DOI：10.13195/j.cd.2010.07.11.liuym.008

第25卷 第7期 控 制 与 决 策 2010年7月
Vol. 25 No. 7 *Control and Decision* Jul. 2010

文章编号： 1001-0920(2010)07-0968-07

一种基于动态邻居和变异因子的粒子群算法

刘衍民[1,2]，赵庆祯[1]，隋常玲[2]，邵增珍[1]

(1. 山东师范大学 管理与经济学院，济南 250014；2. 遵义师范学院 数学系，贵州 遵义 563002)

摘 要： 提出一种基于动态邻居和变异因子的粒子群算法(DNMPSO). 在该算法中, 粒子的邻居是根据它的运行而动态变化. 每个粒子的学习机制分为自己的历史经验和所有邻居的经验两部分. 为了保证有效求解多峰问题, 在每一次迭代, 对当前解采用水平混合变异, 使每个粒子能更好地进行局部搜索, 提升粒子跳出局部最优解的能力. 通过与其他算法比较, 结果表明该算法求解多峰问题的能力最优.

关键词： 动态邻居；变异；粒子群；算法

中图分类号： TP301.6 **文献标识码：** A

Particle swarm optimizer based on dynamic neighborhood topology and mutation operator

LIU Yan-min[1,2], *ZHAO Qing-zhen*[1], *SUI Chang-ling*[2], *SHAO Zeng-zhen*[1]

(1. School of Management and Economics，Shandong Normal University，Ji'nan 250014，China；2. Department of Mathematics，Zunyi Normal College，Zunyi 563002，China. Correspondent：LIU Yan-min，E-mail：yanmin7831@163.com)

Abstract: A variant of particle swarm optimizer(PSO) based on dynamic neighborhood topology and mutation factor(DNMPSO) is proposed in this paper. In DNMPSO, the neighborhood of a particle are not fixed but dynamically changed, and the learning mechanism of a particle includes two parts，the historical best experience of the particle itself，and the experiences of its all neighbors. To effectively solve multimodal problems, the parallel hybrid mutation is used to work for local search, which improves the ability of escaping from local optima. The results demonstrate good performance of the DNMPSO algorithm in solving complicated multimodal problems when compared with other PSO algorithms.

Key words: Dynamic neighborhood topology；Mutation；Particle swarm optimizer；Algorithm

1 引 言

最优化问题是进化计算领域的一个研究热点. 随着优化问题的复杂化, 对其研究也提出了更高的要求. 粒子群算法(PSO)[1]是一个相对新的优化技术, 其中每个成员被叫作"粒子", 代表着一个潜在的可行解, 而食物的位置则被认为是全局最优解. 群体在n维解空间上搜寻全局最优解, 每个粒子都有一个适应函数值和速度来调整它自身的飞行方向.

PSO算法概念简单, 控制参数少, 寻优结果与初值无关, 具有一定的并行性. 自其被提出以来, 受到了学术界的广泛关注. 经验显示, PSO算法在解决大多数优化问题时表现十分出色, 但在解决复杂的多峰问题时极易陷入局部最优解. 因此, 本文提出一种改进的粒子群算法(DNMPSO), 该算法具有动态邻居拓扑结构和广泛的学习策略, 即每个粒子不是向它的邻居中表现最好的粒子学习, 而是向它的所有邻居学习. 同时, 引入水平混合变异, 使得每个粒子能够更好地进行局部搜索, 以提升粒子跳出局部最优解的能力.

2 相关研究

自1995年Kennedy等[2-12]提出PSO算法以来, 该算法已经吸引了许多学者的研究兴趣. 研究者几乎都致力于改良算法性能的研究, 得到了许多有实际意义的PSO算法的变型. 其中一种较为实用的PSO变型是将惯性权重ω引入原始算法中, 该算法的速度更新公式如下[2]:

$$\overline{v_i}^{t+1} =$$

收稿日期： 2009-07-10；**修回日期：** 2009-09-03.

基金项目： 山东省科技攻关项目(2009GG10001008)；贵州省教育厅社科项目(0705204)；遵义科技攻关项目([2008]21号).

作者简介： 刘衍民(1978−), 男, 黑龙江牡丹江人, 讲师, 博士, 从事运筹学理论、进化计算的研究；赵庆祯(1943−), 男, 山东利津人, 教授, 博士生导师, 从事运筹学理论、进化计算等研究.

$$\omega\overline{v_i}^{(t)} + \varphi_1 r_1(\overline{p_i} - \overline{x_i}^{(t)}) + \varphi_2 r_2(\overline{p_g} - \overline{x_i}^{(t)}). \quad (1)$$

其中: $\overline{p_i}$ 表示粒子 i 所经历的最好位置, 即粒子 i 所经历的具有最好适应值的位置, 称为个体最优位置, 也称为粒子的"认知部分"; $\overline{p_g}$ 表示群体中所有粒子所经历的最好位置, 也称为粒子的"社会部分". 此改进PSO算法, 通过引入惯性权重 ω 来权衡粒子的全局和局部搜索能力. 在文献[3]中, 通过对PSO收敛方式的分析, 提出了带有收缩因子的PSO变型, 引入的收缩因子保证了PSO的收敛并且提高了收敛速度. [4]提出一种完全信息粒子群, 粒子的所有邻居都参与位置更新. [5]提出了适应距离比例(Fitness-Distance-Ratio)的带有近邻合作的PSO算法(FDR-PSO). [6]安排每个粒子在不同的等级结构内(用树来表示这种结构), 运行好的粒子上升到树的顶端来影响更多的粒子, 而运行较差的粒子下降到树的根部. [7]提出了自适应逃逸PSO(AEPSO) 算法, 通过逃逸运动使微粒能够有效地进行全局和局部搜索, 具有较快的收敛速度, 但该算法存在不稳定性. [8]提出动态多群PSO算法, 该算法将整个种群分成一些子群, 用这些子群去优化各自的目标, 然后按照一定的规则再重新组群. [9]提出一种协作粒子群最优法(CPSO$-H_k$). 该算法用一维(1-D)群来独立地搜索每一维变量, 这些搜索结果按照事先规定的原则结合在一起, 以提升算法的运行效率. [10]将混沌序列混沌寻优引入粒子群优化算法, 利用混沌遍历性使得混沌优化算法跳出局部最优解, 但算法平均收敛精度不高. [11]提出了一种广义学习粒子群算法, 介绍了一种新的粒子学习机制. [12]提出了将有序(标准PSO)和无序(自适应寻优)进行适当分离, 充分发挥各自的优势. 在自适应寻优阶段, 通过在最优粒子邻域空间探寻更优化的解, 一旦新的优化解被发掘, 便利用标准PSO快速寻优.

以上所提到的PSO算法的改进, 对于提升算法的运行效率、跳出局部最优解具有一定作用, 但它们在求解复杂的多峰问题时, 对于规避早熟、收敛速度及其精度上存在不足.

3 基于动态邻居和变异因子的改进粒子群算法

3.1 动态邻居拓扑结构

在粒子群算法领域, 根据邻居拓扑结构的不同, 可分为局部版本粒子群算法(LPSO)和全局版本粒子群算法(GPSO), 这两种算法的拓扑结构是固定的. 这种固定的邻居拓扑结构导致每个粒子仅有少数的学习样本, 极大地降低了群的多样性. 因此, 为了保证每个粒子都有机会向更多的粒子学习, 避免陷入局部最优解以及出现早熟现象, 本文提出根据粒子自身的运行(粒子的适应函数值)进行动态组建邻居的策略. 这种策略使得粒子的学习样本变得多样化, 能够预防粒子陷入局部最优解, 从而使粒子向着全局最优位置收敛.

在DNMPSO算法中, 每个粒子邻居的选择规则为粒子间的欧氏距离. 根据这一原则, 当前粒子 i 的邻居可由下式确定:

$$\begin{cases} L_i(t) = \{d_{ij}(t) | d_{ij}(t) = \|x_i(t) - x_j(t)\|\}, \\ \qquad j \neq i, j \in p_s; \\ \text{neighbor}_i(t) = \arg(\min(\text{sort}(L_i(t)), n)). \end{cases} \quad (2)$$

其中: $L_i(t)$表示在迭代时刻 t, 当前粒子 i 与种群中其他粒子的欧氏距离集合; p_s 表示种群规模; $\text{neighbor}_i(t)$表示在迭代时刻 t, 当前粒子 i 的所有邻居的集合; n 表示粒子 i 拥有的邻居个数, 它的取值通常为粒子规模的1/4～1/3, 在DNMPSO算法中 $n = p_s/3$, 对于 n 的取值是将来进行研究的重点; sort(A)表示对集合 A 中元素从小到大进行排序. 在算法运行中, 如果一个粒子连续 T 代, 自己最好的位置没有得到提升, 则需对粒子 i 重新选择邻居. 经过反复实验, 当 T=7 时, 算法取得了最好的运行结果, 因此DNMPSO算法中参数 T 的取值为7.

3.2 粒子的学习样本选择

在GPSO算法中, 粒子的每一维速度由 $\overline{p_i}$ 和 $\overline{p_g}$ 调整. 而在LPSO算法中, 粒子的每一维速度由 $\overline{p_i}$ 和 $\overline{p_{\text{neighbori}}}$ (表示粒子 i 的邻居中所有粒子所经历的最好位置, 也称为"社会部分") 共同调整. 也就是说, 在这两种PSO算法中, 对于"社会部分"的学习, 粒子的每一维都是向同一个粒子的相应维数学习. 然而文献[9]中指出, 这种学习策略会造成"two steps forward, one step back"现象. 因此, 在DNMPSO算法中, 当更新粒子速度时, 每个粒子的学习样本不再是它的邻居中表现最好的一个粒子, 而是它邻居中所有的粒子. 图1给出了GPSO算法和DNMPSO算法在种群规模为20个粒子, 运行6 000次迭代时, 粒子学习样本的多样性对比图. 例如, 坐标点(5 000, 20)表示在第5 000次迭代时, 第20个粒子将成为粒子的学习样本. 可以看出, 如果粒子速度的每一维都向它邻居中表现最好的同一个粒子学习, 则在算法迭代过程中, 仅有少数的粒子被选作学习样本(例如GPSO). 而如果一个粒子速度的每一维向它邻居中所有粒子的相应维数学习, 则会有更多的粒子成为学习样本(例如DNMPSO), 这极大地增加了群的多样性, 扩大了粒子的探索空间. 因此, DNMPSO算法的速度更新公式可描述为

$$\overline{v_i}^{t+1} = \omega\overline{v_i}^{(t)} + \varphi_1 r_1(\overline{p_g} - \overline{x_i}^{(t)}) + \varphi_2 r_2(\overline{p_{\text{bin(i)}}} - \overline{x_i}^{(t)}); \quad (3)$$

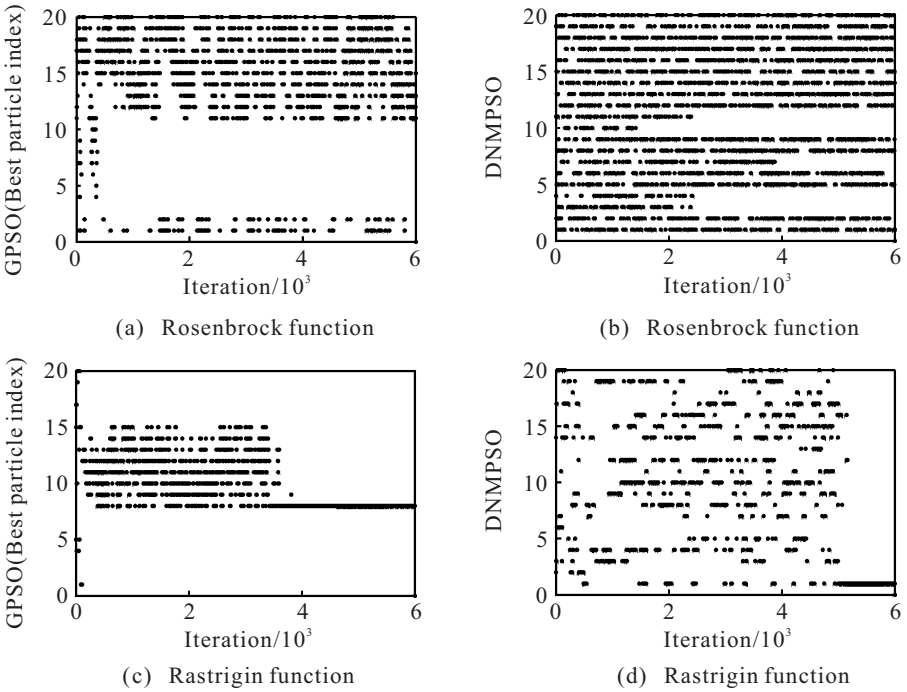


图 1 学习样本与迭代次数关系图

$$p^d_{\text{bin(i)}} = \arg\left\{\max\left[\frac{\text{Fitness}(p_j)-\text{Fitness}(p_i)}{|p_{jd}-x_{id}|}\right]\right\}. \quad (4)$$

其中: $d \in (1,2,\cdots,n)$, $i=1,2,\cdots,p_s$, $j \in \text{neighbor}_i$; $\overline{p_{\text{bin}(i)}}$ 表示粒子 i 的每一维学习样本; neighbor_i 表示粒子 i 的邻居构成的集合; $p^d_{\text{bin}(i)}$ 表示 $\overline{p_{\text{bin}(i)}}$ 的第 d 维, 它被叫作 Fitness-Distance-Ratio[5]. 在 DNMPSO 算法中, $\omega = 0.729$, $\varphi_1 = \varphi_2 = 1.494\,45$. 这种粒子的学习策略被称为广泛学习策略.

3.3 变异因子

为了使每个粒子能够更好地进行局部搜索, 以提升粒子跳出局部最优解的能力. 在 DNMPSO 算法中引入水平混合变异操作.

首先, 为每个粒子指定变异能力 (mc_i), 决定一个粒子是否进行变异. 以经验给出

$$mc_i = 0.05 + 0.45\frac{\left(\exp\left(\frac{5(i-1)}{\text{ps}-1}\right)\right)-1}{\exp(5)-1} \quad (5)$$

来确定每个粒子的变异能力, 其中 i 表示当前粒子.

其次, 每个粒子按照下面的流程选择变异方式:

For $i = 1 : p_s$
  If ceil(mc_i+rand-1)=1
    If rand$\leqslant p_u$
      pos(i,d)=(1+rand)$\times$ pos(i,d)
    Else
      pos(i,d)=Gaussian(σ)$\times$ pos(i,d)
    End
  End
End

其中: Gaussian (σ) 表示返回一个标准差为 σ 的符合高斯分布的随机数, rand 返回一个 (0,1) 之间的均匀分布的随机数; ceil(A) 表示返回一个不超过 A 的最大整数; p_u 称作均匀分布变异概率 (变异因子), 它可以是固定的, 也可以是自适应变化的. 高斯分布变异率为 $1-p_u$. 在文献 [13] 中给出了 3 种变异因子:

1) 固定常数变异. 在 [0,1] 之间均匀取值,通常取值为: $0, 0.1, 0.2, \cdots, 1$.

2) 递减变异因子. 有 3 种经常用到的函数, 即线性函数、指数函数和 Sigmoid 函数, 分别如下:

$$p_u(t) = 1 - t/\text{gen}; \quad (6)$$

$$p_u(t) = 1 - (\exp(t\log 2/\text{gen}) - 1); \quad (7)$$

$$\begin{cases} f(r,t) = 1/(1+\exp(-rt)), \\ p_u(t) = 1 - f(t - \text{gen}/2, r). \end{cases} \quad (8)$$

其中: t 为当前迭代次数, gen 为总的迭代次数.

3) 自适应变异因子. 这种变异因子产生 [0.4, 0.7] 之间均匀分布数, 适合大多数问题.

图 2 给出了最大迭代次数是 2 000 的不同的变异因子比较图. 为了检验哪种变异因子更适合本文提出的算法, 在动态邻居和广泛学习策略的基础上, 通过添加各种变异因子, 在检测函数 Sphere, Rosenbrock, Griewanks, Ackley, Rastrigin-noncont 和 Rastrigin 上运行 DNMPSO 算法. 每个检测函数进行 1 000 次迭代, 由于不同检测函数最优值的量纲不同, 需对所有的适应函数值采用下式:

$$X = \frac{x_{ij} - \mu_{ij}}{\sigma_{ij}} \quad (9)$$

进行标准化处理, 以消除量纲的影响, 同时将标准化的数据按照文献 [4] 方法进行合并. 其中: i 为检测函数, j 为不同的变异因子.

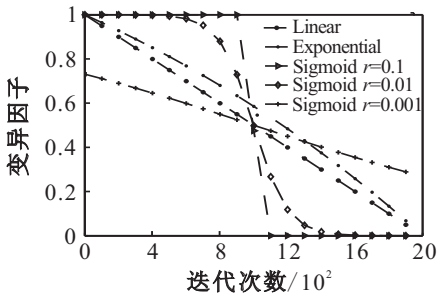


图 2　5 种变异因子

表 1 给出了第 1 000 次迭代时的 X 值(X_{end}). 其中 F 表示固定常数变异因子, Linear 表示线性递减变异因子, Exponent 表示指数递减变异因子, Sigm 表示 Sigmoid 函数递减变异因子, Self-adapt.表示自适应变异因子, No 表示没有变异因子. 可以看出, 线性递减变异因子对算法的提升起到了明显的作用. 因此, 在 DNMPSO 算法中, 采用线性递减变异因子. DNMPSO 算法的伪代码如下:

Begin
  Initialize particles' position, velocity, p_{best_i}, g_{best}.
  $V_{\max} = 0.25(X_{\max} - X_{\min})$
  Initialize particles' mutate capacity(Mc) in terms of Eq.(5)
  staynum=zeros(1, p_s)
  Initialize p_u
  while(fitcount< Max_{FEs}) && (k <**iteration**)
    For each particle ($i = 1 : p_s$)
      If staynum(i) ⩾7
        staynum(i) = 0.
        update particle i's neighbor in terms of Eq.(2)
      End
    End For
    For each particle ($i = 1 : p_s$)
      Construct the neighborhood of the particle i in terms of Eq.(2)
      Construct $\overline{p_{\text{bin}(i)}}$ in terms of Eq.(4)
      For each dimension
        Updating particles' velocity and position in terms of Eq.(3)
        If ceil $(mc_i + \text{rand} - 1) = 1$
          If rand⩽ p_u
            $x(i, d) = (1 + \text{rand}) \times \text{pos}(i, d)$.
          Else
            $x(i, d) = \text{Gaussian}(\sigma) \times \text{pos}(i, d)$.
          End
        End
      End For
      If $x_i \in [X_{\min}, X_{\max}]$
        Compute the fittness value of x_i,
        Update particles' p_{best_i} and g_{best}.
        Update staynum.
      End
      If fitness(x_i) <fitness(p_{best_i})
        $p_{\text{best}_i} = x_i$, $p_{\text{bestval}_i} = \text{fitness}(x_i)$.
      End
    End For
  End While
  Output result.
End Begin

表 1　不同变异因子的标准化函数值

p_u	X_{end}	p_u	X_{end}	p_u	X_{end}
F0.0	3.4e-04	F0.6	7.2e-04	Modular	3.3e-04
F0.1	2.1e-04	F0.7	5.5e-04	Exponent	9.1e-04
F0.2	3.2e-04	F0.8	6.2e-04	Sigm0.1	7.2e-05
F0.3	4.7e-04	F0.9	7.4e-04	Sigm0.01	1.1e-04
F0.4	4.3e-04	F1.0	1.8e-04	Sigm0.001	1.3e-04
F0.5	3.6e-04	Linear	2.2e-05	Self-adapt.	6.5e-04
				No	9.6e-04

4　实验仿真及其分析

4.1　检测函数

为了测试 DNMPSO 算法, 选择 2 个单峰和 4 个多峰 Benchmark 函数. 其表达式见文献 [11]. 表 2 和表 3 给出了这些函数特征. 为了增加这些检测函数的优化难度, 除了 Sphere 函数外, 其他所有的函数都进行旋转. 对于函数旋转方法, 具体参见文献 [14]. 由于对检测函数进行了旋转, 在粒子位置初始化时, 采用无偏初始化.

表 2　非旋转的 Benchmark 函数

Function name	n	Search space	x^*	$f(x^*)$
Sphere (f_1)	30	$[-100, 100]^{30}$	[0,0,···,0]	0
Rosenbrock(f_2)	30	$[-2.048,2.048]^{30}$	[1,1,···,1]	0
Ackley(f_3)	30	$[-32.768,32.768]^{30}$	[0,0,···,0]	0
Griewanks(f_4)	30	$[-600,600]^{30}$	[0,0,···,0]	0
Rastrigin(f_5)	30	$[-5.12,5.12]^{30}$	[0,0,···,0]	0

表 3　旋转的 Benchmark 函数

Function name	n	Search space	x^*	$f(x^*)$
Ackley(f_6)	30	$[-32.768,32.768]^{30}$	[0,0,···,0]	0
Griewanks(f_7)	30	$[-600,600]^{30}$	[0,0,···,0]	0
Rastrigin(f_8)	30	$[-5.12,5.12]^{30}$	[0,0,···,0]	0
Rastrigin -noncont(f_9)	30	$[-5.12,5.12]^{30}$	[0,0,···,0]	0

4.2　PSO算法的参数设置

与本文提出的 DNMPSO 算法进行比较的 6 种 PSO 算法如下:

1) 带收缩因子的局部 PSO 算法(CF-LPSO)[3];

2) 带收缩因子的全局 PSO 算法(CF-GPSO)[3];

3) 基于适应距离比率的 PSO 算法(FDR-PSO)[5];

4) 基于完全信息的 PSO 算法(FIPS)[4];

5) 基于广泛学习的PSO算法(CLPSO)[11];

6) 基于共协作的PSO算法(CPSO-Hk)[9], k =6.

为了使不同的算法具有可比性, 实验的相关设置如下: 所有的PSO算法的种群规模为30个粒子,每个检测函数独立运行30次, 对于非旋转的检测函数每次运行3×10^4函数评价(Fitness Evaluations), 而对于旋转的检测函数每次运行6×10^4函数评价.

4.3 实验结果及分析

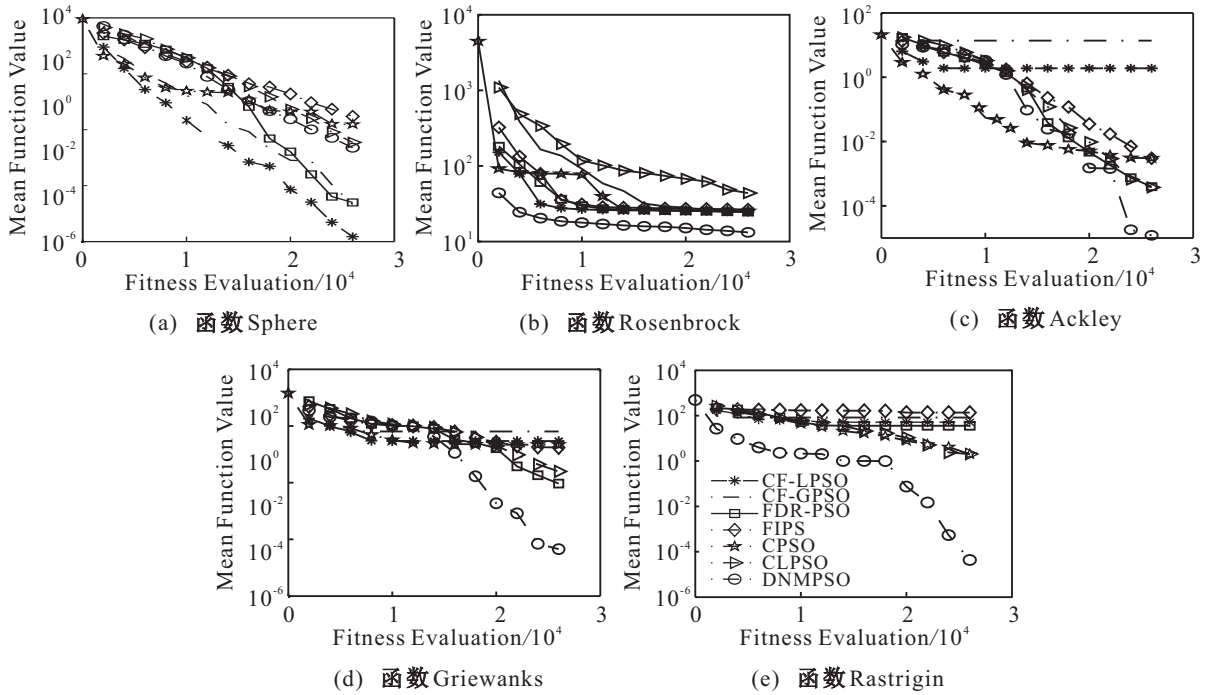
表4和表5分别给出了检测函数在3×10^4函数评价和6×10^4函数评价后的均值和0.95的置信区间, 其中最好运行的结果用黑体字标出. 采用Wilcoxon秩和检验, 对DNMPSO算法所得结果与其他6种PSO算法中最好的结果进行检验, 以验证DNMPSO算法所得结果与其他6种PSO算法所得结果的差别是否有统计学意义. 检验结果列在表4和表5的底部. 表6和表7给出了各种算法的运行时间, 计算方法为: 在型号为ThinkPad-SL 400的电脑上应用Matlab7.0.1.24704(R14)Service Pack 1软件中的(tic, toc)命令. 图3和图4给出了PSO算法收敛特征曲线图, 其中各图图注与图3(e)相同.

表 4 非旋转函数在(3×10^4 FEs)函数评价后的均值和置信区间

Algorithm	Sphere	Rosenbrock	Ackley	Griewanks	Rastrigin
CF-LPSO	**4.550 2e-017±1.320 1e-017**	2.421 8e+001± 1.123 6e+001	1.897 8e+000± 1.137 4e+000	4.661 0e-002± 3.763 1e-002	5.173 8e+001± 2.542 6e+001
CF-GPSO	4.754 6e-014± 2.342 1e-014	2.407 8e+001± 1.745 6e+001	1.376 9e+001± 2.816 8e+001	3.288 6e-001± 1.456 1e-001	8.258 1e+001± 1.634 6e+001
FDR-PSO	9.061 4e-017± 2.195e-017	2.322 7e+001± 1.021 1e+001	1.379 6e-004± 2.151 1e-004	1.376 2e-006± 2.673 1e-006	3.581 9e+001± 2.564 3e+001
FIPS	8.429 8e-006± 2.433e-006	2.612 6e+001± 1.118 4e+001	9.472 3e-004± 3.163 4e-004	1.232 9e-002± 3.953 4e-002	1.348 1e+002± 1.366 4e+002
CPSO	3.058 3e-006± 3.214 3e-006	2.489 9e+000± 1.136 1e+000	2.544 4e-004± 1.243 2e-004	2.215 7e-002± 1.563 2e-002	1.993 8e+000± 1.783 1e+000
CLPSO	1.188 0e-008± 1.651 3e-008	3.954 2e+001± 1.241 2e+001	5.583 1e-005± 1.354 1e-005	1.779 7e-006± 1.954 2e-006	9.969 9e-001± 2.234 1e-001
DNMPSO	5.347 8e-008± 1.212 4e-008	**1.246 2e+001± 1.311 3e+001**	**1.195 7e-005± 2.542 1e-005**	**1.114 7e-011± 1.453 2e-011**	**2.684 4e-005± 1.567 3e-005**
result	0	1	1	1	1

表 5 旋转函数在(6×10^4 FEs)函数评价后的均值和置信区间

Algorithm	Ackley	Griewanks	Rastrigin	Rastrigin-noncont
CF-LPSO	1.385 6e-013±1.536 7e-013	3.446 7e-002± 1.432 5e-002	1.193 9e+001± 1.545 3e+001	5.300 0e+000± 1.245 4e+000
CF-GPSO	7.105 4e-015±3.534 6e-015	9.835 4e-002± 2.657 2e-002	1.094 5e+001± 1.434 1e+001	7.001 2e+000± 2.432 5e+000
FDR-PSO	1.155 1e+000±2.474 4e+000	9.599 9e-002± 4.234 3e-002	9.949 6e+000± 2.017 1e+000	6.000 1e+000± 2.566 5e+000
FIPS	3.552 7e-015±4.542 5e-015	6.777 7e-002± 2.746 7e-002	9.895 4e+000± 5.254 2e+000	7.065 0e+000± 1.567 4e+000
CPSO	1.155 1e+000±1.636 6e+000	3.161 1e-002± 5.543 2e-002	1.022 8e+001± 1.541 2e+001	1.800e+001± 3.056 5e+001
CLPSO	3.552 7e-015±4.245 4e-015	5.368 0e-002± 3.424 6e-002	4.031 0e+000± 4.563 2e+000	6.040 5e+000± 1.456 2e+000
DNMPSO	**3.452 7e-015±1.834 2e-015**	**3.103 6e-002± 1.784 2e-002**	**3.979 8e+000± 3.635 5e+000**	**5.033 0e+000± 1.734 4e+000**
result	0	1	0	1



(a) 函数Sphere (b) 函数Rosenbrock (c) 函数Ackley

(d) 函数Griewanks (e) 函数Rastrigin

图 3 非旋转检测函数收敛特征图

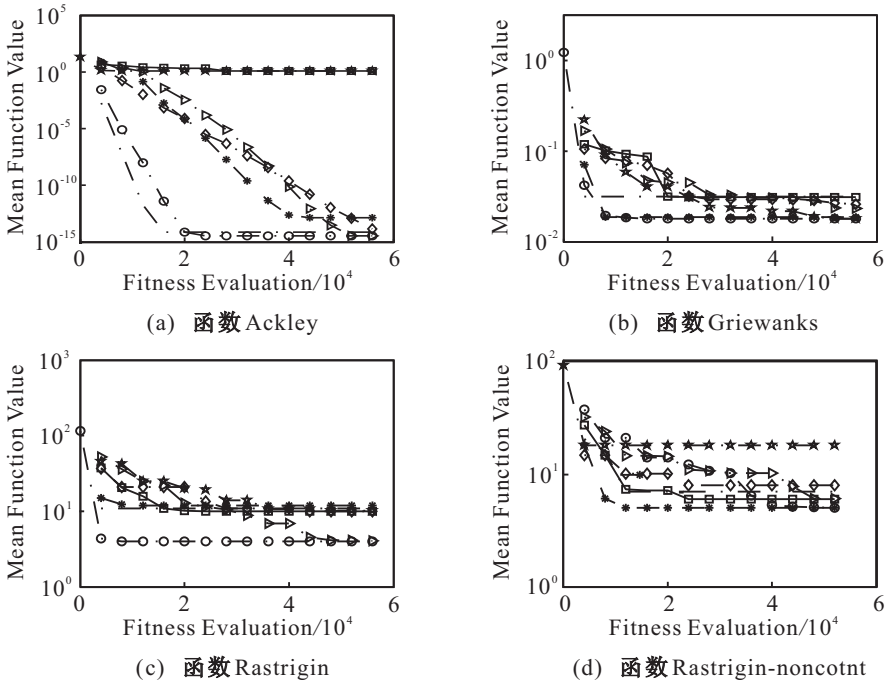


图 4 旋转检测函数收敛特征图

表 6 非旋转检测函数计算时间 (3×10^4 FEs) s

Algorithm	Sphere	Rosenbrock	Ackley	Griewanks	Rastrigin
CF-LPSO	**7**	12	19	29	50
CF-GPSO	8	10	18	27	46
FDR-PSO	9	9.8	20	35	45
FIPS	9	8.9	16	40	70
CPSO	10	8.5	24	43	54
CLPSO	9	7.9	17	39	71
DNMPSO	7.8	10	20	30	51

表 7 旋转检测函数计算时间 (6×10^4 FEs) s

Algorithm	Ackley	Griewanks	Rastrigin	rastrigin-noncont
CF-LPSO	431	262	421	789
CF-GPSO	409	286	452	546
FDR-PSO	387	268	399	654
FIPS	374	363	488	653
CPSO	452	544	515	468
CLPSO	441	357	652	531
DNMPSO	385	246	404	499

从图 3 可以看出, 对于所有的非旋转的检测函数, 除了 Sphere 函数外, DNMPSO 算法明显优于其他 6 种算法, 尤其对 Griewanks 和 Rastrigin 函数, 在搜索过程的后期 DNMPSO 算法表现出快速的收敛特性, 这是由于对粒子位置采用了水平混合变异操作. 使得每个粒子能够更好地进行局部搜索, 提升了粒子跳出局部最优解的能力. 而 CLPSO, CPSO 和 FDR-PSO 也表现出求解多峰问题的能力, 这主要是由他们的学习策略不同于 CF-GPSO 和 CF-LPSO, 使得群体的多样性增加, 进而增强了跳出局部最优解的能力.

由图 4 可看出, 对于 Ackley 函数, 当没有进行旋转时, DNMPSO 算法运行明显地优于其他算法. 然而当函数旋转后, CF-GPSO, CLPSO, FIPS, CLPSO 与 DNMPSO 算法几乎有相同的收敛特性. 但 CF-GPSO 和 DNMPSO 在前期有较快的收敛速度, 而 CPSO 和 FDR-PSO 出现了早熟收敛. 对于 Griekwanks 函数, 在函数旋转的情况下, DNMPSO 算法和 CF-LPSO 算法的运行优于其他算法. 总体来说, 所有算法的收敛性几乎没有太大的差别. 旋转的 Rastrigin 函数表现出与 Griewanks 函数相似的运行, 除了 CLPSO 算法, 其他算法在 1.5×10^4 次函数评价后, 几乎都处于停滞状态, 而 CLPSO 算法有进一步提升的可能. 对于 Rastrigin-noncont 函数, CPSO 算法出现了早熟收敛, 在 0.3×10^4 次函数评价后, 粒子的位置几乎没有得到任何提升; 而 CF-GPSO, CLPSO, FIPS, CF-LPSO 与 CLPSO 几乎有相同的收敛结果. DNMPSO 算法从迭代开始到迭代结束一直向全局最优位置收敛, 表现出跳出局部最优解的能力.

从表 4 和表 5 也可看出, DNMPSO 算法在函数 $f_2 \sim f_8$ 和 f_{10} 上都取得了最好的运行结果. 从 Wilcoxon 和检验结果来分析, 除了检测函数 f_6 和 f_8, DNMPSO 算法所得结果与其他 6 种 PSO 算法所得结果的差别有统计学意义.

表 6 和表 7 说明, DNMPSO 算法在非旋转检测函数上的运行时间与其他算法在同一数量级, 而对于旋转的检测函数, DNMPSO 算法的计算时间要少于其他算法.

5 结 论

本文提出了求解多峰问题的一种改进的粒子群算法 (DNMPSO). 该算法摒弃了基本 PSO 算法中固

定邻居拓扑结构的缺陷(极易陷入局部最优解), 动态地调整粒子的邻居拓扑结构, 极大地增加了群体的多样性. 当粒子速度更新时, 改变了经典PSO算法中仅向自己的历史经验和群体中最好的一个粒子的经验学习的方法, 而是向自己的历史经验和它的邻居中所有粒子学习. 从实验分析结果可以看出, DNMPSO算法相对于其他PSO算法表现出了求解多峰问题的优势, 但是根据"没有免费的午餐理论(No free lunch)"[15], 并不能说DNMPSO算法是最好的算法. 在现实世界, 所要解决问题的曲线形状(Shape of the Fitness Landscape)未知时, 选择一种能较好地解决多峰问题的算法是一种明智的选择, 因为这种算法也能够解决单峰问题. 对于本文提出的算法, 还可以应用更多的检测函数(例如复合函数等)来检测算法规避早熟的能力, 也可进一步分析邻居个数对算法收敛性的影响.

参考文献(References)

[1] Kennedy J, Eberhart R. Particle swarm optimization[C]. Proc of IEEE Int Conf on Neural Networks. Perth, 1995:1942-1948.

[2] Shi Y, Eberhart R C. A modified particle swarm optimizer[C]. Proc of IEEE Congress Evolutionary Computation. 1998: 69-73.

[3] Clerc M, Kennedy J. The particle swarm-explosion, stability, and convergence in a multidimensional complex space[J]. IEEE Trans on Evolutionary Computation, 2002: 6(1): 58-73.

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

[5] Peram T, Veeramachaneni K, Mohan C K. Fitness-distance-ratio based particle swarm optimization[C]. Proc of Swarm Intelligence Symposium. 2003: 174-181.

[6] Janson S, Middendorf M. A hierarchical particle swarm optimizer and its adaptive variant[J]. IEEE Systems, Man and Cybernetics - Part B, 2005, 3(6): 1272-1282.

[7] 赫然, 王永吉, 王青, 等. 一种改进的自适应逃逸微粒群算法及实验分析[J]. 软件学报, 2005, 16(12): 2036-2044.
(He R, Wang Y J, Wang Q, et al. An improved particle swarm optimization based on self-adaptive escape velocity[J]. J of Software, 2005, 16(12): 2036-2044.)

[8] Zhao S Z, Liang J J, Suganthan P N, et al. Dynamic multi-swarm particle swarm optimizer with local search for large scale global optimization[C]. 2008 IEEE Swarm Intelligence Symposium. 2008: 3845-3852.

[9] Van den Bergh F, Engelbrecht A P. A cooperative approach to particle swarm optimization[J]. IEEE Trans on Evolutionary Computation, 2004, 8(2): 225-239.

[10] 孟红记, 郑鹏, 梅国辉, 等. 基于混沌序列的粒子群优化算法[J]. 控制与决策, 2006, 21(3): 263-266.
(Meng H J, Zheng P, Mei G H, et al. Particle swarm optimization algorithm based on chaotic series[J]. Control and Decision, 2006, 21(3): 263-266.)

[11] 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.

[12] 罗辞勇, 陈民铀, 韩力. 适应性粒子群优化算法Ⅱ[J]. 控制与决策, 2009, 24(8): 1135-1144.
(Luo C Y, Chen M Y, Han L. Adaptive particle swarm optimizationⅡ[J]. Control and Decision, 2009, 24(8): 1135-1144.)

[13] Parsopoulos E, Vrahatis M N. Parameter selection and adaptation in unified particle swarm optimization[J]. Mathematical and Computer Modeling, 2007, 46: 198-213.

[14] Mohais A S, Mendes R, et al. Neighborhood re-structuring in particle swarm optimization[C]. Australian Conf on Artificial Intelligence. 2005, 3809: 776-785.

[15] Wolpert D H, Macready W G. No free lunch theorems for optimization[J]. IEEE Trans on Evolutionary Computation, 1997, 1(1): 67-82.