Parameter Evolution for a Particle Swarm Optimization Algorithm

Aimin Zhou[1], Guixu Zhang[1], and Andreas Konstantinidis[2]

[1] East China Normal University, Shanghai, China
{amzhou,gxzhang}@cs.ecnu.edu.cn
[2] Frederick University of Cyprus, Cyprus
com.ca@fit.ac.cy

Abstract. Setting appropriate parameters of an evolutionary algorithm (EA) is challenging in real world applications. On one hand, the characteristics of a real world problem are usually unknown. On the other hand, in different running stages of an EA, the best parameters may be different. Thus adaptively tuning algorithm parameters online is preferred. In this paper, we propose to use an estimation of distribution algorithm (EDA) to do this for a particle swarm optimization (PSO) algorithm. The major characteristic of our approach is that there are two evolving processes simultaneously: one for tackling the original problem, and the other for optimizing PSO parameters. For the former evolving process, a set of particles are maintained; while for the later, a probability distribution model of the PSO parameters is maintained throughout the run. In the reproduction procedure, the PSO parameters are firstly sampled from the model, and then new particles are generated by the PSO operator. The feedback from the newly generated particles is used to evaluate the PSO parameters and thus to update the probability model. The new approach is applied to a set of test instances and the preliminary results are promising.

1 Introduction

The parameter tuning plays a key role in applying *evolutionary algorithms* (EAs) to real world applications [1]. The success of an EA depends not only on the algorithm itself but also on the problem to be solved. In algorithm design, we could tune the parameters either by repeated running or by analyzing the properties of benchmark probelems. However in applications, the characteristics of the problems maybe incomplete or the problems may not have closed forms, and the repeated experiments may be expensive. Thus the strategies for tuning parameters in algorithm design are no longer suitable in such cases. Furthermore, to achieve the best performance, the parameters of an algorithm may not be fixed throughout the run. For example, at the beginning stages, to get better diversity, the mutation probability might be high; while at the later stages, to achieve better convergence, the mutation probability needs to be low. To overcome the shortcomings of offline parameter tuning strategies, many research turn to set algorithm parameters adaptively online [2].

Z. Cai et al. (Eds.): ISICA 2010, LNCS 6382, pp. 33–43, 2010.

Most of widely adaptively parameter tuning methods could be classified into the following categories.

- **Randomly selecting parameters:** With this strategy, the users give a set of candidate parameter settings by guessing or using prior knowledge. The algorithm then randomly select a parameter setting from the given set [3]. The initial parameter set is importance to the algorithm success.
- **Adaptively tuning by feedback:** By this strategy, the parameters are adaptively adjusted by heuristic rules with take feedbacks from previous parameter changes [4]. How to define the feedback is a key issue.
- **Encoding parameters into chromosomes:** The parameters are incorporated into the chromosomes and evolve with decision variables [5]. There is not much additional work to implement this strategy. However, encoding the parameters increases the complexity of the EA search space and may slow down the search.
- **Parameter evolving:** Cooperating with the main algorithm, another EA works on the parameters and its optimal solutions, i.e. the best parameters, are used in the main algorithm [6].

In this paper, we follow the idea of parameter evolving strategy. An estimation of distribution algorithm (EDA) [7] [8] is applied to tune the parameters of a particle swarm optimization (PSO) [9] [10]. In our approach, an EDA and a PSO evolve simultaneously. The EDA works on the PSO parameter space, while the PSO works on the original problem search space. In the running process, the EDA maintains a multivariate histogram probabilistic model of PSO parameters; and the PSO maintain a set of candidate solutions (particles). In each generation, the PSO parameters are firstly sampled from the EDA model; secondly, new particles are generated by the PSO operator; thirdly, the probability model is updated according to the performances of the sampled parameters.

The rest of the paper is organized as follows. The next section introduces the background of our work, including the problems, the PSO model which are used in the paper, and a brief introduction of EDA. Section 3 presents the details of the proposed method. Section 4 describes and analyzes the experimental results. The final section concludes the paper and outlines future research work.

2 Background

2.1 Optimization Problem

In this paper, we consider the following global optimization problems.

$$\begin{aligned} &\min\ f(x) \\ &\text{s.t}\ x \in \Omega \end{aligned} \tag{1}$$

where $\Omega \subset R^n$ is the decision space and $x = (x_1, \cdots, x_n)^T$ is the decision variable vector. $f : \Omega \to R$ is a continuous objective function and R is the objective space.

2.2 Particle Swarm Optimization

Among various techniques for global optimization, *particle swarm optimization* (PSO) is a promising one. PSO is a population based stochastic optimization technique developed by Eberhart and Kennedy in 1995 [9] [10], inspired by social behavior of bird flocking or fish schooling. Mathematically, the ith particle at generation t, $x_i(t)$, is updated as,

$$\begin{cases} v_i(t+1) = wv_i(t) \\ \qquad\qquad +c_1 r_{1,i}(x^G(t) - x_i(t)) \\ \qquad\qquad +c_2 r_{2,i}(x_i^L(t) - x_i(t)) \\ x_i(t+1) = x_i(t) + v_i(t+1) \end{cases}, \tag{2}$$

where v denotes the volocity, $x^G(t)$ is the global best particle, $x_i^L(t)$ is the local best particle found so far, w is the inertia weight, c_1, c_2 are the acceleration constants, $r_{1,i}, r_{2,i}$ are two dialog matrix of which the dialog elements are uniformly randomly sampled from $[0, 1]$.

Let $\alpha_i(t) = (w_i(t), c_{1,i}(t), c_{2,i}(t))^T$ be the parameter vector for generating the $i$tsh particle at generation t, the above generation procedure could be denoted as

$$(v_i(t+1), x_i(t+1)) := \text{generate}(v_i(t), x_i(t), x_i^L(t), x^G(t), \alpha_i(t)). \tag{3}$$

After the generation process, the global best particle the the local best particle are then updated.

2.3 Estimation of Distribution Algorithm

Estimation of distribution algorithms (EDA) are a new evolutionary computation paradigm [7] [8]. A major difference between EDAs and traditional EAs is in the offspring reproduction procedure. There is no crossover or mutation in EDAs. Instead, they build a probability model of promising solutions by extracting the global population distribution information and sample new solutions from the model thus built.

3 Evolving PSO Parameters by EDA

3.1 Algorithm Framework

To adaptively set the PSO parameters, we use an EDA to evolve these parameters with the main PSO procedure. In each generation t, our approach, named *parameter evolution guided particle swarm optimization* (PEPSO), maintains

- a set of particles: $\{x_i(t), i = 1, \cdots, N\}$,
- a set of velocity vectors: $\{v_i(t), i = 1, \cdots, N\}$,
- a set of local best particles: $\{x_i^L(t), i = 1, \cdots, N\}$,
- a global best particle: $x^G(t)$, and
- a probability distribution model of parameters: $P(\alpha(t))$.

where N is the population size, and $\alpha(t) = (w(t), c_1(t), c_2(t))^T$ denotes the PSO parameter vector.

The main framework of PEPSO is as follows.

Step 0 Initialization: Set $t := 0$. Uniformly randomly generate a set of particle $\{x_i(t), i = 1, \cdots, N\}$, a set of velocity vectors $\{v_i(t), i = 1, \cdots, N\}$ in the search space Ω. Evaluate these particles by (1) and find the global best particle $x^G(t)$. Let $\{x_i^L(t) = x_i(t), i = 1, \cdots, N\}$. Initialize the parameter distribution model $P(\alpha(t))$.

Step 1 Stopping Condition: If stopping condition is met, stop and return $x^G(t)$.

Step 2 Reproduction: For each particle $i = 1, \cdots, N$, sample a parameter vector $\alpha_i(t)$ from $P(\alpha(t))$, generate a new particle $x_i(t+1)$ by (3), and evaluate this particle.

Step 3 Particle Updating: Update the global best particle $x^G(t+1)$ and local best particle $x_i^L(t+1), i = 1, \cdots, N$.

Step 4 Model Updating: Update the probability model $P(\alpha(t+1))$ by the improvements of the particles.

Step 5 Set $t := t + 1$ and go to **Step 1**.

In the following, we discuss the probability model definition and implementation.

3.2 Probability Distribution Model of Parameters

Since the algorithm parameters may correlate with each other, we use a multivariate histogram probabilistic model to model the distribution of continuous parameters. Let the boundaries of the parameter vector $\alpha = (\alpha_1, \cdots, \alpha_m)^T$ be $[\alpha_1^L, \alpha_1^U] \times \cdots \times [\alpha_m^L, \alpha_m^U]$, and each dimension be divided into D subsets, the parameter search space is thus divided into D^m bins.

The multivariate histogram probabilistic model is defined as

$$P(\alpha(t)) = (p_1(t), \cdots, p_{D^m}(t))^T \tag{4}$$

where $0 \leq p_i(t) \leq 1$ denotes the probability of a parameter vector from the ith bin, and $\sum_{i=1}^{D^m} p_i(t) = 1$.

Fig.1 shows a multivariate histogram probability model in the case of 2-dimensional parameter vector. The search space is divided into $5^2 = 25$ bins, and the hight of each bar denotes the probability of a parameter vector is from that bin.

3.3 Model Sampling and Updating

Let $q_i(t) > 0$ denotes a frequency of the ith bin to be used in generation t, then the probability is approximated as $p_i(t) = \frac{q_i(t)}{\sum_j q_j(t)}$. Instead of maintaining a probability vector, we maintain a frequency vector $q_i(t), i = 1, \cdots, D^m$ in the running process.

In the initialization step, the frequency vector is set to $q_i(0) = 5.0, i = 1, \cdots, D^m$.

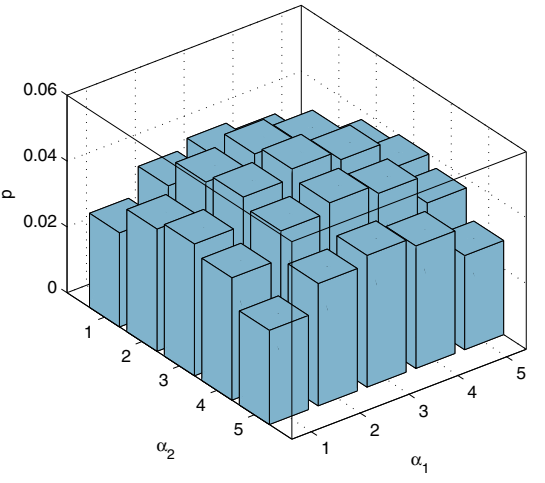

Fig. 1. Illustration of multivariate histogram probabilistic model in the case of 2-dimensional parameter vector

In generating the ith particle, a parameter vector $\alpha_i(t)$ is sampled as follows. Firstly, we randomly select a bin index $I_i(t)$ according to the probability distribution model $P(\alpha(t))$. We then uniformly randomly sample a vector $\alpha_i(t)$ from the $I_i(t)$th bin.

Define the improvement of the ith particle as

$$\Delta_i(t) = \begin{cases} f(x_i(t)) - f(x_i(t+1)) & \text{if } f(x_i(t)) > f(x_i(t+1)) \\ 0 & \text{otherwise} \end{cases}.$$

Let

$$a_{i,j}(t) = \begin{cases} 1 \text{ if } I_i(t) = j \\ 0 \text{ otherwise} \end{cases}.$$

The contribution of parameters from the jth bin is defined as

$$C_j(t) = \sum_{i=1}^{N} a_{i,j}(t) \frac{\Delta_i(t)}{\max_k \Delta_k(t)}.$$

We then update the frequency vector as follows,

$$q_j(t) = \begin{cases} 1.0 & \text{if } (1-\beta)q_j(t) + \frac{C_j(t)}{\max_j C_j(t)} < 1 \\ 10.0 & \text{if } (1-\beta)q_j(t) + \frac{C_j(t)}{\max_j C_j(t)} > 10.0 \\ (1-\beta)q_j(t) + \frac{C_j(t)}{\max_j C_j(t)} & \text{otherwise} \end{cases}$$

where β is an algorithm parameter and it is set to be 0.75 in the experiments. The frequency is fixed in the range of $[1.0, 10.0]$.

4 Experimental Results and Analysis

4.1 Test Instances

We use 13 test instances [11] in the experiments. They are Sphere function, Schwefel 2.22 function, Schwefel 1.2 function, Schwefel 2.21 function, Rosenbrock fuction, Step function, Noisy Quartic function, Schwefel 2.26 function, Rastrigin function, Ackley function, Griewank function, and two Penalized functions. Each of these functions has a global minimum value of 0. The details of these functions are listed in Tab 1.

Table 1. Test instances used in our experiments

Ω	$f(\cdot)$
$[-100, 100]^n$	$f_1(x) = \sum_{i=1}^{n} x_i^2$
$[-10, 10]^n$	$f_2(x) = \sum_{i=1}^{n} \lvert x_i \rvert + \Pi_{i=1}^{n} \lvert x_i \rvert$
$[-100, 100]^n$	$f_3(x) = \sum_{i=1}^{n} (\sum_{j=1}^{i} x_j)^2$
$[-100, 100]^n$	$f_4(x) = \max_i \{\lvert x_i \rvert\}$
$[-30, 30]^n$	$f_5(x) = \sum_{i=1}^{n-1} [100(x_{i+1} - x_i^2)^2 + (x_i - 1)^2]$
$[-100, 100]^n$	$f_6(x) = \sum_{i=1}^{n} \lfloor x_i + 0.5 \rfloor^2$
$[-1.28, 1.28]^n$	$f_7(x) = \sum_{i=1}^{n} i x_i^4 + rand[0, 1)$
$[-500, 500]^n$	$f_8(x) = \sum_{i=1}^{n} -i x_i \sin \sqrt{\lvert x_i \rvert} + 418.98288727243369n$
$[-5.12, 5.12]^n$	$f_9(x) = \sum_{i=1}^{n} [x_i^2 - 10\cos(2\pi x_i) + 10]$
$[-32, 32]^n$	$f_{10}(x) = -20\exp(-0.2\sqrt{\frac{1}{n}\sum_{i=1}^{n} x_i^2}) - \exp(\frac{1}{n}\sum_{i=1}^{n}\cos(2\pi x_i)) + 20 + e$
$[-600, 600]^n$	$f_{11}(x) = \frac{1}{4000}\sum_{i=1}^{n} x_i^2 - \Pi_{i=1}^{n}\cos(\frac{x_i}{\sqrt{i}}) + 1$
$[-50, 50]^n$	$f_{12}(x) = \frac{\pi}{n}\{10\sin^2(\pi y_1) + \sum_{i=1}^{n-1}(y_i - 1)^2[1 + 10\sin^2(\pi y_{i+1})] + (y_n - 1)^2\}$
	$+ \sum_{i=1}^{n} u(x_i, 10, 100, 4)$
	$y_i = 1 + \frac{1}{4}(x_i + 1)$
	$u(x_i, a, k, m) = \begin{cases} k(x_i - a)^m & x_i > a \\ 0 & -a \le x_i \le a \\ k(-x_i - a)^m & x_i < -a \end{cases}$
$[-50, 50]^n$	$f_{13}(x) = 0.1\{10\sin^2(3\pi x_1) + \sum_{i=1}^{n-1}(x_i - 1)^2[1 + 10\sin^2(3\pi x_{i+1})]$
	$+(x_n - 1)^2[1 + \sin^2(2\pi x_n)]\} + \sum_{i=1}^{n} u(x_i, 5, 100, 4)$
	the $u(x_i, a, k, m)$ is the same as in f_{12}

4.2 Experimental Parameter Setting

In the experiments, we compare the proposed PEPSO with a general PSO. The parameters for PSO are $w = 0.5$, and $c_1 = c_2 = 2.05$ as used in most PSO algorithms. The parameter for PEPSO are as follows: the search range of w is

Table 2. The statistical results ($mean \pm std.$) of PSO and PEPSO on the 13 test instances over 50 runs after 1000, 3000 and 5000 generations

		1000	3000	5000
f_1	PSO	1.548e-12±2.693e-12	5.530e-45±1.873e-44	1.471e-77±6.468e-77
	PEPSO	**9.587e-17**±1.623e-16	**1.756e-56**±3.405e-56	**1.398e-95**±7.907e-95
f_2	PSO	1.086e-08±3.602e-08	3.433e-30±9.232e-30	1.824e-51±6.527e-51
	PEPSO	**1.429e-11**±4.260e-11	**7.476e-38**±1.398e-37	**1.263e-63**±5.074e-63
f_3	PSO	6.208e+02±2.916e+02	4.642e+00±3.908e+00	8.114e-02±9.738e-02
	PEPSO	**2.118e+02**±1.067e+02	**4.947e-01**±4.704e-01	**2.550e-03**±2.881e-03
f_4	PSO	4.073e+00±1.258e+00	1.989e-01±1.090e-01	1.312e-02±1.250e-02
	PEPSO	**1.344e+00**±4.469e-01	**8.813e-03**±6.994e-03	**6.424e-05**±7.231e-05
f_5	PSO	1.194e+02±4.243e+02	1.079e+02±4.249e+02	1.015e+02±4.252e+02
	PEPSO	**5.417e+01**±3.916e+01	**3.941e+01**±3.026e+01	**3.477e+01**±2.912e+01
f_6	PSO	**0.000e+00**±0.000e+00	**0.000e+00**±0.000e+00	**0.000e+00**±0.000e+00
	PEPSO	**0.000e+00**±0.000e+00	**0.000e+00**±0.000e+00	**0.000e+00**±0.000e+00
f_7	PSO	2.043e-02±6.191e-03	6.852e-03±2.398e-03	4.202e-03±1.561e-03
	PEPSO	**1.384e-02**±4.793e-03	**5.218e-03**±1.885e-03	**3.113e-03**±1.018e-03
f_8	PSO	**1.416e+03**±3.486e+02	**1.412e+03**±3.468e+02	**1.412e+03**±3.468e+02
	PEPSO	1.572e+03±3.916e+02	1.5699e+03±3.914e+02	1.570e+03±3.914e+02
f_9	PSO	4.208e+01±1.854e+01	3.475e+01±1.971e+01	3.409e+01±1.931e+01
	PEPSO	**3.500e+01**±2.024e+01	**2.840e+01**±1.622e+01	**2.728e+01**±1.645e+01
f_{10}	PSO	1.803e-07±1.589e-07	8.420e-15±1.705e-15	7.923e-15±5.024e-16
	PEPSO	**5.615e-09**±1.371e-08	**7.852e-15**±7.033e-16	**7.709e-15**±9.736e-16
f_{11}	PSO	**8.368e-03**±1.073e-02	**7.928e-03**±9.858e-03	**7.928e-03**±9.858e-03
	PEPSO	1.429e-02±2.018e-02	1.365e-02±2.010e-02	1.218e-02±1.770e-02
f_{12}	PSO	**2.073e-03**±1.466e-02	**1.571e-32**±5.529e-48	**1.571e-32**±5.529e-48
	PEPSO	4.146e-03±2.932e-02	4.146e-03±2.932e-02	4.146e-03±2.932e-02
f_{13}	PSO	3.302e-09±1.837e-08	**1.350e-32**±1.106e-47	**1.350e-32**±1.106e-47
	PEPSO	**2.101e-13**±8.957e-13	**1.350e-32**±1.106e-47	**1.350e-32**±1.106e-47

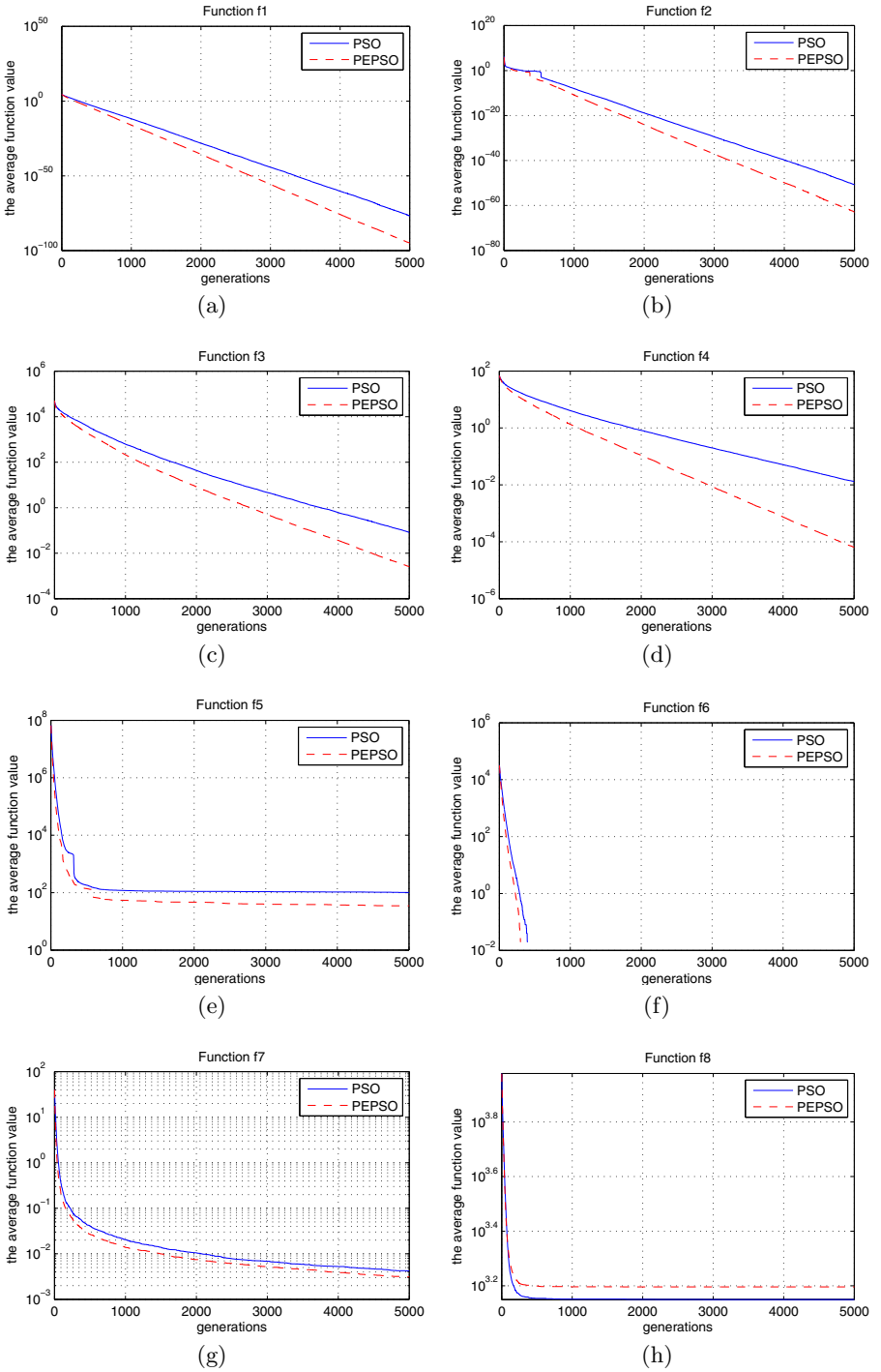

Fig. 2. The mean fitness values versus generations over 50 runs on f_1-f_8

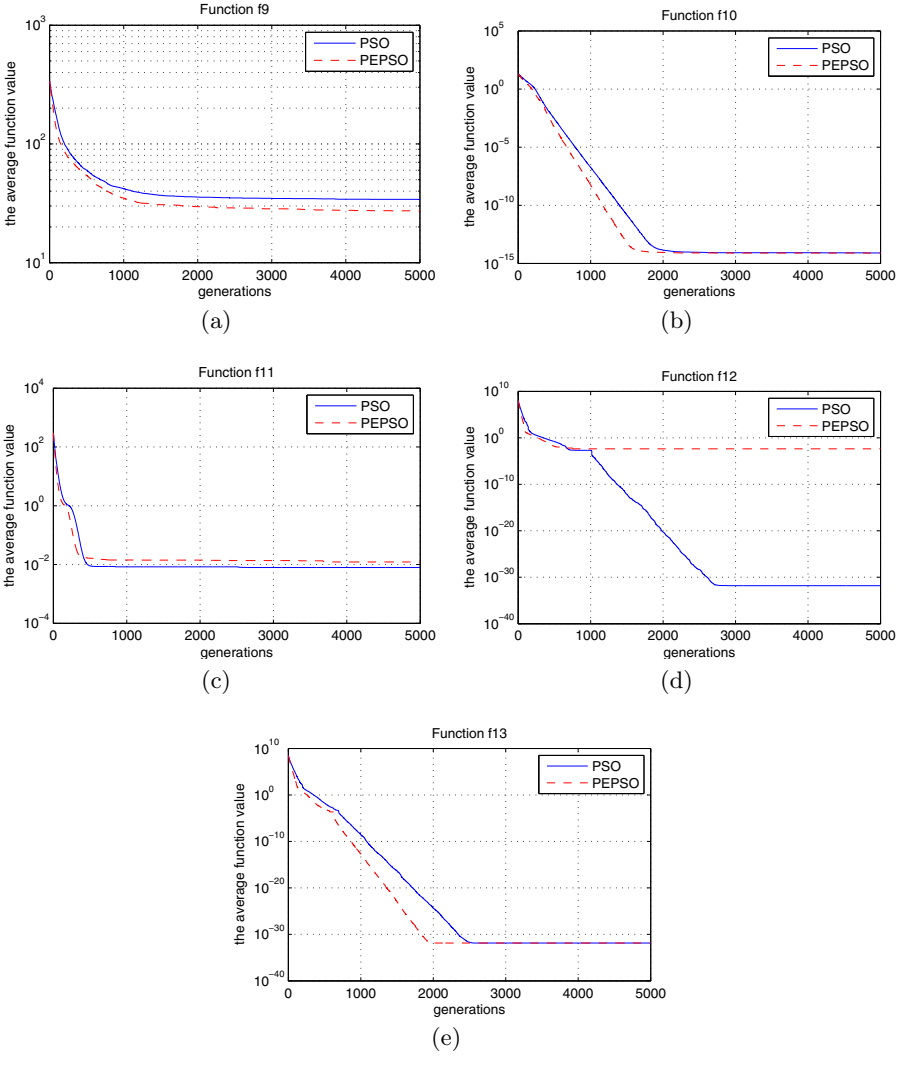

Fig. 3. The mean fitness values versus generations over 50 runs on f_9-f_{13}

$[0.25, 0.75]$ and the search range of c_1 and c_2 is $[1.5, 2.5]$; each range is divided into 20 subranges; to reduce the computational cost, we set $c_1 = c_2$ in the experiments and thus the parameter space is divided into $20^2 = 400$ grids.

The population size for both PSO and PEPSO is 200 and the algorithms will stop after 5000 generations. The decision vector dimensions of all the test problems are set to be $n = 30$. Each algorithm is executed independently for each instance for 50 times.

4.3 Results and Analysis

Table 2 shows the means and standard devisions of PSO and PEPSO on the 13 problems over 50 runs after 1000, 2000, and 3000 generations. Figs 2 and 3 illustrate the mean fitness values versus generations over 50 runs on all the test problems.

From the results, we can see that

- For f_1 - f_5, f_7, and f_9, PEPSO outperforms PSO not only on the converge speed but also on the quality of the obtained solutions.
- For f_6, f_{10}, and f_{13}, both PSO and PEPSO obtain similar results. However, PEPSO converges faster than PSO.
- For f_8, f_{11}, and f_{12}, PSO shows better performance than PEPSO.

The only difference between PSO and PEPSO is that PEPSO adaptively tunes its parameters in (3). The results indicate that for most of the test instances, adaptively online tuning strategy is better than setting the parameters offline. However, for some problems, PEPSO works worse than PSO. The reason might be that the probability model in PEPSO converges to local optimal solutions, i.e., the probabilities of some bad parameter vectors are much higher than those of good parameter vectors.

5 Conclusion and Future Work

In this paper, we proposed a PSO algorithm with an adaptively parameter tuning strategy by an EDA. The proposed algorithm, PEPSO, was compared with a general PSO algorithm on 13 widely used test instances. The preliminary results indicated that for most of the test problems, PEPSO performed better than PSO, either in convergence speed or in both convergence speed and solution quality. Although the adaptive strategy (EDA) still needs some parameters, it leads to the improvements of solution quality and convergence speed.

The research on adaptively tuning EA parameters is still in its very infancy and our work presented in this paper is also rather preliminary. Much work remains to be done in the future, for example, designing more efficient probability model update strategies and comparing PEPSO with other PSO algorithms [12] [13] [14].

Acknowledgement

This work is supported by National Science Foundation of China (No. 60773119) and Program for New Century Excellent Talents in University (No. NCET-08-0193).

References

1. De Jong, K.: Parameter setting in eas: a 30 year perspective. In: Parameter Setting in Evolutionary Algorithms, pp. 1–18 (2007)
2. Eiben, A.E., Michalewicz, Z., Schoenauer, M., Smith, J.E.: Parameter control in evolutionary algorithms. In: Parameter Setting in Evolutionary Algorithms, pp. 19–46 (2007)
3. Zhou, A., Zhang, Q., Jin, Y.: Approximating the set of pareto optimal solutions in both the decision and objective spaces by an estimation of distribution algorithm. IEEE Transactions on Evolutionary Computation 13(5), 1167–1189 (2009)
4. Yen, G.G., Lu, H.: Dynamic multiobjective evolutionary algorithm: Adaptive cell-based rank and density estimation. IEEE Transactions on Evolutionary Computation 7(3), 253–274 (2003)
5. Brest, J., Greiner, S., Boskovic, B., Mernik, M., Zumer, V.: Self-adapting control parameters in differential evolution: A comparative study on numerical benchmark problems. IEEE Transactions on Evolutionary Computation 10(6), 689–699 (2006)
6. Smit, S., Eiben, A.: Comparing parameter tuning methods for evolutionary algorithms. In: Proceedings of the 2009 IEEE Congress on Evolutionary Computation, pp. 399–406. IEEE, Los Alamitos (2009)
7. Mühlenbein, H., Paaß, G.: From recombination of genes to the estimation of distributions I. binary parameters. In: Ebeling, W., Rechenberg, I., Voigt, H.-M., Schwefel, H.-P. (eds.) PPSN 1996. LNCS, vol. 1141, pp. 178–187. Springer, Heidelberg (1996)
8. Larrañaga, P., Lozano, J.A.: Estimation of Distribution Algorithms: A New Tool for Evolutionary Computation. Kluwer Academic Publishers, Dordrecht (2002)
9. Eberhart, R.C., Kennedy, J.: A new optimizer using particle swarm theory. In: 6th International Symposium on Micromachine and Human Science, Nagoya, Japan, pp. 39–43 (1995)
10. Kennedy, J., Eberhart, R.C., Shi, Y.: Swarm Intelligence. Morgan Kaufmann, San Francisco (2001)
11. Yao, X., Liu, Y., Liang, K.-H., Lin, G.: Fast evolutionary algorithms. In: Advances in evolutionary computing: theory and applications, pp. 45–94. Springer, Inc., New York (2003)
12. Liang, J.J., Qin, A.K., Suganthan, P.N., Baskar, S.: Comprehensive learning particle swarm optimizer for global optimization of multimodal functions. IEEE Transactions on Evolutionary Computation 10(3), 281–295 (2006)
13. de Oca, M.A.M., Stützle, T., Birattari, M., Dorigo, M.: Frankenstein's pso: A composite particle swarm frankenstein's pso: A composite particle swarm optimization algorithm. IEEE Transactions on Evolutionary Computation 13(5), 1120–1132 (2009)
14. Zhan, Z.-H., Zhang, J., Li, Y., Chung, H.-H.: Adaptive particles warm optimization. IEEE Transactions on Systems, Man and Cybernetics, Part B (2009)