文章编号: 1001-0920 (2015) 09-1575-08 DOI: 10.13195/j.kzyjc.2014.0853

基于动态学习策略的群集蜘蛛优化算法

王艳娇[1], 李晓杰[1], 肖 婧[2]

(1. 东北电力大学 信息工程学院，吉林省 吉林市 132012；2. 哈尔滨工程大学 自动化学院，哈尔滨 150001)

摘 要: 为了提高群集蜘蛛优化 (SSO) 算法的性能, 提出一种基于动态学习策略的群集蜘蛛优化 (DSSO) 算法. 该算法通过群体协作过程中学习因子的动态选择, 平衡算法的搜索能力和勘探能力; 采用随机交叉策略和云模型改进协作过程个体更新方式, 在维持种群多样性的同时尽量提高收敛速度. 基于标准测试函数的仿真实验表明, DSSO 算法可有效避免早熟收敛, 在收敛速度和收敛精度上较标准 SSO 算法和其余 4 种较具代表性的优化算法均有显著提高.

关键词: 群集蜘蛛优化算法；函数优化；动态学习

中图分类号: TP273 **文献标志码:** A

Social spider optimization with dynamic learning strategy

WANG Yan-jiao[1], *LI Xiao-jie*[1], *XIAO Jing*[2]

(1. College of Information Engineering，Northeast Dianli University，Jilin 132012，China；2. College of Automation, Harbin Engineering University，Harbin 150001，China. Correspondent：WANG Yan-jiao，E-mail：563274435@qq.com)

Abstract: In order to improve the performance of social spider optimization(SSO) algorithm, a social spider optimization algorithm with the dynamic learning strategy(DSSO) is proposed. In this algorithm, a dynamic selection mechanism for the learning factor in population cooperation is applied to balance solution accuracy and search speed. A manner to update individual combining randomized crossover strategy and cloud theory is proposed to improve the collaboration manner, which can maintain the diversity of the population as much as possible and improve searching speed. Experimental results on benchmark functions show that the DSSO algorithm improves convergence property and robustness compared with the representative four algorithms.

Keywords: social spider optimization algorithm；function optimization；dynamic learning strategy

0 引 言

群集蜘蛛优化算法 (SSO)[1]是新近提出的一种基于群智能的模拟蜘蛛群集行为的优化算法. 在 19 个标准函数上测试, 与现在较为优秀的进化算法 —— 人工蜂群算法 (ABC) 和粒子群算法 (PSO) 相比, SSO 算法均获得了更为优秀的优化效果. 此后, 该算法又成功解决了约束优化问题[1-2]. 与其他群智能优化算法一样, SSO 算法在搜索过程中也存在收敛速度慢、易陷入局部最优等不足. 由于其刚刚被提出, 还未受到各领域学者的充分重视, 目前尚未见到与其性能改进相关的研究成果发布, 算法的理论体系还不完善.

鉴于此, 本文提出一种基于动态学习策略的群集蜘蛛优化算法 (DSSO), 通过群体协作过程的学习因子的动态设定, 平衡算法的搜索能力和勘探能力; 综合云模型和随机交叉策略优势改进协作过程个体更新方式, 在维持种群多样性的同时尽量提高收敛速度. 实验证实, 与基本 SSO 算法和其余 4 种较为优秀的优化算法相比, 本文综合上述两方面改进提出的 DSSO 算法在收敛速度、收敛精度上均有明显改善.

1 基本群集蜘蛛优化算法

SSO 算法模拟蜘蛛群集运动规律实现寻优过程, 将整个搜索空间视为蜘蛛运动所依附的蜘蛛网, 蜘蛛位置对应于优化问题的可能解, 相应权值对应于评价个体好坏的适应度值. SSO 算法在解决函数优化问题时, 随机产生蜘蛛位置, 通过雌蜘蛛和雄蜘蛛的内部协作运动以及婚配过程进行信息交互, 最终获得问题的最优解, 其关键步骤如下.

1.1 初始化

初始化 SSO 算法相关参数, 包括种群数目 N 和概率因子 PF, 并按下式分别计算婚配半径 r 及雌蜘蛛

收稿日期: 2014-05-30；**修回日期:** 2014-08-21.

基金项目: 国家自然科学基金项目(61175126)；东北电力大学博士科研启动基金项目(BSJXM-2013-20).

作者简介: 王艳娇(1985−), 女, 副教授, 博士, 从事进化计算、智能信息处理的研究；李晓杰(1988−), 女, 硕士生, 从事进化计算的研究.

和雄蜘蛛数目 N_f 和 N_m:

$$r=\sum_{j=1}^{N}({P_j}^{\rm high}-{P_j}^{\rm low})\Big/2N; \tag{1}$$

$$\begin{cases} N_f=\lfloor(0.9-{\rm rand}\times 0.25)\times N\rfloor,\\ N_m=N-N_f. \end{cases} \tag{2}$$

其中: ${P_j}^{\rm high}$ 和 ${P_j}^{\rm low}$ 为第 j 维变量的上下限, $\lfloor\cdot\rfloor$ 为向下取整, rand 为 [0,1] 之间随机数.

SSO 算法按下式随机产生雌性初始种群和雄性初始种群:

$$\begin{cases} F_{ij}=F_{j\min}+{\rm rand}(F_{j\max}-F_{j\min}),\\ M_{kj}=M_{j\min}+{\rm rand}(M_{j\max}-M_{j\min}). \end{cases} \tag{3}$$

其中: F_{ij} 和 M_{kj} 分别为雌性种群和雄性种群中的个体, $F_{j\max}$ 和 $F_{j\min}$ 为第 j 维变量的上下限, $i\in\{1,2,\cdots,N_f\}, k\in\{1,2,\cdots,N_m\}, j\in\{1,2,\cdots,D\}$.

1.2 雌性种群协作过程

雌性蜘蛛通过自身震动吸引或排斥其他个体, SSO 算法为模拟这一行为针对雌性个体设计了依概率判别吸引或者排斥的两种合作震动模式, 即

$$F_i^{k+1}=\begin{cases} {F_i}^K+\alpha{\rm Vib}_{c_i}(S_c-{F_i}^k)+\\ \beta{\rm Vib}_{b_i}(S_b-{F_i}^k)+\delta({\rm rand}-0.5),\\ \quad {\rm rand}\leqslant {\rm PF};\\ {F_i}^K-\alpha{\rm Vib}_{c_i}(S_c-{F_i}^k)-\\ \beta{\rm Vib}_{b_i}(S_b-{F_i}^k)+\delta({\rm rand}-0.5),\\ \quad {\rm else}. \end{cases} \tag{4}$$

其中: α,β,δ 为 [0,1] 之间随机数, S_c 为距 i 最近且高于其权重的雌性个体 c (即距离自身最近且优于自身的个体), ${\rm Vib}_{c_i}$ 为个体 i 对个体 c 的震动感知能力, S_b 和 ${\rm Vib}_{b_i}$ 分别为整个雌雄种群中拥有最高权重的个体 b 对个体 i 的震动感知能力. 相应权重 (适应度函数) 和震动感知能力的计算方式如下:

$$\begin{cases} w_i=1-\dfrac{J(S_i)-{\rm worst}_i}{{\rm best}_i-{\rm worst}_i},\\ w_i=\dfrac{J(S_i)-{\rm worst}_i}{{\rm best}_i-{\rm worst}_i}; \end{cases} \tag{5}$$

$${\rm Vib}_{i_j}=w_j{\rm e}^{-{d_{ij}}^2}. \tag{6}$$

其中: 式 (5) 上半部分表示最大值问题, 下半部分表示最小值问题; $J(S_i)$ 为个体 i 的目标函数值, ${\rm Vib}_{i_j}$ 为个体 j 对个体 i 的震动感知能力, d_{ij} 为个体 i 和 j 的欧氏距离. 对于最大化问题, 有

$${\rm best}_s=\max\{J(S_i)\},\ {\rm worst}_s=\min\{J(S_i)\},\quad i=1,2,\cdots,N;$$

在最小化问题中, 有

$${\rm best}_i=\min\{J(S_i)\},\ {\rm worst}_i=\max\{J(S_i)\},\quad i=1,2,\cdots,N.$$

1.3 雄性种群协作过程

生物学上, 雄性蜘蛛具有自动识别聚集功能, 可以自动分成较为优异的支配雄性子种群和较差的非支配雄性子种群. 支配雄蜘蛛具有吸引与其靠近的雌性蜘蛛的能力, 非支配雄蜘蛛具有向雄性种群中间个体聚集的趋势. 为模拟这一行为, SSO 算法为其设计了分类协作机制, 即

$$M_i^{k+1}=\begin{cases} {M_i}^k+\alpha\times{\rm Vib}_{f_i}(s_f-{M_i}^k)+\\ \delta({\rm rand}-0.5),\ w_{N_{f+1}}\geqslant w_{N_{f+m}};\\ {M_i}^k+\alpha\Big(\dfrac{\sum\limits_{h=1}^{N_m}{M_h}^k w_{N_{f+h}}}{\sum\limits_{h=1}^{N_m}w_{N_{f+h}}}-{M_i}^k\Big),\ {\rm else}. \end{cases} \tag{7}$$

其中: 个体权值大于中间个体的权值 $w_{N_{f+m}}$ (指对权值排序处于中间位置的权值), 即满足 $w_{N_{f+1}}\geqslant w_{N_{f+m}}$ 的雄性个体称为支配个体, 由其组成的子群体称为雄性支配子种群 T_D, 相应地, 其余个体称为非支配个体. s_f 为距雄性支配个体 i 最近的雌性个体 f, ${\rm Vib}_{f_i}$ 代表个体 i 对个体 f 的震动感知能力, 而 $\sum\limits_{h=1}^{N_m}{M_h}^k w_{N_{f+h}}\Big/\sum\limits_{h=1}^{N_m}w_{N_{f+h}}$ 则代表雄性种群的中心位置.

1.4 婚配行为

对于较为优秀的雄性支配个体 i, 如果某些雌性个体处于其婚配半径内 (即满足 $d_{ij}\leqslant r,\ i\in T_D,\ j=1,2,\cdots,N_f$), 则发生婚配行为. 具体步骤如下.

Step 1: 判断雄性个体 i 婚配半径内是否存在雌性个体, 如果不存在, 则转至 Step 5; 否则, 将其及在其婚配半径内的所有雌性个体结合为一个子种群, 记为 T_G.

Step 2: 计算 T_G 内各个体的分配概率

$$PS_k=\frac{w_k}{\sum\limits_{g\in T_G}w_g},\ k\in T_G. \tag{8}$$

Step 3: 逐维依 Step 2 中概率按照轮盘赌方式[3] 从 T_G 内选择个体, 该个体这一维对应的数值即作为婚配新个体 $S_{\rm new}$ 对应维上的取值.

Step 4: 如果新个体 $S_{\rm new}$ 优于雌、雄整体种群中最差个体, 则代替该个体.

Step 5: 判断所有雄性支配个体是否完成婚配行为, 完成则结束, 否则转至 Step 1.

表 1 给出了单个雄峰的婚配行为. 如表 1 所示, 对于雄性蜘蛛 M_2, 在其婚配半径内的雌性蜘蛛有 F_2、F_3 和 F_4, 将它们组成一个种群, 按照式 (8) 计算每个

蜘蛛的待选概率 P_S, 然后按照轮盘赌方式确定新蜘蛛个体 $S_{\rm new}$ 中各维数值. 按表 1 所示, 对于 $S_{\rm new}$ 中第 1 维数值为轮盘赌方式选择的 M_2 的第 1 维; 相应地, 新个体的第 2 维为 F_2 第 2 维对应的数值.

表 1　婚配行为描述

个体	位置	w	PS
F_1	(−1.9,0.3)	0.00	—
F_2	(1.4,1.1)	0.57	0.22
F_3	(1.5,0.2)	0.42	0.16
F_4	(0.4,1.0)	1.00	0.39
M_1	(0.9,0.7)	0.57	—
M_2	(0.8,−2.6)	0.42	0.22
$S_{\rm new}$	(0.8,1.1)	1.00	—

1.5　SSO 算法流程

为进一步理解 SSO 算法的工作原理, 图 1 给出了其具体操作流程.

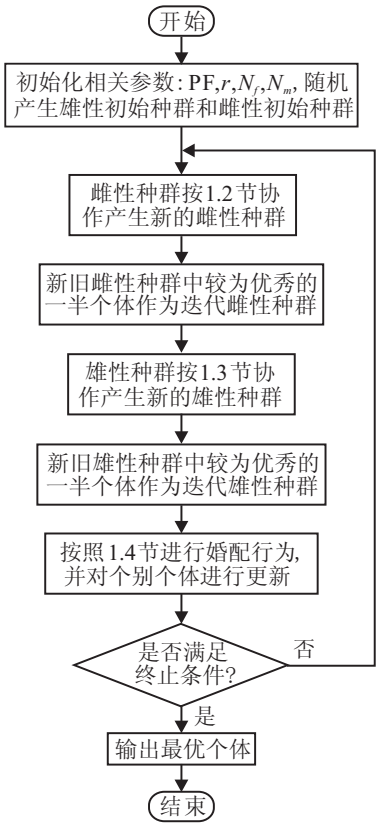


图 1　SSO 算法流程

2　基于动态学习策略的群集蜘蛛优化算法

进化算法的个体更新方式对其收敛速度和收敛精度的影响最大. SSO 算法的个体更新方式对应于协作过程, 为此, 这里对其重点分析. 如式 (4) 和 (5) 所示的协作过程可以理解为: 新个体 = 原个体 + 原个体向较优个体学习 + 随机部分, 各项可分别理解为自我认知部分、学习部分和随机部分. 学习部分保证个体向更优方向进化, 而随机部分可以维护种群多样性. 实验证实 SSO 算法存在收敛精度不佳、收敛速度慢等不足. 本文通过深入研究和大量实验发现, 学习部分和随机部分的设计不够合理是造成上述不足的根本原因, 为此本文综合改进这两大方面, 提出一种基于动态学习策略的群集蜘蛛优化算法 (DSSO).

2.1　协作过程中的学习部分改进

SSO 算法的协作学习部分可以理解为: 学习因子 × 优秀个体与原个体的差向量, 学习因子越大, 新个体保留较优个体的信息越多, 这可在一定程度上提高算法的收敛速度, 但种群多样性维持能力较差; 而学习因子越小, 新个体保持原有信息越多, 多样性维持较好, 但收敛速度较慢. 综上说明, 合理设置学习因子是平衡算法收敛速度和勘探能力的关键因素.

学习因子部分为: 随机数 × 震动感知能力. 由式 (6) 可见, 震动感知能力与个体适应度值和距离有关. 一般地, 算法运行初期, 个体之间的距离都较大, $\mathrm{e}^{-d_{ij}{}^2}$ 几乎为 0, 致使震动感知能力与学习因子都接近于 0, 相应的原个体向其他优秀个体的学习能力也几乎为 0. 显然, 协作过程退化演变为原个体与随机部分的变化, 这样的协作方式几乎不会为种群提供方向性指导信息, 勘探能力及收敛速度都急剧下降, 难以搜索到更为优秀的个体. 综上说明, SSO 算法收敛速度慢、勘探能力低的根本原因之一是学习因子中震动感知能力设置不合理. 为此, 本文将对其进行改进.

算法不同进化过程对优劣程度不同的个体要求差异很大. 一方面, 算法进化初期应加大算法使个体尽量在全局范围内搜索, 增强种群的开采能力, 加强较差个体与全局最优个体之间的信息交流, 加速较差个体向全局最优解的进化速度, 促进种群收敛, 而进化后期则需增强种群探索能力, 即个体的局部搜索能力, 使得个体能够在自身邻域附近进行精细搜索, 增大个体搜索到全局最优解的概率. 随着进化的进行, 应逐渐减少个体向优秀个体的学习、加大自身的调整力度. 另一方面, 对于种群中适应度值较差个体, 应增强其与优秀个体之间的学习和交流, 增强全局搜索能力, 加速其向全局最优解的进化速度. 而对于种群中适应度值较优个体, 则应尽量保留个体内的优良基因, 增大其局部搜索能力.

基于上述思想, 本文对SSO算法的学习因子进行了改进, 特别地, 与基本 SSO 算法不同, 为进一步平衡算法的搜索速度和收敛精度, 对自身也要进行学习调整. 由于每一代中个体的相对优劣程度是动态变化的, 称为动态学习因子, 其自身及向其他优秀个体学习的具体方案如下:

$$K = K_{\min} + (K_{\max} - K_{\min})\Big((\mathrm{e}^{\frac{g}{G}\ln(2)} - 1)w_0 + \frac{f_j - f_{\min}}{f_{\max} - f_{\min}}(1 - w_0)\Big), \tag{9}$$

$$C = C_{\min} + (C_{\max} - C_{\min})\Big((2 - \mathrm{e}^{\frac{g}{G}\ln(2)})w_0 + \frac{f_j - f_{\min}}{f_{\max} - f_{\min}}(1 - w_0)\Big). \tag{10}$$

其中: K 和 C 分别为自身调整因子及动态学习因子, $C_{\min}$ 和 $C_{\max}$ ($K_{\min}$ 和 $K_{\max}$) 分别为最小和最大学习因子, 大量实验证实, 一般当 $w_0 = 0.5$, $C_{\min} = K_{\min} = 0.2$, $C_{\max} = K_{\max} = 0.6$ 时便可取得优异效果; g 和 G 分别为本代和总迭代次数; $f_{\min}$ 和 $f_{\max}$ 分别代表最小和最大目标函数值 (最小化问题); j 为原个体的学习个体, 在式(4) 和 (7) 中对应为 S_b、S_c、S_f 和雄性中间个体.

由式 (9) 可见, 自身调整部分随着进化的进行会逐渐增大向自身学习的趋势, 而式 (10) 表明, 随着进化的进行向优秀个体的学习倾向逐渐变弱, 且都是自身越好则越多地保留自身成分, 与理论分析相符.

2.2 协作过程中的随机部分的改进

SSO 算法协作过程随机部分的主要作用是增加种群多样性, 但这给搜索带来一定的盲目性, 造成算法不易寻找到更优新解, 从而引起过早收敛和提前停滞. 此外, 该协作过程仅考虑个体自身与优秀个体之间的信息交互, 虽然收敛速度明显提高, 但由于其他个体代表更多的局部搜索信息, 忽视这些信息的影响, 算法勘探新解的能力过低, 必然容易陷入局部最优. 综合以上两方面思想, 本文提出一种新型策略指导协作过程, 在维持种群多样性的前提下, 加强对新解的勘探能力, 提高算法收敛速度.

大量实验证实, 李德毅院士提出的云模型具有稳定倾向性的特点[4], 所产生的云滴以较大概率在自身位置附近变化, 以较小概率过多偏离自身位置. 因此, 与随机扰动方式相比, 利用云模型对个体进行扰动, 同样能够达到维持种群多样性的目的, 最重要的是, 可在一定程度上保留进化方向, 从而降低搜索盲目性.

在算法进化前期, 一般种群多样性比较丰富, 单靠自身位置之间的学习就可保证良好的种群多样性, 而在进化后期, 种群多样性较差, 希望增加额外操作补充种群多样性, 降低算法陷入局部最优的概率. 基于上述考虑, 随着进化的进行, 个体逐渐增大按云模型进行扰动的维数. 与每代个体各维都加入扰动变化相比, 该方式同样能够满足算法对种群多样性的要求, 还可降低计算开销.

大量文献和实验证实, 人工蜂群算法 (ABC)[5] 和差分进化算法 (DE)[6] 都具有强大的搜索能力, 其根本原因分析如下: 新个体是由种群中两个体之间的加权差向量加至基向量上产生的, 而这些个体都是从父代种群中随机选取, 因此父代个体之间存在多种组合方式, 保证了这类进化算法具有良好的种群多样性; 随着进化的进行, 个体逐渐优异, 与在基向量上直接附加一个随机偏差扰动的方式相比, 这种加权差向量的方式具有向其余个体学习的能力, 即在保证种群多样性的同时, 具有更优的勘探能力和收敛速度. 鉴于此, 本文尝试将协作过程中加入 ABC 算法的搜索方式, 进一步增强算法的勘探搜索能力.

基于上述考虑, 这里以雄性种群的协作过程改进为例说明此项改进的具体过程.

Step 1 更改协作过程中的个体更新公式, 产生新个体, 即

$$M_i^{k+1} = \begin{cases} K_1 M_i^k + \alpha C_1(S_f - M_i^k) + \\ (-1 + 2\mathrm{rand})(M_i^k - M_j^k),\ w_{N_{f+i}} \geqslant w_{N_{f+m}}; \\ K_2 M_i^k + \alpha C_2\Big(\dfrac{\sum\limits_{h=1}^{N_m} M_h^k w_{N_{f+h}}}{\sum\limits_{h=1}^{N_m} w_{N_{f+h}}} - M_i^k\Big),\ \text{else}. \end{cases} \tag{11}$$

其中: K_1,K_2,C_1,C_2 分别为按照式 (9) 和 (10) 计算的动态学习因子, j 为随机选择的与 i 不同的个体.

Step 2 确定参与云模型扰动的维数数目, 并按随机方式选择需扰动的具体对应维, 即

$$W_S = \left\lfloor D\frac{g}{G} \right\rfloor, \tag{12}$$

其中 W_S 为选择需要扰动的维数.

Step 3 对于新个体 X_i 选择的待扰动的维数 k, 即 $X_i(k)$, 利用云模型 (输入分别为 $E_x = X_i(k)$, $E_n = 0.5$, $E_h = 0.005$) 进行扰动产生相应的新个体 $X_i(k)'$, 有

$$X_i(k)' = \frac{1}{N}\sum_{i=1}^{N} \mathrm{Drop}_i. \tag{13}$$

其中: $\mathrm{Drop}_i = G(E_x, G(E_n, E_h))$, $G(a, b)$ 表示生成以 a 为期望值、b 为标准差的正态随机数.

2.3 DSSO 算法的实现流程

综合以上对 SSO 算法的两个改进措施, 本文提出的基于动态学习策略的群集蜘蛛优化算法 (DSSO) 的具体步骤如下.

Step 1: 初始化参数 PF, $C_{\min}$, $C_{\max}$, w_0, N_f, N_m, r, 随机产生雄性及雌性蜘蛛初始种群;

Step 2: 雌性种群按照上述动态学习策略完成协作过程, 产生新的雌性种群, 根据适应度函数值选择新、旧雌性种群中较优部分个体作为迭代雌性种群;

Step 3: 雄性种群按照上述动态学习策略完成协作过程, 产生新的雄性种群, 并选择较优部分雄性个体作为迭代雄性种群;

Step 4: 对支配雄性个体进行婚配行为, 并完成对

种群中最差个体的更新;

Step 5: 判断是否满足终止条件, 满足则输出最优个体, 否则转到 Step 2.

3　仿真实验与结果分析

为验证本文算法的有效性和先进性, 这里进行了一系列仿真实验, 所有实验都在 Intel Core2 Duo CPU P7570 2.26 GHz、2 G 内存、2.27 GHz 主频的计算机上进行, 软件运行环境为 Matlab7.0.

3.1　测试函数及评价标准

选取 10 个具有不同特点的典型测试函数[7-9]进行大量仿真实验, 表 2 给出了这些测试函数, 其中函数 10 的理论最优值为 8.8818e-016, 其余函数的理论最优值都为 0.

表 2　测试函数

函数名称	范围
$f_1(x)=\sum_{i=1}^{D}{x_i}^2$	$[-100,100]^D$
$f_2(x)=\sum_{i=1}^{D}\left(\sum_{i=1}^{j}x_j\right)^2$	$[-100,100]^D$
$f_3(x)=\sum_{i=1}^{D}\lvert x_i\rvert+\prod_{i=1}^{D}x_i\rvert$	$[-100,100]^D$
$f_4(x)=\max_i\{\lvert x_i\rvert\}$	$[-100,100]^D$
$f_5(x)=\sum_{i=1}^{D}\lfloor x_i+0.5\rfloor^2$	$[-100,100]^D$
$f_6(x)=\sum_{i=1}^{D}i{x_i}^4+\mathrm{rand}[0,1)$	$[-1.28,1.28]^D$
$f_7(x)=\sum_{i=1}^{D}[{x_i}^2-10\cos(2\pi x_i)+10]$	$[-5.12,5.12]^D$
$f_8(x)=0.5+\dfrac{\sin^2\left(\sqrt{\sum_{i=1}^{D}x_i}\right)-0.5}{\left(1+0.001\sum_{i=1}^{D}x_i\right)^2}$	$[-5.12,5.12]^D$
$f_9(x)=\dfrac{1}{4000}\sum_{i=1}^{D}{x_i}^2-\prod_{i=1}^{D}\cos\left(\dfrac{x_i}{\sqrt{i}}\right)+1$	$[-600,600]^D$
$f_{10}=-20\exp\left(-0.2\sqrt{\dfrac{1}{n}\sum_{i=1}^{n}{x_i}^2}\right)-\exp\left(\dfrac{1}{n}\sum_{i=1}^{n}\cos 2\pi x_i\right)+20+e$	$[-32,32]^D$

为评价算法性能优劣采用如下几种评价准则[10].

1) 收敛精度: 达到某一函数评价次数而算法得到的最优解精度;

2) 种群多样性: 可用达到某一预设精度时种群适应度的方差;

3) 收敛速度: 两种算法在达到几种统一函数评价次数时对应的适应度比较, 如果在各种情况下, 适应度都更好, 则收敛速度更快;

4) 稳定性: 算法独立运行多次, 各次所得精度的方差;

5) 鲁棒性: 算法多次运行达到预设精度的成功率.

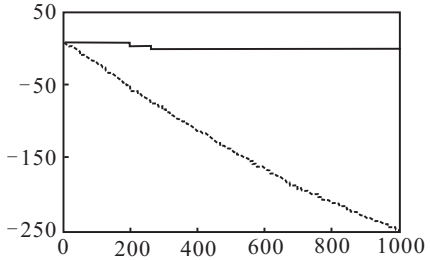
此外, 也可采用进化过程曲线直观观察算法的收敛速度和精度变化.

3.2　各项改进措施的有效性证明

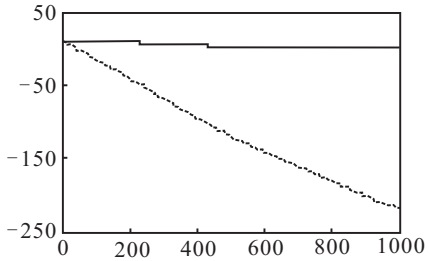
3.2.1　动态学习因子调整的有效性证明

为验证本文提出的动态学习因子调整能够有效改善基本 SSO 算法的收敛速度, 这里只将学习因子改进的 SSO 算法 (称为 LSSO 算法) 与基本 SSO 算法进行对比.

为了对比的公平性, 两种算法的参数设置相同: 种群数目都为 50、各算法迭代次数都为 1 000, 为避免随机性对算法评价造成的不良影响, 需保证初始雌性种群及雄性种群相同. 限于篇幅, 这里仅针对函数 1 和函数 2 进行测试, 变量维数为 30. 两种算法的收敛曲线对比如图 2 所示, 其中横、纵坐标分别为进化代数及每代最优适应度值的对数.



(a)　函数 f_1 的进化曲线



(b)　函数 f_2 的进化曲线

图 2　SSO 算法与 LSSO 算法的性能比较

由图 2 可以看出: 与基本 SSO 算法相比, 对学习因子进行动态调整以后, 收敛精度和收敛速度都有大幅提高, 充分说明了学习因子动态调整的有效性.

3.2.2　随机部分改进的有效性证明

为验证本文对随机部分改进后在保证种群多样性的同时能够提高算法的收敛速度, 这里将随机部分改进的 SSO 算法 (称为 RSSO 算法) 与基本 SSO 算法进行两方面实验: 1) 对比适应度精度为 10^{-6} 时两种算法的种群适应度方差, 比较算法维持种群多样性的能力; 2) 对比相同函数评价次数下, 两种算法的收敛精度评价算法的收敛性能. 限于篇幅这里仅针对函数 1 和函数 2 进行测试. 为避免随机性对算法评价造成的不良影响, 需保证两种算法的初始种群相同.

各算法针对各问题独立运行30次, 统计结果具体如表3所示. 由表3可以看出: 两种方法的适应度方差相差不大, 说明本文对随机部分进行调整后与随机策略一样, 都可以保持相似的种群多样性; 在相同函数评价次数下, 与基本SSO算法相比, 本文方法取得的适应度精度显著提高, 这说明本文方法在提升收敛速度上优势明显. 由于本文算法利用的云模型具有随机性和倾向性的特点, 而随机交叉策略自身具有随机性以及一定的经验学习能力, 这样与SSO算法自身设计的随机方式相比, 改进后方法同样具备良好的多样性, 收敛速度明显加快.

表 3 收敛性能比较

函数	算法	适应度方差	函数评价次数		
			10 000	20 000	30 000
f_1	SSO	8.620 6	642.02	0.757 8	0.267 1
	RSSO	7.821 6	0.408 5	0.000 1	7.03e-8
f_2	SSO	9.192 5e-005	4 735.02	62.11	9.46
	RSSO	1.012 9e-004	30.57	0.006 7	3.35e-6

综合上述两方面实验说明, 与基本SSO算法相比, 本文对随机部分进行改进后, 能在维持种群多样性的同时大幅提升算法收敛速度.

3.3 本文方法的性能分析

由于SSO算法刚被提出, 还未见相关改进算法的研究成果发表, 为充分测试算法性能, 这里将本文算法与基本SSO算法以及函数优化效果较好、较具代表性的IGSA算法[11]、PABC算法[12]、aBBOmDE算法[13]和DMSDELS算法[14]进行对比. 遵循公平性原则, 各算法的初始种群都为50, 函数评价次数都为50 000, 且为避免算法单次运行给算法评价带来的不良影响, 针对每个测试函数, 各算法均分别独立运行30次.

3.3.1 收敛精度指标

为充分评价算法的收敛精度和稳定性, 统计4种方法的平均适应度值及方差, 相关数据如表4所示.

由表4可以看出: 1) 基本SSO算法在上述测试问题上的寻优精度都不甚理想, 且方差较大, 说明该算法本身的寻优能力有待提高, 且很不稳定, 这是由于其自身特性在求解时容易陷入局部最优而造成搜索停滞; DMSDELS算法、aBBOmDE算法、IGSA算法和PABC算法在所有测试问题上的求解精度都较SSO算法有不同程度的提高, 但也仅在 f_5 上PABC算法和DMSDELS算法, f_7 上PABC算法、MSDELS算法和aBBOmDE算法收敛到理论最优解, 在其他问题上都未成功获得理论最优值; 而本文方法除了十分复杂的单模态函数 f_6 以外, 都收敛到理论最优解. 尽管对于 f_6 本文算法没有获得理论最优解, 但是与其

表 4 算法性能对比

函数	算法	平均值	方差	函数	算法	平均值	方差
f_1	DSSO	**0.000 0e+00**	**0.000 0e+00**	f_6	DSSO	$\mathbf{5.208\,9e-44}$	**1.377 7e-43**
	SSO	0.262 2	0.012 3		SSO	2.60e-02	1.98e-02
	IGSA	5.229 1e-017	2.011 2e-017		IGSA	5.386 6e-014	4.037 6e-015
	PABC	1.057 1e-102	7.632 5e-103		PABC	8.345 9e-026	3.198 2e-025
	DMSDELS	4.131 6e-66	5.474 5e-6		DMSDELS	2.190 0e-21	3.691 0e-203
	aBBOmDE	7.312 1e-130	2.571 9e-129		aBBOmDE	1.194 9e-41	2.066 4e-41
f_2	DSSO	**0.000 0e+00**	**0.000 0e+00**	f_7	DSSO	**0.000 0e+00**	**0.000 0e+00**
	SSO	6.254 7	2.273 9		SSO	81.643 4	17.684 4
	IGSA	3.078 6e-014	5.978 2e-014		IGSA	1.983 6e-004	5.599 5e-004
	PABC	4.580 9e-099	3.261 7e-09		PABC	**0.000 0e+00**	**0.000 0e+00**
	DMSDELS	8.182 2e-66	1.343 4e-61		DMSDELS	**0.000 0e+00**	**0.000 0e+00**
	aBBOmDE	4.162 9e-101	1.754 2e-101		aBBOmDE	**0.000 0e+00**	**0.000 0e+00**
f_3	DSSO	**0.000 0e+00**	**0.000 0e+00**	f_8	DSSO	**0.000 0e+00**	**0.000 0e+00**
	SSO	4.978 5e+003	1.089 6e+004		SSO	4.895e-2	0.000 0e+00
	IGSA	3.078 6e-008	5.978 2e-009		IGSA	1.256 6e+04	2.101 3e+01
	PABC	3.081 2e-053	8.182 e-054		PABC	0.009 7	0.000 0e+00
	DMSDELS	3.164 8e-22	2.955 4e-24		DMSDELS	1.209 7e-2	0.000 0e+00
	aBBOmDE	1.471 7e-067	3.676 3e-66		aBBOmDE	2.013 8e-2	0.000 0e+00
f_4	DSSO	**0.000 0e+00**	**0.000 0e+00**	f_9	DSSO	**0.000 0e+00**	**0.000 0e+00**
	SSO	3.742 9	2.440 3		SSO	13.837 7	7.192 1
	IGSA	1.471 7e-07	3.676 3e-14		IGSA	1.065 8e-14	4.789 2e-15
	PABC	7.659 8e-039	2.198 3e-040		PABC	0.602 2	0.036 8
	DMSDELS	2.047 3e-32	4.162 9e-33		DMSDELS	6.661 2e-17	5.438 8e-17
	aBBOmDE	8.616 9e-21	2.810 1e-22		aBBOmDE	9.800e-16	2.442 5e-15
f_5	DSSO	**0.000 0e+00**	**0.000 0e+00**	f_{10}	DSSO	**8.881 8e-16**	**8.881 8e-16**
	SSO	4.371 8e-05	3.186 2e-04		SSO	5.929 1	5.103 5
	IGSA	5.271 8e-017	1.757 1e-017		IGSA	5.481 3e-009	8.749 6e-010
	PABC	**0.000 0e+00**	**0.000 0e+00**		PABC	**8.881 8e-16**	**0.000 0e+00**
	DMSDELS	**0.000 0e+00**	**0.000 0e+00**		DMSDELS	9.414 7e-15	2.842 2e-15
	aBBOmDE	6.698 7e-15	3.016 8e-16		aBBOmDE	6.174 8e-08	3.607 9e-8

他 5 种算法相比, 收敛精度上的优势仍十分明显, 而对于其他函数, 本文算法在收敛精度上明显高于其他 5 种算法, 说明本文方法具有较好的寻优能力. 2) 与其他 5 种方法相比, 本文算法在各测试问题上所获方差都最小, 说明本文算法较其余 5 种算法更为稳定.

3.3.2　收敛速度指标

为充分评价算法的收敛速度, 对比函数评价次数为 10 000、20 000 和 30 000 的收敛精度; 为评价算法的鲁棒性, 对比在 50 000 次函数评价次数内达到预设精度 10^{-8} 的成功率. 相关数据统计结果如表 5 所示.

表 5　算法运行平均时间, 函数评价次数及成功率对比

函数	算法	函数评价次数			成功率
		10 000	20 000	30 000	
f_1	DSSO	1.477 7e-1	4.484 4e-248	0.000 0e+0	100
	SSO	642.02	0.757 8	0.267 1	0
	IGSA	3.230 6e-0	3.131 9e-05	8.746 2e-09	100
	PABC	3.438 2e-1	3.465 5e-38	1.151 7e-60	100
	DMSDELS	1.538 8e-8	1.159e-13	9.514 2e-20	100
	aBBOmDE	1.33e-2	1.984e-46	2.190 3e-70	100
f_2	DSSO	1.195 3e-101	3.627 4e-238	0.000 0e+00	100
	SSO	10 030.89	223.896	10.540 4	0
	IGSA	1.300 5e-0	1.312 5e-05	6.705 6e-06	100
	PABC	8.889 6e-1	3.225 7e-37	1.339 6e-58	100
	DMSDELS	3.514 4e-6	7.491 9e-13	1.870 6e-18	100
	aBBOmDE	2.432 4e-2	1.019 2e-41	8.516 9e-63	100
f_3	DSSO	1.927 8e-4	1.062e-117	7.396e-203	100
	SSO	6.39e+13	9.460 1e+07	8.634 1e+5	0
	IGSA	0.009 1	1.325 1e-003	3.781 1e-04	87
	PABC	1.036 9e-8	7.080 8e-20	5,182 0e-31	100
	DMSDELS	0.002 8	1.347 3e-4	3.694 1e-5	100
	aBBOmDE	1.253e-9	2.812 9e-22	1.192 6e-33	100
f_4	DSSO	1.842 6e-3	1.015e-108	7.069e-194	100
	SSO	12.836 1	8.517 6	5.945 6	0
	IGSA	1.6e-002	ZO	PM	87
	PABC	5.323 6e-7	1.733 6e-14	1.394 2e-22	100
	DMSDELS	7.629 1e-4	4.789 2e-10	1.353 6e-19	100
	aBBOmDE	2.224 4e-1	1.754 2e-5	3.513 2e-11	100
f_5	DSSO	**0.00e+00**	**0.00e+00**	**0.00e+00**	100
	SSO	1.496e+3	9.1e+01	2.1953e-02	0
	IGSA	7.886e+1	6.6090e-03	6.5858e-05	100
	PABC	**0.00e+00**	**0.00e+00**	**0.00e+00**	100
	DMSDELS	**0.00e+00**	**0.00e+00**	**0.00e+00**	100
	aBBOmDE	3.716e+1	1.353 6e-02	1.639 9e-04	100
f_6	DSSO	$\mathbf{8.666\,7e-1}$	4.556 9e-25	**7.626 9e-31**	100
	SSO	0.204 8	0.092 9	0.056 4	0
	IGSA	0.145 9	3.582 9e-05	6.266 2e-09	100
	PABC	7.732 3e-0	5.918 8e-12	3.755 2e-16	100
	DMSDELS	8.461 6e-4	1.280 4e-7	1.639 2e-11	100
	aBBOmDE	0.038	1.640 3e-8	9.017e-18	100
f_7	DSSO	**0.00e+00**	**0.00e+00**	**0.00e+00**	100
	SSO	107.045 9	104.130 8	97.210 4	0
	IGSA	0.156 6	0.025 9	0.021 6	0
	PABC	**0.00e+00**	**0.00e+00**	**0.00e+00**	100
	DMSDELS	**0.00e+00**	**0.00e+00**	**0.00e+00**	100
	aBBOmDE	**0.00e+00**	**0.00E+00**	**0.00e+00**	100
f_8	DSSO	**0.00e+00**	**0.00e+00**	**0.00e+00**	100
	SSO	4.895e-2	4.895e-2	4.895e-2	0
	IGSA	0.037 2	0.037 2	0.037 2	0
	PABC	0.009 7	0.009 7	0.009 7	0
	DMSDELS	1.209 7e-2	1.209 7e-2	1.209 7e-2	0
	aBBOmDE	2.013 8e-2	2.013 8e-2	2.013 8e-2	0
f_9	DSSO	1.682 8e-05	4.038 6e-09	1.493 8e-12	100
	SSO	49.398 0	38.184 1	28.634 9	0
	IGSA	0.106 2	0.010 4	1.694 1e-06	100
	PABC	1.562 7	1.067 3	0.918 2	0
	DMSDELS	1.016 2	6.199 3e-4	1.857 1e-8	100
	aBBOmDE	1.023 9	0.020 6	9.809 2e-4	100
f_{10}	DSSO	**8.881 8e-16**	**8.881 8e-16**	**8.881 8e-16**	100
	SSO	9.487 8	7.366 5	6.707 9	1 000
	IGSA	0.004 2	2.451 8e-03	6.677 0e-05	87
	PABC	1.097e-09	**8.881 8e-16**	**8.881 8e-16**	100
	DMSDELS	8.923 7e-4	1.214 7e-5	1.764 7e-7	100
	aBBOmDE	2.890 1	0.019 1	2.107 0e-4	87

由表5可知, 对于函数 f_5、f_7 和 f_{10}, 本文方法与PABC在相同函数评价次数时均取得相同收敛精度, 即在这3个函数上本文方法表现出与PABC算法相当的收敛速度. 而对于其他测试问题, 在相同函数评价次数下, 与其他5种算法相比, 本文算法均获得更优的收敛精度, 特别对于单模态函数, 本文算法表现出良好的收敛速度. 综上说明, 本文算法的收敛速度上优于其他5种算法. 在鲁棒性上, 对于上述所有测试函数, 基本SSO均未成功收敛到预设精度; IGSA算法在 f_7 和 f_8 上各次运行都未成功收敛, 在 f_3、f_4 和 f_{10} 上也不是每次都能收敛, 对于其余函数都可成功收敛; PABC算法在 f_8 和 f_9 上每次运行都未收敛, 而对于其余函数都成功收敛; DMSDELS算法除 f_8 外都成功收敛; aBBOmDE算法在 f_8 上各次运行都未成功收敛, 在 f_{10} 上也不是每次都能收敛, 对于其余函数都可成功收敛; 而本文算法DSSO对于所有测试函数各次运行均能成功收敛, 说明与其他5种算法相比, 本文方法的鲁棒性更佳.

由上述分析可知, 与基本SSO算法和其余两种较具代表性的优化算法相比, 本文提出的DSSO算法在收敛速度、收敛精度和鲁棒性上均有较大优势, 说明本文算法具有较优较强的跳出局部最优的能力和全局搜索能力, 能够较好平衡算法的开采和勘探能力.

4 结 论

本文提出了一种基于动态学习策略的改进群集蜘蛛优化算法(DSSO), 该算法具有以下两大特点: 1) 群体协作过程的学习因子的动态选择, 平衡算法的搜索能力和勘探能力; 2) 采用随机交叉策略和云模型改进协作个体更新方式, 在维持种群多样性的同时尽量提高搜索速度. 对10个典型标准测试函数进行仿真测试, 结果表明DSSO算法具备强大的跳出局部最优的能力, 收敛速度快、执行力强、收敛精度较优, 在实际应用中具有一定的推广价值.

参考文献(References)

[1] Erik C, Miguel C, Daniel Z, et al. A swarm optimization algorithm inspired in the behavior of the social-spider[J]. Expert Systems with Applications, 2013, 40(1): 6374-6384.

[2] Erik C, Miguel Cienfuegos. A new algorithm inspired in the behavior of social-spider for constrained optimization[J]. Expert Systems with Applications, 2014, 41(1): 412-425.

[3] Karaboga D. An idea based on bee swarm for numerical optimization[R]. Report-TR06. Erciyes University, Engineering Faculty, Computer Engineering Department, 2005.

[4] 张卫光, 何锐, 李德毅, 等. 基于云模型的进化算法[J]. 计算机学报, 2008, 31(7): 1082-1091.
(Zhang W G, He R, Li D Y, et al. An evolutionary algorithm based on cloud model[J]. Chinese J of Computers, 2008, 31(7): 1082-1091.)

[5] Karaboga D, Basturk B. On the performance of artificial bee colony(ABC) algorithm[J]. Applied Soft Computing, 2008, 8(1): 687-697.

[6] Lou Y, Li J L. A differential evolution algorithm based on ordering of individuals[C]. The 2nd Int Conf on Industrial Mechatronics and Automation. Wuhan: IEEE Press, 2010: 105-108.

[7] 蔡之华, 龚文引, Ling Charles-X. 基于进化规划的新型生物地理学优化算法研究[J]. 系统工程理论与实践, 2010, 30(6): 1106-1112.
(Cai Z H, Gong W Y, Ling Charles-X. Research on a novel biogeography-based optimization algorithm based on evolutionary programming[J]. Systems Engineering-Theory & Practice, 2010, 30(6): 1106-1112.)

[8] Saucer T W, Sih V. Optimizing nanophotonic cavity designs with the gravitational search algorithm[J]. Optics Express, 2013, 21(18): 20831-20836.

[9] Tsai H C, Tyan Y Y, Wu Y W, et al. Gravitational particle swarm[J]. Applied Mathematics and Computation, 2013, 219(17): 9106-9117.

[10] 韩俊英, 刘成忠, 王联国. 动态双子群协同进化果蝇优化算法[J]. 模式识别与人工智能, 2013, 26(11): 1057-1066.
(Han J Y, Liu C Z, Wang L G. Dynamic double subgroups cooperative fruit fly optimization algorithm[J]. J of Pattern Recognition and Artifical Intelligence, 2013, 26(11): 1057-1066.)

[11] Sarafrazi S, Nezamabadi-Pour H, Saryazdi S. Disruption: A new operator in gravitational search algorithm[J]. Scientia Iranica, 2011, 18(3): 539-548.

[12] Liu X B, Cai Z X. Artificial bee colony programming made faster[C]. The 5th Int Conf on Natural Computation. Tianjin: IEEE Press, 2009: 154-159.

[13] Lohokare M R, Pattnaik S S, Pranigrahi B K, et al. Accelerated biogeography-basedd optimization with neighborhood search for optimization[J]. Applied Soft Computing, 2013, 13(5): 2318-2342.

[14] 张雪霞, 陈维荣, 戴朝华. 带局部搜索的动态多种群体自适应差分进化算法及函数优化[J]. 电子学报, 2010, 38(8): 1825-1830.
(Zhang X X, Chen W R, Dai C H. Dynamic multi-group self-adaptive differential evolution algorithm with local search for function optimization[J]. Acta Electronica Sinica, 2010, 38(8): 1825-1830.)

(责任编辑: 孙艺红)