Chapter 1

Introduction

1.1 Global optimization

Increasing prominence is given to the role of optimization in engineering, both in academia and in industry. Computational power, combined with increased algorithm flexibility and simplicity, allows for a faster transition of algorithm development in academia to industry.

In convex optimization, the local minimizer is characterized by the Karush-Kuhn-Tucker conditions [1]. These conditions are necessary and *sufficient* to guarantee optimality.

In global optimization in general, no conditions are available to characterize the global optimizer, and the optimization problem is intractable.

The difficulty of the general global optimization problem is further aggravated by

1. the presence of numerical noise,
2. the presence of infeasible regions in the design domain,
3. the presence of discontinuities,
4. a large number of design variables, and
5. the computational cost of evaluating an expensive objective function (simulation).

The large variety of solution techniques in global optimization is therefore not surprising. Recent developments includes evolutionary computational algorithms, taboo searches, fractional programming, dynamical searches, etc. [2].

This study is concerned with the evolutionary particle swarm optimization algorithm (PSOA), introduced by Kennedy and Eberhart [3, 4]. Some advantages of the PSOA are that it is

1. gradient-free,
2. easy to parallelize, and
3. simple and easy to implement.

1.2 Motivation

PSOA investigations are predominantly based on empirical test functions [5, 6, 7, 8, 9]. In these investigations, the complexities of the algorithm are not broken into digestible units. An understanding of the fundamentals of the algorithm is therefore not easily achieved. In these studies, judgment is largely passed based purely on the final objective function values that a particular algorithm obtains.

Some researchers conduct empirical investigations to quantify the sequence effect of the algorithm by observing the particle positions in the design domain at selected iterations for a single run [10]. Alternatively, the trace of positions of the swarm over a single run is observed [11, 12]. Although some insight is gained into the behavior and convergence of the swarm, the stochastic nature of the algorithm is neglected.

Ozcan and Mohan [13] conducted a deterministic trajectory analysis of a single particle by neglecting the stochasticity of the algorithm. Clerc and Kennedy [14] extended the analysis to the stochastic dynamic behavior of an individual particle in complex space, with the main focus on constriction coefficients and achieving desirable dynamic characteristics for a particle. Trelea [15] averaged the stochasticity in the PSOA, in an investigation based on dynamic system theory, to increase the understanding of the dynamics of an individual particle. Zheng et al. [16] extended the study of Ozcan and Mohan [13] and concluded that the inertia weight should increase over time.

Although extensive research on the PSOA has been conducted, a basic understanding of the algorithm still seems lacking. It is the aim of this study to gain fundamental insight into the PSOA. It will be shown that there exist two different formulations of the PSOA. These formulations only differ in the formulation of the velocity updating rule.

1.3 Objectives

The four objectives of this study are

1. to discern between the two *different* formulations of the PSOA,
2. to investigate *observer independence* of the PSOA,
3. to introduce a *novel* observer independent PSOA with diverse, space filling particle trajectories, and
4. to utilize the PSOA in *shape optimization*.

1.4 Approach

In attaining each objective, a different approach is followed:

1. In order to discern between the different formulations, elementary linear algebra is used. In order to quantify the differences, a numerical study is conducted.

2. The investigation of observer independence is conducted using Monte Carlo simulations. For observer independence both translational and rotational invariance have to apply. Objectivity is quantified with a numerical study in both the unrotated, and rotated reference frames.

3. Knowledge developed in 1. and 2. above is then used to develop a novel PSOA formulation.

4. In order to utilize the PSOA in shape optimization, an unstructured remeshing strategy is implemented. This allows for large variations in designs, prominent in the initial phases of the PSOA searches.

1.5 Thesis overview

The chapters in this thesis are self contained.

In Chapter 2, the problem formulation under consideration in Chapters 3 and 4 is presented, as well as a very brief overview of the formulation of the PSOA, to allow for a brief historical overview of the PSOA.

In Chapter 3, a detailed analysis of the particle swarm optimization algorithm (PSOA) is presented. It is shown that implementation subtleties due to ambiguous notation have resulted in two distinctly different implementations of the PSOA, which have been used indiscriminately and unwittingly within the optimization community. However, discerning between these two implementations is shown to be of crucial importance. While the behavior of the respective implementations is markedly different, they only differ in the formulation of the velocity updating rule. In fact, the differences are merely due to subtle differences in the introduction of randomness into the algorithm. For a population of p particles, it is shown that for the first implementation, the particle trajectories collapse to p line searches. The second implementation does not suffer this drawback. Instead, diverse stochastic search trajectories are retained. It is then shown that some popular heuristics like maximum velocity limit, position restriction, craziness and high initial velocities are possibly of less importance than originally thought; their greatest contribution is that they prevent the collapse of particle trajectories to lines. Finally, it is emphasized that the determination of optimal values for parameters like inertia, velocity limit, etc. has to be performed within the context of the formulation used. To this extent, a proposed list of parameters and implementational issues that should be reported when 'tuning' the PSOA is given.

In Chapter 4, the ability of the particle swarm optimization algorithm (PSOA) to satisfy objectivity, also called observer independence or frame indifference, is investigated. In Chapter 3 it was shown that implementation subtleties have resulted in two distinctly different implementations of the PSOA. The first implementation is now shown to be observer independent. In turn, the second implementation of the PSOA is shown to suffer from observer dependence. A novel formulation of the PSOA, in which the particle trajectories do not collapse to line searches, while observer independence is preserved, is then introduced. However, the observer independence is only satisfied in a stochastic sense, i.e. the mean objective function value over a large number of runs is independent of the reference frame. Objectivity and effectiveness of the three different formulations are quantified using a popular test set. The objective functions are evaluated in both the unrotated reference frame, and an arbitrary rotated reference frame.

In Chapter 5, the PSOA is combined with an unstructured remeshing shape optimization environment. The remeshing strategy creates unstructured meshes from triangular elements, based on the truss structure analogy proposed by Persson and Strang [17]. The PSOA is then used to search for optimal shapes. Results for a popular beam problem are presented.

Finally, conclusions and recommendations are offered in Chapter 6.

Chapter 2

Problem formulation and background

In this chapter, the problem formulation under consideration in Chapters 3 and 4 is presented, as well as a very brief overview of the formulation of the PSOA, to allow for a brief historical overview of the PSOA.

2.1 Global optimization problem formulation

For the sake of brevity, we restrict ourselves to the unconstrained or bounded constrained multi-modal global optimization problem which we will define as follows:

Find the global minimum value $f(\boldsymbol{x}^*)$ of a given real-valued function $f : D \subseteq \mathbb{R}^n \rightarrow \mathbb{R}$, such that

$$f^* = f(\boldsymbol{x}^*) \leq f(\boldsymbol{x}),\ \forall\, \boldsymbol{x} \in D, \tag{2.1}$$

where D is the allowable (bounded) design domain. Since this problem is intractable, the aim is usually to find a suitably low approximation $\bar{f}$ to f^*.

2.2 Basic formulation of the PSOA

Consider a swarm of p particles in an n-dimensional design space. The position vector $\boldsymbol{x}_k^i$ of each particle i is updated by

$$\boldsymbol{x}_{k+1}^i = \boldsymbol{x}_k^i + \boldsymbol{v}_{k+1}^i, \tag{2.2}$$

where k is a unit pseudo time increment (iteration number). $\boldsymbol{v}_{k+1}^i$ represents the velocity vector that is obtained from the velocity rule, given by

$$\boldsymbol{v}_{k+1}^i = w\boldsymbol{v}_k^i + \boldsymbol{\nu}_k^i, \tag{2.3}$$

where the inertia factor w [8] is a real number, typically between 0.4 and 0.9, and $\boldsymbol{\nu}_k^i$ is the stochastic 'velocity' vector.

In turn, the term $\boldsymbol{\nu}_k^i$ consists of the summation of the terms $c_1(\boldsymbol{p}_k^i - \boldsymbol{x}_k^i)$ and $c_2(\boldsymbol{p}_k^g - \boldsymbol{x}_k^i)$, which are however randomly scaled in a to be specified way. $\boldsymbol{p}_k^i$ represents the best position vector of particle i, while $\boldsymbol{p}_k^g$ represents the global best position vector of the complete swarm, up to time step k. The cognitive and social scaling factors c_1 and c_2 are real numbers; both are usually equal to 2. By selecting $c_1 = c_2$, the cognitive and social contributions are weighed equally.

The vectors $(\boldsymbol{p}_k^i - \boldsymbol{x}_k^i)$ and $(\boldsymbol{p}_k^g - \boldsymbol{x}_k^i)$ utilize the magnitudes and directions from a given particle's current position $\boldsymbol{x}_k^i$ to the particle's best position $\boldsymbol{p}_k^i$, and to the global best position $\boldsymbol{p}_k^g$.

2.3 Brief history of PSOA

The particle swarm optimization algorithm (PSOA) was introduced by Kennedy and Eberhart [3, 4] in 1995. Since then, the PSOA has been applied to optimization problems in a variety of disciplines. To name but a few, neural network training [3, 4, 5, 6], biochemistry [18], manufacturing [10], electromagnetism [11, 19, 20], electrical power [21, 22], optics [12] and structural optimization [23, 24].

The initial implementations of the PSOA contained two parameters, namely the maximum velocity limit [4] and the acceleration constant. The acceleration constant consists of two parts referred to as the cognitive and social constants c_1 and c_2 [25]. The cognitive and social constants are usually implemented statically, although dynamic implementations have also been studied [26]. Kennedy and Eberhart [3] initially proposed that $c_1 = c_2 = 2$.

Shi and Eberhart [8] introduced an additional parameter, referred to as the inertia weight w. Static and dynamic implementations of the inertia weight exist. Three dynamic inertia implementations are the frequently encountered reducing inertia [8, 26, 27], increasing inertia [16] and unstructured inertial adjustment [28]. Recently, Clerc [14, 29] introduced a new formulation, referred to as constriction.

Particle position limits can be implemented to ensure that the particle searches are confined to the defined design domain. The limits are usually implemented as constraints on each design variable. There are various strategies to accommodate the boundary constraints or to enforce position limitation. A complete discussion is given by Robinson and Rahmat-Samii [30].

The evaluation of the objective function value at a given iteration can occur either synchronously or asynchronously [31]. In the synchronous update method, the swarm's positions are updated before the objective function evaluations occur at the updated positions. Hence the particle best and global best positions can only be updated at the end of each iteration. In contrast, in the asynchronous implementation, the objective function is evaluated directly after a position update for a particle occurs. Therefore, the updates of the particle best position and the global best position occur after each particle updates its position. In the synchronous implementation all the particles move at once, whereas in the asynchronous update implementation, the particles move one after the other.

The PSOA methodology was extended through various hybrid PSOA implementations. Hybrid PSOA's can change the applicability of the algorithm, by incorporating gradient information [32, 33]. Hybrid PSOA's alter the empirical performance of the algorithm by various strategies such as reducing the stochastic terms in the velocity rule [3, 6, 29] or increasing the stochastic terms in the velocity rule [34, 35].

Chapter 3

Diversity in the PSOA

3.1 Introduction

The aim of evolutionary strategies is to improve on the performance of random searches in optimization [36]. This is accomplished by identifying regions containing good designs. It is then assumed that searches in the region of good designs may uncover further improved designs. Various strategies exist to identify and explore regions with a high probability of improved candidate solutions. These strategies distinguish between different evolutionary computation algorithms; the algorithms include genetic programming [37], genetic algorithms [38], evolutionary strategies [39], differential evolution [40] and the particle swarm optimization algorithm (PSOA) [3, 4].

The PSOA was introduced by Kennedy and Eberhart [3] as a gradient free stochastic optimization algorithm. The fundamental principle behind the PSOA is the evolutionary advantages that the sharing of information offers. This is often known as 'collaborative searching'.

The PSOA is quite simple: At first, a swarm of p particles is randomly deployed in an n-dimensional design domain. The particles then update their positions in the design domain over unit time increments using a simple stochastic rule, known as the 'velocity rule'.

The quality of each particle's position at each iteration is then evaluated using the objective or cost function. Each particle's cognitive memory allows it to remember it's own best cost function value, with associated position, over time. Importantly, the social interaction and awareness of the particles allows them to also remember 'fit' or 'good' cost function values the swarm itself found over time. The particles' movement can be over a continuous domain, a discrete domain [6, 41] or a mixed discrete-continuous domain.

The original formulation of Kennedy and Eberhart [3] is repeated here verbatim:

$$\begin{aligned}
&vx[\,][\,] = vx[\,][\,]\\
&+2 * \text{rand}() * (pbest[\,][\,] - presentx[\,][\,])\\
&+2 * \text{rand}() * (pbestx[\,][gbest] - presentx[\,][\,]).
\end{aligned}$$

The bracket pair $[\ ][\ ]$ represents an $n \times p$ matrix, while rand() supposedly represents a scalar uniform random number. The rightmost terms between round brackets $(\cdot)$ in the second and third lines are denoted the cognitive and social components of learning, respectively.

The widespread use of, and the interest in, the PSOA, attests to the brilliance of the idea of Kennedy and Eberhart. However, their notation is not unambiguous: It is unclear whether the random numbers are scalar numbers which simply scale the *magnitude* of the cognitive and social components of learning, or whether the random numbers are vectors that scale *each component* of the cognitive and social components of learning. Ambiguous notation persists in the literature and the *reported* (i.e. notation) implementation is frequently different to the *actual* implementation. Examples of the *magnitude* scaling notation are [41, 42, 43, 44, 45, 46]. Examples of the *component* scaling notation are [3, 12, 13, 25, 35].

While the randomness issue outlined here may seem of minor importance, the implications are quite severe, as will be demonstrated in sections to come.

In fact, it will be demonstrated that for the *magnitude* scaling implementation, the particle trajectories collapse to p n-dimensional line searches. The *component* scaling implementation does not suffer this drawback. Instead, diverse stochastic search trajectories are retained. (In this chapter, the term ‘diversity’ implies the opposite of the collapse of a trajectory to a line.)

Also shown is that some popular heuristics like maximum velocity limit, position restriction, craziness and high initial velocities are possibly of less importance than originally thought; their greatest contribution is that they prevent the collapse of particle trajectories to lines.

3.2 Notes on the PSOA formulation

The investigation of the PSOA is started by defining the *instantaneous search domain* of a particle, viz. the domain to which the search of particle i at iteration k is restricted. Also introduced is the term *limit behavior*, viz. the limiting behavior of the complete swarm when no improvement in objective function value is experienced by any individual particle (and hence the swarm itself).

From Eqs. (2.2) and (2.3), observe that the instantaneous search domain depends on two independent contributions, namely the deterministic contribution due to the term $(\boldsymbol{x}_k^i + w\boldsymbol{v}_k^i)$, and the stochastic contribution due to the term $(\boldsymbol{\nu}_k^i \in \boldsymbol{\chi}_k^i)$. The situation is depicted in Figure 3.1, where we use the notation $x_k^i(j)$ to indicate the j-th component of vector $\boldsymbol{x}_k^i$.

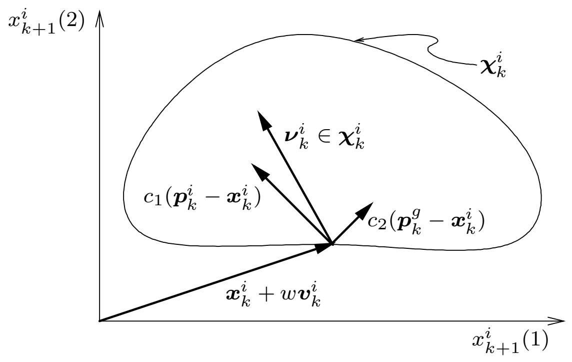

$x_{k+1}^i(2)$
$\boldsymbol{\chi}_k^i$
$\boldsymbol{\nu}_k^i \in \boldsymbol{\chi}_k^i$
$c_1(\boldsymbol{p}_k^i - \boldsymbol{x}_k^i)$
$c_2(\boldsymbol{p}_k^g - \boldsymbol{x}_k^i)$
$\boldsymbol{x}_k^i + w\boldsymbol{v}_k^i$
$x_{k+1}^i(1)$

Figure 3.1: The position vector $\boldsymbol{x}_{k+1}^i$, partitioned into a deterministic contribution $(\boldsymbol{x}_k^i + w\boldsymbol{v}_k^i)$ and a stochastic contribution $(\boldsymbol{\nu}_k^i \in \boldsymbol{\chi}_k^i)$.

The instantaneous search domain depends on the stochastic domain $\boldsymbol{\chi}_k^i$, the position vector $\boldsymbol{x}_k^i$, and the inertia term $w\boldsymbol{v}_k^i$. In turn, the stochastic domain $\boldsymbol{\chi}_k^i$ depends on the particle position $\boldsymbol{x}_k^i$, the particle best position $\boldsymbol{p}_k^i$, the best global position $\boldsymbol{p}_k^g$, and the cognitive and social scaling factors c_1 and c_2.

The stochastic domain $\boldsymbol{\chi}_k^i$ is bounded, and has an associated probability distribution, due to the random scaling of finite scalars.

The position rule described by Eq. (2.2) may be viewed as being constructed in 2 separate steps: 1) stochastically generate a point $\boldsymbol{\nu}_k^i$ in the stochastic domain $\boldsymbol{\chi}_k^i$, and 2) deterministically translate this point by $\boldsymbol{x}_k^i + w\boldsymbol{v}_k^i$.

Let us now proceed with an analysis of the two different formulations of the PSOA that have been used in the literature.

3.3 Implementation subtleties: Formulation 1 (PSOAF1)

For the first formulation of the PSOA, which is denoted PSOAF1 here, the stochastic vector $\boldsymbol{\nu}_k^i$ is given by

$$\boldsymbol{\nu}_k^i = c_1 r_{1k}^i(\boldsymbol{p}_k^i - \boldsymbol{x}_k^i) + c_2 r_{2k}^i(\boldsymbol{p}_k^g - \boldsymbol{x}_k^i), \tag{3.1}$$

where r_{1k}^i and r_{2k}^i represent two uniform real random scalar numbers between 0 and 1. r_{1k}^i and r_{2k}^i are updated at every iteration k, and for each particle i in the swarm. Hence r_{1k}^i and r_{2k}^i simply scale the magnitudes of the cognitive and social vectors $c_1(\boldsymbol{p}_k^i - \boldsymbol{x}_k^i)$ and $c_2(\boldsymbol{p}_k^g - \boldsymbol{x}_k^i)$.

For the sake of clarity, PSOAF1 is also described by the following pseudo code fragment:

```
for I = 1 to number of particles do
  R1 = uniform random number
  R2 = uniform random number
  for J = 1 to number of dimensions do
    V[I][J]=w*V[I][J]
           +C1*R1*(P[I][J]-X[I][J])
           +C2*R2*[G[I][J]-X[I][J])
  enddo
  X[I][J] = X[I][J]+V[I][J]
enddo
```

Let us now study the stochastic contribution $\boldsymbol{\nu}_k^i$ to the composition of the instantaneous search domain given by Eq. (3.1). The cognitive vector $(\boldsymbol{p}_k^i - \boldsymbol{x}_k^i)$ and the social vector $(\boldsymbol{p}_k^g - \boldsymbol{x}_k^i)$ consist of the directions and distances from the current position $\boldsymbol{x}_k^i$ to the best particle position $\boldsymbol{p}_k^i$, and the best global position $\boldsymbol{p}_k^g$; the cognitive and social vectors can be anything from normal to parallel w.r.t. each other.

When the cognitive vector $(\boldsymbol{p}_k^i - \boldsymbol{x}_k^i)$ and the social vector $(\boldsymbol{p}_k^g - \boldsymbol{x}_k^i)$ are not parallel, Eq. (3.1) may be interpreted as the vector equation of a bounded plane $\boldsymbol{\mathcal{P}}_k^i$ in n-dimensional space. The plane is bounded, since the length of the cognitive and social vectors are scaled independently by the finite scalars $c_1 r_{1k}^i$ and $c_2 r_{2k}^i$. The bounded plane $\boldsymbol{\mathcal{P}}_k^i$ is then translated in n-dimensional space by the addition of $\boldsymbol{x}_k^i$ and $w\boldsymbol{v}_k^i$, as depicted in Figure 3.2.

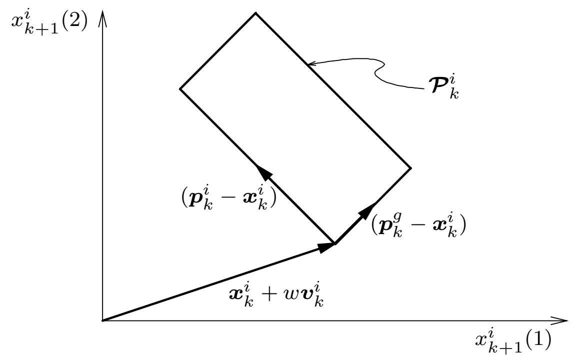

$x^i_{k+1}(2)$
$\mathcal{P}^i_k$
$(\boldsymbol{p}^i_k - \boldsymbol{x}^i_k)$
$(\boldsymbol{p}^g_k - \boldsymbol{x}^i_k)$
$\boldsymbol{x}^i_k + w\boldsymbol{v}^i_k$
$x^i_{k+1}(1)$

Figure 3.2: Partitioning the position vector $\boldsymbol{x}^i_{k+1}$ into a deterministic contribution $(\boldsymbol{x}^i_k + w\boldsymbol{v}^i_k)$, and a stochastic contribution $(\boldsymbol{\nu}^i_k \in \mathcal{P}^i_k)$, for $c_1 = c_2 = 2$.

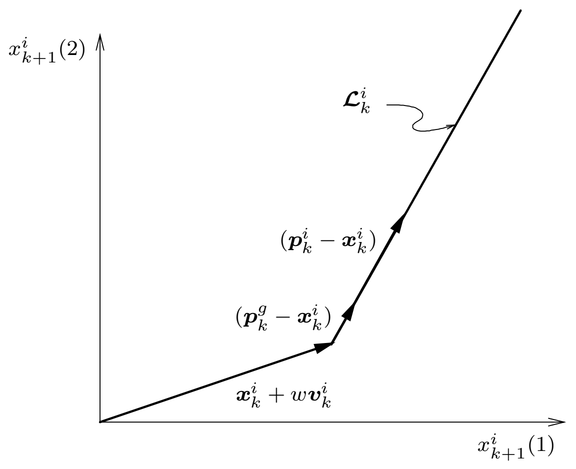

$x^i_{k+1}(2)$
$\mathcal{L}^i_k$
$(\boldsymbol{p}^i_k - \boldsymbol{x}^i_k)$
$(\boldsymbol{p}^g_k - \boldsymbol{x}^i_k)$
$\boldsymbol{x}^i_k + w\boldsymbol{v}^i_k$
$x^i_{k+1}(1)$

Figure 3.3: Partitioning the position vector $\boldsymbol{x}^i_{k+1}$ into a deterministic contribution $(\boldsymbol{x}^i_k + w\boldsymbol{v}^i_k)$, and a stochastic contribution $(\boldsymbol{\nu}^i_k \in \mathcal{L}^i_k)$, for $c_1 = c_2 = 2$.

If the cognitive vector $(\boldsymbol{p}^i_k - \boldsymbol{x}^i_k)$ and the social vector $(\boldsymbol{p}^g_k - \boldsymbol{x}^i_k)$ are parallel, Eq. (3.1) may be interpreted as the vector equation of a bounded line $\mathcal{L}^i_k$ in n-dimensional space. Again, the line is translated in the n-dimensional space by the addition of $\boldsymbol{x}^i_k$ and $w\boldsymbol{v}^i_k$, as depicted in Figure 3.3.

3.3.1 Investigation of the limit behavior of PSOAF1

Let us study the 3-dimensional dynamic limiting behavior of a particle. For the sake of simplicity, it is assumed that the best particle position $\boldsymbol{p}^i_k$, and the best global position $\boldsymbol{p}^g_k$ remain unchanged for 25 consecutive iterations.

Randomly generate the best particle best position $\boldsymbol{p}^i_0$, the best global position $\boldsymbol{p}^g_0$, and the initial position vector $\boldsymbol{x}^i_0$ between -2 and 2 over each dimension. The initial velocity vector $\boldsymbol{v}^i_0$ is assumed to equal $\boldsymbol{0}$. No velocity or position restriction is implemented. The study is conducted with $c_1 = c_2 = 2$ and $w = 0.8$, being values which have frequently been used by others in combination with constant inertia [8, 47, 45].

The sequence of 25 consecutive iterations, for a single particle, is depicted in Figure 3.4. The figure

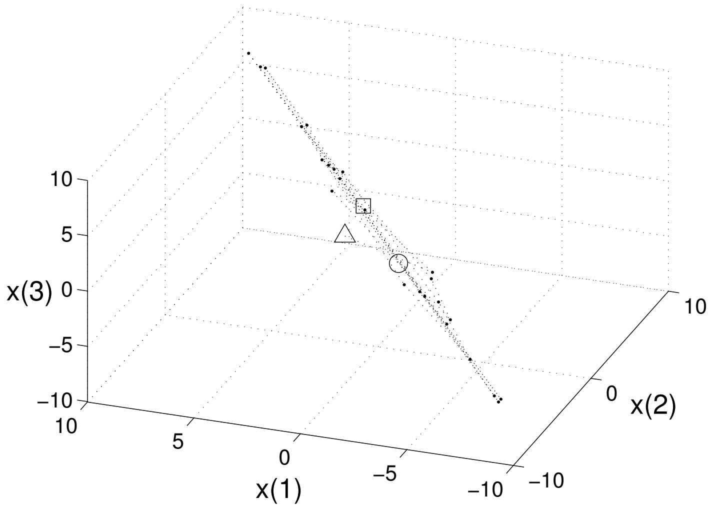


Figure 3.4: PSOAF1: Position vectors $\boldsymbol{x}^i_{k+1}$ generated for 25 consecutive iterations. The best particle position $\boldsymbol{p}^i_k$ and the best global position $\boldsymbol{p}^g_k$ are kept constant. Representing $\boldsymbol{x}^i_0$ is $\triangle$, $\boldsymbol{p}^i_k$ is $\bigcirc$, and $\boldsymbol{p}^g_k$ is $\square$.

illustrates that the particle trajectory collapse to a straight line as the iteration counter increases.

Let us consider this (undesirable) phenomenon in more detail: The angle $\bar{\theta}$ between the cognitive vector $(\boldsymbol{p}^i_k - \boldsymbol{x}^i_k)$ and social vector $(\boldsymbol{p}^g_k - \boldsymbol{x}^i_k)$ may be determined using

$$\bar{\theta} = \cos^{-1}\Big(\frac{|(\boldsymbol{p}^i_k - \boldsymbol{x}^i_k)\cdot(\boldsymbol{p}^g_k - \boldsymbol{x}^i_k)|}{||(\boldsymbol{p}^i_k - \boldsymbol{x}^i_k)||\ ||(\boldsymbol{p}^g_k - \boldsymbol{x}^i_k)||}\Big). \tag{3.2}$$

If $\bar{\theta} = 0°$, the vectors $(\boldsymbol{p}^i_k - \boldsymbol{x}^i_k)$ and $(\boldsymbol{p}^g_k - \boldsymbol{x}^i_k)$ are parallel, when $\bar{\theta} = 90°$, the vectors $(\boldsymbol{p}^i_k - \boldsymbol{x}^i_k)$ and $(\boldsymbol{p}^g_k - \boldsymbol{x}^i_k)$ are perpendicular.

The experiment performed in constructing Figure 3.4 is repeated. However, this time the algorithm runs for 1000 iterations, and the average angle θ is determined from 100 independent runs. The study is conducted for $n = 3$ and $n = 30$, to determine any dependency on dimensionality.

The results for a dimensionality of $n = 3$ and $n = 30$ are respectively depicted in Figures 3.5(a) and 3.5(b). The figures illustrate that as the iterations progress, the average angle θ between the cognitive vector $(\boldsymbol{p}^i_k - \boldsymbol{x}^i_k)$ and the social vector $(\boldsymbol{p}^g_k - \boldsymbol{x}^i_k)$ sharply decreases for both $n = 3$ and $n = 30$. The initial decrease is rapid; after some 150 iterations, the average angle θ is less than $10°$.

The foregoing implies that for a given particle, the position vector $\boldsymbol{x}^i_{k+1}$ is updated in a 'long narrow bounded plane'. This is reminiscent of multiple line searches in a largely restricted domain.

It should be noted that for low values of the inertia w, complete collapse to line searches ($\theta = 0°$) is demonstrated within 1000 iterations. Furthermore: for a given particle, it is impossible to escape from this long narrow bounded plane until *another* particle's best position $\boldsymbol{p}^g_k$ is updated, since each particle's best positions can only be updated in its own long narrow bounded plane.

For a swarm of p particles, this implies that if the best global position vector $\boldsymbol{p}^g_k$ is not updated, the

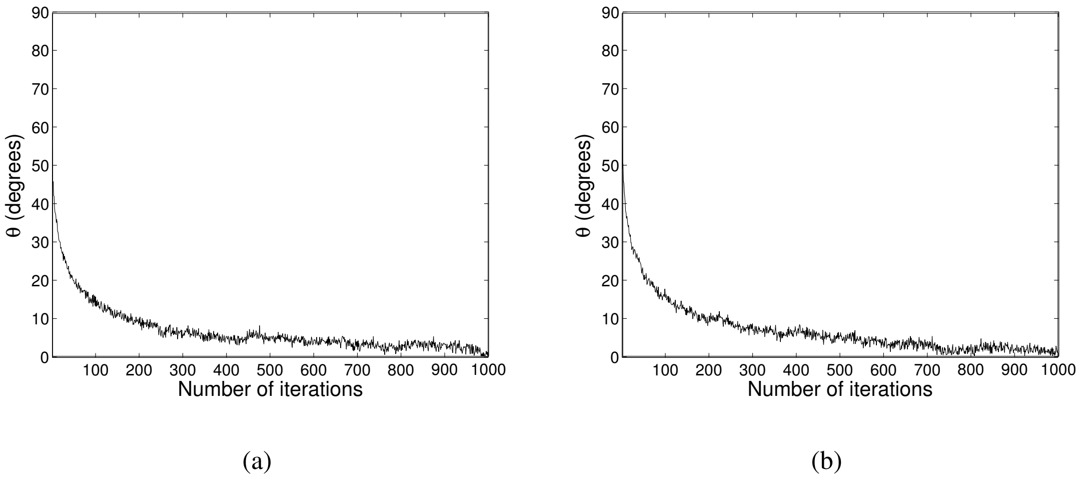


Figure 3.5: PSOAF1: The average angle θ between $(\boldsymbol{p}_k^i - \boldsymbol{x}_k^i)$ and $(\boldsymbol{p}_k^g - \boldsymbol{x}_k^i)$ versus iteration number, for a dimensionality of a) $n = 3$, and b) $n = 30$. For the 1000 iterations, the best particle position $\boldsymbol{p}_k^i$ and the best global position $\boldsymbol{p}_k^g$ are stationary.

swarm conducts p line searches in n-dimensional space for possibly a high number of consecutive iterations. Worse: the line searches all *intersect in the same point* in the design domain, being the best global position $\boldsymbol{p}_k^g$. Again: if a particle updates its own best position vector $\boldsymbol{p}_k^i$, then it remains searching in a line. Only when the global best position vector is updated do all other particles again (briefly) search in planes.

For obvious reasons, the foregoing may have severe implications on algorithm performance.

3.4 Implementation subtleties: Formulation 2 (PSOAF2)

In the alternative implementation of the velocity rule, each component of $(\boldsymbol{p}_k^i - \boldsymbol{x}_k^i)$ and $(\boldsymbol{p}_k^g - \boldsymbol{x}_k^i)$ is scaled independently. The vector directions are then no longer preserved.

In order to scale each component independently, the scalar random numbers r_{1k}^i and r_{2k}^i in the stochastic vector in Eq. (3.1) are replaced by two random diagonal matrices $\boldsymbol{R}_{1k}^i$ and $\boldsymbol{R}_{2k}^i$ as follows:

$$\boldsymbol{\nu}_k^i = c_1 \boldsymbol{R}_{1k}^i (\boldsymbol{p}_k^i - \boldsymbol{x}_k^i) + c_2 \boldsymbol{R}_{2k}^i (\boldsymbol{p}_k^g - \boldsymbol{x}_k^i). \tag{3.3}$$

The $\boldsymbol{R}_{mk}^i$ random diagonal matrices are explicitly given as

$$\boldsymbol{R}_{mk}^i = \begin{bmatrix} \rho_{11k}^i & 0 & \cdots & 0 \\ 0 & \rho_{22k}^i & \cdots & 0 \\ \vdots & & \ddots & \vdots \\ 0 & \cdots & \cdots & \rho_{nnk}^i \end{bmatrix}, \quad m = 1, 2, \tag{3.4}$$

with $0 < \rho_{jjk}^i < 1, \quad j = 1, \ldots, n$ a uniform random number [12, 48].

The above form is used since it has previously been introduced by others. However, for future use, the following equivalent form is proposed

$$\boldsymbol{\nu}_k^i = c_1 \boldsymbol{r}_{1k}^i \circ (\boldsymbol{p}_k^i - \boldsymbol{x}_k^i) + c_2 \boldsymbol{r}_{2k}^i \circ (\boldsymbol{p}_k^g - \boldsymbol{x}_k^i), \tag{3.5}$$

where the $\circ$ operator indicates element by element multiplication. Hence the random vectors $\boldsymbol{r}_{mk}^i$ are given as

$$\boldsymbol{r}_{mk}^i = \left(\rho_{1k}^i,\ \rho_{2k}^i,\ \cdots,\ \rho_{nk}^i\right), \quad m = 1, 2. \tag{3.6}$$

Eq. (3.5) is no longer a vector representation of a bounded plane $\boldsymbol{\mathcal{P}}_k^i$, since the non-zero components of the cognitive vector $(\boldsymbol{p}_k^i - \boldsymbol{x}_k^i)$ and the social vector $(\boldsymbol{p}_k^g - \boldsymbol{x}_k^i)$ are independently scaled. As a result, possible stochastic updates can occur in n-dimensional space.

The pseudo code for PSOAF2 is

```
for I = 1 to number of particles do
  for J=1 to number of dimensions do
    R1=uniform random number
    R2=uniform random number
    V[I][J]=w*V[I][J]
           +C1*R1*(P[I][J]-X[I][J])
           +C2*R2*[G[I][J]-X[I][J])
  enddo
  X[I][J] = X[I][J]+V[I][J]
enddo
```

The difference with the pseudo code given for PSOAF1 is subtle; the only difference is that the random numbers are updated inside the *for-loop* that runs over the design dimensions $(1, \ldots, n)$. However, the implications are severe.

3.4.1 Investigation of the limit behavior of PSOAF2

As with PSOAF1, the 3-dimensional trajectory of a particle when the best particle position $\boldsymbol{p}_k^i$, and the best global position $\boldsymbol{p}_k^g$ are kept constant for 25 consecutive iterations is studied. In performing the experiment, the same settings as for PSOAF1 are used, except for using a lower value of inertia, namely $w = 0.6$, since PSOAF2 is unstable at high values of w.

Figure 3.6 suggests that the particle trajectories now do not collapse to line searches; instead, 'diversity' is retained. To verify this, the average angle θ between the cognitive vector $(\boldsymbol{p}_k^i - \boldsymbol{x}_k^i)$ and social vector $(\boldsymbol{p}_k^g - \boldsymbol{x}_k^i)$ is calculated. A similar approach as for PSOAF1 is used; again the study is conducted for $n = 3$ and $n = 30$ to determine the sensitivity to problem dimensionality.

As opposed to PSOAF1, the trajectories of PSOAF2 do not collapse to line searches. As depicted in Figure 3.7, the angle θ is roughly 47° and 37° for $n = 3$ and $n = 30$ respectively, over the entire range of iterations studied.

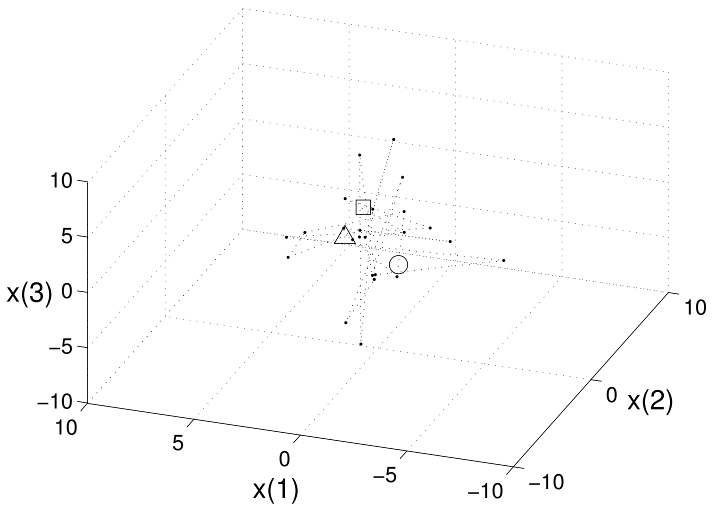


Figure 3.6: PSOAF2: Position vectors $\boldsymbol{x}^i_{k+1}$ generated over 25 iterations without updating the particle best position vector $\boldsymbol{p}^i_k$ and the global best position vector $\boldsymbol{p}^g_k$. No restriction is imposed on the velocity vector.

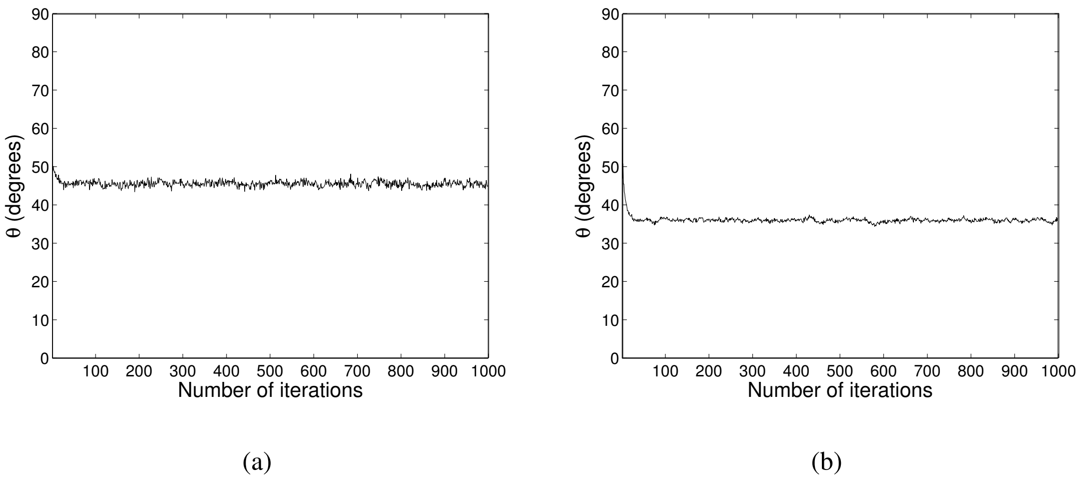


Figure 3.7: PSOAF2: The average angle θ between $(\boldsymbol{p}^i_k - \boldsymbol{x}^i_k)$ and $(\boldsymbol{p}^g_k - \boldsymbol{x}^i_k)$ versus iteration number, for a dimensionality of a) $n = 3$, and b) $n = 30$. For the 1000 iterations, the best particle position $\boldsymbol{p}^i_k$ and the best global position $\boldsymbol{p}^g_k$ are stationary.

3.5 Numerical experiments

Numerical experiments to evaluate the difference in performance between PSOAF1 and PSOAF2 are now conducted.

The aim is not an exhaustive determination of optimal algorithmic parameters, but to illustrate the effects of the different velocity updating rules in the two formulations. Hence a basic PSOA is implemented, without additional heuristics such as maximum velocity restriction, position restriction, craziness or dynamic inertia reduction or increase. (These are however discussed in sections to come.)

Initial velocities are set equal to $\mathbf{0}$. A simple synchronous updating scheme [31] is used. Real variables are implemented using *double-precision floating-point* arithmetic.

For this study the algorithm parameters are $c_1 = c_2 = 2$, the swarm size is $p = 20$ particles and the computations are performed for various constant inertia factors w. Each run is terminated after 10000 iterations, and the reported results are average values obtained from 100 independent runs.

In this study the following five test functions are used:
i) The Rosenbrock function (unimodal, f_0):

$$f_0(\boldsymbol{x}) = \sum_{i=1}^{\frac{n}{2}} \Big(100\big(x_{2i} - x_{2i-1}^2\big)^2 + \big(1 - x_{2i-1}\big)^2\Big).$$

ii) The Quadric function (unimodal, f_1):

$$f_1(\boldsymbol{x}) = \sum_{i=1}^{n} \left(\sum_{j=1}^{i} x_j\right)^2.$$

iii) The Ackley function (multimodal, f_2):

$$\begin{aligned} f_2(\boldsymbol{x}) = \quad & -20\exp\left(-0.2\sqrt{\tfrac{1}{n}\textstyle\sum_{i=1}^{n} x_i^2}\right) \\ & -\exp\left(\tfrac{1}{n}\textstyle\sum_{i=1}^{n}\cos(2\pi x_i)\right) + 20 + e. \end{aligned}$$

iv) The generalized Rastrigin function (multimodal, f_3):

$$f_3(\boldsymbol{x}) = \sum_{i=1}^{n}\Big(x_i^2 - 10\cos(2\pi x_i) + 10\Big).$$

v) Finally, the generalized Griewank function (multimodal, f_4):

$$f_4(\boldsymbol{x}) = \frac{1}{4000}\sum_{i=1}^{n} x_i^2 - \prod_{i=1}^{n}\cos\left(\frac{x_i}{\sqrt{i}}\right) + 1.$$

The parameters used in the study are given in Table 3.1. The *domain* column represents the range of each dimension of the design variables; the test function domains are symmetrical about 0.0 in all dimensions.

Table 3.1: Test function parameters.

Function	n	domain
f_0	30	$\pm$ 2.048
f_1	30	$\pm$ 100.0
f_2	30	$\pm$ 30.0
f_3	30	$\pm$ 5.12
f_4	30	$\pm$ 600.0

3.6 Discussion of numerical results

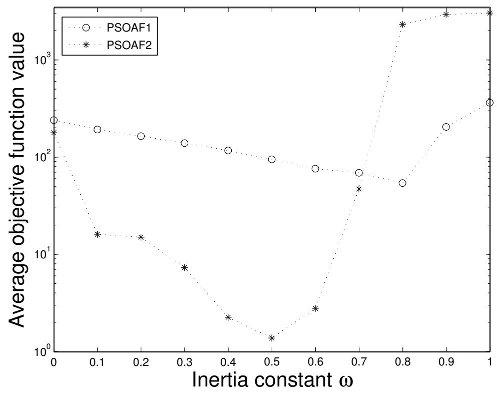


Figure 3.8: Average function value after 2×10^5 function evaluations (10000 iterations) over 100 runs on the Rosenbrock test function (f_0).

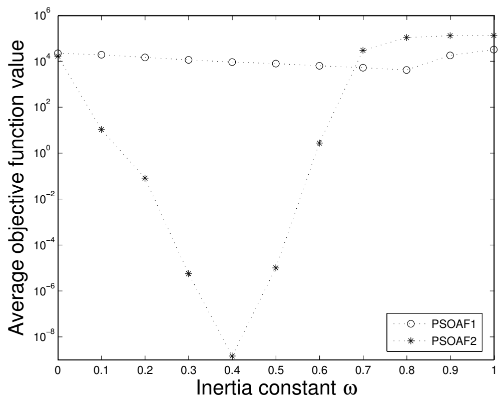


Figure 3.9: Average function value after 2×10^5 function evaluations (10000 iterations) over 100 runs on the Quadric test function (f_1).

Figures 3.8, 3.9, 3.10, 3.11 and 3.12 depict the average objective function values after 2×10^5 function evaluations (10000 iterations) for the 5 test functions under consideration. A summary

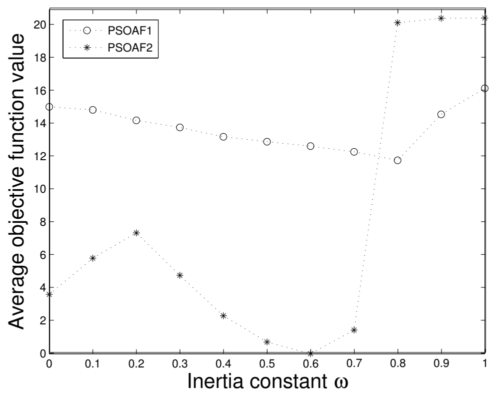


Figure 3.10: Average function value after 2×10^5 function evaluations (10000 iterations) over 100 runs on the Ackley test function (f_2).

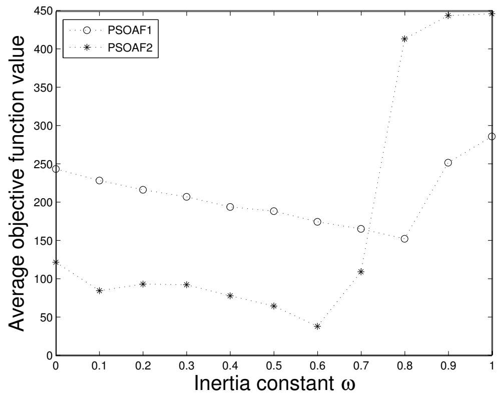


Figure 3.11: Average function value after 2×10^5 function evaluations (10000 iterations) over 100 runs on the Rastrigin test function (f_3).

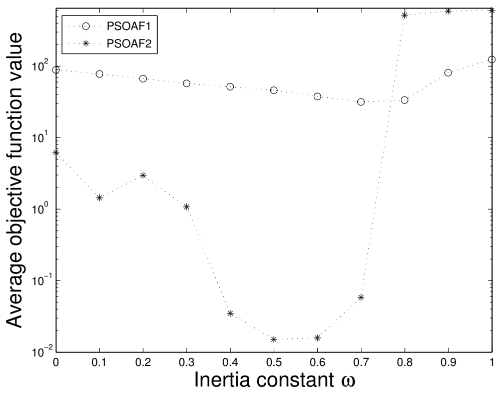


Figure 3.12: Average function value after 2×10^5 function evaluations (10000 iterations) over 100 runs on the Griewank test function (f_4).

Table 3.2: Constant inertia factor at which the best average objective function value is obtained.

	PSOAF1		PSOAF2	
	$f^{\text{best}}_{\text{ave}}$	w	$f^{\text{best}}_{\text{ave}}$	w
f_0	54.845	0.8	1.393	0.5
f_1	4395.919	0.8	1.5×10^{-9}	0.4
f_2	11.804	0.8	9×10^{-15}	0.6
f_3	152.988	0.8	38.425	0.6
f_4	32.078	0.7	1.5×10^{-2}	0.5

of these results may be found in Table 3.2, which tabulates the best function values obtained with PSOAF1 and PSOAF2 after the 10000 iterations, together with the corresponding inertia factor.

The figures and the table clearly illustrate the vast difference in performance between PSOAF1 and PSOAF2, with PSOAF2 superior to PSOAF1 for all the problems considered.

The performance of PSOAF1 improves as the inertia factor w is increased, up to an inertia factor of $w = 0.8$. For higher inertia factors ($w \geq 0.9$) there is a rapid decline in the performance of the algorithm, due to instability. (Instability occurs due to excessive particle velocities, since there is no limit on the maximum value of velocity.) For low values of inertia, performance is hampered, since the collapse of a particle trajectory to a line search occurs earlier than at high values of w.

PSOAF2 performs well for $w \leq 0.6$, but becomes unstable for $w \geq 0.8$. For the test set and conditions used, the optimal performance for PSOAF2 is obtained with $0.4 \leq w \leq 0.6$. For PSOAF1, higher values of w are suitable.

The average convergence history for Ackley's test function, for PSOAF1 and PSOAF2, is depicted in Figures 3.13(a) and 3.13(b) respectively. The two figures are drawn on the same scale; PSOAF2 is clearly superior to PSOAF1, both in terms of average convergence rate and in terms of the quality of the solution found (for reasonable values of w).

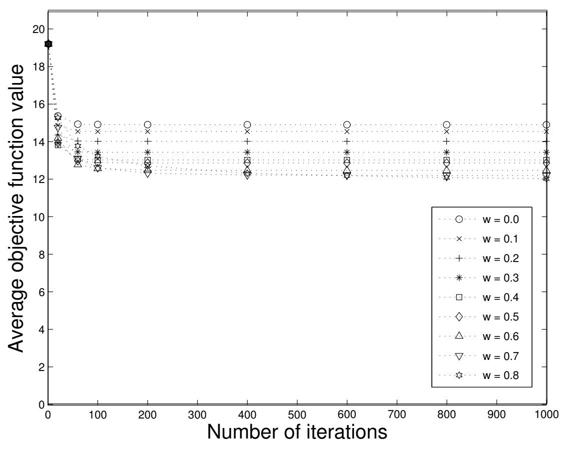
3.7 Notes on some heuristics of the PSOA

An explanation of the effects of some popular heuristics that are widely considered to improve the performance of the PSOA follows. (For obvious reasons, it should now be clear that the gain in performance for PSOAF1 can be expected to be far higher than for PSOAF2. Indeed, most 'successful' heuristics merely prevent or delay the collapse of particle trajectories to lines in PSOAF1.)

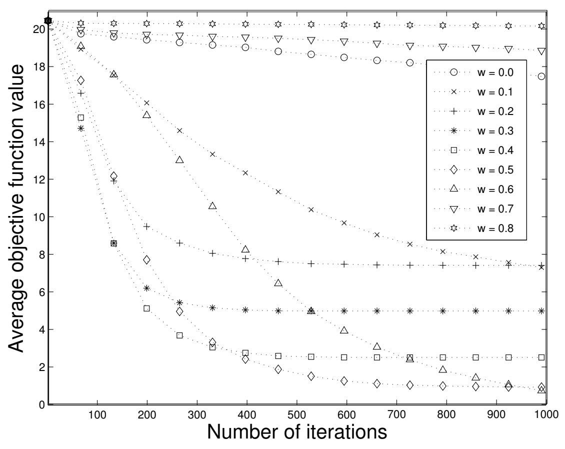
3.7.1 Local best neighborhood

Local neighborhoods [49] are used in an attempt to introduce independent social groups into the swarm; information between these groups is then propagated back into the swarm in some structured fashion.

Local neighborhoods are beneficial since their introduction results in clusters of line searches



(a)



(b)

Figure 3.13: Average convergence history for Ackley's test function, for a) PSOAF1, and b) PSOAF2.

which intersect in the locally best point of each neighborhood, instead of only intersecting in the best global point of the complete swarm. Hence local neighborhoods increase the diversity of the algorithm. In addition, the communication of this information between neighborhoods results in an instantaneous increase in diversity.

3.7.2 Non-zero initial velocities

The introduction of non-zero initial velocities delays line searches in PSOAF1, since the term $w\boldsymbol{v}_k^i$ translates the instantaneous search domains (planes) in the design domain, which increases diversity. Note however that this only helps during initial iterations; these contributions damp out over time.

3.7.3 Maximum velocity restriction

Maximum allowed velocity [4] is a well known heuristic, frequently used in the literature. It is used to stabilize the algorithm.

There are two fundamentally different ways to implement maximum velocity restriction. Firstly, the restriction can be placed on each component of the vector $\boldsymbol{v}_{k+1}^i$, as shown in Figure 3.14(a). Alternatively, the length of $\boldsymbol{v}_{k+1}^i$ can be restricted, as shown in Figure 3.14(b).

The advantage of restricting each component is the ease of implementation. When restricting the components of the velocity vector, the magnitude of the velocity vector depends on the magnitudes of the velocity components. The maximum length that the velocity vector can obtain is

$$\boldsymbol{v}_{k+1}^{max} = \sqrt{\sum_{j=1}^{n} \left(v^{max}(j)\right)^2},$$

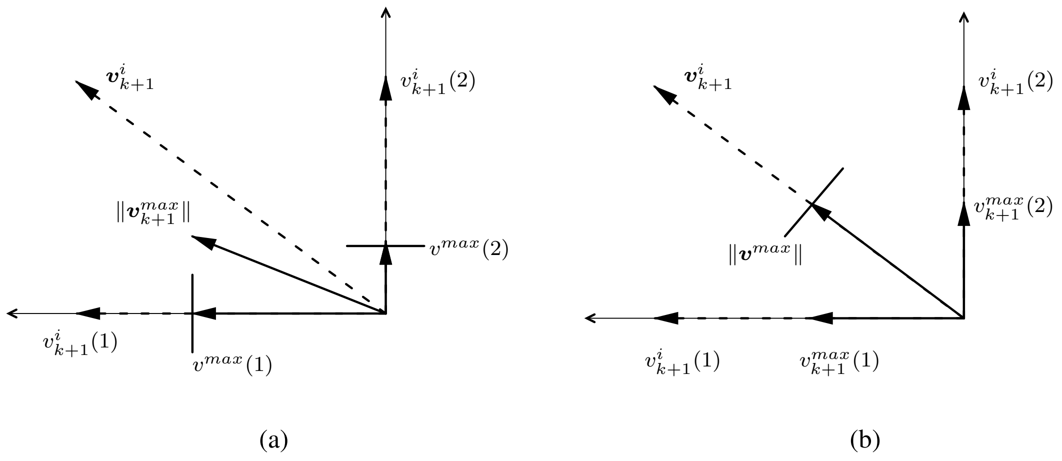


Figure 3.14: The velocity restriction implemented by a) restricting the component values of the velocity vector a) restricting the magnitude of the velocity vector.

when all the vector components are restricted. In addition, by restricting each component of the velocity vector, (Figure 3.14(a)), the direction of the velocity vector is not preserved. The result is that the velocity directions are not limited to the line $\boldsymbol{\mathcal{L}}^i_k$.

On the other hand, restriction of the velocity magnitude (Figure 3.14(b)), does not alter the velocity direction between consecutive iterations.

In summary, this implies that the results of restriction of the maximum velocity component on the PSOA are two-fold. Firstly, this stabilizes the algorithm, by limiting the maximum component value in each dimension. Secondly, this increases diversity through the generation of position vectors, which by their very nature are not confined to the line $\boldsymbol{\mathcal{L}}^i_k$.

To illustrate the foregoing, the average angle θ between the vectors $(\boldsymbol{p}^i_k - \boldsymbol{x}^i_k)$ and $(\boldsymbol{p}^g_k - \boldsymbol{x}^i_k)$ is again studied. As before, randomly generate the best particle position vector $\boldsymbol{p}^i_0$, the best global position vector $\boldsymbol{p}^g_0$, initial positions $\boldsymbol{x}^i_0$, and the initial velocities $\boldsymbol{v}^i_0$ with each vector component between -2 and 2.

Velocity restriction on both the component and the magnitude is implemented. Again, $c_1 = c_2 = 2$ and $w = 0.8$. The algorithm is again terminated after 1000 consecutive iterations, with θ averaged over 100 runs. As before, the best particle position $\boldsymbol{p}^i_k$ and the best global position $\boldsymbol{p}^g_k$ are assumed stationary. The study is conducted for $n = 30$.

When applying restriction to the *components* of velocity, Figure 3.15(a) depicts that the average angle θ between the cognitive vectors $(\boldsymbol{p}^i_k - \boldsymbol{x}^i_k)$ and the and social vectors $(\boldsymbol{p}^g_k - \boldsymbol{x}^i_k)$ is some 60°; the trajectories do not collapse.

However, when the velocity *magnitude* is restricted, the angle θ quickly collapses to 0° (Figure 3.15(b)). (Here, a restriction of $||\boldsymbol{v}^i_{k+1}|| \leq 4$ is used.)

Depicted in Figure 3.16 is the average function value after 2×10^5 function evaluations or 10000 iterations for the Griewank test function f_4. As expected, velocity restriction significantly improves algorithm performance for high values of inertia $(w \geq 0.8)$, since instability at high inertia factors

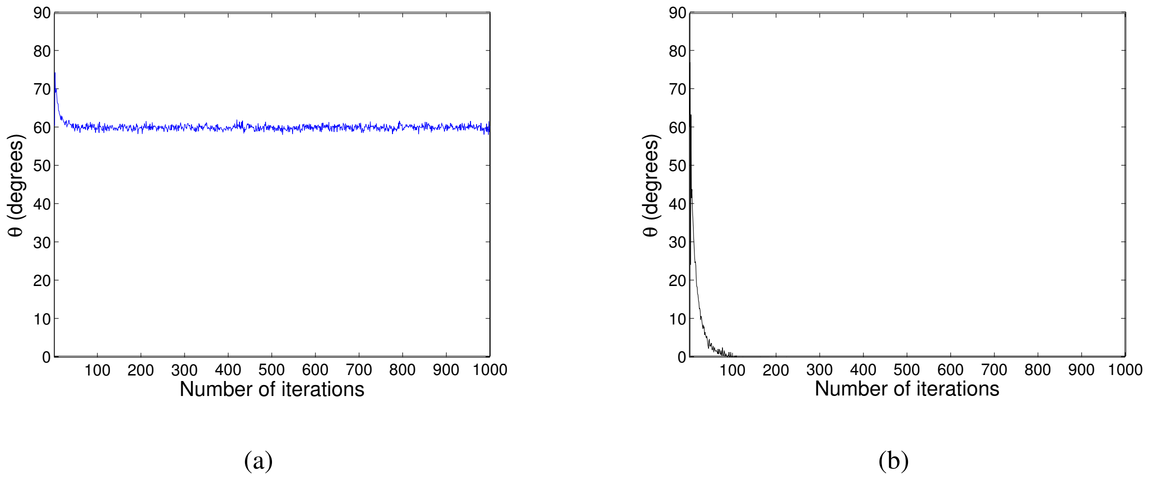


Figure 3.15: Velocity restriction on a) the components, and b) the magnitude of velocity. Depicted is the average angle θ between $(\boldsymbol{p}_k^i - \boldsymbol{x}_k^i)$ and $(\boldsymbol{p}_k^g - \boldsymbol{x}_k^i)$ versus iteration number, for a dimensionality of $n = 30$. For the 1000 iterations, the best particle position $\boldsymbol{p}_k^i$ and the best global position $\boldsymbol{p}_k^g$ are stationary.

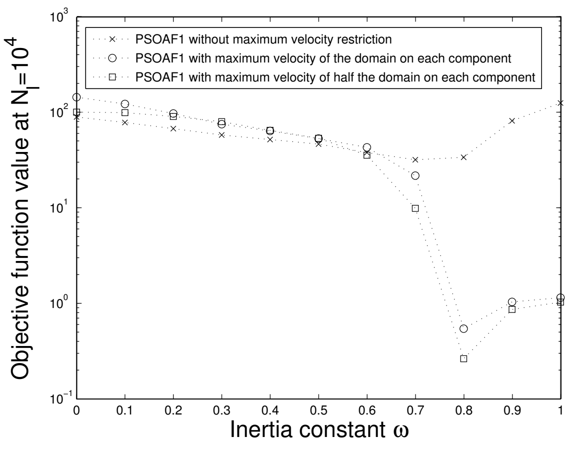


Figure 3.16: Average function value for the Griewank test function (f_4) after 1000 iterations, averaged over 100 runs.

is prevented.

Clearly, a restriction on each vector component of maximum velocity avoids the collapse of search trajectories to lines. This observation was also made by Carlisle and Dozier [31], who demonstrated that lower values of maximum velocity improves algorithm performance. The mechanism for this is now easily understood: lower values of maximum velocity trigger the introduction of diversity more often than higher values of maximum velocity. (Although low values of course decrease the instantaneous search domain.)

3.7.4 Minimum velocity restriction

Velocity may also be restricted on the lower end of the scale, as a mechanism to prevent premature convergence. Again, either the components or the magnitude of velocity may be restricted.

However, firstly demonstrated is the remarkable sensitivity of the algorithm to arithmetic precision, which may be viewed as a special case of minimum velocity restriction: PSOAF1 is implemented using both *single-precision floating-point* arithmetic and *double-precision floating-point* arithmetic. Otherwise, the two implementations are *identical*. Also provided is a minimum velocity limit on each component of the velocity vector of the code implemented with *double-precision floating-point* arithmetic. Using pseudo-code, it is done as follows:

```
for I = 1 to number of particles do
  for J = 1 to number of dimensions do
    calculate V[I][J]
    if abs(V[I][J]) less than Vmin then
      V[I][J] = sign(V[I][J]) * Vmin
    endif
  enddo
enddo
```

Values of $\mathrm{V}_{\text{min}} = 10^{-2}$ and $\mathrm{V}_{\text{min}} = 10^{-3}$ are used, again for runs consisting of 10000 iterations, and reported are the average values for 100 runs. Results for only the Griewank test function (f_4) are presented, but the results for the other test functions are similar. The same initial positions, velocities and random number sequences are used in the three different implementations.

Figure 3.17 depicts the average objective function value. The *single-precision floating-point* implementation is slightly superior to the *double-precision floating-point* implementation, simply because the lower precision results in angles which are not quite zero. For the *double-precision floating-point* implementation, minimum velocity increases the performance of the algorithm dramatically. (Although not shown in the figure, values of V_{min} equal to the precision attainable in single precision, renders the two implementations almost identical.)

As with maximum velocity restriction, the implications of minimum velocity restriction are two-fold: Firstly, premature convergence may be overcome, and the collapse to the line $\boldsymbol{\mathcal{L}}_k^i$ may be delayed.

(The very low values above are for illustrative purposes only, in practice higher values may be desirable.)

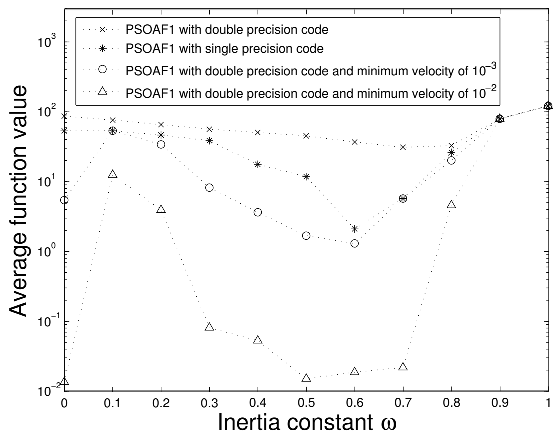


Figure 3.17: Average function value after 2×10^5 function evaluations (10000 iterations) over 100 runs on the Griewank test function (f_4).

3.7.5 Position restriction

There are various ways to implement position restriction, as Robinson and Rahmat-Samii [30] for example point out in their study. Position restriction may be seen as an alternative implementation of velocity restriction, albeit more complicated, since every vector $\boldsymbol{x}^i_{k+1}$ may equivalently be represented as the vector $\boldsymbol{x}^i_k + \boldsymbol{v}^i_{k+1}$.

While the implementation of position restriction has to a large extent been neglected in the literature, note that the desirability and implementation should be judged against the background of collapsed line searches.

3.7.6 Craziness

Craziness [3] is a heuristic which randomly places particles in the design domain. Normally performed at low probability, it obviously increases diversity, and prevents the collapse to line searches. (At high probability, craziness results in (ineffective) pure random search.) The craziness operator is somewhat reminiscent of mutation in the genetic algorithm (GA).

Continuous monitoring of θ may effectively be combined with craziness to prevent premature convergence of the PSOA. In addition, ‘small perturbations’, reminiscent of creep mutation in the GA, may be more effective than the equivalent of jump mutation in the GA.

A number of authors have previously reported that craziness is ineffective. However, it is a simple matter to demonstrate that craziness is indeed effective for PSOAF1, and in particular for problems of high dimensionality.

Consider the very simple unconstrained n-dimensional (convex) quadratic test function

$$f_5(\boldsymbol{x}) = \sum (x_i - i)^2, \quad i = 1, 2, 3, \cdots, n,$$

subject to the bounds

$$-100 \leq x_i \leq 100, \quad i = 1, 2, 3, \cdots, n.$$

Table 3.3: Effect of probability of craziness P_{cr} on $f_5(\boldsymbol{x})$

n	P_{cr}	PSOAF1	PSOAF2
10	0.00	1.158E+02	6.310E-32
	0.05	1.719E-05	1.650E-09
	0.10	1.165E-02	1.434E-03
30	0.00	1.627E+04	4.469E-28
	0.05	2.787E-02	5.597E-04
	0.10	2.528E+00	1.073E+01

Tabulated numerical results are presented in Table 3.3; the optimum solution $f^* = 0.0$. Again, PSOAF1 and PSOAF2 are used without any heuristic whatsoever. Both algorithms are terminated after 10000 iterations, and the reported results are averaged over 100 independent runs. Again $p = 20$ is used. For PSOAF1, select $w = 0.7$, for PSOAF2 select $w = 0.5$. Many implementations of craziness are possible; in this case, P_{cr} indicates the probability of a given particle to become crazy.

Table 3.3 illustrates that craziness is highly beneficial to PSOAF1, while it impairs the performance of PSOAF2. The reasons are obvious: for PSOAF1, craziness increases diversity, and prevents the collapse of trajectories to line searches. (Note that this does not imply that craziness is indispensable for PSOAF1; any of the other heuristics which increases diversity will of course improve the performance of PSOAF1.) The impairment of PSOAF2 is also easily explained: this algorithm is diverse 'enough'. (In the limit, an increase in craziness of course results in pure random search in both algorithms.)

3.7.7 Increasing social awareness

Again, in this approach [34, 35], a number of different implementations are possible. In a simple implementation, the best position of each particle in the swarm is considered when constructing the velocity rule.

It is now clear why this works: diversity is increased; the collapse of trajectories to line searches is delayed.

3.7.8 Inertia factor

Zheng et. al. [16] proposed to increase the inertia w as the iterations progress. They increased w from 0.4 to 0.9 over the prescribed number of iterations. Clearly, this assists in prolonging the time before the trajectories collapse to line searches, due to translation of the instantaneous search domains (planes) in D.

Very high inertia factors (w larger than unity) may of course also be used to prolong diverse searches. The amount $1 - w$ may be viewed as the equivalent damping term in a spring-mass-damper system [19]. Hence $w > 1.0$ implies negative damping, or the introduction of energy into the system as the iterations progress. However, this is only recommended in terminal phases of the PSOA; during initial phases, this may result in pure random search. In addition, $w > 1.0$ should be used in combination with heuristics like position restriction.

3.7.9 Using a single random number

Some authors have attempted to replace the random numbers r^i_{1k} and r^i_{2k} in Eq. (3.1) by a single random number. While this is acceptable, the implications should be understood.

This issue is addressed by asking the following question: *What is the range and the distribution of the sum of two random numbers r_1 and r_2 that are both uniformly distributed between 0 and 1?* (All three authors initially guessed that the range would lie between 0 and 2 and that the probability distribution would remain uniform.)

However, a simple Monte Carlo simulation based on 10^6 instances reveals that the solution is the bi-linear probability distribution. The reason is of course obvious [50]: In order to generate a number close to zero both r_1 and r_2 need to be close to zero. In contrast, to generate a number close to 1, a large number of possibilities exist e.g. $(0.1+0.9)$, $(0.7+0.3)$, $(0.6+0.4)$, $(0.5+0.5)$, etc.

The sum of two random variables $r^i_{3k} = r^i_{1k} + r^i_{2k}$ should not be considered a uniform random number with a range between 0 and 2, e.g. see [16]. Instead it should be chosen from a bi-linear distribution.

3.8 On tuning of PSOA parameters (finding universal optimal parameter values)

In the foregoing, it is demonstrated that it is essential to distinguish between implementations of randomness into the PSOA when reporting results. 'Optimal settings' for the parameters of PSOAF1 are simply not optimal for PSOAF2.

However, the same argument applies for many other implementational issues. Without further elaboration, a check list of implementation issues is given that should in my opinion be reported when 'optimal setting' for the PSOA are published, since the optimality of the parameters strongly depends on the implementation.

The implementational issues to be reported are as follows:

1. General:
   (a) the selected velocity rule (using unambiguous notation),
   (b) the updating strategy (whether synchronous or asynchronous), and
   (c) the stopping criteria used.
2. The heuristics used (with a detailed description thereof):
   (a) minimum velocity restriction (and arithmetic precision),

   (b) maximum velocity restriction,
   (c) the implemented inertia strategy,
   (d) position restriction,
   (e) etc.

3. The parameters used:
   (a) the population size p,
   (b) the initial inertia and velocity values, and
   (c) the social and cognitive scaling factors c_1 and c_2.

Finally, optimal parameters should of course be understood within the context of the ‘no free lunch algorithms [51, 52].

3.9 Closure

Implementation subtleties due to ambiguous notation that resulted in two distinctly different implementations of the PSOA have been pointed out. Discerning between these two implementations is of crucial importance.

While the behavior of the two different implementations is markedly different, they only differ in the formulation of the velocity updating rule. In fact, the differences are merely due to subtle differences in the introduction of randomness into the algorithm.

A scrutiny of PSOA codes has revealed that the reported implementation is often different to the actual computer implementation. Against the background of this chapter, it is now also possible to identify papers for which this discrepancy holds.

As shown for the first implementation, the particle trajectories collapse to ineffective line searches. The second implementation does not suffer this drawback. Instead, diverse stochastic search trajectories are retained.

Also shown is that some popular heuristics like maximum velocity limit, position restriction, craziness and high initial velocities are not of overwhelming importance in their own right; they merely prevent collapse of the particle trajectories to lines.

Finally, it is emphasized that the determination of optimal values for parameters like inertia, velocity limit, etc. has to be performed within the context of the formulation used. A list of parameters and implementational issues that should be included when reports on PSOA parameter settings are written is proposed.