应用反向学习策略的群搜索优化算法

汪慎文[1,2] 丁立新[1] 谢大同[1] 舒万能[1] 谢承旺[3] 杨 华[4]
(武汉大学软件工程国家重点实验室 计算机学院 武汉 430072)[1]
(石家庄经济学院信息工程学院 石家庄 050031)[2]
(华东交通大学软件学院 南昌 330013)[3] (贵州师范大学数学与计算机学院 贵阳 550001)[4]

摘 要 群搜索优化算法(Group Search Optimizer,GSO)是一类基于发现者-加入者(Producer-Scrounger,PS)模型的新型群体随机搜索算法。尽管该算法在解决众多问题中表现优越,但其依然面临着早熟和易陷入局部最优的问题,为此,提出了一种基于一般反向学习策略的群搜索优化算法(GOGSO)。该算法利用反向学习策略来产生反向种群,然后对当前种群和反向种群进行精英选择。通过对比实验表明,该方法效果良好。

关键词 群搜索优化算法,反向学习,数值优化
中图法分类号 TP301 **文献标识码** A

Group Search Optimizer Applying Opposition-based Learning

WANG Shen-wen[1,2] DING Li-xin[1] XIE Da-tong[1] SHU Wan-neng[1] XIE Cheng-wang[3] YANG Hua[4]
(State Key Laboratory of Software Engineering, School of Computer, Wuhan University, Wuhan 430072, China)[1]
(School of Information Engineering, Shijiazhuang University of Economics, Shijiazhuang 050031, China)[2]
(School of Software, East China Jiaotong University, Nanchang 330013, China)[3]
(School of Mathematics and Computer Science, Guizhou Normal University, Guiyang 550001, China)[4]

Abstract Group search optimizer(GSO) is a new swarm intelligence algorithms based on the producer-scrounger model. GSO has been shown to yield good performance for solving various optimization problems. However, it tends to suffer from premature convergence and get stuck in local minima. This paper proposed an enhanced GSO algorithm called GOGSO, which employs generalized opposition-based learning to transform the current population into a new opposition population and uses an elite selection mechanism on the two populations. Experiments were conducted on a comprehensive set of benchmark functions. The results show that OGSO obtains promising performance.

Keywords Group search optimizer, Opposition-based learning, Numerical optimization

1 引言

2006年,S. He和Q. H. Wu等[1]提出一种基于发现者-加入者(Producer- Scrounger,PS)模型[2]的群体智能优化算法——群搜索优化算法(Group Search Optimizer,GSO),该算法首次利用了生物学的视觉搜索原理,主要用于解决连续空间优化问题。不同于其他的群体智能优化算法,比如粒子群、蚁群、蜂群和鱼群等算法,群搜索优化算法把种群中的个体分为3类:搜索资源并共享资源信息的发现者、从发现者获取资源信息并掠夺资源的加入者和兼有发现者和加入者角色的游荡者。任何一个个体在某一个时刻要么纯粹是一个发现者,要么纯粹是一个加入者,要么纯粹是一个游荡者,但是任何个体都可以在这3种角色中切换。自GSO提出后的3年时间里,其并没有在国际学术界引起广泛关注。到了2009年10月,算法原创者S. He和Q. H. Wu等[3]在IEEE Transactions on Evolutionary Computation上发表了"Group search optimizer: an optimization algorithm inspired by animal searching behavior"一文。在这篇文章中,S. He等不仅更加系统地阐述了群搜索优化算法的基本原理和数学模型,还将其与遗传算法(GA)、演化规划(EP)、演化策略(ES)、粒子群(PSO)、快速演化规划(FEP)[4]和快速演化策略(FES)[5]进行了仿真实验比较,且对群搜索优化算法中初始化参数对其性能的影响作了初步探讨。这是群搜索优化算法发展史上的又一篇奠基性文章。

虽然群搜索优化算法在多个优化问题中都有成功的应用,但大量的研究表明,标准的群搜索优化算法也存在许多的

到稿日期:2011-10-12 返修日期:2012-02-28 本文受国家自然科学基金(60975050,61070243,61165004),高等学校博士学科点专项科研基金(20070486081),中央高校基本科研业务费专项资金(6081014),河北省科技支撑计划项目(11213587),江西省自然科学基金(20114BAB201025),江西省教育厅科技项目(GJJ12307)资助。

汪慎文(1979-),男,博士生,讲师,主要研究方向为智能计算;丁立新(1967-),男,博士,教授,博士生导师,主要研究方向为智能计算及其理论;谢大同(1977-),男,博士生,主要研究方向为智能计算;舒万能(1981-),男,博士生,主要研究方向为智能计算;谢承旺(1974-),男,博士,主要研究方向为智能计算;杨 华(1974-),男,博士后,副教授,主要研究方向为复杂网络和自然语言处理。

不足和缺陷，如在处理多峰优化问题时易陷入局部极小值；在优化后期收敛速度明显变慢，甚至处于停滞状态，难以获得很好的全局最优解。为了有效地克服以上缺点，房娟艳[6]将Metropolis准则引入发现者的探索模式，使得算法能以一定的概率接受劣解，使其能有效跳出局部极值点，从而强化算法的全局搜索能力。为了解决局部收敛速度较慢的问题，姚健[7]提出了基于二次插值法的群搜索算法(QIGSO)，其借助于二次插值的理论和方法，在群搜索优化算法每一次迭代中利用预测的局部极值点来代替GSO中的发现者搜索的随机点，提高收敛速度。刘峰[8,9]提出一种快速群搜索优化算法(QGSO)，该算法加大游荡者的数目，改进搜索方式和游荡者的位置更新方式。Hai Shen[10]利用子种群和协同演化规划来提高群搜索优化算法性能，提出一种改进的群搜索优化算法(iGSO)并用其来解决机械设计优化问题。

2005年，Tizhoosh教授首次提出了反向学习(Opposition-based learning，OBL)的概念[11]，他认为智能算法都是以随机猜测的值作为初始群体，然后逐代向最优解靠近并最终找到或者接近最优解。所以随机猜测值对算法的影响很大，如果随机猜测值离最优解很近，算法也许能够很快地收敛，但是如果很远甚至相反的话，算法是非常棘手的，也会耗费更多的时间。若在搜索的过程中，同时搜索当前解和反向解，选择较好的解作为猜测解，会大大提高算法的效率[12,13]。事实上，根据概率定理，当前解有50%的概率比它的反向解更远离于最优解[14]。目前，反向学习已经成功应用于差分算法(ODE)[14]、粒子群算法(OPSO)[15]和蚁群算法(OACO)[15]。

针对GSO算法存在的问题和反向学习的优点，本文提出了一种基于反向学习的GSO算法(OGSO)。该算法利用了反向学习策略来反向搜索，同时评估当前解和反向解，从而能提供更多的机会来发掘潜在的较好解，增加群体的多样性，提升算法的全局搜索能力，避免了局部最优。基于当前群体和反向种群的精英选择策略使得较好的解被选入到下一代群体中，从而加速了算法的收敛。

本文第2节介绍GSO算法的基本概念；第3节介绍基于反向学习的GSO算法(OGSO)，这是本文的主要部分；第4节是实验仿真及分析；最后总结全文。

2 群搜索优化算法

群搜索优化算法的思想[1,3]是在每一次迭代中，选择当前最优个体作为唯一的发现者，发现者根据动物的视觉扫描机制在最大转角范围内按照一定的方式调整视觉角度，在可视范围内寻找更好的资源信息。算法随机选择部分个体为加入者，加入者向发现者特定的方向前进。剩余个体被选为游荡者，游荡者在最大转角范围内随意调整角度，然后根据角度在可视范围内调整自己的位置。周而复始，群体中的3类个体相互作用，协同完成任务，从而使其达到最优。

群搜索优化算法中，在一个n维的搜索空间里，第i个个体在第k次迭代的位置可表示为向量$X_i^k\in R^n$，它的搜索角度为$\phi_i^k=(\phi_{i1}^k,\cdots,\phi_{i(n-1)}^k)\in R^{n-1}$，对应的搜索方向为$D_i^k(\phi_i^k)=(d_{i1}^k,\cdots d_{in}^k)\in R^n$，搜索方向按式(1)计算得到。

$$\begin{aligned}d_{i1}^k&=\prod_{q=1}^{n-1}\cos(\varphi_{iq}^k)\\ d_{ij}^k&=\sin(\varphi_{i(j-1)}^k)\prod_{q=j}^{n-1}\cos(\varphi_{iq}^k)\\ d_{in}^k&=\sin(\varphi_{i(n-1)}^k)\end{aligned}\tag{1}$$

2.1 发现者(Producer)

每次迭代中，选择适应度最好的个体作为本次迭代中的发现者。发现者在其共享资源的同时，还继续去搜索资源。搜索的策略是在当前位置选择3个不同的角度进行视觉扫描看到3个不同的位置，具体如下：

$$\begin{aligned}X_z&=X_p^k+r_1 l_{\max}D_p^k(\phi^k)\\ X_r&=X_p^k+r_1 l_{\max}D_p^k(\phi^k+r_2\theta_{\max}/2)\\ X_l&=X_p^k+r_1 l_{\max}D_p^k(\phi^k-r_2\theta_{\max}/2)\end{aligned}\tag{2}$$

式中，r_1，r_2均是介于0和1之间的随机数，$l_{\max}$为视觉能扫描到的最大距离，$\theta_{\max}$为视觉能转动的最大角度。

如果3个随机位置优于发现者当前的位置，就更新发现者的位置和角度；反之，发现者保持位置不变并通过式(3)更新角度：

$$\phi^{k+1}=\phi^k+r_2\alpha_{\max}\tag{3}$$

2.2 加入者(Scrounger)

随机选择剩余的部分个体为加入者。大量实验表明，加入者的数量控制在大约4倍于游荡者的数量为最佳。加入者根据发现者共享的信息按照式(4)来更新位置，但是角度不变。

$$X_i^{k+1}=X_i^k+r_3(X_p^k-X_i^k)\tag{4}$$

2.3 游荡者(Ranger)

在全部个体中，发现者和加入者确定后，余下的就都是游荡者。其角度按式(5)进行更新，位置按式(6)进行更新。

$$\phi^{k+1}=\phi^k+r_2\alpha_{\max}\tag{5}$$

$$X_i^{k+1}=X_i^k+ar_1 l_{\max}D_p^k(\phi^{k+1})\tag{6}$$

2.4 GSO算法整体框架

一个标准的GSO由4个基本步骤组成，即初始化、发现者个体更新、加入者个体更新、游荡者个体更新，具体如图1所示。

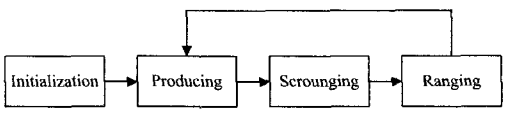

图1 GSO算法主要步骤

标准GSO算法的伪代码如下：

Algorithm 1 标准GSO伪代码

```
Step1  初始化所有个体的位置和角度
Step2  WHILE G<=MaxG
         Step2.1  计算个体的适应度，并进行3种角色分配
         Step2.2  执行 Producing 操作
         Step2.3  执行 Scrounging 操作
         Step2.4  执行 Ranging 操作
       G=G+1
       END WHILE
```

2.5 GSO参数选择

文献[3]给出了如下相关参数的经验参数值：

$\phi^0=(\pi/4,\cdots,\pi/4)$，$\theta_{\max}=\pi/a^2$，$a=\text{round}(\sqrt{n+1})$，$\alpha_{\max}=\theta_{\max}/2$，$l_{\max}=\|U-L\|=\sqrt{\sum_{i=1}^{n}(U_i-L_i)^2}$。

3 应用反向学习策略的GSO算法

反向学习是计算智能中的一个新概念，已经被证明是提

高随机搜索算法的搜索能力的一种有效方法[11,15]。下面给出反向学习的有关定义。

3.1 反向学习基本概念

定义 1(反向数,Opposite Number)[14] 若 $x\in[a,b]$且 $x\in R$,则其方向数 x^* 为:$x^*=a+b-x$。

定义 2(反向点,Opposite Point)[14] 若 $p=(x_1,x_2,\cdots,x_D)$是 D 维空间上的一个点,且 $x_1,x_2,\cdots,x_D\in R$,$x_j\in[a_j,b_j]$,则 p 所对应的反向点 $p^*=(x_1^*,x_2^*,\cdots,x_D^*)$,其中:$x_j^*=a_j+b_j-x_j$。

定义 3(基于反向的优化,Opposition-based Optimization)[14] 若 $p=(x_1,x_2,\cdots,x_D)$是 D 维搜索空间上的一个点(可看作为候选解),其反向点 $p^*=(x_1^*,x_2^*,\cdots,x_D^*)$。假设 $f(\cdot)$是计算候选解适应值的评估函数。如果 $f(p^*)\leqslant f(p)$,则将 p 替换为 p^*。

定义 4(一般反向学习,Generalized Opposition-Based Learning,GOBL)[12,16] 令 $\phi(x)=\Delta-x$,有 $x^*=\Delta-x$,其中 Δ 是一个可计算值,反向解 $x^*\in[\Delta-b,\Delta-a]$。

定义 5(动态的一般反向学习策略,Dynamic GOBL)[12,14] $X_{ij}^*(t)=k(a_j(t)+b_j(t))-X_{ij}(t)$,$a_j(t)=\min(X_{ij}(t))$,$b_j(t)=\max(X_{ij}(t))$,$i=1,2,\cdots,ps$,$j=1,2,\cdots,D$,其中 $X_{ij}(t)$是种群第 i 个解在第 j 维上的分量,$X_{ij}^*(t)$是 $X_{ij}(t)$对应的反向解,$a_j(t)$和 $b_j(t)$分别为当前搜索区间在第 j 维上的最小值和最大值,ps 为群体大小,D 为问题的维数,t 为演化代数。

根据 k 的取值不同,可以得到如下 3 个动态一般反向学习的模型[12,16]:

(1)$k=1/2$(基于区间对称的一般反向学习模型,DSIGOB);

(2)$k=1$(反向学习模型,DGOB);

(3)$k=r$,$r=$rand(0,1)(基于随机的一般反向学习模型,DR-GOBL)。

3.2 应用反向学习策略的 GSO 算法(GOGSO)

算法思想是如果随机数小于反向学习概率,则执行反向学习,否则运行 GSO 算法。伪代码如下:

Algorihtm 2 GOGSO 伪代码

```
Step1  初始化所有个体的位置和角度
Step2  WHILE 停机条件不满足
         Step2.1  if rand(0,1)<=p then
           Step2.1.1  执行反向学习
         else
           Step2.1.2  执行 GSO 算法
         if end
         Step2.2 G=G+1
       END WHILE
```

反向学习包含 3 个步骤:

(1)更新当前群体 $P(t)$的搜索空间的区间$[a_j(t),b_j(t)]$;

(2)根据动态一般反向学习公式,产生当代种群 $P(G)$对应的反向种群 $GOP(G)$;

(3)从 $P(G)$和 $GOP(G)$中选择 ps 个最好的个体组成新的下一代种群。

4 实验仿真及分析

根据动态一般反向学习的 3 种模型,分别得到 3 种不同的 DGOGSO 算法:DSIGOGSO,DGOGSO 和 DRGOGSO。针对不同的 GOGSO 算法,进行了数值仿真实验以判断哪一种模型更适合于 GSO 算法。

本文选择其中 13 个有代表性的经典基准函数来测试算法的有效性,并与标准 GSO 算法进行比较。关于 13 个 Benchmark 函数的详细描述,参见文献[3,4]。

所有的实验均在操作系统为 Windows 7、双核 3.16GHz 的 Intel 处理器和 4G 内存、Matlab 2010 的平台上完成。

实验中各算法所用到的参数均相同,分别设置如下:群体大小为 $PopSize=100$,个体维数为 $DimensionSize=30$,停机条件为最大迭代代数 $MaxGen=5000$ 代,算法运行次数为 $RunTimes=50$。

4.1 一般反向学习概率 P 的研究

表 1 DSIGOGSO 在不同 p 下的平均计算结果

p	f1	f2	f3	f4	f5	f6	f7	f8	f9	f10	f11	f12	f13
0.1	8.2e−24	4.9e−13	7.4e−253	**12.1**	**25.1**	0	2.6e−20	−9236.53	63.3	8.8e−16	0	**2.9e−12**	**6.4e−014**
0.2	8.1e−38	7.3e−18	5.5e−306	**12.1**	**25.0**	0	2.0e−19	−9834.79	18.9	8.8e−16	0	**2.2e−11**	**1.8e−013**
0.3	1.9e−47	6.3e−25	8.6e−020	46.4	25.2	0	**1.4e−20**	−9354.96	109	19.922	0	3.7e−10	1.5e−011
0.4	1.3e−54	4.3e−31	3.3e−192	23.3	25.3	0	2.1e−17	−9834.79	4.94	3.3e−09	0	3.4e−09	7.2e−010
0.5	**5.9e−61**	5.2e−37	9.5e−154	33.5	25.5	0	**8.4e−19**	**−10666.9**	0	19.914	0	1.5e−07	1.4e−008
0.6	**5.0e−62**	4.6e−41	1.2e−030	47.6	25.5	0	**2.0e−19**	**−10070.1**	0	4.4e−15	0	1.6e−4	1.5e−007
0.7	**5.5e−58**	2.5e−45	1.1e−141	35.7	25.7	0	**5.7e−19**	**−10426.7**	0	**8.8e−16**	0	7.2e−3	6.0e−005
0.8	1.6e−41	**6.9e−52**	0	43.3	26.1	0	**2.3e−19**	**−10547.9**	**0.05**	**8.8e−16**	0	2.9	0.01225
0.9	2.5e−23	**9.2e−63**	0	67.8	26.6	0	**6.2e−20**	**−10039.1**	0	**8.8e−16**	0	3.1	2.97963

表 2 DGOGSO 在不同 p 下的平均计算结果

p	f1	f2	f3	f4	f5	f6	f7	f8	f9	f10	f11	f12	f13
0.1	1.6e−16	**3.e−12**	999.84	14.7	15.4	0	**3.5e−23**	−9473	**49.5**	1.0e−8	2.1e−15	9.5e−14	3.0e−017
0.2	3.7e−25	6.e−11	2252.5	34.3	16.3	0	8.5e−22	−11020	79.1	19.8	0	6.8e−18	2.1e−024
0.3	3.7e−33	1.0e−9	**2229.5**	31.7	12.3	0	9.1e−21	−10666	61.1	3.2e−14	0	3.0e−22	1.0e−030
0.4	7.8e−39	4.2e−8	2694.0	38.9	20.4	0	7.8e−21	−9830.2	138	7.9e−15	0	**1.6e−25**	**1.3e−032**
0.5	**8.7e−44**	7.1e−7	2346.8	31.2	20.2	0	3.0e−19	−10307	227	7.9e−15	0	5.4e−17	**1.3e−032**
0.6	**9.4e−47**	2.2e−5	4873.0	36.7	21.6	0	2.2e−18	−9951.7	283	19.8	0	2.5e−5	**1.3e−032**
0.7	**1.8e−47**	7.8e−4	6674.8	49.5	23.5	0	6.2e−18	−9951.7	373	19.8	0	2976.4	2.3e−014
0.8	**5.0e−43**	1.7e−2	8846.7	65.3	24.8	0	4.2e−20	**−10667**	423	**2.8e−5**	0	1.9e+4	43.0
0.9	7.2e−8	2.3e−1	3910.1	52.2	2.e3	1	4.7e−18	−10527	956	0.17	7.3e−3	1.8e+4	4.0e+6

表1—表3给出了DSIGOGSO、DGOGSO和DRGOGSO在不同 p 下运行50次的平均计算结果。当 $p=1$ 时，算法只执行一般反向学习，而不执行GSO算法。尽管没有列举出相关的实验数据，我们也可想而知，算法效果很差，这间接说明反向学习只有与其他算法混合后才可以使用，不能单独使用。当 $p=0$ 时，算法几乎等同于标准GSO算法（除了群体初始化外），所以，没有列举出相关的实验数据。

表3 DRGOGSO在不同 p 下的平均计算结果

p	f1	f2	f3	f4	f5	f6	f7	f8	f9	f10	f11	f12	f13
0.1	1.7e−22	9.5e−12	6.6e−1	**6.5**	**24.9**	**0**	41040	−9118.09	108.8	19.928	**0**	**2.8e−12**	**2.7e−13**
0.2	5.8e−34	8.1e−15	7.3e−3	**12.4**	**25.1**	**0**	3.5e−21	−10190.1	201.9	19.918	**0**	**1.1e−10**	**6.3e−13**
0.3	2.6e−44	1.5e−21	3.4e−3	**27.1**	**25.1**	**0**	3.1e−21	−10076.2	169.4	7.9e−15	0	3.7e−9	2.3e−11
0.4	1.1e−52	1.1e−27	3.8e−3	26.4	25.1	0	**2.7e−24**	−10425.5	170.2	7.9e−15	0	7.2e−9	3.8e−10
0.5	**2.0e−57**	2.5e−33	4.8e−3	28.4	25.5	0	5.6e−23	−10073.2	300.6	7.9e−15	0	2.3e−6	1.2e−8
0.6	**1.2e−60**	**1.6e−37**	2.0e−1	32.5	25.7	0	4.0e−18	−10666.9	91.07	7.9e−15	0	1.4e−4	1.4e−6
0.7	**2.2e−58**	**2.8e−41**	72.1	40.7	25.8	0	1.2e−18	−10668.4	520.8	7.9e−15	0	1.3e−3	3.5e−5
0.8	9.5e−43	**3.0e−42**	9.9e3	65.5	26.3	0	1.1e−21	−10436.5	521.7	4.4e−15	0	1.3e−2	8.e−3
0.9	1.3e−19	6.3e−35	2.0e4	54.5	26.9	0	1.1e−21	−9924.84	1065	2.5e−06	4.6e−15	1.2	2.8e−1

从表中可以看出，较大的 p 使得3个算法在f1—f3函数上获得了较好的计算结果，较小的 p 使得3个算法在f11—f13上取得了较好的结果。

我们很难找到一个使得3个算法都获得最佳性能的最佳概率 p。事实上，根据不同的问题来设置 p 的值能获得更好的性能，这需要试验经验来调节。

4.2 DSIGOGSO、DGOGSO、DRGOGSO和GSO的性能比较

表4给出了DSIGOGSO、DGOGSO、DRGOGSO和GSO在13个Benchmark函数上的50次平均计算结果。

DGOGSO同GSO相比，两种算法在f3—f5上都没有任何优势；在f1，f11，f12和f13上，DGOGSO比GSO有明显的优势；在f7，f8和f10上，DGOGSO比GSO有一定的优势。在DGOGSO中（$k=1$），通过反向学习产生的反向解和当前解都在同一个搜索区间。若当前搜索区间不包含全局最优解，则产生的反向群体所形成的搜索区间也不会包含全局最优解。

表4 DGOGSO、DSIGOGSO、DRGOGSO和GSO的性能比较

p	f1	f2	f3	f4	f5	f6	f7	f8	f9	f10	f11	f12	f13
GSO	7.0e−15	4.3e−13	1026.1	20.4	10.7	0	3.6e−18	−9593.3	17.8	19.9	2.5e−13	9.7e−10	1.2e−10
DGOGSO	1.8e−47	3.e−12	2229.5	14.7	15.4	0	3.5e−23	−11020.	49.5	7.9e−15	0	1.6e−25	1.3e−032
DSIGOGSO	**5.0e−62**	**9.2e−63**	0	12.1	25.0	0	1.4e−20	−10666.9	0	**8.8e−16**	0	2.9e−12	6.4e−014
DRGOGSO	1.2e−60	3.0e−42	3.4e−3	6.5	24.9	0	2.7e−24	−10668.4	91.07	4.4e−15	0	2.8e−12	2.7e−13

DRGOGSO同GSO相比，两种算法在f4，f5，f8和f9上都没有任何优势；在f1，f2和f11上，DRGOGSO比GSO有明显的优势；在f3，f7，f10，f12和f13上，DGOGSO比GSO有一定的优势。但对于单峰函数，DRGOGSO比DGOGSO有优势，对于多峰函数，DGOGSO比DRGOGSO效果要好。在DRGOGSO中（$k=$ random），通过反向学习产生的反向解与当前解都不在同一个搜索区间，DRGOGSO的搜索范围更广、更灵活，不同的 k 值可以在不同的搜索区间中产生不同的反向解。更为灵活的随机反向学习模型更适合单峰问题，更有机会找到更好的解，但是过于灵活会导致多峰问题容易跳出最佳搜索范围而适得其反。

DSIGOGSO同GSO相比，两种算法在f4和f5上效果都比较差，但是在其他函数上DSIGOGSO较有优势，特别是在f1，f2，f3和f9上。在这4种算法中，DSIGOGSO在大部分测试问题上都优于其他3种算法，特别在函数f3和f9上，其他3种算法都表现得很一般，DSIGOGSO却能找到全局最优值。在DSIGOGSO中，尽管通过反向学习产生的反向解与当前解都不在同一个搜索区间，但是反向解的跳跃范围却没有DRGOGSO的灵活，折中的方案取得了意想不到的效果。

结束语 针对群搜索优化算法的局限性，提出了基于一般反向学习模型的群搜索优化算法（GOGSO）。利用一般反向学习策略来初始化群体和在迭代过程中产生反向解通过同时评估当前解和反向解来提供更多的机会发掘潜在的较好解，增加群体的多样性，提高算法的全局搜索能力。通过3种不同的GOGSO算法和标准GSO在13个测试函数上的运算比较表明，基于区间对称的一般反向学习模型更有利于提高群搜索算法的性能。

参 考 文 献

[1] He S, Wu Q H, Saunders J R. A Novel Group Search Optimization Inspired By Animal Behavioral Ecology [C]// IEEE Congress on Evolutionary Computation. 2006: 1272-1278

[2] Vickery W L, Giraldeau L A, Templeton J J, et al. Producers, scroungers, and group foraging[J]. American Naturalist, 1991, 137(6): 847-863

[3] He S, Wu Q H, Saunders J R. Group Search Optimizer: An Optimization Algorithm Inspired by Animal Searching Behavior[J]. IEEE Transactions on Evolutionary Computation, 2009, 13(5): 973-990

[4] Yao X, Liu Y, Liu G. Evolutionary programming made faster [J]. IEEE Transactions on Evolutionary Computation, 1999, 3 (2): 82-102

[5] Yao X, Liu Y. Fast evolution strategies[M]. Evolutionary Programming VI, Berlin, Germany: Springer-Verlag, 1997: 151-161

[6] 房娟艳. 混合群搜索优化算法及其应用研究[D]. 太原：太原科技大学，2010

[7] 姚健. 群搜索算法与二次插值法的混合算法及其应用研究[D]. 太原：太原科技大学，2010

[8] 刘峰，覃广，李丽娟. 快速群搜索优化算法及其应用研究[J]. 工程力学，2010，27(7)：38-44

[9] Qin G, Liu F, Li L, et al. A Quick Group Search Optimizer and

Its Application to the Optimal Design of Double Layer Grid Shells[C]// The 2nd International ISCM Symposium and The 12th International EPMESC Conference. 2009

[10] Shen H, Zhu Y, Niu B, et al. An improved group search optimizer for mechanical design optimization problems[J]. Progress in Natural Science, 2009, 19(1): 91-97

[11] Tizhoosh H. Opposition-based learning: A new scheme for machine intelligence[C]// Proceedings of the International Conference on Computational Intelligence for Modeling Control and Automation. 2005: 695-701

[12] 王晖. 区域变换搜索的智能算法研究[D]. 武汉: 武汉大学, 2011

[13] 汪靖. 群体智能优化算法及其在GPU上的并行化研究[D]. 武汉: 武汉大学, 2011

[14] Rahnamayan S, Tizhoosh H R, Salama M M. Opposition-based differential evolutiona [J]. IEEE Transactions on Evolutionary Computation, 2008, 12(1): 64-79

[15] Wang Hui, Liu Y, Zeng S Y, et al. Opposition-based Particle Swarm Algorithm With Cauchy Mutation[C]// Proc. Congr. Evol. Comput. 2007: 4750-4756

[16] Tizhoosh H, Malisia A R. Applying Opposition-based ideas to the ant colony system[C]// Proceedings of IEEE Swarm Intelligence Symposium. 2007: 79-87

[17] Wang H, Wu Z, Liu Y, et al. Space transformation search: a new evolutionary technique[C]// Proceedings of the first ACM/SIGEVO Summit on Genetic and Evolutionary Computation. 2009: 537-544

(上接第 156 页)

4 模型系数修正

4.1 存储过程

存储过程是数据库的一个重要对象，是一组能完成特定功能的 SQL 语句集合。用户建立存储过程，在其中声明变量，给出参数来执行它，可以设置输出参数或者使用日志显示执行结果。PL/SQL 是 ORACLE 对标准数据库语言的扩展，ORACLE 公司已经将其整合到 ORACLE 服务器和其他工具中，近几年，PL/SQL 在开发人员和 DBA 中得到广泛的使用。存储过程在 PL/SQL 中是一个程序块，可以对数据库进行特定操作。

4.2 评价标准

本研究采用平均误差(average error; mean error)作为在系数修正过程中的评价指标，也就是选择使项目开发估算成本与实际成本平均误差最小的一组参数值作为下一步成本估算的模型系数。

$$Error_i = |E_i - \hat{E}|, ME = \frac{1}{n}\sum_{i=1}^{n} Error_i$$

式中，E_i 是软件开发项目的实际成本，$\hat{E}$是软件开发项目的估算成本，ME 就是 n 个软件开发项目的平均误差。

4.3 修正策略

模型系数修正过程使用两个表，分别是 BEST_FACTOR_HISTORY_TABLE(历史最佳系数表)，储存最佳的模型系数；PROJECT_COST_DATA_TABLE(项目成本数据表)，储存历史项目的成本、代码行等。

本研究开发出 3 个存储过程，通过手动或者存储过程管理软件先后执行，使规模系数动态地适应该软件开发项目的开发环境。图 2 是这 3 个存储过程的工作原理。

$$ME = \min \frac{1}{n}\sum_{i=1}^{n} | TCNC_i + (ER_{LM_i} \times ER_{LM}\,factor + ER_{MM_i} \times ER_{MM}\,factor + ER_{NM_i} \times ER_{NM}\,factor) \times CPS - AC_i |$$

式中，$TCNC_i$ 是 COCOMO 模型估算的第 i 个历史项目新开发部分的成本。ER_{LM}，ER_{MM}，ER_{NM} 是软件开发人员估计的项目各部分规模。AC_i 是第 i 个项目的实际成本。存储过程 PR_BEST_FACTOR_CREATE 就是围绕标准规模系数值(90%，40%，20%)在入口参数 CONSTRAINT 的限制范围内变动规模系数的值，得到使历史项目估算成本与实际成本平均误差最小的 $ER_{LM}\,factor$，$ER_{MM}\,factor$，$ER_{NM}\,factor$ 的值，用于存储过程 PR_COST_ESTIMATE 估算新项目成本。

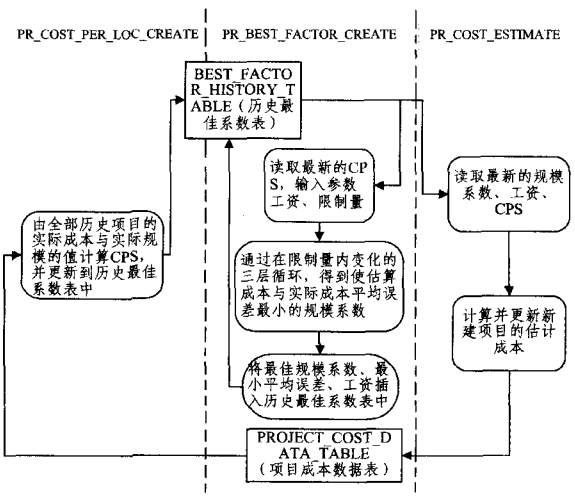

图 2 存储过程工作原理图

结束语 本研究提出非基于软件复用的成本估算模型的复用改造模型来估算使用软件复用技术的软件项目成本。运用存储过程依据项目历史数据对模型中的系数进行修正，从而得到最符合软件开发环境的系数用于下一次的成本估算；并且每一次的估算都是建立在对模型中的系数依据前面所有开发过的项目历史数据进行修正的基础上，因此该模型是一个动态适应的模型。软件开发企业或组织可以很容易地依据本研究的成果进行软件开发成本的初步测定。未来还需要对模型进行进一步的实证分析和验证，以提高模型的可靠性。

参 考 文 献

[1] Boehm B W. Software Engineering Economics [M]. Prentice Hall PTR, 1981

[2] Boehm B W. Software Cost Estimation With Cocomo II [M]. Prentice Hall, 2000

[3] Reifer D J. 软件复用实践[M]. 北京: 机械工业出版社, 2005

[4] Royce W. 软件项目管理[M]. 北京: 机械工业出版社, 2002

[5] 张家浩. 软件项目管理[M]. 北京: 机械工业出版社, 2005

[6] 赵玮. 软件工程经济学[M]. 西安: 西安电子科技大学出版社, 2008

[7] Dagnino A, Srikanth H, Naedele M, et al. A model to evaluate the economic benefits of software components development[J]. IEEE Xplore, 2003, 4: 3792-3797

[8] 吴登生, 等. 软件成本估算的粒子群算法类比模型及自助法推断[J]. 管理科学, 2010, 23(3): 113-120