# Particle Filter with Swarm Move for Optimization

Chunlin Ji[1], Yangyang Zhang[2], Mengmeng Tong[3], and Shengxiang Yang[4]

[1] Department of Statistical Sciences, Duke University
Durham, North Carolina 27708, USA
[2] Department of Engineering Science, University of Oxford
Parks Road, Oxford OX1 3PJ, UK
[3] School of Information Science and Technology
Northeastern University, Shenyang 110004, China
[4] Department of Computer Science, University of Leicester
University Road, Leicester LE1 7RH, UK

**Abstract.** We propose a novel generalized algorithmic framework to utilize particle filter for optimization incorporated with the swarm move method in particle swarm optimization (PSO). In this way, the PSO update equation is treated as the system dynamic in the state space model, while the objective function in optimization problem is designed as the observation/measurement in the state space model. Particle filter method is then applied to track the dynamic movement of the particle swarm and therefore results in a novel stochastic optimization tool, where the ability of PSO in searching the optimal position can be embedded into the particle filter optimization method. Finally, simulation results show that the proposed novel approach has significant improvement in both convergence speed and final fitness in comparison with the PSO algorithm over a set of standard benchmark problems.

## 1 Introduction

Particle filter, also known as sequential Monte Carlo (SMC), is a class of importance sampling and resampling techniques designed to simulate from a sequence of probability distributions, which has gained popularity for the last decade to solve sequential Bayesian inference problems [1]. It was recently extended to a general framework to deal with static and sequential Bayesian inference, as well as the global optimization. In order to deal with an optimization problem, a sequence of artificial dynamic distribution was designed to employ the particle filter algorithm. The basic idea of particle filter optimization (PFO) method was first presented in our previous works [2] to solve discrete optimization problems in wireless communication system. The crucial element in the PFO algorithm is how to design the system dynamic function, which forces the set of particles to move toward the 'promising' area containing optima.

Particle Swarm Optimization (PSO) is a well studied heuristic optimization technique inspired by the social behavior observable in nature, such as flocks of

G. Rudolph et al. (Eds.): PPSN X, LNCS 5199, pp. 909–918, 2008.

birds and schools of fish [3]. In a basic PSO algorithm, a set of particles is generated randomly, and their positions (states) are iteratively updated according to their own experience and the experience of the swarm (or neighboring particles). The heuristic strategy of swarm move works well and therefore enable the PSO algorithm being effective and efficient. However, the PSO is not universal for all optimization problems because it suffers the premature convergence which enables the PSO algorithm easily to get stuck in local minima.

A novel algorithmic framework is presented in this paper, where the swarm move strategy is incorporated into particle filter optimization algorithm. The update equation of particle swarm move in PSO algorithm is treated as the system dynamic of a state space model, while the objective function in optimization problem is designed as the observation model to motivate the swarm moving toward the optimal position. In particle filter, the particles first evolve according to the system dynamic model where they 'learn' the information from the whole population. After that, the particles are updated by information from the observation model where they 'learn' the information from the objective function. Therefore, the particles move towards location of the global optima sequentially by 'learning' the information from these two aspects. By using the swarm move strategy as the system dynamic, the desirable searching mechanism of PSO is incorporated into the PFO algorithm and enhance its searching ability. The proposed novel algorithm incorporates the state space probability modelling and resample strategy to the PSO algorithm, which potentially enhance the ability of PSO algorithm in two aspects: making it easier to jump out local optima and further refining the final results.

This paper is organized as follows. Section 2 reviews the basic PSO algorithm. Section 3 introduces the basic particle filter algorithm. In Section 4, we propose the particle filter with particle swarm move for optimization problem. In section 5, the proposed algorithm is tested on a set of benchmark problems. Finally, we conclude the paper in Section 6.

## 2 Particle Swarm Optimization

### 2.1 The Basic PSO Algorithm

The basic PSO algorithm first starts with a number of particles which are randomly generated in the function domain space. After that, each particle flies through the search space with a velocity which is dynamically adjusted according to its own flying experience and the experience from neighboring particles. Specifically, the behavior of each particle is affected by either the local best or the global best particle to help it fly through a hyperspace. Therefore, by observing the behavior of the flock and memorizing their flying histories, all particles in the swarm can quickly converge to near-optimal geographical positions [3]. The particles are updated according to the following equations:

$$v_n = wv_{n-1} + \phi_1(x_{ibest} - x_{n-1}) + \phi_2(x_{gbest} - x_{n-1}) \tag{1}$$

$$x_n = x_{n-1} + v_n. \tag{2}$$

Equation (1) calculates a new velocity for each particle (potential solution) based on its previous velocity $v_{n-1}$, the particle's location at which the best fitness has been achieved $x_{ibest}$, and the population global (or local neighborhood, in the neighborhood version of the algorithm) location $x_{gbest}$ (or $x_{nbest}$ , in neighborhood version) at which the best fitness so far has been achieved. Equation (2) updates each particle's position in solution hyperspace. The two random control factors $\phi_i$ $(i = 1, 2)$ are drawn from $U(0, 2.05)$. Moreover, $\omega$ is applied to further improve the convergence rate of the PSO, [3].

### 2.2 Improvement for PSO Algorithm

Improvement for PSO algorithms has been studied extensively in various ways such as increased swarm diversity [4], evolutionary selection [5] and adaptive parameters in the velocity update equations [6]. Although these improvements achieve significant success, they generally obtain superior minima at the expense of iterations. In other words, they concentrate on how to obtain better final fitness but not how to obtain them faster.

Obviously, there is a tradeoff between convergence speed and the values of final fitness for nonlinear optimization methods [4], which means that improving one is at the expense of the other. However, given an order of magnitude for the final solution fitness, it is still possible to obtain satisfied solutions faster [7]. The Kalman swarm (KSwarm) proposed in [7], first tried to address this issue in PSO. In KSwarm, a reformulated PSO update equation is treated as the system dynamic in the state space model, while the observation function is a measurement of the best position each particle has obtained in the past. The results in [7] showed that the KSwarm algorithm has a significant improvement in both convergence speed and final fitness. This research works in [7] shows that the possibility or statistical approaches for improvement in PSO algorithm has a great potential. Inspired by the KSwarm, we propose a novel algorithmic framework to combine the particle filter with PSO algorithm, which can effectively get ride of the heavy extra computation problem incurred by KSwarm[1]. Moreover, rather than learning from the best position it has obtained in the past, the particle in the proposed algorithm learns from the information of the whole probability density function formed by all the particles.

## 3 Particle Filter

Particle filter, introduced in [8], is a class of importance sampling and resampling techniques designed to simulate from a sequence of probability distributions for sequential inference problems. These methods have gained popularity in recent

[1] The superior performance of the KSwarm is at the cost of additional computation complexity which is incurred by the matrix operations in Kalman update equations whose complexity order is $O(d^3)$ in the number of dimensions [7].

years, due to their simplicity, flexibility, ease of implementation, and modelling success over a wide range of challenging applications [1].

Give a state space model,

$$x_n = f(x_{n-1}) + w_n, \tag{3}$$

$$y_n = h(x_n) + v_n, \tag{4}$$

where $f(\cdot)$ is the system evolution function, $h(\cdot)$ is the observation function, $w_n$ and $v_n$ are the system noise and observation noise respectively. We refer to $n$ as the time index, which is just a simple counter and has not any relationship with 'real' time. In the context of sequential Bayesian inference, we are interested in the posterior distribution $\pi(x_n|y_{1:n})$ (where $y_{1:n}$ denotes the observations $y_1, y_2, ..., y_n$), which can be recursively obtained from the following two equations:

$$\pi(x_n|y_{1:n-1}) \propto \int p(x_n|x_{n-1})\pi(x_{n-1}|y_{1:n-1})dx_{n-1}, \tag{5}$$

and

$$\pi(x_n|y_{1:n}) \propto L(x_n; y_n)\pi(x_n|y_{1:n-1}), \tag{6}$$

where $p(x_n|x_{n-1})$ represents the system dynamic in (3), $\pi(x_{n-1|}y_{1:n-1})$ in (5) is the posterior distribution at $(n-1)$, and $L(x_n; y_n)$ in (6) refers to the likelihood function obtained in (4). The recursion is initialised with some distribution, for example, $p(x_0)$.

In very limited scenarios, the state space models of interest are 'weakly' nonlinear and Gaussian in which one may utilize the Kalman filter and its derivatives [9], to obtain an approximately optimal solution. In practice, it is well known that the update expression in (6) is generally analytically intractable for most models of interest. We therefore turn to sequential Monte Carlo (SMC) methods [1,8], also known as particle filters, to provide an efficient numerical approximation strategy for recursive estimation of complex models.

### 3.1 Sequential Importance Sampling

The basic idea behind particle filters is very simple: the target distribution is represented by a weighted set of Monte Carlo samples which are called particles in this paper. These particles are propagated and updated using a sequential version of importance sampling as new measurements become available. Hence statistical inferences of the posterior $\pi(x_n|y_{1:n})$ can be computed by these particles.

From a large set of particles $\left\{x_{n-1}^{(i)}\right\}_{i=1}^{N}$ associated importance weights $\left\{w_{n-1}^{(i)}\right\}_{i=1}^{N}$, we approximate the posterior distribution function $\pi\left(x_{n-1}|y_{1:n-1}\right)$ as follows:

$$\pi\left(x_{n-1}|y_{1:n-1}\right) \approx \sum_{i=1}^{N} w_{n-1}^{(i)} \delta\left(x_{n-1} - x_{n-1}^{(i)}\right), \tag{7}$$

where $\delta(\cdot)$ is the Dirac delta function. We would like to generate a set of new particles $\left\{x_n^{(i)}\right\}_{i=1}^{N}$ from an appropriately selected proposal function, i.e.,

$$x_n^{(i)} \sim q\left(x_n | x_{n-1}^{(i)}, y_{1:n}\right), i = \{1, ..., N\}. \tag{8}$$

With the set of state particles $\left\{x_n^{(i)}\right\}$ obtained from (8), the importance weights $w_n^{(i)}$ are recursively updated as follows:

$$w_n^{(i)} \propto w_{n-1}^{(i)} \times \frac{L\left(x_n^{(i)}; y_n\right) p\left(x_n^{(i)} | x_{n-1}^{(i)}\right)}{q\left(x_n^{(i)} | x_{n-1}^{(i)}, y_{1:n}\right)} \tag{9}$$

with $\sum_{i=1}^{N} w_n^{(i)} = 1$. It follows that the new set of particles $\left\{x_n^{(i)}\right\}_{i=1}^{N}$ with the associated importance weights $\left\{w_n^{(i)}\right\}_{i=1}^{N}$ is then approximately distributed according to $\pi\left(x_n | y_{1:n}\right)$.

As the particle filters operate, only a few particles contribute significant importance weights in (9), which leads to a degeneracy problem [10]. To avoid this problem, one possible method is to resample the particles according to the importance weights. With this method, the particles with more significant weights will be selected more frequently than those with less significant weights. More detailed discussions of degeneracy and resampling can be found in [10].

An important element in generating a set of weighted particles which could well approximate the posterior distribution function in (4) is the selection of the proposal importance sampling function $q(x_n^{(i)} | y_{1:n})$ in (8). One choice of the state proposal function is the dynamic prior, $q\left(x_n | x_{n-1}, y_{1:n}\right) = p\left(x_n | x_{n-1}\right)$. Weights become proportional to likelihood, $w_n \propto w_{n-1} L(x_n; y_n)$. In the proposed particle filter optimization algorithm in the following sections, we will restrict to utilize this simple but generic effective proposal.

### 3.2 Particle Filter for Optimization

Particle filter technique has recently been extended to a general framework to deal with the static and sequential Bayesian inference, as well as the global optimization [11], which is also called sequential Monte Carlo sampler. To apply the SMC sampler for optimization problem, a sequence of artificial intermediate distributions is required, for example $\pi_n(x) = [\pi(x)]^{\tau_n}$ where $\{\tau_n\}_{n=1}^{N}$ is such that $0 < \tau_1 < \cdots < \tau_N$ and $1 << \tau_N$ to ensure that $\pi_0(\cdot)$ is easy to sample from and $\pi_N(\cdot)$ is concentrated around the set of global maxima of $\pi(\cdot)$. Given some sequence of distributions, SMC propagates samples forward from one distribution to the next according to a sequence of Markov kernels, $K_n$, and corrects for the discrepancy between the proposal and the target distribution by importance sampling [11]. The choice of forward and backward transition kernels is critical in SMC sampler [11]. For example, in [2], the forward transition kernel was chosen

as the Markov chain transition kernel from an adaptive Metropolized independence sampler. Due to the efficiency of the adaptive Metropolized independence sampler, the resulting optimization algorithm performs well for combination optimization problems. However, heuristic optimization techniques are not easy to incorporate into the SMC sampler, since it is generic too sophisticated to formulate a heuristic optimization method as the forward and/or backward transition kernel.

In this paper, we provide an alternative way to incorporate heuristic optimization techniques, particularly the population based optimization methods, into the particle filter optimization method. The basic idea is to utilize the desirable tracking ability of particle filter to track the movement of the individuals in the population based optimization algorithms. The location of the global optima is treated as the observation of the dynamic system. Therefore, by treating the location of global optima as the destination, the individuals in the population will move towards it sequentially. Moreover, the move strategy can be very heuristic and efficient since it comes from the excellent population based optimization algorithms.

So the PFO algorithm can be designed as follows: the move strategy in the population based optimization algorithms is reformatted as a system dynamic function. The second step is the definition of observation function. Take the minimization problem $x^* = \arg\min_{x\in\mathcal{X}} g(x)$ as an example. The 'best' observation is of course the exact location of the global minimum of the problem of interest, but it is unknown. If the value of the objective function at the global minimum point is known, i.e. $g(x^*)$, then it can be served as the observation. Therefore we can define the observation function (measurement likelihood) as follows

$$L(x; g(x^*)) = \exp\left\{\frac{-[g(x) - g(x^*)]^2}{\tau}\right\}, \tag{10}$$

where $\tau$ is a properly chosen temperature in the Boltman distribution. However, in plenty of problems, the value of $g(x^*)$ is unknown. In such cases, if we can guess a value $g^*$ which is less than the value of $g(x^*)$, we can define the observation function as,

$$L(x; g^*) = \exp\left\{\frac{-[g(x) - g^*]}{\tau}\right\}. \tag{11}$$

If we can not even guess such a value, we can also use an observation function as,

$$L(x; a, b) = \exp\left\{\frac{-a * [g(x) - b]}{\tau}\right\}, \tag{12}$$

where $a$ and $b$ are two constant values, which are properly chosen to make the value of $L(x; a, b)$ reasonable (not extremely large or small).

When the system dynamic and observation in the state space model have been defined, the particle filter is then applied to simulate this model and the particles will move toward the global optima sequentially.

## 4 Particle Filter Optimization with Particle Swarm Movement

In this section, we will introduce the details of particle filter optimization algorithm. The system dynamic function in the state space model is

$$z_n \triangleq \begin{bmatrix} x_n \\ v_n \end{bmatrix} = \begin{bmatrix} x_{n-1} + wv_{n-1} + \phi_1(x_{ibest} - x_{n-1}) + \phi_2(x_{gbest} - x_{n-1}) \\ wv_{n-1} + \phi_1(x_{ibest} - x_{n-1}) + \phi_2(x_{gbest} - x_{n-1}) \end{bmatrix} + \begin{bmatrix} \epsilon_x \\ \epsilon_v \end{bmatrix} \tag{13}$$

where $\epsilon_x \sim N(0, \Sigma_x)$, $\varepsilon_v \sim N(0, \Sigma_v)$, and $\phi_i \sim U(0, 2.05)$ $(i = 1, 2)$. The observation function can be defined as (10), (11), (12) depending on the concrete problem, here denoted as $L(x; \cdot)$. Description of the particle filter optimization algorithm is presented as follows:

**Step 1:** At iteration $n = 1$, sample $N$ particles $\left\{z_n^{(i)}\right\}_{i=1}^{N} \sim U(z; \theta)$ $(i = 1, ..., N)$ according to an uniform distribution with a predefined parameter $\theta$ (i.e. the parameter to define the feasible solution space of the problem), and compute $w_1^{(i)} \propto L(x_1^{(i)}; \cdot)$.

**Step 2:** Evolve particles $\left\{z_n^{(i)}\right\}_{i=1}^{N}$ according to the dynamic function (13).

**Step 3:** Calculate the important weights, $w_n^{(i)} \propto w_{n-1}^{(i)} L\left(x_n^{(i)}; \cdot\right)$.

**Step 4:** Resample the particle representation $\left\{w_n^{(i)}, z_n^{(i)}\right\}$.

**Step 5:** Update the location $x_{ibest}$ and $x_{gbest}$.

**Step 6:** If the stopping criterion is satisfied, then stop; otherwise, set $n := n + 1$ and go back to step 2.

It is worth noting that in the PFO algorithm, compared with the PSO algorithm, the additional computation is the resampling step, which has the complexity $O(N)$ in the number of particles $(N)$. This additional computation complexity is significantly less than the one in KSwarm whose complexity is about $O(Nd^3)$. Moreover, the number of iterations required by the PFO algorithm is much smaller than those of the PSO and KSwarm algorithm.

## 5 Experiments

PFO will be compared with the PSO and KSwarm algorithm via the following four benchmark problems: Sphere, DejongF4, Rosenbrock and Griewank. Herein, the first three are unimodal optimization problems, while the last one is multimodal. In all experiments, the dimensionality $d = 30$. The mathematical definitions of four benchmark functions are given as follows:

$$Sphere(x) = \sum_{i=1}^{d} x_i^2, \quad x \in (-50, 50)^d \tag{14}$$

**Table 1.** Final Values Comparison among PSO, KSwarm and PFO

| Function | PSO [7] | KSwarm [7] | PFO |
|---|---|---|---|
| Sphere | 370.041 | 4.723 | **5.548e-6** |
| DejongF4 | 4346.714 | 4.609 | **2.236e-9** |
| Rosenbrock | 2.61e7 | 3.28e3 | **3.16e-2** |
| Griewank | 13.865 | 0.996 | **2.845e-5** |

$$DeJongF4(x) = \sum_{i=1}^{d} ix_i^4, \quad x \in (-20, 20)^d \tag{15}$$

$$Rosenbrock(x) = \sum_{i=1}^{d-1} \left[100\left(x_{i+1} - x_i^2\right)^2 + (x_i - 1)^2\right], \quad x \in (-100, 100)^d \tag{16}$$

$$Griewank(x) = \frac{1}{4000}\sum_{i=1}^{d} x_i^2 - \prod_{i=1}^{d} \cos\left(\frac{x_i}{\sqrt{i}}\right) + 1, \quad x \in (-600, 600)^d \tag{17}$$

In all experiments, the particle size is set to 100. The stopping criterion is that the algorithm reaches its maximum iteration number 200. Then, the number of objective function evaluations is the same as it is in [7], which makes the comparison reasonable. We run each experiment 50 times for 200 iterations, and averaged the results to account for stochastic differences. The inertia weight $w$ is 0.5 in all the experiments. The variance of system noise is given as $\Sigma_x = \gamma I_d$, $\Sigma_v = \gamma I_d$, where the scalar $\gamma$ indicates the magnitude of the variance in each dimension, specifically, $\gamma = 0.001$ when iteration number $n \leq 100$ and $\gamma = 0.00001$ when $n > 100$ for all experiments. A large variance enables the algorithm to explore the space quickly while a small variance enables the algorithm to improve the final fitness. In design of the observation function, we assumed that we have only 'weak' prior knowledge of the value of the optimal objective function $g(x^*)$. For instance, in all the experiment, we used the observation function (12) by simply setting $a = 1$ and $b = 0$. The temperature in the Boltzmann distribution $\tau$ is set by experience. Herein, in all the experiments, $\tau = 10$.

Table 1 shows the final values of basic PSO and KSwarm algorithms after 1000 iterations, as well as the final values of PFO after 200 iteration. It can be seen that the values obtained by PFO are several orders of magnitude better than the basic PSO and KSwarm algorithms in all four benchmark problems. Afterwards, Figs. 1 - 4 pictorially depict the best value versus the number of iterations. Herein, in order to compare the PFO, PSO and KSarm under the same complexity condition, the number of iterations for PFO is set to 200, while the number of iterations for PSO and KSwarm is set to 1000. Although the number of iterations is different, the number of time of calculating the objective function is same. Through simulations, we observe that, compared with the PSO algorithm, the additional computation efforts of PFO is negligible, while KSwarm take significant additional computation due to its matrix operations.

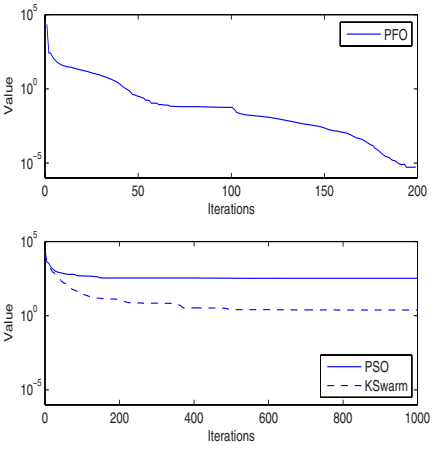


**Fig. 1.** Sphere

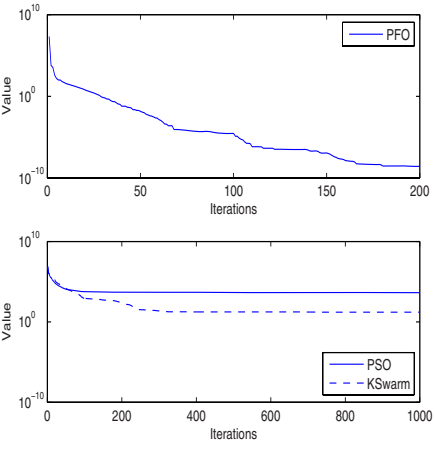


**Fig. 2.** Dejong

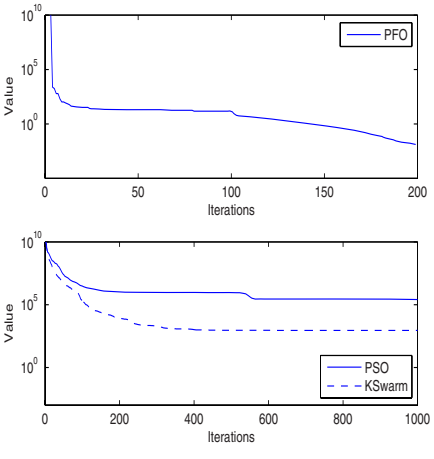


**Fig. 3.** Rosenbrock

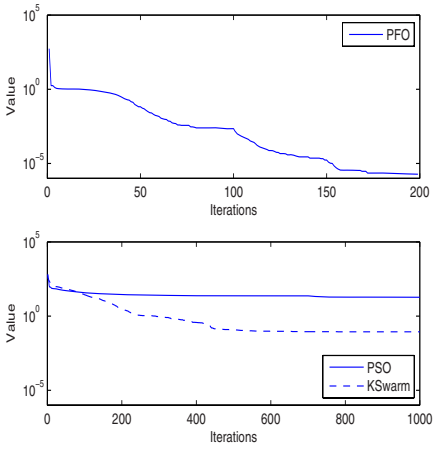


**Fig. 4.** Griewank

Moreover, the plots of the best value obtained per iteration show that the PFO tends to find good solutions faster than the other two methods. These simulation results demonstrate a significant improvement for the PSO, not only in exploring final solutions, but also in the speed to find them.

## 6 Conclusions

In this paper, we propose a novel generalized framework for stochastic optimization in which the PFO is incorporated with the swarm move method in PSO. The particle swarm move in PSO algorithm is treated as the system dynamic

in the state space model, while the objective function in optimization problem is designed as the observation model. Particle filter method is then applied to track the dynamic movement of the particle swarm in PSO algorithm. By incorporating the state space probability modelling and resample strategy into the PSO algorithm, the PFO can potentially enhance the ability of PSO algorithm in two aspects: making it easier to jump out local optima and refining the final result. Compared with the PSO and KSwarm algorithm, the PFO algorithm can obtain better final fitness with a negligible additional computation effort. Finally, simulation results demonstrate a significant improvement for the PFO, not only in exploring final solutions, but also in the speed to find them.

## Acknowledgment

The work by Shengxiang Yang was supported by the Engineering and Physical Sciences Research Council (EPSRC) of UK under Grant EP/E060722/1.

## References

1. Doucet, A., De Freitas, J., Gordon, N. (eds.): Sequential Monte Carlo Methods in Practice. Springer, Heidelberg (2001)
2. Zhang, Y., Ji, C., Walik, W.Q., Liu, Y., O'Brien, D.C., Edwards, D.J.: Joint Antenna and User Selection Algorithm for Uplink of Multiuser MIMO Systems Using the Sequential Monte Carlo Optimization. In: IEEE Workshop on Statistical Signal Processing, SSP 2007, Madison, Wisconsin, pp. 493–496. IEEE Press, Los Alamitos (2007)
3. Shi, Y., Eberhart, R.: Empirical Study of Particle Swarm Optimization. In: IEEE Congress on Evolutionary Computation, CEC 1999, pp. 1945–1950. IEEE Press, Piscataway (1999)
4. Riget, J., Vesterstroem, J.S.: A diversity-guided Particle Swarm Optimizer - the ARPSO. Department of Computer Science, University of Aarhus, Technique Report (2002)
5. Angeline, P.J.: Using Selection to Improve Particle Swarm Optimization. In: IEEE Congress on Evolutionary Computation, CEC 1998, Anchorage, Alaska, USA, pp. 84–89. IEEE Press, Los Alamitos (1998)
6. Clerc, M., Kennedy, J.: The particle Swarm-explosion, Stability, and Convergence in a Multidimensional Complex Space. IEEE Transactions on Evolutionary Computation 6(1), 58–73 (2002)
7. Monson, C.K., Seppi, K.D.: The Kalman Swarm. In: Deb, K., et al. (eds.) GECCO 2004. LNCS, vol. 3102, pp. 140–150. Springer, Heidelberg (2004)
8. Gordon, N., Salmond, D., Smith, A.: Novel Approach to Nonlinear/ Non-gaussian Bayesian State Estimation. IEE Proceedings Radar and Signal Processing 140(2), 107–113 (1993)
9. Harvey, A.C.: Forecasting, Structural Time Series Models and the Kalman Filter. Cambridge University Press, Cambridge (1990)
10. Doucet, A., Godsill, S., Andrieu, C.: On Sequential Monte Carlo Sampling Methods for Bayesian Filtering. Statistics and Computing 10, 197–208 (2000)
11. Moral, P.D., Doucet, A., Jasra, A.: Sequential Monte Carlo Samplers. Royal Statistical Society 68, 411–436 (2006)