x_{i+1}-x_i^2)^2+(1-x_i)^2]$	$[-30,30]$	30	0
$f_8(x)=\sum_{i=1}^{n}ix_i^4+\text{random}[0,1)$	$[-1.28,1.28]$	30	0
$f_9(x)=\sum_{i=1}^{n}[x_i^2-10\cos(2\pi x_i)+10]$	$[-5.12,5.12]$	30	0
$f_{10}(x)=20\exp(-0.2\sqrt{(1/n)\sum_{i=1}^{n}x_i^2})-\exp((1/n)\sum_{i=1}^{n}\cos(2\pi x_i))+20$	$[-32,32]$	30	0
$f_{11}(x)=(1/4000)\sum_{i=1}^{n}x_i^2-\prod_{i=1}^{n}\cos(x_i/\sqrt{i})+1$	$[-600,600]$	30	0
$f_{12}(x)=\sum_{i=1}^{n}x_i^2+\sum_{k=1}^{K}J_n^{2k}$，$J_n=(1/2)\sum_{i=1}^{n}ix_i$，$K=1,2,\cdots,20$	$[-100,100]$	30	0
$f_{13}(x)=-(\sum_{i=1}^{n}\lvert x_i\rvert)\exp(-\sum_{i=1}^{n}x_i^2)$	$[-2\pi,2\pi]$	30	0
$f_{14}(x)=(\pi/n)\{10\sin^2(\pi y_1)+[1+10\sin^2(\pi y_{n+1})]+(y_n-1)^2\}+\sum_{i=1}^{n}u(x_i,10,100,4)$，$y_i=1+(x_i+1)/4$，$u(x_i,a,k,m)=\begin{cases}k(x_i-a)^m, & x_i>a\\ 0, & -a<x_i<a\\ k(-x_i-a)^m, & x_i<-a\end{cases}$	$[-50,50]$	30	0
$f_{15}(x)=[\sum_{i=1}^{n}i\cos(i+(i+1)x)]\cdot[\sum_{i=1}^{n}i\cos(i+(i+1)y)]$	$[-10,10]$	5	-186.7309
$f_{16}(x)=\sum_{i}^{n}\lvert x_i\rvert^{i+1}$	$[-1,1]$	30	0
$f_{17}(x)=[e^{-\sum_{i=1}^{n}(x_i/\beta)^{2m}}-2e^{-\sum_{i=1}^{m}x_i^2}]\cdot\prod_{i=1}^{n}\cos^2x_i$，$m=5$，$\beta=15$	$[-10,10]$	30	-1
$f_{18}(x)=\sum_{i=1}^{n}(\sum_{k=0}^{k_{max}}[a^k\cos(2\pi b^k(x_i+0.5))])-n(\sum_{k=0}^{k_{max}}[a^k\cos(2\pi b^k\cdot 0.5)])$，$a=0.5$，$b=3$，$k_{max}=20$	$[-0.5,0.5]$	30	0

表 2　SMMA 与 GA、PSO、MA 算法在不同函数的测试结果

Function	Algorithm	Best	Worst	Mean	Std
f_1	GA	9.47e−6	4.36e−2	2.42e−2	2.21e−2
	PSO	0	0	0	0
	MA	1.05e−10	4.37e−2	4.36e−3	1.34e−2
	SMMA	0	0	0	0
f_2	GA	−1.031616420	−1.029885625	−1.031272194	4.26e−4
	PSO	−1.031628453	−1.031628453	−1.031628453	0
	MA	−1.031628448	−1.031628351	−1.031628414	2.99e−8
	SMMA	−1.031628434	−1.031623357	−1.031627551	1.24e−6
f_3	GA	0.71	2.70	1.67	0.65
	PSO	5.03e−8	3.28e−7	4.11e−4	7.33e−8
	MA	1.43e−4	0.091	0.017	0.016
	SMMA	5.73e−23	9.84e−23	8.18e−23	1.27e−23
f_4	GA	0.330	0.884	0.887	0.121
	PSO	1.71	8.14	3.95	1.87
	MA	5.66e−2	7.30e−2	6.62e−2	4.82e−3
	SMMA	3.82e−11	4.83e−11	4.37e−11	2.83e−12

续表

Function	Algorithm	Best	Worst	Mean	Std
f_5	GA	86.88	8.93e+2	4.10e+2	2.27e+2
	PSO	2.29e+2	3.34e+4	2.05e+3	4.40e+3
	MA	1.60e−2	5.54e−2	3.72e−2	1.01e−2
	SMMA	1.31e−21	4.50e−21	3.04e−21	9.24e−22
f_6	GA	0.99	1.80	1.27	0.22
	PSO	4.46	12.23	7.85	1.96
	MA	8.13e−3	9.37e−3	8.72e−3	4.19e−4
	SMMA	3.55e−12	4.85e−12	4.22e−12	3.64e−13
f_7	GA	47.58	2.47e+2	1.32e+2	52.50
	PSO	26.31	1.01e+2	44.34	23.54
	MA	20.70	25.23	21.30	0.953
	SMMA	27.74	28.27	28.08	0.118
f_8	GA	1.13e−2	4.66e−2	2.58e−2	9.52e−3
	PSO	0.143	1.21	7.53e−1	0.319
	MA	6.15e−3	2.48e−2	1.49e−2	5.47e−3
	SMMA	4.50e−6	7.84e−4	2.74e−4	2.39e−4
f_9	GA	1.01	2.36	1.65	0.46
	PSO	30.47	111.75	63.55	20.48
	MA	3.02	12.97	7.51	2.73
	SMMA	0	0	0	0
f_{10}	GA	0.28	2.75	0.70	0.53
	PSO	3.46e−3	3.16	9.45e−1	0.945
	MA	8.74e−3	8.79e−4	8.79e−4	8.79e−04
	SMMA	4.88e−16	9.87e−16	7.56e−16	1.47e−16
f_{11}	GA	20.64	7.64e+2	3.00e+2	2.08e+2
	PSO	1.04	4.13	1.48	0.15
	MA	1.05e−5	1.48e−2	1.24e−3	3.88e−3
	SMMA	0	0	0	0
f_{12}	GA	14.58	153.24	40.74	30.20
	PSO	1.08e+2	2.46e+3	6.63e+2	5.61e+2
	MA	4.03e−3	7.01e−3	5.53e−3	8.98e−4
	SMMA	8.88e−22	1.38e−21	1.06e−21	1.31e−22
f_{13}	GA	3.56e−12	3.69e−12	3.62e−12	3.70e−13
	PSO	1.26e−11	8.51e−7	8.00e−8	2.23e−7
	MA	3.65e−6	0.33	3.33e−2	0.10
	SMMA	4.79e−12	2.38e−11	7.37e−12	3.99e−12
f_{14}	GA	9.14e−3	0.14	0.04	0.04
	PSO	1.08	7.94	1.48	0.76
	MA	6.92e−6	0.213	5.28e−2	7.29e−2
	SMMA	5.27e−3	2.29e−2	7.44e−3	3.76e−3
f_{15}	GA	−1.86730e+2	−1.86633e+2	−1.86707e+2	3.15e−2
	PSO	−1.86730e+2	−1.86729e+2	−1.86730e+2	2.66e−4
	MA	−1.86730e+2	−1.86728e+2	−1.86730e+2	7.07e−4
	SMMA	−1.86730e+2	−1.86563e+2	−1.86697e+2	4.14e−2
f_{16}	GA	3.47e−9	2.17e−5	2.87e−6	5.13e−6
	PSO	6.48e−11	3.14e−8	6.55e−9	8.60e−9
	MA	2.98e−12	5.94e−10	6.94e−11	1.31e−10
	SMMA	2.74e−47	1.00e−43	1.68e−44	2.93e−44
f_{17}	GA	7.58e−191	9.12e−169	5.19e−170	0
	PSO	−0.959397	−2.65e−5	−0.346548	0.357796
	MA	−0.999837	−0.999753	−0.999789	2.57e−5
	SMMA	−1	−1	−1	0
f_{18}	GA	0.92	1.24	1.04	8.65e−2
	PSO	11.93	19.27	15.27	2.37
	MA	4.99	6.01	5.42	0.33
	SMMA	1.28e−8	1.81e−8	1.55e−8	1.61e−9

为了直观地反映出算法的寻优结果，图 3－图 20 给出 18 个测试函数的收敛曲线图，从图中可以看出，与 GA、MA、PSA 算法相比，本文提出的 SMMA 有着比较好的寻优能力，但收敛速度由于受到求解精度的影响，比 MA 慢。

MA 花费的时间主要是爬过程中每次迭代时目标函数的伪梯度的计算，即只需要计算当前位置的两个临近位置的目标函数值。在原始 MA 中爬步长的大小设置为定值，当设置爬步长为 0.001 时，爬次数一般为 1000，当爬步长设置为 0.0001 时，爬次数一般设置为 10000，因此要想得到更加精确的结果，必须减小爬步长的大小，同时爬次数相应增加，则算法需要花费大量的时间进行计算。而在 SMMA 中，爬步长是惯性的，即随着迭代次数的增加，爬步长逐渐减小，本文设置的爬次数为 20。由于 MA 中设置的爬次数是 SMMA 的 50 倍，导致 MA 花费的计算时间较长。表 3 列出了 SMMA 和 MA 对部分函数计算时间。从中可以看出，由于爬次数的减少，SMMA 花费的计算时间远远小于 MA。

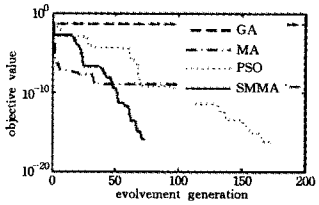

图 3　函数 f_1 的收敛图像

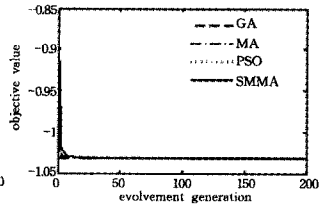

图 4　函数 f_2 的收敛图像

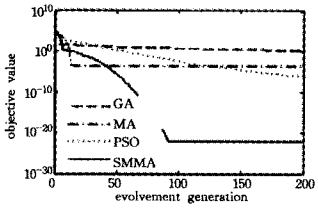

图 5　函数 f_3 的收敛曲线

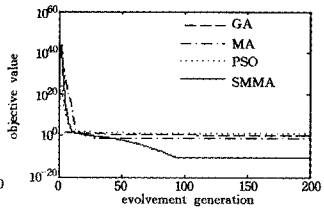

图 6　函数 f_4 的收敛曲线

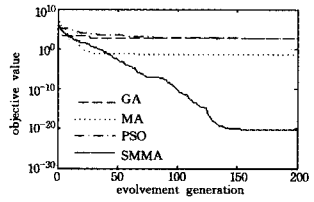

图 7　函数 f_5 的收敛曲线

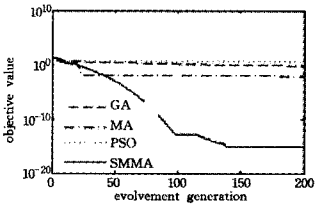

图 8　函数 f_6 的收敛曲线

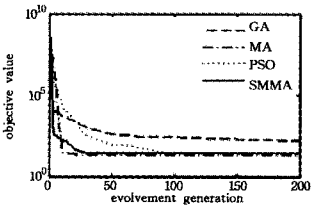

图 9　函数 f_7 的收敛曲线

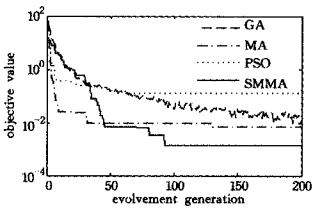

图 10　函数 f_8 的收敛曲线

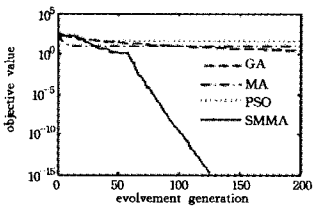

图 11　函数 f_9 的收敛曲线

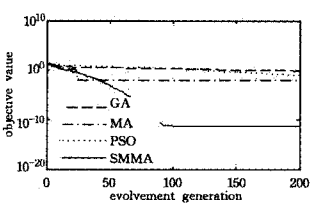

图 12　函数 f_{10} 的收敛曲线

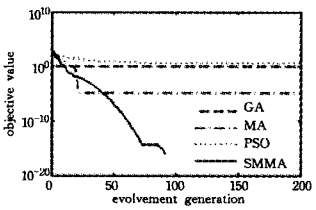

图 13　函数 f_{11} 的收敛曲线

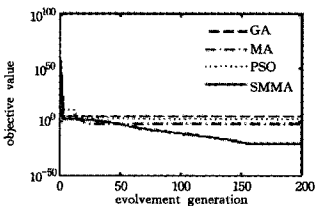

图 14　函数 f_{12} 的收敛曲线

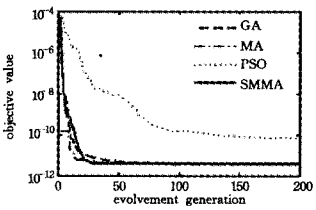

图 15　函数 f_{13} 的收敛曲线

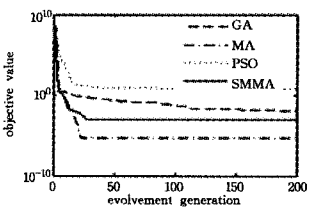

图 16　函数 f_{14} 的收敛曲线

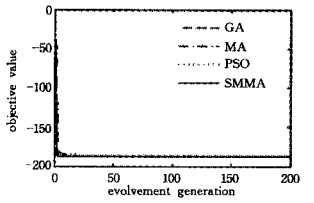

图 17　函数 f_{15} 的收敛曲线

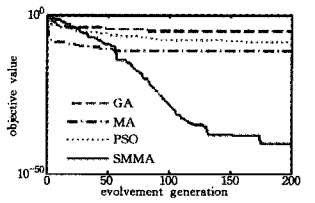

图 18　函数 f_{16} 的收敛曲线

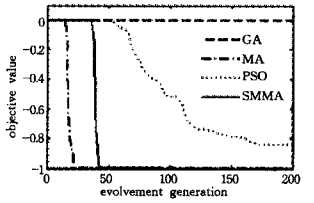

图 19　函数 f_{17} 的收敛曲线

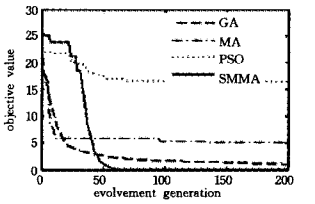

图 20　函数 f_{18} 的收敛曲线

为进一步展示 SMMA 算法的优势，本文将 SMMA 与 GPSO[10]、LPSO[11]、VPSO[11]、DMS-PSO[12]、CLPSO[13]、FIFS[14]、APSO[15]算法对相同测试函数的求解结果进行比较，结果如表 4 所列，其他算法的计算结果出自文献[15]。从表 4 可看出，对于函数 f_5、f_8、f_9、f_{10}、f_{11}，SMMA 的求解结果较好，有着更小的标准差。对于函数 f_7，SMMA 的标准差最小，可以看出其有着更强的稳定性和鲁棒性。

表 3　SMMA 和 MA 对部分函数计算时间对照表(单位：s)

Algorithm	f_2	f_3	f_7	f_8	f_9	f_{11}	f_{14}	f_{17}	f_{18}
SMMA	1.92	4.25	4.29	5.26	4.46	5.08	51.68	5.49	66.67
MA	16.21	25.09	27.57	63.00	32.45	58.61	1824.08	70.48	2417.55

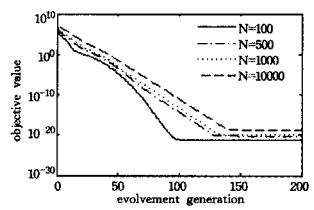

图 21　函数 f_3 在高维收敛曲线图

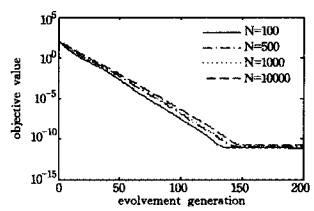

图 22　函数 f_6 在高维收敛曲线图

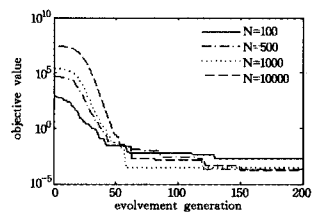

图 23　函数 f_8 在高维收敛曲线

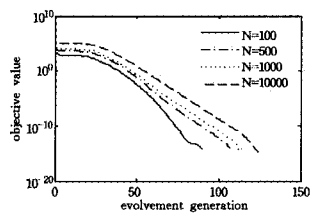

图 24　函数 f_9 在高维收敛曲线图

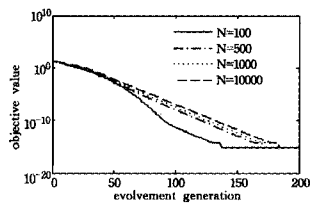

图 25　函数 f_{10} 在高维收敛曲线图

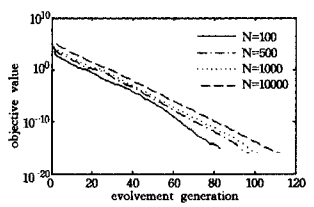

图 26　函数 f_{11} 在高维收敛曲线图

相比较其它搜索算法，MA 和 SMMA 的特点是，在求解优化函数的最优解时，不会随着优化函数维数的增加而陷入“维灾难”。图 22—图 27 给出了若干函数 f_3、f_6、f_8、f_9、f_{10}、f_{11} 在 N=100、500、1000、10000 维时的收敛曲线图。从图中

可看出，SMMA 在高维度下的求解结果和在低维度时的求解结果相差不大，只是在高维时，其收敛速度会受到较小的影响。可见，相对于其他算法，SMMA 在求解高维度的目标函数的最优值时有着更好的效果。

表 4　SMMA 与不同的改进的 PSO 算法在不同函数的测试结果

Function		GPSO	LPSO	VPSO	DMS-PSO	CLPSO	FIPS	APSO	SMMA
f_3	Mean	1.98e−53	4.77e−29	5.11e−38	3.85e−54	1.89e−19	3.21e−30	**1.45e−150**	8.18e−23
	Std	7.08e−53	1.13e−28	1.91e−37	1.75e−53	1.49e−19	3.60e−30	**5.73e−150**	1.27e−23
f_5	Mean	6.45e−2	18.60	1.44	47.5	395	0.77	1.00e−10	**3.04e−21**
	Std	9.46e−2	30.71	1.55	56.4	142	0.86	2.13e−10	**9.24e−22**
f_7	Mean	28.10	21.86	37.65	32.3	11	22.54	**2.84**	28.08
	Std	24.60	11.156	24.93	24.1	14.5	0.31	**3.27**	0.18
f_8	Mean	7.77e−3	1.49e−2	1.08e−2	1.1e−2	3.92e−3	2.55e−3	4.66e−3	**2.74e−04**
	Std	2.42e−3	5.66e−3	3.24e−3	3.94e−3	1.14e−3	6.25e−4	1.7e−3	**2.39e−04**
f_9	Mean	30.70	34.90	34.09	28.10	2.57e−11	29.98	5.8e−15	**0**
	Std	8.68	7.25	8.07	6.42	6.64e−11	10.92	1.01e−14	**0**
f_{10}	Mean	1.15e−14	1.85e−14	1.4e−14	8.52e−15	2.01e−12	7.69e−15	1.11e−14	**7.56e−16**
	Std	2.27e−15	4.80e−15	3.48e−15	1.79e−15	9.22e−13	9.33e−16	3.55e−15	**1.47e−16**
f_{11}	Mean	2.37e−2	1.10e−2	1.31e−2	1.31e−2	6.45e−13	9.04e−4	1.67e−2	**0**
	Std	2.57e−2	1.60e−2	1.35e−2	1.73e−2	2.07e−12	2.78e−3	2.41e−2	**0**
f_{14}	Mean	1.04e−2	2.18e−30	3.46e−3	**2.05e−32**	1.59e−21	1.22e−31	3.76e−31	7.44e−3
	Std	3.16e−2	5.14e−30	1.89e−2	**8.12e−33**	1.93e−21	4.85e−32	1.2e−30	3.76e−3

结合单纯法搜索策略，可以加快猴群寻找最优解的速度，本文选取部分函数，对未引入单纯法搜索策略和引入单纯法搜索策略进行测试，独立运行 20 次，表 5 给出了引入的单纯法搜索策略对 SMMA 求解优化问题寻优能力的影响。从中可以看出，对于函数 f_4，未引入单纯法搜索策略时，SMMA 也能找到全局最优解，但平均迭代次数增加了 19 次。对于函数 f_5，未引入单纯法搜索策略时，在最大迭代次数内，没有求出理想解的解；引入单纯法搜索策略后，最优解的精度达到 e^{-21}。对于函数 f_6，引入单纯法搜索策略后，SMMA 的平均迭代次数减少 34 次。对于函数 f_{16}，引入单纯法搜索策略后，平均迭代次数减少 23 次，平均求解精度也提高了 3 个数量级。这说明，引入单纯法搜索策略能够显著地提高 SMMA 的搜索能力，加快算法的收敛速度。

表 5　引入单纯法搜索策略对 SMMA 寻优能力的影响

函数	是否引入单纯法搜索策略	最优值出现的代数	最优值出现的平均代数	寻优结果平均值
f_4	是	92—96	93.3	4.37e−11
	否	99—131	112.3	3.51e−11
f_5	是	13—152	144.5	3.04e−21
	否	—	—	3.88
f_6	是	135—153	143.7	4.22e−12
	否	142—191	167.4	4.02e−12
f_{16}	是	141—170	157.8	1.68e−44
	否	165—191	181.1	1.09e−41

结束语　针对猴群算法求解全局优化问题时计算精度不高和花费大量计算时间等问题，本文通过引入惯性步长和结合传统单纯法搜索策略，提出一种混合算法，该算法能够减少大量的计算时间，加快猴群算法寻找最优解的速度。通过函数优化实验可以看出，算法是可行的，而且优于基本的猴群算法，不仅减少了迭代的次数，提高了进化速度，而且获得了更精确的函数值。

参 考 文 献

[1] Dorlgo M, Caro Di G. Ant colony optimization a new meta-heuristic[C]// Proceedings of the 1999 Congress on Evolutionary Computation. 1999:1470-1477

[2] Kennedy J, Eberhartr C. Particle swarm optimization[C]// Proc of IEEE Int Conf on Neural Networks. Perth: IEEE Piscataway, 1995:1942-1948

[3] Zhao Rui-qing, Tang Wan-sheng. Monkey algorithm for global numerical optimization[J]. Journal of Uncertain Systems, 2008, 2(3):164-175

[4] 王靖然，余贻鑫，曾沅. 离散猴群算法及其在输电网扩展规划中的应用[J]. 天津大学学报，2010，43(9)：798-803

[5] Yi Ting-hua, Li Hong-nan, Zhang Xu-dong. A modified monkey algorithm for optimalsensor placement in structural health monitoring[J]. Smart Materials and Structures, 2012, 21(10): 65-69

[6] 张佳佳，张亚平，孙济洲. 基于猴群算法的入侵检测技术[J]. 计算机工程，2011，14(37)：131-133

[7] Rahnama S, Tizhoosh H R, Salama M M A. Opposition-based differential evolution [J]. IEEE Trans on Evolutionary Computation, 2008, 12(1): 64-79

[8] 高卫峰，刘三阳，焦合华，等. 引入人工蜂群搜索算子的粒子群算法[J]. 控制与决策，2012，27(6)：833-838

[9] 张美恋. 基于遗传算法和单纯形法的混合优化算法[J]. 集美大学学报：自然科学版，2001，6(2)：106-110

[10] Shi Y, Eberhart R C. A modified particle swarm optimizer[C]// Proc. IEEE World Congr. Comput. Intell. 1998: 69-73

[11] Kennedy J, Mendes R. Population structure and particle swarm performance[C]// Proc of IEEE Congress on Evolutionary Computation. Honolulu, 2002: 1671-1676

[12] Liang J J, Suganthan P N. Dynamic multi-swarm particle swarm optimizer [C]// Proc of IEEE Swarm Intelligence Symposium. Pasadena, 2005: 124-129

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

[14] Mendes R, Kennedy J, Neves J. The fully informed particle swarm: Simpler maybe better[J]. IEEE Trans on Evolutionary Computation, 2004, 8(3): 204-210

[15] Zhan Z H, Zhang J, Li Y, et al. Adaptive particle swarm optimization[J]. IEEE Trans on Systems, Man, and Cybernetics, Part B, 2009, 39(6): 1362-1381