Journal of Computer Applications
计算机应用, 2020, 40(11): 3119 - 3126
ISSN 1001-9081
CODEN JYIIDU
2020-11-10
http://www.joca.cn

文章编号:1001-9081(2020)11-3119-08
DOI:10.11772/j.issn.1001-9081.2020040481

演化信息协助的动态协同随机漂移粒子群优化算法

赵　吉[1,2*],程　成[3]

(1. 无锡环境科学与工程研究中心, 江苏 无锡 214153; 2. 江南大学 物联网工程学院, 江苏 无锡 214122;
3. 中国船舶科学研究中心, 江苏 无锡 214082)

(*通信作者电子邮箱 queenji97@hotmail.com)

摘　要:为了改善随机漂移粒子群算法的群体多样性,通过演化信息的协助,提出动态协同随机漂移粒子群优化(CRDPSO)算法。利用上下文粒子的向量信息,粒子之间的动态协作增加了种群多样性,这有助于提高群体的搜索能力,并使整个群体协同搜索全局最优值。同时在演化过程中的每次迭代,利用二维空间分割树结构来存储算法中的估计解的位置和适应度值,从而实现快速适应度函数逼近。由于适应度函数逼近增强了变异策略,因此变异是自适应且无参数的。通过典型测试函数将CRDPSO算法和差分进化算法(DE)、协方差矩阵适应进化策略算法(CMA-ES)、非重复访问遗传算法(cNrGA)以及三种改进的量子行为粒子群算法(QPSO)进行比较。实验结果表明,不管是对于单峰还是多峰测试函数,CRDPSO的性能均是最优的,证明了该算法的有效性。

关键词:群体智能;动态协同进化;演化信息;自适应无参变异;二维空间分割

中图分类号:TP301.6　　**文献标志码**:A

Dynamic cooperative random drift particle swarm optimization algorithm assisted by evolution information

ZHAO Ji[1,2*], CHENG Cheng[3]

(1. *Wuxi Research Center for Environmental Science and Engineering*, *Wuxi Jiangsu* 214153, *China*;
2. *School of IoT Engineering*, *Jiangnan University*, *Wuxi Jiangsu* 214122, *China*;
3. *China Ship Scientific Research Center*, *Wuxi Jiangsu* 214082, *China*)

Abstract: A dynamic Cooperative Random Drift Particle Swarm Optimization (CRDPSO) algorithm assisted by evolution information was proposed in order to improve the population diversity of random drift particle swarm optimization. By using the vector information of context particles, the population diversity was increased by the dynamic cooperation between the particles, to improve the search ability of the swarm and make the whole swarm cooperatively search for the global optimum. At the same time, at each iteration during evolution, the positions and the fitness values of the evaluated solutions in the algorithm were stored by a binary space partitioning tree structure archive, which led to the fast fitness function approximation. The mutation was adaptive and nonparametric because of the fitness function approximation enhanced the mutation strategy. CRDPSO algorithm was compared with Differential Evolution (DE), Covariance Matrix Adaptation Evolution Strategy (CMA-ES), continuous Non-revisiting Genetic Algorithm (cNrGA) and three improved Quantum-behaved Particle Swarm Optimization (QPSO) algorithms through a series of standard test functions. Experimental results show that the performance of CRDPSO is optimal for both unimodal and multimodal test functions, which proves the effectiveness of the algorithm.

Key words: swarm intelligence; dynamic cooperative evolution; evolution information; adaptive nonparametric mutation; Binary Space Partitioning (BSP)

0　引言

随着科学技术的发展,许多现实世界中的优化问题变得越来越复杂,迫切需要更好、更有效的优化算法。粒子群优化(Particle Swarm Optimization, PSO)算法[1]自1995年首次提出以来,学者们对提高其性能进行了许多尝试[2]。然而,就PSO本身而言,它并不是一种全局优化算法[3]。为此,Sun等[4]根据外部电场中金属导体的自由电子模型[5]提出了一种新的随机漂移粒子群优化(Random Drift PSO, RDPSO)算法,并且能更加高效地估计复杂生化动力系统的参数。RDPSO能保持群体的多样性,利用平均位置的等待机制提高粒子的全局搜索能力,比PSO和遗传算法(Genetic Algorithm, GA)具有更好的性能[6]。RDPSO算法也可以用作数据挖掘中的优化问题的通用工具,例如聚类、分类、回归等,其广泛存在于现实世界的优化问题[7-8]中。

收稿日期:2020-04-17;**修回日期**:2020-07-13;**录用日期**:2020-07-21。　**基金项目**:国家自然科学基金资助项目(61672263);江苏省高校"青蓝工程"中青年学术带头人项目(2019);无锡环境科学与工程研究中心中青年学术带头人项目(2018)。

作者简介:赵吉(1980—),女,江苏泰兴人,副教授,博士,主要研究方向:进化计算、人工智能; 程成(1982—),男,江苏徐州人,高级工程师,硕士,主要研究方向:计算仿真、进化算法。

与其他进化算法一样，RDPSO算法也面临着早熟收敛和陷入局部最优的问题，导致性能损失和次优解。非重复访问策略与智能进化算法相结合[9-14]，通过使用二维空间分割树避免估计候选解被重复访问，引导算法搜索方向趋于有希望的搜索区域，从而提升搜索能力以提高算法的性能。文献[14]提出了基于搜索历史信息的非重复访问量子行为粒子群优化(Non-revisited Quantum-behaved PSO，NrQPSO)算法，利用搜索历史记录访问过的解位置，有效提高算法性能。但是其变异方式采用了简单的单粒子随机翻转变异，没有充分利用群体中粒子间的信息传递。同时NrQPSO需要设置参数阈值e，该值是个经验值，对解的影响很大，需要多次调试才能达到最优值，因此算法性能对该值的依赖度较大，不具备普遍性。为了进一步提高RDPSO算法在复杂优化问题中的性能，提高粒子的多样性，本文提出演化信息协助的动态协同随机漂移粒子群优化(Cooperative RDPSO，CRDPSO)算法，粒子动态地进行协同合作，通过演化过程中的信息协助同时记录访问过的解位置和适应度值，避免算法重复访问，增强自适应变异从而显著提高收敛精度和速度。

1 随机漂移粒子群优化算法

给定一个最小优化问题：

$$F(X);\ X \in S \subseteq \mathbf{R}^D \tag{1}$$

其中$F(X)$是一个在连续可行空间S中的适应度函数(或目标函数)。假设在具有m个粒子的群优化算法中，每个个体都被视为D维空间中的无体积粒子，第k代第i个粒子的位置向量和速度向量分别表示为：$\boldsymbol{X}_i^k=[x_{i,1}^k, x_{i,2}^k, \cdots, x_{i,D}^k]$和$\boldsymbol{V}_i^k=[v_{i,1}^k, v_{i,2}^k, \cdots, v_{i,D}^k]$；以及粒子的个体最优位置(personal best，pbest)$\boldsymbol{P}_i^k=[P_{i,1}^k, P_{i,2}^k, \cdots, P_{i,D}^k]$(给出最佳目标函数值或适应值的位置)。群体有一个称为全局最佳(global best，gbest)位置的向量$\boldsymbol{G}^k=[G_1^k, G_2^k, \cdots, G_D^k]$，它定义为种群中所有粒子中最佳粒子的位置。根据式(1)$\boldsymbol{P}_i^k$和$\boldsymbol{G}^k$分别按式(2)和式(3)计算：

$$\boldsymbol{P}_i^k=\begin{cases}\boldsymbol{X}_i^k, & F(\boldsymbol{X}_i^k)<F(\boldsymbol{P}_i^{k-1})\\ \boldsymbol{P}_i^{k-1}, & F(\boldsymbol{X}_i^k)\geqslant F(\boldsymbol{P}_i^{k-1})\end{cases} \tag{2}$$

$$\boldsymbol{G}^k=\boldsymbol{P}_g^k;\ g=\arg\min_{1\leqslant i\leqslant m}[F(\boldsymbol{P}_i^k)] \tag{3}$$

轨迹分析[15]表明，假设每个粒子都收敛到其局部吸引子，粒子群优化算法的收敛性是可以实现的。在PSO算法中，粒子向个体最优位置的方向运动类似于外部电场中金属导体内电子的漂移运动。然而，根据自由电子模型[5]，除了电场引起的定向运动外，电子还处于随机性的热运动中。电子运动的总体效应是电子倾向最小势能的位置。因此，如果把电子的位置作为问题的候选解，势能函数作为问题的目标函数，电子的运动显然与求最小解的过程非常相似。受此启发，假设PSO中的粒子像是在外部电场中金属导体内移动的电子，粒子的运动是热运动和指向粒子个体最优值方向运动的叠加。因此，粒子的速度可以表示为：$\boldsymbol{V}_{i,j}^{k+1}=\boldsymbol{V}1_{i,j}^{k+1}+\boldsymbol{V}2_{i,j}^{k+1}$，其中$\boldsymbol{V}1_{i,j}^{k+1}$和$\boldsymbol{V}2_{i,j}^{k+1}$分别表示热运动和定向运动的速度。基于这样的假设，文献[4]提出了下列RDPSO粒子更新方程：

$$\boldsymbol{V}_{i,j}^{k+1}=\alpha\left|\boldsymbol{C}_j^k-\boldsymbol{X}_{i,j}^k\right|\varphi_{i,j}^k+\beta(\boldsymbol{p}_{i,j}^k-\boldsymbol{X}_{i,j}^k) \tag{4}$$

$$\boldsymbol{X}_{i,j}^{k+1}=\boldsymbol{X}_{i,j}^k+\boldsymbol{V}_{i,j}^{k+1} \tag{5}$$

式(4)可以看作为$\boldsymbol{V}_{i,j}^{k+1}=\boldsymbol{V}1_{i,j}^{k+1}+\boldsymbol{V}2_{i,j}^{k+1}$。

$$\boldsymbol{V}1_{i,j}^{k+1}=\alpha\left|\boldsymbol{C}_j^k-\boldsymbol{X}_{i,j}^k\right|\varphi_{i,j}^k \tag{6}$$

$$\boldsymbol{V}2_{i,j}^{k+1}=\beta(\boldsymbol{p}_{i,j}^k-\boldsymbol{X}_{i,j}^k) \tag{7}$$

$$\boldsymbol{p}_{i,j}^k=\varphi_{i,j}^k\boldsymbol{P}_{i,j}^k+(1-\varphi_{i,j}^k)\boldsymbol{G}^k;\ 1\leqslant j\leqslant D \tag{8}$$

其中：$\boldsymbol{P}_i^k$表示pbest，$\boldsymbol{G}^k$表示gbest。$\boldsymbol{C}^k=(\boldsymbol{C}_1^k, \boldsymbol{C}_2^k, \cdots, \boldsymbol{C}_D^k)$为最优平均值，是所有粒子pbest的均值，即$\boldsymbol{C}_j^k=\left(\frac{1}{m}\right)\sum_{i=1}^{m}\boldsymbol{P}_{i,j}^k$，$1\leqslant j\leqslant D$。$\varphi_{i,j}^k$为服从标准正态分布的随机数序列，即$\varphi_{i,j}^k\sim U(0,1)$。$\alpha$称为压缩膨胀系数，$\beta$称为加速度系数，是两个非常重要的参数，可以通过调整来平衡粒子的全局搜索和局部搜索。为α和β设置大值意味着算法具有更好的全局搜索能力，而设置小值则意味着更好的局部搜索能力。在PSO算法中，因为$\boldsymbol{V}_{i,j}^k$和$\boldsymbol{V}_{i,j}^{k+1}$是有限的，因此，粒子是在一个有界的采样空间飞行。然而，在RDPSO算法中，$\boldsymbol{V}1_{i,j}^{k+1}$服从正态分布，也就意味着每次迭代时粒子的采样空间为$\mathbf{R}^D$，涵盖了整个搜索空间。因此，RDPSO算法的搜索过程和PSO算法完全不同。此外，由于没有考虑到加速度系数β，PSO算法中粒子收敛于一个确定的轨迹，而在RDPSO中的粒子则是以随机方式收敛。RDPSO算法也不同于漂移粒子群优化(Drift PSO，DPSO)算法[16]，它能更好地平衡粒子群的局部搜索和全局搜索，从而获得比DPSO更好的性能。

2 演化信息协助的CRDPSO算法

2.1 动态协同机制

对于高维问题，包含随机算法(如随机漂移粒子群算法和遗传算法等)在内的搜索算法都存在尺度规模的“维数灾难”问题。协同机制的重要性在于，它能使粒子群中粒子“好”的维度信息得以保留，并能防止丢弃潜在的有用信息。粒子的每个维度通过执行协同操作，可以分别评估每个维度的优劣，并保存最有用的信息加速收敛。协同进化机制可以提高算法在高维问题中的性能[17-18]。在进化的每次迭代中，每个粒子都会动态地连续更新一个上下文粒子向量，以便在协同合作过程的每个连续阶段获得新的信息。为了减少每个维度替换的适应度函数计算量，通过选择最优粒子向量并与其他粒子协作。改进的动态协同进化方式，进一步提高动态协作性能。在进化过程中，每个粒子$\boldsymbol{x}_i$通过进化公式产生5个进化的粒子$\{\boldsymbol{x}_{i_1}, \boldsymbol{x}_{i_2}, \cdots, \boldsymbol{x}_{i_5}\}$。因此，连同$\boldsymbol{x}_i$共得到6个个体，根据式(1)分别计算其适应度值，在6个个体中选择最佳个体$\boldsymbol{x}_{i_b}$；然后将$\boldsymbol{x}_{i_b}$指定为粒子$\boldsymbol{x}_i$的当前较优局部上下文粒子向量，随后将$\boldsymbol{x}_{i_b}$与其余5个个体进行协同进化。对于$\boldsymbol{x}_{i_b}$向量的每个维度，分别用其他5个个体的对应维度替换，并测试替换维度是否提高了适应度值。如果是，则采用替换值，并使用结果向量替换上下文粒子向量。重复此过程，直到所有5个测量向量的所有维度都得到评估，并选择最佳排列以形成新一代的粒子。动态协同进化机制的过程算法描述如下。

步骤1 对于当前粒子$\boldsymbol{x}_i$根据RDPSO进化式(4)、(5)、(8)进化产生5个子代粒子$\{\boldsymbol{x}_{i_1}, \boldsymbol{x}_{i_2}, \cdots, \boldsymbol{x}_{i_5}\}$。

步骤2 从$\{\boldsymbol{x}_i, \boldsymbol{x}_{i_1}, \boldsymbol{x}_{i_2}, \cdots, \boldsymbol{x}_{i_5}\}$中选取最优的粒子作为动态协作的上下文粒子向量$\boldsymbol{x}_{i_b}$。

步骤3 对于剩下的5个粒子，分别对每个粒子的每一个维度用$\boldsymbol{x}_{i_b}$的相应维度替换，同时根据式(1)计算适应度值，如

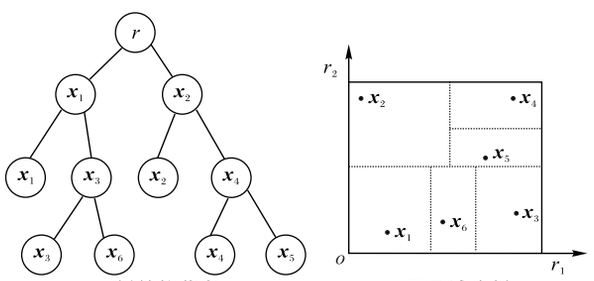
果替换后的粒子适应度值优于$F(\boldsymbol{x}_{i_b})$则保留相应维度,否则保留原有维度值。

步骤4　直到所有剩余5个粒子都和上下文粒子向量比较完,保留具有最优适应度值的粒子作为本次粒子进化的子代粒子。

步骤5　返回步骤1直到粒子群所有粒子都进行了动态协同进化后产生新的子代粒子群。

2.2　基于演化信息的自适应变异

CRDPSO使用二维空间分割(Binary Space Partitioning, BSP)树作为演化历史信息的存储方式,其记录了每个粒子的位置和适应度值$\{[\boldsymbol{x}_i, F(\boldsymbol{x}_i)]\}$,根据$\{\boldsymbol{x}_i\}$的分布将整个搜索空间$S$进行划分。最初,CRDPSO中的BSP树$T$仅包含根节点。树的每个节点记录了RDPSO先前访问的不同解$\boldsymbol{x}_i$。同时,它还记录了搜索空间的子区域S_i。节点使用欧几里得度量来对搜索空间进行二维分割。两个子节点a和b子节点沿着第k维将父节点的子区域线性分割为两个重叠的子区域,k的选取满足a和b在k维上的距离最大,即$\arg\max\left|a(k)-b(k)\right|$。假设$\{\boldsymbol{x}_1,\boldsymbol{x}_2,\boldsymbol{x}_3,\boldsymbol{x}_4,\boldsymbol{x}_5\}$是二维搜索空间内的估计解,它们在二维平面空间的分布如图1(b)所示。记录估计解的BSP树结构如图1(a)所示,BSP树根据估计解将二维搜索空间划分为多个小区域,如图1(b)的虚线区域所示,每个区域对应于现有的解。每个树节点包含其关联解的各种搜索演化信息(包括搜索空间中的解位置、适应度值等)及其在搜索空间分配的子区域。以这种方式,RDPSO生成的每个历史解被记录在树的节点中,并且BSP树用作查询$\boldsymbol{x}_i$子区域的有效数据结构[11]。



(a) BSP树结构节点　　(b) 子区域分割

图1　BSP树结构实例

Fig. 1　Example of BSP tree structure

因为BSP树同时记录了所有解的适应度值$F(\boldsymbol{x}_i)$,该记录可以看作适应度值$F(\boldsymbol{x})$的近似值$\tilde{F}(\boldsymbol{x})$。如果解$\boldsymbol{x}_k$的子区域是$S_k$,一个候选解$\boldsymbol{x}$在子区域$S_k$内,那么解$\boldsymbol{x}$的适应度值可以近似为$\boldsymbol{x}_k$的适应度值,即$F(\boldsymbol{x})\approx\tilde{F}(\boldsymbol{x})=F(\boldsymbol{x}_k)$。$\tilde{F}(x)$通过标准树节点插入操作来实现,其近似误差随着BSP树中一个一个记录的解数量增加而单调减小。本文采用的基于梯度下降式的变异方式是一种自适应的、无参数和引导的变异操作。其基于整个搜索历史自适应地调整变异步长,其搜索方向由近似适应度值控制,而不是像单粒子随机翻转突变那样随机。假设X和Y分别是节点$\boldsymbol{x}$和$\boldsymbol{y}$的子区域,如果$X\subseteq Y$,则Y是X的邻居。与X相关的Y的邻居数量为节点$\boldsymbol{x}$和$\boldsymbol{y}$的深度差。因此,首先找到子区域X的最优值,即节点$\boldsymbol{x}$的适应度值是其子区域邻居中最佳值,那么其所在子区域可以看作是其邻居子区域的最优子区域。引导自适应变异被定义为指向其最近的最优子区域方向,其变异方向就是向着离近似适应值$\tilde{F}(\boldsymbol{x})$局部增长最大的方向移动。设置一个变异方向变量$\boldsymbol{W}$作为指向$\boldsymbol{x}$的最近历史最优$\boldsymbol{y}$的方向,即$\boldsymbol{W}=\boldsymbol{y}-\boldsymbol{x}$。近似适应度值$\tilde{F}(\boldsymbol{x})$被看作一个阶跃函数,因此其最优值的形式是一个$D$维子区域而不是一个点。因为BSP树记录适应度值的拓扑结构代表了子区域之间的邻接关系,因此找到离$\boldsymbol{x}$最近的最优值相当于在$\tilde{F}(\boldsymbol{x})$中寻找$\boldsymbol{x}$的最近最优子区域。为了平衡梯度下降方向分配的搜索效果,引入区间(0,1)内的随机值作为变异步长α。因此$\boldsymbol{x}$变异公式为:

$$\boldsymbol{x}'=\boldsymbol{x}+\alpha\boldsymbol{W}=(1-\alpha)\boldsymbol{x}+\alpha\boldsymbol{y}\tag{9}$$

如果$\boldsymbol{x}$就是局部最优,即$\boldsymbol{x}=\boldsymbol{y}$,那么将从$\boldsymbol{x}$的子区域中随机选择一个突变体$\boldsymbol{x}_m$。图2显示了自适应变异的示例,搜索空间$S$和个体序列(即BSP树)空间分割和图1一样。假设$\boldsymbol{x}$是要变异的个体,相应的变异从搜索$\boldsymbol{x}$子区域开始,在实际编程中是从根节点开始。假设搜索终止于节点$\boldsymbol{x}_5$,S_2是$\boldsymbol{x}$的子区域,然后计算S_2到所有最优子区域的距离,其最近的最优子区域是S_5。因此,$\boldsymbol{x}$的变异范围是位于$\boldsymbol{x}$和$\boldsymbol{x}_5$之间直线上的一个点。变异操作步骤如下:

步骤1　搜索待变异粒子$\boldsymbol{x}$的子区域$s\subseteq S$。

步骤2　找到子区域s的最近最优子区域。

步骤3　$\boldsymbol{y}$是子区域s中的估计解。

步骤4　如果$\boldsymbol{x}=\boldsymbol{y}$,$\boldsymbol{x}$在子区域内进行随机变异,$\boldsymbol{x}_m$=Random($s$);否则$\boldsymbol{x}_m=(1-\alpha)\boldsymbol{x}+\alpha\boldsymbol{y}$(变异步长在自适应调整范围内随机分配,$\alpha$ = Random(0, 1)。

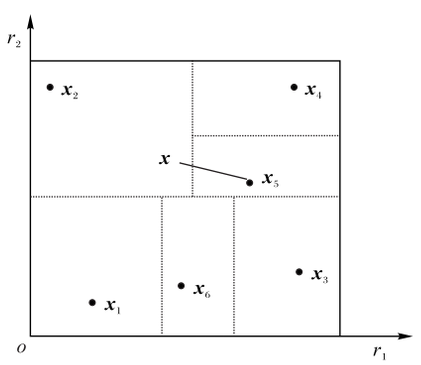


图2　自适应变异

($\boldsymbol{x}$和$\boldsymbol{x}_5$之间的直线是$\boldsymbol{x}$变异的可能范围)

Fig. 2　Adaptive mutation

(straight line between $\boldsymbol{x}$ and $\boldsymbol{x}_5$ is the range of possible mutation range of $\boldsymbol{x}$)

2.3　演化信息协助的动态协同随机漂移粒子群优化算法

本文提出的CRDPSO可以看作是RDPSO算法模块和自适应变异模块的结合体,其中算法模块包含了动态协同进化的随机漂移粒子群优化算法,而自适应模块则是由历史信息记录和随机梯度下降式自变异操作组成,如图3所示。对于一个D维搜索空间$S\subset\mathbf{R}^D$,算法中的单个粒子$\boldsymbol{x}$是一个$1\times D$的实值向量,即$\boldsymbol{x}\in\mathbf{R}^D$。算法通过4个主要步骤来搜索最优值,即种群初始化、搜索、变异和进化,其重点是通过演化历史信息,动态协同进化的自适应变异来增强种群多样性。CRDPSO从初始化粒子群$P=\{\boldsymbol{x}_i\}$开始,同时BSP树初始化为只包含根节点的树T,记录每个粒子的位置和适应度值。之后,粒子群经过自适应变异操作产生粒子群。随后根据动态协同机制,选择最好的置换粒子形成子代粒子群。产生新的

粒子群子代后需要重新计算粒子的适应度值并连同位置一起插入树 T 中，重复算法直到迭代次数超过预定值为止。

图 3 显示了 CPDRSO 算法的流程，数字表示 CPDRSO 迭代中的步骤顺序。

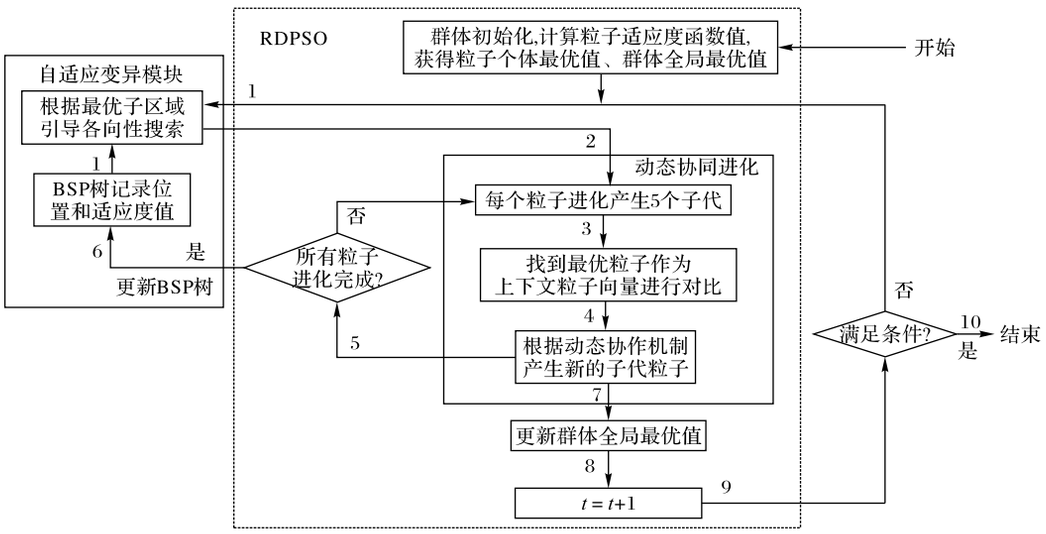


图 3 CRDPSO 算法流程

Fig. 3 Flowchart of CRDPSO algorithm

3 测试及结果

3.1 测试函数

本文采用一组标准测试函数包括多峰或单峰基准函数 $F=\{f_1(x),f_2(x),\cdots,f_{10}(x)\}$[19-20]来测试所提出的算法并验证其性能，表 1 中显示了 10 个测试函数，所有的仿真研究都是在 Matlab 中进行的。

表 1 测试函数

Tab. 1 Test functions

函数	函数名	测试函数	X	Global $f_{\min}$
f_1	Spherical model	$\sum_{i=1}^{D}x_i^2$	$[-100,100]^D$	0
f_2	Schwefel's Problem 2. 22	$\sum_{i=1}^{D}\lvert x_i\rvert+\prod_{i>1}^{D}\lvert x_i\rvert$	$[-10,10]^D$	0
f_3	Schwefel's Problem 1. 2	$\sum_{i=1}^{D}\left(\sum_{j=1}^{i}x_j\right)^2$	$[-100,100]^D$	0
f_4	Schwefel's Problem 2. 21	$\max_{i\in[i,D]}\lvert x_i\rvert$	$[-10,10]^D$	0
f_5	Restrigin's function	$\sum_{i>1}^{D}[x_i^2-10\cos(2\pi x_i)+10]$	$[-5.12,5.12]^D$	0
f_6	Ackley function	$-20\exp\left(-0.2\sqrt{\frac{1}{D}\sum_{i=1}^{D}x_i^2}\right)-\exp\left(\frac{1}{D}\sum_{i=1}^{D}\cos 2\pi x\right)+20+\mathrm{e}$	$[-32,32]^D$	0
f_7	Rosenbrock's function	$\sum_{i=1}^{D-1}[100(x_{i+1}-x_i^2)^2+(x_i-1)^2]$	$[-29,31]^D$	0
f_8	Griewank function	$\frac{1}{4\,000}\sum_{i=1}^{D}x_i^2-\prod_{i=1}^{D}\cos\frac{x_i}{\sqrt{i}}+1$	$[-600,600]^D$	0
f_9	Quaric function	$\sum_{i=1}^{D}ix^4+\mathrm{random}(0,1);x=x*2.56-1.28$	$[-1.28,1.28]^D$	0
f_{10}	Pathological	$\sum_{i=1}^{D}\left(0.5+\frac{\left(\sin\sqrt{100x_i^2+x_{i+1}^2}\right)^2-0.5}{1+0.001(x_i^2-2x_ix_{i+1}+x_{i+1}^2)^2}\right)$	$[-100,100]^D$	0

3.2 算法设置

除了 RDPSO 和 CRDPSO 算法，差分进化算法(Differential Evolution, DE)[21]、协方差矩阵适应进化策略算法(Covariance Matrix Adaptation Evolutionary Strategy, CMA-ES)[22]、非重复访问遗传算法(Continuous Non-revisiting Genetic Algorithm, cNrGA)[11]和三种版本的 QPSO，即 CCQPSO (Cooperative Quantum-behaved PSO)[18]、CLQPSO (Comprehensive Learning Quantum-behaved PSO)[23]和 NrQPSO[14]也通过测试函数进行性能比较。在实验中，CRDPSO 和 RDPSO 的参数设置与文献[4]中推荐的参数相同。对于 CLQPSO，β 从 1 线性下降到 0. 5，其他因子设置同文献[23]；NrQPSO 设置同文献[14]。对于 cNrGA，交叉速率是均匀交叉，设置为 r_x=0. 5。对于所有比

较的算法,群体大小设置为100,最大迭代次数设置为2 000。所有测试函数的维数设置为D=30。每种算法在实验中独立运行30次,测试结果取值为30次独立实验获得最优解的最好,最差和平均值以及标准偏差。

3.3　结果分析

对于所有测试函数,表2中列出了每种算法30次独立运行2 000次迭代的统计值,其中"Best""Worst"和"Ave.",分别代表30次独立运行中获得的最优解的最好值、最差值以及最优解的平均值,"Std."代表获得的最优解平均值的标准偏差。黑体字的值意味着对应算法对于特定测试函数是测试算法中最优的。从表2可以清楚地看到,对于所有测试函数,本文提出的CRDPSO算法性能都优于其他七种算法。虽然RDPSO和CCQPSO在f_1、f_2和f_6上均表现了出色的性能,但CRDPSO算法在所有其他测试函数上均胜过它们。除f_9以外,CRDPSO算法的标准偏差对于几乎所有测试函数都是最优的。这表明CRDPSO对于绝大多数测试函数都能获得最优解,具有最好的算法稳定性。此外,详细的仿真结果("Best""Worst""Ave."和"Std.")表明,CRDPSO的性能表现在十种测试情况下均排名第一。因此表2中列出的仿真结果表明,CRDPSO的整体性能优于其他比较的算法。通过使用演化信息机制,CRDPSO可以确保搜索过程中的非重复访问以及快速收敛到局部最优。同时,它利用自适应突变来指导粒子搜索更好的邻域,进一步提高局部搜索能力,使得CRDPSO算法可以获得最好的平均最优值,并且也是最稳定的。

不同算法的收敛曲线如图4所示。

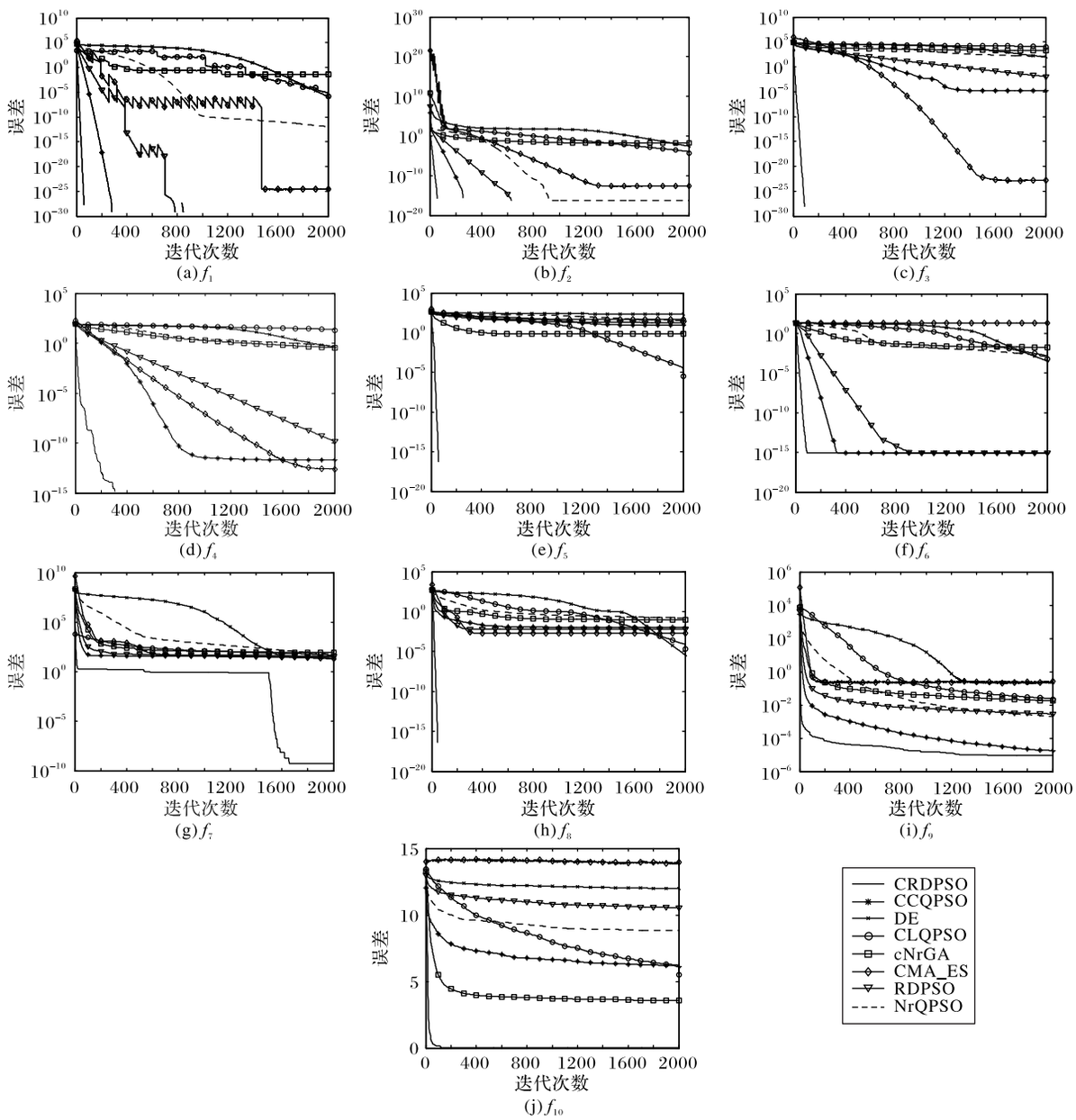


图4　不同算法的收敛曲线

Fig. 4　Convergence curves of different algorithms

表2结果表明,对于f_1、f_2和f_6,RDPSO、CRDPSO和CCQPSO算法最终都达到了最优值。然而,从图4(a)、(b)、(f)的收敛曲线可以看出,CRDPSO的收敛速度远优于其他两种算法,在不到200次的搜索迭代中,它就已经能收敛到全局

最优值。对于f_3~f_5和f_7~f_{10}，CRDPSO的收敛速度和收敛精度也远远超过其他七种算法，在很短的迭代次数内就找到了每个测试函数的全局最优值。CRDPSO算法能够更快地搜索到最优值是因为粒子一旦利用整个演化历史信息找到自己最近的最优邻域子区域，就可以进行自适应变异；同时，利用动态协同机制保持最优信息，使算法能够快速收敛到全局最优值。在图4中，对于f_1~f_5和f_8，CRDPSO已经达到Matlab能够识别的0，并且0的对数比例是负无穷大，因此适应度值不能再在图中显示。这很好地说明，对于这些测试函数，CRDPSO获得的目标函数值的仿真结果无限接近理论值，即0。对于图4(g)中的f_7，即使CRDPSO算法已经趋于收敛，但在后期(即1 500次迭代后)还能够进一步找到一个新的更好的解，这说明CRDPSO算法在搜索过程的最后阶段仍然具有更好的搜索能力，并且有很大的机会逃离局部最优解而得到更好的解。综上所述，CRDPSO算法与其他算法相比，具有最佳的收敛性能和优化精度。

为了确定除精度和收敛性外，CRDPSO算法与其他七种算法之间的性能是否存在显著差异，表3列出了不同算法之间的显著性水平为0.05的Wilcoxon秩和检验结果。从表3中可以看出，对于f_1、f_2和f_6，CRDPSO，RDPSO和CCQPSO没有显著差异。然而，对于绝大多数测试函数，CRDPSO算法可以得到h=1，这表明与其他算法相比，CRDPSO的优化性能有了显著提高。结果表明，无论是在单峰函数还是多峰函数中，CRDPSO算法的最优性能都明显较高，收敛更快、最优值精度更高。

表2 测试函数仿真结果

Tab. 2 Simulation results of test functions

函数	性能指标	RDPSO	CRDPSO	CMA-ES	cNrGA	CCQPSO	DE	CLQPSO	NrQPSO
f_1	Best	**0**	**0**	1.758 97E−25	1.113 9E−08	**0**	4.189 04E−07	7.402 07E−07	1.735 3E−14
	Worst	**0**	**0**	4.321 70E−25	9.734 0E−01	**0**	4.297 66E−06	2.131 62E−06	1.042 7E−11
	Ave.	**0**	**0**	2.848 21E−25	3.500 1E−02	**0**	1.340 00E−06	1.408 09E−06	1.094 9E−12
	Std.	**0**	**0**	6.322 76E−26	1.773 2E−01	**0**	8.676 63E−07	3.581 37E−07	2.337 8E−12
f_2	Best	**0**	**0**	1.385 6E−13	3.114 5E−06	**0**	0.001 373 62	2.964 3E−05	**0**
	Worst	**0**	**0**	2.859 9E−13	1.024 8E−01	**0**	0.005 792 98	6.863 6E−05	1.776 4E−15
	Ave.	**0**	**0**	2.305 7E−13	1.454 5E−02	**0**	0.002 737 26	4.635 3E−05	5.921 2E−17
	Std.	**0**	**0**	3.445 6E−14	2.659 4E−02	**0**	0.001 145 31	9.933 6E−06	3.243 2E−16
f_3	Best	9.362 7E−04	**0**	8.229 4E−24	4.838 7E+02	9.286 4E−12	39.372 560 8	5 608.600 16	6.664 6E+01
	Worst	5.661 0E−02	**0**	3.437 6E−23	2.799 1E+03	0.000 420 61	162.940 307 0	13 287.545 80	2.352 1E+02
	Ave.	1.116 2E−02	**0**	1.687 8E−23	1.726 2E+03	1.726 1E−05	88.368 455 4	9.852 5E+03	1.196 8E+02
	Std.	1.225 0E−02	**0**	6.117 7E−24	6.171 1E+02	7.641 1E−05	35.541 146 4	1.935 4E+03	3.388 4E+01
f_4	Best	6.906 5E−12	**0**	1.705 3E−13	1.449 6E−01	**0**	1.224 3E−01	1.644 9E+01	2.871 4E−01
	Worst	6.587 2E−10	**0**	4.121 1E−13	6.940 0E−01	5.823 6E−11	1.722 8E+00	2.270 2E+01	9.150 6E−01
	Ave.	1.559 6E−10	**0**	2.472 7E−13	3.569 7E−01	1.966 8E−12	4.330 7E−01	1.976 5E+01	5.494 1E−01
	Std.	1.644 2E−10	**0**	5.298 8E−14	1.481 3E−01	1.062 8E−11	3.486 3E−01	1.415 7E+00	1.451 5E−01
f_5	Best	9.012 6E+00	**0**	2.686 4E+01	3.140 6E−08	2.984 9E+00	1.847 6E+02	1.349 5E−06	2.379 1E+01
	Worst	6.523 3E+01	**0**	8.457 1E+01	2.067 4E+00	1.492 4E+01	2.250 1E+02	8.231 8E−06	5.616 2E+01
	Ave.	1.775 6E+01	**0**	4.649 8E+01	6.705 4E−01	9.389 9E+00	2.019 7E+02	3.148 1E−06	3.515 9E+01
	Std.	9.804 9E+00	**0**	1.282 7E+01	8.444 8E−01	2.724 8E+00	9.715 0E+00	1.521 1E−06	7.682 2E+00
f_6	Best	**8.881 8E−16**	**8.881 8E−16**	2.000 0E+01	5.649 8E−05	**8.881 8E−16**	1.731 7E−04	3.500 3E−04	2.168 0E−04
	Worst	**8.881 8E−16**	**8.881 8E−16**	2.159 3E+01	2.366 0E−01	**8.881 8E−16**	5.604 1E−04	9.245 5E−04	3.807 4E−03
	Ave.	**8.881 8E−16**	**8.881 8E−16**	2.143 6E+01	1.832 6E−02	**8.881 8E−16**	3.520 9E−04	6.172 0E−04	1.206 3E−03
	Std.	**1.002 9E−31**	**1.002 9E−31**	2.860 2E−01	4.442 5E−02	**1.002 9E−31**	1.014 3E−04	1.282 4E−04	7.795 1E−04
f_7	Best	2.086 2E−03	**0**	4.842 7E+00	1.555 4E+00	1.335 5E+01	2.203 5E+01	2.639 5E+01	2.961 7E+01
	Worst	9.274 9E+01	**1.477 7E−08**	1.411 2E+02	4.366 0E+02	9.494 6E+01	2.625 7E+01	6.736 2E+01	1.761 1E+02
	Ave.	3.720 7E+01	**5.168 7E−10**	1.991 5E+01	8.697 7E+01	3.342 8E+01	2.372 1E+01	4.157 9E+01	5.761 7E+01
	Std.	3.060 8E+01	**2.696 6E−09**	2.918 2E+01	8.456 5E+01	2.890 3E+01	9.187 1E−01	1.009 1E+01	3.969 5E+01
f_8	Best	**0**	**0**	**0**	7.406 1E−04	**0**	9.491 3E−07	5.864 8E−06	7.022 8E−03
	Worst	2.945 9E−02	**0**	1.232 1E−02	3.039 5E−01	3.677 0E−02	2.604 3E−05	3.420 2E−05	3.256 7E−01
	Ave.	6.484 0E−03	**0**	1.971 4E−03	9.698 4E−02	1.033 0E−02	3.948 3E−06	2.035 2E−05	1.524 9E−01
	Std.	7.763 6E−03	**0**	4.163 4E−03	8.592 4E−02	1.131 2E−02	4.527 2E−06	7.693 1E−06	1.017 8E−01
f_9	Best	1.292 8E−03	**6.724 7E−07**	5.040 4E−02	6.657 0E−03	8.498 3E−06	1.191 8E−01	4.763 2E−03	7.354 1E−04
	Worst	5.523 4E−03	3.489 0E−05	5.771 4E−01	3.147 5E−02	**3.323 2E−05**	3.608 7E−01	1.328 1E−02	4.804 9E−03
	Ave.	2.936 1E−03	**9.505 8E−06**	2.597 5E−01	1.805 6E−02	1.812 9E−05	2.171 0E−01	2.132 8E−02	2.073 1E−03
	Std.	1.004 8E−03	9.030 3E−06	1.243 5E−01	7.495 8E−03	**5.464 2E−06**	5.953 3E−02	3.103 1E−02	7.070 7E−04
f_{10}	Best	9.279 5E+00	**0.000 0E+00**	1.323 1E+01	2.573 7E+00	5.089 6E+00	1.120 7E+01	4.730 7E+00	6.912 1E+00
	Worst	1.147 0E+01	**1.062 9E−12**	1.450 0E+01	4.566 6E+00	7.372 3E+00	1.234 8E+01	6.003 5E+00	9.753 2E+00
	Ave.	1.053 7E+01	**5.314 4E−14**	1.397 0E+01	3.559 2E+00	6.129 7E+00	1.197 4E+01	5.515 1E+00	8.816 1E+00
	Std.	5.834 8E−01	**2.376 7E−13**	2.585 5E−01	6.361 5E−01	5.257 6E−01	2.218 2E−01	3.062 5E−01	5.622 8E−01

进化算法运行时间主要耗费在生成解和适应度函数计算。不同算法采用不同的策略生成可能解,CRSPSO的群体多样性保持比较好,因此目标函数收敛较快,主要运算时间在于访问BSP树(即搜索和插入节点)和动态协作的适应度计算。对比cNrGA和NrQPSO,由于CRDPSO还需要进行额外的动态协同计算,因此花费的时间也会相应增加,但是CRDPSO获得的解精度比cNrGA和NrQPSO有大幅提高。CRDPSO和CCQPSO算法都具备协同操作。CRDPSO每次迭代时,每个粒子都需要和动态上下文粒子向量进行对比获取更好的适应度值从而增加了计算开销。CRDPSO又利用了搜索历史信息,因此比CCQPSO提高了整体收敛速度并进一步提高了准确度。总体而言,CRDPSO算法增加时间和空间成本提升精确度和收敛度性能,这也充分符合了没有免费午餐的理论。对于实际问题如表面配准、加热优化设计和能源管理、通风和空调系统等,适应度函数评价需要比解生成耗费更多的时间。根据目前电脑硬件的发展,对于这类实际应用问题,CRDPSO所需要的额外时间开销和建立BSP树所需要的容量对于提高算法性能都是可以接受的,其综合性能是可以适应的。

表3　Wilcoxon秩和检验结果

Tab. 3　Wilcoxon rank sum test results

函数	CRDPSO vs CCQPSO		CRDPSO vs RDPSO		CRDPSO vs CLQPSO		CRDPSO vs cNrGA		CRDPSO vs DE		CRDPSO vs CMA_ES		CRDPSO vs NrQPSO	
	p	h	p	h	p	h	p	h	p	h	p	h	p	h
f_1	NaN	0	NaN	0	1.2118E-12	1	1.2118E-12	1	1.2108E-12	1	1.2108E-12	1	1.2118E-12	1
f_2	NaN	0	NaN	0	1.2118E-12	1	1.2118E-12	1	1.2118E-12	1	1.2059E-12	1	3.3370E-01	0
f_3	1.2118E-12	1	1.2118E-12	1	1.2118E-12	1	1.2118E-12	1	1.2118E-12	1	1.2118E-12	1	1.2118E-12	1
f_4	3.6097E-13	1	1.2118E-12	1	1.2118E-12	1	1.2118E-12	1	1.1001E-12	1	1.1001E-12	1	1.2118E-12	1
f_5	1.1613E-12	1	1.2108E-12	1	1.2118E-12	1	1.2118E-12	1	1.1001E-12	1	1.2088E-12	1	1.2118E-12	1
f_6	NaN	0	NaN	0	1.2118E-12	1	1.2118E-12	1	1.1001E-12	1	1.2118E-12	1	1.2118E-12	1
f_7	9.9025E-09	1	4.0279E-09	1	1.5647E-11	1	1.6027E-10	1	5.6013E-08	1	2.8306E-08	1	1.2710E-11	1
f_8	2.2016E-06	1	5.2014E-06	1	1.2118E-12	1	1.2118E-12	1	1.2118E-12	1	2.8000E-03	1	1.2118E-12	1
f_9	7.8252E-05	1	4.4618E-11	1	4.4618E-11	1	6.5095E-11	1	4.4618E-11	1	6.5095E-11	1	6.5095E-11	1
f_{10}	4.6908E-07	1	3.1575E-12	1	3.1575E-12	1	3.1575E-12	1	3.1549E-12	1	3.1549E-12	1	1.7203E-12	1

注:NaN表示Matlab中的非有效值;p表示两组样本是否相同的显著性概率;h表示假设检验的结果,0表示无差异,1表示显著差异。

4　结语

在RDPSO算法中,早熟收敛和粒子多样性是急需解决的两个问题。为此,本文提出了演化信息协助的动态协同RDPSO算法(CRDPSO),将动态协作机制与标准RDPSO算法相结合。群体粒子之间的动态协作机制采用动态上下文粒子信息可以更好地利用任何新的信息,增强了群体的搜索能力,有助于防止算法的早熟收敛。该算法通过搜索历史信息方案,利用二维空间分割(BSP)树结构来存储估计解的位置和适应度值。受益于空间划分方案,使用的记录树结构获得一种用于改进CRDPSO中突变策略的快速适应度函数,由此产生无参数的自适应变异,从而提高了粒子的变异能力。与其他比较流行的搜索随机算法相比,对于10个标准测试函数上的实验结果表明,该算法具有最好的全局寻优能力,并且收敛速度和精度都有所提高,证明了CRDPSO的有效性和高效率。改进后的算法在理论上可以证明其优越性,但在实际问题中还需要进一步验证。因此,将CRDPSO算法应用于实际工程问题成为未来的研究重点。下一步研究工作是验证CRDPSO算法在电力经济调度、数据分析和生物路径参数识别等工程应用中的性能。同时,为了进一步充分利用和挖掘有效历史信息,历史演化信息在粒子群搜索中的信息动态可视化也是今后的一个研究内容。

参考文献(References)

[1] KENNEDY J, EBERHART R C. Particle swarm optimization[C]// Proceedings of the 1995 International Conference on Neural Networks. Piscataway: IEEE, 1995: 1942-1948.

[2] 周文峰,梁晓磊,唐可心,等. 具有拓扑时变和搜索扰动的混合粒子群优化算法[J]. 计算机应用, 2020, 40(7): 1913-1918. (ZHOU W F, LIANG X L, TANG K X, et al. Hybrid particle swarm optimization algorithm with topological time-varying and search disturbance[J]. Journal of Computer Applications, 2020, 40(7): 1913-1918.)

[3] VAN DEN BERGH F, ENGELBRECHT A P. A convergence proof for the particle swarm optimiser [J]. Fundamenta Informaticae, 2010, 105(4):341-374.

[4] SUN J, PALADE V, CAI Y, et al. Biochemical systems identification by a random drift particle swarm optimization approach [J]. BMC Bioinformatics, 2014, 15(S6): No. S1.

[5] OMAR M A. Elementary Solid State Physics: Principles and Applications[M]. Boston: Addison-Wesley, 1994:138-169.

[6] 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.

[7] WU J, ZHANG H, LI J, et al. Scenario based stochastic optimal operated for hybrid energy system with random drift swarm optimization [C]// Proceedings of the 2019 IEEE Congress on Evolutionary Computation. Piscataway: IEEE, 2019:1176-1181.

[8] ELSAYED W T, HEGAZY Y G, EL-BAGES M S, et al. Improved random drift particle swarm optimization with self-adaptive mechanism for solving the power economic dispatch problem [J]. IEEE Transactions on Industrial Informatics, 2017, 13(3): 1017-1026.

[9] YANG Z, WU A. A non-revisiting quantum-behaved particle swarm optimization based multilevel thresholding for image segmentation [J]. Neural Computing and Applications, 2020, 32(16):12011-12031.

[10] ZHANG X, ZHANG X. A non-revisiting artificial bee colony

algorithm for phased array synthesis [J]. EURASIP Journal on Wireless Communications and Networking, 2017, 2017: No. 7.

[11] CHOW C K, YUEN S Y. Continuous non-revisiting genetic algorithm with random search space re-partitioning and one-gene-flip mutation [C]// Proceedings of the 2010 IEEE Congress on Evolutionary Computation, Piscataway: IEEE, 2010:1-8.

[12] CHOW C K, YUEN S Y. Continuous non-revisiting genetic algorithm with overlapped search sub-region [C]// Proceedings of the 2012 IEEE Congress on Evolutionary Computation. Piscataway: IEEE, 2012:1-8.

[13] YUEN S Y, CHOW C K. A genetic algorithm that adaptively mutates and never revisits [J]. IEEE Transactions on Evolutionary Computation, 2009, 13(2):454-472.

[14] 赵吉,程成. 基于演化搜索信息的量子行为粒子群优化算法[J]. 计算机工程与应用, 2017, 53(9):41-46, 126. (ZHAO J, CHENG C. Improved QPSO algorithm based on search history [J]. Computer Engineering and Applications, 2017, 53(9): 41-46, 126.)

[15] CLERC M, KENNEDY J. The particle swarm — explosion, stability, and convergence in a multidimensional complex space [J]. IEEE Transactions on Evolutionary Computation, 2002, 6(1): 58-73.

[16] SUN J, ZHAO J, WU X, et al. Parameter estimation for chaotic systems with a drift particle swarm optimization method [J]. Physics Letters A, 2010, 374(28): 2816-2822.

[17] LI Y, XIANG R, JIAO L, et al. An improved cooperative quantum-behaved particle swarm optimization [J]. Soft Computing, 2012, 16(6): 1061-1069.

[18] LI Y, JIAO L, SHANG R, et al. Dynamic-context cooperative quantum-behaved particle swarm optimization based on multilevel thresholding applied to medical image segmentation [J]. Information Sciences, 2015, 294: 408-422.

[19] KOUMOUSIS V K, KATSARAS C P. A saw-tooth genetic algorithm combining the effects of variable population size and reinitialization to enhance performance [J]. IEEE Transactions on Evolutionary Computation, 2006, 10(1):19-28.

[20] YAO X, LIU Y, LIN G. Evolutionary programming made faster [J]. IEEE Transactions on Evolutionary Computation, 1999, 3(2):82-102.

[21] STORN R, PRICE K. Differential evolution — a simple and efficient heuristic for global optimization over continuous spaces [J]. Journal of Global Optimization, 1997, 11(4): 341-359.

[22] HANSEN N. The CMA evolution strategy: a tutorial [EB/OL]. [2020-03-06]. https://arxiv.org/pdf/1604.00772.pdf.

[23] 陈伟,周頔,孙俊,等. 一种采用完全学习策略的量子行为粒子群优化算法[J]. 控制与决策, 2012, 27(5): 719-723, 730. (CHEN W, ZHOU D, SUN J, et al. Improved quantum-behaved particle swarm optimization algorithm based on comprehensive learning strategy [J]. Control and Decision, 2012, 27(5): 719-723, 730.)

This work is partially supported by the National Natural Science Foundation of China (61672263), the Program of "Qing Lan Project" Young and Middle-aged Academic Leaders in Colleges and Universities of Jiangsu Province (2019), the Program of Young and Middle-aged Academic Leaders of Wuxi Research Center for Environmental Science and Engineering (2018).

ZHAO Ji, born in 1980, Ph. D., associate professor. Her research interests include evolutionary computing, artificial intelligence.

CHENG Cheng, born in 1982, M. S., senior engineer. His research interests include computational simulation, evolutionary algorithm.