文章编号: 1001-0920(2017)12-2127-10 DOI: 10.13195/j.kzyjc.2016.1402

基于频繁覆盖策略的随机漂移粒子群优化算法

方 伟†, 周建宏

(江南大学 物联网工程学院，江苏 无锡 214122)

摘 要: 为了进一步提升随机漂移粒子群优化(RDPSO)算法的全局搜索能力、收敛速度以及在高维问题上的优化能力,提出一种基于频繁覆盖策略的RDPSO(FC-RDPSO)算法,并采用概率统计方法和蒙特卡罗方法分析频繁覆盖策略的可行性. 在CEC'2013RPO的测试函数上将FC-RDPSO算法与多种优化算法进行对比,实验结果表明所提算法在收敛速度和全局搜索能力上表现出了突出的性能;在一组被广泛使用的大规模全局优化测试函数上的实验结果表明,FC-RDPSO算法在高维问题上同样表现出了较强的优化能力.

关键词: 频繁覆盖；粒子群优化算法；全局优化；大规模优化问题；高维问题

中图分类号: TP18 **文献标志码:** A

Random drift particle swarm optimization with frequent coverage strategy

FANG Wei†, ZHOU Jian-hong

(School of IoT Engineering，Jiangnan University，Wuxi 214122，China)

Abstract: In order to further improve the exploration ability, convergence speed and optimization ability in high dimensional problems of random dritf particle swarm optimization(RDPSO) algorithm. The frequently coverage RDPSO(FC-RDPSO) algorithm based on the strategy of frequent coverage is proposed. The probability statistical method and Monte Carlo method are used to analyze the feasibility of the frequently coverage strategy. The results show that the proposed FC-RDPSO algorithm has better performance than some classical and state-of-the-art variants of PSO algorithms on the test suite for CEC'2013 special session on real-parameter optimization(RPO) benchmark functions. Results on a set of widely used large-scale global optimization(LSGO) benchmark functions demonstrate that the proposed FC-RDPSO algorithm also has strong optimization ability on a high-dimensional problem.

Keywords: frequent coverage；particle swarm optimization；global optimization；large-scale optimization problem；high-dimensional problem

0 引 言

粒子群优化(PSO)算法是由Kennedy等[1-2]提出的一种群体智能优化算法. 由于PSO算法思想简单、实现容易、参数少,同时具有深刻的智能背景等优点,一经提出便受到各领域学者的关注,并取得了丰硕的研究成果[3-9]. 然而,PSO算法易出现早熟收敛而导致优化性能降低,为此,研究人员在算法的性能改进方面展开了较多的研究,从不同的角度提出了算法的改进方案. 其中一种改进方法是通过设计粒子不同的邻域结构来改善群体的信息交流机制,以提高群体的搜索能力,如文献[10]中研究的环形、冯若依曼拓扑方式、全连接方式等结构. Liang等[11]引入多种群提出了动态多种群粒子群算法(DMS-PSO),而后又提出了用于求解多峰函数的深入学习粒子群算法(CLPSO)[12]; Qu等[13]提出了基于距离的局部邻域结构的小生境PSO算法. 群体多样性降低是PSO算法在进化过程中无法回避的问题,也被认为是引起早熟收敛的原因之一. 文献[14-15]通过设置群体多样性的上下限来控制群体的发散和收缩行为;文献[16]基于邻居节点的搜索行为提出了多样性增强机制,以平衡群体的搜索和勘探能力. 遗传算法中的交叉[17]、变异[18]等进化操作方式也常被应用于PSO算法,以提高算法的优化能力. 不同的概率采样方式可以改变粒子的搜索方式,并起到性能改进的目的,如高斯

收稿日期: 2016-11-04；**修回日期:** 2017-02-03.

基金项目: 国家自然科学基金项目(61673194, 61105128)；江苏省自然科学基金项目(BK20131106)；中国博士后科学基金项目(2014M560390)；江南大学自主科研计划重点项目(JUSRP51410B)；江苏省重点研发计划项目(BE2017630).

作者简介: 方伟(1980–), 男, 副教授, 从事计算智能等研究；周建宏(1990–), 男, 硕士生, 从事粒子群优化算法的研究.

†通讯作者. E-mail: fangwei@jiangnan.edu.cn

采样[19]、Levy采样[20]、双指数采样[21]等. 在PSO算法中改进或引入更为智能的学习机制来影响群体的学习方法,以提升算法的优化性能也是一直以来的研究热点. Zhan等在文献[22]中提出了自适应学习的PSO算法; Cheng等[23]引入竞争学习策略提出了竞争粒子群算法(CSO),随后又提出了社会学习型粒子群优化算法(SL-PSO)[24]; Sun等[25-26]以PSO算法的个体学习和社会学习等智能优化机制为基础,结合金属导体中自由电子的定向漂移运动和随机无规则热运动方式,提出了一种具有较强全局搜索能力的随机漂移粒子群(RDPSO)算法.

为进一步提升RDPSO算法在复杂优化问题中的求解效果,并扩展其在复杂高维问题中的优化能力,本文设计一种频繁覆盖策略用于RDPSO算法,提出基于频繁覆盖策略的随机漂移粒子群优化(FC-RDPSO)算法,利用概率统计方法与蒙特卡洛方法分析频繁覆盖策略的可行性. 将FC-RDPSO算法在CEC'2013RPO的20个复杂测试函数[27]上进行实验,并将实验结果与多个经典PSO改进算法以及最新的多种PSO改进算法进行对比,实验结果表明本文所提算法在较少的计算时间内具有较快的收敛速度和更高的寻优精度. 本文为了评估算法在高维优化问题上的优化性能,在CEC'2008 LSGO[28]和CEC'2010 LSGO[29]的15个测试函数上进行实验,结果表明FC-RDPSO算法在高维问题上具有良好的可扩展性.

1 随机漂移粒子群优化(RDPSO)算法

由物理学中的电磁学知识可知,金属导体内的自由电子一般情况下是做杂乱无章的随机热运动. 当金属导体受到外在磁场作用时,金属导体内的自由电子会在磁场的作用下发生定向漂移运动,同时还存在无规则的随机热运动,两种运动的整体态势使其势能逐渐变小,这类似于优化问题中搜索最优解使得目标函数值逐渐变小的过程. RDPSO算法正是受自由电子运动模型的启发而提出的[28],自由电子的定向漂移运动类似粒子的局部搜索,自由电子的无规则随机热运动则类似于粒子的全局搜索. 假设在RDPSO算法中,第$t+1$代的第i个粒子在第j维的定向漂移运动速度表示为$Vd_{i,j}^{t+1}$,随机热运动的速度表示为$Vr_{i,j}^{t+1}$,粒子运动速度表示为$V_{i,j}^{t+1}$,则第$t+1$代粒子速度的更新公式为

$$V_{i,j}^{t+1}=Vd_{i,j}^{t+1}+Vr_{i,j}^{t+1}. \tag{1}$$

利用PSO算法中粒子趋向局部位置的学习机理,定向漂移运动速度$\mathrm{V}d_{i,j}^{t+1}$的定义如下:

$$Vd_{i,j}^{t+1}=$$

$$c_1\cdot r_{i,j}^t\cdot(P_{i,j}^t-X_{i,j}^t)+c_2\cdot R_{i,j}^t\cdot(G_j^t-X_{i,j}^t). \tag{2}$$

其中: c_1和c_2为常数,均设置为1.732; $r_{i,j}^t$和$R_{i,j}^t$为在(0,1)区间内均匀分布的随机数; $P_{i,j}^t$为第i个粒子到目前为止找到的历史最优位置(pbest); G_j^t为群体所找到的最优位置. 式(2)可以等价地表示为

$$Vd_{i,j}^{t+1}=\beta\cdot(p_{i,j}^t-X_{i,j}^t), \tag{3}$$

$$\beta=c_1\cdot r_{i,j}^t+c_2\cdot R_{i,j}^t, \tag{4}$$

$$p_{i,j}^t=\frac{c_1\cdot r_{i,j}^t\cdot P_{i,j}^t+c_2\cdot R_{i,j}^t\cdot G_j^t}{c_1\cdot r_{i,j}^t+c_2\cdot R_{i,j}^t}. \tag{5}$$

其中: β称为漂移系数, $p_{i,j}^t$称为吸引子. 根据自由电子的运动模型,假设RDPSO算法中粒子的无规则随机热运动遵循Maxwell速率分布率,从而得到无规则随机热运动速度$Vr_{i,j}^{t+1}$的表示形式如下:

$$Vr_{i,j}^{t+1}=\delta_{i,j}^t\cdot\varphi_{i,j}^t, \tag{6}$$

其中$\varphi_{i,j}^t$是高斯分布函数,是高斯分布的标准差,定义为

$$\delta_{i,j}^t=\alpha\cdot|M_j^t-X_{i,j}^t|. \tag{7}$$

α为热敏系数, M_j^t为当前种群中所有粒子个体的历史最优位置平均值(mbest),即

$$M_j^t=\frac{1}{N}\sum_{i=1}^{N}P_{i,j}^t, \tag{8}$$

N为种群规模. 因此,式(6)可以改写为

$$Vr_{i,j}^{t+1}=\alpha\cdot|M_j^t-X_{i,j}^t|\cdot\varphi_{i,j}^t, \tag{9}$$

$$\beta=c_1r_{i,j}^t+c_2R_{i,j}^t. \tag{10}$$

综上,RDPSO算法中粒子速度和位置的更新公式为

$$V_{i,j}^{t+1}=\alpha\cdot|M_j^t-X_{i,j}^t|\cdot\varphi_{i,j}^t+\beta\cdot(p_{i,j}^t-X_{i,j}^t), \tag{11}$$

$$X_{i,j}^{t+1}=X_{i,j}^t+V_{i,j}^{t+1}. \tag{12}$$

RDPSO算法的流程如下:

Step 1: 设定群体大小,初始化粒子位置和速度;

Step 2: 计算漂移系数β,根据式(8)计算群体平均最优位置;

Step 3: 计算目标函数值,更新全局最优位置和粒子的局部最优位置;

Step 4: 根据式(11)和(12)更新粒子的速度和位置;

Step 5: 判断是否满足结束条件,若满足,则输出结果并结束,否则转入Step 2.

2 基于频繁覆盖策略的随机漂移粒子群优化(FC-RDPSO)算法

2.1 频繁覆盖策略思想

粒子群优化算法及一些经典的变体在有些优化问题上不能取得良好的优化效果[12],特别是在不可

分优化问题上的寻优性能较弱. 究其原因,是这些优化问题中有些决策变量的相互依赖性,变量与变量耦合关系较强[8]. 这些算法搜索过程往往是同时改变粒子所有维的值,然后评估,再根据其评估值判断粒子是否向较优的位置移动. 但是,适应值的评价只能代表粒子整体解的质量,粒子适应值变优也只能说明粒子的部分维度向最优方向移动,当变量间耦合关系较强时,寻优就会出现较多的“前进两步,倒退一步”的现象. 进一步研究还发现,随着优化问题维数的增长,包括PSO算法与RDPSO算法在内的群体智能优化算法都面临着优化性能急剧衰减的现象. 引起该现象的原因之一也在于问题规模的增加导致变量之间的耦合关系增强,使问题的复杂度增加. 针对该现象,Frans等[8]将协同演化(CC)方法[30]引入PSO算法来提高算法的优化能力,即将优化问题分解为若干组变量不重合的独立优化问题,每个独立优化问题利用PSO算法各自独立优化,各个独立优化问题的解在优化过程中进行组合以评价解的质量. CC方法的核心思想在于变量的分组方式,这在很大程度上影响算法的优化效果. 针对CC方法中的变量分组方式,有学者提出了固定分组、随机分组和基于学习机制分组等多种方式[31],但是这些分组方式的决定与使用需要较多的目标函数评价, 存在效率较低的缺点,且难以得到完全理想的分组结果. 同时,这类分组方法的应用在可分和部分可分问题上能够取得较好的优化结果,但对于完全不可分的问题优化效果较差.

除此之外,随着优化问题维数的增长,算法优化性能会衰减严重的另一个重要的原因是,维数增长导致算法搜索空间呈现指数式增长. 下面从定量的角度进行分析. 边长为$2r$的D维超立方体的体积为

$$V(C_D(r)) = V(C_D(1)) \cdot r^D, \tag{13}$$

其中$V(C_D(1)) = 2^D$. 该超立方体内切球的体积为

$$V(B_D(r)) = V(B_D(1)) \cdot r^D, \tag{14}$$

其中$V(B_D(1)) = \pi^{\frac{D}{2}} \Big/ \Gamma\Big(1+\frac{D}{2}\Big)$, $\Gamma(x)$为伽马函数. 因此,有

$$\lim_{D\to\infty} \frac{V(B_D(r))}{V(C_D(r))} = \lim_{D\to\infty} \frac{\pi^{\frac{D}{2}}}{\Gamma\Big(1+\frac{D}{2}\Big)\cdot 2^D} = 0. \tag{15}$$

从式(13)可以看出,搜索空间体积会随着维数D的增长呈现指数式增长. 而由式(15)可知,随着维数的增大,搜索空间体积会向各个角落扩散,使得搜索空间中心变得越来越空,这种现象导致粒子群易收敛到某个“角落”,从而给群体寻优带来极大的困难.

综上分析可知,提高算法在复杂问题上的优化性能的关键在于降低决策变量间的耦合度、减小搜索空间的体积,本文采用与CC方法类似的分而治之的思想,提出一种频繁覆盖(FC)策略,并用具有较强全局优化能力的RDPSO算法以应对复杂优化问题. 具体来讲,FC策略不需要对决策变量提前进行分组,而是在每次迭代开始时随机选取一部分决策变量,构成新的搜索子空间来减小搜索空间的体积,然后给子优化器RDPSO分配一定的评估费用,利用子优化器搜索得到每个粒子在此子空间上的最优位置,迭代最后更新所有粒子至各自的最优位置,重复以上过程直至达到算法终止条件. 通过不断地迭代,产生了大量体积不同的搜索子空间,由于决策变量是随机选取的,导致子空间之间可能是包含的,可能是交叉的,可能是相互隔离的,这些子空间相互频繁地交叉叠加,覆盖到了整个搜索空间,进而子优化器RDPSO搜索得到全局最优值.

综上可知,FC-RDPSO优化过程中每次迭代针对其中一部分决策变量进行优化,而保留其他的变量不变,以减少粒子寻优过程中出现的“前进两步,倒退一步”现象,不需要花费大量的评估费用来进行分组,算法效率较高,适用于所有的高维复杂优化问题. 表1对比了CC框架与FC框架的特性.

表1　CC框架与FC框架

	相同特性	不同特性
CC框架	分而治之、降维	先分组后优化,检测耦合变量,算法复杂度高,效率低,适用可分优化问题
FC框架		不需要分组,连续优化,算法简单,效率高,适用所有优化问题

2.2　FC-RDPSO算法

FC-RDPSO算法对问题的优化是通过随机频繁覆盖策略与RDPSO算法有机结合后实现的,在优化问题中,每个决策变量对优化都能产生一定的影响,所有算法要保证在有限的迭代次数内对所有的变量进行优化求解. 每次迭代随机选择的决策变量的个数称之为覆盖规模,用FCSCALE表示. 为了更好地分析本文所提FC策略的可行性,首先给出覆盖率的定义,然后用概率统计方法和蒙特卡罗方法对FC策略进行分析.

定义1　覆盖率. 经过有限次的覆盖后,被选择到的不同决策变量的个数与问题变量规模的比值,用p_{FC}表示.

首先,采用概率统计知识,计算第n次覆盖后将

D维决策变量池中全部决策变量覆盖到的覆盖率. 为了方便计算,假设算法中随机选择变量的个数M为定值,并且与覆盖规模FCSCALE相等,记$A_{n,i}$为第n次覆盖后第i个决策变量未被覆盖到的情形,则第n次覆盖后至少有一个没有被覆盖到的概率为$P(\bigcup_{i=1}^{D} A_{n,i})$. 根据容斥原理,全部决策变量都覆盖到的覆盖率为

$$p_{\rm FC}=1-P(\bigcup_{i=1}^{D} A_{n,i}). \tag{16}$$

由概率的一般加法公式可得

$$P(\bigcup_{i=1}^{D} A_{n,i})=\sum_{1\leqslant i\leqslant D} P(A_{n,i})-\sum_{1\leqslant i\leqslant j\leqslant D} P(A_{n,i}\bigcap A_{n,j})+\sum_{1\leqslant i\leqslant j\leqslant k\leqslant D} P(A_{n,i}\bigcap A_{n,j}\bigcap A_{n,k})+\cdots. \tag{17}$$

由于

$$P(\bigcup_{S=1}^{D-M} A_{n,i_S})=\left(\begin{bmatrix}D-S\\M\end{bmatrix}\Big/\begin{bmatrix}D\\M\end{bmatrix}\right)^n, \tag{18}$$

可得

$$p_{\rm FC}=1-\sum_{S=1}^{D-M}(-1)^{S+1}\begin{bmatrix}D\\S\end{bmatrix}\left(\begin{bmatrix}D-S\\M\end{bmatrix}\Big/\begin{bmatrix}D\\M\end{bmatrix}\right)^n. \tag{19}$$

若给定$D=50, M=10, n=40$,则$p_{\rm FC}=0.993\,3$. 这表明,对于50维决策变量空间,以覆盖规模为10进行频繁覆盖,大约只需覆盖40次,覆盖率即可达到99.33%.

其次,采用蒙特卡罗方法在计算机上模拟频繁覆盖的过程,以不同的维数、不同的覆盖规模模拟1 000次覆盖过程,取其覆盖率平均值,从而得到在不同覆盖规模下覆盖率与覆盖次数关系的曲线,如图1所示.

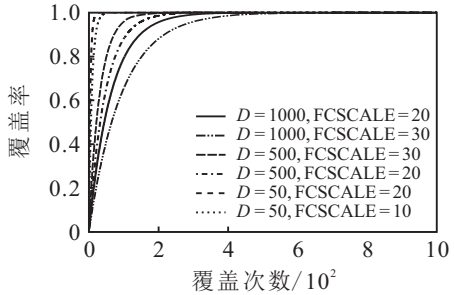

图1 利用蒙特卡洛方法得到的覆盖次数与覆盖率关系曲线

从图1可以看出,在50维的决策变量空间中,以覆盖规模为10进行频繁覆盖,随着覆盖次数的增多,覆盖率直线(点线)上升,大约覆盖50次后,覆盖率达到100%;对于1 000维决策变量空间中,覆盖规模为20的情况下,随着覆盖次数的增多,覆盖率曲线(实线)也上升很快,覆盖约600次后,覆盖率达到100%. FC-RDPSO算法的伪代码如下所示.

Step 1: pop ← and (popsize,dim)//初始化种群

Step 2: best ← fitness (pop)//评估种群

Step 3: **while** FEs < max FEs

Step 4: M ← rand$i$ (FCSCALE)//产生随机整数

Step 5: index ← cover (M, dim)//构成搜索子空间

Step 6: coverpop ← pop[:, index]得到搜索子群

Step 7: newpop ← RDPSO(coverpop)//子优化器优化

Step 8: pop[:, index] ← pbest (newpop)//更新个体位置

Step 9: bestarrow min(best,fitness (pop)//评估并搜索最优值

Step 10: **end while**

3 实验结果与分析

3.1 FC-RDPSO算法在CEC'2013RPO测试函数上的实验

3.1.1 参数设置

为了检验本文所提出的FC-RDPSO算法的寻优性能和收敛效率,首先选择CEC'2013RPO的20个复杂函数[27]作为算法的测试函数,搜索空间都为$[-100,100]^{50}$,全局最优值都为0. $F_1\sim F_6$是单峰可分与单峰不可分函数,$F_7\sim F_{20}$是多峰不可分与可分函数,这些函数在对称性、连续性、旋转性以及可微性等特征上也互有区别. 选择PSO算法、带压缩因子的PSO算法(PSO-Co)[7]、DMS-PSO算法、QPSO算法以及RDPSO算法、SL-PSO算法和CSO算法[23]3个最新的改进算法作为对比算法. 所有算法的群体规模为30,根据相应参考文献,采用的算法参数设置如表2所示. 为了减少实验的随机性,各个算法在每个测试函数上均随机独立运行30次. 每次实验都以最大评估次数(max FEs)作为终止条件,max FEs $=9\times10^4$. 本文实验所用的计算机配置如下: Intel Core i3-4710CPU, 3.7 GHz, 4 G内存, windows7专业版,64位操作系统.

表2 各算法参数设置

算法	参数设置	算法提出年份
PSO	$w:0.9\to0.4, c_1,c_2=2.0$	1998
PSO-Co	$\mathcal{X}=0.729, \sum c_i=4.1$	2002
QPSO	$\beta:1\to0.5$	2004
DMS-PSO	$w:0.9\to0.2, m=3, R=15$	2005
RDPSO	$\alpha:0.9\to0.3, c_1,c_2=2.0$	2010
CSO	$\varphi=0.05$	2014
SL-PSO	$\alpha=0.05, \beta=0.1$	2015
FC-RDPSO	FCSCALE $=10, \alpha=0.95$	2015

3.1.2 实验结果及分析

本文提出的FC-RDPSO算法以及对比算法在CEC'2013RPO测试函数上的实验结果由表3给出,每个测试函数的实验结果由两行数据构成,第1行表示算法在此测试函数上独立运行30次所得的目标函数值的平均值,第2行表示30个目标函数值的标准差. 表4给出了表3数据的t检验结果,其中检验水平$\alpha = 0.05$,符号"+"表示两种算法对比,第1种比第2种有明显的优势;符号"="表示两种算法优化效果差不多,无明显差别;符号"−"表示第1种算法优化结果比第2种算法有明显的劣势;gm表示符号"+"的总数与符号"−"的总数的差值,表明了第1种算法比第2种算法优化性能强的一个量值. 图2给出了各算法优化各个测试函数的收敛曲线.

表 3　各算法在CEC'2013RPO的20个测试函数上的实验结果

50-D	PSO	PSO-Co	DMS-PSO	QPSO	RDPSO	CSO	SL-PSO	FC-RDPSO
F_1	3.09e-05	2.51e-11	3.07e-01	5.27e-09	1.17e-02	5.71e+01	1.21e+01	**0.00e+00**
	7.12e-05	1.35e-10	4.58e-01	1.33e-08	2.90e-02	9.30e+01	5.36e+01	**0.00e+00**
F_2	1.09e+08	2.78e+07	5.19e+07	3.26e+07	5.60e+07	5.23e+07	4.60e+07	**4.23e+06**
	7.05e+07	3.78e+07	1.25e+07	1.43e+07	3.37e+07	1.57e+07	1.68e+07	**1.58e+06**
F_3	7.49e+10	1.92e+10	2.55e+09	3.33e+09	3.91e+09	2.81e+10	2.60e+10	**5.63e+08**
	5.93e+10	2.98e+10	1.30e+09	2.19e+09	3.41e+09	1.49e+10	1.87e+10	**5.15e+08**
F_4	1.19e+05	6.09e+04	**3.57e+04**	3.70e+04	4.56e+04	1.25e+05	1.55e+05	4.02e+04
	3.03e+04	1.49e+04	**5.54e+03**	1.05e+04	1.02e+04	2.45e+04	3.96e+04	9.70e+03
F_5	1.32e-02	1.96e-11	7.97e-01	1.83e-05	7.01e-02	2.46e+02	9.87e+01	**6.85e-24**
	1.37e-02	3.85e-11	6.65e-01	3.21e-05	7.89e-02	3.35e+02	9.97e+01	**9.37e-24**
F_6	6.31e+01	5.32e+01	**4.77e+01**	5.03e+01	5.64e+01	1.55e+02	9.72e+01	5.47e+01
	3.28e+01	1.81e+01	**1.32e+00**	1.44e+01	1.83e+01	4.98e+01	4.94e+01	2.66e+01
F_7	2.83e+02	2.05e+02	**6.23e+01**	8.77e+01	9.76e+01	9.82e+01	1.02e+02	1.23e+02
	5.26e+01	5.95e+01	**1.06e+01**	2.64e+01	3.35e+01	1.98e+01	1.71e+01	2.68e+01
F_8	2.13e+01	2.13e+01	2.12e+01	2.12e+01	2.12e+01	2.12e+01	2.12e+01	**2.11e+01**
	4.84e-02	3.35e-02	4.77e-02	4.48e-02	3.07e-02	5.30e-02	3.85e-02	**3.63e-02**
F_9	7.37e+01	5.89e+01	4.80e+01	5.24e+01	4.50e+01	3.81e+01	**3.68e+01**	4.08e+01
	1.04e+01	6.36e+00	5.12e+00	1.18e+01	1.14e+01	3.59e+00	**4.12e+00**	5.83e+00
F_{10}	4.84e+02	4.16e+00	6.72e+01	1.81e+01	8.45e+01	3.34e+02	2.42e+02	**2.24e-01**
	5.91e+02	1.56e+01	3.58e+01	9.63e+00	5.21e+01	1.47e+02	1.39e+02	**1.44e-01**
F_{11}	9.98e+01	2.46e+02	1.32e+02	7.10e+01	7.00e+01	1.58e+02	1.35e+02	**2.55e-01**
	2.00e+01	6.48e+01	4.93e+01	1.44e+01	1.39e+01	4.51e+01	4.21e+01	**4.06e-01**
F_{12}	5.01e+02	3.48e+02	2.77e+02	2.49e+02	2.69e+02	1.82e+02	**1.61e+02**	2.19e+02
	1.38e+02	1.28e+02	2.22e+01	1.02e+02	1.02e+02	3.65e+01	**4.41e+01**	8.71e+01
F_{13}	5.46e+02	5.07e+02	3.43e+02	3.66e+02	3.51e+02	3.21e+02	**3.16e+02**	3.35e+02
	5.52e+01	1.06e+02	1.97e+01	5.86e+01	6.07e+01	6.91e+01	**4.81e+01**	4.74e+01
F_{14}	3.36e+03	5.03e+03	5.12e+03	1.05e+04	1.21e+04	3.62e+03	3.71e+03	**7.54e+01**
	5.88e+02	7.35e+02	1.19e+03	2.79e+03	1.68e+03	6.98e+02	6.90e+02	**6.31e+01**
F_{15}	1.60e+04	1.40e+04	1.06e+04	1.45e+04	1.47e+04	**6.53e+03**	7.07e+03	1.12e+04
	4.33e+02	3.41e+03	5.84e+02	3.62e+02	5.14e+02	**9.60e+02**	9.07e+02	1.15e+03
F_{16}	4.97e+00	4.99e+00	2.37e+00	3.67e+00	3.77e+00	5.75e-01	**5.67e-01**	2.92e+00
	4.59e-01	3.93e-01	3.70e-01	2.85e-01	2.93e-01	2.19e-01	**3.85e-01**	3.23e-01
F_{17}	2.35e+02	3.13e+02	3.19e+02	2.18e+02	1.77e+02	1.88e+02	1.81e+02	**5.25e+01**
	8.23e+01	7.29e+01	6.16e+01	8.80e+01	6.46e+01	2.56e+01	2.68e+01	**4.48e-01**
F_{18}	6.43e+02	6.22e+02	4.39e+02	4.37e+02	4.36e+02	2.74e+02	**2.45e+02**	3.25e+02
	6.62e+01	1.44e+02	2.54e+01	1.94e+01	2.49e+01	5.76e+01	**4.16e+01**	5.30e+01
F_{19}	1.66e+01	3.35e+01	2.49e+01	8.21e+00	9.88e+00	1.23e+02	4.53e+01	**2.60e+00**
	6.77e+00	1.16e+01	6.49e+00	2.18e+00	4.85e+00	8.24e+01	2.65e+01	**5.55e-01**
F_{20}	2.39e+01	2.35e+01	**2.10e+01**	2.24e+01	2.27e+01	2.23e+01	2.32e+01	2.25e+01
	2.96e-01	8.55e-01	**6.12e-01**	3.44e-01	5.29e-01	1.13e+00	1.42e+00	9.45e-01

表 4 各种算法仿真实验的t检验结果

t-test	F_1	F_2	F_3	F_4	F_5	F_6	F_7	F_8	F_9	F_{10}	F_{11}	F_{12}	F_{13}	F_{14}	F_{15}	F_{16}	F_{17}	F_{18}	F_{19}	F_{20}	+/−/=/gm
FC-RDPSO vs RDPSO	+	+	+	+	+	=	−	+	+	+	+	+	=	+	+	+	+	+	+	=	16/1/3/15
FC-RDPSO vs QPSO	+	+	+	=	+	=	−	+	+	+	+	=	+	+	+	+	+	+	+	=	15/1/4/14
FC-RDPSO vs CSO	+	+	+	+	+	+	−	+	−	+	+	−	=	+	−	−	+	−	+	=	12/6/2/6
FC-RDPSO vs SL-PSO	=	+	+	+	+	+	−	+	−	+	+	−	−	+	−	−	+	−	+	+	12/7/1/5
FC-RDPSO vs DMS-PSO	+	+	+	−	+	=	−	+	+	+	+	+	=	+	−	−	+	+	+	−	13/5/2/8
FC-RDPSO vs PSO-Co	=	+	+	+	+	=	+	+	+	=	+	+	+	+	+	+	+	+	+	+	17/0/3/17
FC-RDPSO vs PSO	+	+	+	+	+	=	+	+	+	+	+	+	+	+	+	+	+	+	+	+	19/0/1/19

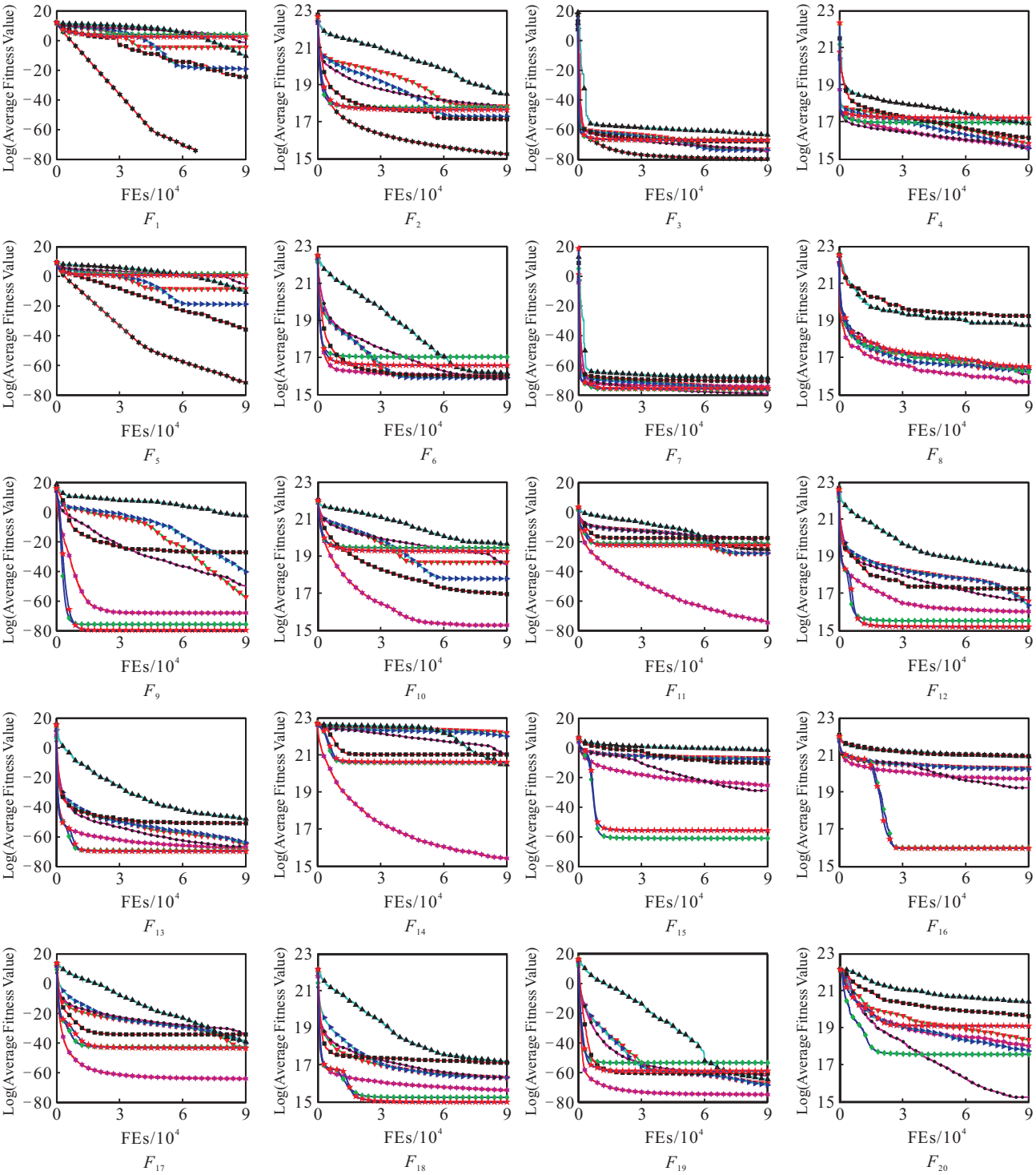

图 2 各算法优化20个测试函数的收敛曲线比较图

从表3的实验数据和图2的收敛曲线可以看出: 本文提出的FC-RDPSO算法在CEC'2013RPO测试函数上的优化性能总体上优于其他7个算法; 与原始的PSO算法对比, 发现FC-RDPSO算法在19个测试函数的性能更优越, 而只有在剩余的1个测试函数上, 两种算法的优化性能差不多, 表明FC-RDPSO完全优于原始PSO; 与较为知名的PSO改进算法DMS-PSO算法和QPSO算法相比, FC-RDPSO算法的优化能力在15个函数上强于QPSO算法, 在13个函数上强于DMS-PSO算法; 与较新的改进算法CSO算法相比, FC-RDPSO算法在12个函数上的优化性能优于CSO算法, 在6个函数上的优化性能弱于CSO; 与SL-PSO算法相比, FC-RDPSO算法在12个函数上的优化性能较好, 在7个函数上的优化性能较弱, 其余函数上的优化性能相当; 与RDPSO算法相比, FC-RDPSO算法在16个测试函数上的优化性能较好, 在3个测试函数上的优化性能相当, 而在测试函数F_7上的优化性能较弱. 从收敛曲线可以看出, 各个算法在F_7上的优化性能差距很小, 这是因为F_7是多峰不可分函数, 其局部最优位置的数量较为庞大, 粒子在搜寻全局最优的过程中易陷入局部最优, 并且很难跳出局部最优. 从收敛曲线图可以看到: 在函数F_1、F_2、F_5、F_{10}、F_{11}、F_{14}、F_{17}上, FC-RDPSO的优化性能比较显著, 收敛精度方面远远超过了同类算法; 同时, FC-RDPSO算法也具有较快的收敛速度, 特别是在函数F_1、F_3、F_{15}、F_{16}上前3×10^4次评估, FC-RDPSO的收敛速度明显比对照算法快得多. 综上可知, 本文提出的频繁覆盖策略的优势和作用明显, 使得RDPSO的全局搜索能力和搜索效率得到了极大的提高.

3.1.3　算法效率分析

一般而言, 良好的算法不仅要有很好的搜索精度, 也应该具有较高的效率. 因此, 对PSO算法的改进应该在尽量不降低计算效率的情况下来提高算法的搜索精度. 本文记录了各个算法优化各个函数达到一定精度所用的时间, 以衡量各个算法的计算效率, 而不是多数文献中所采用的算法运行到最大迭代次数所用的时间. 表5给出了各个函数的精度设置, 是本文所讨论算法基本都可以达到的平均水平. 若算法在结束时仍不能达到设置的精度, 则将算法达到最大评估次数的时间列入计算.

表5　CEC'2013RPO各测试函数精度设置

F_1	F_2	F_3	F_4	F_5	F_6	F_7	F_8	F_9	F_{10}
1e-05	5e+07	5e+09	5e+04	1e-05	5e+01	5e+02	5e+01	5e+01	5e+01
F_{11}	F_{12}	F_{13}	F_{14}	F_{15}	F_{16}	F_{17}	F_{18}	F_{19}	F_{20}
1e+02	1e+02	5e+02	5e+03	5e+04	5e+00	5e+02	5e+02	5e+00	5e+01

各算法在CEC'2013RPO 20个测试函数上独立运行30次, 记录每次运行达到所有函数规定精度所需时间, 然后计算30次的平均时间, 结果如图3所示.

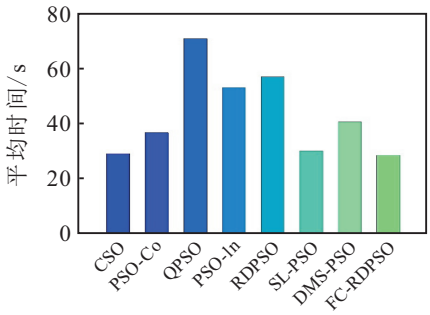

图3　各算法在CEC'2013RPO测试函数上的平均运行时间

从图3可以看到, FC-RDPSO算法运行时间在这8种算法中排名第二, 排名第一的是CSO算法, 这是由于CSO算法每次只更新一半的粒子, 大大节省了算法的运行时间. 与其余6种算法相比, FC-RDPSO算法的效率最高, 这与图2的收敛曲线比较结果是吻合的, 进一步验证了本文算法具有更高的收敛速度和寻优效率.

3.2　FC-RDPSO算法的高维扩展性能分析

由前面的分析可知, FC-RDPSO算法具有良好的优化性能, 从FC-RDPSO算法的框架结构来看, FC-RDPSO算法还应具有高维可扩展性. 这是因为此算法在执行过程中, 每一代不是优化所有的维度而只是优化一部分, 另一部分维度是保持不变的, 优化维度的上限由参数FCSCALE控制. 面对高维的大规模优化问题, 只要参数FCSCALE恰当, FC-RDPSO算法在大规模优化(LSGO)问题中应该依然可以保持良好的优化性能. 为了测试FC-RDPSO算法的高维可扩展性, 采用CEC'2008LSGO测试函数[32]进行实验, 并且选择面向LSGO问题的3个PSO改进算法进行算法性能的比较, 包括CSO算法[23]、CCPSO2算法[33]和EPUS-PSO算法[34]. CCPSO2算法利用协同进化框架下随机分组策略融合PSO算法来求解LSGO问题, CSO算法利用粒子间的竞争学习策略来整体处理所有决策变量来搜索最优解, EPUS-PSO算法根据搜索结果来调整群体的规模进行优化. 由相应的参考文献可知, 这3种算法求解LSGO问题都具有良好的优化性能. FC-RDPSO算法的参数覆盖规模FCSCALE设置为20, α设置为0.95, 算法终止条件为目标函数的最大评估次数$\max \mathrm{FEs} = 5\,000\times D$, D为优化问题维数. 表6给出了FC-RDPSO算法、CSO算法、CCPSO2算法和EPUS-PSO算法在CEC'2008LSGO六个测试函数分别在100维、500维、1 000维情况

下的实验结果,其中对比算法的结果来源于原文献. 表6同时给出了实验结果在检验水平0.05情况下的t检验结果,符号“+”表示FC-RDPSO与对照算法相比,FC-RDPSO有明显的优势;符号“=”表示FC-RDPSO与对照算法优化效果差不多,无明显差别,符号“−”表示FC-RDPSO与对照算法对比,FC-RDPSO算法有明显的劣势,表中最后一行统计了各个符号出现的总数.

表6 各算法在CEC'2008LSGO的6个测试函数在高维情况下的仿真实验结果及t检验结果

		FC-RDPSO		EPUS-PSO		CCPSO2		CSO
100-D	F_1	**0.00e+00** **0.00e+00**	+	7.47e-01 1.70e-01	+	7.73e-14 3.23e-14	+	9.11e-29 1.10e-28
	F_2	3.19e+01 5.80e+00	+	1.86e+01 2.26e+00	−	**6.08e+00** **7.83e+00**	=	3.355e+01 5.38e+01
	F_3	**3.01e+02** **3.11e+02**	+	4.99e+03 5.35e+03	=	4.23e+02 8.65e+02	=	3.90e+00 5.53e+02
	F_4	4.38e-01 8.65e-01	+	4.71e+02 5.94e+01	−	**3.98e-02** **1.99e-01**	+	5.60e+01 7.48e+00
	F_5	3.94e-03 6.32e-03	+	3.72e-01 5.60e-02	=	3.45e-03 4.88e-03	−	**0.00e+00** **0.00e+00**
	F_6	**8.53e-15** **2.51e-15**	+	2.06e+00 4.40e-01	+	1.44e-13 3.06e-14	+	1.20e-14 1.52e-15
500-D	F_1	**0.00e+00** **0.00e+00**	+	8.45e+01 6.40e+00	+	7.73e-14 3.23e-14	+	6.57e-23 3.90e-24
	F_2	6.11e+01 2.87e+00	−	4.35e-01 5.77e+04	=	5.79e+01 4.21e+01	−	**2.60e+01** 2.40e+00
	F_3	6.52e+02 1.98e+02	+	5.77e+04 8.04e+03	=	7.24e+02 1.54e+02	=	**5.74e+02** **1.67e+02**
	F_4	3.62e+00 2.40e+00	+	3.49e+03 1.12e+02	−	**3.98e-02** **1.99e-01**	+	3.19e+02 2.16e+01
	F_5	2.86e-03 5.45e-03	+	1.64e+00 4.69e-02	=	1.18e-03 4.61e-03	−	**2.22e-16** **0.00e+00**
	F_6	**4.68e-14** **6.30e-15**	+	6.64e+00 4.49e-01	+	5.34e-13 8.61e-14	+	4.13e-13 1.10e-14
1 000-D	F_1	8.08e-30 4.04e-29	+	5.53e+02 2.86e+02	+	5.18e-13 9.61e-14	+	1.09e-21 4.20e-23
	F_2	6.58e+01 2.72e+00	−	4.66e+01 4.00e-01	=	7.82e+01 4.25e+01	−	**4.15e+01** **9.74e-01**
	F_3	1.14e+03 1.88e+02	+	8.37e+05 1.52e+05	=	1.33e+03 2.63e+02	−	**1.01e+03** 3.02e+01
	F_4	6.47e+00 3.61e+00	+	7.58e+03 1.51e+02	−	**1.99e-01** **4.06e-01**	+	6.89e+02 3.10e+01
	F_5	2.07e-03 3.88e-03	+	5.89e+00 3.91e-01	=	1.18e-03 3.27e-03	−	**2.26e-16** **2.18e-17**
	F_6	**9.28e-14** **6.67e-15**	+	1.89e+01 2.49e+00	+	1.02e-12 1.68e-13	+	1.21e-12 2.64e-14
	+/=/−		15/0/3		7/7/7		9/2/7	

从表6记录的实验结果可以看到:FC-RDPSO算法在函数F_1和F_6上相比其他的算法表现出了更好的优化能力,能够求出函数F_1的全局最优解;在函数F_4上,CCPSO2算法的结果要优于FC-RDPSO算法,原因在于函数F_4存在大量的局部最优值,影响了其优化性能;对于函数F_5,CSO算法在500维下可以找到其真实值,发现FC-RDPSO算法在实验中有一半以上的运行也可以找到其真实值. 从表6中最后一行“+”的总和来看,FC-RDPSO的优化性能要强于其余3种对照算法. 其实FC-RDPSO和CCPSO2的对比也可以表明FC框架可以达到CC框架的性能,甚至在有些函数上优于CC框架. 纵向看表6中的数据可知,随着维数的增加,从100维到500维再到1 000维,FC-RDPSO算法依然可以保持良好的优化性能,具有良好的高维可扩展性.

3.3 FC-RDPSO算法在大规模全局优化测试函数上的仿真实验

为了进一步说明本文提出的频繁覆盖策略和FC-RDPSO算法不仅拥有高维可扩展性,而且对于大规模优化问题(1 000维及以上)也是一种行之有效的解决方案,在CEC'2010 LSGO测试函数上进行了仿真实验. 对比实验选用当前主流的一些算法,这些算法在大规模全局优化领域内性能突出,包括DECC-DML[34]、MLCC[35]、DECC-DG[36-37]、CSO[23]. 为了减少实验的随机性,算法在每个函数上均独立运行25次. 每次实验以最大函数评估次数(max FEs)作为终

止条件, $\max \mathrm{FEs} = 3 \times 10^6$.

CEC'2010 LSGO测试函数包含20个1 000维的复杂函数, 这些函数分为3类, $F_1 \sim F_3$ 为完全可分函数, $F_4 \sim F_{18}$ 为部分可分函数, $F_{19} \sim F_{20}$ 为完全不可分函数. 本文选用15个部分可分测试函数 $F_4 \sim F_{18}$ 进行仿真实验, FC-RDPSO参数只有一个FCSCALE, FCSCALE设置为30, α 设置为0.95, 对比实验算法参数参考相应文献.

表7　各算法在CEC'2010LSGO的15个测试函数上的仿真实验结果

函数	FC-RDPSO	MLCC	DECC-DML	CSO	DECC-DG
F_4	**7.72e+11**	9.61e+12	3.58e+12	3.65e+12	4.79e+12
	2.80e+11	3.43e+12	1.54e+12	5.73e+11	1.44e+12
F_5	5.37e+08	3.84e+08	2.98e+08	**2.64e+07**	1.55e+08
	6.25e+07	6.93e+07	9.31e+07	**9.00e+06**	2.17e+07
F_6	1.96e+07	1.62e+07	7.93e+05	2.16e+01	**1.64e+01**
	3.52e+05	4.97e+06	3.97e+06	5.96e-03	**2.71e-01**
F_7	**8.07e+01**	6.89e+05	1.39e+08	6.69e+07	1.16e+04
	5.49e+01	7.37e+05	7.72e+07	1.82e+07	7.41e+03
F_8	6.96e+07	4.38e+07	3.46e+07	4.39e+07	**3.04e+07**
	6.76e+07	3.45e+07	3.56e+07	6.50e+04	**2.11e+07**
F_9	**3.91e+07**	1.23e+08	5.92e+07	2.65e+08	5.96e+07
	3.48e+06	1.33e+07	4.71e+06	2.72e+07	8.18e+06
F_{10}	5.56e+03	**3.43e+03**	1.25e+04	1.08e+04	4.52e+03
	5.00e+02	**8.72e+02**	2.66e+02	7.58e+01	1.41e+02
F_{11}	1.99e+02	1.98e+02	**1.80e-13**	1.23e+02	1.03e+01
	1.12e-01	6.98e-01	**9.88e-15**	1.73e+01	1.01e+00
F_{12}	1.78e+04	3.49e+04	3.79e+06	2.97e+06	**2.52e+03**
	5.87e+03	4.92e+03	1.50e+05	3.93e+05	**4.86e+02**
F_{13}	**1.06e+03**	2.08e+03	1.14e+03	3.44e+03	4.54e+06
	6.33e+02	7.27e+02	4.31e+02	4.68e+03	2.13e+06
F_{14}	**1.64e+08**	3.16e+08	1.89e+08	1.08e+09	3.41e+08
	1.37e+07	2.77e+07	1.49e+07	6.55e+07	2.41e+07
F_{15}	1.35e+04	7.11e+03	1.54e+04	1.09e+04	**5.88e+03**
	5.00e+02	1.34e+03	3.59e+02	7.08e+01	**1.03e+02**
F_{16}	3.97e+02	3.76e+02	5.08e-02	2.50e+02	**7.39e-13**
	2.80e-01	4.71e+01	2.54e-01	2.33e+01	**5.70e-14**
F_{17}	**2.87e+04**	1.59e+05	6.54e+06	2.17e+07	4.01e+04
	3.98e+02	1.43e+04	4.63e+05	2.88e+06	2.85e+03
F_{18}	**2.27e+03**	7.09e+03	2.47e+03	7.73e+04	1.11e+10
	5.90e+02	4.77e+03	1.18e+03	5.50e+04	2.04e+09

表7给出了仿真实验结果及对比实验结果数据, 表中第一行表示算法在每个函数上独立运行25次的优化结果平均值, 第二行表示标准差. 表7的实验数据表明, 本文提出的FC-RDPSO算法在CEC'2010 LSGO 15个测试函数上的优化性能总体优于MLCC、DECC-DML、CSO和DECC-DG算法, 尤其在函数 F_7 上有较强的优势, 这说明本文提出的FC-RDPSO对高维复杂优化问题是有效的.

4　结　论

本文提出了一种频繁覆盖随机漂移粒子群优化算法(FC-RDPSO), 包含RDPSO算法和频繁覆盖策略(FC), 利用频繁覆盖策略随机选择若干决策变量空间来生成新的搜索子空间, RDPSO算法作为子优化器来搜索子空间得到 P_{best}, 通过迭代子空间不断地叠加, 覆盖到整个搜索空间, 进而找到 G_{best}. 这种搜索空间分治策略不仅能够增强RDPSO的全局开发和局部搜索能力, 而且具有良好的降维效果. 在CEC'2013RPO的20个复杂测试函数上开展实验, 并将实验结果与多个经典PSO改进算法以及最新的多种PSO改进算法进行对比. 实验结果表明, 本文所提算法在较少的计算时间内具有较快的收敛速度和更高的寻优精度, 在CEC'2008及CEC'2010大规模全局优化测试函数上的实验结果表明, FC-RDPSO算法在高维问题可扩展性能优越.

参考文献(References)

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

[2] Shi Y, Eberhart R. Modified particle swarm optimizer[C]. 1998 IEEE World Congress on Computational Intelligence. Anchorage: IEEE Xplore, 1998: 69-73.

[3] Poli R. Analysis of the publications on the applications of particle swarm optimisation[J]. J of Artificial Evolution & Applications, 2008, 2008: 10.

[4] Banks A, Vincent J, Anyakoha C. A review of particle swarm optimization part I: Background and development[J]. Natural Computing, 2007, 6(4): 467-484.

[5] Banks A, Vincent J, Anyakoha C. A review of particle swarm optimization Part II: Hybridisation, combinatorial, multicriteria and constrained optimization, and indicative applications[J]. Natural Computing, 2008, 7(1): 109-124.

[6] Van d B F, Engelbrecht A P. A new locally convergent particle swarm optimiser[C]. IEEE Int Conf on Systems, Man, and Cybernetics. Hammamet: IEEE, 2002: 6.

[7] 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.

[8] Frans V D B, Engelbrecht A P. A cooperative approach to particle swarm optimization[J]. IEEE Trans on Evolutionary Computation, 2004, 8(3): 225-239.

[9] van den Bergh F. An analysis of particle swarm optimizers[D]. Pretoria: Faculty of Natural and Agricultural Science, University of Pretoria, 2007.

[10] Kennedy J, Mendes R. Neighborhood topologies in fully informed and best-of-neighborhood particle swarms[J]. IEEE Trans on Systems, Man, and Cybernetics, Part C:

Applications & Reviews, 2006, 36(4): 515-519.

[11] Liang J J, Suganthan P N. Dynamic multi-swarm particle swarm optimizer with local search[C]. 2005 IEEE Congress on Evolutionary Computation. Edinburgh: IEEE, 2005: 522-528.

[12] 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.

[13] Qu B Y, Suganthan P N, Das S. A distance-based locally informed particle swarm model for multimodal optimization[J]. IEEE Trans on Evolutionary Computation, 2013, 17(3): 387-402.

[14] Riget J, Vesterstrøm J S. A diversity-guided particle swarm optimizer-the ARPSO[R]. Aarhus: University of Aarhus, 2002.

[15] Pant M, Radha T, Singh V P. A simple diversity guided particle swarm optimization[C]. IEEE Congress on Evolutionary Computation. Singapore: IEEE, 2007: 3294-3299.

[16] Wang H, Sun H, Li C, et al. Diversity enhanced particle swarm optimization with neighborhood search[J]. Information Sciences, 2013, 223(2): 119-135.

[17] Engelbrecht A P. Particle swarm optimization with crossover: A review and empirical analysis[J]. Artificial Intelligence Review, 2016, 45(2): 131-165.

[18] Pehlivanoglu Y V. A new particle swarm optimization method enhanced with a periodic mutation strategy and neural networks[J]. IEEE Trans on Evolutionary Computation, 2013, 17(3): 436-452.

[19] Krohling R A. Gaussian swarm: A novel particle swarm optimization algorithm[C]. IEEE Conf on Cybernetics and Intelligent Systems. Singapore: IEEE, 2004: 372-376.

[20] Richer T J, Blackwell T M. The Lévy particle swarm[C]. IEEE Congress on Evolutionary Computation. IEEE, 2006: 808-815.

[21] Sun J, Feng B, Xu W. Particle swarm optimization with particles having quantum behavior[C]. Congress on Evolutionary Computation. Portland: IEEE, 2004: 325-331.

[22] 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.

[23] Cheng R, Jin Y. A competitive swarm optimizer for large scale optimization[J]. IEEE Trans on Cybernetics, 2015, 45(2): 191-204.

[24] Cheng R, Jin Y. A social learning particle swarm optimization algorithm for scalable optimization[J]. Information Sciences, 2015, 291: 43-60.

[25] Fang W, Sun J, Wu X, et al. Study on the compression-expansion coefficient in drift particle swarm optimization[C]. 2012 IEEE Congress on Evolutionary Computation. IEEE, 2012: 1-6.

[26] Sun J, Wu X, Palade V, et al. Random drift particle swarm optimization algorithm: Convergence analysis and parameter selection[J]. Machine Learning, 2015, 101(1/2/3): 345-376.

[27] Liang J, Qu B, Suganthan P. Problem definitions and evaluation criteria for the CEC 2014 special session and competition on single objective real-parameter numerical optimization[R]. Zhengzhou: Zhengzhou University, 2013.

[28] Sun J, Palade V, Wu X, et al. Multiple sequence alignment with hidden Markov models learned by random drift particle swarm optimization[J]. IEEE/ACM Trans on Computational Biology and Bioinformatics, 2014, 11(1): 243-257.

[29] Tang K, Li X, Suganthan P N, et al. Benchmark functions for the cec'2010 special session and competition on large-scale global optimization[R]. Hefei: University of Science and Technology of China, 2009.

[30] Potter M A, De Jong K A. A cooperative coevolutionary approach to function optimization[C]. Int Conf on Parallel Problem Solving from Nature. Heidelberg: Springer Berlin, 1994: 249-257.

[31] Mahdavi S, Shiri M E, Rahnamayan S. Metaheuristics in large-scale global continues optimization: A survey[J]. Information Sciences, 2015, 295: 407-428.

[32] Tang K, Yao X, Suganthan P N, et al. Benchmark functions for the CEC' 2008 special session and competition on large scale global optimization[R]. Hefei: University of Science and Technology of China, 2007.

[33] Li X, Yao X. Cooperatively coevolving particle swarms for large scale optimization[J]. IEEE Trans on Evolutionary Computation, 2012, 16(2): 210-224.

[34] Hsieh S T, Sun T Y, Liu C C, et al. Solving large scale global optimization using improved particle swarm optimizer[C]. 2008 IEEE World Congress on Computational Intelligence. Hong Kong: IEEE, 2008: 1777-1784.

[35] Yang Z, Tang K, Yao X. Multilevel cooperative coevolution for large scale optimization[C]. 2008 IEEE World Congress on Computational Intelligence. Hong Kong: IEEE, 2008: 1663-1670.

[36] Omidvar M N, Li X, Mei Y, et al. Cooperative co-evolution with differential grouping for large scale optimization[J]. IEEE Trans on Evolutionary Computation, 2014, 18(3): 378-393.

[37] Omidvar M N, Li X, Yao X. Cooperative co-evolution with delta grouping for large scale non-separable function optimization[C]. 2010 IEEE Congress on Evolutionary Computation. Barcelona: IEEE, 2010: 1-8.

（责任编辑：齐 霁）