Chaotic catfish particle swarm optimization for solving global numerical optimization problems

Li-Yeh Chuang a, Sheng-Wei Tsai b, Cheng-Hong Yang b,c,*

a Institute of Biotechnology and Chemical Engineering, I-Shou University, Kaohsiung 84001, Taiwan
b Department of Electronic Engineering, National Kaohsiung University of Applied Sciences, Kaohsiung 80778, Taiwan
c Department of Network Systems, Toko University, Chiayi 61363, Taiwan

Abstract

Chaotic catfish particle swarm optimization (C-CatfishPSO) is a novel optimization algorithm proposed in this paper. C-CatfishPSO introduces chaotic maps into catfish particle swarm optimization (CatfishPSO), which increase the search capability of CatfishPSO via the chaos approach. Simple CatfishPSO relies on the incorporation of catfish particles into particle swarm optimization (PSO). The introduced catfish particles improve the performance of PSO considerably. Unlike other ordinary particles, the catfish particles initialize a new search from extreme points of the search space when the gbest fitness value (global optimum at each iteration) has not changed for a certain number of consecutive iterations. This results in further opportunities of finding better solutions for the swarm by guiding the entire swarm to promising new regions of the search space and accelerating the search. The introduced chaotic maps strengthen the solution quality of PSO and CatfishPSO significantly. The resulting improved PSO and CatfishPSO are called chaotic PSO (C-PSO) and chaotic CatfishPSO (C-CatfishPSO), respectively. PSO, C-PSO, CatfishPSO, C-CatfishPSO, as well as other advanced PSO procedures from the literature were extensively compared on several benchmark test functions. Statistical analysis of the experimental results indicate that the performance of C-CatfishPSO is better than the performance of PSO, C-PSO, CatfishPSO and that C-CatfishPSO is also superior to advanced PSO methods from the literature.

1. Introduction

Evolutionary algorithms with their heuristic and stochastic properties often suffer from getting stuck in local optima. This common characteristic led to the development of evolutionary computation as an increasingly important field. A GA is a stochastic search procedure based on the mechanics of natural selection, genetics and evolution [1]. Since this type of algorithm simultaneously evaluates many points in the search space, it is more likely to find a global solution to a given problem. PSO describes a solution process in which each particle moves through a multidimensional search space [2]. The particle velocity and position are constantly updated according to the best previous performance of the particle or of the particle’s neighbors, as well as the best performance of all particles in the entire population. GAs have demonstrated the ability to reach near-optimal solutions for large problems; however, they may require a long processing time to reach a near-optimal solution. Similarly to GAs, BPSO is also a population-based optimizer. BPSO has a memory, so knowledge of good solutions is retained by all the particles and optimal solutions are found by the swarm particles if they follow the best particle. Unlike GAs, BPSO does not contain any crossover and mutation processes [3]. Hybridization of evolutionary algorithms with local search has...
been investigated in many studies [4,5]. Such hybrids are often referred to as memetic algorithms (MA). An MA can be treated as a genetic algorithm coupled with a local search procedure [6]. The shuffled frog leaping algorithm (SFL algorithm) combines the benefits of an MA and the social PSO algorithm. Unlike in MAs and PSO, the population consists of a set of solutions (frogs), which is partitioned into subsets referred to as memeplexes. In the search space, each group performs a local search, and then exchanges information with other groups [7]. Ant-colony optimization algorithms (ACO) were developed by Dorigo et al. Similar to PSO, they evolve not based on genetics but on social behavior. Unlike PSO, the ACO uses ants to find the shortest route between their ant hill and a source of food; ants can deposit pheromone trails whenever they travel as a form of indirect communication [8].

Generating an ideal random sequence is of great importance in the fields of numerical analysis, sampling and heuristic optimization. Recently, a technique which employs chaotic sequences via the chaos approach (chaotic maps) has gained a lot of attention and been widely applied in different areas, such as the chaotic neural network (CNN) [9], chaotic optimization algorithms (COA) [10,11], nonlinear circuits [12], DNA computing [13], and image processing [14]. All of the above-mentioned methods rely on the same pivotal operation, namely the adoption of a chaotic sequence instead of a random sequence, and thereby improve the results due to the unpredictability of the chaotic sequence [15].

Chaotic can be described as a bounded nonlinear system with deterministic dynamic behavior that has ergodic and stochastic properties [16]. It is very sensitive to the initial conditions and the parameters used. In other word, cause and effect of chaos are not proportional to the small differences in the initial values. In what is called the “butterfly effect”, small variations of an initial variable will result in huge differences in the solutions after some iteration. Mathematically, chaos is random and unpredictable, yet it also possesses an element of regularity.

PSO shows a promising performance on nonlinear function optimization and has thus received much attention [17]. However, the local search capability of PSO is poor [18] since premature convergence occurs often, especially in the case complex multi-peak search problems [19]. In order to overcome these disadvantages of PSO, many improvements have been proposed. Yet another approach introduces a fuzzy system to adapt the inertia weight for three benchmark test functions [20]. Liu et al. proposed center particle swarm optimization (CenterPSO), which introduced a center particle into LDWPSO to improve the performance [17]. Xi et al. proposed an improved Quantum-behaved PSO, which introduces a weight parameter into the calculation of the mean best position in QPSO in order to render the importance of particles in the swarm when they are evolving; this method is called weighted QPSO (WQPSO) [21]. Jiao et al. proposed dynamic inertia weight PSO (IPSO), which uses a dynamic inertia weight to decrease the inertia factor in the velocity update equation of the original PSO [22]. Yang et al. proposed another dynamic inertia weight to modify the velocity update formula in a method called modified particle swarm optimization with dynamic adaptation (DAPSO) [23]. Shelokar et al. proposed particle swarm ant colony optimization (PSACO), a hybrid of particle swarm optimization and ant colony optimization, which uses co-operative, population-based global search swarm intelligence metaheuristics [24]. Zhihua et al. proposed two strategies to improve the exploration and exploitation capability, namely FUSS and RWS. FUSS is a uniform fitness (global optimization or local optima). Recently, numerous improvements, which rely on the chaos approach, have been proposed for PSO in order to overcome this disadvantage. Chaotic maps (including logistic maps) can easily be implemented and avoid entrapment in local optima [30–34]. The inherent characteristics of chaos can enhance PSO by enabling it to escape from local solutions, and thus improve the global search capability of PSO [32]. Logistic maps were introduced in nonlinear dynamics of biological populations evidencing chaotic behavior [35] and are often cited as an archetypal example.

In this paper, we propose chaotic CatfishPSO (C-CatfishPSO), in which chaotic maps are applied to improve the performance of the CatfishPSO algorithm. In CatfishPSO, the catfish effect is applied to improve the performance of particle swarm optimization (PSO). This effect is the result of the introduction of new particles into the search space (“catfish particles”), which replace particles with the worst fitness; these catfish particles are initialized at extreme points of the search space when the fitness of the global best particle has not improved for a certain number of consecutive iterations. This results in further opportunities of finding better solutions for the swarm by guiding the whole swarm to promising new regions of the search space [36]. The logistic map was introduced into our study to improve the search behavior and to prevent entrapment of the particles in a locally optimal solution. The proposed method was applied to several benchmark functions...
from the literature. Statistical analysis of the experimental results show that the performance of C-CatfishPSO is superior to PSO, C-PSO, CatfishPSO, and other advanced PSO methods.

2. Method

2.1. Particle swarm optimization (PSO)

In original PSO [2], each particle is analogous to an individual “fish” in a school of fish. It is a population-based optimization technique, where a population is called a swarm. A swarm consists of N particles moving around in a D-dimensional search space. The position of the ith particle can be represented by \( x_i = (x_{i1}, x_{i2}, \ldots, x_{id}) \). The velocity for the ith particle can be written as \( v_i = (v_{i1}, v_{i2}, \ldots, v_{id}) \). The positions and velocities of the particles are confined within \([X_{min}, X_{max}]^D\) and \([V_{min}, V_{max}]^D\), respectively. Each particle coexists and evolves simultaneously based on knowledge shared with neighboring particles; it makes use of its own memory and knowledge gained by the swarm as a whole to find the best solution. The best previously encountered position of the ith particle is denoted its individual best position \( p_{best} = (p_{i1}, p_{i2}, \ldots, p_{id}) \), a value called \( p_{best} \). The best value of the all individual \( p_{best} \) values is denoted the global best position \( g = (g_1, g_2, \ldots, g_d) \) and called \( g_{best} \). The PSO process is initialized with a population of random particles, and the algorithm then executes a search for optimal solutions by continuously updating generations. At each generation, the position and velocity of the ith particle are updated by \( p_{best} \) and \( g_{best} \) in the swarm. The update equations can be formulated as:

\[
\begin{align*}
t_{id}^{new} &= w \times t_{id}^{old} + c_1 \times r_1 \times (p_{best_{id}} - x_{id}^{old}) + c_2 \times r_2 \times (g_{best_{id}} - x_{id}^{old}), \\
x_{id}^{new} &= x_{id}^{old} + t_{id}^{new},
\end{align*}
\]

(1)

(2)

\( r_1 \) and \( r_2 \) are random numbers between \((0, 1)\), and \( c_1 \) and \( c_2 \) are acceleration constants, which control how far a particle will move in a single generation. Velocities \( t_{id}^{new} \) and \( t_{id}^{old} \) denote the velocities of the new and old particle, respectively. \( x_{id}^{old} \) is the current particle position, and \( x_{id}^{new} \) is the new, updated particle position. The inertia weight \( w \) controls the impact of the previous velocity of a particle on its current one [37]. In general, the inertia weight is decreased linearly from 0.9 to 0.4 throughout the search process to effectively balance the local and global search abilities of the swarm [38]. The equation for the inertia weight \( w \) can be written as:

\[
w = (w_{max} - w_{min}) \times \frac{\text{Iteration}_{max} - \text{Iteration}}{\text{Iteration}_{max}} + w_{min}.
\]

(3)

In Eq. (3), \( w_{max} \) is 0.9, \( w_{min} \) is 0.4 and \( \text{Iteration}_{max} \) is the maximum number of allowed iterations. The pseudo-code of the PSO process is shown below.

---

**PSO pseudo-code**

01: begin
02: Randomly initialize particles swarm
03: while (number of iterations, or the stopping criterion is not met)
04: Evaluate fitness of particle swarm
05: for \( n = 1 \) to number of particles
06: Find \( p_{best} \)
07: Find \( g_{best} \)
08: for \( d = 1 \) to number of dimension of particle
09: update the position of particles by Eq. (1) and (2)
10: next \( d \)
11: next \( n \)
12: update the inertia weight value by Eq. (3)
13: next generation until stopping criterion
14: end

---

2.2. Chaotic particle swarm optimization (C-PSO)

In PSO, the parameters \( w, r_1, \) and \( r_2 \) are the key factors affecting the convergence behavior [39,40]. The inertia weight controls the balance between the global exploration and the local search ability. A large inertia weight favors the global search, while a small inertia weight favors the local search. For this reason, an inertia weight that linearly decreases from 0.9 to 0.4 throughout the search process is usually used [38]. Since logistic maps are frequently used chaotic behavior maps and chaotic sequences can be quickly generated and easily stored, there is no need for storage of long sequences [14]. In C-PSO, sequences generated by the logistic map substitute the random parameters \( r_1 \) and \( r_2 \) in PSO. The parameters \( r_1 \) and \( r_2 \) are modified by the logistic map based on the following equation.
\[
Cr_{(t+1)} = k \times Cr_{(0)} \times (1 - Cr_{(0)}).
\] (4)

In Eq. (4), \(Cr_{(0)}\) is generated randomly for each independent run, with \(Cr_{(0)}\) not being equal to \{0, 0.25, 0.5, 0.75, 1\} and \(k\) equal to 4. The driving parameter \(k\) of the logistic map, controls the behavior of \(Cr_{(t)}\) (as \(t\) goes to infinity) [41].

The velocity update equation for C-PSO can be formulated as:

\[
v_{\text{new}}^{id} = w \times v_{\text{old}}^{id} + c_1 \times Cr \times (p_{\text{best}}^{id} - x_{\text{old}}^{id}) + c_2 \times (1 - Cr) \times (g_{\text{best}}^{id} - x_{\text{old}}^{id}).
\] (5)

In Eq. (5), \(Cr\) is a function based on the results of the logistic map with values between 0.0 and 1.0. Fig. 1 shows the chaotic \(Cr\) value using a logistic map for 300 iterations where \(Cr_{(0)} = 0.001\). The pseudo-code of C-PSO is shown below.

---

### C-PSO pseudo-code

01: begin
02: Randomly initialize particles swarm
03: Randomly generate \(Cr_{(0)}\)
04: while (number of iterations, or the stopping criterion is not met) do
05: Evaluate fitness of particle swarm
06: for \(n = 1\) to number of particles do
07: Find \(p_{\text{best}}\)
08: Find \(g_{\text{best}}\)
09: for \(d = 1\) to number of dimension of particle do
10: update the Chaotic \(Cr\) value by Eq. (4)
11: update the position of particles by Eq. (5) and Eq.(2)
12: next \(d\)
13: next \(n\)
14: update the inertia weight value by Eq. (3)
15: next generation until stopping criterion
16: end

---

### 2.3. Catfish particle swarm optimization (CatfishPSO)

The underlying idea for the development of CatfishPSO was derived from the catfish effect observed when catfish were introduced into large holding tanks of sardines [36]. The catfish in competition with the sardines, stimulate renewed movement amongst the sardines. Similarly, the introduced catfish particles stimulate a renewed search by the other “sardine” particles in CatfishPSO. In other words, the catfish particles can guide particles trapped in a local optimum onto a new regions of the search space, and thus to potentially better particle solutions.

In CatfishPSO, a population is randomly initialized in a first step, and the particles are distributed over the \(D\)-dimensional search space. The position and velocity of each particle are updated by Eqs. (1)–(3). If the distance between \(g_{\text{best}}\) and the surrounding particles is small, each particle is considered a part of the cluster around \(g_{\text{best}}\) and will only move a very small distance in the next generation. To avoid this premature convergence, catfish particles are introduced and replace the 10% of original particles with the worst fitness values of the swarm. These catfish particles are essential for the success of a given optimization task. The pseudo-code for CatfishPSO is shown below. Further details on the CatfishPSO mechanism can be found in Chuang et al. [36].

---

**Fig. 1.** Chaotic \(Cr\) value using a logistic map for 300 iterations; \(Cr_{(0)} = 0.001\).
CatfishPSO Pseudo-code

01: Begin
02: Randomly initialize particles swarm
03: while (number of iterations, or the stopping criterion is not met)
04: Evaluate fitness of particle swarm
05: for n = 1 to number of particles
06: Find \( p_{\text{best}} \)
07: Find \( g_{\text{best}} \)
08: for \( d = 1 \) to number of dimension of particle
09: update the position of particles by Eq. (1) and (2)
10: next \( d \)
11: next \( n \)
12: if fitness of \( g_{\text{best}} \) is the same Seven times then
13: Sort the particle swarm via fitness from best to worst
14: for \( n = \) number of Nine-tenths of particles to number of particles
15: for \( d = 1 \) to number of dimension of particle
16: Randomly select extreme points at Max or Min of the search space
17: Reset the velocity to 0
18: next \( d \)
19: next \( n \)
20: end if
21: update the inertia weight value by Eq. (3)
22: next generation until stopping criterion
23: end

2.4. Chaotic catfish particle swarm optimization (C-CatfishPSO)

In C-CatfishPSO, a logistic map is embedded into CatfishPSO, which updates the parameters \( r_1 \) and \( r_2 \) based on Eq. (4). The logistic map improves the search capability of CatfishPSO significantly. The particle velocities are updated according to Eq. (5). The pseudo-code for C-CatfishPSO is shown below.

C-CatfishPSO Pseudo-code

01: Begin
02: Randomly initialize particles swarm
03: Randomly generate \( Cr_{(0)} \)
04: while (number of iterations, or the stopping criterion is not met)
05: Evaluate fitness of particle swarm
06: for \( n = 1 \) to number of particles
07: Find \( p_{\text{best}} \)
08: Find \( g_{\text{best}} \)
09: for \( d = 1 \) to number of dimension of particle
10: update the Chaotic \( Cr \) value by Eq. (4)
11: update the position of particles by Eqs. (5) and (2)
12: next \( d \)
13: next \( n \)
14: if fitness of \( g_{\text{best}} \) is the same Seven times then
15: Sort the particle swarm via fitness from best to worst
16: for \( n = \) number of Nine-tenths of particles to number of particles
17: for \( d = 1 \) to number of dimension of particle
18: Randomly select extreme points at Max or Min of the search space
19: Reset the velocity to 0
20: next \( d \)
21: next \( n \)
22: end if
23: update the inertia weight value by Eq. (3)
24: next generation until stopping criterion
25: end
3. Numerical simulation

3.1. Benchmark functions

In order to illustrate, compare and analyze the effectiveness and performance of the PSO, C-PSO, CatfishPSO and C-CatfishPSO algorithms for optimization problems, ten representative benchmark functions were used to test the algorithms. These ten benchmark functions are shown below.

Rosenbrock

\[ f_1(x) = \sum_{i=1}^{D-1} \left( 100(x_{i+1} - x_i^2)^2 + (x_i - 1)^2 \right) \]  

Rastrigin

\[ f_2(x) = \sum_{i=1}^{D} (x_i^2 - 10 \cos(2\pi x_i) + 10) \]

Griewark

\[ f_3(x) = \frac{1}{4000} \sum_{i=1}^{D} x_i^2 - \prod_{i=1}^{D} \cos \left( \frac{x_i}{\sqrt{i}} \right) + 1 \]

Sphere

\[ f_4(x) = \sum_{i=1}^{D} x_i^2 \]

Table 1

<table>
<thead>
<tr>
<th>Name</th>
<th>Trait</th>
<th>Search space</th>
<th>Asymmetric initialization range</th>
<th>Xmin</th>
<th>Xmax</th>
<th>Optimum</th>
</tr>
</thead>
<tbody>
<tr>
<td>Rosenbrock</td>
<td>Unimodal</td>
<td>–100x, 1000</td>
<td>15x, 30</td>
<td>–100</td>
<td>100</td>
<td>0</td>
</tr>
<tr>
<td>Rastrigin</td>
<td>Multimodal</td>
<td>–10x, 10</td>
<td>2.56x, 5.12</td>
<td>–10</td>
<td>10</td>
<td>0</td>
</tr>
<tr>
<td>Griewark</td>
<td>Multimodal</td>
<td>–600x, 600</td>
<td>300x, 600</td>
<td>–600</td>
<td>600</td>
<td>0</td>
</tr>
<tr>
<td>Sphere</td>
<td>Unimodal</td>
<td>–100x, 1000</td>
<td>50x, 100</td>
<td>–100</td>
<td>100</td>
<td>0</td>
</tr>
<tr>
<td>Ackley</td>
<td>Multimodal</td>
<td>–100x, 1000</td>
<td>50x, 100</td>
<td>–100</td>
<td>100</td>
<td>0</td>
</tr>
<tr>
<td>Schwefel</td>
<td>Multimodal</td>
<td>–500x, 500</td>
<td>–500x, –250</td>
<td>–500</td>
<td>500</td>
<td>0</td>
</tr>
<tr>
<td>Ellipsoid</td>
<td>Unimodal</td>
<td>–100x, 1000</td>
<td>50x, 100</td>
<td>–100</td>
<td>100</td>
<td>0</td>
</tr>
<tr>
<td>Sum of difference power</td>
<td>Unimodal</td>
<td>–3x, 3</td>
<td>1.5x, 3.0</td>
<td>–3</td>
<td>3</td>
<td>0</td>
</tr>
<tr>
<td>Cigar</td>
<td>Unimodal</td>
<td>–100x, 1000</td>
<td>50x, 100</td>
<td>–100</td>
<td>100</td>
<td>0</td>
</tr>
<tr>
<td>Ridge</td>
<td>Unimodal</td>
<td>–100x, 1000</td>
<td>50x, 100</td>
<td>–100</td>
<td>100</td>
<td>0</td>
</tr>
</tbody>
</table>

Table 2

Mean function value for Rosenbrock function.

<table>
<thead>
<tr>
<th>Pop.</th>
<th>Dim.</th>
<th>Gen.</th>
<th>Optimal</th>
<th>PSO</th>
<th>C-PSO</th>
<th>CatfishPSO</th>
<th>C-CatfishPSO</th>
</tr>
</thead>
<tbody>
<tr>
<td>20</td>
<td>10</td>
<td>1000</td>
<td>0</td>
<td>95.893±2330.136</td>
<td>28.178±426.317</td>
<td>5.855±0.413</td>
<td>3.597±3.708</td>
</tr>
<tr>
<td>20</td>
<td>1500</td>
<td>0</td>
<td>167.604±318.927</td>
<td>27.770±248.084</td>
<td>16.274±0.385</td>
<td>4.527±6.290</td>
<td></td>
</tr>
<tr>
<td>40</td>
<td>10</td>
<td>1000</td>
<td>0</td>
<td>69.868±187.713</td>
<td>7.116±183.456</td>
<td>5.443±0.392</td>
<td>2.994±3.082</td>
</tr>
<tr>
<td>20</td>
<td>1500</td>
<td>0</td>
<td>135.475±269.301</td>
<td>24.960±251.540</td>
<td>15.993±0.445</td>
<td>4.272±4.525</td>
<td></td>
</tr>
<tr>
<td>30</td>
<td>2000</td>
<td>0</td>
<td>207.524±346.617</td>
<td>42.313±338.070</td>
<td>26.238±0.523</td>
<td>2.085±4.835</td>
<td></td>
</tr>
<tr>
<td>80</td>
<td>10</td>
<td>1000</td>
<td>0</td>
<td>39.886±094.309</td>
<td>7.185±088.879</td>
<td>5.086±0.418</td>
<td>1.628±2.667</td>
</tr>
<tr>
<td>20</td>
<td>1500</td>
<td>0</td>
<td>105.067±237.020</td>
<td>15.334±072.234</td>
<td>15.761±0.490</td>
<td>3.173±3.092</td>
<td></td>
</tr>
<tr>
<td>30</td>
<td>2000</td>
<td>0</td>
<td>156.822±258.627</td>
<td>27.890±313.19</td>
<td>26.224±0.584</td>
<td>1.081±2.949</td>
<td></td>
</tr>
<tr>
<td>160</td>
<td>10</td>
<td>1000</td>
<td>0</td>
<td>30.000±76.045</td>
<td>6.956±056.861</td>
<td>4.742±4.885</td>
<td>1.278±2.382</td>
</tr>
<tr>
<td>20</td>
<td>1500</td>
<td>0</td>
<td>70.063±125.544</td>
<td>26.066±343.886</td>
<td>15.501±0.539</td>
<td>0.987±2.586</td>
<td></td>
</tr>
<tr>
<td>30</td>
<td>2000</td>
<td>0</td>
<td>105.13±185.19</td>
<td>27.065±057.849</td>
<td>26.005±0.631</td>
<td>0.558±1.955</td>
<td></td>
</tr>
<tr>
<td>Average</td>
<td></td>
<td></td>
<td></td>
<td>121.014±231.632</td>
<td>22.650±165.380</td>
<td>15.817±0.480</td>
<td>2.179±3.795</td>
</tr>
</tbody>
</table>

C-CatfishPSO V.S. PSO  C-PSO  CatfishPSO  C-CatfishPSO

<table>
<thead>
<tr>
<th>R*</th>
<th>R*</th>
<th>R*</th>
<th>P-value</th>
<th>Significant ((\alpha = 0.05))</th>
</tr>
</thead>
<tbody>
<tr>
<td>12</td>
<td>0</td>
<td>0</td>
<td>0.002</td>
<td>YES</td>
</tr>
<tr>
<td>12</td>
<td>0</td>
<td>0</td>
<td>0.002</td>
<td>YES</td>
</tr>
<tr>
<td>12</td>
<td>0</td>
<td>0</td>
<td>0.002</td>
<td>YES</td>
</tr>
</tbody>
</table>
Ackley

\[ f_5(x) = -20 \exp \left( -0.2 \sqrt{\frac{1}{n} \sum_{i=1}^{n} x_i^2} \right) - \exp \left( \frac{1}{n} \sum_{i=1}^{n} \cos(2\pi x_i) \right) + 20 + e \]  

(10)

Schwefel

\[ f_6(x) = 418.9809D - \sum_{i=1}^{D} x_i \sin \left( \sqrt{|x_i|} \right) \]  

(11)

Ellipsoid

\[ f_7(x) = \sum_{i=1}^{D} |x_i|^2 \]  

(12)

Sum of difference power

\[ f_8(x) = \sum_{i=1}^{D} |x_i|^{1/2} \]  

(13)

### Table 3

Mean function value for Rastrigin function.

<table>
<thead>
<tr>
<th>Pop.</th>
<th>Dim.</th>
<th>Gen.</th>
<th>Optimal</th>
<th>PSO</th>
<th>C-PSO</th>
<th>CatfishPSO</th>
<th>C-CatfishPSO</th>
</tr>
</thead>
<tbody>
<tr>
<td>20</td>
<td>10</td>
<td>1000</td>
<td>0</td>
<td>5.128±0.627</td>
<td>4.399±0.753</td>
<td>0.000±0.000</td>
<td>0.000±0.000</td>
</tr>
<tr>
<td>20</td>
<td>1500</td>
<td>0</td>
<td>22.021±0.716</td>
<td>12.675±0.048</td>
<td>0.000±0.000</td>
<td>0.000±0.000</td>
<td></td>
</tr>
<tr>
<td>30</td>
<td>2000</td>
<td>0</td>
<td>47.735±12.037</td>
<td>22.693±10.869</td>
<td>0.000±0.000</td>
<td>0.000±0.000</td>
<td></td>
</tr>
<tr>
<td>40</td>
<td>10</td>
<td>1000</td>
<td>0</td>
<td>3.375±0.748</td>
<td>2.934±0.176</td>
<td>0.000±0.000</td>
<td>0.000±0.000</td>
</tr>
<tr>
<td>20</td>
<td>1500</td>
<td>0</td>
<td>16.446±0.5294</td>
<td>10.061±0.5714</td>
<td>0.000±0.000</td>
<td>0.000±0.000</td>
<td></td>
</tr>
<tr>
<td>30</td>
<td>2000</td>
<td>0</td>
<td>36.987±9.692</td>
<td>18.849±9.129</td>
<td>0.000±0.000</td>
<td>0.000±0.000</td>
<td></td>
</tr>
<tr>
<td>80</td>
<td>10</td>
<td>1000</td>
<td>0</td>
<td>1.260±0.1476</td>
<td>1.260±0.1259</td>
<td>0.000±0.000</td>
<td>0.000±0.000</td>
</tr>
<tr>
<td>20</td>
<td>1500</td>
<td>0</td>
<td>12.675±4.3777</td>
<td>8.197±4.355</td>
<td>0.000±0.000</td>
<td>0.000±0.000</td>
<td></td>
</tr>
<tr>
<td>30</td>
<td>2000</td>
<td>0</td>
<td>29.579±7.703</td>
<td>15.560±7.870</td>
<td>0.000±0.000</td>
<td>0.000±0.000</td>
<td></td>
</tr>
<tr>
<td>160</td>
<td>10</td>
<td>1000</td>
<td>0</td>
<td>1.260±0.1476</td>
<td>1.260±0.1259</td>
<td>0.000±0.000</td>
<td>0.000±0.000</td>
</tr>
<tr>
<td>20</td>
<td>1500</td>
<td>0</td>
<td>9.517±3.119</td>
<td>6.660±0.040.02</td>
<td>0.000±0.000</td>
<td>0.000±0.000</td>
<td></td>
</tr>
<tr>
<td>30</td>
<td>2000</td>
<td>0</td>
<td>23.468±6.178</td>
<td>13.147±0.8279</td>
<td>0.000±0.000</td>
<td>0.000±0.000</td>
<td></td>
</tr>
</tbody>
</table>

**Average**

17.530±5.190 | 9.877±5.373 | 0.000±0.000 | 0.000±0.000

C-CatfishPSO V.S. | R* \( \neq \) R* | R* \( \neq \) R* | P-value | Significant (\( \alpha = 0.05 \))
| PSO | 12 | 0 | 0 | 0.002 | YES |
| C-PSO | 12 | 0 | 0 | 0.002 | YES |
| CatfishPSO | 0 | 0 | 12 | 1.000 | NO |

### Table 4

Mean function value for Griewark function.

<table>
<thead>
<tr>
<th>Pop.</th>
<th>Dim.</th>
<th>Gen.</th>
<th>Optimal</th>
<th>PSO</th>
<th>C-PSO</th>
<th>CatfishPSO</th>
<th>C-CatfishPSO</th>
</tr>
</thead>
<tbody>
<tr>
<td>20</td>
<td>10</td>
<td>1000</td>
<td>0</td>
<td>0.102±0.056</td>
<td>0.061±0.050</td>
<td>0.000±0.000</td>
<td>0.000±0.000</td>
</tr>
<tr>
<td>20</td>
<td>1500</td>
<td>0</td>
<td>0.480±0.361</td>
<td>0.002±0.009</td>
<td>0.000±0.000</td>
<td>0.000±0.000</td>
<td></td>
</tr>
<tr>
<td>30</td>
<td>2000</td>
<td>0</td>
<td>2.455±14.650</td>
<td>0.361±5.692</td>
<td>0.000±0.000</td>
<td>0.000±0.000</td>
<td></td>
</tr>
<tr>
<td>40</td>
<td>10</td>
<td>1000</td>
<td>0</td>
<td>0.087±0.043</td>
<td>0.067±0.035</td>
<td>0.000±0.000</td>
<td>0.000±0.000</td>
</tr>
<tr>
<td>20</td>
<td>1500</td>
<td>0</td>
<td>0.120±0.2846</td>
<td>0.096±4.87</td>
<td>0.000±0.000</td>
<td>0.000±0.000</td>
<td></td>
</tr>
<tr>
<td>30</td>
<td>2000</td>
<td>0</td>
<td>1.010±9.52</td>
<td>0.273±9.57</td>
<td>0.000±0.000</td>
<td>0.000±0.000</td>
<td></td>
</tr>
<tr>
<td>80</td>
<td>10</td>
<td>1000</td>
<td>0</td>
<td>0.074±0.032</td>
<td>0.061±0.027</td>
<td>0.000±0.000</td>
<td>0.000±0.000</td>
</tr>
<tr>
<td>20</td>
<td>1500</td>
<td>0</td>
<td>0.030±0.026</td>
<td>0.011±0.037</td>
<td>0.000±0.000</td>
<td>0.000±0.000</td>
<td></td>
</tr>
<tr>
<td>30</td>
<td>2000</td>
<td>0</td>
<td>0.193±0.034</td>
<td>0.091±2.858</td>
<td>0.000±0.000</td>
<td>0.000±0.000</td>
<td></td>
</tr>
<tr>
<td>160</td>
<td>10</td>
<td>1000</td>
<td>0</td>
<td>0.066±0.028</td>
<td>0.058±0.031</td>
<td>0.000±0.000</td>
<td>0.000±0.000</td>
</tr>
<tr>
<td>20</td>
<td>1500</td>
<td>0</td>
<td>0.032±0.027</td>
<td>0.016±0.038</td>
<td>0.000±0.000</td>
<td>0.000±0.000</td>
<td></td>
</tr>
<tr>
<td>30</td>
<td>2000</td>
<td>0</td>
<td>0.012±0.015</td>
<td>0.002±0.037</td>
<td>0.000±0.000</td>
<td>0.000±0.000</td>
<td></td>
</tr>
</tbody>
</table>

**Average**

0.388±0.131 | 0.135±1.385 | 0.000±0.000 | 0.000±0.000

C-CatfishPSO V.S. | R* \( \neq \) R* | R* \( \neq \) R* | P-value | Significant (\( \alpha = 0.05 \))
| PSO | 12 | 0 | 0 | 0.002 | YES |
| C-PSO | 12 | 0 | 0 | 0.002 | YES |
| CatfishPSO | 0 | 0 | 12 | 1.000 | NO |
References