Improving on the Kalman Swarm

Extracting Its Essential Characteristics

Christopher K. Monson and Kevin D. Seppi

Brigham Young University, Provo UT 84602, USA
{c,kseppi}@cs.byu.edu

Abstract. The Kalman Swarm (KSwarm) is a new approach to particle motion in PSO that reduces the number of iterations required to reach good solutions [1]. Unfortunately, it has much higher computational complexity than basic PSO. This paper addresses the runtime of KSwarm in a new algorithm called "Linear Kalman Swarm" (LinkSwarm) which has linear complexity and performs even better than KSwarm. Some possible reasons for the success of KSwarm are also explored.

1 Introduction

The Kalman Swarm (KSwarm) [1] is an adaptation of standard PSO [2] that uses the Kalman Filter [3] to calculate the next position of each particle. The best location in its neighborhood (which always includes the particle itself) is used as an observation at each time step, producing a new location through prediction. This approach has been shown to produce better solutions in fewer iterations than standard PSO in a variety of test functions.

KSwarm, however, is not without liabilities. The dimensional complexity of running the algorithm is $O(d^3)$, which is much higher than the $O(d)$ complexity of basic PSO. Furthermore, KSwarm has a number of input parameters that are difficult to tune.

This paper presents the new algorithm "Linear Kalman Swarm" (LinkSwarm), which is an approximation to KSwarm with linear complexity that obtains better performance. The algorithm is developed by analyzing and extracting the essential characteristics of KSwarm. Through its development, some ideas are generated that may help explain the good performance of KSwarm. Results that compare the performance of the two algorithms are given on various test functions.

2 KSwarm Speed

Although KSwarm can often get better results in fewer iterations than basic PSO, each iteration is far more expensive. Because the complexity of KSwarm is $O(d^3)$, this significantly increases the running time of the algorithm and is especially a problem when optimizing high-dimensional functions.

We will therefore attempt to extract the essential characteristics of the KSwarm algorithm in order to provide a fast approximation. The Kalman Filter is the basis for the update equations, and these will be analyzed first. Recall that $\mathbf{F}$ and $\mathbf{H}$ are the transition and sensor characteristic matrices with $\mathbf{V_x}$ and $\mathbf{V_z}$ as their respective covariance matrices, $\mathbf{m}_t$ and $\mathbf{V}_t$ are the mean and covariance parameters of the filtered state at time t, and $\mathbf{K}_t$ is the "Kalman Gain" at time t. These parameters and the following update equations comprise the multivariate Kalman Filter:

$$\mathbf{m}_{t+1} = \mathbf{F}\mathbf{m}_t + \mathbf{K}_{t+1}(\mathbf{z}_{t+1} - \mathbf{H}\mathbf{F}\mathbf{m}_t) \tag{1}$$

$$\mathbf{V}_{t+1} = (\mathbf{I} - \mathbf{K}_{t+1})(\mathbf{F}\mathbf{V}_t\mathbf{F}^\top + \mathbf{V_x}) \tag{2}$$

$$\mathbf{K}_{t+1} = (\mathbf{F}\mathbf{V}_t\mathbf{F}^\top + \mathbf{V_x})\mathbf{H}^\top\left(\mathbf{H}(\mathbf{F}\mathbf{V}_t\mathbf{F}^\top + \mathbf{V_x})\mathbf{H}^\top + \mathbf{V_z}\right)^{-1} \quad . \tag{3}$$

The complexity of KSwarm is due almost entirely to complex matrix operations, like multiplication and inversion. This complexity is present in the update equations above and in sampling from a multivariate Normal distribution. Any approximation intended to speed up the algorithm should address these two key areas.

2.1 Weighted Vectors

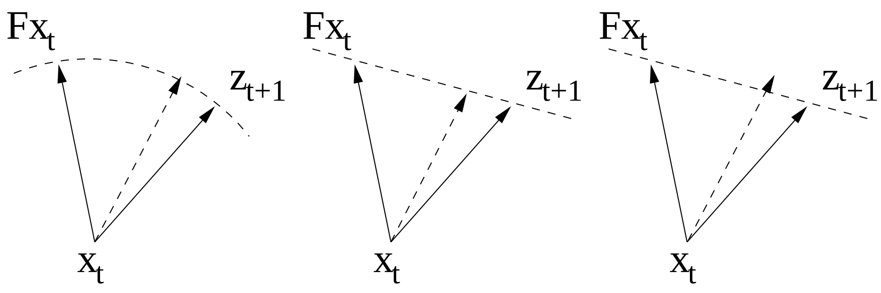
Because each particle knows with certainty where its neighbors are, we may assume that the system has perfect sensors ($\mathbf{H} = \mathbf{I}$), and may rearrange (1) thus:

$$\begin{aligned} \mathbf{m}_{t+1} &= \mathbf{F}\mathbf{m}_t + \mathbf{K}_{t+1}(\mathbf{z}_{t+1} - \mathbf{F}\mathbf{m}_t) \\ &= \mathbf{F}\mathbf{m}_t + \mathbf{K}_{t+1}\mathbf{z}_{t+1} - \mathbf{K}_{t+1}\mathbf{F}\mathbf{m}_t \\ &= (\mathbf{I} - \mathbf{K}_{t+1})\mathbf{F}\mathbf{m}_t + \mathbf{K}_{t+1}\mathbf{z}_{t+1} \quad . \end{aligned} \tag{4}$$

This makes the filtered state look like a convex combination (assuming, of course, that $\mathbf{K}_{t+1}$ has the appropriate properties) of the predicted next state $\mathbf{F}\mathbf{m}_t$ and the observed state $\mathbf{z}_{t+1}$. It seems reasonable to assume that this is an essential characteristic of the Kalman Filter: it balances observations and myopic predictions. If this is indeed a governing principle of a Kalman Filter, then it should be possible to construct a useful approximation of that behavior.

Since the Kalman Filter is basically balancing between myopic predictions and neighborhood observations using the linear operator $\mathbf{K}$, we may view this as generating a vector through rotation and scaling of one of the two other vectors. This operation may be done by rotating one of the vectors by a specified angle through the plane defined by both vectors (a matrix operation that we would like to avoid), or it may be approximated by taking a weighted average of the normalized vectors and performing some post scaling. The two approaches are depicted in Figure 1.

As the weighted average is a much cheaper computation than rotation, this is what is done. Though it does not produce precisely the same results when



(a) Vectors weighted through rotation

(b) Approximation using weighted average of each coordinate

(c) Rescaled result

Fig. 1. Weighting two vectors to generate a third

the new vector is very close to one of the original vectors, it does provide a fast and useful approximation. To make the weighted average work correctly, we first normalize the original vectors, take a weighted sum to generate a new vector, normalize it, and then scale it so that its length is a weighted sum of the original lengths.

More formally, if we have a weight scalar $\alpha \in [0, 1]$ and vectors $\mathbf{u_1}$ and $\mathbf{u_2}$, we may generate a new vector $\mathbf{v}$ thus:

$$\widehat{\mathbf{v}} = \alpha \frac{\mathbf{u_1}}{\|\mathbf{u_1}\|} + (1 - \alpha) \frac{\mathbf{u_2}}{\|\mathbf{u_2}\|} \tag{5}$$

$$\mathbf{v} = \frac{\widehat{\mathbf{v}}}{\|\widehat{\mathbf{v}}\|} \left(\alpha \|\mathbf{u_1}\| + (1 - \alpha) \|\mathbf{u_2}\| \right) \quad . \tag{6}$$

Care must be taken when $\mathbf{u_1}$ and $\mathbf{u_2}$ point in opposite directions. In this case we have two options: point the new vector in some direction orthogonal to one of the vectors (of which there are two if $d = 2$ and infinite if $d > 2$) or pick some other direction. In practice this is solved by simply picking one of the original vectors as the new vector, usually that which has a better associated value. This case is easily detected by calculating $\widehat{\mathbf{v}}$ using (5) and then noting that $\|\widehat{\mathbf{v}}\|$ is extremely small.

The behavior of KSwarm is approximated using this approach to calculate a new velocity for a particle. The particle's current velocity is balanced against the relative best position from its neighborhood. The new velocity is some vector between the two, based on α.

We were previously using $\mathbf{m}_t$ as the current position and velocity of the particle, but now we compute only the velocity using the approximation outlined above. We add the velocity to our current position to get a new position. Therefore, there is no longer any need to compute $\mathbf{m}_t$ nor is there any use for $\mathbf{F}$, as the prediction of our next position is trivially calculated using the current velocity.

It is worth noting that though this is beginning to sound like the original PSO algorithm, it is not (see Section 2.4).

2.2 Sampling From the Normal

So far, the approximation contains no randomness at all, effectively removing the complexity introduced from generating samples from a multivariate Normal. Randomness, however, is an essential part of the original algorithm, so some sampling should be done to determine the final velocity of the particle.

KSwarm particles each sampled their final state from a multivariate Normal with parameters $\mathbf{Fm}_t$ and $\mathbf{V}_t$. They were provided with diagonal matrices for $\mathbf{V}_{t=0}$, $\mathbf{V}_\mathbf{x}$, and $\mathbf{V}_\mathbf{z}$. If $\mathbf{V}_t$ were to remain diagonal, we could optimize the multivariate sample (which involves matrix operations) with d univariate samples.

Therefore, instead of a covariance matrix, we select a variance vector $\boldsymbol{\sigma^2}$ for our approximate algorithm. We further assume that this vector is constant (unlike $\mathbf{V}_t$ in the Kalman Filter). Had $\mathbf{V}_t$ been constant in the Kalman Filter, the gain $\mathbf{K}_t$ would have also been constant, as $\mathbf{V}_t$ was the only time-dependent portion of (3). This assumption of a constant $\boldsymbol{\sigma^2}$ allows us to further assume that α is constant.

To generate randomness, the approximate algorithm uses each component of the normalized final velocity vector $\frac{\hat{\mathbf{v}}}{\|\hat{\mathbf{v}}\|}$ as the mean of a distribution. The variance is given by the corresponding component of $\boldsymbol{\sigma^2}$. Since we are dealing with a normalized mean, it is not unreasonable to assume that $\boldsymbol{\sigma^2} = \sigma^2\mathbf{e}$, where σ^2 is a constant scalar (and $\mathbf{e}$ is a vector of all ones). This effectively reduces the number of tunable parameters and simplifies the algorithm even further.

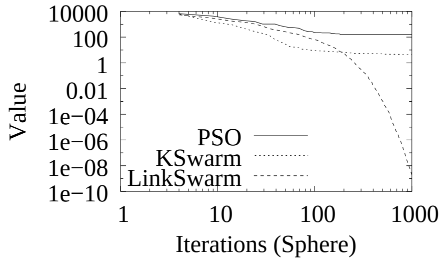
2.3 Linear Kalman Swarm (LinkSwarm)

The two fundamental approximations above complete the development of LinkSwarm, an approximation to KSwarm. The full LinkSwarm algorithm for particle motion is summarized in Table 1.

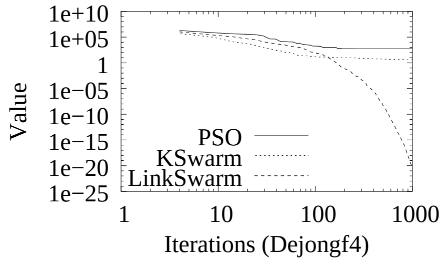
The performance and runtimes of LinkSwarm and KSwarm are shown in Figure 2. For all experiments, $d = 30$, $\sigma^2 = 0.6$, $\alpha = 0.45$, the sociometry is "star", and 20 particles are used. Also, KSwarm uses a prior based on initial particle positions.

It is interesting that LinkSwarm, which runs roughly 10 times faster than the original KSwarm in 30 dimensions, also performs better on every test function. Figure 2(f) also shows that the approximate algorithm is only very slightly slower than basic PSO (because of magnitude calculations and multiple Normal samples) but, in contrast with KSwarm, has the same O(d) computational complexity.

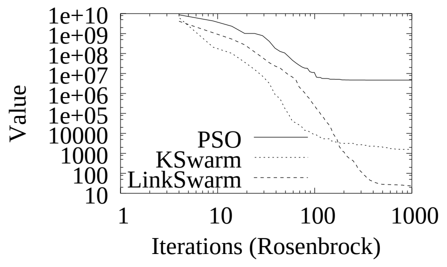
It appears that we have not only maintained the essential characteristics of KSwarm, but that we have improved on it in the process.



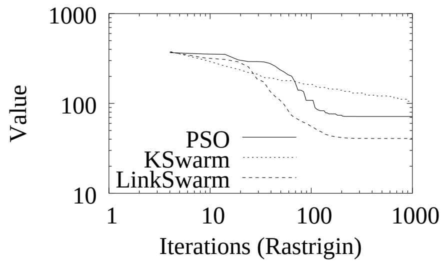
(a) Sphere



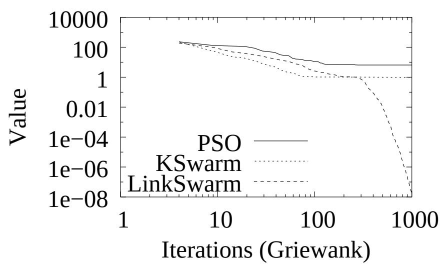
(b) DejongF4



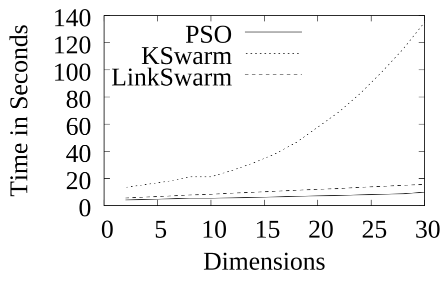
(c) Rosenbrock



(d) Rastrigin



(e) Griewank



(f) Average Runtimes on Sphere

Fig. 2. Performance of LinkSwarm as compared with original KSwarm

Table 1. The LinkSwarm Algorithm

Given a particle's current velocity $\mathbf{v_t}$ and the *relative* position of the neighborhood best $\mathbf{v}_{\text{best}}$ (the particle is always part of its own neighborhood in LinkSwarm), generate a new normal velocity vector $\mathbf{v}'$ as follows (norm generates a normalized vector):

$$\mathbf{v}' = \text{norm}\left(\alpha \frac{\mathbf{v_t}}{\|\mathbf{v_t}\|} + (1-\alpha)\frac{\mathbf{v}_{\text{best}}}{\|\mathbf{v}_{\text{best}}\|}\right) \ . \tag{7}$$

For each coordinate v_i' of $\mathbf{v}'$, generate $\hat{\mathbf{v}}$ by drawing samples from a Normal distribution:

$$\hat{v_i} \sim \text{Normal}(v_i', \sigma^2) \ . \tag{8}$$

Normalize $\hat{\mathbf{v}}$ and scale to a weighted sum of the lengths of $\mathbf{v_t}$ and $\mathbf{v}_{\text{best}}$ to create $\mathbf{v_{t+1}}$:

$$\mathbf{v_{t+1}} = (\alpha\|\mathbf{v_t}\| + (1-\alpha)\|\mathbf{v}_{\text{best}}\|)\,\text{norm}(\hat{\mathbf{v}}) \ . \tag{9}$$

Add this velocity to the current position $\mathbf{x_t}$ to get $\mathbf{x_{t+1}}$.

2.4 LinkSwarm vs. PSO

With all of the approximations made, the algorithm has been reduced to (roughly) a weighted sum of two vectors. Arguably, PSO does the same thing. The question arises, then, as to whether we really do anything different from basic PSO, besides sampling from a Normal distribution to get the final result. The answer is "yes".

Basic PSO and many of its variants also take a "weighted sum" of two vectors in order to generate a new velocity. There are two very important differences, however, between LinkSwarm and basic PSO: the vectors involved in the weighting operation are different, and the velocity is *set* to a new value, not *augmented* by it. In other words, LinkSwarm generates a velocity, not an acceleration.

The fact that LinkSwarm sets rather than augments its current velocity neatly sidesteps the issue of velocity explosion. Of more interest, however, is the fact that it uses different vectors in its decision process. In basic PSO, the two vectors involved in the weighting operation are the relative position of the particle's personal best and the relative position of the neighborhood best. LinkSwarm uses a particle's *current velocity* (which may also be viewed as the relative predicted position at the next time step) instead of the relative position of its personal best. It only uses its personal best in the context of its neighborhood.

This basic difference between KSwarm and PSO is explored further in Section 4 as part of the explanation for its success.

3 KSwarm Deficiencies

When compared with the basic PSO algorithm, KSwarm performs very well, reducing the number of iterations required to reach good solutions and improving on the overall solutions on a number of test functions. This is an interesting result, as it does so by altering the motion characteristics of particles. Fundamental changes in motion have not received much attention in recent research because motion has been considered to be less important than the social and diversity aspects of the algorithm [4–10, 2, 11–15]. KSwarm represents a possible counterexample to that often implicit assumption.

One may ask why Kalman Filter motion works well in this context. There is one rather unflattering reason that it works especially well on the test functions: the original KSwarm is subtly origin-seeking. The Kalman Filter requires a prior distribution on the particle's state. Recall that the state of a particle has been defined to be the position and velocity of that particle [1]. Absent a reasonable prior, this was always set to $\mathbf{0}$, indicating that the particle began its life with the assumption that it was at the origin and that it was not moving. In other words, it assumed it was *already at the optimum*, a deficiency addressed here.

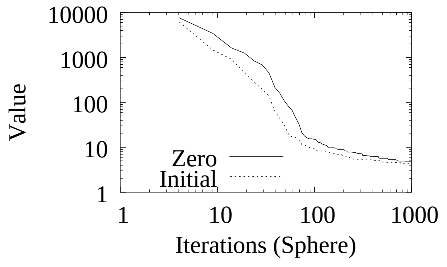
A change was made to KSwarm, setting the prior of each particle to its initial position instead of $\mathbf{0}$. The graphs in Figure 3 compare the two approaches. It is clear that especially for strongly multi-modal functions like Rastrigin, the prior has a significant impact on the algorithm's behavior.

While this change significantly degrades the performance of KSwarm on Rastrigin, it leaves it practically unchanged on the other test functions, most of which are unimodal. Griewank is of particular interest because it is multimodal but the performance did not degrade for that function. Some attempt to explain this behavior is given in the next section.

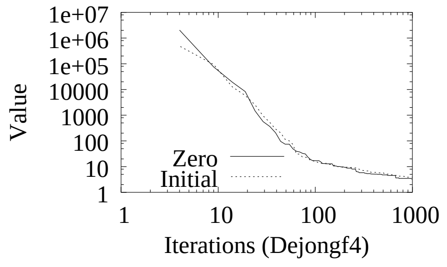
4 Why It Works

It is first important to acknowledge that the basic PSO algorithm has had many changes applied in recent years that substantially improve its performance. These methods are not considered here due to space and time constraints, but it is worth noting that LinkSwarm outperforms many of them where KSwarm did not. The results of those experiments are reserved for a later, more complete paper.

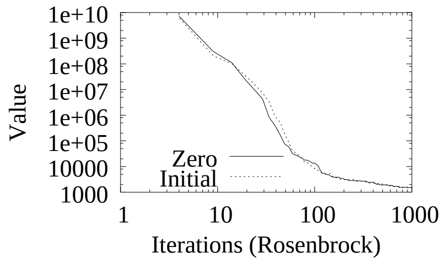
The development of LinkSwarm served to expose some interesting characteristics of the original KSwarm. The most striking of these has already been mentioned, and is best understood in contrast to basic PSO: the vectors involved in the decision process are different. While basic PSO particles use personal and neighborhood bests to make their decisions, LinkSwarm particles throw their personal best in with their neighbors and decide between the neighborhood best and their current trajectory. This difference represents an interesting and useful assumption: a particle's current direction is probably pretty good.



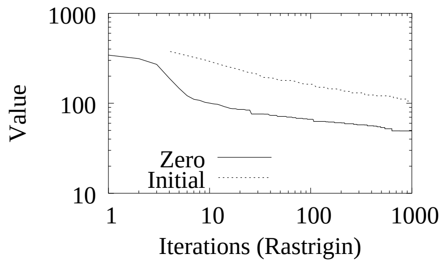
(a) Sphere



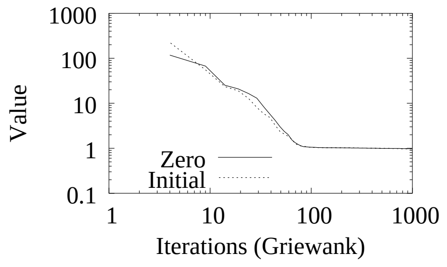
(b) DejongF4



(c) Rosenbrock



(d) Rastrigin



(e) Griewank

Fig. 3. Results of KSwarm with a zero prior and with a more reasonable prior based on initial particle positions

The validity of this assumption depends on the kinds of functions that are being optimized. It is motivated by the fact that the particle was already approaching a good area to begin with, so there is no reason that it should have to stop and turn around just because one of its neighbors wants it to see something; it may be onto something good already.

This assumption works especially well with functions that, though potentially multimodal, have an overall trend. Functions like Rastrigin and Griewank fit this criterion quite nicely. Though they have many local minima, they each exhibit an overall downward trend toward the global minimum. A particle that is able to filter out the "noise" of the local minima in favor of discovering the overall trend is in general going to do well with these types of functions. The Kalman Filter was designed to do precisely that. These particles are expected to do very well on functions where sufficient "blurring" will produce a unimodal function.

5 Conclusions and Future Research

An enormous number of things have yet to be tried. Currently the weight parameter α is constant, but it could be changed dynamically, perhaps based on some confidence parameter that is learned over time. This needs to be explored more fully. The effect of the various parameters in general is not currently known, and that is also begging for more exploration.

These algorithms need to be tested against the state of the art. The basic PSO algorithm is known to be naive and suboptimal, and much research has been done to improve it. In order to be truly interesting, LinkSwarm and its variants must be compared with the best PSO algorithms known to date. Preliminary experiments show that it compares very favorably with recent PSO improvements.

Surprisingly, the approximate algorithm performed better than the original. The reasons for this are unknown, though we have some preliminary ideas. The parameters used to initialize KSwarm might not have been optimal, causing it to perform somewhat more poorly than it might otherwise have done. While this could potentially be explored to good effect, there are simply too many parameters to make such exploration feasible. The number of tunable parameters in KSwarm was one of its major shortcomings. The LinkSwarm algorithm not only has fewer and more intuitive parameters, but it is faster and works better.

LinkSwarm is a major improvement over KSwarm, not only in computational complexity, but also in the results obtained. It represents the elimination of numerous tunable parameters and is much simpler to code and to understand. Along with its improved speed and simplicity, preliminary work suggests that it does better than many recent improvements to PSO, potentially making it a strong contender in the field of swarm optimization.

References

1. Monson, C.K., Seppi, K.D.: The Kalman swarm. In: Proceedings of the Genetic and Evolutionary Computation Conference (GECCO), Seattle, Washington (2004)

(To Appear)

2. Kennedy, J.: Small worlds and mega-minds: Effects of neighborhood topology on particle swarm performance. In Angeline, P.J., Michalewicz, Z., Schoenauer, M., Yao, X., Zalzala, Z., eds.: Proceedings of the Congress of Evolutionary Computation. Volume 3., IEEE Press (1999) 1931–1938
3. Kalman, R.E.: A new approach to linear filtering and prediction problems. Transactions of the ASME–Journal of Basic Engineering **82** (1960) 35–45
4. Kennedy, J.: Bare bones particle swarms. In: Proceedings of the IEEE Swarm Intelligence Symposium 2003 (SIS 2003), Indianapolis, Indiana (2003) 80–87
5. Kennedy, J., Mendes, R.: Population structure and particle swarm performance. In: Proceedings of the Congress on Evolutionary Computation (CEC 2002), Honolulu, Hawaii (2002)
6. Vesterstrøm, J.S., Riget, J.: A diversity-guided particle swarm optimizer – the ARPSO. Technical Report 2002-02, Departemnt of Computer Science, University of Aarhus, EVALife (2002)
7. Richards, M., Ventura, D.: Dynamic sociometry in particle swarm optimization. In: International Conference on Computational Intelligence and Natural Computing. (2003)
8. Mendes, R., Kennedy, J., Neves, J.: Watch thy neighbor or how the swarm can learn from its environment. In: Proceedings of the IEEE Swarm Intelligence Symposium 2003 (SIS 2003), Indianapolis, Indiana (2003) 88–94
9. Kennedy, J., Mendes, R.: Neighborhood topologies in fully-informed and best-of-neighborhood particle swarms. In: Proceedings of the 2003 IEEE SMC Workshop on Soft Computing in Industrial Applications (SMCia03), Binghamton, New York, IEEE Computer Society (2003)
10. Angeline, P.J.: Using selection to improve particle swarm optimization. In: Proceedings of the IEEE Congress on Evolutionary Computation (CEC 1998), Anchorage, Alaska (1998)
11. Kennedy, J.: Stereotyping: Improving particle swarm performance with cluster analysis. In: Proceedings of the IEEE Congress on Evolutionary Computation (CEC 2000), San Diego, California (2000) 1507–1512
12. Vesterstroem, J.S., Riget, J., Krink, T.: Division of labor in particle swarm optimisation. In: Proceedings of the IEEE Congress on Evolutionary Computation (CEC 2002), Honolulu, Hawaii (2002)
13. Shi, Y., Eberhart, R.C.: Empirical study of particle swarm optimization. In: Proceedings of the IEEE Congress on Evolutionary Computation (CEC 1999), Piscataway, New Jersey (1999) 1945–1950
14. Krink, T., Vestertroem, J.S., Riget, J.: Particle swarm optimisation with spatial particle extension. In: Proceedings of the IEEE Congress on Evolutionary Computation (CEC 2002), Honolulu, Hawaii (2002)
15. Shi, Y., Eberhart, R.C.: Parameter selection in particle swarm optimization. In: Evolutionary Programming VII: Proceedings of the Seventh Annual Conference on Evolutionary Programming, New York (1998) 591–600