doi:10.11959/j.issn.1000-436x.2017165

无惯性自适应精英变异反向粒子群优化算法

康岚兰[1,2]，董文永[1]，宋婉娟[1]，李康顺[3]
(1. 武汉大学计算机学院，湖北 武汉 430072；2. 江西理工大学应用科学学院，江西 赣州 341000；
3.华南农业大学信息学院，广东 广州 510642)

摘 要：为解决反向粒子群优化算法计算开销大、易陷入局部最优的不足，提出一种无惯性的自适应精英变异反向粒子群优化算法（NOPSO）。NOPSO 算法在反向学习方法的基础上，广泛获取环境信息，提出一种无惯性的速度（NIV）更新式来引导粒子飞行轨迹，从而有效加快算法的收敛过程。同时，为避免早熟现象的发生，引入了自适应精英变异策略（AEM），该策略在扩大种群搜索范围的同时，帮助粒子跳出局部最优。NIV 与 AEM 这 2 种机制的结合，有效增加了种群多样性，平衡了反向粒子群算法中探索与开发的矛盾。实验结果表明，与主流反向粒子群优化算法相比，NOPSO 算法无论是在计算精度还是计算开销上均具有较强的竞争能力。
关键词：无惯性速度更新式；一般性反向学习；自适应精英变异；粒子群优化
中图分类号：TP181 **文献标识码**：A

Non-inertial opposition-based particle swarm optimization with adaptive elite mutation

KANG Lan-lan[1,2], DONG Wen-yong[1], SONG Wan-juan[1], LI Kang-shun[3]
(1. Computer School, Wuhan University, Wuhan 430072, China;
2. Faculty of Applied Science, Jiangxi University of Science and Technology, Ganzhou 341000, China;
3. College of Information, South China Agricultural University, Guangzhou 510642，China)

Abstract: Non-inertial opposition-based particle swarm optimization with adaptive elite mutation (NOPSO) was proposed to overcome the drawbacks, such as, slow convergence speed, falling into local optimization, of opposition-based particle swarm optimization. In addition to increasing the diversity of population, two mechanisms were introduced to balance the contradiction between exploration and exploitation during its iterations process. The first one was non-inertial velocity (NIV) equation, which aimed to accelerate the process of convergence of the algorithm via better access to and use of environmental information. The second one was adaptive elite mutation strategy (AEM), which aimed to avoid trap into local optimum. Experimental results show NOPSO algorithm has stronger competitive ability compared with opposition-based particle swarm optimizations and its varieties in both calculation accuracy and computation cost.
Key words: non-inertial velocity equation, generalized opposition-based learning, adaptive elite mutation, particle swarm optimization

1 引言

演化算法（EA, evolutionary algorithm）是一类模拟自然进化规律的仿生学算法，广泛应用于大型复杂优化问题的求解[1]。目前，应用比较广泛的演化算法包括遗传算法（GA, genetic algorithm）、粒子群优化（PSO, particle swarm optimization）算法、差分（DE, differential evolution）算法等。其中，PSO 算法[2]是近几年表现比较突出的一种演化算法。该算法概念简单且易于理解和实现，在求解最优化问

收稿日期：2016-11-19；**修回日期**：2017-03-30
通信作者：董文永，duy@whu.deu.cn
基金项目：国家自然科学基金资助项目（No.61170305, No.60873114）；江西省教育厅科学技术研究基金资助项目（No.GJJ161568, No.GJJ151521）
Foundation Items: The National Natural Science Foundation of China (No.61170305, No.60873114), Science and Technology Research in Department of Education of Jiangxi Province (No.GJJ161568, No.GJJ151521)

题，特别是一些复杂优化问题中表现出良好的性能，吸引了大量科研人员对其进一步的研究，并成功应用于多个科学与工程实践领域[3,4]，但与传统优化算法相比，仍存在参数依赖性强、收敛速度较慢且易陷入局部最优等问题。目前，改进方法主要包括参数的自适应设置、算法的融合或混合以及反向学习策略的应用等。Juang 等[5]提出一种自适应模糊 PSO 算法(AFPSO)。在该算法中，惯性权值(w)采用线性递减方式自适应获得，学习因子(c_1, c_2)依据 3 个模糊规则自适应变化。文献[6]依据粒子与全局最优个体间距离式分别对参数 w、c_1、c_2 求导，给出了 3 种参数的确定性变化方向，这种自适应参数控制机制，使算法性能得到有效提高。文献[7]结合人工神经网络方法提出了一种多频振动 PSO 算法。将人工神经网络引入到 PSO 中，有效保持了种群的多样性，防止算法陷入局部最优。2007 年，Wang 等[8]提出了一种基于反向学习的PSO算法(OPSO)，该算法将反向学习(OBL, opposition-based learning)方法引入到 PSO 中，改善了 PSO 的性能。随后，在 OBL 的基础上，提出了一个一般化反向学习(GOBL, generalized OBL)策略，继而给出了一般化的反向学习粒子群算法(GOPSO)，使 OPSO 得到有效改进[9]。2015 年，Karafotias 等[10]对演化算法的发展趋势、参数的控制方法及面临的挑战做出了综合性的讨论。

在 PSO 中，粒子的飞行轨迹除受自身当前极值 *pbest* 与全局极值 *gbest* 影响外，还受到自身运动惯性 w 的影响。为增强种群多样性，彻底消除因 w 的不合理取值对算法性能的影响，本文提出一种无惯性自适应精英变异反向粒子群优化(NOPSO, non-inertial opposition-based particle swarm optimization with adaptive elite mutation)算法。该算法受差分思想启发，取消了传统 PSO 中的惯性项，提出一种新的 NIV 更新式指导粒子下一代的飞行轨迹，新式子充分利用环境信息，增强了种群的多样性，提高了算法的全局勘探能力。同时，算法在反向学习基础上，引入自适应精英变异(AEM, adaptive elite mutation)策略帮助粒子跳出局部最优，从而增强算法的局部开采能力。新算法在 13 个典型测试函数上与多种基于反向学习 PSO 算法展开实验，实验结果表明，本文算法显著提高了 PSO 的收敛速度，在大部分测试函数上性能优于其他对比算法。

2　相关工作

2.1　传统 PSO

PSO 算法是一种基于群体智能的随机搜索算法，由美国 Kennedy 和 Eberhart 等[1]于 1995 年提出，其思想源于对鸟类及鱼群觅食行为的模拟。群体中的个体被看作搜索空间中没有质量和体积的粒子，每个粒子表征一个候选解，具有速度与位置 2 个属性，速度更新式由认知学习、社会学习及惯性运动 3 个部分组成（如图 1 所示）。

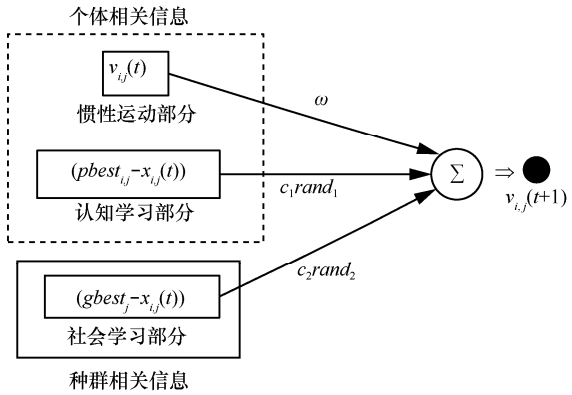


图 1　传统 PSO 速度更新式分析示意

假设第 i 个粒子 $\boldsymbol{x}_i(t)=(x_{i,1},x_{i,2},\cdots,x_{i,D})$ 在第 j 维的速度分量和位置分量分别为 $v_{i,j}$、$x_{i,j}$，则粒子在下一次的迭代中速度更新[11]为

$$v_{i,j}(t+1)=\omega v_{i,j}(t)+c_1rand_1(pbest_{i,j}-x_{i,j}(t))+c_2rand_2(gbest_j-x_{i,j}(t)) \tag{1}$$

其中，$i=1,2,\cdots,N$; $j=1,2,\cdots,D$ (N 是种群规模，D 为空间维度)；ω 是惯性权值(inertia weight)，$\omega\in[0,1]$；c_1 和 c_2 为认知学习因子和社会学习因子，$c_1,c_2\in[0,2]$；$rand_1$ 和 $rand_2$ 是在区间[0, 1]上服从均匀分布的 2 个随机数；*pbest* 为个体最佳，*gbest* 为全局最佳。

粒子在下一次迭代中的位置为

$$x_{i,j}(t+1)=x_{i,j}(t)+v_{i,j}(t+1) \tag{2}$$

2.2　一般化的反向学习策略

反向学习的概念于 2005 年首次由 Tizhoosh[12]提出。OBL 将一个可行解与其反向解同时进行评估，从中选择出较优解进入下一次迭代。2011 年，Wang 等[9]在 OBL 的基础上进一步提出了一般反向学习（GOBL）的概念，具体定义如下。

定义 1 一般化的反向学习[9] (GOBL, generalized opposition-based learning)策略。在 D 维空间，设 $\breve{\boldsymbol{x}}_i = (\breve{x}_{i,1}, \breve{x}_{i,2}, \cdots, \breve{x}_{i,D})$ 是粒子 $\vec{\boldsymbol{x}}_i = (x_{i,1}, x_{i,2}, \cdots, x_{i,D})$ 的反向解，每一维 $\breve{x}_{i,j}$ ($j = 1,2,\cdots,D$)定义为

$$\breve{x}_{i,j} = k(da_j + db_j) - x_{i,j} \tag{3}$$

其中，k 为介于 0 与 1 之间服从均匀分布的随机数，da_j、db_j 为粒子 $\vec{\boldsymbol{x}}_j$ 在第 j 维搜索空间的上、下动态边界，$da_j = \min(x_{i,j})$，$db_j = \max(x_{i,j})$。若 $\breve{x}_{i,j}$ 跳出可行解边界 $[da_j, db_j]$，即 $\vec{\boldsymbol{x}}_j \notin [da_j, db_j]$，则在区间 $[da_j, db_j]$ 内取随机值 $\breve{x}_{i,j} = randn(da_j, db_j)$。此处，$randn(\cdot)$ 为服从正态分布 $N\left(\dfrac{da_j + db_j}{2}, 1\right)$ 的随机数。

3 NOPSO 算法

为充分获取环境信息，保持种群在进化过程中的多样性，本文提出了一种新的粒子运动方程，即 NIV，引导粒子飞行方向，NIV 在提高全局搜索能力的同时显著加快了粒子收敛速度；同时，为避免陷入局部最优及早熟现象的发生，采用自适应精英变异策略（AEM）帮助粒子跳出局部最优。

3.1 NIV

如 2.1 节所述，粒子群中每个粒子在下一代的飞行轨迹由式(1)给出，式(1)由惯性运动、认知学习和社会学习 3 项组成。文献[13]指出，虽然惯性项使种群保持了多样性，但它同时也降低了种群的收敛速度。因此，本文再次对式(1)进行详细剖析（如图 1 所示），将速度更新式所涉及的 3 项分成 2 个部分：第一部分由惯性运动项和认知学习 2 个部分组成，由于它们更多利用了个体相关信息 ($v_{i,j}(t)$, $pbest$)引导粒子飞行轨迹，在此称之为“个体相关信息”部分（如图 1 中的虚线框所示）；第二部分由社会学习部分构成，由于其主要利用全局极值 $gbest$ 来引导粒子飞行方向，称之为“种群相关信息”（如图 1 中的实线框所示）。从图 1 可清晰地看出，传统速度更新式更多是利用个体信息（即自我经验）来引导自身下一步的搜索方向。粒子在下一时刻的运动方向受到很大的惯性作用影响，它表征粒子对个人先验知识的信任。而在一个团队中，个体之间良好的经验交流和相互协作才能更有利于团体目标的达成。因此，本文将传统 PSO 速度更新式中的惯性运动项取消，提出了一种新的速度更新式，来帮助个体获取下一时刻的运动方向。根据信息获取方式的不同，下面给出 3 种不同的 NIV 式：NIV-D、NIV-U 和 NIV-R。

1) Type-I：NIV-D

受 DE 算法启发，随机选择群体中的 2 个个体，利用个体间差异，得

$$vd_{i,j}(t+1) = s(x_{r_1,j}(t) - x_{r_2,j}(t)) + c_1 rand_1(pbest_{i,j} - x_{i,j}(t)) + c_2 rand_2(gbest_j - x_{i,j}(t)) \tag{4}$$

其中，$i \in \{1,2,\cdots,N\}$; $j \in \{1,2,\cdots,D\}$ (N 为种群规模，D 为空间维度)；$s \in (0,1)$ 为差分系数(differential coefficient)，用于控制种群搜索范围；$r_1, r_2 \in [1,N]$ 是 2 个随机整数，且 $r_1 \neq r_2 \neq i$；其他参数定义与式(1)相同。

根据相同的思路，将个体间的相互协作推广到整个种群中，通过获取种群中所有个体信息来扩大协作的范围，由此产生第 2 种 NIV 式，如式(5)所示。

2) Type-II：NIV-U

$$vu_{i,j}(t+1) = s(u(t) - u(t-1)) + c_1 rand_1(pbest_{i,j} - x_{i,j}(t)) + c_2 rand_2(gbest_j - x_{i,j}(t)) \tag{5}$$

其中，$u(t)$和$u(t-1)$ 是种群中所有粒子在 t 时刻与 t−1 时刻所处位置均值。

与 NIV-D 相比，NIV-U 通过广泛获取种群在前 2 个时刻中所有粒子经验来指导当前时刻粒子飞行方向。直觉上，NIV-U 应比 NIV-D 更加快速地收敛到全局最优。后续实验证明了这一结论。值得注意的是，无论是 NIV-D 还是 NIV-U，都无法探知所有环境信息，因此，粒子仍然存在陷入局部最优的可能性。为解决这一问题，NIV 尝试将差分项用一个随机值取代，称之为 NIV-R，如式(6)所示。

3) Type-III：NIV-R

$$vr_{i,j}(t+1) = srand_3 + c_1 rand_1(pbest_{i,j} - x_{i,j}(t)) + c_2 rand_2(gbest_j - x_{i,j}(t)) \tag{6}$$

其中，$rand_3 \sim U[da, db]$ 是服从均匀分布的随机数。da、db 分别是搜索空间的最小、最大边界。

直观上，过多的随机性使 NIV-R 对解空间的搜索较为盲目，但也正是因为这种对搜索空间的随机探索，可能使粒子获得更多的机会逃离局部最优位。后续实验将证明这一推断。

如式(4)～式(6)，NIV 采用一个差分项替代传统 PSO 速度更新式中惯性项，根据信息获取广度的不同，具有 3 种不同的表达形式，在此，统一称之为粒子动量部分。如图 2 所示，新的速度更新式更多利用了种群相关信息（如图 2 中的虚线框所示）指导粒子飞行方向。为比较 3 种不同形式的 NIV 更新式（NIV-D、NIV-U、NIV-R）在进化种群效果上的差异，下面以二维 Sphere 函数 $f(x)=\sum_{i=1}^{2}x_i$ 为例，对同一时刻种群分别采用传统 PSO、NIV-D、NIV-U 以及 NIV-R 这 4 种不同的速度更新式分别进化，结果如图 3 所示。

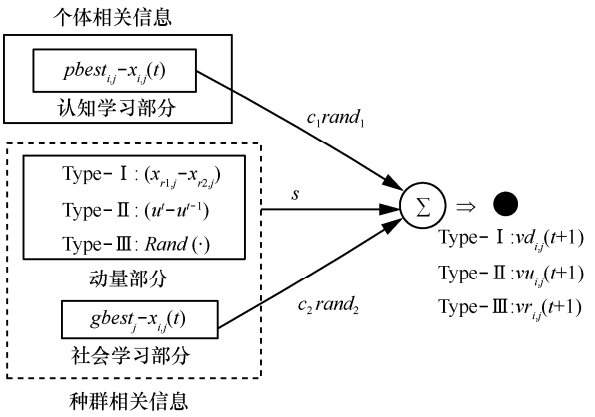


图 2　NIV 更新式分析

图 3(a)展示了初始化种群个体在搜索区域的位置分布，其中，浅色圆点代表初始化个体，种群规模为 10。当种群采用传统 PSO 经过一次迭代后，各个粒子在搜索区域的位置用深色圆点表示，具体位置分布如图 3(b)所示。在经历前 2 代的进化后，粒子分别采用 4 种不同的速度更新式（V、NIV-D、NIV-U、NIV-R）再次进化，进化结果分别如图 3(c)~图 3(f)所示，结果显示：相较于传统 PSO，NIV 更新式使粒子更加聚集；而 3 个不同的 NIV 更新式中，NIV-U 聚集效果更加突出。因此，在后续的实验中，如无特别说明，实验默认采用 NIV-U 更新式。

3.2　自适应精英变异策略（AEM）

为进一步降低粒子陷入局部最优的可能性，本文引入自适应精英变异策略（AEM）帮助粒子跳出局部最优[14]。AEM 将 *gbest* 作为种群精英粒子，在每一代种群进化过程中，根据式(7)和式(8)对 *gbest* 做自适应变异操作。变异后的新个体(*gbest**)适应值若优于原 *gbest* 适应值，则 *gbest**取代原 *gbest*，参与到新一轮的进化过程中。新全局最优个体 *gbest* 在后续进化过程中将吸引其他粒子，从而帮助粒子跳出局部最优位。AEM 变异操作为

$$gbest^{*}=gbest+F(xm) \tag{7}$$

其中，$F(xm)$ 为扰动函数，定义为

$$F(xm)=\frac{1}{\pi}\arctan(xm)+C \tag{8}$$

下面介绍 F 中涉及的关键参数 C 及变量 xm。

1) C 是一个待定常量，它随着适应度标准差 st_d 的不同自适应选取不同变异常量，具体取值为

$$C=\begin{cases}1.5, & st\_d<10^{-2}\\ 1.0, & 10^{-2}\leqslant st\_d<10^{-1}\\ 0.5, & \text{其他}\end{cases} \tag{9}$$

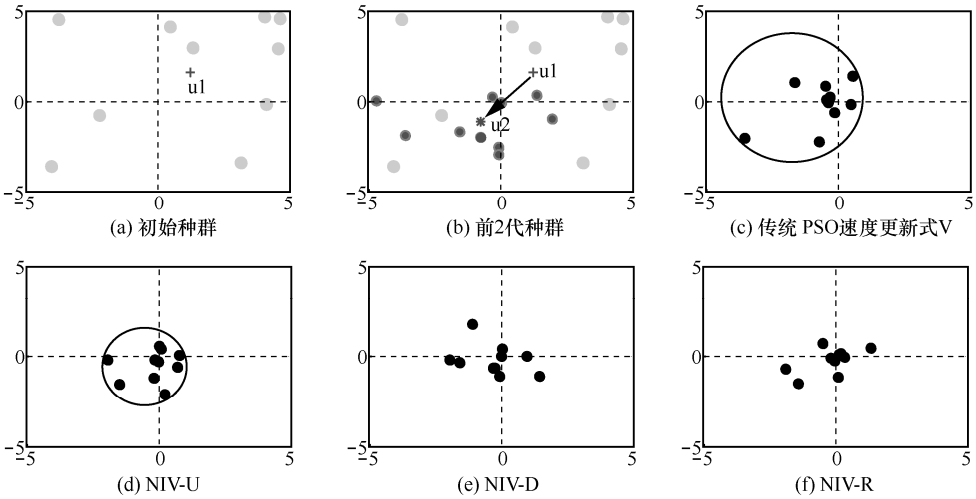


图 3　种群在同一时刻分别采用 4 种不同速度更新式对 Sphere 函数 $f(x)=\sum_{i=1}^{2}x_i$ 进化后的个体分布情况

$$st\_d = \sum_{i=1}^{N} \left| \frac{f_i - f_{gbest}}{f_{gbest}} \right| \tag{10}$$

其中，f_i 为第 i 个个体的适应值，f_{gbest} 为 *gbest* 适应值。当群体越聚集，即 $f_i \to f_{gbest}$ 时，*st_d* 取值越小，表征个体之间的距离越小，从而 C 取值越大，对应 F 函数产生越大的变异量。反之，C 取值越小，F 函数获得的变异量越小。

2) 变量 *xm* 根据式(11)自适应控制变异大小。

$$xm(i) = \exp\left(-\frac{\lambda t}{t_{\max}}\right)\left(1 - \frac{r(i)}{r_{\max}}\right) \tag{11}$$

其中，$i = 1, 2, \cdots, D$ (D 为维度)；λ 是待定常数，根据后续实验分析，本文中 λ 取 10；t 为迭代次数，$t_{\max}$ 是最大迭代次数；$r(i)$ 是所有个体的 *pbest* 的平均值 *avg_pbest* 到 *gbest* 在第 i 维上的距离（定义如式(12)所示）；$r_{\max}$ 是各维度中最大距离。式(13)中，*pbest*[j][i] 是第 j 个粒子在第 i 维上的位置。

$$r(i) = \left| gbest(i) - avg\_pbest(i) \right| \tag{12}$$

$$avg\_pbest(i) = \frac{\sum_{j=1}^{N} pbest[j][i]}{N} \tag{13}$$

对式(11)分析可知，一方面，粒子在种群迭代初期（当 t 较小时）性能较差，变异值较大可对种群造成足够的扰动，从而扩大解空间；但随着迭代的深入，变异值将逐渐减小，从而确保问题平滑收敛到最优值。另一方面，自适应变异在群体极值趋于一致（r 较小）时将获得较大的变异值，增强算法的搜索能力；相反，当群体搜索足够充分时，变异值减小可避免最优值的动荡，从而加快算法收敛速度。

根据上述分析可知，*gbest* 在算法初期通过扰动函数 F 获得较大变异量，从而对搜索空间造成足够的扰动，使算法全局搜索能力得到增强，随着迭代的深入，变异率逐渐减小，从而有效避免最优解的振荡，加快种群的收敛速度。

3.3 无惯性自适应精英变异反向 PSO 算法

本文提出了一种无惯性自适应精英变异反向粒子群优化算法（NOPSO），算法采用 NIV 式对粒子进行速度更新，并结合 AEM 策略对精英粒子采取变异操作。算法 1 用伪代码描述了 NOPSO 的基本步骤。其中，*jr* 为 GOBL 策略使用概率。

算法 1 NOPSO 算法基本步骤

1) 随机初始化 N 个粒子种群 P;

2) 根据式(3)获得 N 个粒子的反向粒子，构成其反向种群 OP;

3) 选取种群 $P \cup OP$ 中适应值较优的 N 个粒子作为初始种群 P;

4) while(未达到终止条件时)do

5) if *rand*(0, 1)<*jr* then

6) 根据式(3)更新 OP，在解空间 $P \cup OP$ 中选取 N 个适应值占优的粒子;

7) 更新 *pbest* 和 *gbest*;

8) else

9) for i=1 to N do

10) 根据 NIV 式获得第 i 个粒子的速度，并根据式(2)更新其位置向量;

11) 更新 *pbest* 和 *gbest*;

12) end for

13) end if

14) for j=1 to D do

15) 对当前获得的 *gbest* 采用 AEM 策略，即式 (7)进行变异操作;

16) end for

17) if *fitness*(*gbest**)< *fitness*(*gbest*) then

18) 用 *gbest** 位取代当前 *gbest* 位;

19) end if

20) end while

对算法 1 分析可知，NOPSO 算法主要涉及初始种群、GOBL 策略、NIV 更新操作、位置更新操作以及 AEM 策略 5 个部分。不难得出，NIV 更新操作、位置更新操作、AEM 策略的计算复杂度均为 $O(ND)$。在前 2 个部分中，初始种群包含随机化种群，GOBL 策略包含边界动态更新，除此之外，两者均包含共同部分：反向种群生成和群体选择机制。显然，随机化种群与反向种群的生成，以及边界动态更新部分的时间复杂度均为 $O(ND)$；对于群体选择机制，排序是其主要操作，计算复杂度为 $O(N^2)$。通常，当维度 D 较低时，种群规模 N 近似于 D，即 $N \approx D$，若维度 D 较高（如 $D \geqslant 100$），则可取 $N<D$，因此，计算复杂度 $O(N^2)$ 可记为 $O(ND)$，从而初始种群和 GOBL 策略部分的计算复杂度为 $O(ND)$。综上，NOPSO 的计算复杂度为 $O(ND)$。

4 数值实验及分析

本文实验将被分成以下 3 个部分。1) 算法性能

测试：将 NOPSO 与基本 PSO 算法以及 4 种基于反向学习策略的 PSO 算法(OBL-based PSO)，即 EOPSO[15]、GOPSO[9]、OVCPSO[16]和 OPSO[8]分别进行性能比较。2) 对 3 类 NIV 式展开性能分析。3) 详细分析了策略 NIV、GOBL 及 AEM 在算法 NOPSO 中发挥的作用，同时，通过对参数 C 的分析，论证了 AEM 策略的有效性。通过对参数 s、λ 的敏感性分析，给出其在算法中的参考值。同时，通过对参数(c_1, c_2, s)正交实验，给出其在 NOPSO 中的最佳参数取值组合。

4.1　实验设置

4.1.1　测试函数

在 Matlab 2012 环境下，算法针对 13 个测试函数展开实验。测试函数按其属性分为单峰函数 f_1~f_7 和多峰函数 f_8~f_{13} 这 2 种，所有测试函数均在零点位取得全局最优值 0。实验均在维数 D=30 下进行，具体测试函数如表 1 所示。

4.1.2　参数设置

为保证比较的公平性，被比较算法参数设置与原文献保持一致，种群规模 N 均取 40。NOPSO 在实验中的默认参数设置如表 2 所示。

4.2　实验结果与分析

4.2.1　OBL-based PSO 算法对比分析

在 6 种 PSO 算法的对比分析中，将每种算法在 13 个测试函数上分别运行 30 次，将迭代 10 000 次后的全局最优值的平均结果记录在表 3 中。对实验结果采用双尾 t 检验进行显著性差异统计分析，显著水平为 0.05，符号“+”“－”“~”分别表示 NOPSO 的性能优于、劣于和相当于对比算法的性能。

表 1　　**数值实验中使用的 13 个测试函数**

	测试函数	搜索空间	函数名称
单峰函数	$f_1(x)=\sum_{i=1}^{D}x_i^2$	$[-100, 100]^D$	Sphere
	$f_2(x)=\sum_{I=1}^{D}(\lfloor x_i+0.5\rfloor)^2$	$[-100, 100]^D$	Step
	$f_3(x)=\sum_{i=1}^{D-1}[100(x_{i+1}-x_i^2)^2+(1-x_i)^2]$	$[-30, 30]^D$	Rosenbrock
	$f_4(x)=\sum_{i=1}^{D}\left(\sum_{j=1}^{i}x_j\right)^2$	$[-100, 100]^D$	Quadric
	$f_5(x)=\sum_{i=1}^{D}\lvert x_i\rvert+\prod_{i=1}^{D}\lvert x_i\rvert$	$[-10, 10]^D$	Schwefel2.22
	$f_6(x)=\sum_{i=1}^{D}(10^6)^{\frac{i-1}{D-1}}x_i^2$	$[-100, 100]^D$	Elliptic
	$f_7(x)=f_6(z), z=xM$	$[-5.12, 5.12]^D$	Rotated Elliptic
多峰函数	$f_8(x)=\sum_{i=1}^{D}[x_i^2-10\cos(2\pi x_i)+10]$	$[-5.12, 5.12]^D$	Rastrigin
	$f_9(x)=-20\exp\left(-0.2\sqrt{\frac{\sum_{i=1}^{D}x_i^2}{D}}\right)-\exp\left(\frac{\sum_{i=1}^{D}\cos(2\pi x_i)}{D}\right)+20+\mathrm{e}$	$[-32, 32]^D$	Ackley
	$f_{10}(x)=\frac{1}{4\,000}\sum_{i=1}^{D}x_i^2-\prod_{i=1}^{D}\cos\left(\frac{x_i}{\sqrt{i}}\right)+1$	$[-600, 600]^D$	Griewank
	$f_{11}(x)=f_8(z), z=xM$	$[-32, 32]^D$	Rotated Rastrigin
	$f_{12}(x)=f_9(z), z=xM$	$[-600, 600]^D$	Rotated Ackley
	$f_{13}(x)=f_{10}(z), z=xM$	$[-32, 32]^D$	Rotated Griewank

表 2　　**NOPSO 参数设置**

c_1	c_2	s	jr	λ	N	D
1.496 18	1.496 18	0.2	0.3	10	40	30

表 3　5 种 OBL-based PSO 算法及 PSO 算法在 13 个测试函数上的结果均值

测试函数		PSO	OPSO	GOPSO	OVCPSO	EOPSO	NOPSO
单峰函数	f_1	1.56×10^{-3}(+)	4.59×10^{-36}(+)	0（最优）(~)	5.00×10^{-1}(+)	0.00(~)	0（最优）
	f_2	7.18×10^{-2}(+)	2.09×10^{-35}(+)	3.02×10^{-321}(+)	6.67×10^{-2}(+)	1.97×10^{-323}(+)	0（最优）
	f_3	1.26(−)	7.18(−)	2.82×10^{1}(+)	8.70×10^{1}(+)	1.57×10^{-24}(−)	3.09
	f_4	5.51×10^{-2}(+)	4.13×10^{4}(+)	1.32×10^{4}(+)	2.94×10^{1}(+)	5.01×10^{-3}(+)	0（最优）
	f_5	-1.48×10^{-3}(+)	-6.51×10^{-12}(+)	-6.98×10^{-162}(+)	−2.49(+)	3.01×10^{-162}(+)	0（最优）
	f_6	8.84(+)	1.43×10^{-32}(+)	4.21×10^{-316}（最优）(+)	1.39×10^{-4}(+)	9.88×10^{-324}(+)	0（最优）
	f_7	4.07×10^{2}(+)	9.82×10^{6}(+)	1.96×10^{6}（最优）(+)	3.06(+)	5.08×10^{2}(+)	0（最优）
多峰函数	f_8	2.06(+)	1.51×10^{1}(+)	0（最优）(~)	7.57×10^{-1}(+)	2.09(+)	0（最优）
	f_9	4.45×10^{-2}(+)	1.85(+)	0（最优）(~)	9.99×10^{-1}(+)	1.86×10^{-1}(+)	0（最优）
	f_{10}	1.14×10^{-3}(+)	3.83×10^{1}(+)	0（最优）(~)	2.05×10^{-2}(+)	1.26×10^{-2}(+)	0（最优）
	f_{11}	2.99(+)	1.51×10^{1}(+)	0（最优）(~)	4.14×10^{1}(+)	4.11(+)	0（最优）
	f_{12}	2.81×10^{-2}(+)	2.98(+)	0（最优）(~)	1.97(+)	3.41×10^{-1}(+)	0（最优）
	f_{13}	7.95×10^{-4}(+)	2.33×10^{-2}(+)	0（最优）(~)	6.14×10^{-1}(+)	1.22×10^{-2}(+)	0（最优）
总计	+	12	12	6	13	11	—
	~	0	0	7	0	1	—
	−	1	1	0	0	1	—

从表 3 中可见，NOPSO 在 6 种算法中除 f_3 外均取得了良好效果。其中，NOPSO 在所有测试函数上均优于 OVCPSO；与 PSO 和 OPSO 的比较中，除在 f_3 中取得相当结果，NOPSO 在其他测试函数上均取得较优解；相反，EOPSO 在 f_3 上取得了最优值，但 NOPSO 在其他 11 个函数上显著优于 EOPSO。需要特别注意的是，GOPSO 与 NOPSO 相同，在 7 个测试函数（f_8~f_{13} 以及 f_1）上取得了相当的最优解，同时，在 f_2 与 f_6 上取得了近似最优解。

为进一步明确 GOPSO 与 NOPSO 性能上的差异，一系列实验在二者之间展开。首先，通过分析 2 种算法在测试函数 f_3 上运行 30 次后的平均值 *Mean*、标准方差 *Std Dev*、最优值 *Best* 及最差值 *Worst* 可知，虽然 NOPSO 与 GOPSO 在迭代初期都出现早熟与停滞现象（如图 4 所示），但 NOPSO 仍较优于 GOPSO（如表 4 所示）。例如，在一次实验中，NOPSO 在 194 次迭代中陷入局部最优值 0.618 5，而 GOPSO 在 264 次迭代中陷入局部最优值 28.82（$28.82 \gg 0.618\,5$，后续实验将验证，利用 NIV-R 将改善 NOPSO 在 f_3 上的性能）。其次，将 2 种算法在求解精度为 10^{-16} 以及最大迭代次数 10 000 次的条件下再次对 13 个测试函数重新实验，记录其运行后适应值 *Fval* 及函数评估次数 *Fes* 列于表 5，由表 5 可知，除 f_3 外，NOPSO 在所有测试函数中的 *Fes* 均明显小于算法 GOPSO。另外，本文将 NOPSO 在上述实验中对每个测试函数在每次运行结束后所经历的迭代次数用箱线（box-whisker）图进行统计，结果如图 5 所示，在对所有测试函数的实验中，30 次运行中所经历的迭代次数都较集中于中位线（图 5 中的方框中的线）。上述 2 个实验结果进一步说明 NOPSO 算法能快速且平稳地收敛到全局最优解。

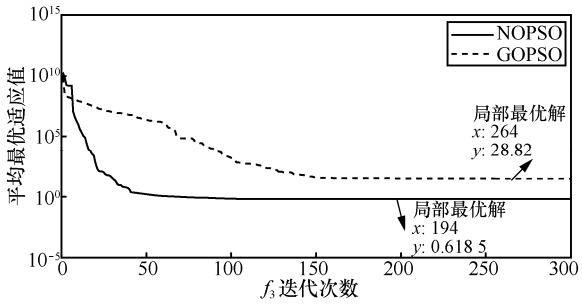


图 4　NOPSO 与 GOPSO 在 f_3 上的收敛趋势比较

表 4　GOPSO 和 NOPSO 算法在测试函数 f_3 上性能比较结果

性能指标	GOPSO	NOPSO
Mean	2.82×10^{1}	3.09（最优）
Std Dev	1.36（最优）	8.61
Best	2.46×10^{1}	2.25×10^{-4}
Worst	2.90×10^{1}	2.89×10^{1}

表 5　GOPSO 和 NOPSO 算法在 13 个测试函数上取值函数评估次数比较

测试函数	GOPSO		NOPSO	
	Fval	*Fes*	*Fval*	*Fes*
f_1	6.64×10^{-17}	49 747	2.65×10^{-17}	6 671（最优）
f_2	0	10 729	0	1 811（最优）
f_3	2.38×10^{1}	400 080	2.09	400 080
f_4	2.48×10^{-4}	254 778	3.66×10^{-17}	8 701（最优）
f_5	8.53×10^{-17}	85 230	2.62×10^{-17}	18 634（最优）
f_6	7.30×10^{-17}	61 003	3.19×10^{-17}	7 748（最优）
f_7	1.96×10^{-6}	170 639	3.31×10^{-17}	8 207（最优）
f_8	0	83 532	0	6 808（最优）
f_9	0	94 337	0	10 999（最优）
f_{10}	0	48 818	0	6 625（最优）
f_{11}	0	217 398	0	6 628（最优）
f_{12}	0	51 078	0	8 993（最优）
f_{13}	0	50 897	0	6 582（最优）

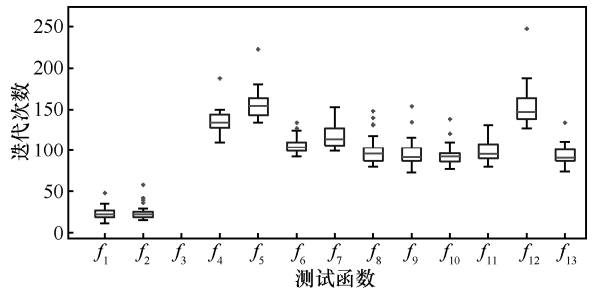


图 5　NOPSO 运行 30 次的迭代次数

4.2.2　不同形式 NIV 性能分析

如 3.1 节所述，NIV 更新式依据环境信息获取方式的不同具有 3 种不同的形式：NIV-D、NIV-U、NIV-R。为研究 3 种 NIV 式对算法产生的影响，接下来将它们分别应用于 NOPSO 中，针对 13 个测试函数展开实验。实验结果将分别记录运行 30 次，最大迭代次数 10 000 次中求解精度为 10^{-16} 的平均值、标准方差、最好值、最差值及函数评估次数如表 6 所示。实验结果表明 NIV-U 在 3 种 NIV 式中表现最突出。较其他 2 种速度更新式，NIV-U 使 NOPSO 以较小的 *Fes* 获得最优值（f_3 除外）。值得注意的是，在测试函数 f_3 上，NIV-R 获得了较好的效果。这说明 NIV-R 虽然在粒子搜索中显得过于盲目，但其随机搜索性使群体在搜索过程中保持了种群的多样性，从而更加有效地避免了粒子陷入局部最优的风险，这也使 NIV-R 在求解一些复杂优化问题时取得了一些意想不到的效果，这一结论正好验证了 3.1 节中对 NIV-R 的分析。

事实上，NIV-D 或 NIV-U 更新式在面对一些带有许多局部极值点，且具有强烈振荡性态的非线性复杂多模态问题时总能表现出良好的性能（NIV-U 速度更快），而 NIV-R 更新式将更加适合于解决一些最优值隐匿于一个极度平滑、狭长等使搜索方向难以确定的优化问题。

4.2.3　策略有效性分析

NOPSO 中主要涉及 3 种策略对反向 PSO 进行改进：NIV、GOBL 和 AEM。为进一步了解 3 种策略对改善算法性能各自产生的影响，本文将上述 3 种策略分别与 PSO 结合，记为 DPSO、PSOL 和 PSOM，与 NOPSO 进行对比实验，实验结果（如表 7 所示）表明：仅一种策略无法有效提高 PSO 性能。例如，GOBL 能加速粒子的收敛，但它也同时使 PSO 更易陷入到局部最优中，NIV 策略也存在相同的问题，而仅使用 AEM 也无法提高算法性能。

为明确 3 种策略在提高 NOPSO 算法性能上各自发挥的作用，对上述实验结果进行详细分析。首先，粒子在搜索空间中利用 NIV 充分获取环境信息；再通过反向策略 GOBL 对信息进行有效提取；最后。利用变异策略 AEM 避免粒子陷入局部最优，使种群获得对空间的连续搜索能力。3 种策略通过相互协作，平衡了局部开发与全局探索之间的矛盾，从而有效提高了算法的收敛速度。

另外，如 3.2 节所述，AEM 变异策略依据粒子聚集程度 *st_d* 的不同，自适应选取待定常数 *C* 值来控制变异量。表 8 通过统计 13 个测试函数中待定常数 *C* 在 30 次实验中自适应取得不同常量值(0.5、 1.0、1.5)时各自的平均概率情况来了解 AEM 在 NOPSO 中的工作机制。当 *C*=0.5 时，式(7)将退化成柯西变异[8]。实验数据表明，相较于柯西变异，AEM 将增加 21%的概率跳出局部最优，虽然它远小于 *C* 取得 0.5 的概率(79%)，但这对于算法是否能成功跳脱局部最优位是非常关键的。同时，注意到在对测试函数 f_3 的实验中，*C* 始终取值 1.5，这一现象反映了粒子群在迭代早期已很快聚集在一起，间接说明了 f_3 易陷入局部最优的原因。

表 6　　NOPSO 算法采用不同 NIV 公式(NIV-D、NIV-U、NIV-R)对性能的影响

测试函数	更新式	*Mean*	*Worst*	*Best*	*Std Dev*	*Fes*
	NIV-D	4.71×10^{-17}	9.77×10^{-17}	2.38×10^{-18}	2.50×10^{-17}	74 700
f_1	NIV-U	3.73×10^{-17}	9.65×10^{-17}	2.99×10^{-20}	2.65×10^{-17}	6 671（最优）
	NIV-R	6.23×10^{-9}	1.79×10^{-8}	4.53×10^{-10}	4.45×10^{-9}	810 081
	NIV-D	0	0	0	0	2 065
f_2	NIV-U	0	0	0	0	1 811（最优）
	NIV-R	0	0	0	0	1 683
	NIV-D	2.89×10^{1}	2.90×10^{1}	2.87×10^{1}	5.54×10^{-2}	810 081
f_3	NIV-U	2.84×10^{1}	2.90×10^{1}	1.54×10^{1}	2.43	810 081
	NIV-R	2.17×10^{-5}（最优）	9.16×10^{-5}（最优）	1.97×10^{-6}（最优）	2.05×10^{-5}（最优）	810 081
	NIV-D	5.25×10^{-17}	9.42×10^{-17}	7.18×10^{-20}	3.07×10^{-17}	11 126
f_4	NIV-U	5.17×10^{-17}	9.97×10^{-17}	3.15×10^{-19}	3.66×10^{-17}	8 701（最优）
	NIV-R	5.59×10^{-4}	5.04×10^{-3}	4.25×10^{-5}	8.64×10^{-4}	810 081
	NIV-D	6.57×10^{-17}	9.83×10^{-17}	1.74×10^{-17}	2.28×10^{-17}	12 713
f_5	NIV-U	6.36×10^{-17}	9.95×10^{-17}	4.71×10^{-18}	2.62×10^{-17}	10 985（最优）
	NIV-R	3.83×10^{-4}	7.59×10^{-4}	7.43×10^{-5}	1.49×10^{-4}	810 081
	NIV-D	4.93×10^{-17}	9.81×10^{-17}	8.20×10^{-19}	3.37×10^{-17}	8 685
f_6	NIV-U	4.49×10^{-17}	9.93×10^{-17}	8.82×10^{-19}	3.19×10^{-17}	7 748（最优）
	NIV-R	3.89×10^{-4}	9.02×10^{-4}	8.42×10^{-5}	2.10×10^{-4}	810 081
	NIV-D	5.29×10^{-17}	1.00×10^{-15}	4.85×10^{-19}	3.17×10^{-17}	9 576
f_7	NIV-U	4.36×10^{-17}	9.52×10^{-17}	2.79×10^{-19}	3.31×10^{-17}	8 207（最优）
	NIV-R	2.51×10^{-4}	7.35×10^{-4}	3.34×10^{-5}	1.75×10^{-4}	810 081
	NIV-D	0	0	0	0	8 156
f_8	NIV-U	0	0	0	0	6 808（最优）
	NIV-R	8.57×10^{-7}	1.87×10^{-6}	1.53×10^{-7}	4.68×10^{-7}	810 081
	NIV-D	0	0	0	0	12 411
f_9	NIV-U	0	0	0	0	10 999（最优）
	NIV-R	4.88×10^{-5}	9.52×10^{-5}	1.10×10^{-5}	1.84×10^{-5}	810 081
	NIV-D	0	0	0	0	7 678
f_{10}	NIV-U	0	0	0	0	6 625（最优）
	NIV-R	3.68×10^{-10}	1.66×10^{-9}	2.34×10^{-11}	3.02×10^{-10}	810 081
	NIV-D	0	0	0	0	8 067
f_{11}	NIV-U	0	0	0	0	6 628（最优）
	NIV-R	1.17×10^{-6}	3.98×10^{-6}	1.69×10^{-7}	9.46×10^{-7}	810 081
	NIV-D	0	0	0	0	12 576
f_{12}	NIV-U	0	0	0	0	10 710（最优）
	NIV-R	4.80×10^{-5}	9.36×10^{-5}	8.19×10^{-6}	2.03×10^{-5}	810 081
	NIV-D	0	0	0	0	7 640
f_{13}	NIV-U	0	0	0	0	6 582（最优）
	NIV-R	2.13×10^{-10}	8.46×10^{-10}	1.04×10^{-11}	2.06×10^{-10}	810 081

表 7　　AEM、GOBL、NIV 3 种策略对 NOPSO 算法各自产生的影响

测试函数	算法	*Mean*	*Worst*	*Best*	*Std Dev*
	PSO	1.77×10^{-2}	5.32×10^{-1}	0	9.55×10^{-2}
	PSOM	8.66×10^{-3}	2.60×10^{-1}	0	4.67×10^{-2}
f_1	PSOL	5.37×10^{3}	1.57×10^{4}	1.64×10^{3}	2.68×10^{3}
	DPSO	7.13×10^{-3}	2.14×10^{-1}	0	3.84×10^{-2}
	NOPSO	0	0	0	0
	PSO	6.67×10^{-2}	2.00	0	3.59×10^{-1}
	PSOM	3.33×10^{-2}	1.00	0	1.80×10^{-1}
f_2	PSOL	4.98×10^{3}	1.32×10^{4}	9.26×10^{2}	2.65×10^{3}
	DPSO	3.33×10^{-2}	1.00	0	1.80×10^{-1}
	NOPSO	0	0	0	0
	PSO	4.81	1.44×10^{2}	0	2.59×10^{1}
	PSOM	4.10	1.23×10^{2}	0	2.21×10^{1}
f_3	PSOL	2.06×10^{6}	9.47×10^{6}	2.13×10^{5}	2.32×10^{6}
	DPSO	2.51	7.54×10^{1}	0	1.35×10^{1}
	NOPSO	2.89×10^{1}	2.90×10^{1}	2.88×10^{1}	4.96×10^{-2}
	PSO	2.25×10^{-1}	6.76	0	1.21
	PSOM	1.96×10^{-1}	5.89	0	1.06
f_4	PSOL	1.83×10^{4}	4.76×10^{4}	3.10×10^{3}	1.05×10^{4}
	DPSO	2.80×10^{-1}	8.39	0	1.51
	NOPSO	0	0	0	0
	PSO	1.09×10^{-1}	3.27	0	5.86×10^{-1}
	PSOM	1.12×10^{-1}	3.35	0	6.02×10^{-1}
f_5	PSOL	2.27×10^{1}	5.39×10^{1}	9.20	1.05×10^{1}
	DPSO	1.67×10^{-1}	5.01	0	8.99×10^{-1}
	NOPSO	0	0	0	0
	PSO	8.84	2.65×10^{2}	0	4.76×10^{1}
	PSOM	7.34×10^{1}	2.20×10^{3}	0	3.95×10^{2}
f_6	PSOL	2.90×10^{7}	7.91×10^{7}	5.94×10^{6}	1.85×10^{7}
	DPSO	5.30	1.59×10^{2}	0	2.86×10^{1}
	NOPSO	0	0	0	0
	PSO	4.07×10^{2}	1.22×10^{4}	0	2.19×10^{3}
	PSOM	5.32×10^{1}	1.60×10^{3}	0	2.87×10^{2}
f_7	PSOL	3.70×10^{7}	1.13×10^{8}	3.74×10^{6}	2.76×10^{7}
	DPSO	3.36×10^{2}	1.01×10^{4}	0	1.81×10^{3}
	NOPSO	0	0	0	0
	PSO	2.93	8.78×10^{1}	0	1.58×10^{1}
	PSOM	3.07	9.22×10^{1}	0	1.66×10^{1}
f_8	PSOL	1.47×10^{2}	2.04×10^{2}	9.80×10^{1}	2.74×10^{1}
	DPSO	2.50	7.51×10^{1}	0	1.35×10^{1}
	NOPSO	0	0	0	0
	PSO	5.47×10^{-2}	1.64	0	2.95×10^{-1}
	PSOM	6.25×10^{-2}	1.88	0	3.37×10^{-1}
f_9	PSOL	1.21×10^{1}	1.63×10^{1}	8.93	1.93
	DPSO	5.63×10^{-2}	1.69	0	3.03×10^{-1}
	NOPSO	0	0	0	0

（续表）

测试函数	算法	*Mean*	*Worst*	*Best*	*Std Dev*
f_{10}	PSO	1.05×10^{-3}	3.14×10^{-2}	0	5.64×10^{-3}
	PSOM	1.62×10^{-3}	4.86×10^{-2}	0	8.72×10^{-3}
	PSOL	3.62×10^{1}	7.14×10^{1}	9.25	1.90×10^{1}
	DPSO	9.11×10^{-4}	2.73×10^{-2}	0	4.91×10^{-3}
	NOPSO	0	0	0	0
f_{11}	PSO	2.47	7.41×10^{1}	0	1.33×10^{1}
	PSOM	2.58	7.74×10^{1}	0	1.39×10^{1}
	PSOL	1.42×10^{2}	1.82×10^{2}	8.07×10^{1}	2.49×10^{1}
	DPSO	3.10	9.29×10^{1}	0	1.67×10^{1}
	NOPSO	0	0	0	0
f_{12}	PSO	4.70×10^{-2}	1.41	0	2.53×10^{-1}
	PSOM	4.78×10^{-2}	1.43	0	2.58×10^{-1}
	PSOL	1.24×10^{1}	1.64×10^{1}	6.83	2.15
	DPSO	5.27×10^{-2}	1.58	0	2.84×10^{-1}
	NOPSO	0	0	0	0
f_{13}	PSO	1.06×10^{-3}	3.16×10^{-2}	0	5.68×10^{-3}
	PSOM	1.08×10^{-3}	3.22×10^{-2}	0	5.79×10^{-3}
	PSOL	3.73×10^{1}	8.90×10^{1}	1.33×10^{1}	1.95×10^{1}
	DPSO	1.03×10^{-3}	3.10×10^{-2}	0	5.56×10^{-3}
	NOPSO	0	0	0	0

表 8 AEM 中参数 *C* 在 13 个测试函数上取值概率

测试函数	*C*=0.5	*C*=1.0	*C*=1.5	变异位占优次数
f_1	0.82	0.11	0.07	6
f_2	0.97	0.03	0.00	5
f_3	0.00	0.00	1.00	4
f_4	0.92	0.05	0.03	2
f_5	0.83	0.07	0.09	2
f_6	0.84	0.08	0.08	5
f_7	0.79	0.09	0.12	6
f_8	0.94	0.04	0.02	1
f_9	0.78	0.10	0.11	3
f_{10}	0.83	0.09	0.08	9
f_{11}	0.92	0.04	0.04	2
f_{12}	0.78	0 .10	0.12	2
f_{13}	0.85	0.09	0.06	8
总体平均概率	0.79	0.07	0.14	4.23

4.2.4 参数敏感性分析

和其他演化算法一样，参数的取值对算法性能往往有很大影响。由于 NOPSO 中很多参数与基本 PSO 相同，在此仅就 NIV 式中新引入的参数 *s* 以及 AEM 策略中参数 λ 进行参数敏感性分析。文献[8]中对反向策略概率 *jr* 的分析指出，当 *jr*=0.3 时，算法取得最佳性能，GOBL 沿用了这一结论。

通过对 13 个测试函数 λ 在 10~60 之间取不同的 6 个值时 NOPSO 收敛过程的观测可知：λ 并不是影响算法性能的关键参数，但由于当 λ=10 时，算法在大多数测试问题中取得了较为稳定的收敛结果，因此，本文仍然建议 λ 取 10。由于篇幅的限制，图 6 仅给出 NOPSO 在测试函数 f_1、f_6、f_8 和 f_{13} 中参数 λ 取 6 个不同值时算法收敛到全局最优值的进化过程。同样的实验在参数 *s* 中展开，通过对 *s* 在 0.1~0.5 区间取不同的 5 个值时算法收敛过程的观测可知（如图 7 所示）：*s* 取值过大将使算法性能显著下降，为使算法达到最佳性能，建议 *s* 的取值范围为 0.1~0.2。

NOPSO 算法中，NIV 式除参数 *s* 外，还包括学习因子 c_1、c_2，参数间相互作用，共同影响着算法的性能。为使 NOPSO 性能达到最佳，在接下来的实验中，本文以(c_1, c_2, *s*)为因子，在 3 个水平(1, 1.5, 2)、(1, 1.5, 2)和(0.1, 0.15, 0.2)上进行 $L_9(3^4)$ 正交实验，并采用 Friedman 检验对实验结果进行分析。表 9 给出了 9 种参数组合对 13 种测试函数在性能比较中的平均秩及排名，当参数组合 (c_1, c_2, *s*)取值为(1, 1, 0.1)时排名第一。

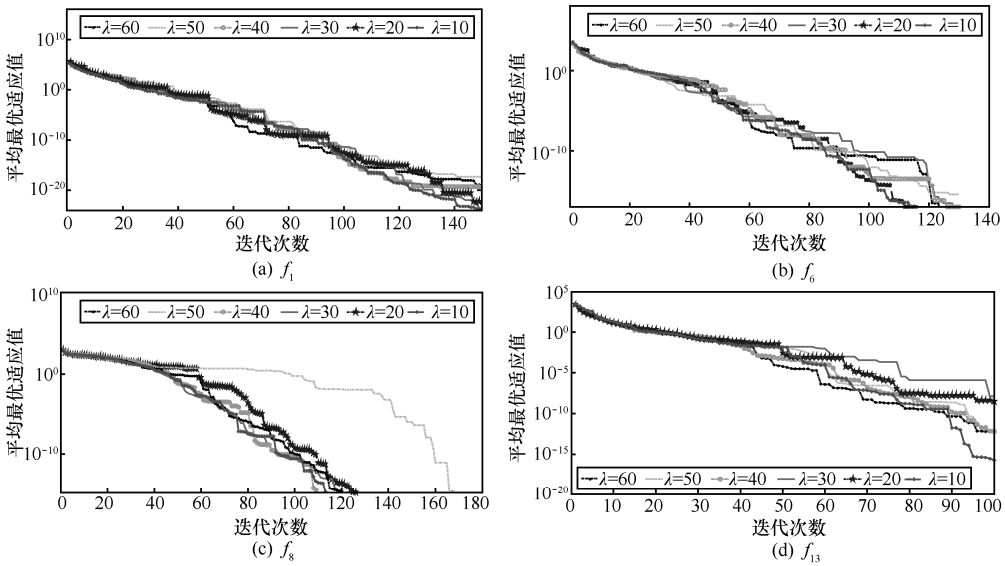


图 6　λ 在不同取值下 NOPSO 全局收敛过程

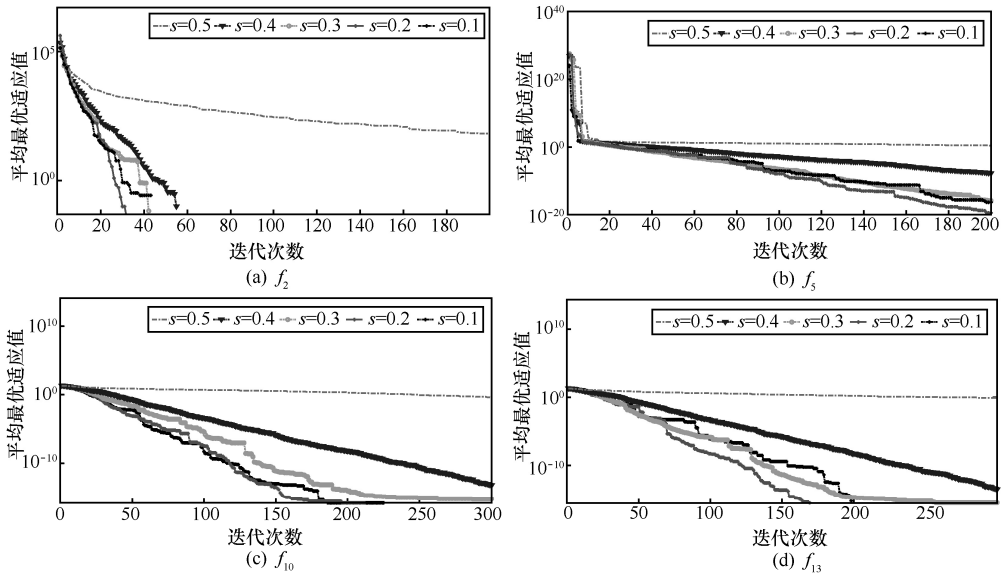


图 7　s 在不同取值下 NOPSO 全局收敛过程

表 9　　参数(c_1, c_2, s)的 $L_9(3^4)$正交实验

组合	c_1	c_2	s	平均秩	排名
1	1.0	1.0	0.10	1.83	1
2	1.0	1.5	0.15	2.25	2
3	1.0	2.0	0.20	5.92	6
4	1.5	1.0	0.15	3.83	3
5	1.5	1.5	0.20	4.08	4
6	1.5	2.0	0.10	7.92	8
7	2.0	1.0	0.20	4.53	5
8	2.0	1.5	0.10	6.75	7
9	2.0	2.0	0.15	9.00	9

5　结束语

本文提出了一种无惯性自适应精英变异反向粒子群优化算法。与传统 PSO 不同，NOPSO 取消速度更新式中的惯性项，采用了一种新的 NIV 更新式引导粒子飞行方向，根据信息获取方式的不同，具有 3 种不同的表达形式，即 NIV-D、NIV-U、NIV-R。NIV 更新式通过对环境信息的充分获取及利用，扩大了粒子的搜索空间。同时，算法利用 GOBL 策略对上述信息进行有效提取，并结合 AEM 策略降低了粒子陷入局部最优的可能性，使种群获得对解空间的连续搜索能力。3 种机制相互协作，

有效平衡了局部开发与全局探索之间的矛盾，从而使算法收敛速度得到显著提高。

算法在 13 个测试函数上进行实验，结果表明 NOPSO 使反向算法在解的精度以及收敛速度上得到了大幅度的提高；同时，本文对 3 种不同形式的 NIV 性能展开深入研究，给出性能分析及适用环境，并通过一系列实验给出了算法最佳参数组合及取值建议。在未来的工作中，将进一步从理论上研究 NIV 工作机制，并将这种机制扩展到其他演化算法中；另外，算法在高维问题及动态优化问题中是否适用及可能存在的问题还需进一步实验。针对动态优化问题的特点，NIV-U 是否仍然在速度上优于 NIV-D 也是未来值得思考的一个问题；最后，如何将算法与实际问题相结合是未来研究工作的重点。

参考文献：

[1] EIBEN A E, SMITH J. From evolutionary computation to the evolution of things[J]. Nature, 2015, 521(7553): 476-482.

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

[3] INBARANI H H, AZAR A T, JOTHI G. Supervised hybrid feature selection based on PSO and rough sets for medical diagnosis[J]. Computer Methods and Programs in Biomedicine, 2014, 113(1): 175-185.

[4] ZAD B B, HASANVAND H, LOBRY J, et al. Optimal reactive power control of DGs for voltage regulation of MV distribution systems using sensitivity analysis method and PSO algorithm[J]. International Journal of Electrical Power and Energy System, 2015, 68: 52-60.

[5] JUANG Y T, TUNG S L, CHIU H C. Adaptive fuzzy particle swarm optimization for global optimization of multimodal functions[J]. Information Sciences, 2011, 181: 4539-4549.

[6] MENG H, TERESA W, WEIR J D. An adaptive particle swarm optimization with multiple adaptive methods[J]. IEEE Transactions on Evolutionary Computation (TEC), 2013, 17(5): 705-720.

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

[8] WANG H, LI H, LIU Y, et al. Opposition-based particle swarm algorithm with Cauchy mutation[C]//IEEE Congress on Evolutionary Computation. 2007: 356-360.

[9] WANG H, WU Z J, RAHNAMAYAN S, et al. Enhancing particle swarm optimization using generalized opposition-based learning[J]. Information Sciences, 2011,181: 4699-4714.

[10] KARAFOTIAS G, HOOGENDOORN M, EIBEN A E. Parameter control in evolutionary algorithms: trends and challenges[J]. IEEE Transactions on Evolutionary Computation, 2015, 19(2): 167-187.

[11] SHI Y, EBERHART R C. A modified particle swarm optimizer[C]// The IEEE Congress on Evolutionary Computation (CEC 1998). 1998: 69-73.

[12] TIZHOOSH H R. Opposition-based learning: a new scheme for machine intelligence[C]//The IEEE International Conference of Intelligent for Modeling, Control and Automation. 2005: 695-701.

[13] SHI Y, EBERHART R C. Empirical study of particle swarm optimization[C]//The IEEE Congress on Evolutionary Computation (CEC). 1999.

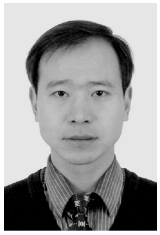
[14] 董文永, 康岚兰, 刘宇航, 等. 带自适应精英扰动及惯性权重的反向粒子群优化算法[J]. 通信学报，2016, 37(12): 1-10.
DONG W Y, KANG L L, LIU Y H, et al. An opposition-based particle swarm optimization with adaptive elite mutation and nonlinear inertia weight[J]. Journal on Communications, 2016, 37(12): 1-10.

[15] 周新宇, 吴志健, 王晖, 等. 一种精英反向学习的粒子群优化算法[J]. 电子学报, 2013, 41(8): 1647-1652.
ZHOU X Y, WU Z J, WANG H, et al. Elite opposition-based particle swarm optimization[J]. Acta Electronica Sinica, 2013, 41(8): 1647-1652.

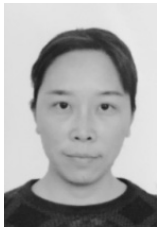
[16] SHAHZAD F, BAIG A R, MASOOD S, et al. Opposition-based particle swarm optimization with velocity clamping (OVCPSO)[J]. Advances in Computational Intell, 2009: 339-348.

作者简介：

康岚兰（1979-），女，江西赣州人，武汉大学博士生，江西理工大学讲师，主要研究方向为演化计算、机器学习等。



董文永（1973-），男，河南南阳人，博士，武汉大学教授、博士生导师，主要研究方向为演化计算、智能仿真优化、系统控制、机器学习等。



宋婉娟（1980-），女，湖北应城人，武汉大学博士生，主要研究方向为图像处理、机器学习等。

李康顺（1962-），男，江西兴国人，博士，华南农业大学教授、博士生导师，主要研究方向为智能计算、多目标优化、视频流图像识别等。