An Estimation of Distribution Particle Swarm Optimization Algorithm

Mudassar Iqbal[1] and Marco A. Montes de Oca[2]

[1] Computing Laboratory, University of Kent
Canterbury, Kent, CT2 7NF, United Kingdom
mi26@kent.ac.uk
[2] IRIDIA, CoDE, Université Libre de Bruxelles, CP 194/6,
Av. Franklin D. Roosevelt 50, 1050 Brussels, Belgium
mmontes@ulb.ac.be

Abstract. In this paper we present an estimation of distribution particle swarm optimization algorithm that borrows ideas from recent developments in ant colony optimization. In the classical particle swarm optimization algorithm, particles exploit their individual memory to explore the search space. However, the swarm as a whole has no means to exploit its collective memory (represented by the array of pbests) to guide its search. This causes a re-exploration of already known bad regions of the search space, wasting costly function evaluations. In our approach, we use the swarm's collective memory to estimate the distribution of promising regions in the search space and probabilistically guide the particles' movement towards them. Our experiments show that this approach is able to find similar or better solutions than the standard particle swarm optimizer with fewer function evaluations.

1 Introduction

The first Particle Swarm Optimization (PSO) algorithm was introduced by Kennedy and Eberhart [4, 7] and was inspired by the social behavior of animals such as birds, fish and humans. Like other population-based optimization algorithms, PSO is initialized with a population of complete solutions (called particles) randomly located in a d-dimensional solution space. A particle i at time step t has a position vector $\boldsymbol{x}_i^t$ and a velocity vector $\boldsymbol{v}_i^t$, which are also randomly initialized. A fitness function $f : S \rightarrow \Re$ where $S \subset \Re^d$, determines the quality of a particle's position, i.e., a particle's position represents a solution to the problem being solved. Particles also have a vector that represents their own best previous position. So, for the i-th particle, the vector $\boldsymbol{p}_i$ has a fitness value $pbest_i = f(\boldsymbol{p}_i)$. Finally, the best position the swarm has ever visited during a run is stored in a vector $\boldsymbol{s}$ whose fitness value is $gbest = f(\boldsymbol{s})$[3].

[3] In most implementations, vector $\boldsymbol{s}$ does not exist. Rather, the index of the particle with the best position is assigned to variable g, so that $\boldsymbol{s} = \boldsymbol{p}_g$.

The algorithm iterates updating particles' velocity and position until a stopping criterion is met, usually a sufficiently good solution value or a maximum number of iterations. The updating rules are:

$$\boldsymbol{v}_i^{t+1} = \boldsymbol{v}_i^t + \varphi_1 \boldsymbol{U}_1(0,1) * (\boldsymbol{p}_i - \boldsymbol{x}_i^t) + \varphi_2 \boldsymbol{U}_2(0,1) * (\boldsymbol{s} - \boldsymbol{x}_i^t) \tag{1}$$

$$\boldsymbol{x}_i^{t+1} = \boldsymbol{x}_i^t + \boldsymbol{v}_i^{t+1} \tag{2}$$

where φ_1 and φ_2 are two constants called the *cognitive* and *social* coefficients respectively, $\boldsymbol{U}_1(0,1)$ and $\boldsymbol{U}_2(0,1)$ are two d-dimensional uniformly distributed random vectors in which each component goes from zero to one, and $*$ is an element-by-element vector multiplication operator.

Clerc and Kennedy [2] introduced the concept of *constriction* in PSO. Since it is based on a rigorous analysis of the dynamics of a simplified model of the original PSO, it became highly influential in the field to the point that it is now referred to as the *canonical* PSO. The difference with respect to the original PSO is the addition of a constriction factor in Equation 1. The modified velocity updating rule becomes

$$\boldsymbol{v}_i^{t+1} = \chi(\boldsymbol{v}_i^t + \varphi_1 \boldsymbol{U}_1(0,1) * (\boldsymbol{p}_i - \boldsymbol{x}_i^t) + \varphi_2 \boldsymbol{U}_2(0,1) * (\boldsymbol{s} - \boldsymbol{x}_i^t)) \tag{3}$$

with

$$\chi = \frac{2k}{\left|2 - \varphi - \sqrt{\varphi^2 - 4\varphi}\right|} \tag{4}$$

where $k \in [0,1]$, $\varphi = \varphi_1 + \varphi_2$ and $\varphi > 4$. Usually, χ is set to 0.729 and φ_1 and φ_2 are set to 2.05 [5, 16]. This is the PSO version we use in our comparisons[4].

From Equations 1 and 3, it is clear that the behavior of every particle is partially determined by its previous experience. This memory allows a particle to search somewhere around its own previous best position and the best position ever found by a particle in its neighborhood. However, during a search different particles move and test (i.e., evaluate the objective function) over and over again the same or approximately the same region in the search space without any individual improvement. While this is part of the search process and allows the swarm to escape from local optima, it is also a waste of computing power when the explored regions have been visited before by the swarm. This happens because the swarm as a single entity does not learn.

In this paper, we present a generic modification to the PSO paradigm that allows a particle swarm estimate the distribution of promising regions of the fitness landscape by exploiting the information it gains during the optimization process. This distribution is in turn used to try to keep the particles within the promising regions. It is a modular extension that can be used in any PSO variant

[4] This is perhaps not the best choice. The canonical PSO has not been *proved* to be the best performing PSO. However, since there is no agreement on this issue, we decided to use this version as our reference.

that uses a position update rule based on previously found solutions. The estimation of the distribution is done by means of a mixture of normal distributions taking into account the array of *pbests*. It borrows some ideas from recent developments in Ant Colony Optimization (ACO) [3] in which an archive of solutions is used to select the next point to explore in the search space. The underlying assumption of independence between variables common to many Estimation of Distribution Algorithms (EDAs) for continuous optimization problems (see [12]) is also present in this work.

The rest of the paper is organized as follows. Section 2 presents some background information on the class of estimation of distribution optimization algorithms to which our proposed algorithm belongs. Section 3 presents in detail the estimation of distribution particle swarm optimizer proposed in this paper. In section 4 we describe the experimental setup we used to assess the performance of our proposed algorithm. Section 5 presents our empirical results along with some discussion. In section 6 we conclude.

2 Estimation of Distribution Optimization Algorithms

Evolutionary Algorithms (EAs) that use information obtained during the optimization process to build probabilistic models of the distribution of good regions in the search space and that use these models to generate new solutions are called Estimation of Distribution Algorithms (EDAs) [12]. The fully joint probability distribution characterizes the problem being solved. Depending on whether there is *a priori* knowledge about the underlying distribution or not, one can use a suitable parameterization to get fast convergence rates or use machine learning methods to approximate this unknown distribution, respectively. The latter case is the most commonly found in practice.

EDAs differ in the way they gather information during the optimization process, use the gathered information to build probabilistic models, and in the way they use these models to generate new solutions. An experimental comparison of some of the best known EDAs has been done by Kern et al. [9].

A pseudo-code view of the algorithmic structure behind most EDAs can be seen in Algorithm 1. An EDA starts with a solution population $\mathbf{X}^0$ and a solution distribution model $\mathcal{P}^0$. The mail loop consists of four principal stages. The first stage is to select the best individuals (according to some fitness criteria f) from the population. These individuals are used in a second stage in which the solution distribution model $\mathcal{P}^t$ is updated or recreated. The third stage consists of sampling the updated solution distribution model $\mathcal{P}^{t+1}$ to generate new solutions $\mathbf{X}^{t+1}_{offspring}$. The last stage involves the base population $\mathbf{X}^t_{base}$, the new solutions and the fitness criteria. The end result is a new base population and the process starts over again until the stopping criteria are satisfied.

There has been a growing interest for EDAs in the last years. It is out of the scope of this paper to describe the approaches taken to implement the ideas just described. For a comprehensive presentation of the field see the work of Larrañaga and Lozano [10].

Algorithm 1 Algorithmic structure of EDAs.

/* Initialization */
Initialize population of solutions $\mathbf{X}_{base}^{0}$ and solution distribution model $\mathcal{P}^{0}$

/* Main Loop */
while Stopping criteria are not satisfied **do**
$\mathbf{X}_{parent}^{t} = select(\mathbf{X}_{base}^{t}, f)$ /* Selection */

$\mathcal{P}^{t+1} = estimate(\mathbf{X}_{parent}^{t}, \mathcal{P}^{t})$ /* Estimation */

$\mathbf{X}_{offspring}^{t+1} = sample(\mathcal{P}^{t+1})$ /* Sampling */

$\mathbf{X}_{base}^{t+1} = replacement(\mathbf{X}_{offspring}^{t+1}, \mathbf{X}_{base}^{t}, f)$ /* Replacement */

$t = t + 1$
end while

3 Estimation of Distribution Particle Swarm Optimization Algorithm

PSO algorithms are considered to be part of the emerging field of *Swarm Intelligence* [1,8]. Swarm Intelligence is the discipline that studies natural and artificial systems comprised of multiple simple entities that collectively exhibit adaptive behaviors. Some examples of natural swarm intelligent systems are ant colonies, slime molds, bee and wasp swarms.

Besides PSO, the other prominent representative of artificial swarm intelligent systems is Ant Colony Optimization (ACO) [3]. ACO is usually used for solving combinatorial optimization problems. In ACO, artificial ants build solutions incrementally selecting one solution component at a time. The probabilistic selection is biased by a trail of *pheromone* deposited by other ants in previous iterations of the algorithm. The amount of pheromone is proportional to the quality of complete solutions, so that ants will prefer to choose previously known good solution components than bad ones. In fact, the role of the so-called pheromone matrix is to approximate the distribution of good solutions in the search space. Seen from this point of view, ACO is an EDA.

A recent development of ACO that extends it to continuous optimization is called ACO_R [13, 14]. ACO_R approximates the joint probability distribution one dimension at a time by using mixtures of weighted Gaussian functions. This allows the algorithm to deal with multimodal functions. Figure 1 illustrates the idea of approximating the distribution of good regions in a single dimension using a mixture of weighted Gaussian functions.

The source of information to parameterize these univariate distributions is an archive of solutions of size k. The i-th component of solution the l-th solution is denoted by s_l^i. For an n-dimensional problem, $1 \leq i \leq n$ and $1 \leq l \leq k$. For each dimension i, the vector $\boldsymbol{\mu}_i =< s_1^i, \ldots, s_k^i >$ is the vector of means that is used to model the univariate probability distribution of the i-th dimension. The

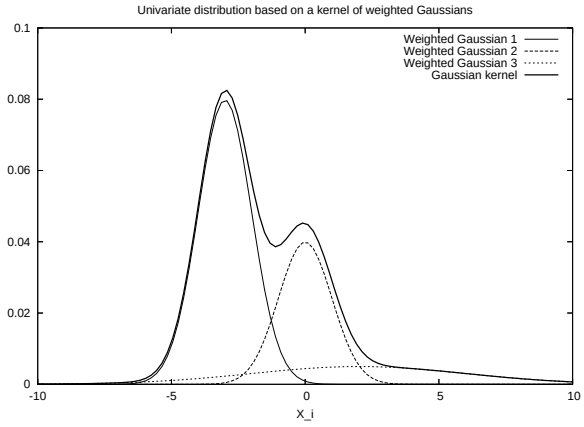


Fig. 1. Mixture of weighted Gaussian functions

vector of weights $\boldsymbol{w} =< w_1, \ldots, w_k >$ is the same across all dimensions because it is based on the relative quality of the complete solutions. Every iteration, after the solutions are ranked, the weights are determined by

$$w_l = \frac{1}{qk\sqrt{2\pi}} e^{-\frac{(l-1)^2}{2(qk)^2}} \tag{5}$$

where q is a parameter that determines the degree of preferability of good solutions. With a small q the best solutions are strongly preferred to guide the search.

Since ACO$_R$ samples the mixture of Gaussians, it has to first select one of the Gaussian functions from the kernel. The selection is done probabilistically. The probability of choosing the l-th Gaussian function is

$$p_l = \frac{w_l}{\sum_{j=1}^{k} w_j} \tag{6}$$

Then, ACO$_R$ computes the standard deviation of the chosen Gaussian function as

$$\sigma_l^i = \xi \sum_{e=1}^{k} \frac{|x_e^i - x_l^i|}{k-1} \tag{7}$$

where ξ is a parameter that allows the algorithm balance its exploration – exploitation behaviors. ξ has the same value for all the dimensions. Having computed all the needed parameters, ACO$_R$ samples the Gaussian function to generate a new solution component. The process is repeated for every dimension, for every ant until a stopping criterion is met.

This lengthy presentation of ACO_R was needed to introduce our Estimation of Distribution Particle Swarm Optimization (EDPSO) algorithm. The reason is that EDPSO borrows some ideas from ACO_R. First, the array of *pbests* plays the role of the solution archive in ACO_R. In EDPSO, k (i.e., the size of the solution archive) is equal to the number of particles. The dynamics of the algorithm, however, is somewhat different. EDPSO works as a canonical PSO as described in section 1 but with some modifications: after the execution of the velocity update rule shown in Equation 3 the EDPSO selects one Gaussian function just as ACO_R does. Then, the selected Gaussian function is evaluated (not sampled) to probabilistically move the particle to its new position. If the movement is successful, the algorithm continues as usual, but if the movement is unsuccessful, then the selected Gaussian function is sampled in the same way as in ACO_R. The result is a "hybrid" algorithm that explores the search space using the PSO dynamics but when this approach fails (i.e., when a particle's tendency is to move far away from good solutions) a direct sampling of the probability distribution is used instead. It is important to mention that when the selected Gaussian function is evaluated, we use an *unscaled* version of it, so that its range is [0,1] (i.e., a true probability). A pseudo-code version of EDPSO can be seen in Algorithm 2.

4 Experimental Setup

To evaluate the performance of EDPSO we used the most commonly used benchmark functions in the PSO literature (see [6] for details). We have compared our algorithm with the canonical PSO as described in section 1. Table 1 shows the initialization ranges and the goals that had to be achieved by the algorithms in terms of solution quality, although this goal was not used as a stopping criterion. We ran 30 independent runs for each function in 30,40 and 50 dimensions for a maximum of 120 000, 160 000, and 200 000 function evaluations respectively. The number of particles was equal to 40.

Table 1. Parameters for the test functions

Function	Initialization range	Goal
Sphere	$[-100, 100]^D$	0.01
Rosenbrock	$[-30, 30]^D$	100
Rastrigin	$[-5.12, 5.12]^D$	100
Griewank	$[-600, 600]^D$	0.1
Ackley	$[-32, 32]^D$	0.1

All the benchmark functions we used have the global optimum at or very near the origin, i.e., at the center of the search domain and hence a symmetric uniform initialization would induce a possible bias [11]. To avoid this problem, all functions were shifted to a random location within the search range. This

Algorithm 2 Pseudocode version of the EDPSO algorithm

/* Initialization. k is the number of particles, and n is the dimension of the problem */
for $i = 1$ to k **do**
  Create particle i with random position and velocity
end for
Initialize *gbest* and all $pbest_i$ to some sensible values

/* Main Loop */
$t = 0$
while *gbest* is not good enough or $t < t_{max}$ **do**
  /* Evaluation Loop */
  for $i = 1$ to k **do**
    if $f(\boldsymbol{x}_i)$ is better than $pbest_i$ **then**
      $\boldsymbol{p}_i = \boldsymbol{x}_i$
      $pbest_i = f(\boldsymbol{x}_i)$
    end if
    if $pbest_i$ is better than *gbest* **then**
      $gbest = pbest_i$
      $\boldsymbol{s} = \boldsymbol{p}_i$
    end if
  end for
  /* Update Loop */
  Rank all $pbest_i$ according to their quality
  Compute $\boldsymbol{w} =< w_1, \ldots, w_k >$ using Equation 5
  Compute all p_l using Equation 6
  for $i = 1$ to k **do**
    for $j = 1$ to n **do**
      $v_{ij} = \chi(v_{ij} + \varphi_1 U_1(0,1)(p_{ij} - x_{ij}) + \varphi_2 U_2(0,1)(s_{ij} - x_{ij}))$
      $x_{ij}^{candidate} = x_{ij} + v_{ij}$
      Select a Gaussian function from the kernel according to p_l, name it g_l^i.
      Compute σ_l^i using Equation 7
      $prob_{move} = \sigma_l^i \sqrt{2\pi} g_l^i(x_{ij}^{candidate})$ /* $\sigma_l^i \sqrt{2\pi}$ unscales the function */
      if $U_3(0,1) < prob_{move}$ **then**
        $x_{ij} = x_{ij}^{candidate}$ /* The particle moves normally */
      else
        $x_{ij} = sample(g_l^i)$ /* New position is a sample from the chosen function */
      end if
    end for
  end for
  $t = t + 1$
end while

approach has been used before and does not confine the swarm to a small region of the search space as is usually done with asymmetrical initializations [15].

Table 2 shows the parameter settings for the algorithms used in our experiments.

Table 2. Parameters used by the algorithms

Algorithm	Parameter	Value
Canonical PSO	φ_1	2.05
	φ_2	2.05
	χ	0.729
EDPSO	φ_1	2.05
	φ_2	2.05
	χ	0.729
	q	0.1
	ξ	0.85

5 Results

The benefits of estimating the probability distribution of good regions in the search space and to guide the swarm to search them are reflected (in general) in the quality of the solutions achieved, as well as in the number of function evaluations needed to achieve a solution of certain quality. Table 3 shows the average fitness value (of the best particle in the swarm) after the maximum number of allowed function evaluations.

Table 3. Average fitness value after the maximum number of allowed function evaluations over 30 runs

Algorithm	Dimension	Sphere	Rosenbrock	Rastrigin	Griewank	Ackley
Canonical PSO	30	0.0	37.48	73.52	0.023	13.35
	40	0.0	55.06	133.15	0.037	18.78
	50	0.0	102.4	203.8	0.1	18.3
EDPSO	30	0.0	22.3	25.6	0.0012	0.000019
	40	0.0	37.3	33.43	0.00098	0.00004
	50	0.0	48.12	56.18	0.0029	0.7

In all benchmark functions, except in the case of the Sphere function, a tendency can be immediately recognized: EDPSO can find better solution qualities after the same number of function evaluations. This is particularly true in the case of the Rastrigin and Ackley functions.

Regarding the issues of speed and reliability, Table 4 shows the average number of function evaluations needed to achieve the solution qualities defined in

Table 1 and the probability of achieving them, defined as the success rate (a successful run is one that achieves the specified goal). The average was computed over the successful runs only and rounded off to the nearest integer number greater than the actual number.

Table 4. Average number of function evaluations needed to achieve the solution qualities defined in Table 1 and the probability of achieving them.

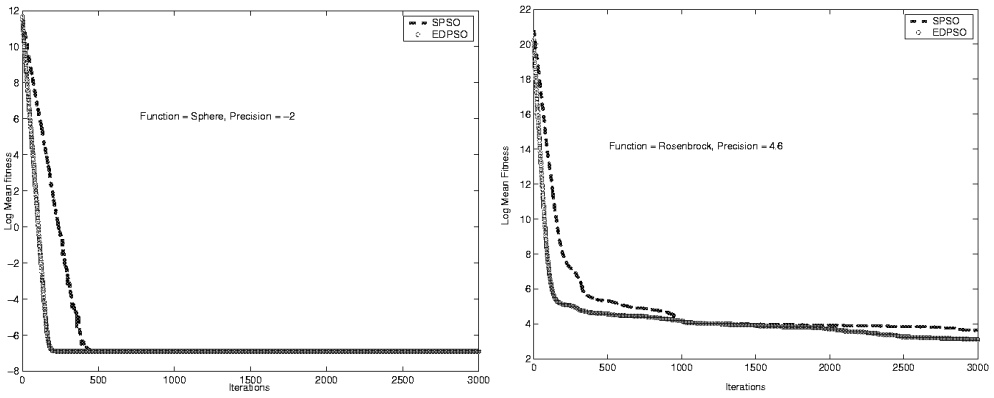
Algorithm	Dimension	Sphere	Rosenbrock	Rastrigin	Griewank	Ackley
Canonical PSO	30	13049,1.0	20969,0.86	7880,0.9	11907,0.93	13980,0.06
	40	19365,1.0	38442,0.83	13296,0.16	17563,0.93	–
	50	27451,1.0	61124,0.66	–	24584,0.76	–
EDPSO	30	5988,1.0	20921,0.96	18549,1.0	5520,1.0	5656,1.0
	40	8717,1.0	24896,0.9	28045,1.0	7866,1.0	8437,1.0
	50	11971,1.0	50442,0.86	41659,1.0	10741,1.0	20284,0.96

From Table 4, it can be seen that EDPSO is faster than the Canonical PSO in getting to the desired objective function value in all functions except in Rastrigin. Entries marked with "–" specify cases in which the goal was not reached in any run. From our experiments, it can be observed that EDPSO shows a significant improvement in terms of the number of function evaluations it needs to get to a certain solution quality. It should also be noted that the behavior of the Canonical PSO is not robust as we go into higher dimensions. In contrast, EDPSO is quite consistent. The Rastrigin case is better explained after examining Figure 5 which shows how the solution quality improves over time for the benchmark problems in 30 dimensions.

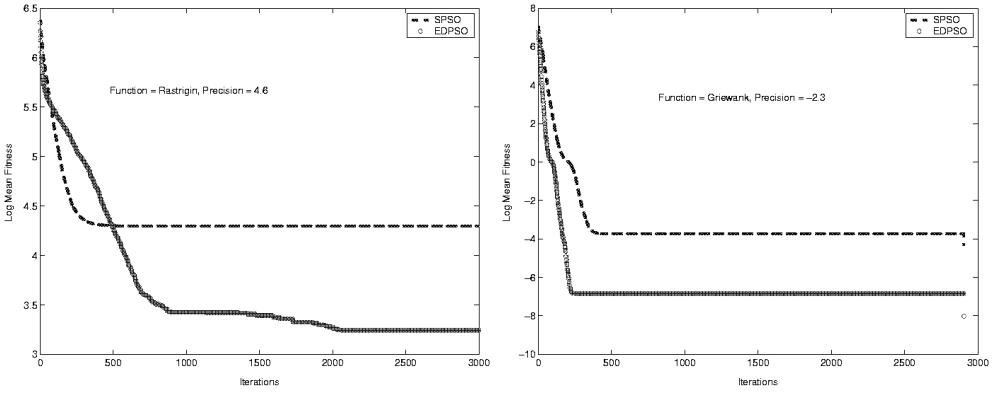
The data in Tables 3 and 4 give only a partial view of the behavior of the algorithms. Specifically, they do not show how the solution quality evolves over time. Knowing this is particularly useful to identify which algorithm is best suited for real-time applications in which there are hard time limits or for applications in which we are interested in the solution quality only. In Figure 2(c) it can be seen how the goal defined in Table 1 was reached first by the Canonical PSO but it can also be seen how it stagnates and cannot find better solutions after some more iterations.

6 Conclusions

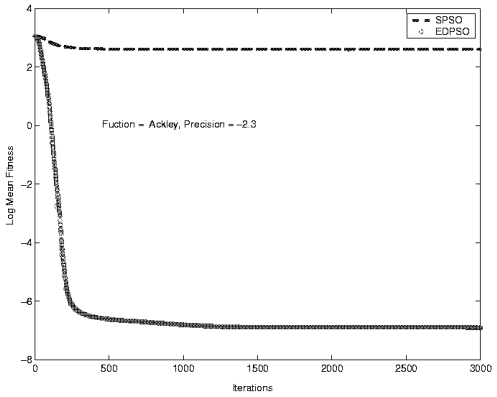
In this paper we have introduced an Estimation of Distribution Particle Swarm Optimization (EDPSO) algorithm. It is in fact a modular extension that can be used in any other PSO variant that uses a position update rule based on previously found solutions. In effect, it is a learning mechanism that helps the particle swarm explore potentially good regions of the search space. It benefits from the information gathered during the optimization process that is encoded in the array of *pbests*. The end result is a PSO variant that not only finds



(a) Sphere (b) Rosenbrock



(c) Rastrigin (d) Griewank



(e) Ackley

Fig. 2. Solution quality over time. Lines represent the average solution value.

better solutions than the Canonical PSO, but also does it with fewer function evaluations. There are some cases, however in which speed is sacrificed for the sake of finding better solutions.

EDPSO is not a pure Estimation of Distribution Algorithm (EDA). It explores the search space in the same way as the Canonical PSO but becomes an EDA whenever particles are pushed further away from good regions (so learnt by the whole swarm). It remains a research issue the problem of handling interactions between variables and the correct parameterization of the probability distributions. The results reported here are encouraging enough to continue looking for ways to allow the particle swarm learn from its past experience.

Acknowledgments

References

1. E. Bonabeau, M. Dorigo, and G. Theraulaz. *Swarm Intelligence: From Natural to Artificial Systems*. Santa Fe Institute Studies on the Sciences of Complexity. Oxford University Press, USA, 1999.
2. M. Clerc and J. Kennedy. The particle swarm–explosion, stability, and convergence in a multidimensional complex space. *IEEE Transactions on Evolutionary Computation*, 6(1):58–73, 2002.
3. M. Dorigo and T. Stützle. *Ant Colony Optimization*. Bradford Books. The MIT Press, 2004.
4. R. Eberhart and J. Kennedy. A new optimizer using particle swarm theory. In *Proceedings of the 6th International Symposium on Micro Machine and Human Science*, pages 39–43, 1995.
5. R. Eberhart and Y. Shi. Comparing inertia weights and constriction factors in particle swarm optimization. In *Proceedings of the 2000 IEEE Congress on Evolutionary Computation*, pages 84–88, 2000.
6. S. Janson and M. Middendorf. A hierarchical particle swarm optimizer and its adaptive variant. *IEEE Transactions on Systems, Man and Cybernetics–Part B*, 35(6):1272–1282, 2005.
7. J. Kennedy and R. Eberhart. Particle swarm optimization. In *Proceedings of IEEE International Conference on Neural Networks*, pages 1942–1948, 1995.
8. J. Kennedy, R. Eberhart, and Y. Shi. *Swarm Intelligence*. Morgan Kaufmann Publishers, San Francisco, CA, USA, 2001.
9. S. Kern, S. D. Müller, N. Hansen, D. Büche, J. Ocenasek, and P. Koumoutsakos. Learning probability distributions in continuous evolutionary algorithms–a comparative review. *Natural Computing*, 3(1):77–112, 2004.
10. P. Larrañaga and J. Lozano. *Estimation of Distribution Algorithms: A New Tool for Evolutionary Computation*. Genetic Algorithms and Evolutionary Computation, Vol. 2. Springer, 2001.
11. C. K. Monson and K. D. Seppi. Exposing origin-seeking bias in pso. In H. G. B. et al., editor, *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO)*, pages 241–248, New York, NY, June 2005. ACM Press.
12. M. Pelikan, D. E. Goldberg, and F. Lobo. A survey of optimization by building and using probabilistic models. *Computational Optimization and Applications*, 21(1):5–20, 2002.

13. K. Socha. ACO for Continuous and Mixed-Variable Optimization. In M. Dorigo, M. Birattari, and C. Blum, editors, *Proceedings of ANTS 2004 – Fourth International Workshop on Ant Colony Optimization and Swarm Intelligence*, volume 3172 of *LNCS*, pages 25–36. Springer-Verlag, Berlin, Germany, 5-8 September 2004.
14. K. Socha and M. Dorigo. Ant colony optimization for continuous domains. Technical Report TR/IRIDIA/2005-037, IRIDIA, Université Libre de Bruxelles, December 2005.
15. P. N. Suganthan, N. Hansen, J. J. Liang, K. Deb, Y.-P. Chen, A. Auger, and S. Tiwari. Problem definitions and evaluation criteria for the CEC 2005 special session on real-parameter optimization. Technical Report 2005005, Nanyang Technological University, Singapore and IIT Kanpur, India, May 2005.
16. I. C. Trelea. The particle swarm optimization algorithm: Convergence analysis and parameter selection. *Information Processing Letters*, 85(6):317–325, 2003.