Parameter Evolution for a Particle Swarm Optimization Algorithm

Aimin Zhou[1], Qingfu Zhang[2], and Andreas Konstantinidis[3]

[1] East China Normal University, Shanghai, China. amzhou@cs.ecnu.edu.cn
[2] University of Essex, Colchester, UK. qzhang@essex.co.uk
[3] 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 states 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 tune the parameters of a particle swarm optimization (PSO) algorithm. A probability distribution model of the parameters is maintained throughout the run. To generate a new particle, the parameters will be sampled from the model. The model is then updated by the performance improvements of all the particles. The new approach is applied to a set of test instances and the results show that it could improve the performance of a PSO algorithm.

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 analyzing the properties of benchmark probelems. However, these strategies may not be applicable in real world applications. Furthermore, to achieve the best performance, the parameters of an algorithm may be different in different running states. To overcome the shortcomings of offline parameter tuning strategies, many research turn to set algorithm parameters adaptively online [2].

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

- **Randomly selecting parameters:** The idea is not to fix the parameters but to select the parameters in a given set [3].
- **Adaptively tuning by feedback:** By this strategy, the parameters will be adaptively adjusted by heuristic rules with take feedbacks from previous parameter changes [4].
- **Encoding parameters into chromosomes:** The parameters are incorporated into the chromosomes and evolve with decision variables [5].

– **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 each generation, the EDA maintains a multivariate histogram probabilistic model of PSO parameters. To generate a new particle, the parameters are sampled from the probability model thus built. After generating all particles, 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 problems and PSO model which are used in the paper. 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 Particle Swarm Intelligence

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.

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 itsh particle at generation t, the above generation procedure could be denoted as

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

3 Evolving PSO Parameters

3.1 Algorithm Framework

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. 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 based 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 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 subranges, the parameter search space is thus divided into D^m grids.

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 grid, and $\sum_{i=1}^{D^m} p_i(t) = 1$.

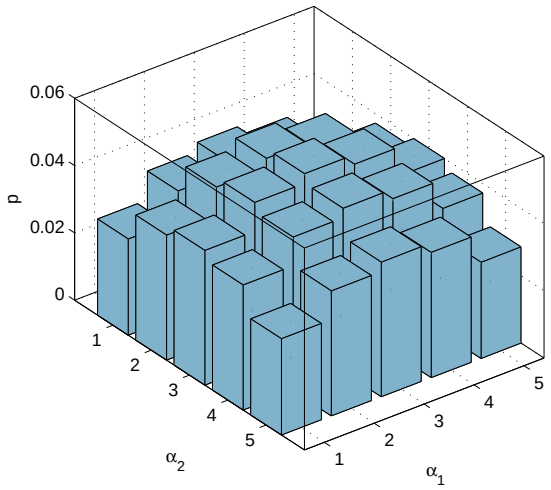

Fig. 1. Illustration of multivariate histogram probabilistic model in the case of 2-dimensional parameter vector.

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

3.3 Model Sampling and Updating

Let $q_i(t) > 0$ denotes a frequency of the ith grid to be used in generation t, then the probability $p_i(t) = \frac{q_i(t)}{\sum_j q_j(t)}$. 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 $q_i(0) = 5.0, i = 1, \cdots, D^m$.

In generating the ith particle, a parameter vector $\alpha_i(t)$ is sampled as follows. Firstly, we randomly select a grid 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 grid.

Define the improvement of the ith particle as

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

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 grid 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 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 could be found in [11]. For simplicity, we rename these functions as f_1 to f_{13} respectively.

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 $[0.25, 0.75]$ 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.

Table 1. The statistical results (*mean* ± *std.*) of PSO and PEPSO on the 13 test instances over 50 runs after 1000, 3000 and 5000 generations.

		1000	3000	5000
f1	PSO	1.5484e-12±2.6930e-12	5.5303e-45±1.8732e-44	1.4706e-77±6.4683e-77
	PEPSO	**9.5868e-17**±1.6225e-16	**1.7562e-56**±3.4047e-56	**1.3978e-95**±7.9066e-95
f2	PSO	1.0859e-08±3.6020e-08	3.4331e-30±9.2315e-30	1.8235e-51±6.5272e-51
	PEPSO	**1.4287e-11**±4.2595e-11	**7.4762e-38**±1.3983e-37	**1.2631e-63**±5.0738e-63
f3	PSO	6.2080e+02±2.9164e+02	4.6415e+00±3.9078e+00	8.1142e-02±9.7379e-02
	PEPSO	**2.1178e+02**±1.0667e+02	**4.9468e-01**±4.7038e-01	**2.5498e-03**±2.8807e-03
f4	PSO	4.0730e+00±1.2577e+00	1.9888e-01±1.0895e-01	1.3121e-02±1.2501e-02
	PEPSO	**1.3437e+00**±4.4686e-01	**8.8128e-03**±6.9937e-03	**6.4241e-05**±7.2307e-05
f5	PSO	1.1944e+02±4.2427e+02	1.0790e+02±4.2487e+02	1.0148e+02±4.2516e+02
	PEPSO	**5.4165e+01**±3.9157e+01	**3.9412e+01**±3.0257e+01	**3.4770e+01**±2.9120e+01
f6	PSO	**0.0000e+00**±0.0000e+00	**0.0000e+00**±0.0000e+00	**0.0000e+00**±0.0000e+00
	PEPSO	**0.0000e+00**±0.0000e+00	**0.0000e+00**±0.0000e+00	**0.0000e+00**±0.0000e+00
f7	PSO	2.0433e-02±6.1907e-03	6.8519e-03±2.3983e-03	4.2024e-03±1.5614e-03
	PEPSO	**1.3839e-02**±4.7925e-03	**5.2176e-03**±1.8849e-03	**3.1133e-03**±1.0175e-03
f8	PSO	**1.4163e+03**±3.4863e+02	**1.4116e+03**±3.4684e+02	**1.4116e+03**±3.4684e+02
	PEPSO	1.5723e+03±3.9155e+02	1.5699e+03±3.9137e+02	1.5699e+03±3.9137e+02
f9	PSO	4.2079e+01±1.8540e+01	3.4745e+01±1.9708e+01	3.4088e+01±1.9313e+01
	PEPSO	**3.4998e+01**±2.0242e+01	**2.8398e+01**±1.6218e+01	**2.7282e+01**±1.6447e+01
f10	PSO	1.8034e-07±1.5886e-07	8.4199e-15±1.7046e-15	7.9226e-15±5.0243e-16
	PEPSO	**5.6148e-09**±1.3713e-08	**7.8515e-15**±7.0325e-16	**7.7094e-15**±9.7361e-16
f11	PSO	**8.3678e-03**±1.0734e-02	**7.9278e-03**±9.8584e-03	**7.9278e-03**±9.8584e-03
	PEPSO	1.4288e-02±2.0182e-02	1.3650e-02±2.0097e-02	1.2182e-02±1.7700e-02
f12	PSO	**2.0734e-03**±1.4661e-02	**1.5705e-32**±5.5294e-48	**1.5705e-32**±5.5294e-48
	PEPSO	4.1463e-03±2.9319e-02	4.1463e-03±2.9319e-02	4.1463e-03±2.9319e-02
f13	PSO	3.3020e-09±1.8370e-08	**1.3498e-32**±1.1059e-47	**1.3498e-32**±1.1059e-47
	PEPSO	**2.1011e-13**±8.9573e-13	**1.3498e-32**±1.1059e-47	**1.3498e-32**±1.1059e-47

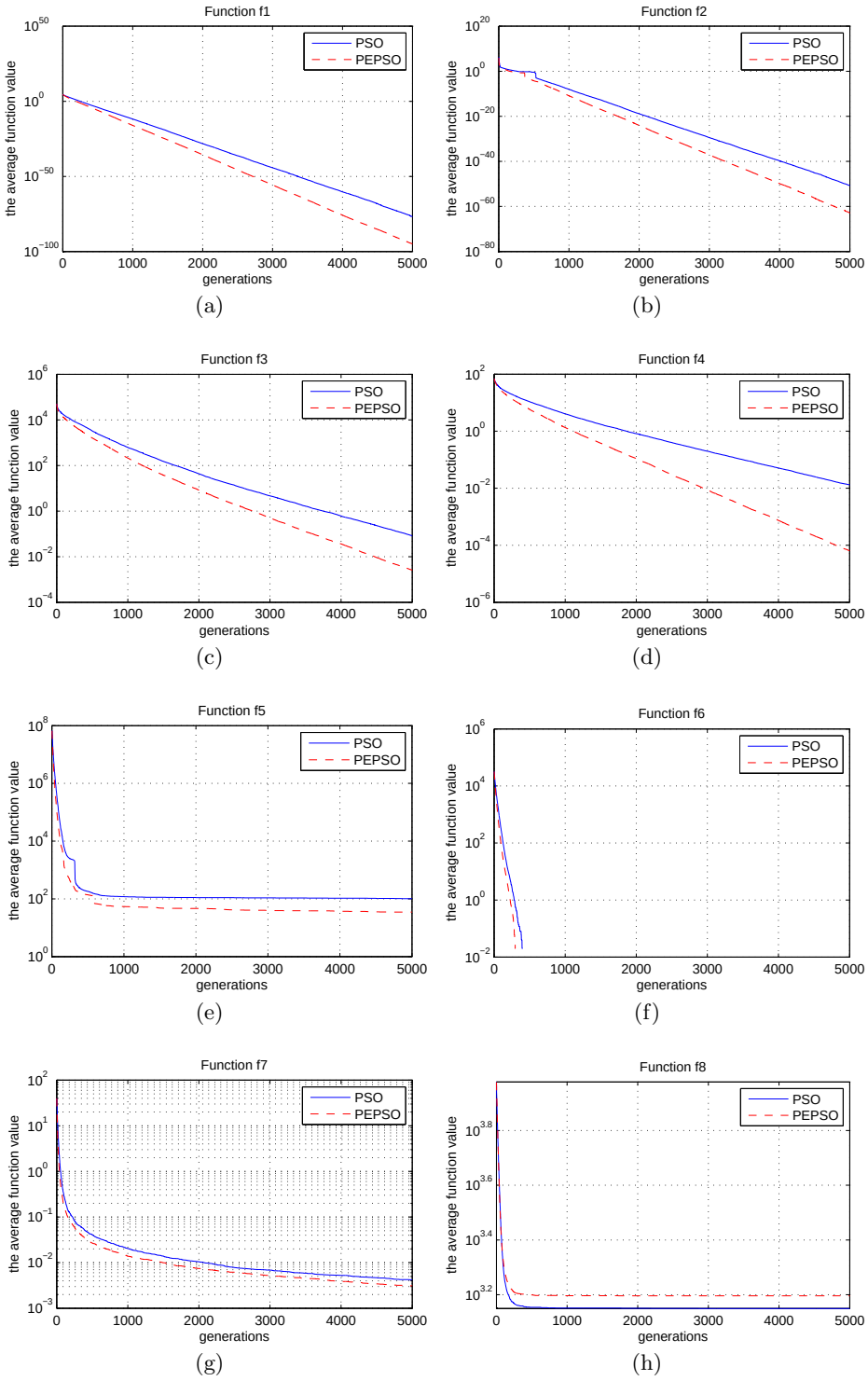

Fig. 2. The mean fitness values versus generations over 50 runs on f_1-f_8.

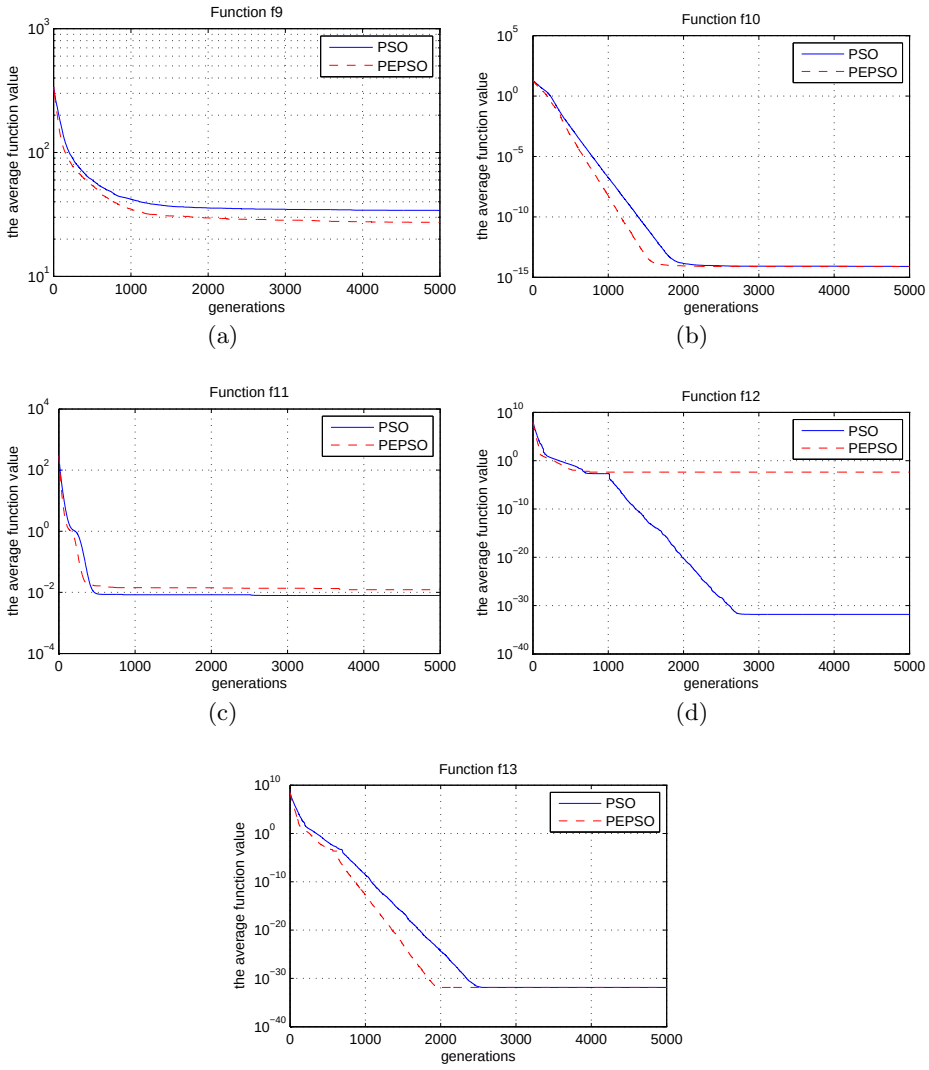

Fig. 3. The mean fitness values versus generations over 50 runs on f_9-f_{13}.

4.3 Results and Analysis

Table 1 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 illustrates 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 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].

References

1. K. DeJong, "Parameter setting in eas: a 30 year perspective," in *Parameter Setting in Evolutionary Algorithms*, 2007, pp. 1–18.
2. A. E. Eiben, Z. Michalewicz, M. Schoenauer, and J. E. Smith, "Parameter control in evolutionary algorithms," in *Parameter Setting in Evolutionary Algorithms*, 2007, pp. 19–46.
3. A. Zhou, Q. Zhang, and Y. Jin, "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*, vol. 13, no. 5, pp. 1167–1189, 2009.

4. G. G. Yen and H. Lu, "Dynamic multiobjective evolutionary algorithm: Adaptive cell-based rank and density estimation," *IEEE Transactions on Evolutionary Computation*, vol. 7, no. 3, pp. 253–274, 2003.
5. J. Brest, S. Greiner, B. Boskovic, M. Mernik, and V. Zumer, "Self-adapting control parameters in differential evolution: A comparative study on numerical benchmark problems," *IEEE Transactions on Evolutionary Computation*, vol. 10, no. 6, pp. 689 – 699, 2006.
6. S. Smit and A. Eiben, "Comparing parameter tuning methods for evolutionary algorithms," in *Proceedings of the 2009 IEEE Congress on Evolutionary Computation*. IEEE, 2009, pp. 399–406.
7. H. Mühlenbein and G. Paaß, "From recombination of genes to the estimation of distributions I. binary parameters," in *Parallel Problem Solving from Nature (PPSN IV)*, ser. LNCS, vol. 1411. Berlin: Springer, 1996, pp. 178–187.
8. P. Larrañaga and J. A. Lozano, *Estimation of Distribution Algorithms: A New Tool for Evolutionary Computation*. Kluwer Academic Publishers, 2002.
9. R. C. Eberhart and J. Kennedy, "A new optimizer using particle swarm theory," in *6th International Symposium on Micromachine and Human Science*, Nagoya, Japan, 1995, pp. 39–43.
10. J. Kennedy, R. C. Eberhart, and Y. Shi, *Swarm Intelligence*. San Francisco,US: Morgan Kaufmann, 2001.
11. X. Yao, Y. Liu, K.-H. Liang, and G. Lin, "Fast evolutionary algorithms," in *Advances in evolutionary computing: theory and applications*. New York, NY, USA: Springer-Verlag New York, Inc., 2003, pp. 45–94.
12. J. J. Liang, A. K. Qin, P. N. Suganthan, and S. Baskar, "Comprehensive learning particle swarm optimizer for global optimization of multimodal functions," *IEEE Transactions on Evolutionary Computation*, vol. 10, no. 3, pp. 281–295, 2006.
13. M. A. M. de Oca, T. Stützle, M. Birattari, and M. Dorigo, "Frankenstein's pso: A composite particle swarm frankenstein's pso: A composite particle swarm optimization algorithm," *IEEE Transactions on Evolutionary Computation*, vol. 13, no. 5, pp. 1120–1132, 2009.
14. Z.-H. Zhan, J. Zhang, Y. Li, and H.-H. Chung, "Adaptiveparticleswarmoptimization," *IEEE Transactions on Systems, Man and Cybernetics, Part B*, 2009.