**Pure Mathematics** 理论数学, 2023, 13(3), 732-749
Published Online March 2023 in Hans. https://www.hanspub.org/journal/pm
https://doi.org/10.12677/pm.2023.133077

Hans汉斯

# 一种新的带马尔可夫跳跃的粒子群优化算法

**刘　晨[1]，舒慧生[1*]，阚　秀[2]**

[1]东华大学理学院，上海
[2]上海工程技术大学电子电器工程学院，上海

收稿日期：2023年2月23日；录用日期：2023年3月24日；发布日期：2023年3月31日

## 摘　要

**本文提出了一种新的带有马尔可夫跳跃的PSO算法(MJPSO)，在MJPSO中，通过评估每一代的进化因子，粒子的惯性权重参数和加速度系数均可以根据一个齐次马尔可夫链自适应地调整。设计的新型变异策略，可以帮助粒子根据适当的概率，提高逃离局部优化陷阱的可能性。该策略不仅降低了计算成本，还改进了全局搜索，提高了粒子的全局搜索效率和搜索能力。一系列广泛使用的优化基准实验的结果表明，所开发的MJPSO算法优于现有的六种流行的PSO算法的变体。**

## 关键词

**粒子群优化，马尔科夫链，进化计算**

# A Novel Particle Swarm Optimization Algorithm with Markov Jumping

**Chen Liu[1], Huisheng Shu[1*], Xiu Kan[2]**

[1]College of Science, Donghua University, Shanghai
[2]School of Electronic and Electrical Engineering, Shanghai University of Engineering Science, Shanghai

Received: Feb. 23rd, 2023; accepted: Mar. 24th, 2023; published: Mar. 31st, 2023

## Abstract

**In this paper, a new Particle Swarm Optimization algorithm with Markov Jumping (MJPSO) is proposed, in which both the inertia weight parameter and the acceleration coefficient of a particle can**

*通讯作者。

文章引用: 刘晨, 舒慧生, 阚秀. 一种新的带马尔可夫跳跃的粒子群优化算法[J]. 理论数学, 2023, 13(3): 732-749.
DOI: 10.12677/pm.2023.133077

**be adaptively adjusted according to a chi-square Markov chain by evaluating the evolution factor of each generation. A new variational strategy is designed to help the particles to improve the possibility of escaping the local optimization trap according to the appropriate probability. The strategy not only reduces the computational cost, but also improves the global search and increases the global search efficiency and search capability of the particles. The results of a series of widely used optimization benchmark experiments show that the developed MJPSO algorithm outperforms six existing variants of the popular PSO algorithm.**

## Keywords

**Particle Swarm Optimization, Markov Chain, Evolutionary Computation**



## 1. 引言

优化在计算机科学、运筹学以及其他学科中起着重要的作用，试图在在合理的范围限制内为优化问题找到一个最佳解决方案。随着科学技术的发展与工业社会的进步，优化问题越来越展现出高维、不连续等特征，传统优化方法并不适用于解决此类优化问题。不同于传统搜索，进化算法(EA)作为优化算法的一个重要分支，通过选择、变异和繁殖来模拟个体的进化过程。包括遗传算法(GA)、记忆算法、微分进化(DE)以及其他在内的一些 EAs 近年来受到了广泛的关注[1] [2]。其中，粒子群优化算法(PSO)的搜索性能相当，甚至优于 GAs 等其他 EA，具有更快和更稳定的收敛率。由于容易理解、快速收敛和基于种群的特点，PSO 受到了研究者的高度关注。目前，PSO 算法已广泛应用于神经网络，模糊系统控制，图像处理等许多现实的领域[3]-[8]。

粒子群优化(PSO)是一种基于种群的 EA，由 Kennedy 和 Eberhart 于 1995 年首次提出[9] [10]。开发这种方法的动机是基于对简化的动物社会行为的模拟，如鱼群、鸟群等。粒子群算法从搜索空间中的个体粒子的初始化开始，在每一个时间步中，通过个体最优位置和全局最优位置来调整自身的位置和速度从而找到全局最佳解决方案[9] [11] [12]。但 PSO 存在的问题是对于多模态和高维任务的应用，当单个粒子的最优位置等于全局最优位置时，粒子容易陷入到局部极值点中[13]。PSO 算法虽然提供了全局搜索的可能，但是并不能保证收敛到全局最优点上。因此，许多改进的 PSO 变体在最近几年被开发出来，以克服这个缺陷，其实验结果与原始 PSO 算法相比有了较大的改善。例如，调整惯性权重系数和加速度系数，添加扰动项，时变策略，时延策略，以及基于进化因子和马尔可夫跳的自适应切换，都通过实验验证了有助于粒子在空间进行更彻底的探索和开发[8] [14]-[22]。切换 PSO (SPSO)的速度更新方程具有马尔可夫切换参数，并且在其基础上通过添加时滞和突变概率矩阵的新切换 PSO 算法，也在动态优化问题上展现出各自的性能[23] [24] [25] [26]。

虽然 SPSO 算法在改进过程中省略了逐代计算加速度系数的步骤，但是在多模态函数上求解全局最优值时仍然存在收敛速度较慢，容易陷入局部最优值的问题。为了进一步提高 SPSO 的搜索性能，自然希望惯性权重系数也能够识别或利用进化状态信息，从而增强种群的多样性。马尔可夫链可以依据概率分布，在不同状态间切换或维持原有状态，能够提高粒子全局搜索和局部搜索能力，有效降低由于参数设置不合理而导致的风险。因此，如何合理给出合适的突变策略并设置参数的最佳值是我们需要认真考

虑的核心问题。

我们的目的是希望能够在 SPSO 算法的基础上，开发一种新的 PSO 算法，以缓解过早收敛问题并降低计算复杂度。本文提出了一种改进的带马尔可夫跳的 PSO 算法，称为 MJPSO。文章的创新性主要体现在，改进了 SPSO 中加速度系数由马尔可夫链控制，而惯性权重系数线线性递减的迭代策略，通过评估进化因子划分进化状态，基于一个齐次马尔科夫链，根据进化状态自适应地调整模型的惯性权重参数和加速度系数。因此，粒子能够快速移动到更有潜力的区域，极大地提高搜索效率和加快搜索进程。同时，惯性权重参数和加速度系数不必在每一次迭代中重复计算，从而在一定程度上了降低计算成本。最后，通过对比实验的验证，与一些现有的 PSO 流行变体相比，MJPSO 算法在收敛精度、速度和稳定性上有一定的优势。

本文的其余部分组成如下。传统 PSO 和基于 PSO 的一些变体算法在第二节中给出。提出的带马尔可夫跳的 MJPSO 算法在第三节中给出。第四节与现有各种 PSO 算法的收敛性能比较。最后，第五节总结这篇文章，并给出未来展望。

## 2. 粒子群优化算法

### 2.1. 传统 PSO

PSO 的基本概念是源于鸟类觅食的研究[9]，与其他进化算法类似，PSO 算法也是通过种群中个体间的协作与竞争，实现复杂空间最优解的搜索。PSO 算法就是模拟一群鸟寻找食物的过程，每只鸟就是 PSO 中的粒子，也就是需要求解问题的可能解，这些鸟在寻找食物的过程中，不停的改变自己在空中的飞行位置和速度。PSO 初始化为一群随机粒子(随机解)，然后通过迭代找到最优解。

在 $D$ 维空间中，初始设置为种群数量为 $N$ 的随机粒子。以其中的第 $i$ 个粒子为例，假设在第 $k$ 次迭代时，其速度和位置分别为 $v_i(k)=\left(v_{i1}(k),v_{i2}(k),\cdots,v_{iD}(k)\right)$ 和 $x_i(k)=\left(x_{i1}(k),x_{i2}(k),\cdots,x_{iD}(k)\right)$，之后每个粒子通过两个最优位置来更新自己的速和位置，即自己的历史最优位置 $p_i=\left(p_{i1},p_{i2},\cdots,p_{iD}\right)$ 和全部粒子的历史全局最优位置 $p_g=\left(p_{g1},p_{g2},\cdots,p_{gD}\right)$。则基本的 PSO 算法可以描述如下：

$$\begin{aligned} v_i(k+1)&=\omega v_i(k)+c_1r_1\left(p_i(k)-x_i(k)\right)+c_2r_2\left(p_g(k)-x_i(k)\right)\\ x_i(k+1)&=x_i(k)+v_i(k+1)\end{aligned} \tag{1}$$

这里，$\omega$ 是惯性权重以控制粒子在搜索空间中的探索，$c_1$ 和 $c_2$ 是加速度参数，分别称为认知和社会系数。$r_1$ 和 $r_2$ 是分布在 $[0,1]$ 区间的均匀随机数。为了防止粒子冲出搜索空间并取得较好的效果，粒子位置限制在 $[x_{\min},x_{\max}]$ 范围内，速度限制在 $[v_{\min},v_{\max}]$ 范围内。

### 2.2. PSO 的一些变体算法

标准 PSO 由于其概念的简单性和实现的效率性，被广泛应用以解决实际问题。同时，许多研究人员致力于开发各种变体以提高搜索性能。

一种常见的策略是引入惯性权重[14] [15]，以实现局部搜索和全局搜索之间的适当平衡。此外在[17]中提出了具有线性递减惯性权重 $\omega$ 的 PSO 变体(PSO-LDIW)，展示了惯性权重值较大有利于实现全局搜索，惯性权重值较小更有助于局部搜索微调。惯性权重 $\omega$ 的更新公式定义如下：

$$\omega=(\omega_1-\omega_2)\times\frac{maxiter-iter}{maxiter}+\omega_2 \tag{2}$$

这里，$\omega_1$ 和 $\omega_2$ 分别是惯性权重的初值和终值，$iter$ 是当前迭代次数，$maxiter$ 是最大迭代次数。通常初

值设置为 $\omega_1 = 0.9$，终值设置为 $\omega_2 = 0.4$。

另一方面，加速度系数作为 PSO 算法中的重要参数，通常取值为 2.0。同样，时变策略也可同样应用在加速度系数中，一种具有线性时变加速度系数的 PSO (PSO-TVAC)在[16]中被引入，其中认知系数 $c_1$ 随时间线性递减，社会系数 $c_2$ 随时间线性增加。加速度系数 $c_1$ 和 $c_2$ 的更新公式如下：

$$\begin{aligned} c_1 &= \left(c_{1f} - c_{1i}\right) \times \frac{maxiter - iter}{maxiter} + c_{1i} \\ c_2 &= \left(c_{2f} - c_{2i}\right) \times \frac{maxiter - iter}{maxiter} + c_{2i} \end{aligned} \tag{3}$$

这里，$c_{1i}$ 和 $c_{2i}$ 是加速度系数 $c_1$ 和 $c_2$ 的初始值；$c_{1f}$ 和 $c_{2f}$ 分别是加速系数 $c_1$ 和 $c_2$ 的终值。通常，$c_{1i} = 2.5$，$c_{2i} = 0.5$，$c_{1f} = 0.5$，$c_{2f} = 2.5$。此外，收缩因子被引入到 PSO 中以提高搜索性能[11]，在 PSO-CK 算法中，建议使用 $\omega = 0.729$ 和 $c_1 = c_2 = 1.49$。另外，RPSO 算法[18]利用强度可调的高斯白噪声为加速度系数添加随机扰动，以增强种群的多样性。

此外，还有一类学者研究设计不同类型的拓扑以改进 PSO 的搜索性能。最近，在[19]中引入了一种自适应 PSO 算法，采用进化因子，并在每一代中定义探索、开发、收敛和跳出四种进化状态，通过模糊分类的方法来自动控制惯性权重 $\omega$ 和加速度系数 $c_1$、$c_2$。除此之外，基于马尔科夫链控制进化状态，从而实现加速度系数自动控制的 SPSO 算法在[21]中提出，以提高优化器的搜索能力。SDPSO 算法[24]在 SPSO 算法的基础上为局部和全局最优粒子添加延迟信息，提高粒子的搜索效率和收敛速度。然而，这些 PSO 的变体算法仍然存在过早收敛问题。而且计算每次迭代的系数，导致较高的计算成本。因此，为了能进一步提高优化器搜索性能的同时降低计算复杂度，在理论和实践上开发一种新的方法是重要的。

## 3. 一种新的带马尔科夫跳的 PSO 算法

在本节中介绍了一种新型的马尔可夫跳 PSO 算法(MJPSO)来平衡粒子的局部搜索和全局搜索，从而能够优化搜索性能，缓解过早收敛的问题。在速度更新方程中，惯性权重参数 $\omega$ 和加速度系数 $c_1$、$c_2$ 均可由马尔可夫跳跃参数自适应的控制。

### 3.1. 进化因子

基于每个进化状态，粒子的种群分布是一个非常重要的研究方向。由进化因子描述的种群分布特性在[19]中被引入。由进化因子定义种群的收敛、探索、开发、和跳出 4 种状态。首先，群中各粒子之间的平均距离计算如下：

$$d_i = \frac{1}{S} \sum_{j=1}^{S} \sqrt{\sum_{k=1}^{D} \left(x_i^k - x_j^k\right)^2} \tag{4}$$

这里，$S$ 和 $D$ 分别是种群规模和维度。因此进化因子可按文献[19]定义如下：

$$E_f = \frac{d_g - d_{\min}}{d_{\max} - d_{\min}}. \tag{5}$$

这里，$d_g$ 代表 $d_i$ 中的全局最优粒子，$d_{\max}$ 和 $d_{\min}$ 分别代表了 $d_i$ 中的最大值和最小值。进化因子 $E_f$ 描述了全局最优粒子到其他粒子的平均距离。

### 3.2. 更新策略

在[21]中，进化因子由一个马尔可夫链控制，基于马尔科夫链，加速度系数可以自适应调整。但是，

这种改进方法的缺点是，惯性权重系数需要在每一次迭代中重新计算，使得计算成本较高。同时，在多模态函数上求解全局最优值时仍然存在收敛速度较慢，容易陷入局部最优值的问题。

接下来，我们引入新的带马尔可夫跳跃参数的PSO的更新方程来克服上述缺点并进一步提高粒子的全局搜索能力。在速度更新方程中，马尔可夫跳跃参数可以自适应的控制惯性权重参数 $\omega$ 和加速度系数 $c_1$、$c_2$，该策略可以有效地防止粒子群算法的过早收敛。马尔可夫跳跃参数满足以下假设。

假设：代表第 $k$ 时刻粒子状态的跳跃参数 $\theta(k)(k\geq 0)$ 是离散时间齐次马氏链，在有限状态空间 $S=\{1,2,\cdots,N\}$ 中。概率转移矩阵 $\Pi=(\pi_{ij})_{N\times N}$ 按如下公式给出：

$$Prob=\{\theta(k+1)=j\mid\theta(k)=i\}=\pi_{ij},i,j=1,2,\cdots,N, \tag{6}$$

这里，$\pi_{ij}\geq 0(i,j\in S)$ 是从 $i$ 到 $j$ 的转移概率且满足 $\sum_{j=1}^{N}\pi_{ij}=1$。

所提出的MJPSO算法的速度和位置更新方程由下式给出：

$$\begin{aligned}v_i(k+1)&=\omega(\theta(k))v_i(k)+c_1(\theta(k))r_1(p_i(k)-x_i(k))\\&\quad+c_2(\theta(k))r_2(p_g(k)-x_i(k)),\\x_i(k+1)&=x_i(k)+v_i(k+1),\end{aligned} \tag{7}$$

这里，$\omega(\theta(k))$，$c_1(\theta(k))$ 和 $c_2(\theta(k))$ 是依赖于马氏链的惯性权重系数和加速度系数。进化状态的详细定义如下：

$$\theta(k)=\begin{cases}1, & 0\leq E_f\leq\dfrac{1}{N}\\2, & \dfrac{1}{N}\leq E_f\leq\dfrac{2}{N}\\3, & \dfrac{2}{N}\leq E_f\leq\dfrac{3}{N}\\\vdots & \vdots\\N, & \dfrac{N-1}{N}\leq E_f\leq 1\end{cases} \tag{8}$$

$$\Pi=\begin{pmatrix}\varphi & 1-\varphi & 0 & 0\cdots & 0 & 0 & 0\\\dfrac{1-\varphi}{2} & \varphi & \dfrac{1-\varphi}{2} & 0\cdots & 0 & 0 & 0\\0 & \dfrac{1-\varphi}{2} & \varphi & \dfrac{1-\varphi}{2} & 0 & 0 & 0\\\vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots\\0 & 0 & 0 & 0\cdots & \dfrac{1-\varphi}{2} & \varphi & \dfrac{1-\varphi}{2}\\0 & 0 & 0 & 0\cdots & 0 & 1-\varphi & \varphi\end{pmatrix} \tag{9}$$

因此，在每一次迭代中，马尔科夫过程都可以根据概率转移矩阵 $\Pi$ 更新状态。需要说明的是，$\varphi$ 的值对于收敛速度和精度来说是重要的[21]。在本文中，$\varphi$ 的值在迭代过程中设置为定值0.9。所提出的MJPSO的流程图如图1所示。马尔可夫跳跃过程见算法1。

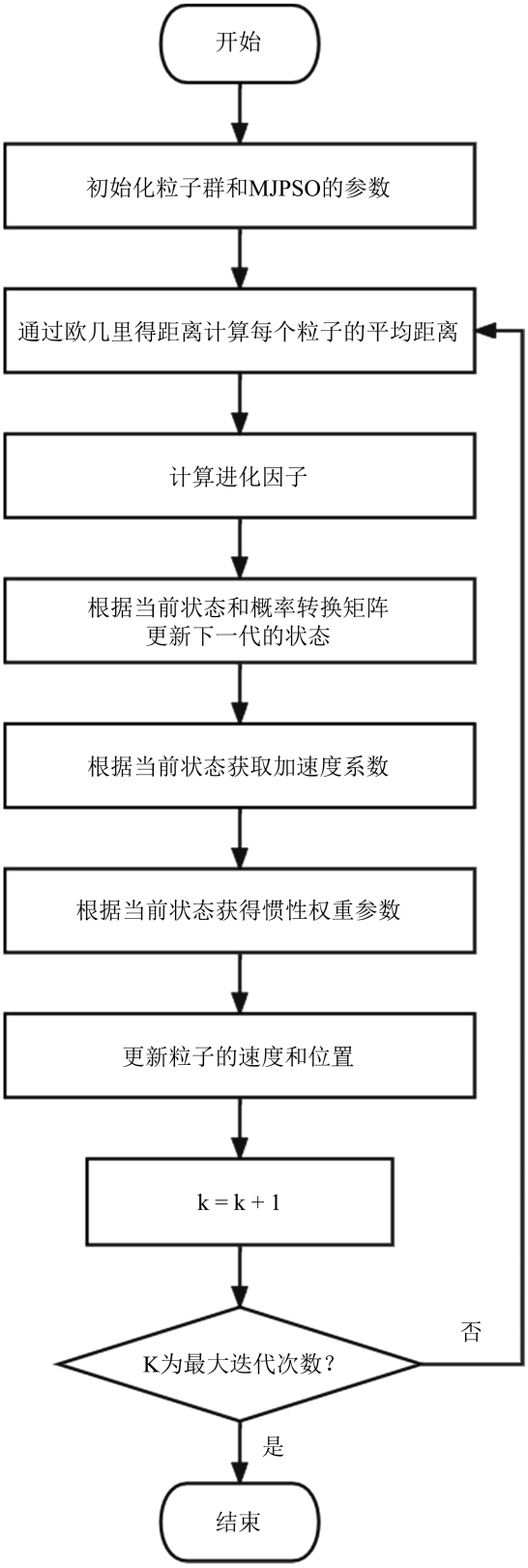


**Figure 1.** Flowchart of MJPSO algorithm
**图 1.** MJPSO 算法的流程图

**算法 1.** 马尔可夫跳跃过程

$N \leftarrow$ 迭代状态个数

$D \leftarrow$ 搜索空间维度

$S \leftarrow$ 粒子群总数

$X \leftarrow$ 粒子的当前位置

//根据方程(4)计算平均距离

$var_2 \leftarrow 0$

for $j = 1:S$ do

$var_2 \leftarrow 0$

for $k = 1:D$ do

$var_2 \leftarrow \left(X(i,k) - X(j,k)\right)^2 + var$

end for

$var_2 \leftarrow var_2 + \sqrt{var_2}$

end for

$Dis_i \leftarrow var_2 / S$

//计算 $E_f$ 和当前状态

$E_f \leftarrow \left(Dis_j - \min(Dis)\right) / \left(\max(Dis) - \min(Dis)\right)$

for $i = 0:N$ do

if $i/N \le E_f < (i+1)/N$ then

$\theta \leftarrow$ 当前状态 $i$

end if

end for

//根据概率转移矩阵 $\prod$ 计算下一个进化状态

$\varphi(1,1) \leftarrow 0.9$

$\varphi(N_S, N_S) \leftarrow 0.1$

for $i = 1:N_S$ do

for 粒子 $i$ do

利用 $\varphi$ 确定下一个状态

end if

end for

## 4. 收敛性实验

### 4.1. 参数设置

在[19]中，用 $E_f$ 定义了四种集合 $S_1$， $S_2$， $S_3$ 和 $S_4$ 分别代表收敛、探索、开发、和跳出 4 种状态。因此在仿真中，我们以 $N = 4$ 为例，令 $\theta(k) = 1$， $\theta(k) = 2$， $\theta(k) = 3$， $\theta(k) = 4$ 分别代表这 4 种状态。所提出的 MJPSO 算法的加速度参数的初始值 $c_1$ 和 $c_2$ 设为 1.75，惯性权重 $\omega$ 设为 0.75，并且他们可以根据进化状态自动调整他们的值。另外，我们注意到，一个较大的 $\omega$ 更倾向于在跳出和探索状态中进行全局搜索，而一个较小的 $\omega$ 更倾向于局部微调[24]。因此我们引入其控制策略如下：

1) 在跳出状态，当前的全局最优粒子更倾向于飞向更好的最优粒子，而逃离局部最优，因此我们分配给 $c_2$ 一个较大的值 2.1，而分配给 $c_1$ 一个较小的值 1.8， $\omega_4 = 0.95$ 。

2) 在探索状态，粒子尽可能多地探索更多的最优解。因此我们分配给 $c_1$ 一个较大的值 2.1，而分配给 $c_2$ 一个较小的值 1.8，并令 $\omega_3 = 0.9$ 。

3) 在开发状态，例子期望尽可能地利用其历史最佳位置和当前迭代的全局最佳位置来增强开发本区域。因此我们设置 $c_1$ 为 1.9， $c_2$ 为 1.7，并令 $\omega_2 = 0.8$ 。

4) 在收敛状态，粒子期望尽可能快的收敛到最优。因此我们设置 $c_1$ 为 1.75， $c_2$ 为 1.75，并令 $\omega_1 = 0.75$ 。

设置参数 $\varphi = 0.9$ ，我们有概率转移矩阵如下：

$$\Pi = \begin{pmatrix} 0.9 & 0.1 & 0 & 0 \\ 0.05 & 0.9 & 0.05 & 0 \\ 0 & 0.05 & 0.9 & 0.05 \\ 0 & 0 & 0.1 & 0.9 \end{pmatrix} \tag{10}$$

## 4.2. 实验设置

为了测试提出的 MJPSO 算法的性能，(11)~(18)和表 1 给出了一些著名的基准函数，所有函数都在 20 个维度上进行测试。 $f_1(x)$， $f_2(x)$， $f_3(x)$ 描述单峰优化问题。 $f_1(x)$ 和 $f_2(x)$ 通常被用来检测 PSOs 算法的收敛速度。 $f_3(x)$ 很难优化，其全局最小值位于香蕉形状的抛物山谷中。然而，尽管该山谷很容易找到，但很难收敛到最小值。因此 $f_3(x)$ 可以被视为一个多模态问题以用来检测算法收敛到全域最小值的能力。 $f_5(x)$ 在每个阶跃间会产生大量局部极值，具有较高的寻优难度。 $f_6(x)$ 是较难优化的多模态问题，以用来检测算法跳出局部最小值的能力。需要说明的是，所有的函数描述最小化问题且存在全局最小值。

Sphere: $f_1(x) = \sum_{i=1}^{D} x_i^2$ , (11)

Schwefel's Problem 1.2: $f_2(x) = \sum_{i=1}^{D} \left( \sum_{j=1}^{i} x_j \right)^2$ (12)

Rosenbrock: $f_3(x) = \sum_{i=1}^{D-1} \left( 100\left(x_{i+1} - x_i\right)^2 + \left(x_i - 1\right)^2 \right)$ (13)

Schwefel's Problem 2.21: $f_4(x) = \max_i \left\{ \left| x_i \right|, 1 \le i \le 30 \right\}$ (14)

Step: $f_5(x) = \sum_{i=1}^{D} \left( \left\lfloor x_i + 0.5 \right\rfloor \right)^2$ (15)

Ackley: $f_6(x) = -20e^{-0.2\sqrt{\frac{1}{D}\sum_{i=1}^{D} x_i^2}} - e^{\frac{1}{D}\sum_{i=1}^{D} \cos 2\pi x_i} + 20 + e$ (16)

Penalized1: $f_7(x) = \frac{\pi}{D}\left( 10\sin^2\left(\pi y_1\right) + \sum_{i=1}^{D-1} \left(y_i - 1\right)^2 \left(1 + 10\sin^2\left(\pi y_{i+1}\right)\right) + \left(y_D - 1\right)^2 \right) + \sum_{i=1}^{D} u\left(x_i\right)$

$y_i = 1 + 1/4\left(x_i + 1\right)$ (17)

$$u\left(x_i\right) = \begin{cases} 100\left(-x_i - 10\right)^4, & x_i < -10, \\ 0, & \left|x_i\right| \le 10, \\ 100\left(x_i - 10\right)^4, & x_i > 10. \end{cases}$$

$$\text{Penalized2: } f_8(x)=0.1\left(\sin^2(3\pi x_1)+\sum_{i=1}^{D-1}(x_i-1)^2\left(1+\sin^2(3\pi x_{i+1})\right)\right.$$
$$\left.+(x_D-1)^2\left(1+\sin^2(2\pi x_D)\right)\right)+\sum_{i=1}^{D}u(x_i) \tag{18}$$
$$u(x_i)=\begin{cases}100(-x_i-5)^4, & x_i<-5,\\ 0, & |x_i|\le 5,\\ 100(x_i-5)^4, & x_i>5.\end{cases}$$

**Table 1.** Benchmark configurations
**表 1.** 基准配置

| 函数 | 名称 | 搜索范围 | 维度 | 最小值 | 可接受精度 |
|---|---|---|---|---|---|
| $F_1(x)$ | Sphere | [−100, 100] | 20 | 0 | 0.01 |
| $F_2(x)$ | Schwefel's Problem 1.2 | [−100, 100] | 20 | 0 | 0.01 |
| $F_3(x)$ | Rosenbrock | [−10, 10] | 20 | 0 | 100 |
| $F_4(x)$ | Schwefel's Problem 2.21 | [−100, 100] | 20 | 0 | 0.01 |
| $F_5(x)$ | Step | [−100, 100] | 20 | 0 | 0.01 |
| $F_6(x)$ | Ackley | [−32, 32] | 20 | 0 | 0.01 |
| $F_7(x)$ | Penalized1 | [−50, 50] | 20 | 0 | 0.01 |
| $F_8(x)$ | Penalized2 | [−50, 50] | 20 | 0 | 0.01 |

我们通过实验来比较包含所建议的 MJPSO 算法在内的 7 种 PSO 算法在 8 种测试函数上的表现。表 2 给出 6 种现有的 PSO 算法的详细信息。所提出的 MJPSO 算法的参数在第 VI 节的参数设置小节中给出。所有 PSO 算法的种群规模设置为 20，最大迭代测次数为 10,000。此外，为了消除随机误差，奖对每个算法进行独立重复试验 20 次。所有实验均在同一台机器上进行，测试环境如表 3。

**Table 2.** Comparison of PSOs
**表 2.** PSOs 的比较

| 算法 | 参数 |
|---|---|
| PSO-CK | $\omega$ : 0.729, $c_1=c_2=2.05$ |
| SPSO | Automatically chosen |
| PSO-LDIW | $\omega$ : 0.9~0.4, $c_1=c_2=2$ |
| PSO-TVAC | $\omega$ : 0.9~0.4, $c_1$ : 2.5~0.5, $c_2$ : 2.5~0.5 |
| SDPSO | Automatically chosen |
| RPSO | $\omega$ : 0.9~0.4, $c_1$ : 2.5~0.5, $c_2$ : 2.5~0.5, $\delta_1=0$, $\delta_2=0.07$ |

**Table 3.** Test environment
**表 3.** 测试环境

| | |
|---|---|
| 硬件 | Intel Core i7 |
| 软件 | Win 10 (64bit) Python 3.8 |

### 4.3. 仿真结果和讨论

我们取平均值、标准差(Std.Dev.)和最优值作为测试结果，具体测试结果详细见表 4 (其中黑体加粗的表示对应项的最好结果)。图 2~9 描述了在 8 个基准函数上各算法的收敛平均解和迭代过程的比较。

**Table 4.** Comparisons of search results among six PSOs on five benchmark functions
**表 4.** 六种 PSOs 在五个基准函数上的搜索结果的比较

| | | PSO-CK | SPSO | PSO-LDIW | PSO-TVAC | SDPSO | RPSO | MJPSO |
|---|---|---|---|---|---|---|---|---|
| F1 | Best value | 2.76e−03 | 4.33e−03 | 1.95e−04 | 1.29e−03 | 4.70e−05 | 1.76e−03 | **8.71e−08** |
| | Mean | 3.79e−02 | 1.42e−02 | 8.65e−04 | 7.35e−03 | 5.81e−04 | 1.02e−02 | **8.62e−07** |
| | Std. Dev. | 3.48e−02 | 1.01e−02 | 4.30e−04 | 5.14e−03 | 9.58e−04 | 8.27e−03 | **9.61e−07** |
| F2 | Best value | 3.10e−02 | 8.41e+00 | 5.88e−02 | 4.17e−01 | 1.43e+00 | 2.77e−01 | **9.63e−04** |
| | Mean | 1.62e−01 | 4.31e+01 | 2.51e−01 | 2.52e+02 | 2.33e+01 | 5.02e+02 | **4.48e−03** |
| | Std. Dev | 1.17e−01 | 4.05e+01 | 1.42e−01 | 1.09e+03 | 3.00e+01 | 1.50e+03 | **2.86e−03** |
| F3 | Best value | 1.16e−01 | 3.68e−02 | 5.23e−03 | 8.88e−02 | 6.63e−04 | 8.60e−02 | **3.38e−05** |
| | Mean | 4.70e−01 | 2.04e−01 | 1.70e−01 | 2.63e+00 | 2.97e−01 | 1.75e+00 | **6.68e−03** |
| | Std. Dev. | 4.54e−01 | 2.07e+01 | 1.82e−01 | 2.69e+00 | 4.95e−01 | 1.69e+00 | **7.39e−03** |
| F4 | Best value | 2.81e−01 | 2.44e+00 | 6.13e−02 | 2.27e−01 | 1.21e−01 | 2.49e−01 | **1.27e−02** |
| | Mean | 6.56e−01 | 1.33e+01 | 1.76e−01 | 6.35e−01 | 2.98e−01 | 5.38e−01 | **6.77e−02** |
| | Std. Dev. | 4.74e−01 | 1.03e+01 | 8.04e−02 | 3.38e−01 | 1.30e−01 | 3.61e−01 | **4.42e−02** |
| F5 | Best value | **0** | 4.00e+01 | **0** | **0** | **0** | **0** | **0** |
| | Mean | 2.40e+00 | 1.29e+03 | 2.50e−01 | 1.05e+00 | 2.30e+00 | 1.40e+00 | **1.00e−01** |
| | Std. Dev. | 1.28e+00 | 1.70e+03 | 4.33e−01 | 8.65e−01 | 1.52e+00 | 1.02+00 | **3.00e−01** |
| F6 | Best value | 1.87e+00 | 6.19e+00 | 1.67e−02 | 4.29e−02 | 1.50e−02 | 6.41e−02 | **6.31e−04** |
| | Mean | 4.28e+00 | 1.25e+00 | 1.22e+00 | 1.83e+00 | 2.43e+00 | 1.88e+00 | **3.82e−01** |
| | Std. Dev. | 1.13e+00 | **5.14e−01** | 6.54e−01 | 7.05e−01 | 1.04e+00 | 8.11e−01 | 5.71e−01 |
| F7 | Best value | 1.21e−03 | 6.51e−04 | 6.99e−05 | 4.88e−02 | 1.69e−04 | 2.37e−01 | **7.57e−07** |
| | Mean | 4.02e+00 | 1.82e+00 | 1.13e+00 | 3.69e+00 | 2.40e+00 | 4.40e+00 | **7.00e−02** |
| | Std. Dev. | 3.32e+00 | 2.22e+00 | 1.67e+00 | 2.75e+00 | 2.73e+00 | 5.57e+00 | **1.87e−01** |
| F8 | Best value | 4.58e−02 | 9.41e+00 | 2.15e−04 | 4.31e−03 | 1.26e−05 | 3.15e−03 | **1.77e−06** |
| | Mean | 2.66e+00 | 3.23e+01 | 8.46e−03 | 4.62e−02 | 6.66e−03 | 3.36e−02 | **2.78e−03** |
| | Std. Dev. | 4.10e+00 | 2.08e+01 | 6.31e−03 | 4.61e−02 | 7.28e−03 | 3.55e−02 | **4.76e−03** |

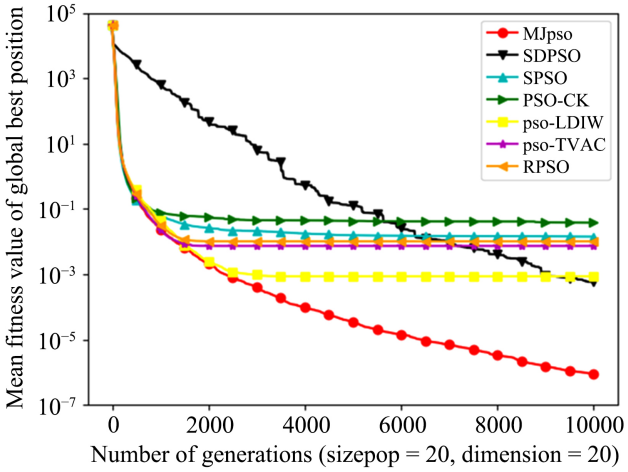


**Figure 2.** Performance of seven PSOs for 20-dimensional $f_1(x)$
**图 2.** 在 20 维 $f_1(x)$函数上 7 种 PSOs 的表现

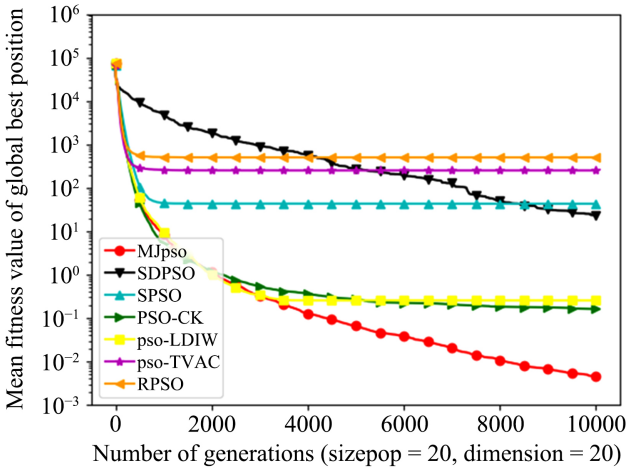


**Figure 3.** Performance of seven PSOs for 20-dimensional $f_2(x)$
**图 3.** 在 20 维 $f_2(x)$函数上 7 种 PSOs 的表现

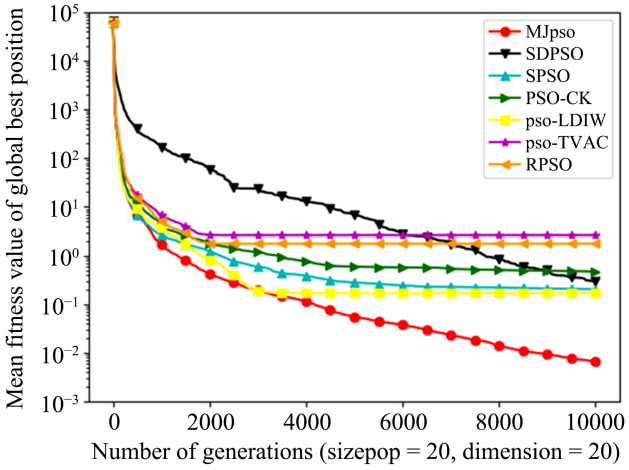


**Figure 4.** Performance of seven PSOs for 20-dimensional $f_3(x)$
**图 4.** 在 20 维 $f_3(x)$函数上 7 种 PSOs 的表现

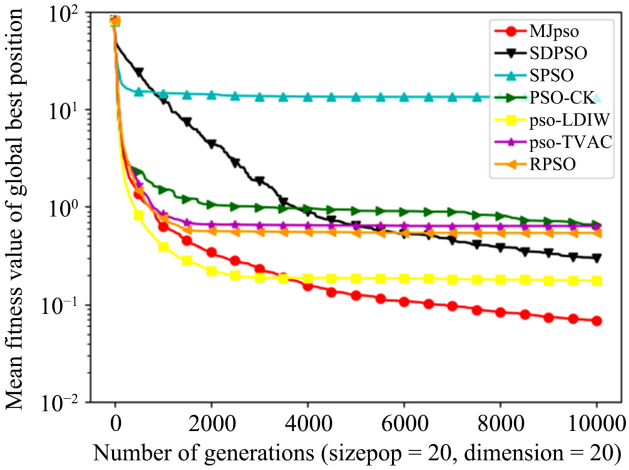


**Figure 5.** Performance of seven PSOs for 20-dimensional $f_4(x)$
**图 5.** 在 20 维 $f_4(x)$函数上 7 种 PSOs 的表现

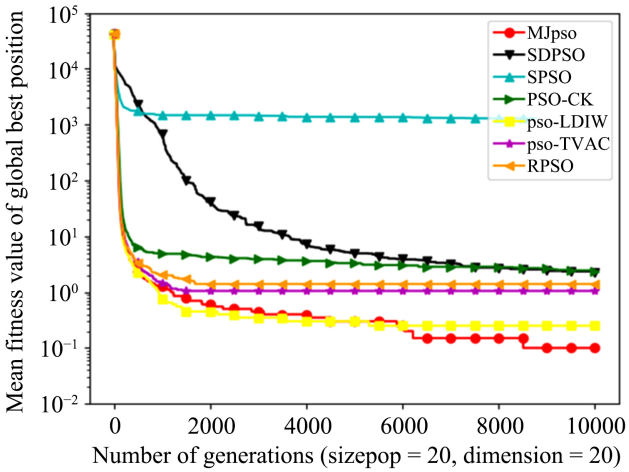


**Figure 6.** Performance of seven PSOs for 20-dimensional $f_5(x)$
**图 6.** 在 20 维 $f_5(x)$函数上 7 种 PSOs 的表现

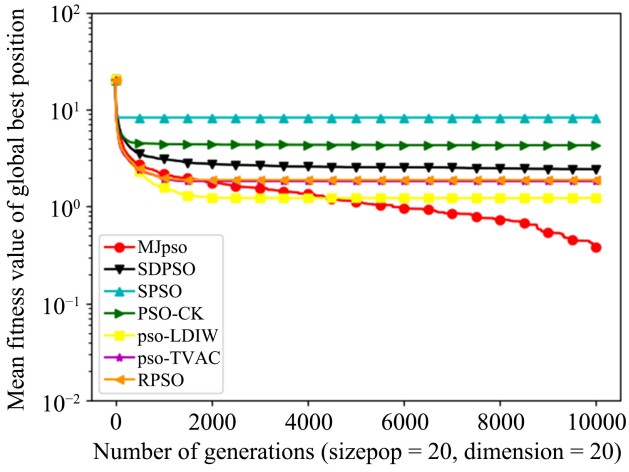


**Figure 7.** Performance of seven PSOs for 20-dimensional $f_6(x)$
**图 7.** 在 20 维 $f_6(x)$函数上 7 种 PSOs 的表现

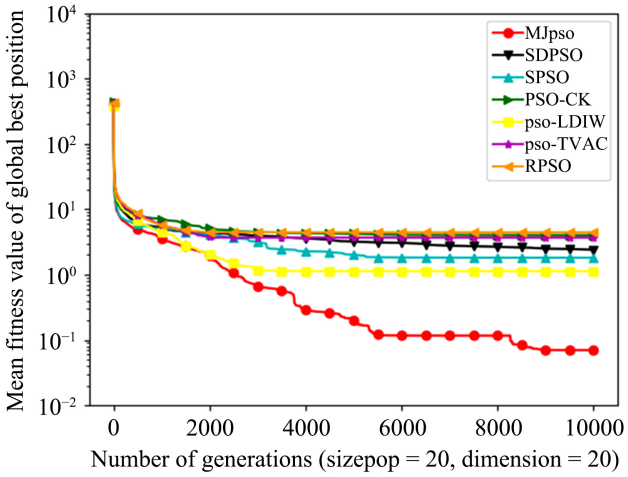


**Figure 8.** Performance of seven PSOs for 20-dimensional $f_7(x)$
**图 8.** 在 20 维 $f_7(x)$函数上 7 种 PSOs 的表现

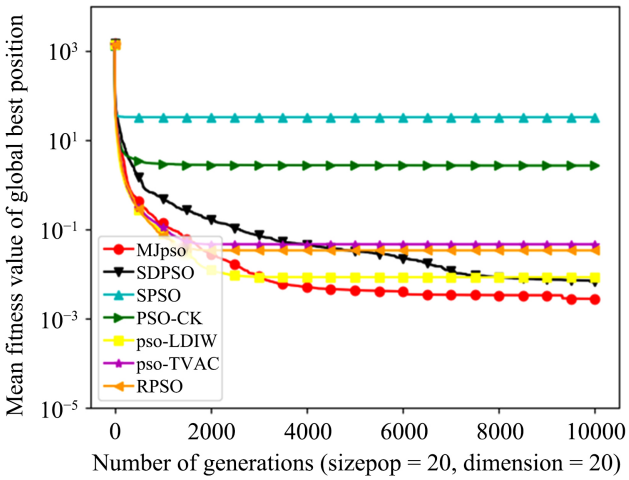


**Figure 9.** Performance of seven PSOs for 20-dimensional $f_8(x)$
**图 9.** 在 20 维 $f_8(x)$函数上 7 种 PSOs 的表现

正如表 4 和图 2~9 所示，所提出的 MJPSO 算法优于其他六种流行的 PSO 算法。特别是 PSOs 算法的在每一个基准函数上的最佳结果均以黑体标出。可以发现，MJPSO 算法均可以提供一个最优解。球体函数和 schwefel 函数用来检验算法的收敛速率。从表 4 和图 2、图 3、图 5 中可以看出，对于 $f_1(x)$ 和 $f_4(x)$，PSO-CK 算法和 SPSO 算法分别具有较差的平均全局最优值，收敛情况较差；对于 $f_2(x)$，PSO-TVAC 算法和 RPSO 算法出现了较大的偏差值，且 PSO-TVAC 算法，RPSO 算法，SPSO 算法，SDPSO 算法的平均全局最优值表现较差。而在六种测试函数上，所提出的 MJPSO 算法明显表现得更好。$f_3(x)$ 被视为单模态优化问题，除了验证本地搜索能力外，还用来验证全局搜索能力。从表 4 和图 4 中可以看出，很明显所提出的 MJPSO 算法优于其他六种算法，尤其 PSO-TVAC 算法出现了较明显的偏差值。从表 4 和图 6~9 中可以看出，所提出的 MJPSO 算法能够实现在复杂函数上的全局优化。对于 $f_5(x)$ 和 $f_8(x)$，SPSO 算法不仅具有较差的平均全局最优值，还出现了较大的偏差值。PSO-CK 算法在 $f_8(x)$ 上表现也较差。总体来说，无论是处理单峰函数问题还是处理多峰函数问题时，所提出的 MJPSO 算法的性能都优于其他算法，并且所提出的 MJPSO 算法的局部搜索和全局搜索能力较强。因此，能够帮助

PSO 搜索到最佳位置并保持高收敛速度，避免局部最优并找到全局最优位置。

对于 8 个基准函数，我们分别剔除上文提到的表现较差的算法。图 10~17 更加详细展示了在 20 次独立实验中，各算法在各基准函数上的全局最优收敛值的分散情况。箱型图显示数据到四分位点的分布，突出显示平均值和离群值。箱形具有可垂直延长的名为“须线”的线条。这些线条指示超出四分位点上限和下限的变化程度，处于这些线条或须线之外的任何点都被视为离群值。当箱形图很短时，就意味着很多数据点是相似的，因为很多值是在一个很小的范围内分布；当箱形图较高时，就意味着大部分的数据点之间的差异很大，因为这些值分布的很广。

通过图 10~17 的展示，我们可以发现与其余各 PSOs 算法相比，MJPSO 算法在 8 个基准函数上的全局收敛值数据更为集中，且拥有较低的数据变异性。实验结果表明，基于摆脱局部最优的强性能和令人满意的收敛表现，所提出的 MJPSO 算法在与六种流行的 PSO 算法的比较中，取得了优异的成绩。

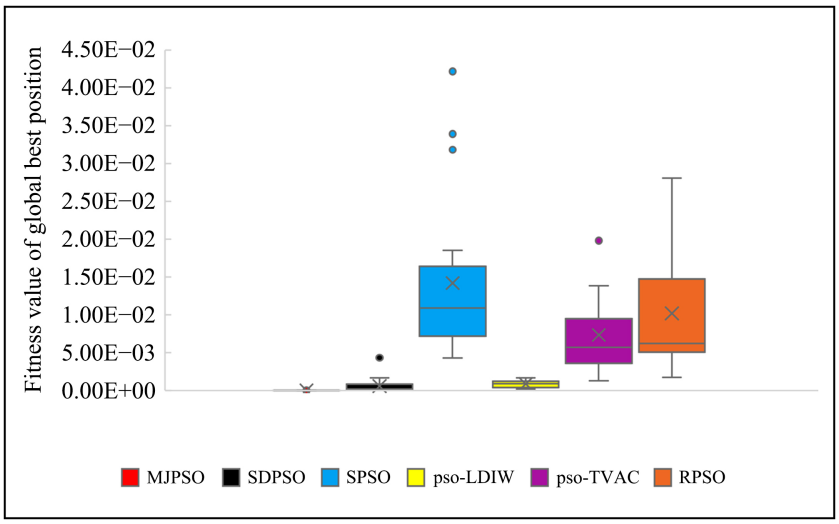


**Figure 10.** Box diagram of six PSOs for 20-dimensional $f_1(x)$
**图 10.** 在 20 维 $f_1(x)$函数上 6 种 PSOs 的箱型图

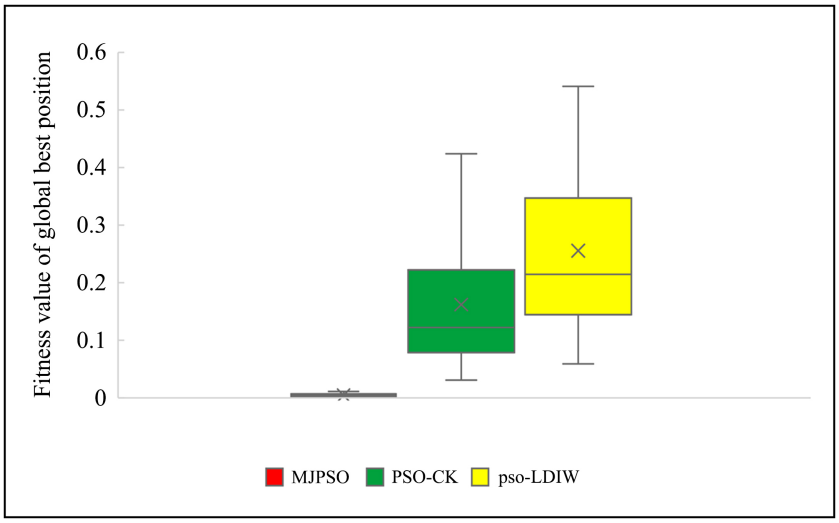


**Figure 11.** Box diagram of three PSOs for 20-dimensional $f_2(x)$
**图 11.** 在 20 维 $f_2(x)$函数上 3 种 PSOs 的箱型图

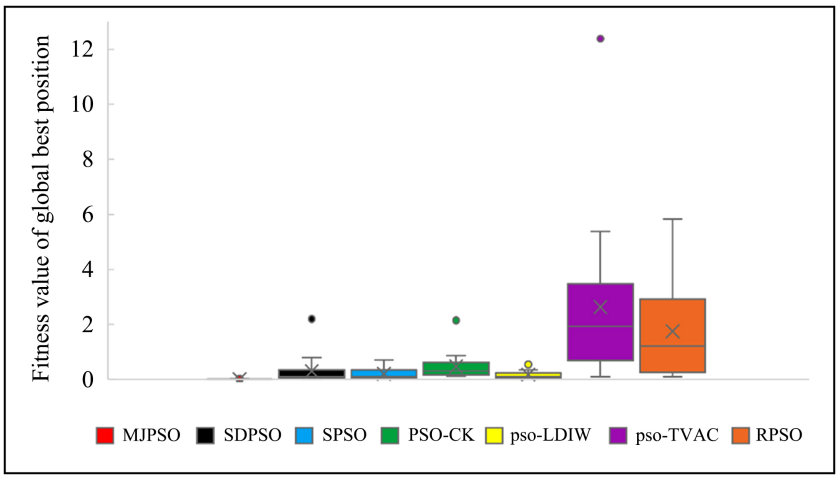


**Figure 12.** Box diagram of seven PSOs for 20-dimensional $f_3(x)$
**图 12.** 在 20 维 $f_3(x)$函数上 7 种 PSOs 的箱型图

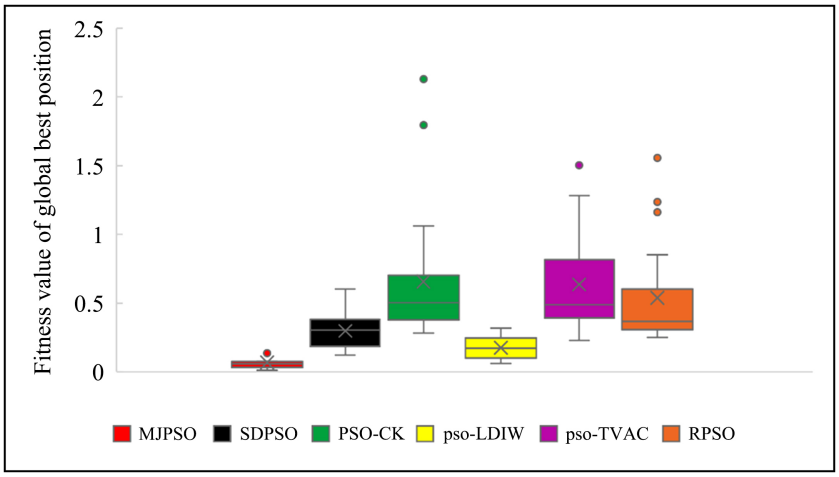


**Figure 13.** Box diagram of six PSOs for 20-dimensional $f_4(x)$
**图 13.** 在 20 维 $f_4(x)$函数上 6 种 PSOs 的箱型图

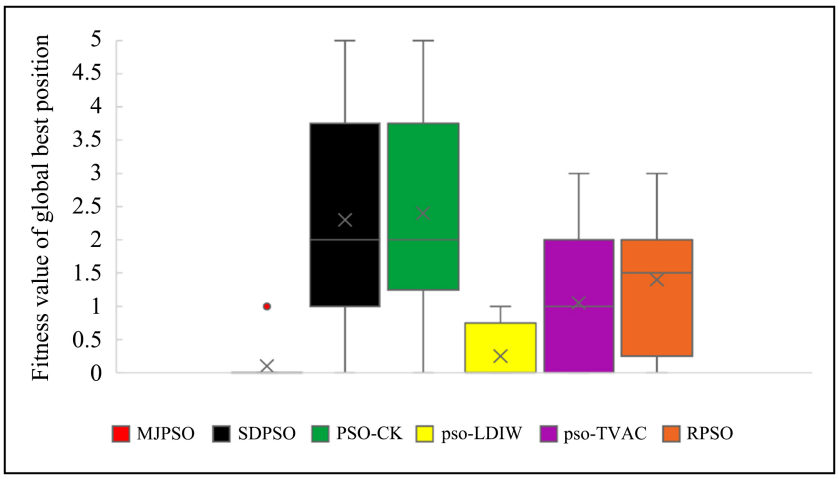


**Figure 14.** Box diagram of six PSOs for 20-dimensional $f_5(x)$
**图 14.** 在 20 维 $f_5(x)$函数上 6 种 PSOs 的箱型图

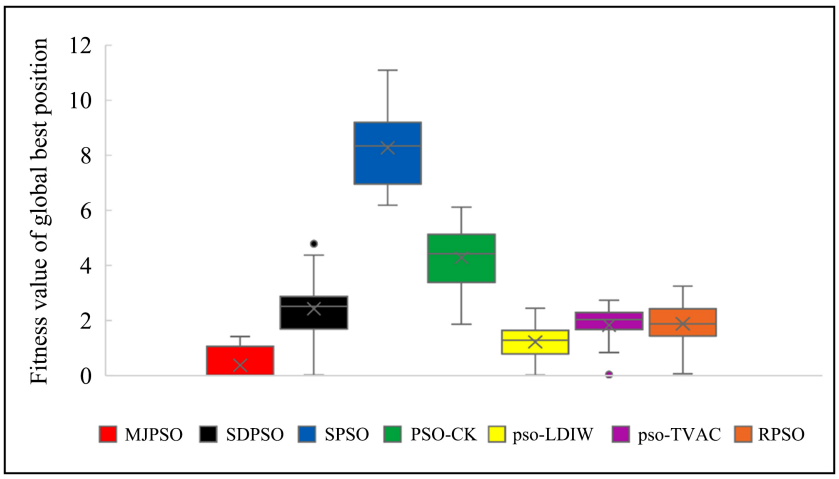


**Figure 15.** Box diagram of seven PSOs for 20-dimensional $f_6(x)$
**图 15.** 在 20 维 $f_6(x)$函数上 7 种 PSOs 的箱型图

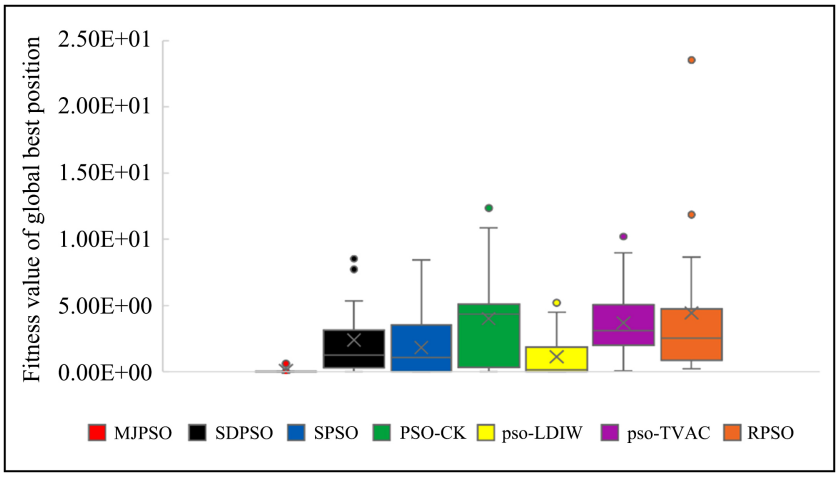


**Figure 16.** Box diagram of seven PSOs for 20-dimensional $f_7(x)$
**图 16.** 在 20 维 $f_7(x)$函数上 7 种 PSOs 的箱型图

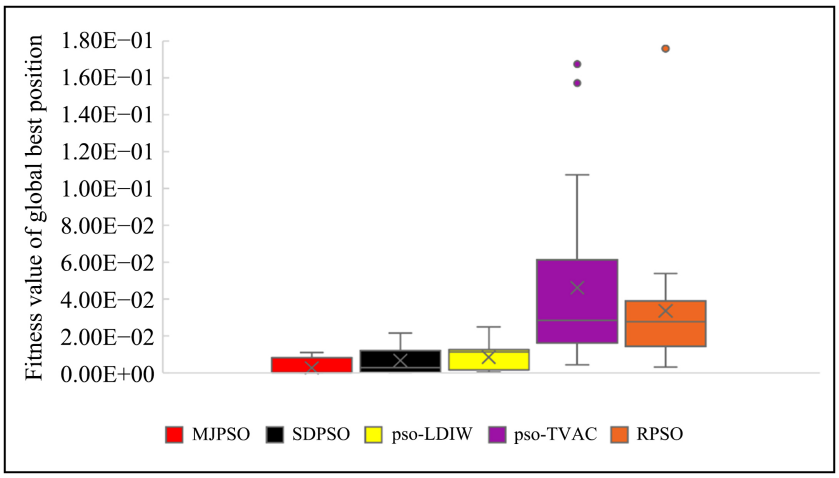


**Figure 17.** Box diagram of five PSOs for 20-dimensional $f_8(x)$
**图 17.** 在 20 维 $f_8(x)$函数上 5 种 PSOs 的箱型图

## 5. 结论

为了提高标准 PSO 方法的搜索能力，本文介绍了一种 MJPSO 算法。设计了一种新的变异策略来调整 MJPSO 算法中粒子的惯性权重参数和加速度系数。收敛性实验的结果表明，在八个典型基准函数上，所提出的 MJPSO 算法的性能优于六种流行的 PSO 算法。未来，我们的研究主题将集中在选择更复杂的动态策略以进一步有效跳出局部最优，以及对所提出的算法在理论上进行收敛性和稳定性的分析。

## 参考文献


[1] Deb, K. (2012) Optimization for Engineering Design—Algorithms and Examples, Second Edition. PHI Learning Private Limited, Delhi.

[2] Mitchell, M. (1996) An Introduction to Genetic Algorithms. MIT Press, Cambridge.

[3] Song, Y., Chen, Z. and Yuan, Z. (2007) New Chaotic PSO-Based Neural Network Predictive Control for Nonlinear Process. *IEEE Transactions on Neural Networks*, **18**, 595-600. https://doi.org/10.1109/TNN.2006.890809

[4] Jeng, J.H., Tseng, C.C. and Hsieh, J.G. (2009) Study on Huber Fractal Image Compression. *IEEE Transactions on Image Processing*, **18**, 995-1003. https://doi.org/10.1109/TIP.2009.2013080

[5] Paoli, A., Melgani, F. and Pasolli, E. (2009) Clustering of Hyperspectral Images Based on Multiobjective Particle Swarm Optimization. *IEEE Transactions on Geoscience and Remote Sensing*, **47**, 4175-4188. https://doi.org/10.1109/TGRS.2009.2023666

[6] 张娟芝, 段中兴, 熊福力. 一种自适应粒子群算法在云资源调度中的应用[J]. 计算机测量与控制, 2020, 28(12): 217-226.

[7] Pozna, C., Precup, R.-E., Horváth, E. and Petriu, E.M. (2022) Hybrid Particle Filter-Particle Swarm Optimization Algorithm and Application to Fuzzy Controlled Servo Systems. *IEEE Transactions on Fuzzy Systems*, **30**, 4286-4297. https://doi.org/10.1109/TFUZZ.2022.3146986

[8] 孙一凡, 张纪会. 基于模拟退火机制的自适应粘性粒子群算法[J]. 控制与决策, 2022.

[9] Kennedy, J. and Eberhart, R.C. (1995) Particle Swarm Optimization. *IEEE International Conference on Neural Networks*, Perth, 27 November-1 December 1995, 1942-1948.

[10] Eberhart, R.C. and Kennedy, J. (1995) A New Optimizer Using Particle Swarm Theory. *MHS*'95. *Proceedings of the Sixth International Symposium on Micro Machine and Human Science*, Nagoya, 4-6 October 1995, 39-43.

[11] Clerc, M. and Kennedy, J. (2002) The Particle Swarm—Explosion, Stability, and Convergence in a Multidimensional Complex Space. *IEEE Transactions on Evolutionary Computation*, **6**, 58-73. https://doi.org/10.1109/4235.985692

[12] Clerc, M. (1999) The Swarm and the Queen: Towards a Deterministic and Adaptive Particle Swarm Optimization. *IEEE International Conference on Evolutionary Computation*, Vol. 3, 1952-1957.

[13] Clerc, M. (2006) Particle Swarm Optimization. Wiley, Hoboken. https://doi.org/10.1002/9780470612163

[14] Shi, Y. and Eberhart, R. (1998) A Modified Particle Swarm Optimizer. *IEEE International Conference on Evolutionary Computation*, Anchorage, 4-9 May 1998, 69-73.

[15] Shi, Y. and Eberhart, R.C. (1999) Empirical Study of Particle Swarm Optimization. 1999 *IEEE Congress on Evolutionary Computation*, Washington DC, 6-9 July 1999, 1945-1950.

[16] Ratnaweera, A., Halgamuge, S.K. and Watson, H.C. (2004) Self-Organizing Hierarchical Particle Swarm Optimizer with Time-Varying Acceleration Coefficients. *IEEE Transactions on Evolutionary Computation*, **8**, 240-255. https://doi.org/10.1109/TEVC.2004.826071

[17] Shi, Y. and Eberhart, R.C. (1998) Parameter Selection in Particle Swarm Optimization. *International Conference on Evolutionary Programming*, San Diego, 25-27 March 1998, 591-600. https://doi.org/10.1007/BFb0040810

[18] Liu, W., Wang, Z., Zeng, N., *et al*. (2020) A Novel Randomised Particle Swarm Optimizer. *International Journal of Machine Learning and Cybernetics*, **12**, 529-540. https://doi.org/10.1007/s13042-020-01186-4

[19] Zhan, Z.H., Zhang, J., Li, Y. and Chung, H.S. (2009) Adaptive Particle Swarm Optimization. *IEEE Transactions on Systems*, *Man*, *and Cybernetics Part B*: *Cybernetics*, **39**, 1362-1381. https://doi.org/10.1109/TSMCB.2009.2015956

[20] Liu, J., Ma, X., Li, X., Liu, M., Shi, T. and Li, P. (2021) Random Convergence Analysis of Particle Swarm Optimization Algorithm with Time-Varying Attractor. *Swarm and Evolutionary Computation*, **61**, Article ID: 100819. https://doi.org/10.1016/j.swevo.2020.100819

[21] Tang, Y., Wang, Z. and Fang, J.-A. (2011) Parameters Identification of Unknown Delayed Genetic Regulatory Net-

works by a Switching Particle Swarm Optimization Algorithm. *Expert Systems with Applications*, **38**, 2523-2535. https://doi.org/10.1016/j.eswa.2010.08.041

[22] Rahman, I.U., Wang, Z., Liu, W., *et al*. (2021) An N-State Markovian Jumping Particle Swarm Optimization Algorithm. *IEEE Transactions on Systems*, *Man*, *and Cybernetics*: *Systems*, **51**, 6626-6638. https://doi.org/10.1109/TSMC.2019.2958550

[23] Zeng, N., Wang, Z., Li, Y., *et al*. (2012) A Hybrid EKF and Switching PSO Algorithm for Joint State and Parameter Estimation of Lateral Flow Immunoassay Models. *IEEE/ACM Transactions on Computational Biology and Bioinformatics*, **9**, 321-329. https://doi.org/10.1109/TCBB.2011.140

[24] Zeng, N., Wang, Z., Zhang, H. and Alsaadi, F.E. (2016) A Novel Switching Delayed PSO Algorithm for Estimating Unknown Parameters of Lateral Flow Immunoassay. *Cognitive Computation*, **8**, 143-152. https://doi.org/10.1007/s12559-016-9396-6

[25] Zeng, N., Zhang, H., Liu, W., *et al*. (2017) A Switching Delayed PSO Optimized Extreme Learning Machine for Short-Term Load Forecasting. *Neurocomputing*, **240**, 175-182. https://doi.org/10.1016/j.neucom.2017.01.090

[26] Zeng, N., Qiu, H., Wang, Z., Liu, W., Zhang, H. and Li, Y. (2018) A New Switching-Delayed-PSO-Based Optimized SVM Algorithm for Diagnosis of Alzheimer's Disease. *Neurocomputing*, **320**, 195-202. https://doi.org/10.1016/j.neucom.2018.09.001