Available online at www.sciencedirect.com
SCIENCE DIRECT®
Chaos, Solitons and Fractals 25 (2005) 1261–1271
CHAOS SOLITONS & FRACTALS
www.elsevier.com/locate/chaos

Improved particle swarm optimization combined with chaos

Bo Liu [a,*], Ling Wang [a], Yi-Hui Jin [a], Fang Tang [b], De-Xian Huang [a]

[a] Department of Automation, Tsinghua University, Beijing 100084, China
[b] Department of Physics, Beijing University of Aeronautics and Astronautics, Beijing 100083, China

Accepted 29 November 2004

Abstract

As a novel optimization technique, chaos has gained much attention and some applications during the past decade. For a given energy or cost function, by following chaotic ergodic orbits, a chaotic dynamic system may eventually reach the global optimum or its good approximation with high probability. To enhance the performance of particle swarm optimization (PSO), which is an evolutionary computation technique through individual improvement plus population cooperation and competition, hybrid particle swarm optimization algorithm is proposed by incorporating chaos. Firstly, adaptive inertia weight factor (AIWF) is introduced in PSO to efficiently balance the exploration and exploitation abilities. Secondly, PSO with AIWF and chaos are hybridized to form a chaotic PSO (CPSO), which reasonably combines the population-based evolutionary searching ability of PSO and chaotic searching behavior. Simulation results and comparisons with the standard PSO and several meta-heuristics show that the CPSO can effectively enhance the searching efficiency and greatly improve the searching quality.

1. Introduction

Chaos is a kind of characteristic of non-linear systems, which is a bounded unstable dynamic behavior that exhibits sensitive dependence on initial conditions and includes infinite unstable periodic motions. Although it appears to be stochastic, it occurs in a deterministic non-linear system under deterministic conditions. In recently years, growing interests from physics, chemistry, biology and engineering have stimulated the studies of chaos for control [1,2], synchronization [3] and optimization [4–6]. Due to the easy implementation and special ability to avoid being trapped in local optima, chaos has been a novel optimization technique and chaos-based searching algorithms have aroused intense interests [7].

Currently, there are two ways of application of chaos for optimization. The first one is chaotic neural network (CNN) [4] by incorporating chaotic dynamics into neural network. Through the rich non-equilibrium dynamics with various concomitant attractors, chaotic neuron-dynamics can be used to continually search for the global optimum by following chaotic ergodic orbits. The other one is chaotic optimization algorithm (COA) [5,6] based on chaotic evolution of variables. The simple philosophy of the COA includes two main steps: firstly mapping from the chaotic space to the solution space, and then searching optimal regions using chaotic dynamics instead of random search. However,

* Corresponding author.
E-mail address: liub01@mails.tsinghua.edu.cn (B. Liu).

0960-0779/$ - see front matter © 2005 Elsevier Ltd. All rights reserved.
doi:10.1016/j.chaos.2004.11.095

simple CNN and COA often need a large number of iterations to reach the global optimum and are sensitive to the initial conditions.

Recently, a new evolutionary technique, the particle swarm optimization (PSO), is proposed [8,9] as an alternative to genetic algorithm (GA) [10]. Its development was based on observations of the social behavior of animals such as bird flocking, fish schooling, and swarm theory. PSO is initialized with a population of random solutions. Each individual is assigned with a randomized velocity according to its own and its companions' flying experiences, and the individuals, called particles, are then flown through hyperspace. Compared with GA, PSO has some attractive characteristics. It has memory, so knowledge of good solutions is retained by all particles; whereas in GA, previous knowledge of the problem is destroyed once the population changes. It has constructive cooperation between particles, particles in the swarm share information between them. Due to the simple concept, easy implementation and quick convergence, nowadays PSO has gained much attention and wide applications in different fields [11]. However, the performance of simple PSO greatly depends on its parameters, and it often suffers the problem of being trapped in local optima so as to be premature convergence [12].

In this paper, an adaptive inertia weight factor (AIWF) is proposed to efficiently control the global search and convergence to the global best solution. Moreover, chaos is incorporated into PSO to construct a chaotic PSO (CPSO), where the parallel population-based evolutionary searching ability of PSO and chaotic searching behavior are reasonably combined. Simulation results and comparisons demonstrate the effectiveness and efficiency of the proposed CPSO.

The remaining content of this paper is organized as follows. In Section 2, the standard PSO is overviewed. Then the PSO with adaptive inertia weight factor (AIWF) and the CPSO are proposed in Section 3. Numerical simulations and comparisons are provided in Section 4. Finally, Section 5 provides some concluding remarks.

2. Simple PSO

Many real optimization problems can be formulated as the following functional optimization problem.

$$\begin{aligned} &\min \quad f(X), X = [x_1, \ldots, x_n], \\ &\text{s.t.} \quad x_i \in [a_i, b_i], i = 1, 2, \ldots, n, \end{aligned} \tag{1}$$

where f is the objective function, and X is the decision vector consisting of n variables.

PSO is a population-based optimization technique proposed firstly for the above unconstrained minimization problem. In a PSO system, multiple candidate solutions coexist and collaborate simultaneously. Each solution called a "particle", flies in the problem search space looking for the optimal position to land. A particle, as time passes through its quest, adjusts its position according to its own "experience" as well as the experience of neighboring particles. Tracking and memorizing the best position encountered build particle's experience. For that reason, PSO possesses a memory (i.e. every particle remembers the best position it reached during the past). PSO system combines local search method (through self experience) with global search methods (through neighboring experience), attempting to balance exploration and exploitation.

A particle status on the search space is characterized by two factors: its position and velocity, which are updated by following equations.

$$V_i[t+1] = wV_i[t] + c_1 \times \text{rand}(\cdot) \times (P_i - X_i) + c_2 \times \text{Rand}(\cdot) \times (P_g - X_i), \tag{2}$$

$$X_i[t+1] = X_i[t] + V_i[t+1], \tag{3}$$

where, $V_i = [v_{i,1}, v_{i,2}, \ldots, v_{i,n}]$ called the velocity for particle i, which represents the distance to be traveled by this particle from its current position; $X_i = [x_{i,1}, x_{i,2}, \ldots, x_{i,n}]$ represents the position of particle i; P_i represents the best previous position of particle i (i.e. local-best position or its experience); P_g represents the best position among all particles in the population $\mathbf{X} = [X_1, X_2, \ldots, X_N]$ (i.e. global-best position); *Rand*() and *rand*() are two independently uniformly distributed random variables with range $[0, 1]$; c_1 and c_2 are positive constant parameters called acceleration coefficients which control the maximum step size; w is called the inertia weight that controls the impact of previous velocity of particle on its current one. In the standard PSO, Eq. (2) is used to calculate the new velocity according to its previous velocity and to the distance of its current position from both its own best historical position and its neighbors' best position. Generally, the value of each component in V_i can be clamped to the range $[-v_{\max}, v_{\max}]$ to control excessive roaming of particles outside the search space. Then the particle flies toward a new position according Eq. (3). This process is repeated until a user-defined stopping criterion is reached. The simple PSO is illustrated in Fig. 1, where N denotes the size of population, f_i represents the function value of the ith particle, and $f_{\text{best}}[i]$ represents the local-best function value for the best position visited by the ith particle. Interested readers could refer to Kennedy and Eberhart [9] for more detail.

Step 1 (Initialization): For each particle i in the population:

Step 1.1: initialize X_i randomly.

Step 1.2: initialize V_i randomly.

Step 1.3: evaluate f_i.

Step 1.4: initialize P_g with the index of the particle with the best function value among the population.

Step 1.5: initialize P_i with a copy of X_i, $\forall i \leq N$.

Step 2: Repeat until a stopping criterion is satisfied:

Step 2.1: find P_g such that $f[P_g] \leq f_i, \forall i \leq N$.

Step 2.2: for each particle i, $P_i = X_i$ *if* $f_i < f_{best}[i], \forall i \leq N$.

Step 2.3: for each particle i, update V_i and X_i according to equation2 and 3.

Step 2.4: evaluate f_i for all particles.

Fig. 1. Simple PSO algorithm.

3. Chaotic PSO (CPSO)

3.1. Adaptive inertia weight factor (AIWF)

In PSO, proper control of global exploration and local exploitation is crucial in finding the optimum solution efficiently [13]. Obviously, the performance of PSO greatly depends on its parameters. It is clear that the first part of Eq. (2) represents the influence of previous velocity, which provides the necessary momentum for particles to roam across the search space. The inertia weight w is the modulus that controls the impact of previous velocity on the current one. So, the balance between exploration and exploitation in PSO is dictated by the value of w. Thus proper control of the inertia weight is very important to find the optimum solution accurately and efficiently.

It is regarded that a larger inertia weight pressures towards global exploration, while a smaller inertia weight pressures toward fine-tuning of the current search area. Thus, Shi and Eberhart [14] made a significant improvement in the performance of the PSO with a linearly varying inertia weight over the generations, which linearly vary from 0.9 at the beginning of the search to 0.4 at the end. To achieve trade-off between exploration and exploitation, we set w varying adaptively in response to the objective values of the particles. In particular, the adaptive inertia weight factor (AIWF) is determined as follows.

$$w = \begin{cases} w_{\min} + \frac{(w_{\max} - w_{\min})(f - f_{\min})}{f_{\text{avg}} - f_{\min}}, & f \leqslant f_{\text{avg}}, \\ w_{\max}, & f > f_{\text{avg}}, \end{cases} \tag{4}$$

where $w_{\max}$ and $w_{\min}$ denote the maximum and minimum of w respectively, f is the current objective value of the particle, f_{avg} and $f_{\min}$ are the average and minimum objective values of all particles, respectively.

Under the guidance of Eq. (4), w is varied depending on the objective value of the particle so that particles with low objective values can be 'protected' while particles with objective values over average will be 'disrupted'. That is, good particles tend to perform exploitation to refine results by local search, while bad particles tend to perform large modification to explore space with large step. In other words, AIWF provides a good way to maintain population diversity and to sustain good convergence capacity.

3.2. Chaotic local search (CLS)

To enrich the searching behavior and to avoid being trapped into local optimum, chaotic dynamics is incorporated into the above PSO with AIWF. In this paper, the well-known logistic equation [15], which exhibits the sensitive dependence on initial conditions, is employed for constructing hybrid PSO. The logistic equation is defined as follows.

$$x_{n+1} = \mu \cdot x_n(1 - x_n), \quad 0 \leqslant x_0 \leqslant 1, \tag{5}$$

where μ is the control parameter, x is a variable and $n = 0, 1, 2, \ldots$ Although the above equation is deterministic, it exhibits chaotic dynamics when $\mu = 4$ and $x_0 \notin \{0, 0.25, 0.5, 0.75, 1\}$. That is, it exhibits the sensitive dependence on initial conditions, which is the basic characteristic of chaos. A minute difference in the initial value of the chaotic variable would result in a considerable difference in its long time behavior. The track of chaotic variable can travel ergodically over the whole search space. In general, the above chaotic variable has special characters, i.e. ergodicity, pseudo-randomness and irregularity.

The process of the chaotic local search could be defined through the following equation:

$$cx_i^{(k+1)} = 4cx_i^{(k)}(1 - cx_i^{(k)}), \quad i = 1, 2, \ldots, n, \tag{6}$$

where cx_i is the ith chaotic variable, and k denotes the iteration number. Obviously, $cx_i^{(k)}$ is distributed in the range $(0, 1.0)$ under the conditions that the initial $cx_i^{(0)} \in (0, 1)$ and that $cx_i^{(0)} \notin \{0.25, 0.5, 0.75\}$. Fig. 2 shows its chaotic dynamics, where $cx_i^0 = 0.01$, $k = 300$.

The procedures of chaotic local search (CLS) can be illustrated as follows:

Step 1: Setting $k = 0$, and mapping the decision variables $x_i^{(k)}$, $i = 1, 2, \ldots, n$ among the intervals $(x_{\min,i}, x_{\max,i})$, $i = 1, 2, \ldots, n$ to chaotic variables $cx_i^{(k)}$ located in the interval $(0, 1)$ using the following equation.

$$cx_i^{(k)} = \frac{x_i^{(k)} - x_{\min,i}}{x_{\max,i} - x_{\min,i}}, \quad i = 1, 2, \ldots, n. \tag{7}$$

Step 2: Determining the chaotic variables $cx_i^{(k+1)}$ for the next iteration using the logistic equation according to $cx_i^{(k)}$.

Step 3: Converting the chaotic variables $cx_i^{(k+1)}$ to decision variables $x_i^{(k+1)}$ using the following equation.

$$x_i^{(k+1)} = x_{\min,i} + cx_i^{(k+1)}(x_{\max,i} - x_{\min,i}), \quad i = 1, 2, \ldots, n. \tag{8}$$

Step 4: Evaluating the new solution with decision variables $x_i^{(k+1)}, i = 1, 2, \ldots, n$.

Step 5: If the new solution is better than $X^{(0)} = [x_1^{(0)}, \ldots, x_n^{(0)}]$ or the predefined maximum iteration is reached, output the new solution as the result of the CLS; otherwise, let $k = k + 1$ and go back to Step 2.

3.3. Chaotic PSO (CPSO)

Based on the proposed PSO with AIWF and the chaotic local search, a two-phased iterative strategy named Chaotic PSO (CPSO) is proposed, in which PSO with AIWF is applied to perform global exploration and CLS is employed to perform locally oriented search (exploitation) for the solutions resulted by PSO. The procedure of CPSO is described in Fig. 3.

It can be seen that PSO with AIFW is used for exploration by updating particle swarm, and chaos dynamic is applied for exploitation by locally modified the best particle resulted by PSO. Besides, to maintain population diversity, several new particles are randomly generated and incorporated in the new population. Especially, the region for generating new particles is dynamically decreased so as to speed up convergence.

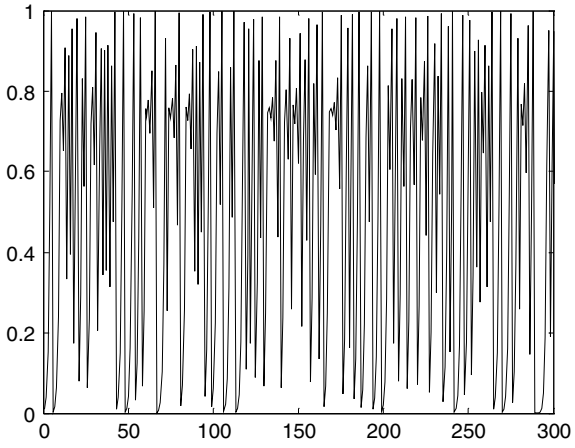

Fig. 2. Dynamics of logistic map.

Step 1 (Initialization): Set $k = 0$. For each particle i in the population:

Step 1.1: initialize X_i randomly.

Step 1.2: initialize V_i randomly.

Step 1.3: evaluate f_i.

Step 1.4: initialize P_g with the index of the particle with the best function value among the population.

Step 1.5: initialize P_i with a copy of X_i, $\forall i \leq N$.

Step 2: Repeat until a stopping criterion is satisfied:

Step 2.1: find P_g such that $f[P_g] \leq f_i, \forall i \leq N$.

Step 2.2: for each particle i, $P_i = X_i$ *if* $f_i < f_{best}[i], \forall i \leq N$.

Step 2.3: for each particle i, update V_i and X_i according to equation 2, 3 and 4.

Step 2.4: evaluate f_i for all particles.

Step 3: Reserve the top $N/5$ particles.

Step 4: Implement the chaotic local search (CLS) for the best particle, and update the best particle using the result of CLS with variables $x_{g,i}^{(k)}, i = 1,2,...,n$.

Step 5: If a stopping criterion is satisfied, output the solution found best so far.

Step 6: Decrease the search space:

$$x_{\min i} = \max\left(x_{\min i}, x_{g,i}^{(k)} - r\left(x_{\max i} - x_{\min i}\right)\right), \quad 0 < r < 1$$

$$x_{\max i} = \min\left(x_{\max i}, x_{g,i}^{(k)} + r\left(x_{\max i} - x_{\min i}\right)\right), \quad 0 < r < 1$$

Step 7: Randomly generate $4N/5$ new particles within the decreased search space and evaluate them.

Step 8: Construct the new population consisting of the $4N/5$ new particles and the old top $N/5$ particles in which the best particle is replaced by the result of CLS.

Step 9: Let $k = k + 1$ and go back to step 2.

Fig. 3. CPSO algorithm.

4. Numerical simulation

4.1. Experimental setup

To test the performance of the proposed algorithms, six famous benchmark optimization problems [16] are used, which are described in Appendix A.

Firstly, we compare the CPSO with the standard PSO [9] and GA [10]. In both PSO and CPSO, the population size is 20, c_1 and c_2 are set to 2.0, and $v_{\max}$ is clamped to be 15% of the search space. The standard PSO [9] uses a linearly varying inertia weight over the generations, varying from 1.2 at the beginning of the search to 0.2 at the end. The CPSO uses the AIWF defined in Eq. (4) with $w_{\max} = 1.2$ and $w_{\min} = 0.2$. The GA [10] with a population of 100 is real-valued with random initialization, tournament selection with tournament size four, a 2-element elitist strategy, arithmetic crossover with random weight, Gaussian mutation with distribution $N(0,a) = \frac{1}{\sqrt{2\pi a}} \exp(-x^2/2a)$, where a is linearly decreasing from 1 to 0. Crossover and mutation probabilities are set as 0.8 and $1/n$ respectively, where n is the dimension of the problem.

4.2. Numerical results

4.2.1. Fixed-iteration results

Fixed the total number of function evaluation as 2000, Table 1 lists the average best function value and the standard deviation of 50 independent runs. Figs. 4–9 show the performances of the above algorithms for solving the six functions.

Table 1
Fixed-iteration results of 50 runs

f	CPSO	PSO	GA
f_{GP}	3.0000 ± 5.0251e−15	4.6202 ± 11.4554	3.1471 ± 0.9860
f_{BR}	0.3979 ± 3.3645e−16	0.4960 ± 0.3703	0.4021 ± 0.0153
f_{H3}	−3.8610 ± 0.0033	−3.8572 ± 0.0035	−3.8571 ± 0.0070
f_{H6}	−3.1953 ± 0.1352	−2.8943 ± 0.3995	−3.0212 ± 0.4291
f_{RA}	−1.9940 ± 0.0248	−1.9702 ± 0.0366	−1.9645 ± 0.0393
f_{SH}	−186.7274 ± 0.0218	−180.3265 ± 10.1718	−182.1840 ± 5.3599

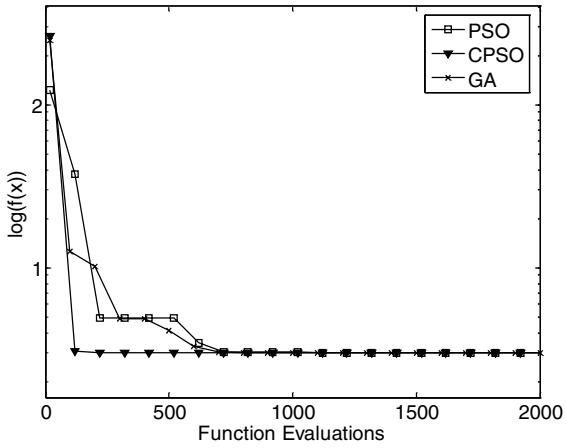

Fig. 4. Comparison of the algorithms for GP.

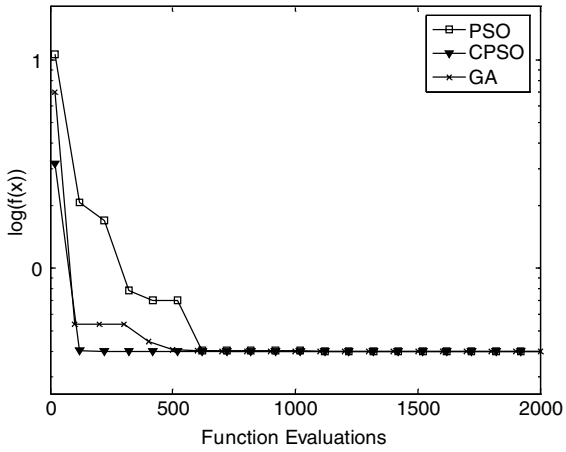

Fig. 5. Comparison of the algorithms for BR.

From Table 1, it can be seen that the results of CPSO are much closer to the theoretical optima, and CPSO is superior to PSO and GA in term of searching quality and derivation of the results. So, it is concluded that CPSO is more effective and it is more robust on initial conditions. From Figs. 4–9, it can be seen that the varying curves of objective values using CPSO descend much faster than those using PSO and GA, and the objective values descent to lower level by using CPSO than by using PSO and GA. So, it is concluded that CPSO is more efficient than PSO and GA, and it is also concluded that the final searching quality of CPSO is better than PSO and GA.

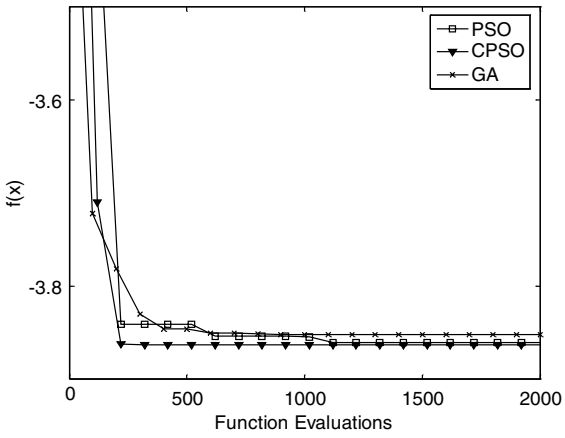

Fig. 6. Comparison of the algorithms for H3.

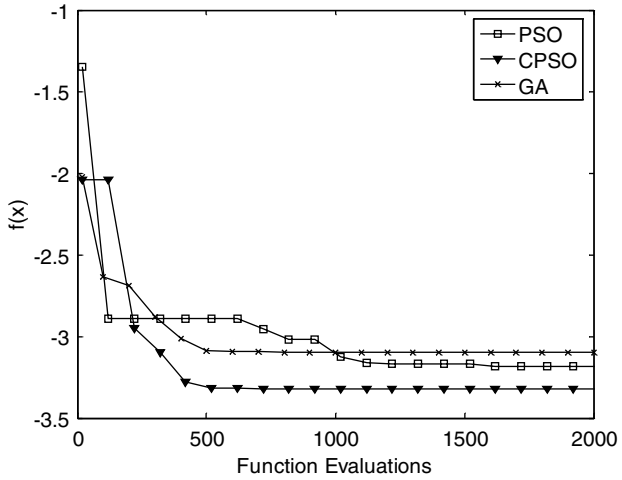

Fig. 7. Comparison of the algorithms for H6.

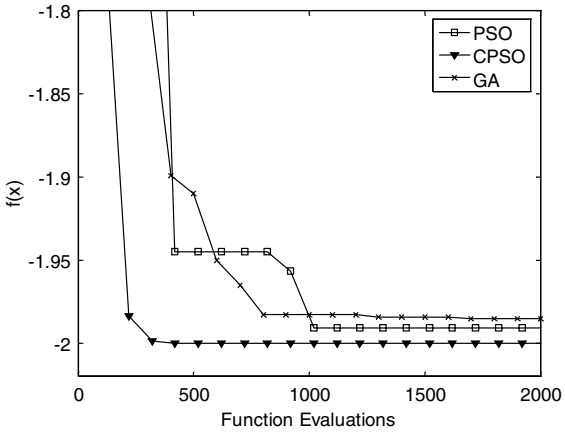

Fig. 8. Comparison of the algorithms for RA.

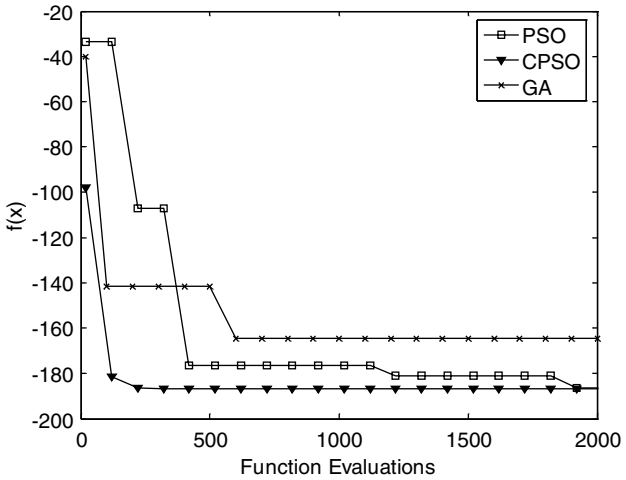

Fig. 9. Comparison of the algorithms for SH.

4.2.2. Robustness analysis

Setting the total number of function evaluation as 2×10^3, we run the three algorithms independently each with 50 times. If the final searching quality is within 3.5% of the global optimal value, we call it as a success run and its evaluation number will be stored. We define two indexes named "succeed ratio (SR)" and "average valid evaluation number (AVEN)" as follows.

$$\mathrm{SR} = \frac{N_v}{50} \times 100\%, \tag{9}$$

$$\mathrm{AVEN} = \frac{\sum_{i=1}^{N_v} n_i}{N_v}, \tag{10}$$

where N_v denotes the number of success run among 50 independent runs, n_i denotes the number of function evaluation of the ith success run.

Table 2 lists the SR and AVEN of the three algorithms when solving the six functions. From Table 2, it can be seen that CPSO can find global optima with very high probability for every function even with small function evaluation number. Besides, for those valid runs, CPSO costs the smallest average function evaluation number. So, it is concluded that CPSO is much effective and reliable for complex numerical optimization.

To further show the effectiveness of CPSO, we carry out some comparisons with several other meta-heuristics, such as pure random search (PRS) [17], multistart (MS) [18], simulated annealing (SA) [18], taboo search (TS) [19], chaotic simulated annealing (CSA) [20], PSO [9] and GA [10]. Table 3 lists the "AVEN" indexes of each algorithm for solving the six functions. Seen from the results, it can be concluded that CPSO is more efficient.

Table 2
Robustness analysis

f	CPSO		PSO		GA	
	SR	AVEN	SR	AVEN	SR	AVEN
f_{GP}	100	192	98	1397	98	536
f_{BR}	100	154	94	743	92	1682
f_{H3}	90	119	96	183	16	112
f_{H6}	96	2551	26	3796	94	5727
f_{RA}	98	653	100	1160	84	238
f_{SH}	100	360	98	1337	98	1516
Average	97	672	85	1436	80	1635

Table 3
Average number of function evaluations in optimization of test functions

Method	f_{GP}	f_{BR}	f_{H3}	f_{H6}	f_{RA}	f_{SH}
PRS [17]	5125	4850	5280	18,090	5964	6700
MS [18]	4400	1600	2500	6000	N/A	N/A
SA1 [18]	5439	2700	3416	3975	N/A	241,215
SA2 [18]	563	505	1459	4648	N/A	780
TS [19]	486	492	508	2845	540	727
CSA2 [20]	300	281	379	1865	441	289
GA	536	1682	112	5727	238	1516
PSO	1397	743	183	3796	1160	1337
CPSO	192	154	119	2551	653	360

4.2.3. Effect of the dimension

To investigate the effect of dimension on searching quality of CPSO, Ackley function [16] is chosen for test.

$$f_A(x) = -20\exp\left(-0.2\sqrt{\frac{1}{n}\sum_{i=1}^{n}x_i^2}\right) - \exp\left(\frac{1}{n}\sum_{i=1}^{n}\cos(2\pi x_i)\right) + 20 + e, \quad -32 \leqslant x_i \leqslant 32. \tag{11}$$

Ackley function has thousands of minima in the region and is very difficult to be optimized, while the global minimum is 0 at $x = 0$.

Independently running CPSO, PSO and GA (total number of function evaluation is set as 4×10^5) each with 10 times for Ackley function with different dimension, Fig. 10 illustrates the varying curves of the mean objective value resulted by each approaches with respect to dimension. It can be seen that the effect of dimension on searching performance is weaker for CPSO than for other two algorithms. Even when the dimension is very high, the searching quality of CPSO is also acceptable.

In addition, Fig. 11 shows that the orbital points of Gaussian distribution used in GA mainly distribute near zero, while Fig. 12 shows that the orbital points of Logistic map distribute near the edges. To some extent, it can be used for explaining why CPSO has advantage to escape from local optimum.

5. Conclusion

To our knowledge, this is the first report of hybridizing particle swarm optimization and chaos to propose an effective and efficient optimization algorithm for numerical functions. The proposed approach not only performs exploration by using the population-based evolutionary searching ability of PSO with AIFW, but also performs exploitation by

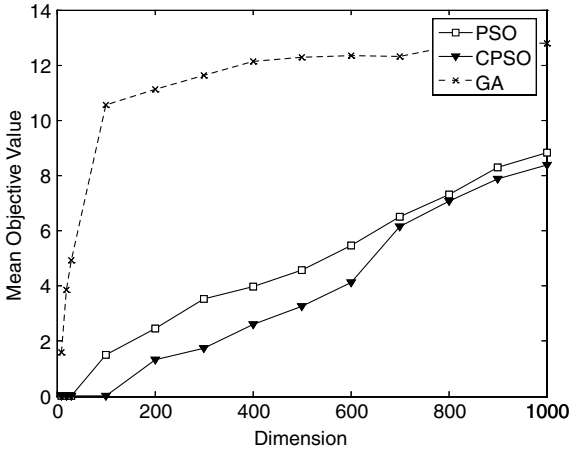

Fig. 10. Effect of dimension in terms of mean objective value.

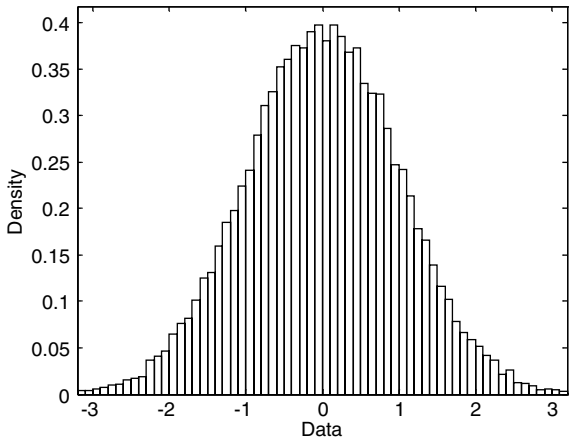

Fig. 11. Orbital points of Gaussian distribution.

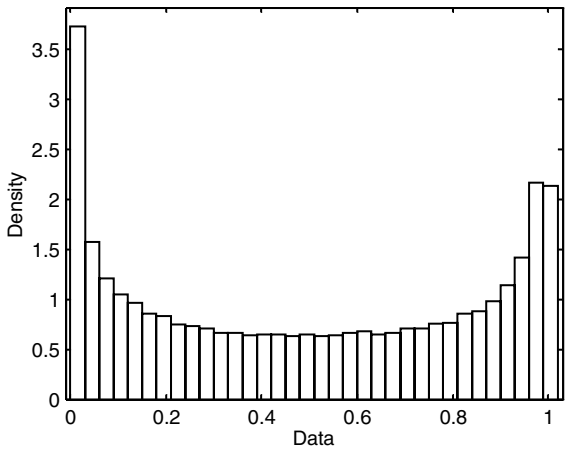

Fig. 12. Orbital points of logistic Map.

using the chaotic local searching behavior. The proposed CPSO is superior in term of searching quality, efficiency and robustness on initial conditions. Besides, to achieving the same searching quality CPSO is much faster than other meta-heuristics, such as PRS, MS, SA, CSA, TS, PSO and GA. The future work is to theoretically investigate the effect of chaos incorporating into PSO and apply the CPSO for some real engineering optimization problems.

Acknowledgement

This paper is partially supported by National Natural Science Foundation of China (Grant Nos. 60204008 and 60374060) and National 973 Program (Grant No. 2002CB312200).

Appendix A. Six famous benchmark optimization problems used in this paper

(1) GP—Goldstein–Price, ($n = 2$):

$$f_G(x) = \left[1 + (x_1 + x_2 + 1)^2\left(19 - 14x_1 + 3x_1^2 - 14x_2 + 6x_1x_2 + 3x_2^2\right)\right] \times \left[30 + (2x_1 - 3x_2)^2\left(18 - 32x_1 + 12x_1^2 + 48x_2 - 36x_1x_2 + 27x_2^2\right)\right], \quad -2 \leqslant x_i \leqslant 2, \quad i = 1, 2.$$

The global minimum is equal to 3 and the minimum point is $(0,-1)$. There are four local minima in the minimization region.

(2) BR—Branin ($n = 2$):

$$f_B(x) = \left(x_2 - \frac{5.1}{4\pi^2}x_1^2 + \frac{5}{\pi}x_1 - 6\right)^2 + 10\left(1 - \frac{1}{8\pi}\right)\cos x_1 + 10, \quad -5 \leqslant x_1 \leqslant 10, \quad 0 \leqslant x_2 \leqslant 15.$$

The global minimum is approximately 0.398 and it is reached at the three points $(-3.142, 12.275)$, $(3.142, 2.275)$ and $(9.425, 2.425)$.

(3 and 4) Hn–Hartman ($n = 3, 6$):

$$f_H(x) = -\sum_{i=1}^{4} c_i \exp\left[-\sum_{j=1}^{n} a_{ij}\left(x_j - p_{ij}\right)^2\right], \quad 0 \leqslant x_i \leqslant 1, \quad i = 1, \ldots, n.$$

For $n = 3$ the global minimum is equal to -3.86 and it is reached at the point $(0.114, 0.556, 0.852)$. For $n = 6$ the minimum is -3.32 at the point $(0.201, 0.150, 0.477, 0.275, 0.311, 0.657)$.

(5) RA—Rastrigin ($n = 2$):

$$f_R(x) = x_1^2 + x_2^2 - \cos 18x_1 - \cos 18x_2, \quad -1 \leqslant x_i \leqslant 1, \quad i = 1, 2.$$

The global minimum is equal to -2 and the minimum point is $(0, 0)$. There are about 50 local minima arranged in a lattice configuration.

(6) SH—Shuber ($n = 2$):

$$f_S(x) = \left\{\sum_{i=1}^{5} i\cos((i+1)x_1 + i)\right\}\left\{\sum_{i=1}^{5} i\cos((i+1)x_2 + i)\right\} \quad -10 \leqslant x_1, x_2 \leqslant 10.$$

The function has 760 local minima, 18 of which are global with $f_S = -186.7309$.

References

[1] Ott E, Grebogi C, Yorke JA. Controlling chaos. Phys Rev Lett 1990;64:1196–9.
[2] Kapitaniak T. Continuous control and synchronization in chaotic systems. Chaos, Solitons & Fractals 1995;6:237–44.
[3] Pecora L, Carroll T. Synchronization in chaotic systems. Phys Rev Lett 1990;64:821–4.
[4] Aihara K, Takabe T, Toyoda M. Chaotic neural networks. Phys Lett A 1990;144:333–40.
[5] Li B, Jiang WS. Optimizing complex functions by chaos search. Cybernet Syst 1998;29:409–19.
[6] Lu Z, Shieh LS, Chen GR. On robust control of uncertain chaotic systems: a sliding-mode synthesis via chaotic optimization. Chaos, Solitons & Fractals 2003;18:819–27.
[7] Wang L, Zheng DZ, Lin QS. Survey on chaotic optimization methods. Comput Technol Automat 2001;20:1–5.
[8] Kennedy J, Eberhart RC. Particle swarm optimization. In: Proc IEEE Int Conf on Neural Networks, 1995, WA Australia. p. 1942–8.
[9] Kennedy J, Eberhart RC, Shi Y. Swarm intelligence. San Francisco: Morgan Kaufmann Publishers; 2001.
[10] Goldberg DE. Genetic algorithms in search, optimization, and machine learning. MA: Addison Wesley; 1989.
[11] Eberhart RC, Shi Y. Particle swarm optimization: developments, applications and resources. In: Proc. Congress on evolutionary computation, 2001, Seoul, Korea. p. 81–6.
[12] Angeline PJ. Evolutionary optimization versus particle swarm optimization: philosophy and performance differences. In: Evolutionary programming VII. Springer; 1998. p. 601–10.
[13] Shi Y, Eberhart RC. A modified particle swarm optimizer. In: Proc IEEE Int Conf on Evolutionary Computation. USA: Anchorage; 1998. p. 69–73.
[14] Shi Y, Eberhart RC. Empirical study of particle swarm optimization. In: Proc IEEE Int Congr Evolutionary Computation, 1999, Washington, DC. p. 1945–50.
[15] May R. Simple mathematical models with very complicated dynamics. Nature 1976;261:459–67.
[16] Wang L. Intelligent optimization algorithms with applications. Beijing: Tsinghua University & Springer Press; 2001.
[17] Anderssen RS, Jennings LS, Ryan DM. Optimization. St. Lucia, Australia: University of Queensland Press; 1972.
[18] Dekkers A, Aarts E. Global optimizations and simulated annealing. Math Program 1991;50:367–93.
[19] Cvijović D, Klinowski J. Taboo search: an approach to the multiple-minima problem. Science 1995;267:664–6.
[20] Ji MJ, Tang HW. Application of chaos in simulated annealing. Chaos, Solitons & Fractals 2004;21:933–41.