A Polar Coordinate Particle Swarm Optimiser

Wiehann Matthysen and Andries P. Engelbrecht

Department of Computer Science, University of Pretoria, South Africa
engel@cs.up.ac.za

Abstract

The Particle Swarm Optimisation (PSO) algorithm consists of a population (or swarm) of particles that are "flown" through an n-dimensional space in search of a global best solution to an optimisation problem. PSO operates in Cartesian space, producing Cartesian solution vectors. By making use of an appropriate mapping function the algorithm can be modified to search in polar space. This mapping function is used to convert the position vectors (now defined in polar space) to Cartesian space such that the fitness value of each particle can be calculated accordingly. This paper introduces the Polar PSO algorithm that is able to search in polar space. This new algorithm is compared to its Cartesian counterpart and the experimental results show that the Polar PSO outperforms the Cartesian PSO in low dimensions when both algorithms are applied to the search for eigenvectors of different $n \times n$ square matrices.

Key words: Particle Swarm Optimization, Polar Coordinates, Boundary Constraints

1 Introduction

Optimisation algorithms such as Differential Evolution (DE) [1] and Particle Swarm Optimisation (PSO) [2] were originally designed to find solutions to problems that are defined in n-dimensional Cartesian space. However, modified versions of these algorithms appeared as the benefits that these algorithms offered were sought in other problem areas as well. The standard versions of the algorithms were adapted by either modifying their underlying logic or by introducing an appropriate mapping function.

An example of algorithm modification is where Kennedy and Eberhart [3] modified the PSO algorithm to enable it to search for discrete solutions in binary space (Binary PSO or BinPSO). Each component of a particle's velocity vector is used to determine a probability that the bit in that same dimension of its position vector will flip from a zero to a one and vice versa. For example, a velocity component $v_{ij} = 0.8$ of particle i at index j implies that the position component at the same index has a 69% chance to be bit one.

An example of incorporating a mapping function into the PSO algorithm is the Angle-Modulated PSO (AMPSO) [4]. The original discrete problem is redefined such that the search is now carried out in 4-dimensional Euclidean space to locate the coefficient vectors of an n-dimensional trigonometric bit string generator. This mapping technique was later applied to the Differential Evolution (DE) algorithm [5].

This paper introduces the Polar PSO that is both a modification of the standard PSO algorithm as well as making use of an appropriate mapping function that takes particle positions in polar space and convert it to Cartesian space. Similar research has already been done to enable Evolutionary Algorithms (EAs) to search in polar space. An example of this is the Polar Evolutionary Strategy (Polar ES) [6]. Empirical results show that the Polar ES outperforms its Cartesian counterpart when both algorithms are applied to the search for unit-length projection vectors.

What sets this research apart is that the PSO algorithm itself as well as its behaviour during the search process are fundamentally different from an EA [7]. This means that specialised modifications that are unique to the PSO algorithm are required to enable it to search in polar space. These modifications are needed to address some of the difficulties that are inherent in changing from Cartesian to polar space.

The first of these difficulties is that the new polar search space is a distorted version of the original Cartesian space. Local optimum regions near the Cartesian origin become enlarged in polar space while global optimum regions further away are reduced in size. This distortion causes the PSO algorithm to prematurely converge to these larger local optimum regions. The other side effect of this distortion is that random particle positions in polar space are not uniformly distributed when converted to Cartesian space and vice versa. This causes a decrease in initial swarm diversity which results in less efficient exploration. The third issue is related to the way in which the bounded polar position vectors should be handled. A naïve approach of allowing the particles to search beyond the search space bounds increases the size of the search space which results in lower quality solutions to be produced.

All of these issues as well as the modifications required to address them are discussed in the following sections: Section 2 describes the PSO, BinPSO and AMPSO algorithms. The polar conversion function is defined in Section 3 while Section 4 describes the effects of converting to polar coordinates. Section 5 describes the modifications required to correctly handle the bounded angular components. The complete Polar PSO algorithm is summarised in Section 6 while Sections 7 and 8 discuss the experimental setup and the results obtained. Finally, Section 9 concludes with some remarks.

2 Background

2.1 Particle Swarm Optimisation

PSO is an algorithm that models the social behaviour of birds within a flock. The algorithm was first introduced by Kennedy and Eberhart [2] as a simulation of this behaviour, but quickly evolved into one of the most powerful optimisation algorithms in the Computational Intelligence field.

The algorithm consists of a population (or swarm) of particles that are “flown” through an n-dimensional space. The position of each particle represents a potential solution to the optimisation problem and is used in determining the *fitness* (or performance) of a particle. These fitness values are used in recording the neighbourhood as well as personal best positions. The neighbourhood best position represents the best position found by the neighbourhood of particles connected to the current particle per execution of the algorithm, whereas the personal best position represents the historic best position of the particle. Different neighbourhood topologies have been defined which led to the *lbest* (ring topology) and *gbest* (entire swarm as neighbourhood) variants of the algorithm. The neighbourhood and personal best positions are used in guiding a particle through the search space allowing it to discover more promising regions that will lead to further exploration as this information is shared among the rest of the particles in the swarm.

The velocity of each particle i is calculated using the following rule:

$$v_{i,j}(t) = wv_{i,j}(t-1) + \alpha_1(t)(y_{i,j}(t) - x_{i,j}(t)) + \alpha_2(t)(\widehat{y}_{i,j}(t) - x_{i,j}(t)) \quad (1)$$

for dimensions $j = 1, \ldots, n$, where w (referred to as the momentum or inertia weight) determines the influence that the velocity at the previous time-step has on the current velocity, and $\alpha_1(t)$ and $\alpha_2(t)$ determines the influence that the personal best position $y_{i,j}(t)$ and the neighbourhood best position $\widehat{y}_{i,j}(t)$ has. These values are defined as $\alpha_1(t) = c_1 \cdot r_1(t)$ and $\alpha_2(t) = c_2 \cdot r_2(t)$ where c_1 and c_2 are the acceleration constants and $r_1(t), r_2(t) \sim U(0, 1)$.

The current position of particle i is then calculated by adding this updated velocity to the previous position:

$$x_{i,j}(t) = x_{i,j}(t-1) + v_{i,j}(t) \tag{2}$$

resulting in a new position that is potentially closer to a local or global optimum.

2.2 Binary PSO

The first PSO able to search in binary space were developed by Kennedy and Eberhart [3]. This version of the PSO is a modification to the original algorithm where the particle positions are equal length bit strings and the velocity of each particle is used to determine the probability that a bit at a certain index in the bit string will change from a zero to a one, and vice versa.

These modifications are captured in the following position update rule:

$$x_{i,j}(t+1) = \begin{cases} 0 \text{ if } r_i(t) \geq f(v_{i,j}(t)) \\ 1 \text{ if } r_i(t) < f(v_{i,j}(t)) \end{cases} \tag{3}$$

where

$$f(v_{i,j}(t)) = \frac{1}{1 + e^{-v_{i,j}(t)}} \tag{4}$$

and $r_i(t) \sim U(0, 1)$.

For large velocity values of $v_{i,j}$ the corresponding values of $f(v_{i,j}(t))$ will be close to 1.0 which means that the pseudo-random number generated by $r_i(t)$ will have a high probability of being less than this value giving a position component of 1. A similar argument can be made to derive the conditions for a 0 to be generated.

2.3 Angle Modulated PSO

The Angle Modulated PSO (AMPSO) [4] was developed to enable the standard PSO algorithm to search in binary space. However, AMPSO is different from BinPSO in that it makes use of an appropriate mapping function to achieve this goal as opposed to modifying the logic of the original algorithm.

The mapping is performed by defining an n-dimensional trigonometric bit string generator of the form:

$$g(x) = sin(2\pi(x - a) \times b \times \cos(A)) + d \tag{5}$$

where

$$A = 2\pi \times c(x - a) \tag{6}$$

and x are the evenly spaced values determined by the number of bits to be generated. The bits are generated according to the following rule:

$$b_i = \begin{cases} 0 \text{ if } g(x_i) \leq 0 \\ 1 \text{ if } g(x_i) > 0 \end{cases} \tag{7}$$

The standard PSO algorithm is then used to search in 4-dimensional Euclidean space to locate the coefficient vector (a, b, c, d) needed in equations (5) and (6).

3 Polar Conversion Function

Polar coordinates allow Cartesian vectors to be described in terms of independent angles and a positive radius. To enable a PSO to search in polar space a conversion function needs to be defined that maps the n-dimensional polar vectors back to Cartesian space. This conversion function (as formulated by Kendall [8]) is defined as:

$$\vec{x} = \mu(\vec{\theta})$$

$$\begin{aligned} x_1 &= r \cdot \sin(\theta_1) \cdot \sin(\theta_2) \ldots \sin(\theta_{n-2}) \cdot \cos(\theta_{n-1}) \\ x_2 &= r \cdot \sin(\theta_1) \cdot \sin(\theta_2) \ldots \sin(\theta_{n-2}) \cdot \sin(\theta_{n-1}) \\ x_3 &= r \cdot \sin(\theta_1) \cdot \sin(\theta_2) \ldots \cos(\theta_{n-2}) \\ \ldots & \quad \ldots \\ x_j &= r \cdot \sin(\theta_1) \ldots \sin(\theta_{n-j}) \ldots \cos(\theta_{n-j+1}) \\ \ldots & \quad \ldots \\ x_n &= r \cdot \cos(\theta_1) \end{aligned} \tag{8}$$

with $0 \leq r \leq \infty, 0 \leq \theta_j \leq \pi$ for $j = 1, \ldots, n-2$ and $0 \leq \theta_{n-1} \leq 2\pi$. Simplifying this function gives the well known polar coordinates in two dimensions:

$$\begin{aligned} x_1 &= r \cdot \cos(\theta) \\ x_2 &= r \cdot \sin(\theta) \end{aligned} \tag{9}$$

with $0 \leq r \leq \infty$ and $0 \leq \theta \leq 2\pi$, and the spherical coordinates in three dimensions:

$$\begin{aligned} x_1 &= r \cdot \sin(\theta_1) \cdot \cos(\theta_2) \\ x_2 &= r \cdot \sin(\theta_1) \cdot \sin(\theta_2) \\ x_3 &= r \cdot \cos(\theta_1) \end{aligned} \tag{10}$$

with $0 \leq r \leq \infty, 0 \leq \theta_1 \leq \pi$ and $0 \leq \theta_2 \leq 2\pi$. The search is then carried out in n-dimensional polar space to locate position vectors of the form $(r, \theta_1, \theta_2, \ldots, \theta_{n-1})$.

4 Effects of Polar Coordinate Conversion

Converting the original Cartesian search space to polar coordinates have a number of implications. These implications are discussed in this section as well as the effect that it has on the performance of the PSO algorithm.

4.1 Search Space Distortion

The most prevalent issue in this search space transformation is that the new polar search space becomes a distorted version of the Cartesian space, causing the search to be carried out in a space where it might be more difficult (depending on the problem) to locate a local or global optimum.

As an example, consider the n-dimensional Ackley function formulated as:

$$f(x) = 20 + e - 20 \cdot e^{(-0.2\sqrt{\frac{1}{n}\sum_{i=1}^{n} x_i^2})} - e^{(\frac{1}{n}\sum_{i=1}^{n}\cos(2\pi \cdot x_i))} \tag{11}$$

To search for the global optimum of the horizontally shifted version of this function (located at $(10, 10)$ in Cartesian space as shown in Fig. 1) in polar coordinates, the mapping as defined in equation (8) is used. This results in 2-dimensional particle positions of the form (r, θ) to be used with $0 \leq r \leq \infty$ and

$0 \leq \theta \leq 2\pi$. Plotting the function in this space results in a distorted version of the original as shown in Fig. 2.

Fig. 1. The 2-dimensional Cartesian version of the Ackley function with horizontal offset of -10.

Fig. 2. The 2-dimensional, distorted, polar coordinate version of the Ackley function in Fig. 1.

The distorted version of the function has a number of characteristics:

The function is stretched out along the radius-axis (shown here for values $10 \leq r \leq 20$) which has the effect that small steps along the θ-axis (for large values of r) will result in large corresponding steps in the original Cartesian search space. As the length of r increases the likelihood of particles missing a local or global optimum (as they move around) will also increase. The reason for this is that the distorted local or global optimum regions decrease in size as the distance of these regions from the Cartesian origin increases. This is shown in Fig. 2 where the contour located between θ-values 1.25 and 0.5 is slightly narrower at regions near $r = 18$ than compared to the opposite region of $r = 10.5$. At the other extreme, the Ackley function is significantly distorted at regions with small values of r such that any local or global optimum occurring at these regions becomes large in comparison with the rest of the optima occurring at positions further away from the Cartesian origin. This effect is clearly shown in Fig. 3.

Fig. 3. Illustration of the increase in size of local minima at positions with small values of r for the Ackley function in Fig. 1.

4.2 Implications

The implications that this distortion hold are that particles may easily get stuck at local optimum regions near the Cartesian origin (positions with r close to 0), or may easily miss global optimum regions that are further away.

This suggests that optimisation algorithms that operate in polar space are more suited to finding solutions of specialised problems. In particular, polar coordinates provide a convenient way to express fixed length Cartesian vectors by keeping the value of r fixed in the polar vector $(r, \theta_1, \theta_2, \ldots, \theta_{n-1})$.

This was exploited in [6] where the standard Evolutionary Strategy (ES) algorithm was modified to enable it to search for unit length projection vectors in polar space. The outcome of this was that the Polar ES produced better results when compared to its Cartesian counterparts. The reason is that by constricting the search to a unit-length hypersphere resulted in the search space to be reduced enough to enable the Polar ES to explore more effectively and produce better solutions.

4.3 Particle Position Initialisation

The distortion of the search space also has the effect that polar and Cartesian diversities will differ. Thus, random particle positions in polar space will not be uniformly distributed when converted to Cartesian space and vice versa.

(a) (b)

Fig. 4. Search space distortion effecting diversity of particle positions. (a) Random positions in polar space converted to Cartesian space. (b) Random positions in Cartesian space converted to polar space.

The diagrams in Fig. 4(a) and 4(b) illustrate the difference between polar and Cartesian diversity. Fig. 4(a) shows positions that were originally randomly generated in polar space (in terms of a radius and angle) and converted to Cartesian space using equation (9), while Fig. 4(b) illustrates how random 2-dimensional Cartesian vectors look like in polar space. The elliptical shapes in Fig. 4(b) are due to the initialisation of these Cartesian vectors within a bounded hypercube. Thus, the diagrams illustrate that as soon as a transformation is made to a new search space and particle positions are randomly initialised in this search space, the diversity of these positions (when they are converted back to the original search space) will be less than when they were randomly initialised in the original search space.

This distortion effect gets more severe as the number of dimensions increases. The reason for this is explained next. Suppose the particles in a swarm have n-dimensional polar position vectors. These position vectors are initialised randomly when the algorithm starts its execution such that each position vector are of the form $(r, \theta_1, \theta_2, \ldots, \theta_{n-1})$ with $0 \leq r \leq \infty, 0 \leq \theta_j \leq \pi$ for $j = 1, \ldots, n-2$ and $0 \leq \theta_{n-1} \leq 2\pi$. Using the conversion function in equation (8) will result in Cartesian components of the form:

$$\begin{aligned}
x_1 &= r \cdot \sin(\theta_1) \cdot \sin(\theta_2) \ldots \sin(\theta_{n-2}) \cdot \cos(\theta_{n-1}) \\
x_2 &= r \cdot \sin(\theta_1) \cdot \sin(\theta_2) \ldots \sin(\theta_{n-2}) \cdot \sin(\theta_{n-1}) \\
x_3 &= r \cdot \sin(\theta_1) \cdot \sin(\theta_2) \ldots \cos(\theta_{n-2}) \\
\ldots & \quad \ldots
\end{aligned}$$

However, the effect that this conversion will have is that Cartesian components $x_i \to 0$ for small values of i. The reason for this can be seen by taking the calculation of x_1 as an example: For $x_1 \approx r$ the θ_j-values need to be such that $\theta_j \approx \pi/2$ for $j = 1, \ldots, n-2$ and $\theta_{n-1} \approx 0$. If the θ_j values differ by a substantial amount from $\pi/2$ it will cause consecutive fractions to be multiplied $n-2$ times giving a component value that is close to zero for large values of n.

Thus, to force diversity in the original Cartesian space, particle positions need to be randomly initialised in this Cartesian space and by making use of an appropriate conversion function, the angle and radius values can be derived to give these positions in polar space. This function is the inverse of equation (8) and is defined as:

$$\vec{\theta} = \eta(\vec{x})$$

$$\begin{aligned}
r &= \sqrt{x_1^2 + x_2^2 + \cdots + x_n^2} \\
\theta_1 &= \cos^{-1}(x_n/r) \\
\theta_2 &= \cos^{-1}(x_{n-1}/r \cdot \sin(\theta_1)) \\
&\ldots\ldots \\
\theta_j &= \cos^{-1}(x_{n-j+1}/(r \cdot \sin(\theta_1) \ldots \sin(\theta_{j-1}))) \\
&\ldots\ldots \\
\theta_{n-2} &= \cos^{-1}(x_3/(r \cdot \sin(\theta_1) \ldots \sin(\theta_{n-3})))
\end{aligned} \tag{12}$$

with $\cos(\theta_{n-1}) = x_1/(r \cdot \sin(\theta_1) \ldots \sin(\theta_{n-2}))$ and $\sin(\theta_{n-1}) = x_2/(r \cdot \sin(\theta_1) \ldots \sin(\theta_{n-2}))$. To calculate the value of θ_{n-1} both the values of $\cos(\theta_{n-1})$ and $\sin(\theta_{n-1})$ are used in determining the quadrant that θ_{n-1} is located in.

5 Bounded Angular Components

The polar position vector of each particle in the swarm is defined as $(r, \theta_1, \theta_2, \ldots, \theta_{n-1})$ with $0 \leq r \leq \infty, 0 \leq \theta_j \leq \pi$ for $j = 1, \ldots, n-2$ and $0 \leq \theta_{n-1} \leq 2\pi$. These angular constraints need to be incorporated into the position as well as velocity update rules (refer to equations (2) and (1) respectively) to keep particles within the defined search region. This will have the effect of reducing the size of the search space to enable the algorithm to explore more effectively.

5.1 ϕ-Boundary

The ϕ-angle refers to the last angle (θ_{n-1}) in the polar position vector $(r, \theta_1, \theta_2, \ldots, \theta_{n-1})$ with $0 \leq \theta_{n-1} \leq 2\pi$. To update this angle during the position update step, modular arithmetic is used to ensure that the angle stays within its defined region. Thus, the new position update rule that implements this mechanism is given as:

$$x_{i,j}(t) = x_{i,j}(t-1) + v_{i,j}(t) \tag{13}$$

for dimensions $j = 1, \ldots, d-1$ and

$$x_{i,d}(t) = \begin{cases} (x_{i,d}(t-1) + v_{i,d}(t)) \ mod \ 2\pi + 2\pi \\ \text{if } x_{i,d}(t-1) + v_{i,d}(t) < 0, \\ (x_{i,d}(t-1) + v_{i,d}(t)) \ mod \ 2\pi \quad \text{otherwise.} \end{cases}$$

which is similar to *Periodic* mode as described in [9]. Negative ϕ-angles are handled appropriately by adding 2π to give the equivalent positive angle as equation (13) shows.

However, if this new position update rule is used without any modification to velocity update rule (1), then the search will be carried out inefficiently. The reason for this is explained next. Suppose the neighbourhood best particle is located at a position with a ϕ-angle close to 2π. If another particle in the swarm moves towards this position it may easily overstep the ϕ-boundary, causing position update rule (13) to give a position that is close to zero again (due to modular arithmetic).

(a) (b)

Fig. 5. (a) Swarm movement using standard velocity update rule. (b) Illustration of ϕ-angle movement.

The effect that this will have is illustrated in Fig. 5(a) for the Ackley function. A sample of five particle positions from the swarm shows that the particles move (or spin around) such that they stay within a disk-shaped region in Cartesian space.

To solve this problem, the velocity update rule as defined in equation (1) needs to be modified such that the particles move more efficiently with regard to their ϕ-angles. The required update is illustrated in Fig. 5(b). The arrow pointing upwards in an anti-clockwise direction indicates the direction of the ϕ_1-angle of a particle moving towards ϕ_2 (its personal or neighbourhood best position). If the particle moves in the opposite ϕ-direction it will take a shorter route towards the ϕ_2-angle as the arrow pointing downwards illustrates. This modification is captured in the following velocity update rule:

$$\begin{aligned} v_{i,j}(t) &= wv_{i,j}(t-1) \\ &\quad + \alpha_1(t)(y_{i,j}(t) - x_{i,j}(t)) \\ &\quad + \alpha_2(t)(\widehat{y}_{i,j}(t) - x_{i,j}(t)) \end{aligned} \tag{14}$$

for dimensions $j = 1, \ldots, d-1$ and

$$\begin{aligned} & v_{i,d}(t) = wv_{i,d}(t-1) \\ & + \\ & \begin{cases} \alpha_1(t)(y_{i,d}(t) - x_{i,d}(t)) \text{ if } (y_{i,d}(t) - x_{i,d}(t)) \leq \pi \\ \alpha_1(t)(sign(x_{i,d}(t) - y_{i,d}(t)) \times 2\pi - x_{i,d}(t) + y_{i,d}(t)) \\ \text{otherwise} \end{cases} \\ & + \\ & \begin{cases} \alpha_2(t)(\widehat{y}_{i,d}(t) - x_{i,d}(t)) \text{ if } (\widehat{y}_{i,d}(t) - x_{i,d}(t)) \leq \pi \\ \alpha_2(t)(sign(x_{i,d}(t) - \widehat{y}_{i,d}(t)) \times 2\pi - x_{i,d}(t) + \widehat{y}_{i,d}(t)) \\ \text{otherwise} \end{cases} \end{aligned}$$

5.2 θ-Boundaries

The θ-angles refer to the angles $\theta_1, \theta_2, \ldots, \theta_{n-2}$ of the polar position vector $(r, \theta_1, \theta_2, \ldots, \theta_{n-1})$. These angles are defined such that $0 \leq \theta_j \leq \pi$ for $j = 1, \ldots, n-2$.

If an update is made to these angles that is similar to the ϕ-angle update in equation (13) then particles close to the θ-boundary will be moved by a large amount if they overstep the boundary only by a small amount. The reason for this is explained next. Suppose an angle in the second quadrant is given by $\theta = \pi - 0.1$. If the angular velocity is 0.15 then the new angle given by modular arithmetic will be equal to $\theta = ((\pi - 0.1) + 0.15)\ mod\ \pi = 0.05$. This new angle is $\pi - 0.15$ distance away from the original angle. To solve this boundary constraint issue, a number of methods can be used to update the θ-angle more effectively [10]:

5.2.1 Random mode

With *Random* mode, the j-th θ-component of the polar vector that oversteps a boundary (i.e. $\theta_j < 0$ or $\theta_j > \pi$) is randomly re-initialised to keep it within the boundaries. This method of boundary enforcement is illustrated in Fig. 6(a).

5.2.2 Boundary mode

If a particle oversteps any of its j-th component's boundaries the particle is re-initialised such that the j-th component is equal to the value of the overstepped boundary. Thus, if $\theta_j < 0$ then $\theta_j = 0$ or if $\theta_j > \pi$ then $\theta_j = \pi$ as illustrated in Fig. 6(b). An additional *turbulence* factor can be incorporated which will perturb this boundary position with a low probability to introduce some diversity.

5.2.3 Shr

A particle can have the magnitude of its velocity vector scaled such that it exactly reaches the boundary upon the next position update. This is illustrated in Fig. 6(c).

(a) (b) (c)

Fig. 6. The different methods of handling the boundary constraints with (a) Random mode, (b) Boundary mode and (c) Shr

In all of the above situations the velocity needs to be updated to reflect the updated position that are now within the boundaries of the search space. This is done by subtracting the original position from the new position (after boundary enforcement has been applied).

Algorithm 1 Polar Coordinate Boundary Transformation

```
Let polarVector = $n$-dimensional vector with indices $i = 0, \dots, n-1$;
Let $\phi = NumberOfDimensions - 1$;
$i = NumberOfDimensions - 2$ downto 1
polarVector$_i > \pi$
$j = i$ to $NumberOfDimensions - 1$
polarVector$_j = \pi -$ (polarVector$_j$ $mod$ $\pi$);
polarVector$_\phi =$ (polarVector$_\phi + \pi$) $mod$ $2\pi$;
polarVector$_i < 0$
polarVector$_i = |$polarVector$_i|$;
$j = i + 1$ to $NumberOfDimensions - 1$
polarVector$_j = \pi -$ (polarVector$_j$ $mod$ $\pi$);
polarVector$_\phi =$ (polarVector$_\phi + \pi$) $mod$ $2\pi$;
```

5.2.4 Polar mode

A characteristic of the methods just discussed is that a transformed particle position does not correspond to its untransformed position if both positions are converted back to Cartesian coordinates. To achieve this the following transformation function is defined:

$$\vec{\theta_b} = \tau(\vec{\theta_u}) \tag{15}$$

with $0 \le r_u \le \infty, \theta_{u,j} < 0$ or $\theta_{u,j} > \pi$ for $j = 1, \ldots, n-2$ and $0 \le \theta_{u,n-1} \le 2\pi$ and also with $0 \le r_b \le \infty, 0 \le \theta_{b,j} \le \pi$ for $j = 1, \ldots, n-2$ and $0 \le \theta_{b,n-1} \le 2\pi$.

The vector $\vec{\theta_u}$ represents the unconstrained polar coordinate vector and $\vec{\theta_b}$ the resulting constrained polar coordinate vector. The requirement of the transformation function is that the relation $\mu(\theta_b) = \mu(\theta_u)$ must hold (see equation (8)) to ensure that equivalent vectors will be produced in Cartesian space. Pseudo code for the transformation function τ is listed as Algorithm 1.

6 The Polar PSO Algorithm

The steps of the Polar Coordinate PSO Algorithm are summarised in Algorithm 2.

Algorithm 2 Polar PSO Algorithm

```
Let $m$ = size of swarm;
*each particle* $i = 1$ to $m$
Let cartesianVector = random $n$-dimensional vector;
Let polarVector = $\eta$(cartesianVector); (equation (12))
Set position vector $\vec{x}_i$ of particle $i$ equal to polarVector;
Set $r$ in $\vec{x}_i$ equal to a fixed value;
Initialise particle $i$'s velocity vector $\vec{v}_i$ using some initialisation scheme;
*each particle* $i = 1$ to $m$
Let tempCartesianVector = $\mu(\vec{x}_i)$; (equation (8));
Evaluate fitness $f$(tempCartesianVector) and set particle $i$'s fitness value $F_i$ to be equal to this value;
Using the fitness value $F_i$ update particle $i$'s personal and neighbourhood best positions;
*each particle* $i = 1$ to $m$
Update the velocity $\vec{v}_i$ using equation (14);
Update the position $\vec{x}_i$ using equation (13);
Transform the position $\vec{x}_i$ by applying one of the boundary constraint methods to constrain the $\theta$-angles as discussed in Section 5.2;
If the position $\vec{x}_i$ changed as a result of this transformation, calculate new velocity by subtracting original position from transformed position;
*stopping condition is true*;
```

7 Experimental Approach

The experiments that were conducted focused on comparing the performance of the standard PSO algorithm operating in Cartesian space with the performance of the Polar PSO algorithm[1]. Control parameter values for both algorithms (as listed in equation (1)) were selected according to the guidelines in [11] with a value of 1.496180 being assigned to both c_1 and c_2 and w being set to 0.729844. These values satisfies the condition that will allow for convergent particle trajectories where the condition is [11]: $1 > w > \frac{1}{2}(c_1 + c_2) - 1 \geq 0$, as well as being the most optimal control parameters for a number of benchmark functions as empirically shown in [12]. For both algorithms, the velocity vector of all the particles were initialised to zero. A collection of 20 particles were configured in a *gbest* topology. Each algorithm was allowed to execute for a maximum of 1000 iterations and each experiment consisted of 100 of these executions. The results were averaged and reported with a 95% confidence interval using the *t-test.*

7.1 Benchmark functions

Table 1
PSO Settings for the Ackley function

Setting Number	Setting Description
S1	Cartesian Coordinates.
S2	Polar Coordinates.
S3	Polar Coordinates and Cartesian Initialisation.
S4	Polar Coordinates, Cartesian Initialisation and ϕ-position update. (see equation (13))
S5	Polar Coordinates, Cartesian Initialisation and bounded ϕ-component. (see equations (13) and (14))
S6	Polar Coordinates, Cartesian Initialisation and bounded ϕ and θ-components.

[1] Both algorithms have been implemented in the Computational Intelligence Library (CIlib) (`http://cilib.sourceforge.net`).

7.1.1 Ackley Function

The first benchmark function is the horizontally shifted version of the Ackley function that was introduced in Section 4.1. Table 1 lists the different settings for the PSO algorithm that were used to produce the corresponding results that are shown in Table 3.

Setting S1 corresponds to the standard *gbest* PSO that operates in Cartesian space while setting S2 corresponds to the same PSO, but makes use of the mapping function defined in equation (8) to enable it to operate in polar space. No modifications were made to the underlying PSO algorithm for setting S2. The settings S3 through to S6 correspond to the modifications that were made to the PSO algorithm to enable it to search more effectively in polar space and were discussed in detail in Sections 4 and 5.

For the PSO operating in Cartesian space (corresponding to setting S1), the position vector of each particle were randomly initialised as $x_j \sim U(-30, 30)$ for $j = 1, \ldots, n$.

The Polar PSO algorithm (shown as Algorithm 2) was slightly modified for this particular benchmark function. The radius r of each polar position vector were not initialised to a fixed value as the algorithm describes, but was instead calculated from a random Cartesian vector using equation (12). These Cartesian vectors were generated in the same way as the Cartesian position vectors were generated for the PSO corresponding to setting S1 as just discussed. The θ-component values of the particle position vectors for the PSO corresponding to setting S6 were constrained by making use of the Random mode boundary constraint method.

7.1.2 Eigenvector Function

The second set of benchmark functions involve finding an eigenvector for different $n \times n$ matrices. These $n \times n$ matrices are generated by making use of a pseudo-random number generator (Mersenne Twister [13]).

The fitness function is defined as the length of the vector calculated by subtracting a unit length input vector $\vec{x}'$ from the unit length resultant vector $\vec{u}'$ that is obtained from multiplying the input vector $\vec{x}$ with the random matrix and normalising. This function is formally defined as:

$$f(\vec{x}) = \left\| \frac{\vec{u}}{\|\vec{u}\|_2} - \frac{\vec{x}}{\|\vec{x}\|_2} \right\|_2 \tag{16}$$

where

$$u_i = \sum_{j=1}^{n} (x_j \times U(\alpha, \beta))$$

for $i = 1, \ldots, n$ and $\vec{x}$ is an n-dimensional input vector. The goal of an optimisation algorithm is then to minimise the value of $f(\vec{x})$. When the value of $f(\vec{x})$ is sufficiently close to zero the vector $\vec{x}$ can be considered an eigenvector of the matrix.

Each call to the function $f(\vec{x})$ needs to ensure that the pseudo-random number generator U is re-seeded with the same value to produce identical matrices in consecutive calls. The value of α and β determines the lower and upper bound values produced by the generator. Three different seed values combined with two different ranges were used to produce six different matrices per dimension.

The PSO operating in Cartesian space was initialised with random particle position vectors (input vector to $f(\vec{x})$) of the form $x_j \sim U(-1, 1)$ for $j = 1, \ldots, n$.

For the Polar PSO, the r values were set to 1.0 during initialisation and kept fixed during execution. The experiments were conducted using the Random, Boundary and Polar mode boundary constraint methods. A turbulence probability of 0.5 were used for the Boundary mode constraint method. This value determines the probability that a particle that is placed on the boundary of the search space (due to the particle overstepping the boundary) will move away from this boundary. The amount of turbulence was randomly sampled from the range $(0, \pi)$ and was either added or subtracted from the lower or upper boundary values depending on whether the component value was greater or less than the upper or lower boundary.

8 Results

8.1 Ackley Function

The values in Table 3 show the performance results that were obtained from applying the PSO algorithm to the problem of locating the global minimum of the horizontally shifted version of the Ackley function (as previously introduced) using the different settings listed in Table 1. The mean and best fitness values for the different runs are shown in Figures 7(a) - 13(b).

The results corresponding to settings S1 and S2 empirically illustrate the effect that the distortion of the search space has on the performance of the PSO algorithm. Performance results for setting S2, corresponding to the standard PSO algorithm operating in polar space, are significantly worse than compared to the results for S1, the PSO operating in Cartesian space. However, when both algorithms were executed in one hundred dimensional space, the PSO operating in Cartesian space produced slightly worse results than compared to the PSO operating in polar space.

In an attempt to improve these results, the PSO operating in polar space had the position vectors of its particles initialised in Cartesian space before being converted to polar coordinates using equation (12). The results obtained from this initialisation scheme are listed as setting S3 in Table 3 and are a significant improvement compared to the results obtained from using setting S2. This improvement is especially apparent for dimensions five through to thirty as the results show.

The next improvement to the polar PSO algorithm involved reducing the size of the search space by restricting the range of valid values for the angular components of each particle position vector in the swarm.

The first attempt at this improvement involved using modular arithmetic to restrict the range of values for the ϕ-component. This modification was captured in position update rule (13). However, Section 5.1 illustrated that if this position update rule is used without any modification to velocity update rule (1), then the search will be carried out inefficiently. This is confirmed in Table 3 with the results corresponding to setting S4 being significantly worse than compared to the previous results for dimensions three through to ten. By making use of velocity update rule (14) the performance of the PSO (corresponding to setting S5) was improved to the point where it outperformed the results of the previous polar PSO settings for the majority of the dimensions.

The final improvement to the polar PSO algorithm involved restricting the range of values for the θ-components. The Random mode boundary constraint method (discussed in Section 5.2) was used to restrict these values and produced the results that are shown as setting S6 in Table 3. For dimensions five through to one hundred the results for this setting were either significantly better than compared to the results of setting S5, or were only sightly worse. What is interesting to note is that this also resulted in the polar PSO algorithm to outperform its Cartesian counterpart in one hundred dimensional search space. This can be confirmed by referring to Figure 13(a) that shows the mean fitness value for the Ackley function in 100 dimensional search space.

8.2 Standard Benchmark Functions

The values in Tables 4 - 10 show the performance results that were obtained from applying the same PSO algorithm as listed in Section 8.1 to different standard benchmark functions. The same settings as listed in Table 1 were used. However, to ensure fair results, some of the standard benchmark functions were shifted in the same manner as the Ackley function and these settings are listed in Table 2.

Table 2
Horizontal shift applied to standard benchmark functions

Function	Horizontal Shift
Griewank	-300.0
Quadric	-50.0
Quartic	-0.5
Rastrigin	-10.0
Rosenbrock	0.0
Salomon	-300.0
Spherical	-10.0

For the Griewank, Quadric, Quartic, Rastrigin, Salomon and Spherical functions a similar trend can be observed in Tables 4 - 7, and Tables 9 - 9 when compared to the Ackley function, except for the 100 dimensional case, where the Cartesian PSO outperformed all of the polar PSO algorithms. However, with the Rosenbrock function a different trend can be observed as shown in Table 8. In this particular instance the polar PSO corresponding to setting S6 managed to outperform the Cartesian PSO in the majority of cases. The polar PSO also managed to outperform the Cartesian PSO at higher dimensional cases when compared to the rest of the benchmark functions, including the Ackley function.

The mean and best fitness values for the different runs of the Rastrigin function are shown in Figures 14(a) - 20(b). These figures show a similar trend to the Ackley function except for the 100 dimensional case where the Cartesian PSO clearly outperforms the polar PSO in all instances.

8.3 Eigenvector Function

Tables 11 - 16 summarise the accuracy obtained from performing the experiments to locate the eigenvectors of six different $n \times n$ matrices. Figures 21(a) -

23(b) show the mean fitness values for the eigenvector function where $\alpha = 0.0$ and $\beta = 1.0$ and a seed value of 1000 were used.

The results obtained from using values of $\alpha = 0.0$ and $\beta = 1.0$ in equation (16) clearly indicates that the Cartesian PSO outperformed its polar coordinate counterparts in higher dimensions ranging from forty to one hundred dimensions. In lower dimensions the polar coordinate versions of the PSO outperformed the Cartesian PSO in all three cases corresponding to the different seed values, except for the case where a seed value of 10000 in twenty dimensions were used. The Random mode boundary constraint method applied to the θ-angles proved to be an effective way of keeping the particles within the search space as Random Mode dominated in this particular setting and produced the best overall accuracy results when compared to Polar or Boundary mode. This is also evident in Figures 21(a) - 23(b) where the Random mode boundary constraint method's mean fitness value outperformed the other boundary constraint methods as well as the Cartesian PSO in ten to thirty dimensions.

The results obtained from using values of $\alpha = -1.0$ and $\beta = 1.0$ were slightly different when compared to the previous results. The Cartesian PSO produced the best results in the lowest and highest dimensional versions of the problem in all three cases corresponding to the different seed values. In the remainder of the cases corresponding to dimensions twenty through to forty the Polar PSO produced better results. The Polar mode boundary constraint method proved to be more effective in sixteen out of the eighteen cases when compared to Random mode.

Boundary mode did not perform very well when compared to the other boundary constraint methods or the Cartesian PSO in this particular case.

9 Conclusion and Future Work

This paper investigated the consequences of making use of an appropriate mapping function to allow a PSO to search in polar space. A major effect of transforming a Cartesian search space to polar coordinates is that the new search space is a distorted version of the original. This resulted in the problem that searching for the global minimum of the horizontally shifted version of the Ackley function became significantly more difficult in polar space than compared to Cartesian space.

In an attempt to address this problem a number of modifications were made to the standard PSO algorithm to enable it to search more effectively in polar space.

The first of these modifications addressed one of the side effects of transforming to polar coordinates. This side effect is that random particle positions in polar space are not uniformly distributed when converted back to Cartesian space. The effect that this had was that the initial particle positions of the swarm were not diverse enough to cover a sufficiently large portion of the search space. This resulted in lower quality solutions to be produced. To force Cartesian diversity a collection of vectors were randomly initialised in Cartesian space and by making use of an appropriate mapping function the radius and angular components of these vectors were extracted to form the polar position vectors of the particles in the swarm.

The second modification were made to reduce the size of the search space by restricting particles to search within valid ranges of the different angular components in a polar position vector. Modifications were made to both the position as well as velocity update rules to achieve this.

Despite the above mentioned modifications, the polar PSO could still not outperform its Cartesian counterpart when both algorithms were applied to well-known benchmark functions such as the Ackley function. However, polar coordinates provide a convenient way to express fixed-length Cartesian vectors by keeping the value of the radius fixed in the corresponding polar coordinate vector. This lead to the definition of a new benchmark function that involved finding the eigenvector of different $n \times n$ matrices. By keeping the value of the radius fixed in each polar position vector the search space was reduced enough to allow the Polar PSO algorithm to search within a unit-length hypersphere to produce results that outperformed its Cartesian counterpart at lower dimensions in the majority of the cases. As the number of dimensions increased the Polar PSO struggled to perform good. The reason for this can be attributed to the loss in floating point accuracy as consecutive sin and cos terms were multiplied in the conversion function that converted polar coordinates to Cartesian coordinates as shown in equation (8). This means that the benefits that the Polar PSO algorithm offer can only be exploited in lower dimensional problems.

Future research will be undertaken to determine the effect of search space transformations on the performance of certain EAs, particularly where the search space transformation could allow the EA to exploit the search landscape to improve its performance relative to other Optimisation algorithms not making use of the transformation.

Table 3
Performance results for the Ackley function (95% Confidence)

PSO Setting	Dimensions						
Number	3	5	10	20	30	50	100
S1	4.4409E-16	8.7041E-16	1.7825E-01	1.7600E+00	4.0594E+00	9.9076E+00	1.7670E+01
	6.6063E-32	2.3021E-16	9.1187E-02	2.3132E-01	4.4569E-01	5.7276E-01	2.9121E-01
S2	4.7073E-15	8.9017E-02	9.4197E+00	1.6829E+01	1.7079E+01	1.7170E+01	1.7238E+01
	3.1681E-16	7.8290E-02	1.4107E+00	1.7536E-01	2.0970E-02	1.0383E-02	4.8329E-03
S3	4.6008E-15	8.2311E-02	2.4430E+00	1.4070E+01	1.6809E+01	1.7175E+01	1.7266E+01
	3.6198E-16	7.1542E-02	9.7952E-01	1.0420E+00	2.0363E-01	4.2711E-02	8.2416E-03
S4	1.0838E-01	7.0124E-01	4.0496E+00	1.3847E+01	1.6644E+01	1.7188E+01	1.7260E+01
	9.2122E-02	1.9657E-01	1.0850E+00	9.8492E-01	2.6274E-01	3.4909E-02	9.7871E-03
S5	3.9968E-15	6.5849E-02	3.0491E+00	1.3901E+01	1.6639E+01	1.7163E+01	1.7257E+01
	5.2850E-31	6.4325E-02	1.0968E+00	1.0351E+00	3.3741E-01	3.3315E-02	1.0275E-02
S6	4.1034E-15	1.6462E-02	9.3402E-01	1.2627E+01	1.6297E+01	1.7148E+01	1.7275E+01
	1.2085E-16	3.2661E-02	4.9169E-01	1.2424E+00	5.1512E-01	1.6310E-01	6.1906E-03

Table 4
Performance results for the Griewank function (95% Confidence)

PSO Setting	Dimensions						
Number	3	5	10	20	30	50	100
S1	8.2792E-03	4.6167E-02	9.6656E-02	4.2939E-02	1.8432E-01	1.0995E+00	1.1722E+02
	1.2534E-03	6.4873E-03	1.2718E-02	1.4927E-02	1.5196E-01	2.2848E-01	1.4190E+01
S2	2.4053E-02	1.0718E-01	1.2380E-01	1.4222E-01	6.9949E+00	1.9449E+02	1.3724E+03
	2.7751E-03	1.4059E-02	1.5019E-02	1.0433E-01	6.1301E+00	3.0517E+01	3.4995E+01
S3	2.3585E-02	9.9147E-02	1.3616E-01	1.4949E-01	1.9890E+00	4.9562E+01	7.8701E+02
	4.2079E-03	1.1532E-02	2.0692E-02	9.5071E-02	1.7340E+00	1.6958E+01	5.1328E+01
S4	7.4730E-02	2.2711E-01	3.3630E-01	5.5178E-01	3.9824E+00	7.8747E+01	7.9551E+02
	1.6123E-02	3.3891E-02	6.7298E-02	1.7337E-01	1.5770E+00	1.8438E+01	5.2443E+01
S5	2.4171E-02	1.0113E-01	1.3113E-01	6.1493E-02	1.6073E+00	6.0871E+01	8.1697E+02
	4.5012E-03	1.1161E-02	1.6374E-02	2.9338E-02	2.1200E+00	1.7888E+01	4.6613E+01
S6	2.4530E-02	1.0629E-01	1.2566E-01	4.3857E-02	2.3358E+00	4.5992E+01	6.4760E+02
	3.5680E-03	1.2964E-02	1.3978E-02	1.0758E-02	2.4081E+00	2.1213E+01	5.5755E+01

Table 5
Performance results for the Quadric function (95% Confidence)

PSO Setting	Dimensions						
Number	3	5	10	20	30	50	100
S1	0.0000E+00	6.3109E-32	1.5661E-15	2.9074E-01	3.3600E+02	9.7179E+03	1.1811E+05
	0.0000E+00	8.8087E-32	1.0416E-15	1.2824E-01	9.4513E+01	6.9953E+02	7.5015E+03
S2	6.6138E-29	3.2564E-28	1.8156E-20	1.9832E+01	5.2643E+03	5.3971E+05	7.5018E+07
	1.2549E-29	1.5620E-29	3.5419E-20	3.1841E+01	1.9691E+03	1.8355E+05	9.5087E+06
S3	2.2416E-28	3.2362E-28	2.5604E-20	6.6812E-01	5.7901E+02	2.6919E+04	4.0715E+06
	3.2847E-28	1.3363E-29	5.0734E-20	1.0085E+00	1.3310E+02	3.9990E+03	6.9195E+06
S4	2.4392E-03	1.8288E-01	5.1719E+01	1.7313E+03	7.8280E+03	5.4973E+04	6.4308E+05
	2.0873E-03	1.3814E-01	2.8372E+01	5.2187E+02	1.8903E+03	9.5900E+03	1.0728E+05
S5	5.0487E-29	3.0494E-28	5.9655E-24	1.6688E-02	5.8246E+02	2.6987E+04	7.3322E+06
	5.2751E-45	1.9727E-30	4.8058E-24	1.1788E-02	1.2732E+02	3.8454E+03	9.3778E+06
S6	5.0487E-29	3.0646E-28	1.4093E-24	9.6982E-03	1.0831E+03	2.6721E+04	5.2217E+05
	5.2751E-45	2.5686E-30	1.0880E-24	7.6336E-03	1.2534E+03	5.6126E+03	1.0101E+05

Table 6
Performance results for the Quartic function (95% Confidence)

PSO Setting	Dimensions						
Number	3	5	10	20	30	50	100
S1	0.0000E+00	0.0000E+00	0.0000E+00	2.2226E-39	2.4319E-20	4.7863E-07	3.2922E+00
	0.0000E+00	0.0000E+00	0.0000E+00	2.5109E-39	1.7974E-20	2.7404E-07	9.3234E-01
S2	7.5015E-66	1.9789E-64	9.2923E-60	3.5592E-04	2.5877E-02	3.0240E+00	8.8770E+01
	5.9653E-66	5.6999E-65	1.8410E-59	7.0570E-04	2.2585E-02	7.9001E-01	5.0418E+00
S3	2.1610E-61	3.3869E-52	6.6859E-62	8.7390E-08	2.2945E-02	1.3789E+00	5.3951E+01
	2.8909E-61	6.7197E-52	8.3985E-62	1.0756E-07	2.5638E-02	4.9525E-01	4.7181E+00
S4	7.5686E-59	1.4608E-22	1.7648E-14	1.0246E-05	2.7293E-02	1.2334E+00	4.8040E+01
	1.5007E-58	2.8983E-22	3.5015E-14	1.6606E-05	2.8653E-02	3.7626E-01	3.5102E+00
S5	1.0359E-61	2.8795E-62	1.0331E-62	2.7056E-05	1.2133E-02	9.3566E-01	5.2052E+01
	1.1101E-61	4.8647E-62	3.6076E-63	5.2691E-05	1.3992E-02	2.9917E-01	5.5530E+00
S6	4.3021E-62	3.9996E-62	1.8810E-03	1.0074E-02	8.8031E-02	1.2521E+00	4.9195E+01
	6.2903E-62	7.6570E-62	3.7319E-03	1.4490E-02	6.7156E-02	7.8145E-01	6.0005E+00

Table 7
Performance results for the Rastrigin function (95% Confidence)

PSO Setting	Dimensions						
Number	3	5	10	20	30	50	100
S1	1.9899E-01	1.4128E+00	9.6327E+00	6.1617E+01	1.6331E+02	5.5043E+02	2.6077E+03
	8.4172E-02	2.3336E-01	1.0818E+00	5.2329E+00	1.2317E+01	3.3825E+01	1.1170E+02
S2	6.4672E-01	3.9841E+00	3.0465E+01	6.0667E+02	1.6955E+03	3.6432E+03	8.6692E+03
	1.4133E-01	5.9516E-01	4.6412E+00	6.3095E+01	5.9897E+01	6.3825E+01	5.4568E+01
S3	7.1637E-01	3.2093E+00	2.5972E+01	4.4388E+02	1.2530E+03	3.3820E+03	8.4540E+03
	1.5891E-01	4.5381E-01	3.2248E+00	7.1331E+01	9.5032E+01	1.0487E+02	1.3826E+02
S4	1.3068E+00	5.7532E+00	3.6224E+01	4.6861E+02	1.3892E+03	3.3587E+03	8.4437E+03
	2.7238E-01	8.7865E-01	4.5645E+00	6.8434E+01	9.4581E+01	1.1588E+02	1.4770E+02
S5	6.3861E-01	3.7063E+00	3.3372E+01	4.3175E+02	1.4001E+03	3.2277E+03	8.4341E+03
	1.2985E-01	4.8544E-01	1.0046E+01	6.8829E+01	9.9559E+01	1.1326E+02	1.4707E+02
S6	6.8659E-01	3.5819E+00	2.1967E+01	2.3738E+02	1.1404E+03	3.1169E+03	8.2199E+03
	1.3656E-01	4.4891E-01	1.9833E+00	4.7557E+01	1.1524E+02	1.3281E+02	1.6253E+02

Table 8
Performance results for the Rosenbrock function (95% Confidence)

PSO Setting	Dimensions						
Number	3	5	10	20	30	50	100
S1	5.9914E-05	2.9397E-01	1.6186E+00	1.4356E+01	3.1660E+01	8.5061E+01	5.0179E+02
	1.0737E-05	1.9969E-01	3.2920E-01	2.3676E+00	3.7055E+00	7.4577E+00	2.3399E+01
S2	1.2539E-05	1.4981E-02	1.0967E+00	1.3527E+01	2.7546E+01	4.8617E+01	9.8632E+01
	1.2683E-05	1.6305E-03	2.0057E-01	3.1249E-01	2.4295E-01	1.7350E-02	1.3002E-02
S3	1.0036E-02	5.4767E-02	1.0491E+00	1.3609E+01	2.7305E+01	4.8567E+01	9.8491E+01
	1.9406E-02	5.7616E-02	1.7471E-01	3.1097E-01	2.6092E-01	1.5984E-02	2.0580E-02
S4	1.8532E-01	6.0284E-01	3.9097E+00	1.5911E+01	2.7918E+01	4.8572E+01	9.8481E+01
	2.4569E-01	1.8583E-01	6.9819E-01	5.5094E-01	2.3426E-01	1.7005E-02	1.7343E-02
S5	5.6686E-04	1.4276E-02	1.0364E+00	1.4968E+01	2.7178E+01	4.8562E+01	9.8490E+01
	9.6144E-04	1.3379E-03	1.7975E-01	3.0991E+00	2.6138E-01	1.8227E-02	2.2103E-02
S6	4.5988E-06	1.4027E-02	9.4917E-01	1.7526E+01	2.6574E+01	4.9998E+01	1.1241E+02
	8.4955E-07	1.3283E-03	2.0617E-01	4.6065E+00	2.9127E-01	2.9482E+00	2.7698E+01

Table 9
Performance results for the Salomon function (95% Confidence)

PSO Setting	Dimensions						
Number	3	5	10	20	30	50	100
S1	8.5891E-02	9.9873E-02	1.8687E-01	5.9887E-01	1.7264E+00	1.2151E+01	7.1723E+01
	6.9101E-03	2.6071E-17	1.2502E-02	1.5851E-01	5.4659E-01	2.3483E+00	3.4537E+00
S2	8.2895E-02	1.0387E-01	1.9087E-01	1.7390E+00	1.2713E+01	1.1863E+02	2.4791E+02
	7.4806E-03	3.9074E-03	1.3842E-02	1.0444E+00	3.9640E+00	5.4919E+00	2.4937E+00
S3	8.5891E-02	1.0187E-01	2.0787E-01	1.7404E+00	7.8842E+00	6.7673E+01	1.9478E+02
	6.9101E-03	2.7916E-03	2.7727E-02	1.1174E+00	2.5984E+00	7.0808E+00	5.8066E+00
S4	1.7229E-01	4.0948E-01	7.9776E-01	4.6223E+00	1.6995E+01	6.9386E+01	1.9748E+02
	2.9918E-02	9.2939E-02	2.1866E-01	1.3263E+00	3.2231E+00	5.8831E+00	5.3992E+00
S5	7.5283E-02	1.0487E-01	2.3990E-01	2.0119E+00	7.6752E+00	6.0843E+01	1.8912E+02
	8.5247E-03	4.3458E-03	6.6179E-02	2.5006E+00	2.3423E+00	6.4858E+00	4.7252E+00
S6	7.3920E-02	1.0287E-01	1.8887E-01	6.8493E-01	6.6100E+00	3.8604E+01	1.7709E+02
	8.7308E-03	3.4015E-03	1.4894E-02	2.8430E-01	2.6819E+00	5.7192E+00	5.9854E+00

Table 10
Performance results for the Spherical function (95% Confidence)

PSO Setting	Dimensions						
Number	3	5	10	20	30	50	100
S1	0.0000E+00	0.0000E+00	0.0000E+00	1.4482E-17	4.2009E-07	1.7560E+00	2.8495E+02
	0.0000E+00	0.0000E+00	0.0000E+00	2.8056E-17	6.3320E-07	2.4718E+00	4.1050E+01
S2	3.9128E-30	2.4392E-29	6.0269E-29	1.4824E-01	2.3448E+01	7.5088E+02	5.9384E+03
	5.9770E-31	2.2961E-30	1.3411E-29	2.0725E-01	2.3473E+01	1.3413E+02	1.3289E+02
S3	4.7016E-30	2.3445E-29	4.0926E-29	2.6403E-02	2.6110E+00	2.8237E+02	4.5005E+03
	8.1793E-31	1.9214E-30	2.8578E-30	5.2382E-02	2.5978E+00	7.6116E+01	1.7765E+02
S4	3.6544E-06	6.5487E-04	8.2435E-03	2.3232E+00	1.6394E+01	3.8002E+02	4.6468E+03
	4.2354E-06	4.6760E-04	7.0895E-03	2.9425E+00	1.0968E+01	7.4605E+01	1.8386E+02
S5	3.3763E-30	1.9185E-29	3.6130E-29	8.5724E-18	9.6292E-01	2.9769E+02	4.5780E+03
	2.0398E-31	5.3151E-31	2.1639E-30	1.1671E-17	1.4923E+00	8.1270E+01	1.7291E+02
S6	3.2817E-30	1.8964E-29	3.2249E-29	3.6193E-05	4.7194E+01	2.9913E+02	4.1586E+03
	1.7617E-31	6.2604E-32	6.5133E-31	7.1807E-05	4.2743E+01	1.2553E+02	2.0512E+02

(a) (b)

Fig. 7. Mean (a) and best (b) fitness values for Ackley function in 3 dimensions.

(a) (b)

Fig. 8. Mean (a) and best (b) fitness values for Ackley function in 5 dimensions.

(a) (b)

Fig. 9. Mean (a) and best (b) fitness values for Ackley function in 10 dimensions.

(a) (b)

Fig. 10. Mean (a) and best (b) fitness values for Ackley function in 20 dimensions.

(a) (b)

Fig. 11. Mean (a) and best (b) fitness values for Ackley function in 30 dimensions.

(a) (b)

Fig. 12. Mean (a) and best (b) fitness values for Ackley function in 50 dimensions.

(a) (b)

Fig. 13. Mean (a) and best (b) fitness values for Ackley function in 100 dimensions.

(a) (b)

Fig. 14. Mean (a) and best (b) fitness values for Rastrigin function in 3 dimensions.

(a) (b)

Fig. 15. Mean (a) and best (b) fitness values for Rastrigin function in 5 dimensions.

(a) (b)

Fig. 16. Mean (a) and best (b) fitness values for Rastrigin function in 10 dimensions.

(a) (b)

Fig. 17. Mean (a) and best (b) fitness values for Rastrigin function in 20 dimensions.

(a) (b)

Fig. 18. Mean (a) and best (b) fitness values for Rastrigin function in 30 dimensions.

(a) (b)

Fig. 19. Mean (a) and best (b) fitness values for Rastrigin function in 50 dimensions.

(a) (b)

Fig. 20. Mean (a) and best (b) fitness values for Rastrigin function in 100 dimensions.

Table 11
Results for the eigenvector function (95% Confidence) for $\alpha = 0.0$, $\beta = 1.0$

Seed	Dimensions	Cartesian PSO	Polar PSO		
			Polar Mode	Random Mode	Boundary Mode
1000	10	4.1860E-03	2.3556E-03	2.9117E-16	**2.3951E-16**
		8.3049E-03	2.7357E-03	8.7004E-17	**2.4437E-17**
	20	2.0299E-02	2.7226E-02	1.8283E-02	**1.7394E-02**
		8.7276E-03	1.2748E-02	7.8985E-03	**1.5622E-02**
	30	3.7509E-02	6.4655E-02	**3.5889E-02**	6.1059E-02
		1.4421E-02	1.8489E-02	**1.3793E-02**	2.4258E-02
	40	**2.8304E-02**	5.0193E-02	5.8064E-02	1.0390E-01
		1.1317E-02	1.5373E-02	1.9190E-02	3.3909E-02
	50	**2.6903E-02**	5.0193E-02	6.6596E-02	1.4360E-01
		1.0053E-02	1.5373E-02	1.9133E-02	3.2415E-02
	100	**2.0132E-01**	3.8411E-01	3.8225E-01	4.0434E-01
		1.4305E-02	1.9070E-02	2.3275E-02	2.5463E-02

Table 12
Results for the eigenvector function (95% Confidence) for $\alpha = 0.0$, $\beta = 1.0$

Seed	Dimensions	Cartesian PSO	Polar PSO		
			Polar Mode	Random Mode	Boundary Mode
5000	10	1.9810E-02	**1.7334E-02**	2.2287E-02	1.9810E-02
		1.3395E-02	**1.2598E-02**	1.4131E-02	1.3395E-02
	20	1.2026E-02	1.2647E-02	**4.4800E-03**	2.4268E-02
		6.7315E-03	7.1128E-03	**3.8262E-03**	1.5784E-02
	30	2.0533E-02	1.9760E-02	**1.9213E-02**	4.4925E-02
		9.4130E-03	1.2917E-02	**1.3405E-02**	2.6886E-02
	40	**2.2588E-02**	2.5762E-02	3.9557E-02	8.3531E-02
		1.0615E-02	1.0261E-02	2.1061E-02	3.1630E-02
	50	**4.3604E-02**	6.0571E-02	7.6254E-02	1.2968E-01
		1.5272E-02	1.4275E-02	2.1009E-02	3.4310E-02
	100	**1.8873E-01**	3.4526E-01	4.0692E-01	4.1676E-01
		1.5090E-02	1.8452E-02	2.5726E-02	2.7672E-02

Table 13
Results for the eigenvector function (95% Confidence) for $\alpha = 0.0$, $\beta = 1.0$

Seed	Dimensions	Cartesian PSO	Polar PSO		
			Polar Mode	Random Mode	Boundary Mode
10000	10	2.3848E-02	2.1512E-02	**1.3405E-02**	1.8207E-02
		8.7415E-03	9.6211E-03	**6.6356E-03**	7.9909E-03
	20	**1.3681E-03**	1.1266E-02	7.1368E-03	1.3784E-02
		1.0297E-03	1.1760E-02	8.1060E-03	1.2113E-02
	30	1.8492E-02	2.2698E-02	**1.4978E-02**	5.2538E-02
		7.7146E-03	1.1003E-02	**9.1301E-03**	2.6387E-02
	40	**3.3897E-02**	6.1326E-02	4.6534E-02	9.6507E-02
		1.2746E-02	1.9072E-02	1.7845E-02	4.1353E-02
	50	**3.6867E-02**	8.5597E-02	6.2310E-02	1.3576E-01
		1.3234E-02	1.9147E-02	1.8952E-02	3.6901E-02
	100	**3.6867E-02**	8.5597E-02	6.2310E-02	1.3576E-01
		1.3234E-02	1.9147E-02	1.8952E-02	3.6901E-02

Table 14
Results for the eigenvector function (95% Confidence) for $\alpha = -1.0$, $\beta = 1.0$

Seed	Dimensions	Cartesian PSO	Polar PSO		
			Polar Mode	Random Mode	Boundary Mode
1000	10	**5.6356E-17**	2.4340E-16	2.4936E-16	2.5084E-16
		6.0559E-18	1.3031E-18	3.6825E-18	4.7195E-18
	20	3.7294E-08	6.7271E-12	**1.0782E-12**	8.9624E-03
		7.3538E-08	1.0277E-11	**1.0259E-12**	1.3002E-02
	30	5.2497E-05	**7.1351E-06**	8.1647E-03	3.1170E-02
		6.3831E-05	**5.0518E-06**	1.1424E-02	2.3983E-02
	40	**1.4505E-03**	5.0548E-03	7.2093E-03	5.8142E-02
		9.6928E-04	5.1709E-03	6.5170E-03	2.9186E-02
	50	**9.7062E-03**	2.5944E-02	4.5194E-02	1.3262E-01
		3.1056E-03	9.3444E-03	1.6963E-02	3.8288E-02
	100	**1.5725E-01**	3.5267E-01	4.6498E-01	4.6887E-01
		9.6010E-03	2.1217E-02	2.7096E-02	3.1190E-02

Table 15
Results for the eigenvector function (95% Confidence) for $\alpha = -1.0$, $\beta = 1.0$

Seed	Dimensions	Cartesian PSO	Polar PSO		
			Polar Mode	Random Mode	Boundary Mode
5000	10	**4.9441E-17**	2.2204E-16	2.4971E-16	2.2410E-16
		6.8835E-18	5.1318E-32	3.3950E-18	1.6808E-18
	20	9.2531E-12	**1.6965E-12**	1.7721E-12	1.5141E-02
		8.6386E-12	**1.2695E-12**	1.0643E-12	1.5430E-02
	30	1.2571E-04	**1.7009E-06**	2.7456E-03	3.6330E-02
		1.1003E-04	**6.4499E-07**	5.4233E-03	2.4597E-02
	40	3.4782E-03	**1.1849E-03**	1.3456E-02	5.7214E-02
		2.7275E-03	**7.3123E-04**	1.2559E-02	3.2035E-02
	50	**1.0916E-02**	2.0723E-02	4.3335E-02	1.9858E-01
		4.0822E-03	6.3167E-03	1.8697E-02	4.6962E-02
	100	**1.6976E-01**	3.5468E-01	4.4065E-01	4.7312E-01
		9.2625E-03	2.1266E-02	2.7846E-02	3.1420E-02

Table 16
Results for the eigenvector function (95% Confidence) for $\alpha = -1.0$, $\beta = 1.0$

Seed	Dimensions	Cartesian PSO	Polar PSO		
			Polar Mode	Random Mode	Boundary Mode
10000	10	**5.0569E-17**	2.2374E-16	2.3751E-16	4.5061E-03
		6.9136E-18	1.7538E-18	4.9979E-18	8.9401E-03
	20	5.5674E-10	**9.2701E-13**	7.5148E-12	3.9398E-03
		1.0478E-09	**4.8142E-13**	9.3831E-12	7.8166E-03
	30	5.7262E-05	2.5624E-06	**2.4886E-06**	6.3704E-03
		5.4529E-05	1.2484E-06	**1.1786E-06**	9.0245E-03
	40	1.3039E-03	**1.2746E-03**	9.3630E-03	4.4826E-02
		5.0892E-04	**5.5057E-04**	9.5705E-03	2.4732E-02
	50	**8.4320E-03**	2.0034E-02	4.6414E-02	1.1549E-01
		2.0571E-03	5.0830E-03	1.9575E-02	3.8421E-02
	100	**1.6614E-01**	3.4307E-01	4.2897E-01	4.6495E-01
		9.4576E-03	1.7428E-02	2.9183E-02	3.0485E-02

(a)(b)

Fig. 21. Mean fitness values for eigenvector function in 10 (a) and 20 (b) dimensions.

(a)(b)

Fig. 22. Mean fitness values for eigenvector function in 30 (a) and 40 (b) dimensions.

(a)(b)

Fig. 23. Mean fitness values for eigenvector function in 50 (a) and 100 (b) dimensions.

References

[1] R. Storn and K. Price, “Differential Evolution - A Simple and Efficient Heuristic for global Optimization over Continuous Spaces,” *Journal of Global Optimization*, vol. 11, no. 4, pp. 341–359, Dec. 1997.

[2] J. Kennedy and R. C. Eberart, “Particle Swarm Optimization,” in *Proc. IEEE International Conference on Neural Networks*, vol. 4, Nov. 1995, pp. 1942–1948.

[3] J. Kennedy and R. C. Eberhart, “A discrete binary version of the particle swarm algorithm,” in *Proc. IEEE International Conference on Computational Cybernetics and Simulation*, vol. 5, Oct. 1997, pp. 4104–4108.

[4] G. Pampara, N. Franken, and A. P. Engelbrecht, “Combining particle swarm optimisation with angle modulation to solve binary problems,” in *Proc. IEEE Congress on Evolutionary Computation*, vol. 1, Sept. 2005, pp. 89–96.

[5] G. Pampara, A. P. Engelbrecht, and N. Franken, “Binary Differential Evolution,” in *Proc. IEEE Congress on Evolutionary Computation*, July 2006, pp. 1873–1879.

[6] A. Sierra and A. Echeverría, “The Polar Evolution Strategy,” in *Proc. IEEE Congress on Evolutionary Computation*, July 2006, pp. 2301–2306.

[7] D. Srinivasan and T. H. Seow, “Particle swarm inspired evolutionary algorithm (PS-EA) for multiobjective optimization problems,” in *Proc. IEEE Congress on Evolutionary Computation*, vol. 4, Dec. 2003, pp. 2292–2297.

[8] M. G. Kendall, *A Course in the Geometry of n Dimensions.* Dover Publications, 2004.

[9] W.-J. Zhang, X.-F. Xie, and D.-C. Bi, “Handling boundary constraints for numerical optimization by particle swarm flying in periodic search space,” in *Proc. IEEE Congress on Evolutionary Computation*, vol. 2, June 2004, pp. 2307–2311.

[10] S. Helwig and R. Wanka, "Particle Swarm Optimization in High-Dimensional Bounded Search Spaces," in *Proc. IEEE Swarm Intelligence Symposium*, Apr. 2007, pp. 198–205.

[11] F. van den Bergh, "An Analysis of Particle Swarm Optimizers," Ph.D. dissertation, Department of Computer Science, University of Pretoria, South Africa, 2002.

[12] R. Eberhart and Y. Shi, "Comparing inertia weights and constriction factors in particle swarm optimization," in *Proc. IEEE Congress on Evolutionary Computation*, vol. 1, July 2000, pp. 84–88.

[13] M. Matsumoto and T. Nishimura, "Mersenne twister: a 623-dimensionally equidistributed uniform pseudo-random number generator," *ACM Transactions on Modelling and Computer Simulation*, vol. 8, no. 1, pp. 3–30, 1998.