UNIVERSITEIT VAN PRETORIA
UNIVERSITY OF PRETORIA
YUNIBESITHI YA PRETORIA

A Study of Gradient Based Particle Swarm Optimisers

by

Daniel Barla-Szabó

Submitted in partial fulfillment of the requirements for the degree
Master of Science (Computer Science)
in the Faculty of Engineering, Built Environment and Information Technology
University of Pretoria, Pretoria

May 2010

Publication data:

Daniel Barla-Szabó. A Study of Gradient Based Particle Swarm Optimisers. Master's dissertation, University of Pretoria, Department of Computer Science, Pretoria, South Africa, March 2010.

UNIVERSITEIT VAN PRETORIA
UNIVERSITY OF PRETORIA
YUNIBESITHI YA PRETORIA

A Study of Gradient Based Particle Swarm Optimisers

by

Daniel Barla-Szabó
E-mail: danielbarla@gmail.com

Abstract

Gradient-based optimisers are a natural way to solve optimisation problems, and have long been used for their efficacy in exploiting the search space. Particle swarm optimisers (PSOs), when using reasonable algorithm parameters, are considered to have good exploration characteristics.

This thesis proposes a specific way of constructing hybrid gradient PSOs. Heterogeneous, hybrid gradient PSOs are constructed by allowing the gradient algorithm to optimise local best particles, while the PSO algorithm governs the behaviour of the rest of the swarm. This approach allows the distinct algorithms to concentrate on performing the separate tasks of exploration and exploitation. Two new PSOs, the Gradient Descent PSO, which combines the Gradient Descent and PSO algorithms, and the LeapFrog PSO, which combines the LeapFrog and PSO algorithms, are introduced. The GDPSO represents arguably the simplest hybrid gradient PSO possible, while the LeapFrog PSO incorporates the more sophisticated LFOP1(b) algorithm, exhibiting a heuristic algorithm design and dynamic time step adjustment mechanism. The strong tendency of these hybrids to prematurely converge is examined, and it is shown that by modifying algorithm parameters and delaying the introduction of gradient information, it is possible to retain strong exploration capabilities of the original PSO algorithm while also benefiting from the exploitation of the gradient algorithms.

Keywords: Artificial intelligence, particle swarm optimisers, gradient methods, hybrid.
Supervisor : Prof. A. P. Engelbrecht
Department : Department of Computer Science
Degree : Master of Science

Acknowledgements

I would like to thank the following people:

- My supervisor, Prof A.P. Engelbrecht for his support, knowledge and direction.
- My family, friends, and wonderful wife Rita, who supported me during the writing of the thesis.
- Dr. Ian Kennedy and my brother, Gabor, for language editing, proof-reading and generally helping with ideas when stuck.
- The members of the CILib project, especially Edwin Peer, whose work heavily influenced my own approach to software development.

Contents

List of Figures

List of Tables

List of Algorithms

Chapter 1

Introduction

Optimisation is a concept taken for granted by many, without giving it a second thought. From the availability and speed of communication networks to the strategic placement of convenience stores, today's modern society has come to rely on optimisation to ensure that people can carry on with their lives efficiently. Featured in almost all aspects of human civilisation, optimisation has truly become both ubiquitous and critical.

In order to better define optimisation, consider the analogy of a racing team, attempting to fine-tune their car for a race. In racing, the goal of optimisation is relatively straight forward and well defined; namely configure the car to travel around a race track as quickly as possible. There are a myriad of settings on the race car (known as *variables* in the context of optimisation) which can be adjusted to accomplish this task, such as tyre width and compound, the size, angle and placement of aerodynamic wings, chassis height, and so on. This seemingly simple task hides much complexity. Even considering a single setting in isolation, such as the size of an aerodynamic wing, can be complex. In this case, an increase in the size of the wing will provide more downward force at the cost of higher drag on the vehicle. Worse still, some of the settings may influence other settings, and the problem can quickly become unmanageable. The traditional solution to this problem would be to employ race engineers, who have prior experience in the field. These experts could make intuitive guesses regarding the correct settings, perhaps resulting in a solution which is acceptable.

While many problems can be solved with the above human approach, a problem could be encountered for which no pool of human experts is available. Other problems may have such a large number of variables that it is unfeasible for humans to attempt to guess an acceptable solution. In these situations, it is possible to employ computer algorithms to do much of the work. In the above racing team analogy, it would be possible to create a computer simulation of the physics surrounding a race car. Given such a simulation, it would be possible for a computer to try different settings, typically orders of magnitude more rapidly and cost effectively than attempting to do the same with the actual race car.

1.1 Motivation

Particle swarm optimisers (PSOs) are just one of the many different types of optimisation algorithms available in the field of computational intelligence. They mimic the movement of groups of animals, such as a flock of birds, in the virtual space of the problem being optimised. Simple to implement, they are surprisingly apt at finding good solutions early on, a task referred to as exploration.

Gradient based algorithms are another class of optimisation algorithms. They make use of the direction of the largest improvement in the quality of solutions in order to obtain successive solutions. This direction is represented by the gradient of the problem. The process of repeatedly modifying a candidate solution aims to eventually end up with a solution which can no longer be improved upon using this method. Gradient algorithms are a natural way to solve simple optimisation problems, and they are generally regarded as being efficient at finding good solutions close to the solution that they start out with. This task is known as exploitation.

It seems to make sense to combine one algorithm which is good at exploration under certain conditions (PSO), with another which is good at exploitation (gradient algorithms). This idea is not entirely new; such hybrid algorithms exist, even given the relatively recent rise to popularity of PSOs. Examples of PSOs which are conceptually similar include the Hybrid Gradient Descent PSO (HGPSO) discussed in section 2.5.5, the Division of Labour PSO discussed in section 2.5.8, and arguably the two-step PSO discussed in section 2.5.4. The algorithm that is the closest conceptually is the HG-

PSO, which uses an actual gradient descent component in the velocity update equation, instead of the usual cognitive update. While the intent of the HGPSO is in the spirit of the original PSO, namely to balance the tasks of exploration and exploitation, it is quite different to the approach used in this dissertation to construct hybrid algorithms, making an investigation into other approaches worthwhile. Furthermore, this dissertation argues that by replacing portions of the original PSO velocity update equation and applying gradient search capability to all particles in the swarm, the approach used in the HGPSO may detract from the ability of the swarm to efficiently explore.

By separating the concern of individual particles within the swarm to that of exploration and exploitation, a *heterogeneous* swarm results. This approach is significantly different to most existing gradient hybrid PSOs, having perhaps more in common with the DoLPSO. This thesis intends to show that these heterogeneous swarms offer a natural and feasible way to introduce a second, local-search oriented algorithm into the PSO algorithm, resulting in retained exploration capability and increased exploitation capability.

1.2 Objectives

The objectives of this dissertation can be summarised as follows:

- Formally define a new, novel way to construct gradient hybrid PSOs.
- Present new and specific gradient hybrid PSO algorithms constructed using the new approach.
- Explore the performance characteristics of these hybrid algorithms.
- Attempt to correct or suggest corrections for the deficiencies of these algorithms.

1.3 Contributions

The dissertation provides the following contributions to the field of computational intelligence:

- A new approach to constructing gradient hybrid algorithms, formalised as the Generalised Local Search PSO (GLSPSO).
- The Gradient Descent PSO (GDPSO), which differs from existing hybrid PSOs which make use of the gradient descent algorithm in the way the algorithm is incorporated into PSO.
- The LeapFrog PSO, a combination of the LFOP1(b) and PSO algorithms which has never been done previously.
- A discussion on the nature of the problems resulting from the combination of the gradient and PSO algorithms, specifically premature convergence.
- Simple solutions to the problem of premature convergence by way of algorithm parameter adjustment, as well as delayed introduction of gradient information. Validation of these solutions using empirical experimentation.

1.4 Dissertation outline

The remainder of the dissertation is structured as follows:

- **Chapter 2** gives background information on the concept of optimisation, gradient optimisers, PSOs, hybrid gradient PSOs, as well as existing research that is relevant to the dissertation.
- **Chapter 3** examines the concept of constructing a gradient hybrid PSO. The Gradient Descent PSO (GDPSO), LeapFrog PSO (LFPSO), and a generalisation of the two, the Generalised Local Search PSO (GLSPSO), are formally described. An argument is put forward for the specific approach to constructing these hybrid PSOs. Several potential issues, such as premature convergence, and large variances in problem gradients, are discussed, and potential solutions are presented. Gradient estimation is briefly discussed.
- **Chapter 4** details the experimental methodology used to obtain empirical results. The benchmark functions used are clearly defined, and modifications used to the

functions, such as function shifting, is discussed. All algorithm and experimental assumptions are given, as well as the statistical method used to determine statistical significance of results.

- **Chapter 5** provides and discusses empirical results obtained via experimentation. By systematically providing a series of hypotheses, constructing experiments to validate them, and analysing the results of the experiments, this chapter aims to meet the performance related objectives of this dissertation.

- **Chapter 6** gives a conclusion to the dissertation. A summary of the experimental results, and the significance of each, is discussed. The initial objectives of the dissertation are revisited, and compared to the actual findings.

Chapter 2

Background

This chapter serves as an overview of optimisation problems and algorithms relevant to this thesis. First, the concept of optimisation is discussed in section 2.1. Two gradient-based optimisation algorithms, viz. gradient descent (section 2.2) and the LeapFrog algorithm (section 2.3), are examined. The particle swarm optimiser (PSO) is discussed in section 2.4, followed by a discussion of modifications to the basic PSO algorithm involving techniques such as mutation, gradient-based approaches and division of labour (section 2.5). The latter modifications and algorithms are deemed relevant due to conceptual similarities to the algorithms introduced in this dissertation, or in certain cases, due to their role as benchmark algorithms to compare against in experimental results. Section 2.6 gives conclusions about the background material presented.

2.1 Optimisation

This section provides a compact review of optimisation. Section 2.1.1 formalises the concept of optimisation. An illustration of minima is given in section 2.1.2.

2.1.1 Formal Definitions

Chapter 1 introduced the concept of optimisation with the race car analogy, however a more formal definition is required. An optimisation problem can be defined as follows:

The task of finding values for a set of variables, such that a defined measure of optimality is obtained and a set of constraints is satisfied.

Let A denote the set of all permissible solutions, also referred to as the *search space*. The *objective function*, denoted by f, is a measure of the quality of a given solution. The task is then to find a solution, denoted by $\mathbf{x}$, such that $f(\mathbf{x})$ is acceptable, and $\mathbf{x} \in A$. An algorithm which performs this task is known as an *optimisation algorithm*.

In the case of the analogy from chapter 1, $\mathbf{x}$ would represent a configuration of the car (i.e., all the individual settings which together define a possible configuration for the car), $f(\mathbf{x})$ would be a measurement of how fast the car completed a lap given the settings, and A would define all possible configurations for the car. The optimisation task is then to find a configuration, $\mathbf{x}$, which minimises the time, $f(\mathbf{x})$, the car takes to complete a lap.

Optimisation problems can be classified based on the following criteria:

- Problems where solutions must satisfy a set of constraints are known as *constrained* optimisation problems. *Unconstrained* problems typically only have boundary constraints, which define upper and lower limits for each of the variables being optimised.

- If $A \subseteq \mathbb{R}^n$, then the optimisation problem is said to be *continuous-valued*. *Discrete* optimisation problems are those where $A \subseteq \mathbb{Z}^n$. It is possible for an optimisation problem to involve solutions where a mixture of discrete- and continuous-valued variables are being optimised; these are referred to as *mixed integer* optimisation problems.

- Problems where the objective function can be expressed as a linear equation are known as *linear* optimisation problems. *Non-linear* problems involve a non-linear objective function, and are typically more difficult to solve.

- Considering regions of the search space which contain only a single "best" solution, it is possible to differentiate between problems that contain only a single such region and problems which exhibit multiple such regions. The former are referred to as *unimodal* whilst the latter are referred to as *multimodal*. Multimodal problems are typically far more complex.

- Problems where the quality of a solution is defined by a single objective function are referred to as *single-objective* problems. *Multi-objective* problems involve the optimisation of variables in order to find an acceptable solution to multiple objective functions at the same time.

- Optimisation problems where the goal is to find $\min(f(\mathbf{x}))$ are known as *minimisation problems*, whilst problems where the goal is to find $\max(f(\mathbf{x}))$ are known as *maximisation problems.*

- *Dynamic* optimisation problems exhibit search spaces which change with time. *Static* optimisation problems have search spaces which remain constant.

This thesis is primarily concerned with continuous-valued, nonlinear, static, single-objective function minimisation problems. We can see that $\max(f(\mathbf{x}))$ is equivalent to $\min(-f(\mathbf{x}))$, therefore no further distinction between minimisation and maximisation problems will be made in this chapter, and minimisation will be assumed unless otherwise stated. This subset of optimisation problems can be formally defined as:

$$\begin{aligned}&\text{Given } f : A \to \mathbb{R} \text{ where } A \in \mathbb{R}^n \\ &\text{find a solution } \mathbf{x}^* : f(\mathbf{x}^*) < f(\mathbf{x}),\ \forall \mathbf{x} \in A\end{aligned} \tag{2.1}$$

Solutions can be classified by their quality when compared to other solutions. In the case of function minimisation, a solution is referred to as a minimum if it is the best solution within a certain region B of the search space A. Minima can be classified as global or local, depending on the extent of the region B, and the latter can be further categorised into strong and weak local minima. The solution, $\mathbf{x}^*$, from definition 2.1 represents a global minimum, since it satisfies the criteria for global minima. Formal definitions for the different types of minima are given below.

Global minimum

The solution $\mathbf{x}^* \in A$ is a global minimum of f if

$$f(\mathbf{x}^*) < f(\mathbf{x}), \forall \mathbf{x} \in A \tag{2.2}$$

Thus global minima represent the best possible solutions out of the set of all feasible candidate solutions A. In contrast, local minima are the best solutions only within some subset of A. The distinction between strong and weak local minima is formally defined in equations (2.3) and (2.4):

Strong local minimum

The solution $\mathbf{x}_B^* \in B \subseteq A$ is a strong local minimum of f if

$$f(\mathbf{x}_B^*) < f(\mathbf{x}), \ \forall \mathbf{x} \in B \tag{2.3}$$

where B is a set of candidate solutions in the neighbourhood of $\mathbf{x}_B^*$.

Weak local minimum

The solution $\mathbf{x}_B^* \in B \subseteq A$ is a weak local minimum of f if

$$f(\mathbf{x}_B^*) <= f(\mathbf{x}), \ \forall \mathbf{x} \in B \tag{2.4}$$

where B is a set of candidate solutions in the neighbourhood of $\mathbf{x}_B^*$.

2.1.2 An Illustration of Minima

Figure 2.1 illustrates the concept of minima, as discussed in section 2.1.1. The function $f(x)$ is a one-dimensional, multimodal objective function. For the purpose of the illustration, it can be assumed that the entire domain A of f is shown. The horizontal axis represents values of x, while the vertical axis represents the quality of these solutions. The global minimum, x^* is readily identifiable as the lowest point of the objective function. The figure also contains two local minima, x_B^* and x_C^*, where B and C represent the regions immediately surrounding the minima. The leftmost minimum, x_B^*, can be classified as a strong local minimum, since as per definition 2.3, it is possible to identify a single point for which the value of the objective function f is lower than all the other points in its vicinity. The rightmost minimum, x_C^*, can be identified as a weak local minimum, because multiple points in the area C result in the same, locally lowest point for f.

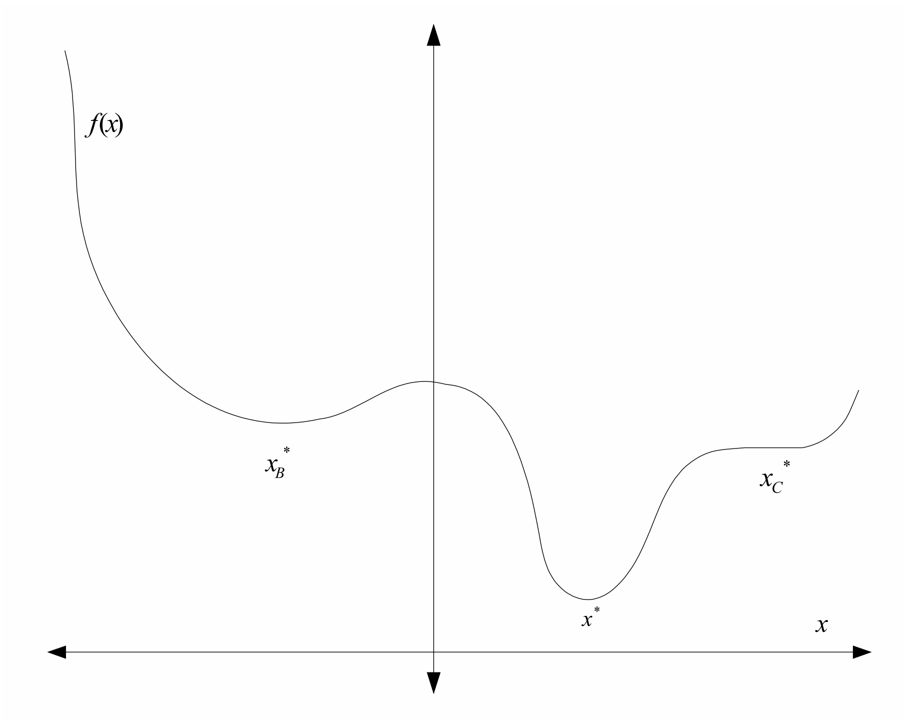

Figure 2.1: Global and local minima

2.2 Gradient Descent

This section describes the gradient descent optimisation algorithm. An overview is given in section 2.2.1, followed by a discussion of parameter selection issues (section 2.2.2). A graphical illustration of the algorithm is given in section 2.2.3.

2.2.1 Overview

Gradient descent (GD) [39] is one of the simplest algorithms for unconstrained optimisation, and can be defined as follows:

$$\mathbf{x}(t+1) = \mathbf{x}(t) - \delta \nabla f(\mathbf{x}(t)) \tag{2.5}$$

where $\mathbf{x}(t)$ represents a candidate solution at time step t, $\nabla f(\mathbf{x}(t))$ represents the gradient of the objective function f at the position represented by $\mathbf{x}(t)$, and δ is an arbitrary ratio, referred to as the *step size*.

The algorithm operates on the basis that for any given value of $\mathbf{x}$, the value of $f(\mathbf{x})$ increases most rapidly in the direction of the gradient, $\nabla f(\mathbf{x})$. Conversely, the direction of $-\nabla f(\mathbf{x})$ provides the most rapid decrease in the value $f(\mathbf{x})$; for this reason, gradient descent is sometimes also referred to as the *method of steepest descent*.

It follows that given an initial position $\mathbf{x}(0) \in B \subseteq A$, iterating equation (2.5) with sufficiently small values for δ will result in $\mathbf{x}(t) \approx \mathbf{x}^*_B$ for increasing values of t .

Algorithm 2.1 The Gradient Descent Algorithm

```
1: Initialise $\mathbf{x}(0) \in A$;
2: repeat
3:     $\mathbf{x}(t+1) = \mathbf{x}(t) - \delta \nabla f(\mathbf{x}(t))$;
4: until stopping condition is true
```

Gradient descent (refer to algorithm 2.1) can be used as an unconstrained optimisation algorithm where the gradient of f is known, or can be approximated. The algorithm's simplicity has lead to widespread use in optimisation, e.g. in the training of neural networks [19, 9, 61].

2.2.2 Parameter Selection

The basic gradient descent algorithm defines only one algorithm parameter, namely the step size, δ. The selection of a good value for δ is essential, because it influences the performance of the gradient descent algorithm greatly. Too small a value consumes too much computing time, while too large a value can result in the optimum point being missed. It should be noted that the choice of value for δ is highly problem dependent, as can be seen in the empirically determined values given later on in section 4.4.2, and specifically table 4.4 (where the optimal values for δ were determined using a standard GD algorithm).

Typically, parameters are selected during trials where two or more parameter selections are evaluated, and the results compared against each other. The outcomes of these trials serve as an indication of good values for δ for a given optimisation problem.

There are a number of approaches to parameter selection, examples of which would be constant values, linearly decreasing values and dynamic methods. A short overview of each approach is given below.

Constant values for δ

In this approach to setting the parameter δ, the value for δ is set once, and never changes. Assigning a constant value for δ is the simplest option, but can result in other complications:

- Large values usually allow for rapid improvement early on, but run the risk of overshooting a local minimum, and tend to produce less accurate results.
- Small values tend to produce more accurate results, but take proportionally longer to find the local minimum. They are also more likely to get trapped in local minima.

Decreasing values for δ

The parameter δ can also be set by starting with a given value for δ, and defining a scheme whereby it is decreased after each iteration [35], e.g.:

$$\delta(t+1) = \delta(t) - \gamma \tag{2.6}$$

where $\gamma \in \mathbb{R}$ is an arbitrary, small value and $\delta(t) > \gamma$ so as to ensure that δ remains positive. Updating δ thus after each iteration would result in δ decreasing after each iteration, with the aim being to balance benefits of having a large value for δ early on, and having a small value as time goes on and focus shifts from exploration to exploitation.

These methods, although often providing better results than with constant values, do still suffer from various issues:

- A value for γ needs to be defined, which in effect attempts to estimate the desired number of steps for the algorithm to find a solution.

- Many objective functions exhibit a landscape such that a linearly decreasing value for δ may not be the most appropriate choice, since the scheme does not cater for increasing δ.

Dynamic methods for δ

More complex approaches can be created for managing the choice of δ, such as the dynamic time step control mechanism employed in the LeapFrog algorithm [54], which is discussed in detail in section 2.3. It should be noted that while the LeapFrog algorithm differs from gradient descent, the concept of the time step variable Δt is analogous to the GD parameter δ. These approaches are usually aimed at addressing the shortcomings of the above approaches, or increasing the level of autonomy of the algorithm, which the use of additional algorithm parameters tend to detract from. In the case of the LeapFrog algorithm, the dynamic method employed is able to both increase or decrease the value that algorithm's equivalent of δ, dictated by certain heuristic decisions based on the recent history of the algorithm's position updates.

2.2.3 An Illustration of the Gradient Descent Algorithm

Figure 2.2 is a graphical illustration of the constant step size version of the gradient descent algorithm applied to minimising the one-dimensional version of the spherical function, $f(x) = x^2$. The spherical function is unimodal, consisting of a single global minimum, and no local minima. Since $\nabla f(x) = f'(x) = 2x$, it is apparent that applying the gradient descent position update for any value of $x(t)$ will result in $x(t+1)$ moving closer to $x^* = 0$, as long as δ is small enough. It is also worthwhile noting that the change in x is determined by not only δ, but also the steepness of the gradient at x, since the gradient is typically not normalised.

Since the combined effect of δ and the steepness of the gradient may result in a positional update which overshoots x^*, it is not possible to deduce that $f(x(t+1)) <= f(x(t))$. Algorithms which do guarantee this property are known as *hill climbing* or *hill*

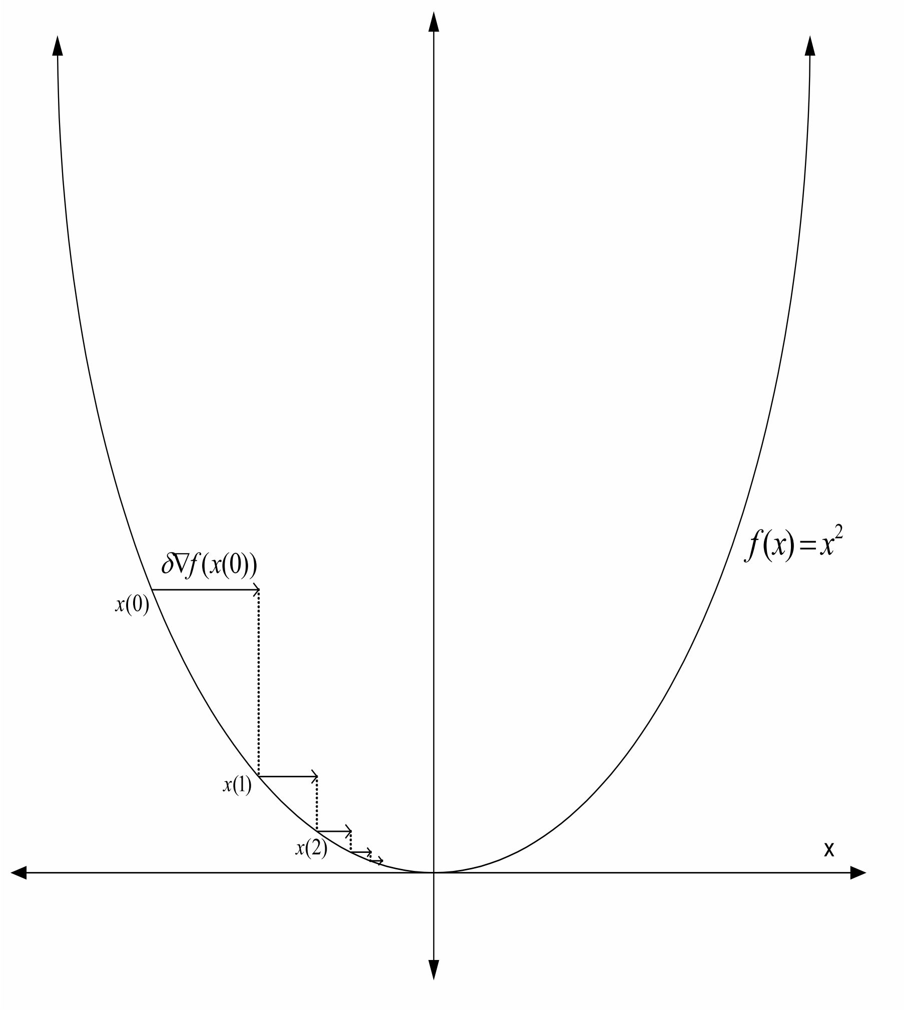

Figure 2.2: A geometric illustration of the gradient descent algorithm

descent algorithms, depending on the context of problem maximisation or minimisation. In both cases, the terms imply a an algorithm which guarantees that the solution to the objective function at each subsequent step does not deteriorate. Although gradient descent is strictly speaking not a hill descent algorithm, it does exhibit extremely good local optimisation characteristics.

Gradient-based algorithms, such as gradient descent, typically perform extremely well on unimodal functions, but often get trapped by local minima on multimodal functions, where they tend to find the best solution in the localised area where the algorithm is started.

2.3 The LeapFrog Algorithm

In his original 1982 paper [52], Snyman proposed a new gradient method for unconstrained optimisation, which eventually became known as the LeapFrog algorithm. Snyman's algorithm views the search space of the problem as a force field, through which a particle moves. The algorithm attempts to predict the trajectory of the particle using established methods from the field of molecular dynamics, thus effectively finding a locally optimal solution. An improved version of the algorithm, known as LFOP1(b), was proposed in [54]. The improved version included a mechanism for dynamically adjusting the time-step of the algorithm, which enabled the algorithm to be effective on a wider range of problems.

The LeapFrog algorithm belongs to a class of algorithms which are referred to as Quasi-Newton methods. The application of Newton's method to function minimisation involves the observation that local minima occur where $\nabla f(\mathbf{x}^*) = f'(\mathbf{x}^*) = 0$. Therefore if $\mathbf{x}^*$ is a local minimum, then it is also a root of $f'(\mathbf{x})$, and as such one can solve for it by applying Newton's method [13, 25]. In contrast with Newton's Method [48], Quasi-Newton methods [12] approximate the Hessian matrix of $f''(\mathbf{x})$ by observing the gradient in sequential time steps.

An overview of the LeapFrog algorithm is given in section 2.3.1, followed by a discussion on parameter selection in section 2.3.2. A summary is given in section 2.3.3.

2.3.1 Overview of the LeapFrog Algorithm

Consider the minimisation problem specified in equation (2.1), namely to minimise $f(\mathbf{x})$, where $\mathbf{x} = (x_1, x_2, ..., x_n)^T \in \mathbb{R}^n$. Suppose that $\mathbf{x}$ represents the position of a particle of unit mass in a conservative force field and $f(\mathbf{x})$ represents the potential energy of the particle. By conservation of energy, it follows that:

$$f(\mathbf{x}) = -\int_{\mathbf{x}^*}^{\mathbf{x}} a(s)^T \mathrm{d}s + f(\mathbf{x}^*) \tag{2.7}$$

where $a(s)$ represents the force on the particle at each point s in the force field. The kinetic energy of the particle is:

$$K(\mathbf{x}) = \frac{1}{2}\sum_{i=1}^{n} \dot{\mathbf{x}}_i^2 = \frac{1}{2}||\dot{\mathbf{x}}||^2 \tag{2.8}$$

where $\dot{\mathbf{x}}$ is the shorthand for $\frac{\mathrm{d}x}{\mathrm{d}t}$, the derivative of $\mathbf{x}$ with respect to time. The Lagrangian can be written as:

$$L(\mathbf{x}) = K(\mathbf{x}) - f(\mathbf{x}) \tag{2.9}$$

The Euler-Lagrange equation is:

$$\frac{\mathrm{d}}{\mathrm{d}t}\left(\frac{\partial L}{\partial \dot{x}_i}\right) - \frac{\partial L}{\partial x_i} = 0 \tag{2.10}$$

where $i = 1, ..., n$. Substitution of equations (2.7) and (2.8) into equation (2.9) and subsequently into equation (2.10) yields:

$$\ddot{\mathbf{x}} = -\nabla f = a \tag{2.11}$$

where $\ddot{\mathbf{x}}$ is the second-order derivative of $\mathbf{x}$ with respect to time, i.e. $\frac{\mathrm{d}^2\mathbf{x}}{\mathrm{d}t^2}$. Snyman equates $\ddot{\mathbf{x}}$ to a new variable, a, presumably a shortening for *acceleration* (since it represents the second order derivative), and primarily serving to differentiate from $\mathbf{x}$ in Snyman's FORTRAN code listing. This new variable is used in parts of the algorithm description in [52], and will be the notation used in this summary in turn. Given the following initial conditions,

$$\begin{aligned} \mathbf{x}(0) &= \mathbf{x}_0 \\ \dot{\mathbf{x}}(0) &= \mathbf{v}(0) = \mathbf{v}_0 = 0 \end{aligned} \tag{2.12}$$

and the fact that the force field is conservative,

$$K(\mathbf{x}) + f(\mathbf{x}) = K(\mathbf{x}_0) + f(\mathbf{x}_0) = E_0 \tag{2.13}$$

where E_0 is the total energy at time step 0. Since $K(\mathbf{x}_0) = 0$, it is apparent that along the path of the particle, $f(\mathbf{x}) \leq f(\mathbf{x}_0)$.

Snyman [52] observed that in the absence of frictional forces, the particle will be in constant motion, and will not necessarily pass through $\mathbf{x}^*$. In order to ensure convergence of the particle, Snyman proposed an interfering strategy, whereby the energy of the particle is systematically reduced. A damping term is considered, whereby:

$$\ddot{\mathbf{x}} = -\nabla f - \alpha \dot{\mathbf{x}} \tag{2.14}$$

This approach, however, is found to be too dependent on an appropriate choice of the damping coefficient α, which brings about complexities similar to the choice of δ in the gradient descent algorithm. Instead, an alternative approach was subsequently proposed by Snyman [52]. Consider that by conservation of energy, the following relation holds:

$$\begin{aligned} \Delta f_t &= f(\mathbf{x}(t+1)) - f(\mathbf{x}(t)) \\ &= -K(\mathbf{x}(t+1)) + K(\mathbf{x}(t)) \\ &= -\Delta K(t) \end{aligned} \tag{2.15}$$

If $\Delta K(\mathbf{x}(t)) > 0$, then $\Delta f(\mathbf{x}(t)) < 0$, which is desirable. Snyman proposed [52] the following strategy to be implemented at each time step t:

$$\begin{aligned} &\text{Monitor } ||\mathbf{v}(t+1)|| \\ &\text{if and only if } ||\mathbf{v}(t+1)|| \leq ||\mathbf{v}(t)||; \\ &\text{set } \mathbf{v}(t) = 0 \end{aligned} \tag{2.16}$$

where the notation $||\mathbf{v}(t+1)||$ and $||\mathbf{v}(t)||$ refers to the Euclidean length of the vectors $\mathbf{v}(t+1)$ and $\mathbf{v}(t)$, respectively. Snyman states that the success of the algorithm rests on the selection of an integration method which is accurate and stable, but also retains the conservation of energy property. An integration mechanism known as the 'leap-frog' method, found by Greenspan to be stable [24], is employed. The entire optimisation algorithm would eventually come to be called the 'LeapFrog' algorithm, abbreviated to LFOP, after the choice of the integration method, which can be defined as:

$$\begin{aligned}\mathbf{x}(t+1) &= \mathbf{x}(t) + \mathbf{v}(t)\Delta t \\ \mathbf{v}(t+1) &= \mathbf{v}(t) + \mathbf{a}(t+1)\Delta t\end{aligned} \tag{2.17}$$

where $\mathbf{a}(t) = -\nabla f(\mathbf{x}(t))$.

Having established a basic algorithm, Snyman proposed [52] several heuristic arguments to enhance the behaviour of the algorithm. Consider the strategy proposed in equation (2.16). Snyman argued that by setting $\mathbf{v}(t) = 0$, information regarding the correct search direction is lost, resulting in slower convergence. In order to decrease velocity, but maintain the search direction, the following empirically established approach is adopted:

$$\mathbf{v}(t) = (\mathbf{v}(t) + \mathbf{v}(t+1))/4 \tag{2.18}$$

Furthermore, empirical results showed that a similar argument could be made for setting $\mathbf{x}(t+1) = \mathbf{x}(t)$, resulting in the following approach for restarts:

$$\mathbf{x}(t+1) = (\mathbf{x}(t) + \mathbf{x}(t+1))/2 \tag{2.19}$$

In order to prevent a scenario where the particle continues to climb uphill, Snyman proposed [52] that $\mathbf{v}(t)$ be set to 0, and $\mathbf{x}(t+1) = (\mathbf{x}(t)+\mathbf{x}(t+1))/2$ if $||\mathbf{v}(t+1)|| \leq ||\mathbf{v}(t)||$ for more than two consecutive time steps. Thereafter, equation (2.18) is employed only on the first occasion when $||\mathbf{v}(t+1)|| \leq ||\mathbf{v}(t)||$.

In certain areas of the objective function f, the gradient could be very large, resulting in large changes in the position of the particle. Extremely large changes in position are typically not desirable and Snyman proposed the following approach, effectively requiring:

$$||\Delta\mathbf{x}(t)|| \leq \delta \tag{2.20}$$

where δ is an algorithm parameter, specifying the maximum allowed step size. The algorithm is regarded as having converged when the following condition holds:

$$||\nabla f(t) = ||\mathbf{a}(t)|| \leq \epsilon \tag{2.21}$$

where $\epsilon \in \mathbb{R}$ is a small, positive algorithm parameter.

The above definition became known as *LFOP1*, or version 1 of the LeapFrog algorithm. Subsequently, Snyman developed a second version of the algorithm [53] known as *LFOP2*, which was constructed with less reliance on heuristic arguments. Despite this, Snyman found that the performance of the original LFOP1 was surprisingly good and proposed further changes to the algorithm in [54].

In 1982, Snyman pointed out that the effectiveness of LFOP1 relied heavily on the selection of an appropriate value for the time step parameter, Δt [52]. The original paper proposed one possible solution, built around the heuristic argument that a number of steps taken at the maximum allowed step size δ would lead to inaccuracy. This original suggestion involved additional an algorithm parameter, M, which would dictate the number of consecutive steps of δ in size which would be allowed before a reduction was enforced. To cater for rapid exploitation of relatively flat areas, another parameter, N, would serve as a means to limit the number of times this reduction itself was enforced, essentially allowing for the algorithm to disable the limit if it again accelerated past δ. While this provided a dynamic way to manage Δt, it was not until [54] that Snyman was satisfied with the algorithm. The new algorithm, LFOP1(b), proposes the following heuristic argument for managing Δt:

Large values of Δt result in a trajectory which is inaccurate when compared to the theoretical true trajectory of the particle. The closer the particle is to $\mathbf{x}^*$ the more important an accurate trajectory becomes to ensure convergence. By considering the angle between two successive gradient directions, $\mathbf{a}(t)$ and $\mathbf{a}(t-1)$, the algorithm has some measure of the inaccuracy. Specifically, if the angle between two successive gradient directions is greater than 90°, Δt has become very large, and should be reduced. Reducing Δt at the first occurrence of the above condition, however, might be premature, and lead to slower convergence. Snyman proposed [54] the following approach:

$$\begin{aligned}
&\text{At each time step } t \text{ compute } \mathbf{a}^T(t)\mathbf{a}(t-1)\\
&\text{If the quantity is greater than 0 for } m \text{ consecutive steps, then}\\
&\Delta t = \Delta t/2, \text{ and}\\
&\text{restart with position } (\mathbf{x}(t)+\mathbf{x}(t-1))/2 \text{ and}\\
&\text{velocity } (\mathbf{v}(t)+\mathbf{v}(t-1))/4.
\end{aligned} \tag{2.22}$$

Snyman suggested [54] using the parameter setting $m = 3$, however, the value for m can be adjusted for different problems, although no empirical evidence was provided to evaluate the result of such adjustments. The algorithm also caters for excessive reduction in step size by magnifying Δt at each successful step, where a successful step is defined as a step where $\mathbf{a}^T(t)\mathbf{a}(t-1) > 0$ and $||\Delta\mathbf{x}(t)|| < \delta$. In such time steps, Δt is thus scaled by a factor of p, computed as follows:

$$p = (1 + N\delta_1) \tag{2.23}$$

whenever $\mathbf{a}^T(t)\mathbf{a}(t-1) > 0$, where δ_1 is a small, positive constant, and N is the number of successful steps carried out. N is reset to 1 whenever $\mathbf{a}^T(t)\mathbf{a}(t-1) \leq 0$, and N is not incremented if $||\Delta\mathbf{x}(t)|| \geq \delta$. Suggested starting values for the parameters are $\delta_1 = 0.001$ and $\Delta t = 0.5$.

The LeapFrog algorithm with the above enhanced time step control mechanism is known as *LFOP1(b)*, and is the preferred version of the algorithm. In the rest of this thesis, unless otherwise stated, this version of the algorithm is assumed.

The full LFOP1(b) algorithm [54] is summarised in algorithm listing 2.2, which has been taken from Engelbrecht [20].

Algorithm 2.2 The LeapFrog Algorithm

```
1: Create a random initial solution $\mathbf{x}(0)$, let $t$ = -1;
2: Let $\Delta t = 0.5, \delta = 1, m = 3, \delta_1 = 0.01, \epsilon = 10^{-5}, i = 0, j = 2, s = 0, p = 1$;
3: Compute initial acceleration $a(0) = \nabla f(\mathbf{x}(0))$ and velocity $\mathbf{v}(0) = \frac{1}{2}a(0)\Delta t$;
4: repeat
5:     $t = t + 1$;
6:     Compute $\|\Delta\mathbf{x}(t)\| = \|v(t)\|\,\Delta t$
7:     if $\|\Delta\mathbf{x}(t)\| < \delta$ then
8:         $p = p + \delta_1, \Delta t = p\Delta t$;
9:     else
10:        $\mathbf{v}(t) = \delta\mathbf{v}(t)/(\Delta t\,\|\mathbf{v}(t)\|)$;
11:    end if
12:    if $s \geq m$ then
13:        $\Delta t = \Delta t/2, s = 0$;
14:        $\mathbf{x}(t) = (\mathbf{x}(t) + \mathbf{x}(t-1))/2$;
15:        $\mathbf{v}(t) = (\mathbf{v}(t) + \mathbf{v}(t-1))/4$;
16:    end if
17:    $\mathbf{x}(t+1) = \mathbf{x}(t) + \mathbf{v}(t)\Delta t$;
18:    repeat
19:        $a(t+1) = -\nabla f(\mathbf{x}(t+1))$;
20:        $\mathbf{v}(t+1) = \mathbf{v}(t) + \mathbf{a}(t+1)\Delta t$;
21:        if $\mathbf{a}^T(t+1)\mathbf{a}(t) > 0$ then
22:            $s = 0$;
23:        else
24:            $s = s + 1, p = 1$
25:        end if
26:        if $\|\mathbf{a}(t+1)\| > \epsilon$ then
27:            if $\|\mathbf{v}(t-1)\| > \|\mathbf{v}(t)\|$ then
28:                $i = 0$;
29:            else
30:                $\mathbf{x}(t+2) = (\mathbf{x}(t+1) + \mathbf{x}(t))/2$;
31:                $i = i + 1$;
32:                if $i \leq j$ then
33:                    $\mathbf{v}(t+1) = (\mathbf{v}(t+1) + \mathbf{v}(t))/4$;
34:                    $t = t + 1$;
35:                else
36:                    $\mathbf{v}(t+1) = 0, j = 1, t = t + 1$;
37:                end if
38:            end if
39:        end if
40:    until $\|v(t+1)\| > \|v(t)\|$;
41: until $\|a(t+1)\| \leq \epsilon$; Return $x(t)$ as solution;
```

Figure 2.3 has been created and included here to give an overview of Algorithm 2.2. The algorithm's main conceptual points are shown as a flowchart for comprehensibility.

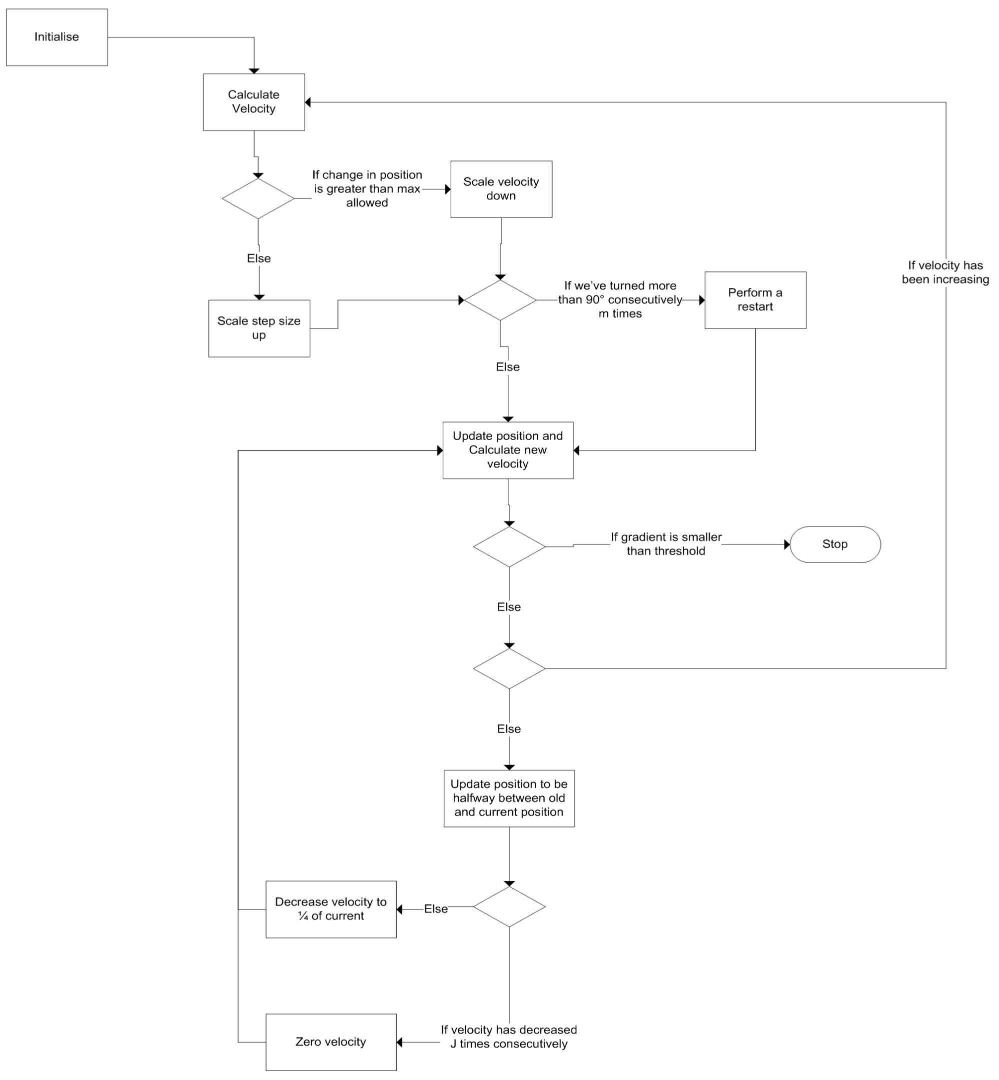
Initialise
Calculate Velocity
If change in position is greater than max allowed
Scale velocity down
Else
Scale step size up
If we've turned more than 90° consecutively m times
Perform a restart
If velocity has been increasing
Else
Update position and Calculate new velocity
If gradient is smaller than threshold
Stop
Else
Else
Update position to be halfway between old and current position
Decrease velocity to ¼ of current
Else
Zero velocity
If velocity has decreased J times consecutively

Figure 2.3: An overview of the LFOP1(b) algorithm

2.3.2 Parameter Selection

The LeapFrog algorithm defines several parameters which can be modified to change the behaviour of the algorithm. A short discussion follows:

- Δt is the step size, used to scale the velocity component at each time step. In the LFOP1(b) version of the algorithm, Δt is constantly updated by a dynamic method, as explained in section 2.3.1. However, the initial choice for step size still influences the overall algorithm.

- Introduced in [54], p_1 is used to modify the ratio p, which scales Δt at appropriate points in the algorithm. Essentially, p_1 controls how aggressively Δt is scaled by the algorithm.
- The parameter δ is used as a threshold to determine whether Δt should be increased or decreased. The algorithm effectively attempts to scale Δt such that $\Delta t||\mathbf{v}(t)|| \approx \delta$, making δ the maximum velocity allowed.
- The control parameter j is used as a threshold to determine whether a velocity reset should occur or not. If the number of consecutive time steps where velocity has decreased, i, is greater than j, then the velocity is set to 0. The value of j can be modified to change how quickly the algorithm punishes a decrease in velocity, which is taken to indicate that the algorithm is finding worse solutions at each subsequent time step.
- The parameter m is a threshold for determining whether a restart should happen. If the number of consecutive time steps where gradient components have changed direction by more than 90°(denoted by s) is greater than m, a restart is performed. Conceptually similar to j, the parameter m can be modified to change how quickly the algorithm punishes a Δt value which causes a large change in direction of successive gradients.

Snyman provides reasonable defaults for the above parameters in [52, 54], however provides no further guidance or empirical evidence regarding parameter choices. During the experiments which provided empirical results for this dissertation, problem specific values were determined for the parameter Δt, which was found to have a significant effect on the performance of the algorithm. These values are given in full in section 4.4.3, table 4.5. In the case of the other algorithm parameters, the reasonable defaults suggested by Snyman were used.

2.3.3 Summary

The LeapFrog algorithm has been empirically shown to perform well against comparable algorithms on benchmark functions [52, 54]. Snyman [52, 54] specifically points to the scalability of LeapFrog with increasing dimensionality which compares very favourably with other Quasi-Newton methods. The following is a list of interesting features and characteristics of the algorithm, which may contribute toward its good performance:

- A dynamic mechanism for adjusting the step size for each iteration. The mechanism allows both upward and downward scaling of the step size, based on a heuristic condition which seems to perform well on many benchmark functions.
- Velocity clamping to ensure that velocities do not become too extreme.
- Heuristically triggered restarts which use information from previous time steps.

2.4 Particle Swarm Optimiser

The Particle Swarm Optimiser (PSO) was originally proposed by Eberhart and Kennedy [14, 28] as a model of the localised movements of groups of animals.

The algorithm has been successfully applied to a diverse set of problems and fields, such as function minimisation [28], nonlinear mapping [17, 18], neural network training [21], inversion of neural networks [58], data mining [55] and software testing [63], to name a few.

In this section, the particle swarm optimiser is described. The basic algorithm is discussed in section 2.4.1. A graphical illustration of the algorithm in given in section 2.4.2. An overview of particle topologies is given in section 2.4.3. A discussion of swarm concepts is given in section 2.4.4, followed by discussions on particle trajectories (section 2.4.5) and modifications to the basic algorithm in section 2.4.6. Further modifications to the algorithm are discussed in section 2.5.

2.4.1 Basic Algorithm

The PSO maintains a swarm of particles, each representing a candidate solution to the problem being optimised. The algorithm iteratively updates the position of each particle in the swarm by calculating a velocity and applying it to each particle's current position.

Let $\mathbf{x}_i(t)$ denote the position of particle i, at time step t. The position of the particle is then updated at each time step as follows:

$$\mathbf{x}_i(t+1) = \mathbf{x}_i(t) + \mathbf{v}_i(t+1) \tag{2.24}$$

When considering a particle i, the velocity consists of three major components:

- A *cognitive component*, derived from the particle's previous personal best position.
- A *social component*, derived from the position of the best particle that particle i is aware of.
- An *inertia component*, derived from the particle's previous velocity.

Each particle maintains its personal best position, allowing the cognitive component to be calculated. The social component is calculated based on the position of a particle regarded as the best in a neighbourhood of particles. A neighbourhood refers to a social structure whereby particles are aware of each other; the concept is further discussed in section 2.4.3.

Originally, two basic variations of PSO were developed, referred to as the global best PSO (*gbest*) and the local best PSO (*lbest*). They differ from each other only in the size of the particle neighbourhoods; in fact *gbest* can be described as a special case of *lbest* where only a single neighbourhood (containing the entire swarm) exists. Since *gbest* is the simpler of the two algorithms, it will be discussed first.

Global Best PSO

In the case of the *gbest* PSO, the social structure is such that all particles are fully interconnected. Effectively, at each time step, all the particles are updated with a social component derived from the position of the best particle in the entire swarm, denoted by $\hat{\mathbf{y}}(t)$.

The velocity is then calculated as follows:

$$v_{i,j}(t+1) = v_{i,j}(t) + c_1 r_{1,j}(t)[y_{i,j}(t) - x_{i,j}(t)] + c_2 r_{2,j}(t)[\hat{y}_j(t) - x_{i,j}(t)] \tag{2.25}$$

where $v_{i,j}(t)$ is the velocity of particle i in dimension j at time step t. The equation also includes the constants c_1 and c_2, known as *acceleration coefficients*, which are used to balance the social and cognitive components, as well as $r_{1,j}(t) \sim U(0,1)$ and $r_{2,j}(t) \sim U(0,1)$, sampled from a uniform random distribution. The vectors $\mathbf{r}_1$ and $\mathbf{r}_2$ add a stochastic element to the algorithm.

Assuming minimisation, the personal best position $\hat{\mathbf{y}}_i$ of each particle i, is calculated as:

$$\mathbf{y}_i(t+1) = \begin{cases} \mathbf{y}_i(t) & \text{if } f(\mathbf{x}_i(t+1)) \geq f(\mathbf{y}_i(t)) \\ \mathbf{x}_i(t+1) & \text{if } f(\mathbf{x}_i(t+1)) \leq f(\mathbf{y}_i(t)) \end{cases} \tag{2.26}$$

where f is the objective function.

The global best position, $\hat{\mathbf{y}}$ can be calculated as

$$\hat{\mathbf{y}}(t) \in \{\mathbf{y}_0(t), ..., \mathbf{y}_{n_s}\} | f(\hat{\mathbf{y}}(t)) = \min\{f(\mathbf{y}_0(t)), ..., f(\mathbf{y}_{n_s}(t))\} \tag{2.27}$$

where the global best position is defined as the best position discovered by any of the particles since $t = 0$. Alternatively, $\hat{\mathbf{y}}$ can also be calculated as

$$\hat{\mathbf{y}}(t) = \min\{f(\mathbf{x}_0(t)), ..., f(\mathbf{x}_{n_s}(t))\} \tag{2.28}$$

where the global best position is defined as the best current position of any particle in the swarm. It is important to note the difference between equations (2.27) and (2.28); the former represents the best solution found so far by the swarm, whereas the latter only takes into consideration the solutions found in the last time step. One of the implications of equation (2.27) is that the best solution found is never lost, making it fit the condition of a hill descent algorithm.

Algorithm 2.3 The *gbest* PSO Algorithm

```
1: Create and initialise an $n_x$-dimensional swarm, S;
2: repeat
3:   for all particles $i \in [1, ..., S.n_s]$ do
4:     if $f(S.\mathbf{x}_i) < f(S.\mathbf{y}_i)$ then
5:       $S.\mathbf{y}_i = S.\mathbf{x}_i$;
6:     end if
7:     if $f(S.\mathbf{y}_i) < f(S.\hat{\mathbf{y}})$ then
8:       $S.\hat{\mathbf{y}} = S.\mathbf{y}_i$;
9:     end if
10:  end for
11:  for all particles $i \in [1, ..., S.n_s]$ do
12:    update the velocity using equation 2.25
13:    update the position using equation 2.24
14:  end for
15: until stopping condition is true
```

Using conventions from [20] and [9], the algorithm is summarised in algorithm listing 2.3.

Local Best PSO

In contrast to the global best PSO, local best PSO (*lbest* PSO) makes use of a social structure whereby all the particles are not directly connected with each other. A social structure known as the *ring topology* is used. The ring topology groups particles into smaller, interleaved neighbourhoods which slow the spread of social information. It is important to note that the particle indices - rather than spatial information - are used to determine particle neighbourhoods (discussed further in section 2.4.3). Particle topologies, such as the ring topology, are discussed in greater detail in section 2.4.3.

The velocity calculation changes as follows:

$$v_{i,j}(t+1) = v_{i,j}(t) + c_1 r_{1,j}(t)[y_{i,j}(t) - x_{i,j}(t)] + c_2 r_{2,j}(t)[\hat{y}_{i,j}(t) - x_{i,j}(t)] \tag{2.29}$$

where $\hat{y}_{i,j}$ is the best position in dimension j, found by particles in particle i's neighbourhood.

Using conventions from [20] and [9], the *lbest* PSO algorithm is summarised in algorithm listing 2.4.

Algorithm 2.4 The *lbest* PSO Algorithm

```
1: Create and initialise an $n_x$-dimensional swarm, S;
2: repeat
3:   for all particles $i \in [1, ..., S.n_s]$ do
4:     if $f(S.\mathbf{x}_i) < f(S.\mathbf{y}_i)$ then
5:       $S.\mathbf{y}_i = S.\mathbf{x}_i$;
6:     end if
7:     if $f(S.\mathbf{y}_i) < f(S.\hat{\mathbf{y}})$ then
8:       $S.\hat{\mathbf{y}} = S.\mathbf{y}_i$;
9:     end if
10:  end for
11:  for all particles $i \in [1, ..., S.n_s]$ do
12:    update the velocity using equation 2.29
13:    update the position using equation 2.24
14:  end for
15: until stopping condition is true
```

2.4.2 An Illustration of the PSO Algorithm

Figure 2.4 illustrates the main forces acting on a single particle. The idea for the geometric diagram is taken from [19]. The figure shows how the social and cognitive components, together with the previous time step's velocity, contribute toward the velocity for the next time step. The illustration visualises how the PSO velocity equation balances the various forces acting on a particle, such as the desire to perform a global search (social component) and the desire to perform local search (cognitive component). The simplicity of the core PSO algorithm is apparent, requiring only elementary mathematical concepts such as multiplication and vector addition to be implemented.

2.4.3 Topologies

In the context of Particle Swarm Optimisation, the term *topology* refers to the social structure, or social connectivity of the particles. A topology determines how a particle's

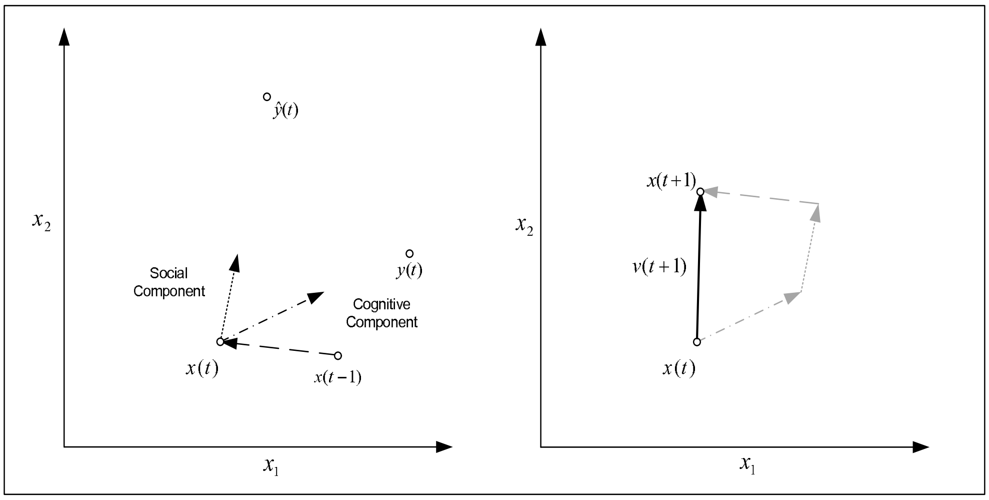

Figure 2.4: A geometric illustration of the PSO velocity components

neighbourhood is defined. The terms *neighbourhood* and topology are sometimes used interchangeably. Numerous topologies have been proposed and studied [15, 26, 29, 37], each exhibiting different characteristics. The choice of topology influences the rate at which information is exchanged in the swarm, thus influencing the rate of convergence.

It is important to note that whilst the various topologies prescribe connections which can be visualised as certain shapes, topologies are normally not implemented to take a particle's position in the search space into account. Neighbourhoods are typically determined using the index of each particle, and since particles are usually randomly placed during initialisation of the algorithm, particles in the same social neighbourhood might be far apart in terms of the search space. Over time, the social sharing of information tends to lead to the convergence of inter-connected particles. Neighbourhoods where the Euclidean distance between particles is taken into account has been proposed by Suganthan [56]. The Euclidean distance-based topologies are more complex and have a large computational cost, making the index-based topologies the preferred choice.

Several of the most popular topologies are listed and briefly discussed below.

The star topology

The social structure employed in the *gbest* algorithm is known as the *star topology*. All particles in the swarm are directly connected with each other, and as a result at each time step, particles are attracted toward the best particle in the entire swarm. The star topology leads to rapid convergence of the swarm, and is thought to perform well on unimodal problems.

The ring topology

The social structure employed in the *lbest* algorithm is known as the ring topology. Here, each particle is connected to $n_{\mathcal{N}}$ of its closest neighbours. To facilitate the sharing of information between distinct neighbourhoods, at least one particle in each neighbourhood belongs to another neighbourhood as well. Over time, this overlap leads to the different neighbourhoods converging toward each other, and as a result, a single solution. The name of the topology is taken from the special case $n_{\mathcal{N}} = 2$, where the topology resembles a ring structure.

The indirect link between the neighbourhoods also leads to a lower rate of convergence. The effect is that the swarm is able to explore more of the search space, and as a result, the ring topology tends to perform better than the star topology on multimodal problems.

The wheel topology

The wheel topology aims to slow down the rate of convergence by effectively isolating the particles from each other. A single particle is designated as a focal point, and all other particles have a single connection to this particle. The focal particle is thus connected to all particles, but the rest of the swarm are only indirectly connected with each other, through the focal particle.

The pyramid topology

The pyramid topology can be visualised as a three-dimensional pyramid, with particles being placed at equal distances on its surface. Particles are connected to their conceptual neighbours, resulting in a topology which balances the rate of convergence by having a high degree of connectivity, whilst at the same time maintaining a large information gap between particles on opposite sides of the structure.

The four clusters topology

This topology prescribes four neighbourhoods of five particles each, with there being two inter-neighbourhood connections between each pair. Intra-neighbourhood connections take the form of the star topology, thus all five particles are connected with each other.

The Von Neumann topology

The Von Neumann topology sets out the particles in the shape of a three dimensional wire grid. It is conceptually similar to the pyramid topology, and has been empirically shown to perform superior to the other topologies on a large number of problems [29, 45].

2.4.4 Swarm Concepts

This section introduces and discusses concepts which are particularly relevant to swarm algorithms.

Swarm Convergence

A swarm whose particles tend cluster spatially over time is said to be *convergent*. By adjusting algorithm parameters, e.g. selecting large values for c_1 and c_2, it is possible to create swarms which exhibit *divergent* behaviour [9], however this is not desirable. Swarm convergence has received much attention from researchers in the past (to be discussed in more depth in the following sections), and remains an important concept. Examples of research on swarm convergence include velocity clamping [15], inertia weights [50], Clerc and Kennedy's constriction coefficients [8], Poli's Canonical PSO [46, 47], Van den

Bergh's extensive analysis of swarm convergence and the Guaranteed Convergence PSO (GCPSO) [9], to name a few.

Swarm Diversity

The term *swarm diversity* is used to describe the degree to which particles in a swarm are spread out. As particles converge toward the global or neighbourhood best particles, swarm diversity decreases. A swarm which lacks diversity is unlikely to uncover new regions of interest in the search space, and as a result is generally less likely to experience large improvements in the solutions found.

The term *premature convergence* refers to the phenomenon of a swarm converging too quickly, effectively missing opportunities to better explore the search space. Swarms that exhibit premature convergence perform poorly, especially on highly multimodal functions, where they tend to get trapped in local minima early on.

Exploration vs. Exploitation

The term *exploration* refers to the ability of an algorithm to explore diverse regions of the search space, and to find regions where good local minima may be located.

Exploitation refers to the ability of an algorithm to refine a candidate solution to a local minimum, having already found a region of interest.

The terms are by no means unique to swarm optimisers, however they remain central themes in research, with most suggested modifications aiming to improve either the exploration or exploitation abilities of PSO. Examples include the GCPSO [9], the PSO with a non-uniform mutating operator [22], the PSO with a mutating operator [33], the two-step PSO [58], the HGPSO [40] and the DoLPSO [60], amongst many others [7, 20, 27]. The concepts of exploration, exploitation, diversity and convergence are related - essentially, PSO algorithms aim to achieve a balance whereby the swarm has high diversity, preventing premature convergence, yet allowing the swarm to converge when a good solution is found.

Particle swarm optimisers have been shown empirically to exhibit good exploration characteristics on numerous problems [2, 56]. However, their ability to refine these solutions in an efficient manner is generally not as impressive [2, 30, 9, 60].

Later sections examine approaches that attempt to enhance the exploitation ability of the PSO.

2.4.5 Particle Trajectories

Ozcan and Mohan presented the first formal analysis of particle trajectories in [43]. Their first study considered a simplified swarm with several assumptions, but was followed by a second study of a more generalised, multi-dimensional PSO system in [44]. A full treatment is beyond the scope of this thesis, but their conclusions are summarised here:

- Particles conduct their search in different manners in different regions of the search space. The steps made by particles in each dimension is sampled from a random sine wave, with the amplitude of the wave decreasing over time; in some regions the step size is greater than in others.
- Particles do not “fly” through the search space - a term borrowed from the algorithm’s origins in the study of swarming animals, such as birds - but rather, particles “surf” on the aforementioned sine waves.

It has been shown by Van den Bergh [9] and also Trelea [59] that a swarm will display convergent behaviour if the following conditions hold:

$$1 > w > \frac{1}{2}(c_1 + c_2) - 1 \geq 0 \tag{2.30}$$

where w is the interia weight (discussed in section 2.4.6). The characterestics of the swarm with respect to convergence or divergence largely depend on values of w, c_1, and c_2.

Further, in-depth studies by Clerc and Kennedy [8], Trelea [59], and Van den Bergh and Engelbrecht [11] on particle trajectories have reinforced the finding that particles will converge to a stable point under certain conditions. The convergence relationship given in equation (2.30) satisfies the convergence conditions.

2.4.6 Modifications to the Basic Algorithm

Modifications to the basic PSO have been proposed to enhance the performance of the algorithm. Some of these modifications, are examined in this section.

Velocity Clamping

Large velocities can cause particles to leave the defined boundary constraints of the problem, which not only results in wasted effort due to discarded solutions, but may also cause the swarm to diverge rather than converge. To counter this problem, the following mechanism has been proposed by Eberhart *et al.* [15], whereby large velocities calculated using the normal velocity equations (2.25) and (2.29) can be reduced:

$$v_{i,j}(t+1) = \begin{cases} v'_{i,j}(t+1) & \text{if } v'_{i,j}(t+1) \leq V_{max,j} \\ V_{max,j} & \text{if } v'_{i,j}(t+1) \geq V_{max,j} \end{cases} \tag{2.31}$$

$V_{max,j}$ therefore defines the maximum value for a velocity component in dimension j. The choice of $V_{max,j}$ defines whether the swarm will tend toward exploration or exploitation. Small values of $V_{max,j}$ tend to cause particles to explore only their local regions, and may cause them to become trapped in local minima. The number of steps required to reach a good local solution may also increase.

$V_{max,j}$ is usually defined as a proportion of the boundary constraint:

$$V_{max,j} = \delta\left(x_{max,j} - x_{min,j}\right) \tag{2.32}$$

where $\delta \in (0,1]$ and $x_{max,j}$ and $x_{min,j}$ represent the maximum and minimum allowed values for x in dimension j.

Since the choice of δ defines the effective value for $V_{max,j}$ it indirectly controls the trade-off between exploration and exploitation. As a result, its value needs to be adjusted for each distinct problem [42, 51].

Velocity clamping also has the effect of changing the direction of the velocity vector [20]. This is due to the way in which $V_{max,j}$ is applied according to equation (2.31). Each component of the velocity is evaluated - and potentially modified - in isolation from the other components. This implies that the velocity is not scaled back by an equal ratio in all dimensions. For very small values of $V_{max,j}$, it is not uncommon for all the components of v to be set equal to $V_{max,j}$. No studies have been done to examine whether this phenomenon has a significant impact on the performance of the algorithm.

Inertia Weight

Similar in its goals to velocity clamping, the inertia weight aims to balance exploration and exploitation in the swarm [50]. This is achieved by applying the inertia weight, $w \in \mathbb{R}$, to the previous time step's velocity during the velocity update. The *gbest* velocity update equation (2.25) now becomes:

$$v_{i,j}(t+1) = wv_{i,j}(t) + c_1 r_{1,j}(t)[y_{i,j}(t) - x_{i,j}(t)] + c_2 r_{2,j}(t)[\hat{y}_j(t) - x_{i,j}(t)] \tag{2.33}$$

The *lbest* velocity update equation (2.29) is modified as follows:

$$v_{i,j}(t+1) = wv_{i,j}(t) + c_1 r_{1,j}(t)[y_{i,j}(t) - x_{i,j}(t)] + c_2 r_{2,j}(t)[\hat{y}_{i,j}(t) - x_{i,j}(t)] \tag{2.34}$$

Large values for w will keep particle velocities large, while smaller values result in particle velocities becoming smaller more rapidly with each iteration, unless c_1 and c_2 are adjusted to satisfy the condition in equation (2.30). In this way, the value of w determines whether the swarm will be driven more toward exploration, or toward exploitation. Similar to the selection of the value of V_{max}, it determines a trade-off that is problem dependent. It is also important to realise that w is applied to the previous velocity vector, which has already been scaled by c_1 and c_2. Therefore, a decision on the value of w needs to take into consideration the values of c_1 and c_2.

Asynchronous Updates

The manner in which the updates in the original PSO algorithm (defined in algorithm 2.3) has since been termed *synchronous*. An alternative, *asynchronous* approach to perform the updates has been proposed [20]. In the asynchronous PSO, the updates to each particle's velocity, position, personal best and global or neighbourhood best particles are done immediately. As a result, information regarding better solutions is spread through the swarm more quickly than in the synchronous PSO. The asynchronous PSO has been shown to exhibit faster convergence as a result [20]. The asynchronous PSO algorithm is summarised in algorithm 2.5.

Algorithm 2.5 The asynchronous *gbest* PSO Algorithm

```
1: Create and initialise an $n_x$-dimensional swarm, S;
2: repeat
3:   for all particles $i \in [1, ..., S.n_s]$ do
4:     if $f(S.\mathbf{x}_i) < f(S.\mathbf{y}_i)$ then
5:       $S.\mathbf{y}_i = S.\mathbf{x}_i$;
6:     end if
7:     if $f(S.\mathbf{y}_i) < f(S.\hat{\mathbf{y}})$ then
8:       $S.\hat{\mathbf{y}} = S.\mathbf{y}_i$;
9:     end if
10:    update the velocity using equation 2.25
11:    update the position using equation 2.24
12:  end for
13: until stopping condition is true
```

2.5 Modified PSO Algorithms

Several modified PSO algorithms are described in this section, each relevant to the main topic of the thesis. The GCPSO is examined in section 2.5.1, followed by the PSO with Non-Uniform Mutating Operator (section 2.5.2), the PSO with Mutation Operator (section 2.5.3), the Two-step PSO (section 2.5.4), the Hybrid Gradient Descent PSO (section 2.5.5), the FR-PSO (section 2.5.6), the GTPSO (section 2.5.7), and finally the Division of Labour PSO in section 2.5.8.

The algorithms were selected for their relevance to this thesis, specifically where the algorithm employs or estimates gradient information. To a lesser degree, algorithms where specific particles are driven by a separate velocity or position update equation are included. The GCPSO is included because it fulfills the latter criteria, and because it is a well established algorithm, making it easier to compare performance against. The HGPSO was included for its combination of the PSO algorithm with a direct gradient-based algorithm, and the Two-Step PSO is included because it makes use of a very rough gradient estimate. The PSO with Non-Uniform Mutating Operator was included for performance comparisons due to its increased swarm diversity [22]. The DoLPSO was included for its treatment of the division of labour concept, which is similar to the concept of separate velocity and position update equations for different particles.

2.5.1 Guaranteed Convergence PSO

In this section, the Guaranteed Convergence PSO (GCPSO) [9, 10] is discussed.

Motivation

Van den Bergh and Engelbrecht [9, 10] examined the convergence properties of particle swarm optimisers, and presented the following observation:

Consider a situation where $\mathbf{x}_i = \mathbf{y}_i = \hat{\mathbf{y}}_i$, i.e. a particle's current position coincides with the global best position (and, thus also the particle's personal best position). Using the velocity update equation (2.33) with inertia weight, it can be observed that $y_{i,j}(t) - x_{i,j}(t) = 0$ and $\hat{y}_j(t) - x_{i,j}(t) = 0$, effectively implying that for this situation, $v_{i,j}(t+1) = wv_{i,j}(t)$; the particle is dependent entirely on the inertia component for the velocity update. This condition has the following implications:

- Since the velocity update is dependent solely on the previous velocity, there is no way for the particle to change direction. There is no guarantee that the previous velocity leads toward a local minimum, leading to a potential decrease in the ability of the particle to exploit its surroundings and a stagnation in the quality of the solution found by the particle.

- Once all the particles in the swarm are near $\hat{\mathbf{y}}$, the velocity of the entire swarm will be driven by the inertia component. In turn, the inertia weight may decrease velocities to the point where they are close to zero. At this point, the swarm has effectively converged and will have no chance of finding a better solution. While this condition should eventually occur naturally in a swarm (when the swarm has been paramaterised according to equation (2.30)), here it is seen as premature convergence, since it is possible for this convergence to occur before the swarm has even reached a local minimum.

In summary, the local optimisation and convergence characteristics of standard PSO can be improved upon, and with this in mind, the Guaranteed Convergence PSO (GCPSO) was introduced [9, 10].

Overview

The proposed solution to the above problem involves modifying the velocity update equation of the best particle as follows:

$$v_{i,j}(t+1) = wv_{i,j}(t) - x_{i,j}(t) + y_{i,j} + \rho(t)r_j \tag{2.35}$$

where $r_j \sim U(-1,1)$ and $\rho(t)$ is a scaling factor calculated as follows:

$$\rho(t+1) = \begin{cases} 2\rho(t) & \text{if } \#successes > s_c \\ 0.5\rho(t) & \text{if } \#failures > f_c \\ \rho(t) & \text{otherwise} \end{cases} \tag{2.36}$$

where s_c and f_c are threshold parameters which can be configured. A default value of $\rho(0) = 1.0$ was used in the original study [10], with acceptable results.

Assuming minimisation, a failure is defined as $f(\hat{\mathbf{y}}(t)) \leq f(\hat{\mathbf{y}}(t+1))$, and in such a case, $\#failures$ is incremented and $\#successes$ is set to zero. A success is defined as $f(\hat{\mathbf{y}}(t)) > f(\hat{\mathbf{y}}(t+1))$, and here $\#successes$ is incremented and $\#failures$ is set to zero. Whenever the best particle changes, both counters are set to zero.

The choice of values for s_c and f_c are problem dependent, and can be independently adjusted to conform to various strategies. The original work examines using $f_c = 5, s_c = 15$, which is empirically found to produce acceptable results on highly multimodal functions. An alternative mechanism of dynamically adjusting f_c and s_c is also discussed by Van den Bergh [9].

The modified velocity update equation (2.36) results in the particle performing a directed random search around the best particle. Under the majority of conditions, this results in the best particle moving toward the local minimum much faster than is the case with the standard PSO velocity update. This suggests that neighbourhoods, being a mechanism to slow down the propagation of social information, play an increased role in the GCPSO. This topic is further investigated by Peer *et al* [45], where the *lbest* algorithm was generally found to produce a superior result when compared to *gbest*, especially for multimodal objective functions.

Van den Bergh [9] provided local convergence proofs for the GCPSO, as well as proofs that neither the standard PSO nor GCPSO satisfy the global search algorithm criteria.

It is suggested that the algorithm can be altered to be a global search algorithm by adding randomised particles (RPSO) or performing multiple starts (MSPSO).

2.5.2 PSO with Non-Uniform Mutating Operator

Esquivel and Coello Coello [22] proposed the introduction of a mutation operator to the PSO algorithm as a means of increasing swarm diversity, a key factor for good performance on highly multimodal functions.

A non-uniform mutation operator is adopted from [38]:

Assume $\mathbf{x}(t) = [x_1(t), ..., x_n(t)]$; if x_j is the element being mutated, then $\mathbf{x}(t+1) = [x_1(t), ...x'_j(t), ..., x_n(t)]$, where

$$x'_j(t) = \begin{cases} x_j(t) + \Delta(t, UB - x_j(t)) & \text{if a random digit is 0} \\ x_j(t) - \Delta(t, x_j(t) - LB) & \text{if a random digit is 1} \end{cases} \tag{2.37}$$

UB and LB are the upper and lower bounds for x_j, respectively. The function $\Delta(t, y)$ returns a value in the range $[0, y]$ such that the probability of $\Delta(t, y)$ being close to 0 increases as t increases. The definition of the function $\Delta(t, y)$ used by Esquivel and Coello Coello [22] is from [38]:

$$\Delta(t, y) = y\left(1 - r^{(1-\frac{t}{T})^b}\right) \tag{2.38}$$

where $r \in [0, 1]$ is a random number, T is the total number of iterations allowed for the algorithm, and b is a parameter which determines the degree of dependency on iteration number.

The mutating PSO was benchmarked and found to be competitive with, and in some cases superior to the 6 topologies investigated by Peer *et al.* in [45]. Specifically, the mutating PSO seemed to have performed better against the other algorithms on the more complex functions, such as Rastrigin and Schwefel.

Esquivel and Coello Coello [22] concluded that the results validated their hypothesis, specifically that the use of a non-uniform mutation operator increases swarm diversity, which enables the algorithm to perform better on highly multimodal functions.

2.5.3 PSO with a Mutation Operator

Li *et al.* proposed a Particle Swarm Optimiser with a mutation operator in [33]. The authors observed that the standard PSO has a tendency to get trapped in local minima, making note of the convergence issue discussed in section 2.5.1 and [9], where $\mathbf{x}_i = \mathbf{y}_i = \hat{\mathbf{y}}_i$.

Li *et al* made reference to the Multi-start PSO (MPSO) proposed in [9], whereby particles are re-initialised after a certain number of iterations, which increases swarm diversity and allows the MPSO to escape local minima. Li *et al.* maintain that whilst the MPSO approach achieves the desired global convergence characteristics, the re-initialisation of all particles hampers local search and results in lower accuracy.

In the approach proposed by Li *et al*, the algorithm keeps track of the historic optimal position of the swarm, P_l. If the optimal position of the swarm does not change by a significant amount for $MaxStep$ number of consecutive steps, and the radius of the swarm is below a specified threshold $BorderRadius$, then the position and velocity of several particles is re-randomised according to some mutation probability, ρ.

The radius of the swarm, $MaxRadius$, is defined as:

$$MaxRadius = \max_{j=1...m}\left(\sqrt{\sum_{d=1}^{D}(p_{id} - x_{id})^2}\right) \tag{2.39}$$

Thus $MaxRadius$ is the maximum Euclidean distance between all the particles and the historic optimal position of the neighbourhood.

By adjusting the algorithm parameters $MaxStep$, $MaxRadius$ and ρ, the convergence rate and speed of searching can be influenced.

In empirical trials, Li *et al* assert that the PSO with mutation performed better than either the standard PSO or the Multi-start PSO.

2.5.4 Two-step PSO

The Two-step PSO has been developed by Thompson *et al.* [58]. The approach essentially involves calculating two possible velocities for a particle, and then selecting the one which results in a steeper decline in the objective function, thereby essentially approximating the gradient of the function.

The algorithm can be summarised as follows: Calculate candidate position $\mathbf{x}'_i(t+1)$ using the standard PSO position update:

$$\mathbf{x}'_i(t+1) = \mathbf{x}_i + \mathbf{v}_{t+1} \tag{2.40}$$

and candidate position $\mathbf{x}''_i(t+1)$ using a scaled down velocity:

$$\mathbf{x}'_i(t+1) = \mathbf{x}_i + \delta\mathbf{v}_{t+1} \tag{2.41}$$

where $\delta \in (0,1)$.

The position update equation now becomes:

$$\mathbf{x}_i(t+1) = \begin{cases} \mathbf{x}'_i(t+1) & \text{if } -\mathbf{x}'_i(t+1) \geq -\frac{f(\mathbf{x}''_i(t+1))}{\delta} \\ \mathbf{x}''_i(t+1) & \text{otherwise} \end{cases} \tag{2.42}$$

The two-step PSO introduces a single new parameter, the step size δ, to the PSO algorithm. The algorithm was designed to cater for scenarios where the standard position updates would result in particles overshooting local minima.

Some observations about the algorithm can be made:

- It should be noted that whilst in essence, the two-step PSO approximates the gradient at a point, it does not make direct use of the gradient of the objective function.
- The candidate positions $\mathbf{x}'_i(t+1)$ and $\mathbf{x}''_i(t+1)$ lie in the same direction with respect to $\mathbf{x}_i(t)$; the algorithm does not attempt to find a better search direction, it merely attempts to check if local minima have been missed.
- The modified algorithm can potentially only scale back the position updates of particles, and it does not directly deal with the convergence problems introduced in section 2.5.1. Thompson *et al.* [58] predicted premature convergence issues with the algorithm and attempt to counter the effects by combining the algorithm with a cluster PSO.

2.5.5 Hybrid Gradient Descent PSO (HGPSO)

The Hybrid Gradient Descent PSO, introduced by Noel and Jannett in [40], makes direct use of the gradient of the objective function to enhance the exploitation characteristics of the standard PSO.

The algorithm proposes making use of Gradient Descent as discussed in section 2.2. The standard gradient descent position update (see equation (2.5)) is incorporated into the standard PSO velocity update, and the following hybrid velocity update equation is the result:

$$v_{i,j}(t+1) = v_{i,j}(t)+c_1r_{1,j}(t)[y_{i,j}(t)-x_{i,j}(t)]-(c_1r_{1,j}/\epsilon)[f(x_{i,j}(t)+\epsilon E_i)-f(x_{i,j}(t))] \quad (2.43)$$

where ϵ is a small constant, and E_i is the i^{th} standard basis vector for $\mathbb{R}^n$.

Essentially, the algorithm replaces the standard cognitive component which attracts the particle toward its personal best position in a standard PSO, with the gradient descent term, which attracts the particle in the direction of the steepest decrease in the values of the objective function.

One of the goals of equation (2.43) is to lessen the tendency of the gradient descent algorithm to get trapped in local minima. By combining the gradient descent position update with the inertia and social components, Noel and Jannett propose that swarm diversity is maintained.

It should be noted that the original HGPSO [40] does not make use of inertia weights, but rather, velocity clamping by means of V_{max} was employed. The algorithm was implemented using inertia weights for the purpose of empirical analysis in order to make comparisons to the other algorithms easier (as discussed in section 4.5.7).

2.5.6 FR-PSO

Borowska and Nadolski [6] proposed a gradient-hybrid PSO which makes use of the Flecher-Reeves conjugate gradient method [23], called the FR-PSO. The algorithm is partially inspired by Noel and Jannett's HGPSO [40], however, instead of replacing the cognitive component of the velocity update equation, a third component is added:

$$\begin{aligned} v_{i,j}(t+1) = wv_{i,j}(t) &+ c_1 r_{1,j}(t)[y_{i,j}(t) - x_{i,j}(t)] \\ &+ c_2 r_{2,j}(t)[\hat{y}_j(t) - x_{i,j}(t)] \\ &+ c_3 r_{3,j}(t)[mfr_{i,j}(t) - x_{i,j}(t)] \end{aligned} \tag{2.44}$$

where c_3 is a constant algorithm parameter, $r_{3,j}$ is a random number sampled from a uniform distribution and the $mfr_{i,j}(t)$ term is the solution obtained by the Flecher-Reeves algorithm for particle i in the j-th dimension.

The hybrid algorithm was found by the authors to be more efficient than standard PSO when applied to the problem of finding the optimal geometry of two cylindrical coil arrangements evoking a magnetic field of specific parameters [6]. The largest differences in performance were found when using a relatively large swarm size of 100 particles. It should be noted that this result is limited to a single problem, and that a relatively small number of iterations was observed (100 to 150 iterations), presumably because further iterations did not yield differences in solution quality.

The addition of a third term to the standard PSO velocity update equation is an interesting approach to incorporating a gradient algorithm into PSO. An unfortunate implication of the new term, however, is that the behaviour of such a modified swarm is no longer well researched. One example of this would be Van den Bergh's relationship between inertia weight and c_1 and c_2 [9], for which no version exists which has been generalised to further terms.

2.5.7 GTPSO

Zhang *et. al.* [64] proposed a new hybrid-gradient PSO, which makes use of a new gradient approach. Termed the GTPSO, the new algorithm combines the core algorithm from standard PSO, with a way to approximate the gradient of the problem. The GTPSO algorithm can be summarised as follows:

- Perform an iteration of the standard PSO algorithm.
- Starting at the position of the *gbest* particle, find a new position using a gradient method.

- Repeat the previous step until the gradient method no longer provides a better fitness or a stopping condition is met.
- If the new position found by the gradient method is of a better fitness, update the position and fitness of the *gbest* particle to that of the new position.
- Continue performing iterations until a stopping condition is met.

The GTPSO algorithm was applied by Zhang *et. al.* [64] to the problem of the control of polarization mode dispersion (PMD) compensation in optical fibers. The GTPSO algorithm was found to have a faster convergent speed than standard PSO. Zhang *et. al.* [64] concluded that since convergent speed is an important performance criterion for real time PMD compensation, GTPSO was a better algorithm than standard PSO for that task.

2.5.8 Division of Labour PSO (DoLPSO)

In their research, Vesterstrøm *et al.* found three major problems with the basic PSO algorithm, namely: "premature convergence, parameter control and lack of dynamic velocity adjustment, resulting in the inability to hill climb solutions". The authors set out to address the last issue by proposing the Division of Labour PSO (DoLPSO) [60].

The term *division of labour* refers to the cooperation of individuals which have been specialised to perform different tasks [4]. This type of behaviour is displayed by many of the social insects; for example, ants specialise into different roles such as worker, soldier, queen, and male. This concept is applied to the PSO model to enhance the exploitation capabilities of the algorithm.

Response thresholds

The division of labour mechanism is implemented using response thresholds [5], adapted here to particle swarm optimisation. If i represents a particle, and d represents a task, then $\theta_{i,d}$ is a threshold for particle i to perform task d. Associated with each particle, and for each task, is a stimulus $s_{i,d}$. A response function T is defined to determine the probability of particle i performing task d, given inputs $\theta_{i,d}$ and $s_{i,d}$.

The authors further adapt the mechanism to a single task, leading to the following threshold function:

$$P(X_i = 0 \rightarrow X_i = 1) = T(s_i, \theta_i) = \frac{s^n}{\theta_i^n + s^n} \tag{2.45}$$

where X_i represents the state of particle i, such that when $X_i = 0$, the particle is not performing the task, and is performing the task when $X_i = 1$. The constant n determines the steepness of the response function. The probability of an active particle becoming inactive is defined as:

$$P(X_i = 1 \rightarrow X_i = 0) = p \tag{2.46}$$

where p is an algorithm parameter.

Local search and stimulus

Vesterstrøm *et al.* [60] define the one additional task of the DoLPSO to be local search. When a particle i experiences a lack of improvement, the stimulus s_i is increased, leading to an increased probability that the local search task will be activated.

Given particle i has triggered the local search task, its position $\mathbf{x}_i$ is set to the position of the global best particle, $\hat{\mathbf{y}}$. Furthermore, the velocity of the particle, $\mathbf{v}_i$, is re-initialised using random values, adjusted for the the length of the velocity of the global best particle. By introducing the search direction of the particle to the global best position, it is reasoned that a more efficient local search is produced.

It is clear from the above local search algorithm that the diversity of the swarm will typically be decreased by its application. The authors note this and suggest mechanisms to prevent its application to cases where the local search would cause the diversity to decrease too much, such as when a candidate particle's distance to the global best particle is larger than a certain threshold.

Conclusions

From their empirical analysis, Vesterstrøm *et al.* [60] conclude that when compared to standard PSO, the DoLPSO performs well on unimodal functions, but suffers on multimodal ones.

Beyond the performance of the basic DoLPSO, the research done by Vesterstrøm *et al.* also served to formalise the need for a type of division of labour, as well as one possible implementation for DoL mechanisms in PSO. While modifications to PSO that effectively achieve division of labour, such as the GCPSO [9] had been published prior to [60], they had not been explicit and formal about the fact that division of labour was in effect implemented.

2.6 Conclusion

This chapter has served to provide background to the central theme of the thesis, which is the combination of gradient-based algorithms with Particle Swarm Optimisers. A brief, formal overview of optimisation has been provided. Focus was placed on two gradient based optimisers, the Gradient Descent and LeapFrog algorithms, and the Particle Swarm Optimiser (PSO) was discussed. Several areas of PSO research, as well as specific modified PSO algorithms were discussed, where deemed to be relevant to the central theme of this thesis.

In conclusion, it is apparent that the topic of gradient hybrid PSOs is not one that has been heavily researched in the past (when compared to the focus that other research areas, such as the topic of convergence, have received), and an opportunity exists for new approaches. The remainder of this thesis will focus on this topic, namely the proposal of a new way to construct hybrid gradient PSOs, the introduction of two specific new gradient-hybrid PSOs, and empirical experimentation to determine and verify the characteristics of the new algorithms.

Chapter 3

Using Gradient Information in PSO

This section discusses the concept of making use of gradient information in particle swarm optimisers. Motivation is given in section 3.1, followed by a discussion on the challenges involved in incorporating gradient information in PSO (section 3.3). Two new gradient-based PSO algorithms, namely the gradient descent PSO (GDPSO), the LeapFrog PSO (LFPSO) are given in section 3.2. A generalisation of the GDPSO and LFPSO, the generalised local search PSO (GLSPSO) is also described. Finally, a short treatment of gradient estimation is given in section 3.4.

3.1 Motivation

Section 2.5.1, discussed a condition under which the basic PSO algorithm exhibits poor convergence characteristics. This is a well known condition which has inspired numerous suggestions aimed at improving the basic PSO algorithm [33, 9].

Very few studies have considered making use of gradient information directly within PSO. Notable exceptions are the HGPSO [40] (discussed in section 2.5.5), and the GTPSO [64] (discussed in section 2.5.7), which make use of the gradient descent algorithm, and the FR-PSO [6], which makes use of the Flecher-Reeves method. The HGPSO effectively combines equation (2.5) from the gradient descent algorithm with the PSO velocity update equation (2.25), to circumvent the anticipated premature convergence that gradient descent might cause [40]. The GTPSO and FR-PSO are less concerned with premature convergence, and more with convergent speed [6, 64], due to the nature of the problems they were designed for.

The following interesting observation can be made about the basic PSO algorithm: it makes use of a single type of particle. This thesis refers to this behaviour as *homogeneous*, since all the particles in the swarm are updated using a single equation. Examples of homogeneous PSOs include the basic *gbest* and *lbest* PSOs (algorithms 2.3) and 2.4, respectfully), the HGPSO (described in equation (2.43)), the Two-Step PSO (equation (2.42)). The PSO with Mutation Operator (section 2.5.2) and PSO with Non-Uniform Mutating Operator (described in section 2.5.3) can also be described as being homogeneous. Certain PSO algorithms can also be described as being *heterogeneous*, because different particles in the swarm may be driven by completely different equations. Examples of heterogeneous PSOs include the GCPSO from section 2.5.1 and the Division of Labor PSO (described in section 2.5.8).

The concept of a heterogeneous swarm enables certain particles to be focused on finding good areas (exploration) and other particles to be focused on finding the best solution within these good areas (exploitation). It can be argued that these are two conceptually separate tasks, and therefore should be performed by two separate algorithms. This makes the creation of a hybrid PSO using a gradient-based algorithm an attractive proposition.

3.2 Gradient-Based PSO Algorithms

In section 3.2.1, the differences and similarities between PSO and gradient-based algorithms are discussed. The construction of the gradient descent PSO and LeapFrog PSO is discussed in sections 3.2.2 and 3.2.3 respectively. Finally, a more generalised PSO algorithm, the generalised local search PSO, is proposed in section 3.2.4.

3.2.1 PSO and Gradient Algorithms

Contrasting the standard PSO and gradient-based algorithms discussed so far (namely the gradient descent and LeapFrog algorithms), certain differences can be highlighted:

- The gradient descent and LeapFrog algorithms work with a single candidate solution, while particle swarm optimisers are population-based, and thus deal with multiple solutions at any given point. The population approach allows the PSO to sample a larger area of the search space before determining the direction of future search; it can be argued that this gives it better exploration capability.

- The basic PSO algorithm does not make use of the gradient information of the objective function, whilst gradient descent and LeapFrog do. This allows the gradient algorithm to directly determine the direction of largest improvement in fitness, which the PSO has to attempt to work out by sampling the search space at multiple points. In the context of local optimisation, and assuming gradient information is available, it can be argued that on average, the gradient approach is more efficient. One downside of using gradient information is that typically, gradient based algorithms are local optimisers, which can easily be trapped in local minima.

- Gradient descent and LeapFrog are deterministic algorithms, while Particle Swarm Optimisers are stochastic in nature.

It is clear from the above that PSOs are quite different to the gradient-based algorithms presented so far. Despite this, the algorithms share several critical aspects, which are useful when attempting to combine the two:

- Both categories of algorithms have a similar concept of a solution; the position of a particle in PSO is analogous to a solution in a gradient-based algorithm.

- The concept of an iteration is similar in both cases.

The above similarities mean that PSOs can be successfully combined with gradient-based algorithms.

3.2.2 Gradient Descent PSO

The gradient descent PSO (GDPSO), as its name suggests, is a combination of the PSO (algorithms 2.3 and 2.4) with the simplest of gradient based methods, namely gradient descent (algorithm 2.1).

Considering the similarities between the algorithms discussed in section 3.2.1, the following strategy for the hybrid PSO is proposed:

- Assume the basic PSO algorithm is being used.
- In the case of the *gbest* or *lbest* particles, the position update equation from the standard PSO is ignored. Instead, the gradient descent algorithm is applied to determine the position of the particle in the next iteration. It is important to note that this approach differentiates the algorithms proposed in this thesis from the the HGPSO [40].
- The behaviour of all other particles is unmodified, and work using the principles of the standard PSO algorithm.

Since the *gbest* algorithm can be thought of as a special case of the *lbest* algorithm, in the interest of brevity, the term *lbest* particle will be used to represent both the *gbest* and *lbest* particle for the rest of this chapter. The more generic *lbest* version of the algorithm is presented as algorithm 3.1.

Algorithm 3.1 The GDPSO Algorithm

```
1: Create and initialise an $n_x$-dimensional swarm, S;
2: repeat
3:   for all particles $i \in [1, ..., S.n_s]$ do
4:     if $f(S.x_i) < f(S.y_i)$ then
5:       $S.y_i = S.x_i$;
6:     end if
7:     if $f(S.y_i) < f(S.\hat{y})$ then
8:       $S.\hat{y} = S.y_i$;
9:     end if
10:  end for
11:  for all particles $i \in [1, ..., S.n_s]$ do
12:    if particle $i$ is a neighbourhood best then
13:      update the position using equation (2.5)
14:    else
15:      update the velocity using equation (2.29)
16:      update the position using equation (2.24)
17:    end if
18:  end for
19: until stopping condition is true
```

The following observations can be made about the algorithm:

- The behaviour of the *lbest* particles are directly affected at each iteration of the algorithm, however the rest of the swarm is driven by the standard PSO algorithm. Assuming neighbourhoods that aren't unusually small, this would imply that the majority of the swarm is still operating in an identical fashion to the standard PSO algorithm, retaining the positive performance characteristics of that algorithm.

- By limiting the usage of gradient information to a subset of the particles, the approach attempts to improve exploitation without negatively affecting exploration. The difference in position update equations between standard particles and the *lbest* particles effectively specialises the swarm into two roles, namely: particles specialising more in exploration, and particles specialising more in exploitation.

- The application of a local search algorithm to the *lbest* particles also solves a specific problem with the PSO position update equation when $\mathbf{x}_i = \mathbf{y}_i = \hat{\mathbf{y}}_i$ (as explained by Van den Bergh [9] and discussed in section 2.5.1). This is due to the fact that the above condition depends on two major assumptions, namely the particle in question itself also being the local best particle, and the usage of the standard PSO velocity update equation. By ensuring that the standard PSO velocity update equation is not used for neighbourhood best particles, the situation is circumvented.

- Popular modifications to standard PSO, such as V_{max}, inertia weights and advanced topologies can be applied without significant effort.

The inclusion of the gradient descent algorithm also introduces a new swarm parameter, δ (discussed in section 2.2.2), which can be adjusted for different problems. The approach used in this thesis was to select problem-specific, reasonable defaults for δ. The most basic gradient descent model, involving a static value for δ is used.

GDPSO, GTPSO, and HGPSO all make use of a similar gradient algorithm in the context of a PSO. As a result, some justification for the existence of GDPSO is necessary. It is important to highlight the following differences between the algorithms:

- The HGPSO is homogeneous while the GDPSO is heterogeneous. This distinction allows a separation of concern between exploration and exploitation for the GDPSO, and the specific application of the local search algorithm to local best particles implies exploitation at positions in the search space where it might be most advantageous.

- All particles in the HGPSO make use of the gradient descent algorithm at all times, while only the *gbest* or *lbest* particles in the GDPSO do so. In the case of the HGPSO, this gives each particle a strong local search component which may lead to premature convergence [40].

- The gradient descent algorithm is incorporated into the HGPSO such that the cognitive component from the velocity update equation is removed, and replaced completely by a gradient-based component. This results in the swarm categorically losing memory of personal best positions and the ability to use this information. This is not the case with the GDPSO.

- In the case of the GTPSO, the gradient method is applied repeatedly until it no longer provides any further fitness benefits. This repeated application of the gradient method, within one iteration of the rest of the swarm, is significantly different to the approach used in the GDPSO.

The above differences are significant enough to differentiate the GDPSO from the HGPSO and GTPSO algorithms, and in turn, the inclusion of GDPSO in this thesis.

3.2.3 Leapfrog PSO

The LeapFrog PSO (LFPSO) is GDPSO, in that it replaces the position update equation of the *lbest* particle with a local search algorithm. In the case of the LFPSO, the algorithm performing the local search is the LeapFrog, as documented in algorithm 2.2.

The LeapFrog algorithm is a significantly more complicated and sophisticated algorithm than gradient descent, with several capabilities which make it a good candidate for this thesis. One of the core differentiating aspects is the ability of the LeapFrog algorithm to automatically adjust Δt, which serves a similar purpose as δ in the GD algorithm. This difference also influenced the decision to keep the GDPSO's implementation of the GD algorithm as simple as possible - in this manner, a simplistic gradient algorithm can be contrasted with a complex one.

Since the LeapFrog algorithm has knowledge of previous time steps and can potentially make changes to its own algorithm parameters at each time step, the following approach is used:

- Each particle in the swarm carries its own LeapFrog algorithm parameters and other LeapFrog related state information. Due to a small number of particles typically being used by PSO, this is a relatively small trade-off.

- The LeapFrog algorithm parameters are reset to their defaults if a particle that wasn't *lbest* in the previous time step becomes *lbest* in the current one. These parameters serve to maintain memory of certain aspects of the path of the particle, relevant to the LeapFrog algorithm heuristics (such as the number of times the velocity has subsequently decreased, and the number of times the algorithm has performed a turn greater than 90° - refer to section 2.3). These parameters would no longer be accurate in the case that the *lbest* particle was not the *lbest* particle in the previous timestep. In such a case, the previous position update would have been performed by a standard PSO velocity and position update equation instead of the LeapFrog algorithm. There is no clear way to update the relevant parameters without making use of the LeapFrog algorithm itself. If the algorithm is allowed to make use of the old algorithm parameters, the LeapFrog algorithm's heuristics may make decisions based on invalid memory; it is thus better to perform a reset of the algorithm and allow it to build up a new memory from the new position.

The LeapFrog PSO algorithm is summarised in algorithm 3.2.

Algorithm 3.2 The LFPSO Algorithm

```
1: Create and initialise an $n_x$-dimensional swarm, S;
2: repeat
3:   for all particles $i \in [1, ..., S.n_s]$ do
4:     if $f(S.x_i) < f(S.y_i)$ then
5:       $S.y_i = S.x_i$;
6:     end if
7:     if $f(S.y_i) < f(S.\hat{y})$ then
8:       $S.\hat{y} = S.y_i$;
9:     end if
10:  end for
11:  for all particles $i \in [1, ..., S.n_s]$ do
12:    if particle $i$ is neighbourhood best then
13:      if particle $i$ was not neighbourhood best in previous time step then
14:        reset particle $i$'s LeapFrog parameters
15:        set $\mathbf{x}(0)$ as the current position of the particle
16:      end if
17:      update the position using by using one iteration of the LeapFrog algorithm (algorithm 2.2)
18:    else
19:      update the velocity using equation (2.29)
20:      update the position using equation (2.24)
21:    end if
22:  end for
23: until stopping condition is true
```

3.2.4 Generalised Local Search PSO

There is a clear set of similarities between the GDPSO and LFPSO algorithms, specifically in terms of what they respectively try to achieve and how they achieve this. These similarities can be highlighted by generalising them into a new proposed algorithm, the generalised local search PSO (GLPSO), summarised as algorithm (3.3).

Algorithm 3.3 The GLSPSO Algorithm

```
1: Create and initialise an $n_x$-dimensional swarm, S;
2: **repeat**
3:   **for all** particles $i \in [1, ..., S.n_s]$ **do**
4:     **if** $f(S.x_i) < f(S.y_i)$ **then**
5:       $S.y_i = S.x_i$;
6:     **end if**
7:     **if** $f(S.y_i) < f(S.\hat{y})$ **then**
8:       $S.\hat{y} = S.y_i$;
9:     **end if**
10:  **end for**
11:  **for all** particles $i \in [1, ..., S.n_s]$ **do**
12:    **if** particle $i$ is neighbourhood best **then**
13:      **if** particle $i$ was not neighbourhood best in previous time step **then**
14:        reset particle $i$'s local search parameters, where applicable
15:      **end if**
16:      update the position using a local search algorithm
17:    **else**
18:      update the velocity using equation 2.29
19:      update the position using equation 2.24
20:    **end if**
21:  **end for**
22: **until** stopping condition is true
```

It is apparent that both the GDPSO and LFPSO are specialised versions of the GLSPSO, obtained by simply specifying state information for the particles as well as the position update equation. It is interesting to note, however, that some existing PSOs conform to this pattern as well, e.g. the GCPSO [9].

As with the LFPSO algorithm, GLSPSO algorithm incorporates a step to reset any applicable local search parameters in the case that the neighbourhood best particle was not the neighbourhood best in the previous time step. The exact parameters which are reset may vary depending on the algorithm, e.g. it is likely that in a GDPSO implementation, the algorithm parameter δ would not be reset, as it would not make sense to do so. This step in the algorithm is included specifically to deal with special cases, such as the LFPSO, which rely on local search parameters that are only relevant and accurate given consecutive iterations with the gradient algorithm, for a given particle.

It is also interesting to contrast the GLSPSO with other similar concepts, such as the Division of Labor PSO (DolPSO) and the Hybrid Gradient Descent PSO (HGPSO). It can be argued that the specialisation of particles into the exploration and exploitation roles is simply a division of labor, a concept which is central to DolPSO. In fact, the GLSPSO can be described in terms of Vesterstrøm *et al.*'s DolPSO. GLSPSO is logically equivalent to a DolPSO where the stimulus is defined as whether a given particle is the *lbest* particle or not, with a response threshold so that the local search algorithm is always triggered if the stimulus is present.

Contrasted with the HGPSO, the GLSPSO highlights several critical differences. The most important of these is the lack of specialisation, or division of labor within the HGPSO swarm. In HGPSO, the cognitive component of each particle's velocity update equation is replaced with the Gradient Descent position update equation. Effectively, all particles in the swarm are driven by a single position update equation; they are attempting to perform the dual task of exploration and exploitation using this single equation. This is a completely different approach to the GLSPSO, where exploration and exploitation are treated as separate roles and can be configured individually to suit a given problem. By contrasting the performance of GDPSO and LFPSO against that of the HGPSO using empirical methods (chapter 5), this thesis aims to determine whether this design choice is desirable.

3.3 Challenges with Gradient Information in PSO

Potential challenges arising due to the inclusion of gradient information in PSO are discussed in this section. Section 3.3.1 discusses the possibility of gradient-based PSOs to prematurely converge. This is followed by a discussion in section 3.3.2 of how large gradient variances can adversely affect gradient based algorithms.

3.3.1 Tendency for Premature Convergence

By definition, the gradient of a function gives the direction of the largest change in fitness. It is reasonable to deduct that algorithms which make use of gradient information would be efficient at exploitation of their immediate surroundings [40]. One possible implication of the inclusion of gradient-enabled particles in a swarm would be the rapid improvement in quality of these particles. The unfair advantage afforded by the gradient algorithms to these particles means that it is likely that they would effectively hold the title of neighbourhood best or global best particle for longer than they would otherwise have (empirical evidence for this tendency is given in section 5.2.3). This behaviour could easily unbalance the swarm and adversely affect diversity, and ultimately lead to premature convergence and inferior performance.

Noel and Jannett's work [40] on the HGPSO was discussed in section 2.5.5. One of the primary concerns of the authors was a tendency of gradient hybrid algorithms to prematurely converge, which influenced the eventual design of the HGPSO.

Three possible approaches to counter premature convergence are proposed in this section, namely adjusting social and cognitive acceleration coefficients, delayed introduction of gradient information, and using of alternative PSO topologies. These approaches are empirically evaluated in section 5.3.

Adjusted social and cognitive acceleration coefficients

The social and cognitive acceleration coefficients, c_1 and c_2, can be used to change the balance of forces acting on the particles in a swarm. Larger values for c_1 mean that particles tend to move more towards their own previous best positions, while larger values for c_2 imply that particles tend to move more towards their global and neighbourhood best particles. It seems likely that by decreasing c_2 while increasing or otherwise adjusting c_1, could result in a swarm that is less affected by immediate exploitation of the gradient algorithms.

Delayed introduction of gradient information

By introducing a delay before the gradient information is enabled in the swarm, it is possible to give the swarm time to find good regions without being adversely affected by premature convergence. The following mechanism is proposed in this thesis:

Let $\rho_{delay} \in [0, 1]$ denote the delay coefficient, such that $\rho_{delay} = 0$ implies introduction of gradient information at 0% completion of an experimental run, and $\rho_{delay} = 1$ implies introduction at 100% completion (i.e. no introduction at all). In this thesis, experiments were performed to a fixed number of iterations, and this provided a natural approach for the calculation.

Alternatively, the introduction of gradient information could also be triggered upon the realisation of one or more of the following criteria:

- swarm radius being smaller than a certain threshold,
- no improvement in the fitness of the *gbest* particle over a certain number of iterations,
- the approximated gradient of the fitness function being close to 0.

Use of alternative topologies

In section 2.4.3, alternative topologies such as the ring topology (*lbest* PSO) were discussed). These topologies often serve to delay the spread of information in PSO, thereby reducing the tendency for premature convergence. It is possible that these approaches would work equally well in the case of gradient PSOs.

Synchronous and asynchronous PSO

The choice of a synchronous versus asynchronous PSO was discussed in section 2.4.6. Since information regarding better solutions is spread more quickly in an asynchronous swarm, in a hybrid gradient PSO, this would imply quicker introduction of gradient information. Conversely, a synchronous PSO would imply a slightly delayed introduction of gradient information.

Memory and iteration based *gbest*

The approach of determining the *gbest* position can vary, as discussed in section 2.4.1. The position can be memory based, where the best position out of all previous iterations is considered to be the *gbest* position (equation (2.27)), or itertation based, where the best position in the current iteration is considered (equation (2.28)). It is possible that the choice of memory or iteration based *gbest* may also affect the convergence characteristics of a gradient-based PSO.

3.3.2 Large Gradients

One challenge that gradient-based methods encounter is that of large gradients. The term *large gradient* refers to a position in the search space, where at least one component of the gradient vector can be considered to be unusually large, possibly causing gradient-based methods to exhibit poor performance. A more formal definition is given below, followed by an example scenario involving large gradients.

Let $\nabla_{\min} f(\mathbf{x})$ and $\nabla_{\max} f(\mathbf{x})$ denote the smallest and largest components, respectively, of the gradient vector for all possible solutions $\mathbf{x}$ for the objective function f.

Given a continuous objective function, $f(\mathbf{x})$, which contains at least one minimum that does not lie on the boundary, it follows that $\exists\ \mathbf{x}^B$ such that $\nabla f(\mathbf{x}^B) = [0, ..., 0]$, where $\mathbf{x}^B$ is an extremum of f. Therefore, $\nabla_{\min} f(\mathbf{x}) = 0$. The value of $\nabla_{\max} f(\mathbf{x})$ depends on the function f and the boundary constraints specified.

An example of large gradients

Consider the one-dimensional function $f(x) = x^4$, with boundary constraints $x \in [-5, 5]$. Since $x^* = 0$, it follows that the smallest gradient will occur at x^*, thus $\nabla_{\min} f(x) = 0$. The gradient of the function is defined by the first order derivative, $f'(x) = 4x^3$. The most extreme values for the gradient will occur at the boundary constraints. With this information, it can be calculated that $\nabla_{\max} f(x) = 500$.

Consider the gradient descent algorithm 2.2, where the position update equation is $\mathbf{x}(t + 1) = \mathbf{x}(t) - \delta \nabla f(\mathbf{x}(t))$ (equation (2.5)). Suppose that the step size algorithm parameter has been assigned an arbitrary value of $\delta = 0.5$.

In the context of the gradient descent algorithm, let the terms *convergent* and *divergent* mean that the quality of the solution found is increased and decreased, respectively, with each subsequent iteration of the algorithm. The objective function f is strictly unimodal, and subsequently it can be said that the algorithm will converge if the size of the position updates decreases with each subsequent time step, i.e. $f(x(t+1)) < f(x(t))$. This condition is satisfied for the above function f when $|x(t) - \delta\nabla f(x(t))| < x(t)$. An analysis of the gradient $f'(x(t))$ yields the following observations:

- When $x(0) = 1$ or $x(0) = -1$, the algorithm will oscillate between the positions of 1 and -1 indefinitely. Consider that if $x(0) = 1$, $x(1) = x(0) - \delta\nabla f(x(0)) = -1$. Similarly, given $x(1) = -1$, it follows that $x(2) = 1$, and so the sequence carries on.

- When $x(0) \in (-1, 1)$, the algorithm will exhibit convergent behaviour. Each subsequent position update satisfies the convergence condition above, and therefore $\lim_{t\to\infty} f(x(t)) = 0$.

- When $x(0) \notin [-1, 1]$, the algorithm will exhibit divergent behaviour. Each subsequent position update is greater than the previous, leading to $\lim_{t\to\infty} f(x(t)) = \infty$.

Figure 3.1 shows the different areas of the objective function f. As an example, the trajectory of $x(0) = 1.1$ is shown to be divergent.

Further observations reveal that if $x(t) \notin [-1.48, 1.48]$ the subsequent position update will be large enough such that $x(t+1) \notin [-5, 5]$, i.e. outside the boundary constraints. It is useful to note that given a random position drawn from a uniform distribution such that $x(0) \in [-5, 5]$, the algorithm will converge just less than 10% of the time, and will diverge 90% of the time. Approximately 85% of all possible starting positions result in an update to the position which immediately removes it from the feasible area. Considering a 2 dimensional version of f, i.e. $f(x_1, x_2) = x_1^4 + x_2^4$, a simple numerical analysis reveals the situation would be even worse, with the percentages being 4% for convergence, 96% for divergence and just over 91% of starting positions resulting in the algorithm immediately leaving the feasible area. The increase of the scope of the problem is due to the fact that in the 2 dimensional case, if either x_1 or x_2 are outside of the

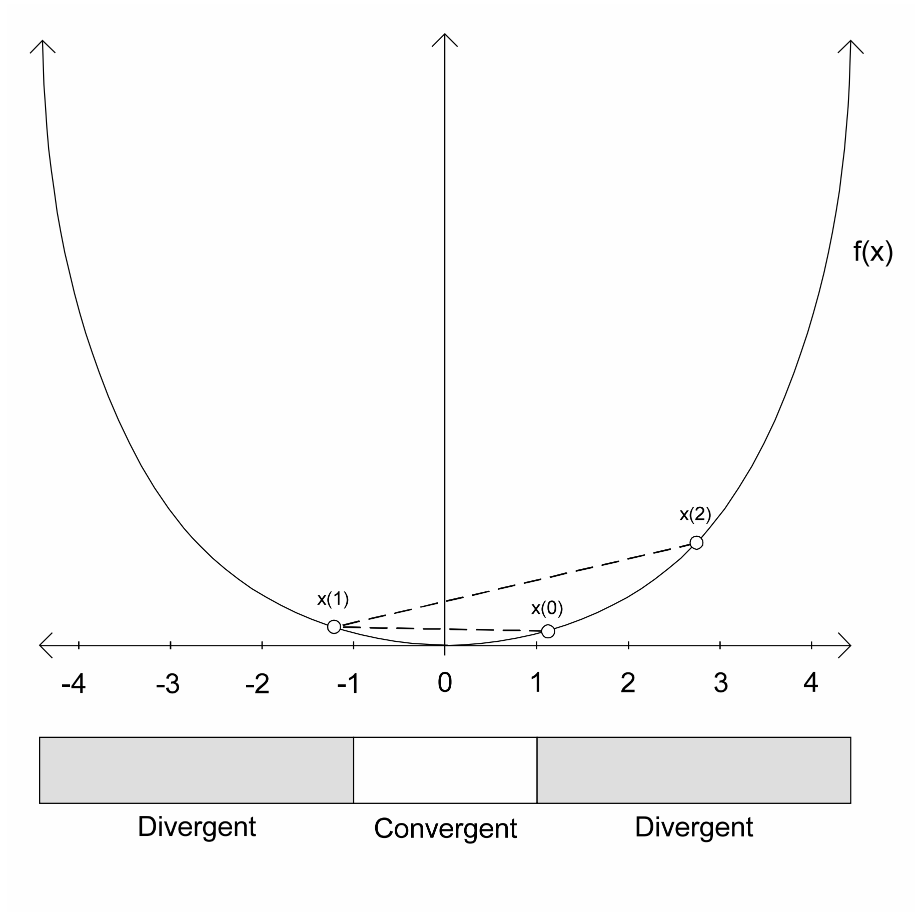

Figure 3.1: Behaviour of gradient descent where $f(x) = x^4$ and $\delta = 0.5$

convergent range, the resulting position update itself will be outside of bounds in at least 1 dimension, making the entire solution invalid.

This analysis is possible due to the simple nature of the objective function; other objective functions have far more complex surfaces that would require far more complex analyses. The analysis presented above was not intended to create any formal definitions of when gradient based algorithms will converge or diverge, rather it is simply meant to highlight that under certain conditions gradient based methods can overshoot local minima, resulting in a new solution that is possibly in a new region of the search space. This new region of the search space can even be outside the boundary constraints of the problem, resulting in a solution which is not usable. There is no guarantee that the algorithm will be able to return to the feasible region without additional intervention, resulting in further solutions which are not usable.

Numerous approaches can be applied to solving, or lessening the impact of this problem. The approaches are generally concerned with the selection of the step size parameter, some of which have been mentioned in Section 2.2.2. A short discussion regarding the approaches, and their applicability to the GDPSO and LFPSO algorithms follows.

Selecting smaller step sizes

One possible solution to the problem of large gradients would be to simply use smaller step sizes - in the case of gradient descent, this would mean a smaller value for δ. The main problem with this approach is that a very small value for δ, whilst preventing large position updates, also slows down the rate of exploitation near flat regions. In the case of objective functions which contain both large, flat regions as well as steep ones, convergence would likely be unacceptably slow.

Consider the function $f(x) = x^4$ from above. A step size of $\delta < 0.02$ would mean that all starting positions within the entire search space of $[-5, 5]$ would converge on x^*. Given this step size, and a starting position of $x(0) = 0.1$, it would take the gradient descent algorithm almost 2500 iterations to reach a position where $x(t) \in (-0.01, 0.01)$.

This observation highlights the tendency of objective functions to have regions with extremely steep gradients as well as regions with very shallow gradients. While selecting small step sizes potentially solves the problem of large gradients, the efficacy of the algorithm in flat regions is negatively affected.

It is worthwhile to note that small step sizes may also increase the tendency of the gradient algorithm to get stuck in local minima. This is particularly relevant in the realm of highly multimodal problems, where a large number of bad local minima may exist. In this thesis, however, gradient algorithms are primarily employed for their local optimisation characteristics. While the possibility of the gradient algorithms to overshoot bad local minima does exist, rather than relying on this possibility, the task of finding good areas to explore is largely left to the particles in the hybrid swarm which are not using gradient methods.

Step sizes decreasing with time

The step size can be made to be a function of time, where time is the number of iterations the algorithm has encountered. One approach might be to define lower and upper bounds for the step size, and to interpolate the step size between these two extremes, based on the progress of the algorithm. The progress would typically be calculated as the number of iterations performed out of the maximum allowed.

However, this approach does not completely address the issue. Starting with a large step size puts the algorithm at risk of region-hopping, or even leaving the feasible area entirely. Furthermore, the mechanism works under the assumption that the gradient of the objective function would be smaller with increased time steps. While this assumption holds for many benchmark functions, it is not always the case.

Clamping of position updates

The concept of V_{max} (definition 2.31 from chapter 2) can also be applied to gradient-based algorithms. When a steep gradient is encountered (which would potentially cause a large update in position), the change in position can be limited to some predefined maximum amount.

Such clamping mechanisms are typically implemented per-dimension, so that the gradient component in each dimension is updated in isolation from the other gradient components. This process results in a change of direction when clamping is applied. The method is typically preferred over more complex ones because it is easy to implement and computationally less intensive.

One issue with clamping is that its application changes the direction and length of the position update vector, as mentioned in Section 2.4.6 and [20]. In the case of the LeapFrog algorithm, where conservation of momentum of the particle is a primary concern [52], the application of V_{max} would not be advisable. It should be noted that the LeapFrog algorithm already implements a mechanism for the control of the size of position updates via the parameter δ.

Application to gradient-based PSO

In hybrid algorithms such as the GDPSO and LFPSO, gradient-based algorithms are employed specifically for their exploitation, i.e. local search ability. In this situation, even if the algorithm finds a solution in a new region of better quality than the previous region, it could be argued that this region-hopping behaviour is undesirable, even if in some cases it may be beneficial. By overshooting the local minimum, the algorithm has displayed an inability to fully exploit its immediate surroundings. Furthermore, finding new regions of interest is a function of exploration rather than exploitation, and therefore should be handled by the other algorithms in the hybrid.

With the above reasoning in mind, it is proposed that when employing a gradient-based algorithm to perform local search in PSO, it is undesirable to have the gradient algorithm perform region-hopping. By appropriately limiting the function of the gradient algorithm to exploitation only, the workings of the hybrid algorithm are more easily understood. This makes the task of selecting reasonable default parameters much easier, as well as applying logic to select any modifications to the hybrid algorithm.

3.4 Gradient Estimation

In real-world scenarios, it cannot be assumed that gradient information will be available for the problem at hand. In this section, a way of calculating a relatively accurate estimate of the gradient is shown.

Given a function, $f(\mathbf{x})$, where $\mathbf{x} \in \mathbb{R}^n$, an estimate of $\nabla f(\mathbf{x})$ needs to be calculated. From first principles [31], $\nabla f(\mathbf{x})$ can be expanded as follows:

$$\nabla f(\mathbf{x}) = [\frac{\partial}{\partial x_1} f(\mathbf{x}), \frac{\partial}{\partial x_2} f(\mathbf{x}), ..., \frac{\partial}{\partial x_n} f(\mathbf{x})]^T \tag{3.1}$$

In turn, the partial derivatives can be written as:

$$\frac{\partial}{\partial x_j} f(\mathbf{x}) = \lim_{\Delta x \to 0} \frac{f([x_1, ..., x_j + \Delta x, ..., x_n]) - f([x_1, x_2, ..., x_n])}{\Delta x} \tag{3.2}$$

where $j \in [1, n]$.

An approximate solution for equation (3.2) can be found by assigning an arbitrarily small value for Δx.

The original HGPSO [40] uses this method of gradient approximation, stated in a slightly different form in equation (2.43):

$$\frac{\partial}{\partial x_i} f(\mathbf{x}) = \frac{f(\mathbf{x} + E_i \epsilon) - f(\mathbf{x})}{\epsilon} \tag{3.3}$$

where ϵ is an arbitrarily small number, conceptually identical to Δx, and E_i is the i^{th} standard basis vector for $\mathbb{R}^n$. The notation used to describe the HGPSO is thus mathematically identical, but manages to be more terse by using a vector notation.

By using small values for Δx, this approach can be used to obtain gradient approximations that exhibit a high degree of accuracy (as evident from equation (3.2)). Whilst a simple mechanism to implement, it unfortunately does not scale well with increasing dimensionality; to approximate the gradient of a function of n dimensions, n evaluations of the function are required.

In this thesis, the actual gradient of the benchmark functions was used for experimental runs with the exception of the HGPSO algorithm [40], which was originally created with a means for estimating the gradient. The actual gradient is defined as the partial derivative of the function with respect to each dimension.

3.5 Conclusion

This chapter discussed the concept of making use of gradient information within PSO. An approach for the creation of heterogeneous gradient hybrid PSOs was given, along with the rationale behind such an algorithm. The differences and similarities between this new approach and existing ones was investigated. The GDPSO and LFPSO were introduced, along with a generalisation of the two, the GLSPSO. The potential problems of using gradient information, namely premature convergence and large gradients, was investigated. Finally, an approach to estimate the gradient of a function by first principles was given. The remainder of the thesis will be concerned with empirical experimentation, specifically the experimental process used, the empirical evidence itself, and conclusions derived from it.

Chapter 4

Experimental Methodology

The experimental methodology used to obtain empirical results is discussed in this chapter. Section 4.1 details the benchmark functions used. Section 4.2 discusses the process used to shift the functions. The use of external libraries is discussed in section 4.3, while section 4.4 describes the selection of algorithm parameters. A range of implementation specific topics is discussed in section 4.5. Sections 4.6 and 4.7 discuss measurements and statistical comparisons, respectively. Finally, a conclusion to the chapter is given in section 4.8.

The intention of this chapter is to disclose in full the experimental process used to obtain empirical evidence, as well as any implementation-specific details.

4.1 Benchmark Functions

Benchmark functions are sample objective functions which can be used to compare the performance of algorithms.

The functions in this section have been selected in part for their use in related literature [45, 52, 53, 54, 9], which makes them suitable for empirical analysis and comparison to other algorithms. Their availability and correctness as part of the Computational Intelligence Library (CILib) [41] was an important consideration as well (see section 4.3). The functions also exhibit different characteristics, which allow for testing hypotheses which attempt to find a correlation between the performance of specific algorithms and the characteristics of the benchmark functions. Below are a list of characteristics that were taken into consideration during the selection process - the definitions are applicable in the context of function minimisation algorithms:

- *Modality:* A function is said to be *unimodal* if it has a single mode. In terms of function minimisation this implies a single trough, i.e., a single "best" solution. A *multimodal* function has many troughs of varying depths, representing solutions that are sub-optimal on a global scale, but are the best solutions in their immediate area. These are known as local minima, and algorithms which focus on finding good solutions in a localised area (known as *local optimisers*) have a tendency to become trapped in these sub-optimal solutions. Gradient-based algorithms traditionally fall into this category.
- *Interaction between components:* Many benchmark functions do not have interaction between their components (e.g. the Spherical function below). Certain algorithms might take advantage of this fact, reducing a complex problem of n dimensions into n one-dimensional problems. The search space of possible solutions would then scale in a linear fashion with increased dimensionality, instead of exponentially as is the case where there is interaction between the components. This fact played a role in the selection of functions that do exhibit interaction between components, such as the Rosenbrock function.
- *Symmetry:* Benchmark functions display symmetry to different degrees. Some, such as the Spherical function, exhibit several types of symmetry, such as reflection symmetry (where some part of the function is mirrored or reflected through some axis) and rotation symmetry (where the function is the same after a certain amount of rotation). Other functions may display different, more subtle types of symmetry (e.g. localised parts of Six Hump Camel Back), or no symmetry at all (Schwefel, Quartic function with noise).
- *Variation in gradients:* Gradient methods can have trouble in dealing with functions which have large gradients (refer to section 3.3.2). It's important to select functions which exhibit a selection of relatively flat and hilly terrains.

The use of gradient algorithms necessitated the availability of gradient information. While section 3.4 discussed the concept of gradient estimation, the mechanism given in that section was not used for experimental runs involving GD, GDPSO and LFPSO. The actual gradient was used for all experiments, with the exception of the HGPSO algorithm [40], which was originally created with a means for estimating the gradient. This played an important role in the selection of the functions, since they would ideally need to be differentiable, and by implication also continuous. The partial derivative with respect to each component was calculated in order to provide the actual gradient.

A definition and description of each of the benchmark functions used in this thesis follows. Each function is also accompanied by one or two graphs showing various features in a graphical way. These renditions are a two-dimensional representation of the actual function, where $x = x_1$ and $y = x_2$, and the z-axis represents the value of the objective function at the relevant coordinates. It is important to note that the graphs do not necessarily show the entire domain of the function, but rather a limited domain selected in order to highlight interesting features of the function.

Spherical (f_1):

Spherical is a simple, unimodal function with no interaction between its components. It is one of the simplest optimisation problems available. It exhibits both reflection and rotation symmetry, and also has a special condition which allows certain gradient algorithms to trivially solve it; specifically, the gradient of the function is exactly twice the change in position needed to reach the global minimum, for all points on the function. This allows, for example, a gradient descent algorithm with $\delta = 0.5$ to find the global minimum in exactly one iteration.

A formal definition of the spherical function is given in equation (4.1), followed by the partial derivative given in equation (4.2). A graphical representation of the 2-dimensional version of the spherical function is given in figure (4.1).

$$f_1(\mathbf{x}) = \sum_{i=1}^{n} x_i^2 \tag{4.1}$$

$$\frac{\partial}{\partial x_i} f_1(\mathbf{x}) = 2x_i \tag{4.2}$$

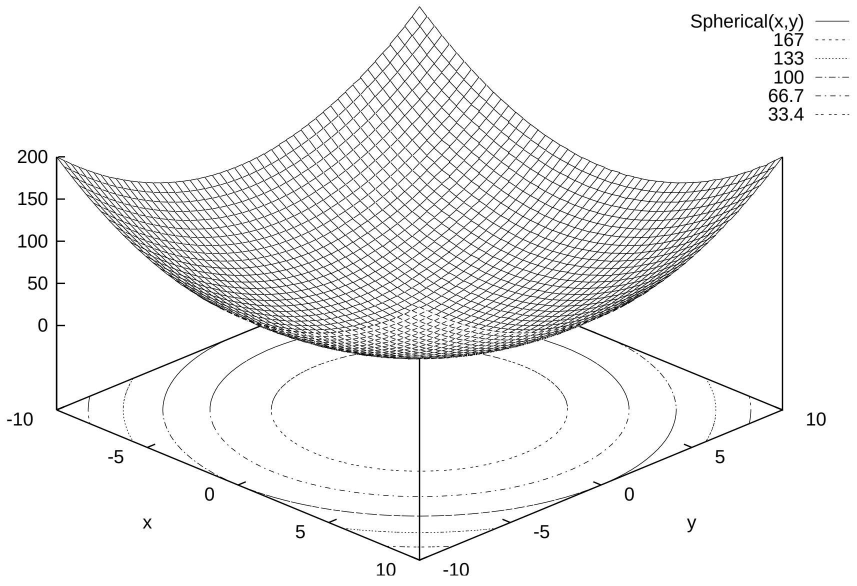

Figure 4.1: f_1 - The Spherical function

where $x_i \in [-5.12, 5.12]$, $n = 30$ and $\mathbf{x}^* = [0, ..., 0]$ with $f_1(\mathbf{x}^*) = 0$.

Rosenbrock (f_2):

Rosenbrock exhibits a parabolic valley which is shallow enough that gradient optimisers have trouble finding a good direction towards the minimum [52, 53, 54]. There is also some interaction between the components of the function. Later on, in the experimental results chapter's table 5.5, it will be shown that Rosenbrock also exhibits the largest differences in its gradient of any of the benchmark functions. This also makes it difficult to configure algorithms such as Gradient Descent to perform well on it.

A formal definition of the Rosenbrock function is given in equation (4.3). The partial derivative is given in equations (4.4), (4.5) and (4.6). A graphical representation of the 2-dimensional version of the Rosenbrock function is given in figure (4.2).

$$f_2(\mathbf{x}) = \sum_{i=1}^{n-1} \left[100(x_{i+1} - x_i^2)^2 + (x_i - 1)^2\right] \tag{4.3}$$

$$\frac{\partial}{\partial x_1} f_2(\mathbf{x}) = -400x_1x_2 + 400x_1^3 - 2 + 2x_1 \tag{4.4}$$

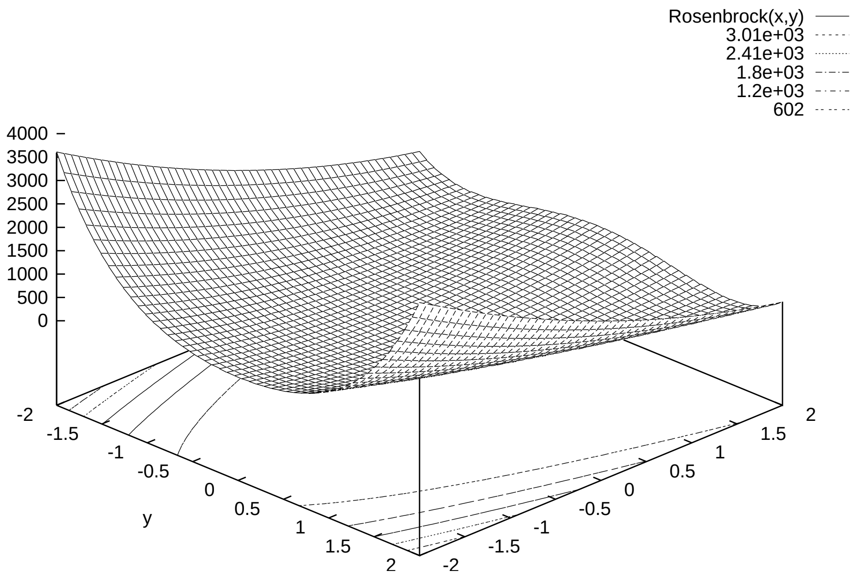

Figure 4.2: f_2 - The Rosenbrock function

$$\frac{\partial}{\partial x_i} f_2(\mathbf{x}) = 200x_i - 200x_{i-1}^2 - 400x_i x_{i+1} + 400x_i^3 - 2 + 2x_i \tag{4.5}$$

where $2 <= i <= n - 1$.

$$\frac{\partial}{\partial x_n} f_2(\mathbf{x}) = 200x_n - 200x_{n-1}^2 \tag{4.6}$$

where $x_i \in [-2.048, 2.048], N = 30$ and $\mathbf{x}^* = [1, ..., 1]$ with $f_2(\mathbf{x}^*) = 0$.

Quartic Function with Noise (f_3):

The Quartic Function with Noise (Quartic) is a unimodal function with steep sides that level out rapidly towards the origin, making it difficult for optimisation algorithms to find good directions. A random noise component, sampled from a uniform distribution, is added to the function to simulate real-world conditions. The addition of the random noise also means that the function does not exhibit any symmetries. Without the random component, the function displays both reflection and rotational symmetry.

A formal definition of the quartic function with noise is given in equation (4.7), followed by the partial derivative given in equation (4.8). Figures (4.3) and (4.4) give graphical representations of the 2-dimensional version of the quartic function without and with random noise, respectively.

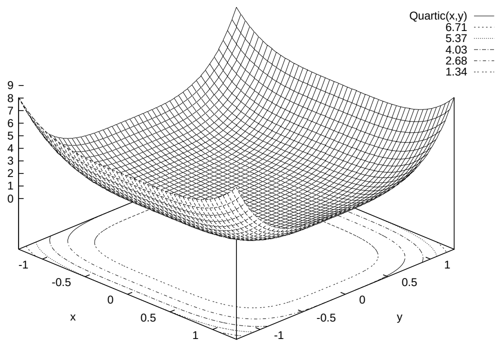

Figure 4.3: f_3 - The Quartic function, without random noise

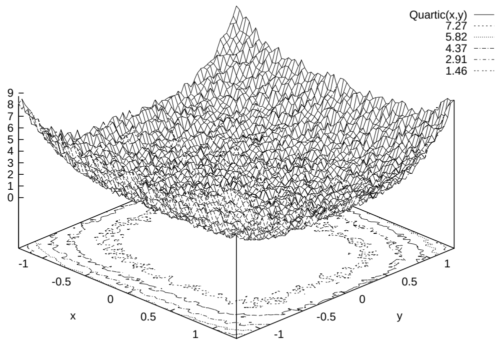

Figure 4.4: f_3 - The Quartic function, with random noise

$$f_3(\mathbf{x}) = \sum_{i=1}^{N} i x_i^4 + random[0,1) \tag{4.7}$$

$$\frac{\partial}{\partial x_i} f_3(\mathbf{x}) = 4 i x_i^3 \tag{4.8}$$

where $x_i \in [-1.28, 1.28], N = 30$ and $\mathbf{x}^* = [0, ..., 0]$ with $f_3(\mathbf{x}^*) = 0$. The random component is sampled from a uniform distribution.

Generalised Schwefel's Problem 2.26 (f_4):

Generalised Schwefel's Problem (Schwefel) is a multimodal function with a large number of local minima. The function differs from the previous functions in that the global minimum is located close to the boundary constraint of the function. The Schwefel function does not seem to exhibit any symmetries.

A formal definition of the generalised Schwefel's problem is given in equation (4.9), followed by the partial derivative given in equation (4.10). A graphical representation of the 2-dimensional version of the generalised Schwefel's problem is given in figure (4.5).

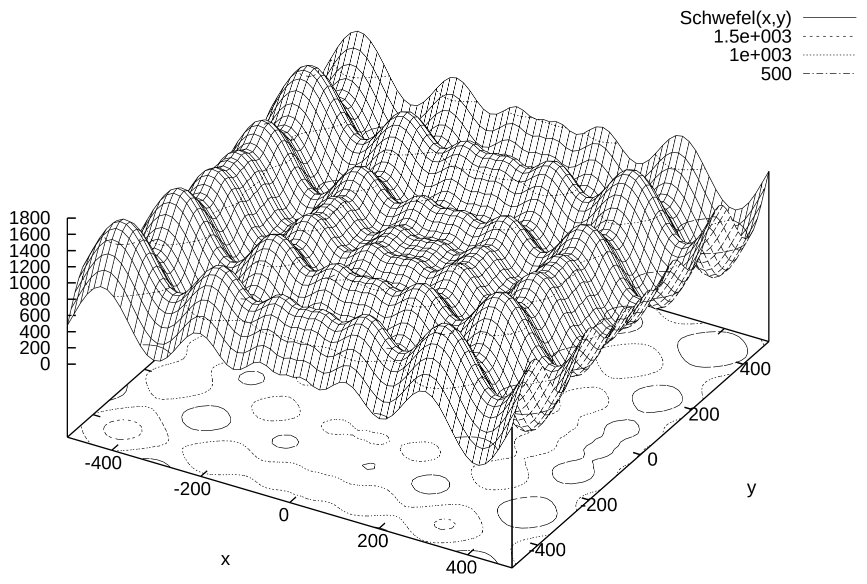

Figure 4.5: f_4 - The Schwefel function

$$f_4(\mathbf{x}) = -\sum_{i=1}^{N} \left(x_i sin(\sqrt{|x_i|}\right) \tag{4.9}$$

$$\frac{\partial}{\partial x_i} f_4(\mathbf{x}) = -x_i cos(\sqrt{|x_i|}) \frac{1}{2} |x_i|^{-\frac{1}{2}} \frac{x_i}{|x_i|} - sin(\sqrt{|x_i|}) \tag{4.10}$$

where $x_i \in [-512, 512], N = 30$ and $\mathbf{x}^* = [-420.9687, -420.9687, ..., -420.9687]$ with $f_4(\mathbf{x}^*) = 0$.

Generalised Rastrigin Function (f_5):

The generalised Rastrigin function (Rastrigin) is a multimodal function, with a large number of local minima.

A formal definition of the Rastrigin function in equation (4.11), followed by the partial derivative given in equation (4.12). Figures (4.6) and (4.7) give graphical representations of the 2-dimensional version of the Rastrigin function. The Rastrigin function exhibits both reflection and rotational symmetry.

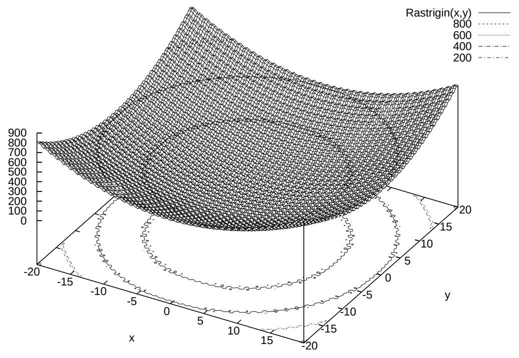

Figure 4.6: f_5 - The Rastrigin function, shown over a large domain

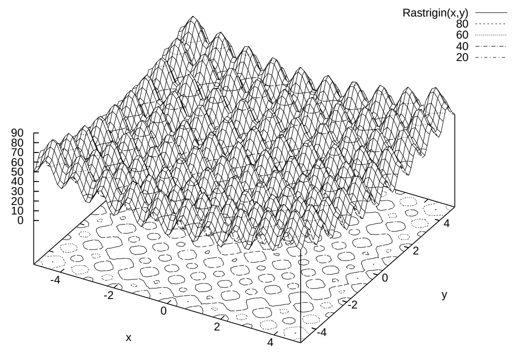

Figure 4.7: f_5 - The Rastrigin function, shown over a small domain

$$f_5(\mathbf{x}) = \sum_{i=1}^{N} \left[x_i^2 - 10cos(2\pi x_i) + 10\right] \tag{4.11}$$

$$\frac{\partial}{\partial x_i} f_5(\mathbf{x}) = 2x_i + 20\pi sin(2\pi x_i) \tag{4.12}$$

where $x_i \in [-5.12, 5.12], N = 30$ and $\mathbf{x}^* = [0, ..., 0]$ with $f_5(\mathbf{x}^*) = 0$.

Ackley's Function (f_6):

Ackley's function (Ackley) is a multimodal function, first defined in [1] and later generalised in [3]. It exhibits no interaction between the components, but is highly multimodal.

A formal definition of the Ackley's function in equation (4.13), followed by the partial derivative given in equation (4.14). Figures (4.8) and (4.9) give graphical representations of the 2-dimensional version of the Ackley function. The Ackley function exhibits both reflection and rotational symmetry.

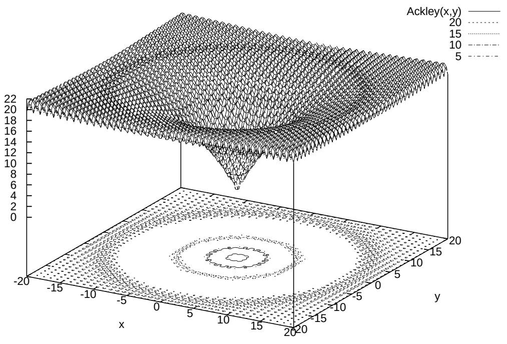

Figure 4.8: f_6 - Ackley's function, shown over a large domain

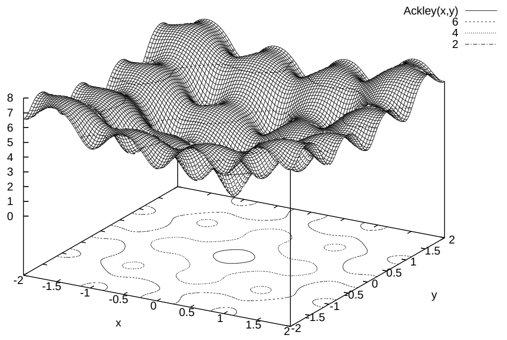

Figure 4.9: f_6 - Ackley's function, shown over a small domain

$$f_6(\mathbf{x}) = -20exp\left(-0.2\sqrt{\frac{1}{30}\sum_{i=1}^{N} x_i^2}\right) - exp\left(\frac{1}{N}\sum_{i=1}^{N} cos(2\pi x_i)\right) \tag{4.13}$$

$$\begin{aligned}\frac{\partial}{\partial x_i} f_6(\mathbf{x}) &= \frac{2x_i}{15}(\frac{1}{N}\sum_{i=1}^{N} x_i)^{-\frac{1}{2}} exp\left(-0.2\sqrt{\frac{1}{N}\sum_{i=1}^{N} x_i^2}\right) \\ &+ \frac{\pi}{15} sin(2\pi x_i) exp\left(\frac{1}{N}\sum_{i=1}^{N} cos(2\pi x_i)\right)\end{aligned} \tag{4.14}$$

where $x_i \in [-30, 30], N = 30$ and $\mathbf{x}^* = [0, ..., 0]$ with $f_6(\mathbf{x}^*) = 0$.

Generalised Griewank Function (f_7):

The generalised Griewank function (Griewank) is a highly multimodal function, where the product term introduces significant interaction between its components.

A formal definition of the Griewank function is given in equation (4.13), followed by the partial derivative given in equation (4.16). Figures (4.10) and (4.11) give graphical representations of the 2-dimensional version of the Griewank function. Griewank has been shown to become easier to solve in higher dimensions [62]. The Griewank function exhibits both reflection and rotational symmetry.

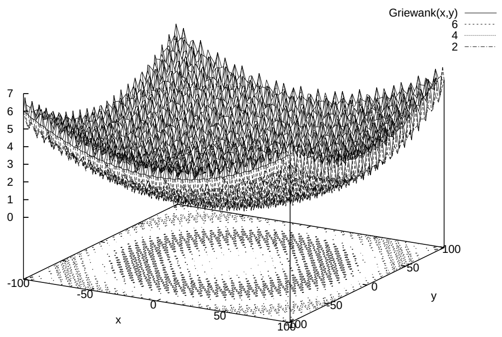

Figure 4.10: f_7 - The Generalised Griewank function, shown over a large domain

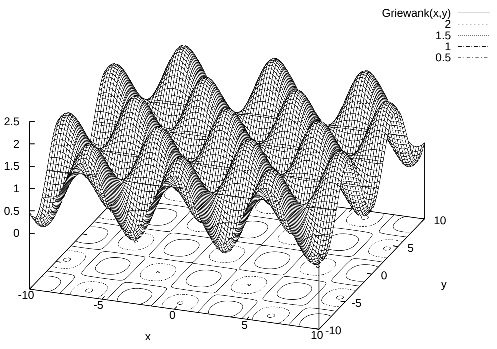

Figure 4.11: f_7 - The Generalised Griewank function, shown over a small domain

$$f_7(\mathbf{x}) = \frac{1}{4000}\sum_{i=1}^{N} x_i^2 - \prod_{i=1}^{N} cos\left(\frac{x_i}{\sqrt{i}}\right) + 1 \tag{4.15}$$

$$\frac{\partial}{\partial x_i} f_7 = \frac{1}{2000}x_i + \frac{\prod_{j=1}^{N} cos\left(\frac{x_j}{\sqrt{j}}\right)}{cos\left(\frac{x_i}{\sqrt{i}}\right)} sin(\frac{x_i}{\sqrt{i}})\frac{1}{\sqrt{i}} \tag{4.16}$$

where $x_i \in [-600, 600], N = 30$ and $\mathbf{x}^* = [0, ..., 0]$ with $f_7(\mathbf{x}^*) = 0$.

Six Hump Camel Back Function (f_8):

The six hump camel back function is a relatively simple, 2-dimensional function with 6 possible minima in the domain.

A formal definition of the six hump camel back function is given in equation (4.17). The partial derivative is given in equations (4.18) and (4.19). A graphical representation of the 2-dimensional version of the problem is given in figure (4.12).

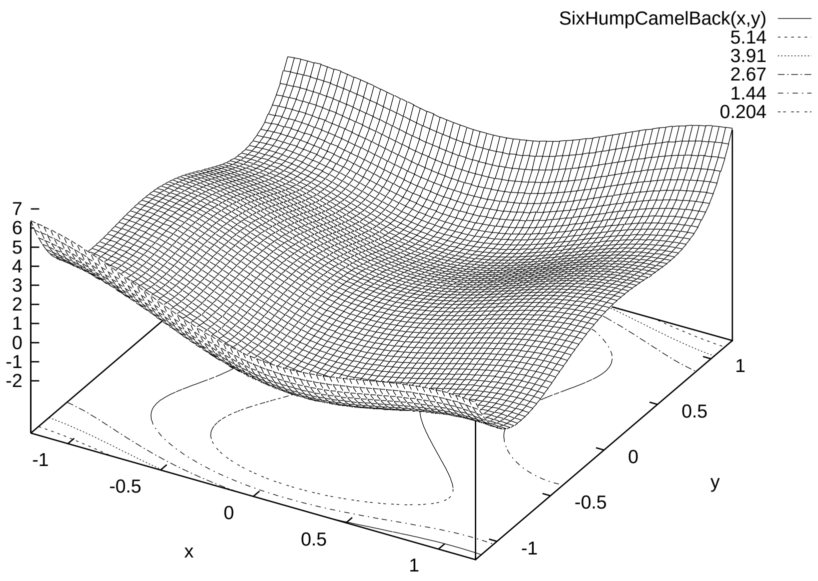

Figure 4.12: f_8 - Six-hump Camel-Back function

$$f_8(x) = 4x_1^2 - 2.1x_1^4 + \frac{1}{3}x_1^6 + x_1x_2 - 4x_2^2 + 4x_2^4 \tag{4.17}$$

$$\frac{\partial}{\partial x_1} f_8 = 8x_1 + 8.2x_1^3 + 2x_1^5 + x_2 \tag{4.18}$$

$$\frac{\partial}{\partial x_2} f_8 = x_1 - 8x_2 + 16x_2^3 \tag{4.19}$$

where $x_1 \in [-3, 3], x_2 \in [-2, 2]$ and $\mathbf{x}^* = [-0.0898, 0.7126], [0.0898, -0.7126]$ with $f_8(\mathbf{x}^*) = -1.0316$.

4.2 Shifted Benchmark Functions

Liang *et al.* [32] argued that in recent years, optimisation algorithms have been developed to exploit certain characteristics in typical benchmark functions. These characteristics in benchmark functions detract from their value as good benchmark functions. Liang *et al.* listed the following characteristics that can be unfairly exploited by algorithms:

1. global optimum has the same value for different dimensions;
2. global optimum is at the origin;
3. global optimum is lying at the centre of the search range;
4. global optimum is lying on the boundary;
5. local optima are lying along coordinate axes or there is no linkage between dimensions.

Liang *et al.* [32] suggest shifting and rotating the functions to alleviate the above problems. Whilst function rotation is usually a good way to deal with the above issues, it greatly complicates the calculation of the partial derivatives (and hence the gradient) of the function. It was decided that for this reason, only function shifting would be used. This allowed for the partial derivatives of the non-shifted function to be used on the shifted versions. The method of function shifting was adopted from Liang *et al.* [32]:

$$F(\mathbf{x}) = f(\mathbf{x} - \mathbf{o}_{new} + \mathbf{o}_{old}) \tag{4.20}$$

where $f(\mathbf{x})$ is the original function, $F(\mathbf{x})$ is the shifted function, $\mathbf{o}_{old}$ is the global optimum of the original function and $\mathbf{o}_{new}$ is the new global optimum. The value of $\mathbf{o}_{new}$ is selected so that the new global optimum has different values for different dimensions, and it is not in the centre of the search space.

In order to ensure fair comparison across the different algorithms, each function was paired with a single, randomly generated $\mathbf{o}_{new}$. As a result, the various algorithms were tested on the same functions, with the same random shift in global optimum for each algorithm. The random numbers were obtained using the Mersenne Twister [36] random number generator.

Two main issues needed to be addressed when deciding on the range for the random shifts:

- The shifted global optimum should still be within the domain of the original function, and
- the shift must not introduce new global minima to the function.

To address the first issue, $\mathbf{o}_{new}$ was sampled from a random distribution such that:

$$\mathbf{o}_{new,i} \in [-\text{shiftboundary}_i, \text{shiftboundary}_i] \tag{4.21}$$

where $\text{shiftboundary}_i = \min(o_{old,i} - lb_i, ub_i - o_{old,i})$; lb_i and ub_i are the lower and upper boundary constraint respectively of the function being shifted, in the i-th dimension.

The issue regarding the introduction of new global minima is a more difficult problem to deal with. Essentially, an analysis of each function needed to be done to ensure that a shift in the function does not introduce any new global minima, or alternatively that the area outside the original domain does not contain any new global minima (this is relatively simple with unimodal functions, for example). This analysis can be done using several different approaches, a full treatment of which is outside the scope of this thesis.

Functions Spherical, Quartic, Rastrigin, Ackley and Griewank were shifted by a random amount sampled from each respective function's complete domain. Since in the case of the above functions the global optimum is located at the center of the domain for each dimension, this ensured that the shifted global optimum would still be part of the original domain. In the case of Rosenbrock, the global optimum is located off-center, and as a result the random shift was selected from a range that would ensure the global optimum remains in the original domain. The unshifted Schwefel function's global optimum is already significantly shifted compared to the origin, and the area outside of the domain also contains better minima than the original global minimum. As a result, it was decided that the Schwefel function would not be shifted. The six hump camel back function is likewise already shifted to different degrees on both of its two dimensions, therefore no further shifting was required. The exact range and random seed values used for the shift generation process is summarised in table (4.1). The precise shift amounts that resulted from the above process are given fully in Appendix A for completeness. All empirical results presented in this thesis were obtained using the above shifted versions of the functions.

4.3 External Libraries Used

To ensure openness and implementation correctness, the University of Pretoria's open source Computational Intelligence Library (CILib) [41] was used for all experiments. A private fork of the source code was eventually maintained due to the rapid development of the library and the associated breaking changes, as well as for the complete repeatability of experiments.

4.4 Algorithm Parameters

The algorithms benchmarked have various parameters which can affect the algorithm's performance. The values of these parameters play a significant role in the suitability of a given algorithm for a specific problem. Unless otherwise stated, reasonable defaults were used. In general, this includes the PSO parameters c_1, c_2 and w, where the defaults have been obtained from Van den Bergh's suggested values [9], since they satisfy Van den Bergh's convergence condition, $w > \frac{1}{2}(c_1+c_2)-1$. In specific experiments, attempts were made to counter specific behaviour by varying c_1 and c_2, or delaying the introduction of gradient information. In these cases, the experiments and algorithms were clearly labelled according to the schemes given in sections 4.4.2 and 4.4.3.

Table 4.2 lists the default PSO parameters used by all algorithms, unless specifically stated otherwise.

4.4.1 Algorithm Parameters for GCPSO

All GCPSO-specific parameters were set to the default CILib parameters [41], which are identical to the ones suggested by Van den Bergh [9], and are given in table 4.3. The standard PSO defaults were used for all other parameters, given in table 4.2.

Table 4.1: Function Shift Parameters

Function	Shifted	Shift Range	Random Seed
Spherical (f_1)	Yes	[-5.12, 5.12]	5001
Rosenbrock (f_2)	Yes	[-1, 1]	5002
Quartic (f_3)	Yes	[-1.28, 1.28]	5003
Schwefel (f_4)	No		
Rastrigin (f_5)	Yes	[-5.12, 5.12]	5005
Ackley (f_6)	Yes	[-30, 30]	5006
Griewank (f_7)	Yes	[-600, 600]	5007
Six Hump Camel Back (f_8)	No		

4.4.2 Algorithm Parameters for GDPSO

The experiments make use of the following labeling scheme to differentiate between slightly different versions of GDPSO:

- GDPSO: The basic version of GDPSO, which uses the PSO defaults, and a function specific step-size (given in table 4.4).
- GDPSO$_M$: GDPSO, but with function specific values for c_1 and c_2, in order to counter premature convergence.
- GDPSO$_D$: GDPSO, but with delayed introduction of gradient information. The values for c_1 and c_2 are the PSO defaults, but function specific values are used to introduce the gradient algorithm at a certain percentage of completion of the experiment, according to the mechanism discussed in section 3.3.1.
- GDPSO$_{DM}$: GDPSO with both function specific values for c_1 and c_2, as well as delayed introduction of gradient information.

Table 4.4 gives the parameters used by GDPSO, where δ represents the step size, and ρ_{delay} the factor by which the introduction gradient information is delayed for the GDPSO$_D$ variant. The optimised parameter values were obtained via experimentation with different combinations of values, the values which seemed to result in the best

Table 4.2: Default Algorithm Parameters for PSO

Parameter	Value
w	0.729844
c_1	1.496180
c_2	1.496180

performance being used. To minimise the complexity of the experiments, the values were optimised in isolation, with the exception of c_1 and c_2. It should be pointed out that the standard GDPSO (as opposed to the variants GDPSO$_M$, GDPSO$_D$ and GDPSO$_{DM}$) uses the unoptimised PSO defaults for c_1 and c_2, given in table 4.2.

4.4.3 Algorithm Parameters for LFPSO

The parametrisation scheme for LFPSO is similar to that used for GDPSO given in section 4.4.2. As with GDPSO, four versions of LFPSO are labelled distinctly. LFPSO represents a basic version with values for c_1 and c_2 from table 4.2. LFPSO$_D$ is the same algorithm, but with the introduction of gradient information being delayed according to the mechanism discussed in section 3.3.1. LFPSO$_M$ is LFPSO with values for c_1 and c_2 being set to different values per function. LFPSO$_{DM}$ uses both function specific values for c_1 and c_2 as well as the delayed introduction of gradient information.

The parameters used for LFPSO are given in table 4.5. As in the case of GDPSO, the standard LFPSO (as opposed to LFPSO$_M$, LFPSO$_D$ and LFPSO$_{DM}$) uses the standard PSO defaults for c_1 and c_2, given in table 4.2.

4.4.4 Algorithm Parameters for Mutating PSO

The standard CILib [41] PSO parameters given in table 4.2 were used for Mutating PSO.

4.4.5 Algorithm Parameters for HGPSO

Since both the GDPSO and HGPSO make use of the gradient descent algorithm, the process yielding optimised values for δ was identical in both cases, yielding to the same values being obtained. The values used for δ in HGPSO are given in table 4.6. The

Table 4.3: Default Algorithm Parameters for GCPSO

Parameter	**Value**
ρ	1
s_c	5
f_c	5
ρ expansion coefficient	1.2
ρ contraction coefficient	0.5

Table 4.4: Default Algorithm Parameters for GDPSO

Function	$\boldsymbol{\delta}$	$\boldsymbol{c_1}$	$\boldsymbol{c_2}$	$\boldsymbol{\rho_{delay}}$
Spherical	0.5	1.5	1.0	0.0
Rosenbrock	0.0004	0.25	1.5	0.25
Quartic	0.001	1.5	1.5	0.0
Schwefel	0.001	0.75	1.5	0.75
Rastrigin	0.0005	0.25	1.75	0.25
Ackley	0.005	0.25	1.3	0.25
Griewank	11.0	0.25	2.5	0.25
Six Hump Camel Back	0.01	1.5	0.25	0.0

Table 4.5: Default Algorithm Parameters for LFPSO

Function	$\boldsymbol{\Delta t}$	$\boldsymbol{c_1}$	$\boldsymbol{c_2}$	$\boldsymbol{\rho_{delay}}$
Spherical	0.5	1.5	1.0	0.0
Rosenbrock	0.00005	0.75	1.25	0.0
Quartic	0.00001	0.75	1.0	0.0
Schwefel	1.0	1.5	1.5	0.5
Rastrigin	0.0001	1.5	1.5	0.0
Ackley	1.0	1.49	1.49	0.25
Griewank	0.5	1.0	0.25	0.25
Six Hump Camel Back	0.001	1.5	0.25	0.0

CILib defaults for the standard PSO parameters c_1, c_2 and w were used, given in table 4.2.

4.5 Further Algorithm Assumptions

In this section, a range of implementation-specific topics are discussed. The scheme used to initialise swarms is discussed in section 4.5.1, followed by a discussion of the calculation of random values in section 4.5.2. The exact definition of *gbest* and *lbest* used is given in section 4.5.3. Swarm and neighbourhood size is defined in section 4.5.4. The treatment of particles which leave the acceptable domain is detailed in section 4.5.6. Section 4.5.7 discusses the usage of inertia weights and velocity clamping, while section 4.5.8 clarifies whether the synchronous or asynchronous update model was used.

4.5.1 Initialisation Scheme

Throughout the experiments, the Mersenne Twister [36] random number generator algorithm was used to create a uniform distribution of random numbers. The Mersenne Twister is generally considered to be a high quality random number generator for Monte Carlo simulations, and is the default in a number of mathematical and statistical suites, including R [57] and MATLAB.

The positions of all particles were initialised randomly using the Mersenne Twister, using the domain of the function to be optimised as the bounding area for initialisation.

4.5.2 Other Stochastic Factors

All random numbers, unless explicitly stated otherwise, were sampled from a uniform distribution obtained using the Mersenne Twister algorithm. Specifically, this includes the PSO random components $\mathbf{r}_1$ and $\mathbf{r}_2$.

4.5.3 Global and Neighbourhood Best

As discussed in section 2.4.1, the PSO global best position is defined as the best position found by any particle so far (refer to equation (2.27)).

Table 4.6: Default Algorithm Parameters for HGPSO

Function	δ
Spherical	0.5
Rosenbrock	0.0004
Quartic	0.001
Schwefel	0.001
Rastrigin	0.0005
Ackley	0.005
Griewank	11.0
Six Hump Camel Back	0.01

4.5.4 Swarm and Neighbourhood Size

All swarms were set up with 20 particles. In the case of swarms where the *lbest* PSO was used, the neighbourhood size was set to 3.

4.5.5 Stopping Conditions

Unless otherwise stated, all PSO algorithms were executed for 10^4 iterations. Under normal circumstances, given the swarm size of 20 particles, this would equate to 2 x 10^5 function evaluations. In the case of gradient PSOs, however, this statement is impossible to confirm, since the gradient particles do not directly evaluate the function itself, but rather its derivative. An attempt at formally equating the evaluation of a function and the evaluation of the partial derivative of a function is outside the scope of this thesis. An exception to this rule is made in section 5.1, where a Gradient Descent algorithm is compared directly to a PSO with 20 particles, giving the PSO an unfair advantage. In this special case, the experiments were run so that the GD algorithm was given the same number of function evaluations as the PSO algorithm.

4.5.6 Domain Constraints

Every function used in the experimental analysis has a finite domain. In certain cases, the regions outside the given boundary may contain better solutions than the global best inside the domain. For this reason, it is essential to use some mechanism to ensure that the swarms do not explore the regions outside the domain of the function. The function minimisation error $F(\mathbf{x}_i)$ was calculated as follows:

$$F(\mathbf{x}_i) = \begin{cases} \infty & \text{if } \exists\, j \mid x_{i,j} \notin [x_{min,j}, x_{max,j}] \\ f(\mathbf{x}_i) & \text{otherwise} \end{cases} \tag{4.22}$$

where $f(\mathbf{x})$ is the benchmark function, $\mathbf{x}_i$ is the position of particle i, $x_{i,j}$ is the value of the j-th dimension of particle i, and $x_{min,j}$ and $x_{max,j}$ are the minimum and maximum allowable values in the j-th dimension of the domain of the function f.

The above scheme was used to ensure that areas outside of the defined boundaries of the problem were not considered as valid results. Since all particles are initialised within the bounds of the problem initially, there will always be a valid *gbest* position, and all particles will have valid *pbest* positions. When a particle leaves the bounds of the problem, the fitness of the particle will be worse than at any position within the bounds of the problem, ensuring that *gbest* and *pbest* are never updated to positions outside of the boundaries. In this way, swarms that are configured with convergent parameters for w, c_1 and c_2, as per equation (2.30), should return to an area within the boundary constraints of the problem.

4.5.7 Inertia Weights and Velocity Clamping

The method of limiting particle velocities was inertia weights, as defined in equations (2.33) and (2.34).

Velocity clamping, as defined in equation (2.31), was not used to limit particle velocities. The use of inertia weights and convergent choices for c_1, c_2 and w meant that velocity clamping would be effectively redundant. Velocity clamping has also been shown to alter the direction and magnitude of the velocity of particles [20], which, in the specific case of the LeapFrog PSO, is not desirable since it interferes with the algorithm's own attempts to manage the kinetic energy of the particle.

It should be noted that the HGPSO [40], which originally made use of velocity clamping, was implemented using inertia weights instead. This was done in order to highlight that the differences in performance (if any) are due to the core differences in the algorithms, not the choice of velocity clamping versus inertia weights.

4.5.8 Velocity Updates

In all experiments, the synchronous PSO (as defined in algorithm 2.3), rather than the asynchronous PSO (as defined in algorithm 2.5) was used. Discussed in sections 2.4.6 and 3.3.1, an asynchronous PSO implies faster spread of information through the swarm, potentially accelerating the tendency for premature convergence. As a result, the synchronous PSO model was used.

4.6 Measurements

In this section, several measurements relating to algorithm performance are described. The equation for the calculation of mean error is given in section 4.6.1. Reliability and efficiency are discussed in sections 4.6.2 and 4.6.3, respectively. The swarm diversity measurement is discussed in section 4.6.4.

4.6.1 Mean Error

The best solution found at the end of a specified number of swarm iterations or function evaluations will be used to represent the solution from the entire algorithm.

For any given algorithm and function f, the mean error $\overline{x}_f$ for is given by:

$$\overline{x}_f = \frac{\sum_{s=1}^{S} f(\hat{\mathbf{y}}_s)}{S} \tag{4.23}$$

where S is the number of experimental runs performed. The experiments in this thesis were performed with $S = 50$.

The mean error is thus simply the average of the best solutions found by the swarm in each experimental run. The values are not modified in order to have a minimum value of 0, although many of the benchmark functions are designed so that their global minimum yields an error of 0. Since we are dealing with minimization problems, small values represent better quality solutions.

4.6.2 Reliability

The reliability of algorithms was measured as the percentage of experimental runs that achieved a mean error below some arbitrary threshold.

Table 4.7 gives the reliability thresholds used. Where thresholds were provided by Van den Bergh [9], they were used in this thesis. Other thresholds were arbitrarily set at some level that would result in some difference being shown between the algorithms being tested.

4.6.3 Efficiency

In this thesis, efficiency is measured as the average number of swarm iterations elapsed before the above reliability threshold was reached. Experimental runs which did not achieve a mean error below the threshold are excluded from the calculation.

4.6.4 Swarm Diversity

Swarm diversity is a measure of the degree to which the particles in the swarm are dispersed. It is an important aspect of swarm-based algorithms, since it allows for a measure of the degree to which a swarm has converged. Swarm diversity was calculated using the following equations (4.24) and (4.25), as taken from [20, 30, 60].

$$diversity(S(t)) = \frac{1}{n}\sum_{i=1}^{n_s}\sqrt{\sum_{j=1}^{n_x}[x_{ij}(t) - \bar{x}_j(t)]^2} \tag{4.24}$$

where $\bar{x}_j(t)$ is the average of the j-th dimension over all particles:

$$\bar{x}_j = \frac{\sum_{i=1}^{x_n} x_{ij}(t)}{n_s} \tag{4.25}$$

4.7 Statistical Comparisons

An open-source statistical tool, the R Project for Statistical Computing [57], was used to perform all statistical calculations. The version of the software used was 2.6.2.

Table 4.7: Reliability Thresholds

Problem	Threshold
Spherical	1.0e-025
Rosenbrock	1.0e+002
Quartic	0.4e+000
Schwefel	6.0e+003
Rastrigin	1.0e+002
Ackley	5.0e+000
Griewank	1.0e+000
Six Hump Camel Back	0.0e+000

To determine whether a particular set of data is normally distributed, the Shapiro-Wilk test [49] for normality at the 0.05 level was used.

In all cases in this thesis, one or more of the data sets would not be normally distributed, and as a result non-parametric statistics would be used to validate hypotheses.

To validate hypotheses, the Mann-Whitney U test [34] at the 0.05 level was used. The Bonferroni correction was used to guard against type I errors, where a rejection of the null hypothesis occurs even though the null hypothesis is true. In order to protect against type II errors, the failure to reject the null hypothesis when the null hypothesis is false, a reasonably large sample size of 50 samples was used. In all cases, an uncorrected p-value is given for completeness.

Mean error measurements are provided as the mean and a 95% confidence interval, calculated using the mean obtained from the samples and the t-function with 49 degrees of freedom.

4.8 Conclusion

This chapter has provided a detailed account of the experimental methodology used to obtain empirical results in this thesis. The benchmark functions themselves, as well as their partial derivatives, were given in full. The rationale and process behind the shifting of the functions is given. All assumptions regarding algorithm implementation and algorithm parameters are given, and the supporting rationale is provided where relevant. The measurements taken during the experiments are detailed, as well as the statistical process used to evaluate the results.

The full disclosure of the experimental methodology is important for the repeatability of experiments and results. Additionally, the transparency in the process allows for validation that the empirical evidence obtained in the remaining chapters was obtained in a scientific and statistically fair way.

Chapter 5

Experimental Results

This chapter presents and analyses empirical results using statistical methods. Section 5.1 examines the difference in performance of gradient descent and PSO. Section 5.2 examines the performance of the GDPSO hybrid algorithm, followed by an analysis of modifications to the GDPSO algorithm in section 5.3. The performance of LFPSO is investigated in section 5.4, followed by an analysis of how gradient information affects the diversity of gradient-hybrid swarms in section 5.4.3. The performance of GDPSO and LFPSO is compared against that of three existing PSOs, the HGPSO, GCPSO and Mutating PSO in section 5.5. Finally, conclusions are given in section 5.6.

5.1 Gradient Algorithms Compared Against PSO

This section examines the performance of gradient descent (GD) against that of standard PSO. Section 5.1.1 aims to determine whether there is any significant difference between the performance of the two algorithms. Section 5.1.2 examines the characteristics of the benchmark functions and correlates these to the results found. Section 5.1.3 formalises the main performance trends, and Section 5.1.4 investigates some additional aspects of the experiment. Section 5.1.5 provides a summary of the findings.

5.1.1 Difference in Performance of GD and PSO

In order to highlight the main differences in the performance of gradient-based algorithms and PSO, the following hypothesis is proposed:

Hypothesis 5.1 *PSO and gradient-based algorithms are not equal in performance.*

To validate hypothesis 5.1, Experiment 1 was created, in which the gradient descent algorithm (GD) is compared against a standard PSO on the 8 benchmark functions described in section 4.1. The shifted versions of the functions, as discussed in section 4.2, were used for Experiment 1, and all experiments in this thesis.

Table 5.1 shows the mean error obtained for each algorithm and problem. Table 5.2 gives the Mann-Whitney U test p-values prior to correction, indicating whether there is a significant difference between the distributions of the error measurement of the respective algorithms. Table 5.3 provides another aspect of algorithm performance, namely reliability. Reliability is defined in the section 4.6.2. In addition, the table also provides the mean number of position updates taken to reach the threshold.

Table 5.2 shows a significant difference in the performance of the two algorithms across all of the benchmark functions. As a result, hypothesis 5.1 is valid.

5.1.2 Correlation of Results and Function Characteristics

Table 5.2 showed that the algorithms exhibited a significant difference in accuracy on all of the benchmark functions. The mean error from Table 5.1 indicates that on some functions PSO performed better than GD, while on others the situation was reversed. The performance of the algorithms can be summarised as follows:

- PSO exhibited superior performance on Schwefel, Rastrigin, Ackley, Griewank and Six Hump Camel Back.
- GD showed superior performance on Spherical, Rosenbrock and Quartic.

Table 5.1: Mean Error for Experiment 1

Problem	**Algorithm**	**Error**
Spherical	GD	0.000e+000 ± 0.000e+000
	PSO	8.883e-029 ± 3.384e-028
Rosenbrock	GD	7.973e-001 ± 1.611e+000
	PSO	4.471e+000 ± 4.496e+000
Quartic	GD	4.354e-001 ± 1.462e-001
	PSO	6.128e-001 ± 1.291e-001
Schwefel	GD	9.833e+003 ± 2.364e+003
	PSO	4.307e+003 ± 5.039e+002
Rastrigin	GD	5.405e+002 ± 1.677e+002
	PSO	1.254e+002 ± 3.469e+001
Ackley	GD	1.987e+001 ± 3.020e-001
	PSO	1.007e+001 ± 6.727e+000
Griewank	GD	1.555e+003 ± 2.770e+002
	PSO	1.430e-001 ± 4.716e-001
Six Hump Camel Back	GD	9.210e-001 ± 1.177e+000
	PSO	-2.845e-005 ± 0.000e+000

Table 5.2: Mann-Whitney U Test P-Values for Experiment 1

Problem	**P-Value**
Spherical	2.20e-016
Rosenbrock	3.64e-013
Quartic	1.65e-004
Schwefel	2.20e-016
Rastrigin	2.20e-016
Ackley	2.20e-016
Griewank	2.20e-016
Six Hump Camel Back	1.02e-009

Table 5.3: Reliability and Efficiency for Experiment 1

Function	**Algorithm**	**Reliability**	**Efficiency**
Spherical	GD	100%	2.000e+001
	PSO	100%	4.958e+004
Rosenbrock	GD	100%	1.000e+003
	PSO	100%	4.760e+003
Quartic	GD	42%	4.340e+003
	PSO	4%	6.604e+004
Schwefel	GD	12%	1.017e+004
	PSO	100%	4.400e+003
Rastrigin	GD	0%	—
	PSO	22%	6.273e+003
Ackley	GD	0%	—
	PSO	36%	8.833e+003
Griewank	GD	0%	—
	PSO	96%	7.938e+003
Six Hump Camel Back	GD	44%	1.000e+003
	PSO	100%	1.020e+003

Function Modality

Categorising the benchmark functions can help explain the polarisation in results. Table 5.4 shows the list of the benchmark functions, whether the function is unimodal or multimodal, and which algorithm performed better in Experiment 1. The results suggest that the modality of the problem plays an important role in the performance of the algorithms on each of the benchmark functions, specifically suggesting that GD is superior on unimodal problems while PSO is superior on multimodal problems.

Hypotheses 5.2 and 5.3 in the following section are used to formally state and validate the above finding.

Table 5.4: Problem Characteristics and Algorithm Performance for Experiment 1

Problem	**Superior Algorithm**	**Problem Modality**
Spherical	GD	Unimodal
Rosenbrock	GD	Unimodal
Quartic	GD	Unimodal
Schwefel	PSO	Multimodal
Rastrigin	PSO	Multimodal
Ackley	PSO	Multimodal
Griewank	PSO	Multimodal
Six Hump Camel Back	PSO	Multimodal

5.1.3 Performance of GD and PSO versus Modality

The experimental results obtained to validate Hypothesis 5.1 strongly suggested a correlation between algorithm performance and the modality of the problem. In this section, these findings are formally validated.

Hypothesis 5.2 *Gradient-based algorithms perform better with respect to accuracy than standard PSO on unimodal problems used in this study.*

Using results from Table 5.1, it can be seen that gradient descent produced a lower mean error on the unimodal functions Spherical, Rosenbrock and Quartic. Table 5.2 shows that these results are significant at the 0.05 level. Consequently, it can be stated that Hypothesis 5.2 is valid.

Hypothesis 5.3 *Standard PSO performs better than gradient-based algorithms on multimodal problems.*

Using results from Table 5.1, it can be seen that PSO produced a lower mean error on the multimodal functions Schwefel, Rastrigin, Ackley, Griewank and Six Hump Camel Back. Table 5.2 shows that these results are significant at the 0.05 level. Consequently, it can be stated that Hypothesis 5.3 is valid.

5.1.4 Additional Performance Characteristics

Table 5.3 provides two additional measures of algorithm performance for Experiment 1, namely algorithm reliability and efficiency. These measurements were defined in sections 4.6.2 and 4.6.3 respectively.

The general finding that PSO outperforms GD on multimodal functions seems to hold here as well. On each multimodal function, PSO showed higher reliability than GD. In the case of the unimodal functions Spherical and Rosenbrock, reliability was tied at 100%. Comparing the efficiency measurements, GD was able to optimise the Spherical function far more efficiently than PSO, however the numbers were far closer on Rosenbrock.

The difference in the behaviour of GD on Spherical and Rosenbrock can be attributed to the nature of the respective functions. Section 3.3.2 discussed the topic of large gradients affecting gradient algorithms negatively. Consider Table 5.5, which shows some estimated statistics concerning the gradient of the 2-dimensional version of each of the benchmark functions. The data is the result of sampling the gradient at equally spaced positions in the domain of the function. It is immediately apparent that the Rosenbrock function exhibits some of the largest gradients of all the functions used. There is a also a large difference between the mean gradient and the maximum gradient, and the standard deviation is also the highest of all the functions shown. Gradient-based algorithms, such as GD, are typically configured such that they are able to deal with the largest gradients in the domain of the function, without completely leaving the search space. This means that when the algorithm finds itself in the banana-shaped valley of Rosenbrock - which typically exhibits very small gradients - the algorithm is ill-configured to efficiently find the global minimum, taking a larger number of iterations than would otherwise be necessary.

5.1.5 Summary

Hypothesis 5.1 showed that with respect to accuracy, the performance of GD is not equal to that of PSO for the benchmark functions used. Further analysis yielded that with respect to accuracy, GD performed better than PSO on unimodal functions (Hypothesis

Table 5.5: Gradient Characteristics for Benchmark Functions Where $N = 2$

Function		**Gradient**		
Name	**Domain**	**Maximum**	**Mean**	**Std. Dev.**
Spherical	[-5.12, 5.12]	10.240	5.125	2.959
Rosenbrock	[-2.048, 2.048]	5119.791	642.812	829.993
Quartic	[-1.28, 1.28]	4.194	0.789	0.979
Schwefel	[-512.03, 511.97]	11.097	5.031	3.130
Ackley	[-30, 30]	0.940	0.234	0.166
Rastrigin	[-5.12, 5.12]	71.331	39.547	19.670
Griewank	[-600, 600]	1.290	0.377	0.275
Six Hump Camel Back	[-3, 3], [-2, 2]	285.198	31.084	52.117

5.2), while PSO performed better than GD on multimodal functions (Hypothesis 5.3).

5.2 GDPSO Compared Against PSO

This section examines the performance of the first proposed gradient-based hybrid PSO, the Gradient Descent PSO (GDPSO). First, motivation for the GDPSO is given in section 5.2.1, followed by a comparison against standard PSO in section 5.2.2.

5.2.1 Motivation for a Gradient Hybrid PSO

Section 5.1 showed that there is a significant difference between the performance of GD and PSO. The gradient algorithm excels on unimodal problems, while PSO performed well on multimodal problems. In order to perform well on a unimodal problem, an algorithm needs to exhibit strong exploitation characteristics, whereas good performance on multimodal problems indicates strong exploration characteristics. It seems possible that a hybrid algorithm which combines both gradient-based methods as well as PSO would exhibit both strong exploitation and strong exploration capabilities, a desirable characteristic for an optimisation algorithm.

5.2.2 Performance of GDPSO and PSO

A hybrid gradient-based PSO, the GDPSO (Algorithm 3.1) is compared with the standard PSO. Hypotheses 5.4 and 5.5 test whether the hybrid algorithm has successfully incorporated the the gradient algorithm's performance on unimodal problems while retaining PSO's performance on multimodal.

Experiment 2 was created to validate Hypotheses 5.4 and 5.5, by testing GDPSO and standard PSO on the 8 benchmark functions. Tables 5.6 and 5.7 give the mean error, and Mann-Whitney U test p-values, respectively, for the experiment. Table 5.8 provides the reliability and efficiency measurements.

Hypothesis 5.4 *GDPSO performs better than standard PSO on unimodal problems, with respect to accuracy.*

Table 5.6 shows that for the unimodal functions Spherical, Rosenbrock and Quartic, GDPSO had a lower mean error than standard PSO. The Mann-Whitney U test p-values from Table 5.7 show that for these functions, there was a significant difference in the error found by the algorithms. It is therefore possible to say that GDPSO exhibited a significantly lower mean error than standard PSO for all the unimodal functions in the benchmark suite. Hypothesis 5.4 is therefore valid. The reliability and efficiency measurements from Table 5.8 further support the findings, with both reliability and efficiency of GDPSO being either lower or equal to that of PSO, for all the unimodal functions.

Hypothesis 5.5 *GDPSO performs no worse than standard PSO on multimodal problems.*

The Mann-Whitney U test p-values from Table 5.7 reveal that for the Six Hump Camel Back and Griewank functions, there was no significant difference between the performance of PSO and GDPSO. The remaining 3 multimodal functions did show a significant difference, and Table 5.6 shows that in each case, PSO had a lower mean error than GDPSO. This result indicates that GDPSO was not able to retain the performance of standard PSO for all multimodal functions; thus Hypothesis 5.5 is false. Table 5.8 once again provides further evidence, showing that GDPSO exhibited lower reliability and efficiency than PSO on several of the multimodal functions.

Table 5.6: Mean Error for Experiment 2

Problem	**Algorithm**	**Error**
Spherical	PSO	2.480e-028 ± 1.403e-027
	GDPSO	0.000e+000 ± 0.000e+000
Rosenbrock	PSO	5.739e+000 ± 9.887e+000
	GDPSO	5.582e-001 ± 1.397e+000
Quartic	PSO	5.668e-001 ± 1.816e-001
	GDPSO	4.571e-001 ± 1.526e-001
Schwefel	PSO	4.239e+003 ± 5.857e+002
	GDPSO	4.666e+003 ± 7.258e+002
Rastrigin	PSO	1.244e+002 ± 3.447e+001
	GDPSO	1.527e+002 ± 4.162e+001
Ackley	PSO	8.669e+000 ± 6.784e+000
	GDPSO	1.550e+001 ± 4.475e+000
Griewank	PSO	7.189e-002 ± 1.004e-001
	GDPSO	1.191e-001 ± 2.222e-001
Six Hump Camel Back	PSO	-2.845e-005 ± 0.000e+000
	GDPSO	-2.845e-005 ± 0.000e+000

Table 5.7: Mann-Whitney U Test P-Values for Experiment 2

Problem	**P-Value**
Spherical	2.20e-016
Rosenbrock	2.79e-013
Quartic	6.03e-002
Schwefel	9.71e-004
Rastrigin	2.32e-004
Ackley	2.07e-006
Griewank	1.37e-001
Six Hump Camel Back	1.00e+000

Table 5.8: Reliability and Efficiency for Experiment 2

Function	Algorithm	Reliability	Efficiency
Spherical	PSO	100%	4.970e+004
	GDPSO	100%	1.000e+003
Rosenbrock	PSO	100%	5.072e+003
	GDPSO	100%	1.340e+003
Quartic	PSO	20%	3.910e+004
	GDPSO	38%	2.110e+004
Schwefel	PSO	100%	4.736e+003
	GDPSO	94%	5.043e+003
Rastrigin	PSO	20%	6.480e+003
	GDPSO	14%	9.429e+003
Ackley	PSO	50%	7.488e+003
	GDPSO	2%	7.000e+003
Griewank	PSO	100%	7.872e+003
	GDPSO	98%	8.898e+003
Six Hump Camel Back	PSO	100%	1.020e+003
	GDPSO	100%	1.000e+003

5.2.3 Rate of Exploitation of PSO and GD

The gradient of a function represents the direction of the greatest change in fitness in the immediate vicinity of any given point. Gradient-based optimisers use this property to find the nearest minimum in the search space. Given a unimodal region and relatively uniform slopes, gradient-based algorithms can rapidly exploit their local areas.

Contrasting the rate of exploitation of gradient algorithms with non gradient algorithms on a unimodal surface can reveal some interesting characteristics and challenges. Consider Figure 5.1, which shows standard PSO compared to GD, with respect to accuracy. After only 100 iterations, the single GD algorithm achieves a function minimisation error dozens of orders of magnitude smaller than the PSO algorithm.

Gradient-based algorithms are well suited to optimise the Spherical function, however the above result highlights the potential difference in the rate of improvement of fitness between PSOs and gradient-based algorithms under ideal conditions.

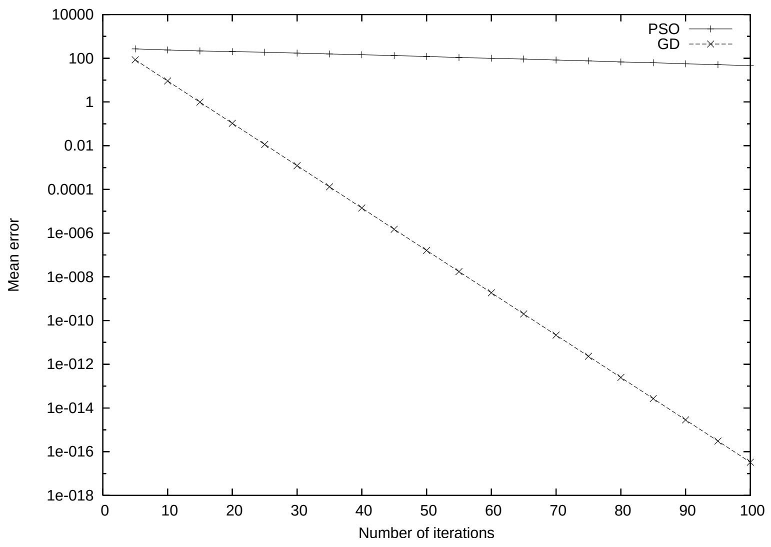

Figure 5.1: PSO and GD Performance on the Spherical Function

In order to predict the effect of the GD algorithm within GDPSO, the construction of the algorithm needs to be considered. Specifically, the GDPSO algorithm prescribes that the *gbest* (or *lbest*) particle in each iteration of the swarm will use the GD algorithm instead of the standard PSO velocity update.

Consider a multimodal function, with several local minima in the search space, where each local minimum is located in a relatively uniform and smooth region. In the case of a standard PSO, the global best particle would act as an attractor to the other particles in the swarm, causing them to explore the area in between their current position and that of the global best particle. During this exploration, the likelihood of the swarm uncovering areas with better solutions than the previous *gbest* particle's region is relatively high. GDPSO, however, changes the balance by giving the *gbest* particle an unfair advantage when compared with the rest of the swarm, in the form of the gradient descent algorithm. Considering the ability of GD to rapidly exploit a local region, it seems reasonable that the occurrence of situations where the rest of the swarm finds new areas that immediately outperform the GD-enabled *gbest* particle is rarer than with standard PSO. The end result of this new behaviour is that the swarm converges prematurely, severely compromising the swarm's ability to effectively explore the search space.

Figure 5.2 shows the performance of GDPSO and standard PSO on the Rastrigin function. The mean error for the first 200 swarm iterations from Experiment 2 is shown against the iteration number. It is important to keep in mind that for GDPSO has been configured identically to standard PSO Experiment 2; the only difference is the presence of the gradient descent algorithm. The performance of GDPSO in the initial 15 swarm iterations is superior to that of PSO. The rapid decrease in mean error is in line with the ability of GD to rapidly exploit its region. This rapid exploitation is also negatively affecting the exploration of the search space, which can be seen after iteration 25. The slope of the PSO mean error is steeper than that of the GDPSO. At 125 iterations, the algorithms have roughly equal mean error, and at 200 iterations, standard PSO has developed a distinct advantage over GDPSO. The results from Experiment 2 show that the trend carries on, and standard PSO exhibits performance that is superior to the GDPSO at the 0.05 level.

The above tendency of GDPSO to prematurely converge on multimodal problems is also detectable on the Schwefel, Ackley and Griewank functions.

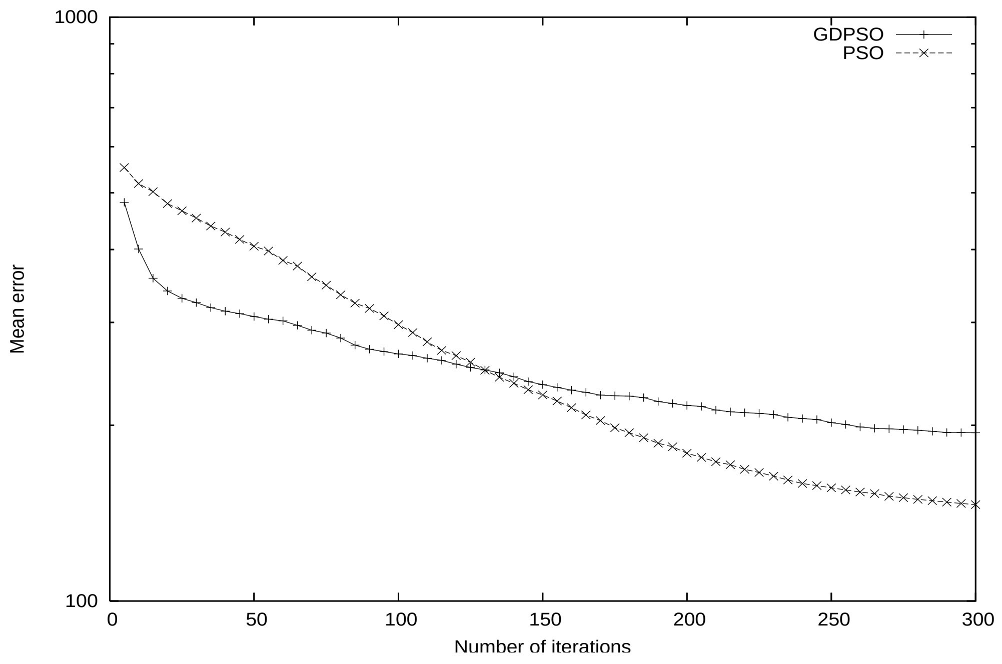

Figure 5.2: GDPSO and PSO Behaviour on the Rastrigin Function

5.2.4 Summary

Hypothesis 5.4 showed that by combining gradient descent and PSO, GDPSO was able to outperform standard PSO on unimodal functions. Performance on multimodal functions was not retained however, as shown by Hypothesis 5.5. Analysis of the rate of exploitation of PSO and GD indicated that the inclusion of GD in the PSO algorithm could lead to premature convergence.

5.3 Modifications to GDPSO

In this section, three possible modifications to the GDPSO algorithm are examined, aiming to improve on the algorithm in light of the findings in section 5.2.3. The effect of modifications to the social and cognitive acceleration coefficients in order to lower the tendency for premature convergence is examined in section 5.3.1, followed by an examination of how delaying the introduction of gradient information affects performance in section 5.3.2. Finally, the impact of the *lbest* topology is examined in section 5.3.3.

5.3.1 Modified Acceleration Coefficients

The PSO algorithm includes the social and cognitive acceleration coefficients, c_1 and c_2, which control the degree to which particles are attracted to their neighbourhood's best particle and their own personal best position, respectively. The acceleration coefficients provide a natural way to change the balance of forces within the swarm, which can be used to lessen the tendency for premature convergence.

Hypothesis 5.6 states that by adjusting the above coefficients, equality in performance (with respect to accuracy) to PSO on the multimodal functions can be achieved. In order to highlight the difference in the configuration of the algorithm, GDPSO with modified acceleration coefficients is referred to as GDPSO$_M$. The term *modified acceleration coefficients* refers to the adjustment of the acceleration coefficients c_1 and c_2 on a per-problem basis, as defined in section 4.4.2.

Hypothesis 5.6 *By adjusting social and cognitive acceleration coefficients, GDPSO$_M$ performs no worse than standard PSO on multimodal problems, with respect to accuracy.*

Experiment 3 aims to evaluate Hypothesis 5.6. A series of experimental runs were created in order to identify the best values for c_1 and c_2. Combinations of values such that $c_1, c_2 \in [0.1, 2.0]$ were tested. Table 5.9 lists the values obtained. In each case, the combination resulting in the lowest mean error was taken.

Tables 5.10 and 5.11 provide the mean error, and Mann-Whitney U test p-values respectively, for Experiment 3. Table 5.12 gives the reliability and efficiency measures. Standard PSO is shown to have lower error on the Ackley function at the 0.05 level, meaning that Hypothesis 5.6 is false. It should be noted that the format of Table 5.11 differs to that of Tables 5.2 and 5.7. In each of the latter cases, only two algorithms were being compared against each other, resulting in a single p-value per function. In the case of Experiment 3, there were 3 algorithms involved, resulting in 3 possible pairings, each having a separate p-value.

Further analysis of the results reveals the following:

- The results for GDPSO$_M$ on Schwefel are not significantly worse than standard PSO. Adjusting the acceleration coefficients has helped narrow the gap between PSO and GDPSO.
- On the Rastrigin function, GDPSO$_M$ actually has a lower mean error than standard PSO. However, the p-values show this difference not to be significant. As with Schwefel, the adjustment to the acceleration coefficients has helped.
- On the Ackley function, PSO still significantly outperformed both GDPSO and GDPSO$_M$. There was no significant difference between the performance of the latter two.

The above results indicate that adjusting the social and cognitive acceleration coefficients can help decrease the tendency of the swarm to prematurely converge on certain functions. The result on the Ackley function, however, means that 5.6 is not valid. This result is nevertheless useful, since it states that the modification of acceleration coefficients alone cannot solve the problem of premature convergence in GDPSO.

The efficiency and reliability results in Table 5.12 are less clear than the accuracy measurements given in Table 5.10. While the modified acceleration coefficients have resulted in greatly increased reliability and efficiency in the case of the Rastrigin function, as well as a moderate increase in reliability on the Schwefel function, reliability seems to have suffered on the Ackley function. The difference on the latter function can possibly be attributed to small variances between experimental runs, since the reliability level of 2% for GDPSO meant that a single sample out of 50 reached the required threshold for the reliability measure. It seems reasonable that given another experimental run, a slightly different result might emerge, as in this case.

5.3.2 Delayed Introduction of Gradient Information

Section 5.2.3 strongly suggested that the premature convergence exhibited by GDPSO was caused by rapid exploitation of inferior regions by the gradient descent algorithm. One possible approach to avoiding this effect is to prevent the usage of gradient information in the initial stages of the GDPSO algorithm.

The mechanism for managing the delayed introduction of gradient information was described in section 3.3.1.

Hypothesis 5.7 proposes that the delayed introduction of gradient information will lead to performance parity with PSO on multimodal functions.

Hypothesis 5.7 *By making use of delayed gradient information, GDPSO$_D$ performs no worse than standard PSO on multimodal problems, with respect to accuracy.*

Experiment 4 was designed to validate hypothesis 5.7 by testing the performance of GDPSO using delayed gradient information on the multimodal functions where GDPSO performed worse than standard PSO in Experiment 2.

It is important to note that Experiment 4 is only concerned with the multimodal functions Schwefel, Rastrigin and Ackley. The remaining multimodal functions Griewank and Six Hump Camel Back showed no significant difference in the performance of GDPSO and standard PSO. Since GDPSO$_M$ with $\rho_{delay} = 1$ and $\rho_{delay} = 0$ is logically equivalent to standard PSO and standard GDPSO respectively, it is unnecessary to re-test

Table 5.9: Acceleration Coefficients for Experiment 3

Problem	c_1	c_2
Schwefel	1.5	1.5
Rastrigin	1.5	1.75
Ackley	1.4	1.3

Table 5.10: Mean Error for Experiment 3

Problem	**Algorithm**	**Error**
Schwefel	GDPSO	4.666e+003 ± 7.258e+002
	PSO	4.239e+003 ± 5.857e+002
	GDPSO$_M$	4.543e+003 ± 5.813e+002
Rastrigin	GDPSO	1.527e+002 ± 4.162e+001
	PSO	1.244e+002 ± 3.447e+001
	GDPSO$_M$	1.107e+002 ± 2.933e+001
Ackley	GDPSO	1.550e+001 ± 4.475e+000
	PSO	8.669e+000 ± 6.784e+000
	GDPSO$_M$	1.640e+001 ± 4.548e+000

Table 5.11: Mann-Whitney U Test P-Values for Experiment 3

		GDPSO	PSO	GDPSO$_M$
Schwefel	GDPSO		9.71e-004	2.57e-001
	PSO	9.71e-004		1.37e-002
	GDPSO$_M$	2.57e-001	1.37e-002	
Rastrigin	GDPSO		2.32e-004	1.64e-007
	PSO	2.32e-004		7.25e-002
	GDPSO$_M$	1.64e-007	7.25e-002	
Ackley	GDPSO		2.07e-006	4.38e-002
	PSO	2.07e-006		1.89e-008
	GDPSO$_M$	4.38e-002	1.89e-008	

Table 5.12: Reliability and Efficiency for Experiment 3

Function	Algorithm	Reliability	Efficiency
Schwefel	GDPSO	94%	5.043e+003
	PSO	100%	4.736e+003
	GDPSO$_M$	100%	5.072e+003
Rastrigin	GDPSO	14%	9.429e+003
	PSO	20%	6.480e+003
	GDPSO$_M$	40%	3.338e+004
Ackley	GDPSO	2%	7.000e+003
	PSO	50%	7.488e+003
	GDPSO$_M$	0%	—

performance on the Griewank and Six Hump Camel Back functions for the purpose of Hypothesis 5.7.

Table 5.13 provides the mean error, and Table 5.14 the Mann-Whitney U test p-values respectively for Experiment 4. Table 5.15 provides the efficiency and reliability measures for the experiment.

Table 5.14 shows that there is no significant difference in the performance of GDPSO$_D$ and standard PSO, validating Hypothesis 5.6. Furthermore, a significant difference between the performance of GDPSO$_D$ and GDPSO was shown, in each case with a lower mean error for GDPSO$_D$.

The reliability and efficiency measures in Table 5.15 further support the findings above, with the reliability of GDPSO$_D$ being higher than that of GDPSO on all functions in the experiment. Perhaps more importantly, the reliability of GDPSO$_D$ is very close to the reliability of PSO, showing that the delayed introduction of gradient information has greatly helped the performance of the GDPSO algorithm on multimodal functions.

5.3.3 The *lbest* Topology

The results in the previous sections were obtained using the GDPSO algorithm with the *gbest* topology. It is therefore reasonable to explore the effects of the *lbest* topology on the GDPSO algorithm. It is important to bear in mind the exact implementation for the

Table 5.13: Mean Error for Experiment 4

Problem	Algorithm	Error
Schwefel	GDPSO	4.666e+003 ± 7.258e+002
	PSO	4.239e+003 ± 5.857e+002
	GDPSO_D	4.232e+003 ± 6.385e+002
Rastrigin	GDPSO	1.527e+002 ± 4.162e+001
	PSO	1.244e+002 ± 3.447e+001
	GDPSO_D	1.211e+002 ± 3.445e+001
Ackley	GDPSO	1.550e+001 ± 4.475e+000
	PSO	8.669e+000 ± 6.784e+000
	GDPSO_D	7.624e+000 ± 5.328e+000

lbest topology in terms of GDPSO algorithm 3.1, which states that each neighbourhood will have one gradient-based particle. This means that each particle in the swarm will be directly influenced by a gradient algorithm via the social component, however the delay of information between neighbourhoods may serve to counter premature convergence.

Experiment 5 was created to compare the *lbest* topology GDPSO against standard PSO. Table 5.16 gives the mean error for each function, and table 5.17 gives the Mann-Whitney U test p-values prior to correction. Table 5.18 provides the efficiency and reliability measurements.

The following hypothesis is proposed:

Hypothesis 5.8 *By making use of the delayed introduction of gradient information, GDPSO_{lbest} performs no worse than standard PSO on multimodal problems, with respect to accuracy.*

Table 5.17 shows that a significant difference exists for multimodal functions Schwefel, Rastrigin, Ackley and Griewank. Considering table 5.16, it can be seen that GDPSO_{lbest} performed better on Griewank, standard PSO was still superior for functions Schwefel, Rastrigin and Ackley. Hypothesis 5.8 is thus false.

The efficiency and reliability meausres in Table 5.18 further support the above finding. While the use of the *lbest* topology in the GDPSO_{lbest} algorithm did result in

Table 5.14: Mann-Whitney U Test P-Values for Experiment 4

		GDPSO	PSO	GDPSO$_D$
Schwefel	GDPSO		9.71e-004	9.71e-004
	PSO	9.71e-004		9.95e-001
	GDPSO$_D$	9.71e-004	9.95e-001	
Rastrigin	GDPSO		2.32e-004	1.46e-004
	PSO	2.32e-004		5.84e-001
	GDPSO$_D$	1.46e-004	5.84e-001	
Ackley	GDPSO		2.07e-006	3.31e-010
	PSO	2.07e-006		6.97e-001
	GDPSO$_D$	3.31e-010	6.97e-001	

increased reliability on the Schwefel, Ackley and Griewank functions, reliability as well as effciency was actually decreased on the Rastrigin function. Compared with standard PSO, GDPSO$_{lbest}$ still exhibited lower reliability on the multimodal functions Schwefel, Rastrigin and Ackley.

The failure of the *lbest* topology to yield significant improvements over the *gbest* in the GDPSO algorithm led to a decision to present only the results for the *gbest* topology for the rest of the chapter. This decision greatly simplifies and shortens the number of experiments in the chapter.

5.3.4 Summary

Hypothesis 5.6 showed that modifying the acceleration coefficients of GDPSO did not improve performance significantly enough to make GDPSO$_M$ on par with standard PSO on the multimodal functions. The delayed introduction of gradient information as used in the GDPSO$_D$ algorithm, worked well, and Hypothesis 5.7 showed that the modified GDPSO algorithm was on par with standard PSO on the multimodal functions. The usage of the *lbest* topology did not improve the GDPSO algorithm enough to bring it up to par with the standard PSO, as shown in hypothesis 5.8.

The significance of the above finding is that it is possible to create a hybrid gradient-PSO algorithm which displays the good characteristics of both of its parent algorithms,

Table 5.15: Reliability and Efficiency for Experiment 4

Function	Algorithm	Reliability	Efficiency
Schwefel	GDPSO	94%	5.043e+003
	PSO	100%	4.736e+003
	$GDPSO_D$	100%	4.376e+003
Rastrigin	GDPSO	14%	9.429e+003
	PSO	20%	6.480e+003
	$GDPSO_D$	26%	6.769e+003
Ackley	GDPSO	2%	7.000e+003
	PSO	50%	7.488e+003
	$GDPSO_D$	42%	8.095e+003

Table 5.16: Mean Error for Experiment 5

Problem	**Algorithm**	**Error**
Spherical	PSO	2.480e-028 ± 1.403e-027
	$GDPSO_{lbest}$	0.000e+000 ± 0.000e+000
Rosenbrock	PSO	5.739e+000 ± 9.887e+000
	$GDPSO_{lbest}$	4.836e-002 ± 9.100e-002
Quartic	PSO	5.668e-001 ± 1.816e-001
	$GDPSO_{lbest}$	4.500e-001 ± 1.252e-001
Schwefel	PSO	4.239e+003 ± 5.857e+002
	$GDPSO_{lbest}$	4.884e+003 ± 5.589e+002
Rastrigin	PSO	1.244e+002 ± 3.447e+001
	$GDPSO_{lbest}$	1.656e+002 ± 4.828e+001
Ackley	PSO	8.669e+000 ± 6.784e+000
	$GDPSO_{lbest}$	1.692e+001 ± 5.137e+000
Griewank	PSO	7.189e-002 ± 1.004e-001
	$GDPSO_{lbest}$	1.669e-002 ± 2.518e-002
	PSO	-2.845e-005 ± 0.000e+000
Six Hump Camel Back	$GDPSO_{lbest}$	-2.845e-005 ± 0.000e+000

Table 5.17: Mann-Whitney U Test P-Values for Experiment 5

		PSO	GDPSO$_{lbest}$
Spherical	PSO		2.20e-016
	GDPSO$_{lbest}$	2.20e-016	
Rosenbrock	PSO		2.94e-013
	GDPSO$_{lbest}$	2.94e-013	
Quartic	PSO		7.71e-003
	GDPSO$_{lbest}$	7.71e-003	
Schwefel	PSO		5.90e-007
	GDPSO$_{lbest}$	5.90e-007	
Rastrigin	PSO		6.42e-006
	GDPSO$_{lbest}$	6.42e-006	
Ackley	PSO		5.82e-009
	GDPSO$_{lbest}$	5.82e-009	
Griewank	PSO		3.46e-005
	GDPSO$_{lbest}$	3.46e-005	
Six Hump Camel Back	PSO		1.00e+000
	GDPSO$_{lbest}$	1.00e+000	

while suppressing the negatives.

5.4 Performance of LFPSO

The performance of a second proposed gradient-based hybrid PSO, the LeapFrog PSO (LFPSO), is examined. First, motivation for the LFPSO is given in section 5.4.1, followed by a comparison against standard PSO and GDPSO$_D$ in section 5.4.2. The effect of gradient information on the diversity of gradient-hybrid swarms is examined in section 5.4.3. The performance of modified LFPSO is investigated in section 5.4.4, and a summary is given in section 5.4.5.

Table 5.18: Reliability and Efficiency for Experiment 5

Function	Algorithm	Reliability	Efficiency
Spherical	PSO	100%	4.970e+004
	GDPSO	100%	1.000e+003
	GDPSO$_{lbest}$	100%	1.000e+003
Rosenbrock	PSO	100%	5.072e+003
	GDPSO	100%	1.340e+003
	GDPSO$_{lbest}$	100%	1.000e+003
Quartic	PSO	20%	3.910e+004
	GDPSO	38%	2.110e+004
	GDPSO$_{lbest}$	38%	1.734e+004
Schwefel	PSO	100%	4.736e+003
	GDPSO	94%	5.043e+003
	GDPSO$_{lbest}$	98%	2.324e+004
Rastrigin	PSO	20%	6.480e+003
	GDPSO	14%	9.429e+003
	GDPSO$_{lbest}$	6%	8.567e+004
Ackley	PSO	50%	7.488e+003
	GDPSO	2%	7.000e+003
	GDPSO$_{lbest}$	10%	1.242e+005
Griewank	PSO	100%	7.872e+003
	GDPSO	98%	8.898e+003
	GDPSO$_{lbest}$	100%	1.400e+004
Six Hump Camel Back	PSO	100%	1.020e+003
	GDPSO	100%	1.000e+003
	GDPSO$_{lbest}$	100%	1.000e+003

5.4.1 Motivation for the LFPSO

It has been shown in the previous sections, that gradient-based algorithms such as GD have good characteristics that can be successfully combined into a hybrid PSO algorithm such as the GDPSO. The work in sections 5.1.1 - 5.4.2 built upon the gradient descent algorithm, since it is the simplest of gradient-based algorithms. The LeapFrog algorithm is generally considered to be a more sophisticated gradient-based algorithm, and it is possible that its use in a hybrid PSO might result in even better performance.

5.4.2 Performance of Unmodified LFPSO

In order to establish a baseline for the performance characteristics of LFPSO, the following hypothesis is proposed:

Hypothesis 5.9 *LFPSO performs no worse than GDPSO$_D$ with respect to accuracy.*

To validate hypothesis 5.9, Experiment 6 was created. Here the standard LFPSO algorithm is compared to the GDPSO$_D$ from Experiment 4, across all 8 benchmark functions described in section 4.1. The LFOP1(b) version of the LeapFrog algorithm is used for all LFPSO implementations. The GDPSO$_D$ algorithm was selected for comparison, because out of the 3 modifications to GDPSO, with respect to accuracy, the delayed introduction of gradient information provided the best results.

Table 5.19 shows the mean error obtained for each algorithm and problem. Table 5.20 gives the Mann-Whitney U test p-values prior to correction, indicating whether there is a significant difference between the distributions of the error measurement of the respective algorithms. Table 5.21 shows the reliability measurements for the experiment.

The p-values from Table 5.20 show a significant difference only for the functions Rosenbrock and Quartic. In both of these, LFPSO showed a lower mean error, thus outperforming GDPSO$_M$. The other 6 benchmark functions showed no significant difference between the algorithms, proving hypothesis 5.9.

5.4.3 Gradient Information and Diversity

Section 5.2.2 investigated the performance of GDPSO in relation to standard PSO, and found that the inclusion of gradient information negatively influenced the ability of the algorithm to find good results on certain problems. Section 5.2.3 investigated the possible causes for this, and found that for in some cases, the gradient algorithms were able to

Table 5.19: Mean Error for Experiment 6

Problem	Algorithm	Error
Spherical	GDPSO$_D$	0.000e+000 ± 0.000e+000
	LFPSO	0.000e+000 ± 0.000e+000
Rosenbrock	GDPSO$_D$	1.757e+001 ± 2.797e+000
	LFPSO	9.384e-001 ± 3.044e+000
Quartic	GDPSO$_D$	5.854e-001 ± 1.540e-001
	LFPSO	4.485e-001 ± 1.488e-001
Schwefel	GDPSO$_D$	4.232e+003 ± 6.385e+002
	LFPSO	4.421e+003 ± 6.816e+002
Rastrigin	GDPSO$_D$	1.211e+002 ± 3.445e+001
	LFPSO	1.145e+002 ± 4.625e+001
Ackley	GDPSO$_D$	7.624e+000 ± 5.328e+000
	LFPSO	1.065e+001 ± 9.557e+000
Griewank	GDPSO$_D$	1.179e-001 ± 1.250e-001
	LFPSO	9.351e-002 ± 1.197e-001
Six Hump Camel Back	GDPSO$_D$	-2.845e-005 ± 0.000e+000
	LFPSO	-2.845e-005 ± 0.000e+000

rapidly exploit their local region, resulting in a situation where the areas around the initial *gbest* position of the search space was given greater focus early on. This pattern of behaviour tended to yield better results early on for the GDPSO, but eventually resulted in the standard PSO achieving lower error over the duration of the experiment.

The above observation raises the question of in what way (if any) the inclusion of gradient information affects swarm diversity. With the LFPSO now also introduced, this question can be examined in more detail. Figure 5.3 shows the change in diversity of standard PSO, GDPSO, and LFPSO swarms as the experimental iterations progressed, for the Spherical, Rosenbrock, Quartic and Schwefel functions, while Figure 5.4 shows the same information for functions Rastrigin, Ackley, Griwank, and Six Hump Camel Back.

In the context of the heterogeneous construction of the GDPSO and LFPSO, the

Table 5.20: Mann-Whitney U Test P-Values for Experiment 6

		$GDPSO_D$	LFPSO
Spherical	$GDPSO_D$		1.00e+000
	LFPSO	1.00e+000	
Rosenbrock	$GDPSO_D$		2.20e-016
	LFPSO	2.20e-016	
Quartic	$GDPSO_D$		1.47e-003
	LFPSO	1.47e-003	
Schwefel	$GDPSO_D$		1.43e-001
	LFPSO	1.43e-001	
Rastrigin	$GDPSO_D$		3.83e-001
	LFPSO	3.83e-001	
Ackley	$GDPSO_D$		5.93e-001
	LFPSO	5.93e-001	
Griewank	$GDPSO_D$		3.00e-001
	LFPSO	3.00e-001	
Six Hump Camel Back	$GDPSO_D$		1.00e+000
	LFPSO	1.00e+000	

following observations can be made from the figures:

- Overall, there is no strong tendency for gradient information to decrease diversity. The diversity of the swarms was very closely matched for all functions except Rosenbrock and Ackley. This finding is reasonable, considering that in the *gbest* swarms, gradient information is only applied directly to one particle in the swarm, and the convergence behaviour itself for the rest of the particles is in no way modified.

- The Rosenbrock function provided some deviations from the general trend, with the gradient optimisers having slightly lower diversity initially, and later on exhibiting higher diversity than the standard PSO. Unlike the standard swarm particles, the gradient-driven *gbest* particles are not influenced by constriction of the swarm, and

Table 5.21: Reliability and Efficiency for Experiment 6

Function	Algorithm	Reliability	Efficiency
Spherical	GDPSO$_D$	100%	4.000e+002
	LFPSO	100%	2.080e+003
Rosenbrock	GDPSO$_D$	100%	1.120e+003
	LFPSO	100%	1.160e+003
Quartic	GDPSO$_D$	12%	1.289e+004
	LFPSO	40%	1.624e+004
Schwefel	GDPSO$_D$	100%	4.376e+003
	LFPSO	100%	3.740e+003
Rastrigin	GDPSO$_D$	26%	6.769e+003
	LFPSO	40%	5.105e+004
Ackley	GDPSO$_D$	42%	8.095e+003
	LFPSO	44%	7.677e+004
Griewank	GDPSO$_D$	100%	8.160e+003
	LFPSO	100%	8.040e+003
Six Hump Camel Back	GDPSO$_D$	100%	1.000e+003
	LFPSO	100%	1.000e+003

may perform relatively large steps from time to time, resulting in an increase in the diversity of the swarm. The tendency of gradient based algorithms to oscillate from one position to the other on the Rosenbrock function was noted by Snyman [52]. The initial decrease in diversity can potentially be explained by the fact that Rosenbrock is a unimodal function, although a similar tendency was not observed on the other unimodal functions, Spherical and Quartic.

- The Ackley function also provided some deviations from the general trend, with the LFPSO exhibiting higher diversity than the GDPSO and standard PSO. Neither of the gradient-based hybrids exhibited lower diversity than the standard PSO, however.

- The plot for the Six Hump Camel Back reveals a greater tendency for variation

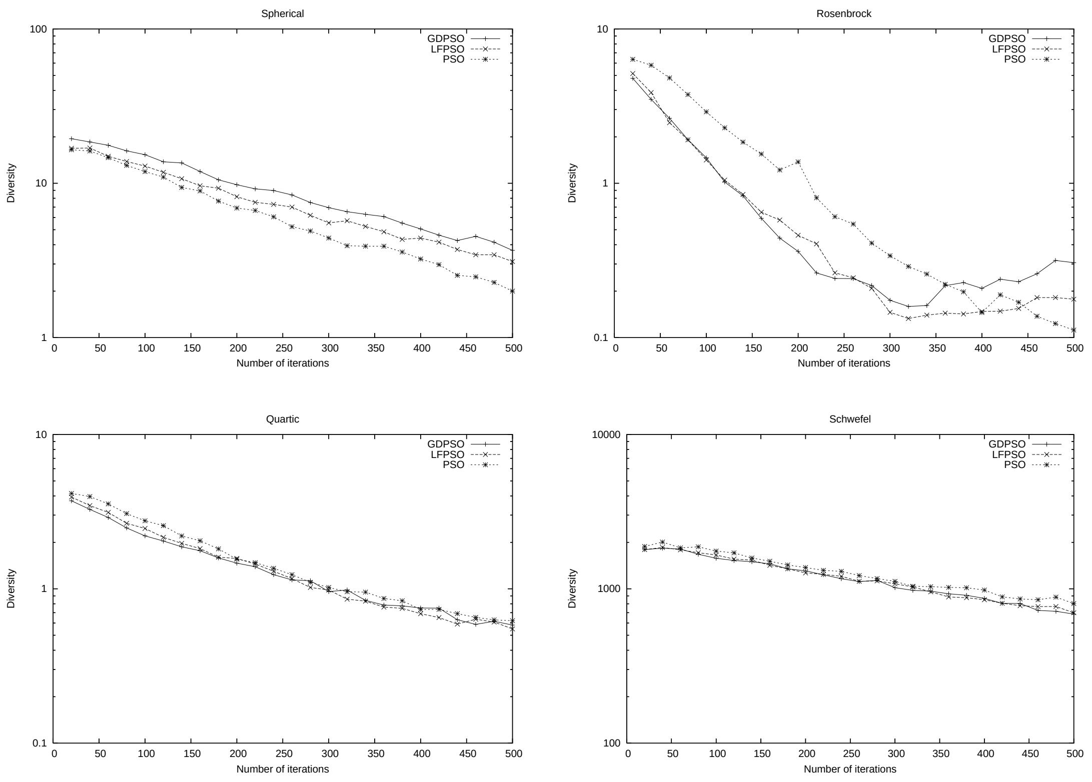

Figure 5.3: Diversity for functions f_1 to f_4

than for the rest of the functions. The overall trend for all three algorithms is nevertheless a gradually decreasing diversity with successive iterations. The higher variation in the diversity measurement can be attributed to the fact that the Six Hump Camel Back is a two-dimensional function, and as a result, the diversity measure has not been averaged out as much as in the case of the rest of the benchmark functions, each of which is 30-dimensional.

In summary, the diversity of the gradient-based GDPSO and LFPSO swarms was not negatively influenced by gradient information. This finding implies that the negative influence of gradient information on multimodal functions (as shown and discussed in section 5.2.2), can not be attributed solely to a lack of diversity.

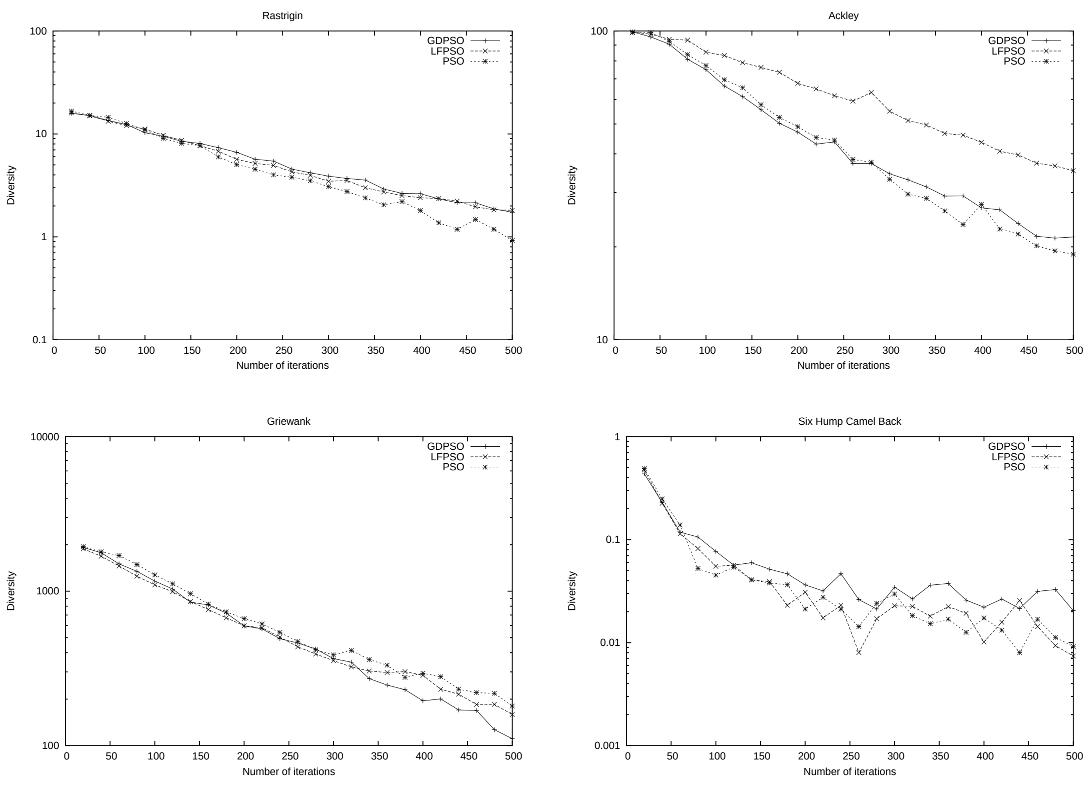

Figure 5.4: Diversity for functions f_5 to f_8

5.4.4 Performance of Modified LFPSO

Sections 5.3.1 and 5.3.2 investigated the effects of modified acceleration coefficients, as well as the delayed introduction of gradient information in GDPSO, with positive effects. In this section, similar modifications to the LFPSO are investigated.

Experiment 7 was designed to assess the relative difference between standard LFPSO and a variant with modified acceleration coefficients and delayed introduction of gradient information. The latter variant is labelled LFPSO$_{DM}$. The experiment also compares the performance of the above LFPSO algorithms against a GDPSO variant with similar modifications as LFPSO$_{DM}$, namely modified acceleration coefficients as well as delayed introduction of gradient information. This GDPSO variant is labelled GDPSO$_{DM}$, and differs from both the GDPSO$_D$ and GDPSO$_M$ algorithms in that both modifications have been enabled for the same algorithm.

Tables 5.22 and 5.23 show the mean error obtained for each algorithm and problem, and Mann-Whitney U test p-values prior to correction, respectively. Table 5.24 provides the reliability and efficiency measurements.

The following hypothesis is proposed:

Hypothesis 5.10 *LFPSO$_{DM}$ performs no worse than LFPSO with respect to accuracy. There exist problems where LFPSO$_{DM}$ exhibits superior performance to LFPSO.*

Table 5.23 shows the following regarding the relative performance of LFPSO and LFPSO$_{DM}$:

- No significant difference can be found for functions Spherical, Quartic, Schwefel, Rastrigin and Six Hump Camel Back.
- Functions Rosenbrock, Ackley and Griewank do show a significance difference.

Table 5.22 shows that for the above functions which did show a significant difference, in each case LFPSO$_{DM}$ exhibited a lower mean error. This proves hypothesis 5.10.

The reliability and efficiency measurements from Table 5.24 provide mixed results. In the case of the Ackley function, the LFPSO$_{DM}$ exhibits both better reliability and efficiency when compared to the LFPSO, which no significant differences can be seen for the Rosenbrock function. The Quartic function is an interesting case, since as shown by Table 5.22 and Table 5.23, the LFPSO$_{DM}$ exhibited a lower mean error than the LFPSO, while the Mann-Whitney U test p-values showed that this result was significant. Contrary to this, Table 5.24 results show a slight decrease in reliability and efficiency for the LFPSO$_{DM}$. The difference in the reliability and efficiency figures is small enough, however, to be explained as a difference in the experimental sampling of the measurements, especially given the stochastic nature of the function.

In order to explore the performance of LFPSO$_{DM}$ and GDPSO$_{DM}$, the following hypothesis is proposed:

Hypothesis 5.11 *LFPSO$_{DM}$ performs no worse than GDPSO$_{DM}$ with respect to accuracy. There exist problems where LFPSO$_{DM}$ exhibits superior performance to GDPSO$_{DM}$.*

Table 5.22: Mean Error for Experiment 7

Problem	**Algorithm**	**Error**
Spherical	GDPSO$_{DM}$	0.000e+000 ± 0.000e+000
	LFPSO	0.000e+000 ± 0.000e+000
	LFPSO$_{DM}$	0.000e+000 ± 0.000e+000
Rosenbrock	GDPSO$_{DM}$	1.196e+000 ± 1.845e+000
	LFPSO	9.384e-001 ± 3.044e+000
	LFPSO$_{DM}$	8.771e-001 ± 1.668e+000
Quartic	GDPSO$_{DM}$	4.532e-001 ± 1.401e-001
	LFPSO	4.485e-001 ± 1.488e-001
	LFPSO$_{DM}$	4.557e-001 ± 1.432e-001
Schwefel	GDPSO$_{DM}$	4.394e+003 ± 6.324e+002
	LFPSO	4.421e+003 ± 6.816e+002
	LFPSO$_{DM}$	4.226e+003 ± 5.085e+002
Rastrigin	GDPSO$_{DM}$	8.674e+001 ± 2.136e+001
	LFPSO	1.145e+002 ± 4.625e+001
	LFPSO$_{DM}$	1.175e+002 ± 5.177e+001
Ackley	GDPSO$_{DM}$	1.034e+001 ± 5.489e+000
	LFPSO	1.065e+001 ± 9.557e+000
	LFPSO$_{DM}$	2.260e+000 ± 6.190e+000
Griewank	GDPSO$_{DM}$	6.968e+002 ± 1.925e+002
	LFPSO	9.351e-002 ± 1.197e-001
	LFPSO$_{DM}$	1.182e-003 ± 7.072e-003
Six Hump Camel Back	GDPSO$_{DM}$	-2.845e-005 ± 0.000e+000
	LFPSO	-2.845e-005 ± 0.000e+000
	LFPSO$_{DM}$	-2.845e-005 ± 0.000e+000

Table 5.23 shows the results regarding the relative performance of GDPSO$_{DM}$ and LFPSO$_{DM}$:

- No significant difference can be found for functions Spherical, Quartic, Schwefel and Six Hump Camel Back.

Table 5.23: Mann-Whitney U Test P-Values for Experiment 7

		GDPSO$_{DM}$	LFPSO	LFPSO$_{DM}$
Spherical	GDPSO$_{DM}$		1.00e+000	1.00e+000
	LFPSO	1.00e+000		1.00e+000
	LFPSO$_{DM}$	1.00e+000	1.00e+000	
Rosenbrock	GDPSO$_{DM}$		3.19e-001	5.60e-004
	LFPSO	3.19e-001		6.86e-004
	LFPSO$_{DM}$	5.60e-004	6.86e-004	
Quartic	GDPSO$_{DM}$		9.04e-001	7.70e-001
	LFPSO	9.04e-001		9.64e-001
	LFPSO$_{DM}$	7.70e-001	9.64e-001	
Schwefel	GDPSO$_{DM}$		6.82e-001	2.51e-001
	LFPSO	6.82e-001		1.22e-001
	LFPSO$_{DM}$	2.51e-001	1.22e-001	
Rastrigin	GDPSO$_{DM}$		1.50e-003	1.33e-003
	LFPSO	1.50e-003		6.47e-001
	LFPSO$_{DM}$	1.33e-003	6.47e-001	
Ackley	GDPSO$_{DM}$		9.09e-001	9.46e-012
	LFPSO	9.09e-001		3.08e-007
	LFPSO$_{DM}$	9.46e-012	3.08e-007	
Griewank	GDPSO$_{DM}$		2.20e-016	2.20e-016
	LFPSO	2.20e-016		2.22e-016
	LFPSO$_{DM}$	2.20e-016	2.22e-016	
Six Hump Camel Back	GDPSO$_{DM}$		1.00e+000	1.00e+000
	LFPSO	1.00e+000		1.00e+000
	LFPSO$_{DM}$	1.00e+000	1.00e+000	

- Functions Rosenbrock, Rastrigin, Ackley and Griewank do show a significance difference.

Table 5.22 shows that in the case of functions Rosenbrock, Ackley and Griewank, LFPSO$_{DM}$ has a lower mean error than GDPSO$_{DM}$. The situation was reversed however for Rastrigin, with GDPSO$_{DM}$ showing a lower mean error. This result may be unexpected, however nevertheless shows hypothesis 5.11 to be false.

Table 5.24: Reliability and Efficiency for Experiment 7

Function	Algorithm	Reliability	Efficiency
Spherical	GDPSO_{DM}	100%	1.000e+003
	LFPSO	100%	2.080e+003
	LFPSO_{DM}	100%	2.100e+003
Rosenbrock	GDPSO_{DM}	100%	4.880e+003
	LFPSO	100%	1.160e+003
	LFPSO_{DM}	100%	1.300e+003
Quartic	GDPSO_{DM}	36%	1.864e+004
	LFPSO	40%	1.624e+004
	LFPSO_{DM}	34%	2.388e+004
Schwefel	GDPSO_{DM}	98%	5.776e+003
	LFPSO	100%	3.740e+003
	LFPSO_{DM}	100%	4.580e+003
Rastrigin	GDPSO_{DM}	76%	2.187e+004
	LFPSO	40%	5.105e+004
	LFPSO_{DM}	34%	7.306e+004
Ackley	GDPSO_{DM}	12%	5.667e+003
	LFPSO	44%	7.677e+004
	LFPSO_{DM}	88%	3.543e+004
Griewank	GDPSO_{DM}	0%	—
	LFPSO	100%	8.040e+003
	LFPSO_{DM}	100%	5.398e+004
Six Hump Camel Back	GDPSO_{DM}	100%	1.000e+003
	LFPSO	100%	1.000e+003
	LFPSO_{DM}	100%	1.000e+003

5.4.5 Summary

Hypothesis 5.9 showed that an unmodified LFPSO was able to perform as well as a GDPSO_M with optimised acceleration coefficients. Tables 5.19 and 5.20 also showed that in some cases, the LFPSO was able to significantly outperform the GDPSO$_M$ algorithm. This result seems to suggest that PSOs making use of LeapFrog algorithm are less prone to premature convergence than ones which make use of GD.

By modifying the acceleration coefficients and delaying the introduction of gradient information in the LFPSO$_{DM}$ algorithm, further performance gains were obtained. This result is formalised in hypothesis 5.10. A combination of the same two modifications on the GDPSO algorithm yielded the GDPSO$_{DM}$ algorithm, which was compared against LFPSO$_{DM}$. Although significantly outperformed by LFPSO$_{DM}$ on Rosenbrock, Ackley and Griewank, the GDPSO$_{DM}$ algorithm performed surprisingly well on the Rastrigin function, making it impossible to confirm hypothesis 5.11. The implication of the above finding is that neither LFPSO$_{DM}$ or GDPSO$_{DM}$ can be said to be of superior performance for the benchmark suite used.

5.5 Gradient PSOs Compared To Other PSO Variants

This section compares the gradient algorithms GDPSO$_{DM}$ and LFPSO$_{DM}$ to other PSO variants. Section 5.5.1 gives motivation for the comparisons. The GDPSO$_{DM}$ and LFPSO$_{DM}$ are compared with the HGPSO in section 5.5.2. This is followed by a comparison of the above gradient-based algorithms with GCPSO in section 5.5.3. GDPSO$_{DM}$ and LFPSO$_{DM}$ are compared to the Mutating PSO in section 5.5.4. Finally, a summary of the findings is given in section 5.5.5.

5.5.1 Motivation

The differences between PSOs and gradient-based algorithms have been explored in the preceding chapters, and new hybrid algorithms, the LFPSO and GDPSO, have been benchmarked against each other and the standard PSO algorithm. This was done in order to systematically and critically argue the positive and negative aspects of including gradient information in PSO. Since the introduction of the original PSO [14], a significant portion of the research around Particle Swarm Optimisers have been to introduce new and novel ways to improve their performance [22, 30, 33, 40, 45, 56, 9, 60]. The result of this research is that the original algorithm's performance is no longer the best frame of reference for new algorithms.

Three algorithms, the GCPSO [9], Mutating PSO [16] and HGPSO [40] have been selected for comparison against the GDPSO and LFPSO. The GCPSO was selected due to the significant amount of research behind it and the good performance it exhibits [45, 9]. The Mutating PSO was selected due to similarly good performance [16], but also due to the nature of the modifications it brings to standard PSO. Specifically, Mutating PSO adds a stochastic element of mutation to the PSO algorithm, which serves as an interesting contrast to the exploitation-oriented gradient-based PSOs. The HGPSO was selected simply because it is one of the few gradient-based PSO implementations available.

5.5.2 GDPSO$_{DM}$, LFPSO$_{DM}$ and HGPSO

In this section, the gradient PSO variants GDPSO$_{DM}$ and LFPSO$_{DM}$ are compared to the HGPSO. Experiment 8 was designed to contrast the performance of the above algorithms. Table 5.25 shows the mean error, while table 5.26 gives the Mann-Whitney U test p-values prior to correction. Table 5.27 provides reliability and efficiency figures.

In order to explore the relative performance of LFPSO$_{DM}$ and HGPSO, the following hypothesis is proposed:

Hypothesis 5.12 *LFPSO$_{DM}$ performs no worse than HGPSO with respect to accuracy. There exist problems where LFPSO$_{DM}$ exhibits superior performance to HGPSO.*

Table 5.26 shows that no significant difference in mean error exists for the problems Spherical, Rosenbrock and Six Hump Camel Back. It is worthwhile noting that the results for the Rosenbrock function in table 5.25 seem to show a rather large difference in mean error, however this seems to be caused by outliers in the results. Table 5.27 is provided as supporting evidence for the above observation, specifically, the reliability measure indicates that roughly 6% of the HGPSO solutions were not within the threshold for the Rosenbrock function. The relatively high p-value, however, indicates that the rest of the solutions were closely matched with the solutions from LFPSO$_{DM}$.

Considering the functions where a significant difference was indicated, namely Quartic, Schwefel, Rastrigin, Ackley and Griewank, table 5.25 clearly shows that the LFPSO$_{DM}$ algorithm achieved lower mean error. Hypothesis 5.12 is therefore valid.

Table 5.27 further supports the above finding, clearly showing that the LFPSO$_{DM}$ exhibited better reliability than the HGPSO on the functions with a significant difference in error.

The following hypothesis is proposed in order to evaluate the performance of GDPSO$_{DM}$ and HGPSO:

Hypothesis 5.13 *GDPSO$_{DM}$ performs no worse than HGPSO with respect to accuracy. There exist problems where GDPSO$_{DM}$ exhibits superior performance to HGPSO.*

The p-values from table 5.26 indicate that only the functions Quartic, Schwefel, Rastrigin, Ackley and Griewank showed a significant difference. Table 5.25 shows that while GDPSO$_{DM}$ exhibited lower mean error on Quartic, Schwefel, Rastrigin and Ackley, HGPSO exhibited significantly lower mean error on Griewank. One possible explanation for this result is that Griewank has been shown to behave more like a unimodal function at higher dimensions [62], where the HGPSO algorithm might have an advantage. Nevertheless, this result clearly invalidates hypothesis 5.13.

The reliability results from Table 5.27 also support the above finding. The performance of the GDPSO$_{DM}$ algorithm on the Griewank function was such that none of the experimental runs found a solution of good enough quality to contribute to the reliability measurement, leading to a reliability figure of 0%. In the case of the remaining functions which exhibited significant differences in error, namely Quartic, Schwefel, Rastrigin and Ackley, GDPSO$_{DM}$ was found to have superior reliability to the HGPSO.

5.5.3 GDPSO$_{DM}$, LFPSO$_{DM}$ and GCPSO

The gradient PSO variants GDPSO$_{DM}$ and LFPSO$_{DM}$ are compared to the GCPSO. Experiment 9 was designed to contrast the performance of the above algorithms. Table 5.28 shows the mean error, while table 5.29 gives the Mann-Whitney U test p-values prior to correction. Table 5.30 gives the reliability and efficiency measurements.

Table 5.25: Mean Error for Experiment 8

Problem	Algorithm	Error
Spherical	GDPSO$_{DM}$	0.000e+000 ± 0.000e+000
	LFPSO$_{DM}$	0.000e+000 ± 0.000e+000
	HGPSO	0.000e+000 ± 0.000e+000
Rosenbrock	GDPSO$_{DM}$	1.196e+000 ± 1.845e+000
	LFPSO$_{DM}$	8.771e-001 ± 1.668e+000
	HGPSO	4.028e+002 ± 1.826e+003
Quartic	GDPSO$_{DM}$	4.532e-001 ± 1.401e-001
	LFPSO$_{DM}$	4.557e-001 ± 1.432e-001
	HGPSO	7.711e-001 ± 1.434e-001
Schwefel	GDPSO$_{DM}$	4.394e+003 ± 6.324e+002
	LFPSO$_{DM}$	4.226e+003 ± 5.085e+002
	HGPSO	5.227e+003 ± 5.754e+002
Rastrigin	GDPSO$_{DM}$	8.674e+001 ± 2.136e+001
	LFPSO$_{DM}$	1.175e+002 ± 5.177e+001
	HGPSO	2.660e+002 ± 6.395e+001
Ackley	GDPSO$_{DM}$	1.034e+001 ± 5.489e+000
	LFPSO$_{DM}$	2.260e+000 ± 6.190e+000
	HGPSO	1.842e+001 ± 2.199e+000
Griewank	GDPSO$_{DM}$	6.968e+002 ± 1.925e+002
	LFPSO$_{DM}$	1.182e-003 ± 7.072e-003
	HGPSO	8.444e-001 ± 1.924e+000
Six Hump Camel Back	GDPSO$_{DM}$	-2.845e-005 ± 0.000e+000
	LFPSO$_{DM}$	-2.845e-005 ± 0.000e+000
	HGPSO	-2.845e-005 ± 3.140e-017

The following hypothesis is proposed in order to evaluate the performance of LFPSO$_{DM}$ and GCPSO:

Hypothesis 5.14 *LFPSO$_{DM}$ performs no worse than GCPSO with respect to accuracy. There exist problems where LFPSO$_{DM}$ exhibits superior performance to GCPSO.*

Table 5.26: Mann-Whitney U Test P-Values for Experiment 8

		GDPSO$_{DM}$	LFPSO$_{DM}$	HGPSO
Spherical	GDPSO$_{DM}$		1.00e+000	1.00e+000
	LFPSO$_{DM}$	1.00e+000		1.00e+000
	HGPSO	1.00e+000	1.00e+000	
Rosenbrock	GDPSO$_{DM}$		5.60e-004	2.48e-002
	LFPSO$_{DM}$	5.60e-004		6.57e-001
	HGPSO	2.48e-002	6.57e-001	
Quartic	GDPSO$_{DM}$		7.70e-001	8.05e-014
	LFPSO$_{DM}$	7.70e-001		3.80e-013
	HGPSO	8.05e-014	3.80e-013	
Schwefel	GDPSO$_{DM}$		2.51e-001	4.73e-009
	LFPSO$_{DM}$	2.51e-001		3.59e-012
	HGPSO	4.73e-009	3.59e-012	
Rastrigin	GDPSO$_{DM}$		1.33e-003	2.20e-016
	LFPSO$_{DM}$	1.33e-003		6.69e-016
	HGPSO	2.20e-016	6.69e-016	
Ackley	GDPSO$_{DM}$		9.46e-012	2.89e-010
	LFPSO$_{DM}$	9.46e-012		2.38e-014
	HGPSO	2.89e-010	2.38e-014	
Griewank	GDPSO$_{DM}$		2.20e-016	2.20e-016
	LFPSO$_{DM}$	2.20e-016		2.20e-016
	HGPSO	2.20e-016	2.20e-016	
Six Hump Camel Back	GDPSO$_{DM}$		1.00e+000	3.27e-001
	LFPSO$_{DM}$	1.00e+000		3.27e-001
	HGPSO	3.27e-001	3.27e-001	

The p-values from table 5.29 indicate that no significant difference between the algorithms could be found for functions Quartic, Schwefel and Six Hump Camel Back. Table 5.28 shows that while LFPSO$_{DM}$ exhibited lower mean error on Spherical, Rosenbrock and Griewank, GCPSO exhibited significantly lower mean error on functions Rastrigin and Ackley. This clearly invalidates hypothesis 5.14.

The reliability and efficiency figures from Table 5.30 highlight similar trends as above. The LFPSO$_{DM}$ algorithm provided 100% reliability on the Spherical function, while the GCPSO did not manage to reach the threshold for the measurement. On the Rosenbrock

Table 5.27: Reliability and Efficiency for Experiment 8

Function	Algorithm	Reliability	Efficiency
Spherical	GDPSO$_{DM}$	100%	1.000e+003
	LFPSO$_{DM}$	100%	2.100e+003
	HGPSO	100%	9.000e+003
Rosenbrock	GDPSO$_{DM}$	100%	4.880e+003
	LFPSO$_{DM}$	100%	1.300e+003
	HGPSO	94%	1.000e+003
Quartic	GDPSO$_{DM}$	36%	1.864e+004
	LFPSO$_{DM}$	34%	2.388e+004
	HGPSO	0%	1.401e+005
Schwefel	GDPSO$_{DM}$	98%	5.776e+003
	LFPSO$_{DM}$	100%	4.580e+003
	HGPSO	94%	3.957e+003
Rastrigin	GDPSO$_{DM}$	76%	2.187e+004
	LFPSO$_{DM}$	34%	7.306e+004
	HGPSO	0%	—
Ackley	GDPSO$_{DM}$	12%	5.667e+003
	LFPSO$_{DM}$	88%	3.543e+004
	HGPSO	0%	—
Griewank	GDPSO$_{DM}$	0%	—
	LFPSO$_{DM}$	100%	5.398e+004
	HGPSO	86%	1.721e+004
Six Hump Camel Back	GDPSO$_{DM}$	100%	1.000e+003
	LFPSO$_{DM}$	100%	1.000e+003
	HGPSO	100%	1.020e+003

and Griewank functions, both algorithms were tied at 100% for reliability, with the LFPSO$_{DM}$ proving more efficient on the Rosenbrock function, and the GCPSO on the Griewank. On both of the functions where the GCPSO exhibited a significantly lower mean error, namely Rastrigin and Ackley, the GCPSO also showed greater reliability

Table 5.28: Mean Error for Experiment 9

Problem	Algorithm	Error
Spherical	GDPSO$_{DM}$	0.000e+000 ± 0.000e+000
	LFPSO$_{DM}$	0.000e+000 ± 0.000e+000
	GCPSO	8.937e-023 ± 1.699e-022
Rosenbrock	GDPSO$_{DM}$	1.196e+000 ± 1.845e+000
	LFPSO$_{DM}$	8.771e-001 ± 1.668e+000
	GCPSO	1.886e+001 ± 1.150e+001
Quartic	GDPSO$_{DM}$	4.532e-001 ± 1.401e-001
	LFPSO$_{DM}$	4.557e-001 ± 1.432e-001
	GCPSO	5.443e-001 ± 1.556e-001
Schwefel	GDPSO$_{DM}$	4.394e+003 ± 6.324e+002
	LFPSO$_{DM}$	4.226e+003 ± 5.085e+002
	GCPSO	4.192e+003 ± 5.881e+002
Rastrigin	GDPSO$_{DM}$	8.674e+001 ± 2.136e+001
	LFPSO$_{DM}$	1.175e+002 ± 5.177e+001
	GCPSO	8.178e+000 ± 3.807e+000
Ackley	GDPSO$_{DM}$	1.034e+001 ± 5.489e+000
	LFPSO$_{DM}$	2.260e+000 ± 6.190e+000
	GCPSO	1.432e+000 ± 4.927e+000
Griewank	GDPSO$_{DM}$	6.968e+002 ± 1.925e+002
	LFPSO$_{DM}$	1.182e-003 ± 7.072e-003
	GCPSO	1.431e-002 ± 1.743e-002
Six Hump Camel Back	GDPSO$_{DM}$	-2.845e-005 ± 0.000e+000
	LFPSO$_{DM}$	-2.845e-005 ± 0.000e+000
	GCPSO	-2.845e-005 ± 0.000e+000

than the LFPSO$_{DM}$.

A similar hypothesis is proposed to evaluate the performance of GDPSO$_{DM}$ and HGPSO:

Hypothesis 5.15 *GDPSO$_{DM}$ performs no worse than GCPSO with respect to accuracy.*

Table 5.29: Mann-Whitney U Test P-Values for Experiment 9

		GDPSO$_{DM}$	LFPSO$_{DM}$	GCPSO
Spherical	GDPSO$_{DM}$		1.00e+000	2.20e-016
	LFPSO$_{DM}$	1.00e+000		2.20e-016
	GCPSO	2.20e-016	2.20e-016	
Rosenbrock	GDPSO$_{DM}$		5.60e-004	2.20e-016
	LFPSO$_{DM}$	5.60e-004		2.20e-016
	GCPSO	2.20e-016	2.20e-016	
Quartic	GDPSO$_{DM}$		7.70e-001	8.37e-003
	LFPSO$_{DM}$	7.70e-001		5.29e-003
	GCPSO	8.37e-003	5.29e-003	
Schwefel	GDPSO$_{DM}$		2.51e-001	2.40e-001
	LFPSO$_{DM}$	2.51e-001		8.39e-001
	GCPSO	2.40e-001	8.39e-001	
Rastrigin	GDPSO$_{DM}$		1.33e-003	2.20e-016
	LFPSO$_{DM}$	1.33e-003		2.20e-016
	GCPSO	2.20e-016	2.20e-016	
Ackley	GDPSO$_{DM}$		9.46e-012	8.95e-014
	LFPSO$_{DM}$	9.46e-012		2.40e-008
	GCPSO	8.95e-014	2.40e-008	
Griewank	GDPSO$_{DM}$		2.20e-016	2.20e-016
	LFPSO$_{DM}$	2.20e-016		1.10e-009
	GCPSO	2.20e-016	1.10e-009	
Six Hump Camel Back	GDPSO$_{DM}$		1.00e+000	1.00e+000
	LFPSO$_{DM}$	1.00e+000		1.00e+000
	GCPSO	1.00e+000	1.00e+000	

There exist problems where GDPSO$_{DM}$ exhibits superior performance to GCPSO.

The p-values from table 5.29 indicate that for functions Quartic, Schwefel and Six Hump Camel Back, no significant difference between the algorithms could be found. Table 5.28 shows that while GDPSO$_{DM}$ exhibited lower mean error on Spherical and Rosenbrock, GCPSO exhibited significantly lower mean error on functions Rastrigin, Ackley and Griewank. Hypothesis 5.15 is thus invalid.

The above findings are once again supported by the reliability and efficiency measures in Table 5.30. In the case of the Spherical function, GDPSO$_{DM}$ exhibited superior

Table 5.30: Reliability and Efficiency for Experiment 9

Function	Algorithm	Reliability	Efficiency
Spherical	GDPSO_{DM}	100%	1.000e+003
	LFPSO_{DM}	100%	2.100e+003
	GCPSO	0%	—
Rosenbrock	GDPSO_{DM}	100%	4.880e+003
	LFPSO_{DM}	100%	1.300e+003
	GCPSO	100%	5.920e+003
Quartic	GDPSO_{DM}	36%	1.864e+004
	LFPSO_{DM}	34%	2.388e+004
	GCPSO	16%	4.502e+004
Schwefel	GDPSO_{DM}	98%	5.776e+003
	LFPSO_{DM}	100%	4.580e+003
	GCPSO	100%	4.060e+003
Rastrigin	GDPSO_{DM}	76%	2.187e+004
	LFPSO_{DM}	34%	7.306e+004
	GCPSO	100%	1.114e+004
Ackley	GDPSO_{DM}	12%	5.667e+003
	LFPSO_{DM}	88%	3.543e+004
	GCPSO	92%	1.291e+004
Griewank	GDPSO_{DM}	0%	—
	LFPSO_{DM}	100%	5.398e+004
	GCPSO	100%	7.380e+003
Six Hump Camel Back	GDPSO_{DM}	100%	1.000e+003
	LFPSO_{DM}	100%	1.000e+003
	GCPSO	100%	1.120e+003

reliability. The reliability measurements were tied at 100% for Rosenbrock, however GDPSO_{DM} did exhibit superior efficiency. In the case of all the functions where GCPSO was found to have a significantly lower mean error than GDPSO_{DM}, namely Rastrigin, Ackley and Griewank, the GCPSO exhibited superior reliability.

The following hypothesis is proposed in order to correlate the performance differences between the algorithms to the problem types:

Hypothesis 5.16 *Considering the functions where a significant difference in performance was found, GCPSO outperforms GDPSO$_{DM}$ and LFPSO$_{DM}$ on multimodal problems with respect to accuracy, and the opposite is true on unimodal problems.*

The functions that exhibited a significant difference in the performance of the above algorithms are Spherical, Rosenbrock, Rastrigin, Ackley and Griewank. GCPSO outperformed the gradient algorithms on Rastrigin and Ackley, which are clearly multimodal functions. The gradient algorithms outperformed GCPSO on Spherical and Rosenbrock, which are classified as unimodal problems. The performance of the algorithms on the Griewank function, however, causes a problem for hypothesis 5.16. Griewank is a multimodal function which tends to become unimodal at higher dimensions [62], making it difficult to classify as outright multimodal or unimodal. The results in table 5.25 for the Griewank function would be problematic in either case; LFPSO$_{DM}$ outperformed GCPSO, which in turn outperformed GDPSO$_{DM}$. This exception to the above pattern means that 5.16 cannot be outright confirmed, and is thus invalid.

Hypotheses 5.14 and 5.15 have been shown to be invalid. It can thus be said that none of the algorithms is outright superior to any other algorithm. The failure of hypothesis 5.16 to confirm the trend of gradient algorithms performing better on unimodal problems also highlights the difficulty of categorising the performance characteristics of these algorithms.

5.5.4 GDPSO$_{DM}$, LFPSO$_{DM}$ and Mutating PSO

The gradient PSO variants GDPSO$_{DM}$ and LFPSO$_{DM}$ are compared to the Mutating PSO. Experiment 10 was designed to contrast the performance of the above algorithms. Table 5.31 shows the mean error, while table 5.32 gives the Mann-Whitney U test p-values prior to correction. Table 5.33 provides the reliability and efficiency measurements for the experiment.

The following hypothesis is proposed in order to evaluate the performance of LFPSO$_{DM}$ and Mutating PSO:

Table 5.31: Mean Error for Experiment 10

Problem	Algorithm	Error
Spherical	GDPSO$_{DM}$	0.000e+000 ± 0.000e+000
	LFPSO$_{DM}$	0.000e+000 ± 0.000e+000
	Mutating PSO	1.013e-029 ± 5.911e-029
Rosenbrock	GDPSO$_{DM}$	1.196e+000 ± 1.845e+000
	LFPSO$_{DM}$	8.771e-001 ± 1.668e+000
	Mutating PSO	1.273e+001 ± 2.057e+000
Quartic	GDPSO$_{DM}$	4.532e-001 ± 1.401e-001
	LFPSO$_{DM}$	4.557e-001 ± 1.432e-001
	Mutating PSO	5.518e-001 ± 1.578e-001
Schwefel	GDPSO$_{DM}$	4.394e+003 ± 6.324e+002
	LFPSO$_{DM}$	4.226e+003 ± 5.085e+002
	Mutating PSO	7.077e+002 ± 5.466e+002
Rastrigin	GDPSO$_{DM}$	8.674e+001 ± 2.136e+001
	LFPSO$_{DM}$	1.175e+002 ± 5.177e+001
	Mutating PSO	1.124e+001 ± 5.896e+000
Ackley	GDPSO$_{DM}$	1.034e+001 ± 5.489e+000
	LFPSO$_{DM}$	2.260e+000 ± 6.190e+000
	Mutating PSO	1.700e-014 ± 6.531e-014
Griewank	GDPSO$_{DM}$	6.968e+002 ± 1.925e+002
	LFPSO$_{DM}$	1.182e-003 ± 7.072e-003
	Mutating PSO	1.553e-002 ± 1.678e-002
Six Hump Camel Back	GDPSO$_{DM}$	-2.845e-005 ± 0.000e+000
	LFPSO$_{DM}$	-2.845e-005 ± 0.000e+000
	Mutating PSO	-2.845e-005 ± 0.000e+000

Hypothesis 5.17 *LFPSO$_{DM}$ performs no worse than Mutating PSO with respect to accuracy. There exist problems where LFPSO$_{DM}$ exhibits superior performance to Mutating PSO.*

Considering the p-values from Table 5.32, no significant difference can be seen for the

Table 5.32: Mann-Whitney U Test P-Values for Experiment 10

		GDPSO$_{DM}$	LFPSO$_{DM}$	Mutating PSO
Spherical	GDPSO$_{DM}$		1.00e+000	2.20e-016
	LFPSO$_{DM}$	1.00e+000		2.20e-016
	Mutating PSO	2.20e-016	2.20e-016	
Rosenbrock	GDPSO$_{DM}$		5.60e-004	2.20e-016
	LFPSO$_{DM}$	5.60e-004		2.20e-016
	Mutating PSO	2.20e-016	2.20e-016	
Quartic	GDPSO$_{DM}$		7.70e-001	3.00e-003
	LFPSO$_{DM}$	7.70e-001		4.00e-003
	Mutating PSO	3.00e-003	4.00e-003	
Schwefel	GDPSO$_{DM}$		2.51e-001	2.20e-016
	LFPSO$_{DM}$	2.51e-001		2.20e-016
	Mutating PSO	2.20e-016	2.20e-016	
Rastrigin	GDPSO$_{DM}$		1.33e-003	2.20e-016
	LFPSO$_{DM}$	1.33e-003		2.20e-016
	Mutating PSO	2.20e-016	2.20e-016	
Ackley	GDPSO$_{DM}$		9.46e-012	2.20e-016
	LFPSO$_{DM}$	9.46e-012		2.20e-016
	Mutating PSO	2.20e-016	2.20e-016	
Griewank	GDPSO$_{DM}$		2.20e-016	2.20e-016
	LFPSO$_{DM}$	2.20e-016		2.81e-011
	Mutating PSO	2.20e-016	2.81e-011	
Six Hump Camel Back	GDPSO$_{DM}$		1.00e+000	1.00e+000
	LFPSO$_{DM}$	1.00e+000		1.00e+000
	Mutating PSO	1.00e+000	1.00e+000	

Quartic and Six Hump Camel Back functions. In the case of the remaining functions, Table 5.31 shows that Mutating PSO outperformed LFPSO$_{DM}$ on Schwefel, Rastrigin and Ackley. The situation was reversed for functions Rosenbrock and Griewank, however the above result nevertheless renders hypothesis 5.17 invalid. If Griewank is classified as a unimodal function, it can be said that the general tendency was for the LFPSO$_{DM}$ to perform better on unimodal functions, while the Mutating PSO performed better on multimodal ones.

The reliability measure from Table 5.33 provides further support for the above finding.

Table 5.33: Reliability and Efficiency for Experiment 10

Function	Algorithm	Reliability	Efficiency
Spherical	GDPSO$_{DM}$	100%	1.000e+003
	LFPSO$_{DM}$	100%	2.100e+003
	MutatingPSO	100%	1.224e+005
Rosenbrock	GDPSO$_{DM}$	100%	4.880e+003
	LFPSO$_{DM}$	100%	1.300e+003
	MutatingPSO	100%	6.300e+003
Quartic	GDPSO$_{DM}$	36%	1.864e+004
	LFPSO$_{DM}$	34%	2.388e+004
	MutatingPSO	26%	4.804e+004
Schwefel	GDPSO$_{DM}$	98%	5.776e+003
	LFPSO$_{DM}$	100%	4.580e+003
	MutatingPSO	100%	7.760e+003
Rastrigin	GDPSO$_{DM}$	76%	2.187e+004
	LFPSO$_{DM}$	34%	7.306e+004
	MutatingPSO	100%	1.076e+004
Ackley	GDPSO$_{DM}$	12%	5.667e+003
	LFPSO$_{DM}$	88%	3.543e+004
	MutatingPSO	100%	9.720e+003
Griewank	GDPSO$_{DM}$	0%	—
	LFPSO$_{DM}$	100%	5.398e+004
	MutatingPSO	100%	1.568e+004
Six Hump Camel Back	GDPSO$_{DM}$	100%	1.000e+003
	LFPSO$_{DM}$	100%	1.000e+003
	MutatingPSO	100%	1.320e+003

While for the functions Schwefel, Rosenbrock and Griewank, the reliability measurement was tied at 100%, on the Rastrigin and Ackley functions, the Mutating PSO exhibited superior reliability to the LFPSO$_{DM}$.

In order to compare the performance of GDPSO$_{DM}$ and Mutating PSO, the following hypothesis is proposed:

Hypothesis 5.18 *GDPSO$_{DM}$ performs no worse than Mutating PSO with respect to accuracy. There exist problems where GDPSO$_{DM}$ exhibits superior performance to Mutating PSO.*

Table 5.32 shows that of all the functions, only Quartic and Six Hump Camel Back did not show a significant difference. Table 5.31 indicates that GDPSO$_{DM}$ was indeed superior on the Rosenbrock function, while Mutating PSO was superior on Schwefel, Rastrigin, Ackley and Griwank. Hypothesis 5.18 is thus shown to be invalid.

Taking into consideration the reliability measurement from Table 5.33, the above finding can be reinforced. With the exception of the measurement for Rosenbrock, which was tied at 100%, the reliability of the Mutating PSO was superior to that of the GDPSO$_{DM}$ on Schwefel, Rastrigin, Ackley and Griwank.

The results are similar to that of the LFPSO$_{DM}$, GDPSO$_{DM}$, GCPSO comparison from section 5.5.3. It is perhaps worthwhile to re-evaluate hypothesis 5.16, this time with Mutating PSO in place of GCPSO:

Hypothesis 5.19 *On functions which exhibited a significant difference, Mutating PSO outperforms GDPSO$_{DM}$ and LFPSO$_{DM}$ on multimodal problems with respect to accuracy, and the opposite is true on unimodal problems.*

As in the case of hypothesis 5.16, we can see that the gradient algorithms offer improved performance on Spherical and Rosenbrock, the two unimodal functions that showed a significant difference in their p-values. Mutating PSO is superior on multimodal functions Schwefel, Rastrigin and Ackley. The result for Griewank is once again the problem; as was the case in hypothesis 5.16, LFPSO$_{DM}$ exhibits a significantly lower mean error than Mutating PSO. Mutating PSO outperforms GDPSO$_{DM}$, making it inconsequential to argue the case of whether the Griewank function should be considered unimodal or multimodal. Hypothesis 5.19 is thus shown to be false.

All 3 hypotheses in this section (5.17, 5.18, 5.19), as was the case with section 5.5.3 have been shown to be invalid. It can thus be said that none of the algorithms are outright superior to each other. The failure of hypothesis 5.19 to show with absolute certainty the superiority of Mutating PSO for multimodal problems nevertheless does not detract from a strong lean towards that conclusion.

5.5.5 Summary

The performance of GDPSO$_{DM}$ and LFPSO$_{DM}$ was contrasted against HGPSO, GCPSO and Mutating PSO. The modified PSO algorithms introduce significant improvements to the standard PSO algorithm, and this was reflected in the results in general. Hypotheses 5.12 - 5.19 attempted to indicate whether one algorithm was superior to another, with only hypothesis 5.12 shown to be valid. In all other cases, contradictory results made it impossible to state that one algorithm was preferred over another.

The results obtained for the Griewank function in sections 5.5.3 and 5.5.4 prevented the further refinement of the hypotheses into comparisons which take into account the modality of the problem.

These results show that the GDPSO$_{DM}$ and LFPSO$_{DM}$ algorithms are generally competitive with other PSOs which are considered to exhibit good performance.

5.6 Conclusions

This chapter provides empirical evidence and an analysis of the performance characteristics of standard PSO, gradient-based algorithms, the new gradient-based PSOs, GDPSO and LFPSO, and some existing PSO algorithms, namely the GCPSO, HGPSO and Mutating PSO.

A systematic approach was used to highlight the different performance characteristics of the algorithms, and to justify the direction of the experiments. A series of hypotheses were given, each one building upon the findings of the previous ones.

The strength of the gradient-based methods, including standard GD, but also specifically GDPSO and LFPSO was shown on unimodal functions. The gradient-based algorithms generally tend to find solutions of better quality, with higher reliability and speed. The gradient-based PSOs retain this performance characteristic on unimodal functions, which is their primary benefit.

It has been shown that without any modifications, the GDPSO exhibited decreased performance, with respect to accuracy, on multimodal functions. The logical construction of the algorithms, however, lent itself well to simple yet effective means of countering the problems of premature convergence. The approach which resulted in the best results was the delayed introduction of gradient information. Using delayed introduction of gradient information, the GDPSO$_D$ algorithm achieved equal performance to standard PSO even on multimodal functions. The LFPSO, even without any modifications, performed on par with GDPSO$_D$, and thus standard PSO. This result seems to indicate that the LFPSO is slightly more robust than the GDPSO, although later results made it impossible to say that LFPSO always outperforms GDPSO.

The diversity of the gradient-hybrid swarms was contrasted to that of the standard PSO. The inclusion of gradient information did not seem to affect the diversity of the swarms, in general. This result is logical, considering that only one particle per swarm was directly influenced by gradient information. Nevertheless, the performance of unmodified gradient-hybrid swarms on multimodal functions was negatively influenced.

In order to be competitive with the remaining non-standard PSO algorithms, namely HGPSO, GCPSO and Mutating PSO, both modified acceleration coefficients and delayed introduction of gradient information were employed. Under these conditions, the GDPSO and LFPSO were found to perform well against the HGPSO, with the LFPSO achieving a lower mean error than the HGPSO on all functions where a statistically significant difference was shown. The GDPSO and LFPSO performed reasonably well against the GCPSO and Mutating PSO, with the gradient algorithms generally performing well on unimodal functions and worse on multimodal functions. In all cases, exceptions to this general rule made it impossible to categorically state this tendency. Nevertheless, the ability of the GDPSO and LFPSO to compete against advanced PSOs (which are arguably more designed for and suited to multimodal functions) even on multimodal functions, is encouraging.

To summarise the conclusions of this chapter, the new gradient based algorithms GDPSO and LFPSO were found to perform well overall. A general tendency for premature convergence was found, but could be effectively countered using simple methods. The main strength of the new algorithms lies in solving unimodal problems, but they perform reasonably well on multimodal ones as well.

Chapter 6

Conclusions

The conclusions of the thesis are summarised in section 6.1. Section 6.2 suggests directions for future research.

6.1 Summary

This thesis investigated the construction and performance characteristics of gradient based particle swarm optimisers.

Chapter 3 investigated the concept of using gradient information in PSO.

Two new gradient based PSOs were constructed, the Gradient Descent PSO (GDPSO; algorithm 3.1), and the LeapFrog PSO (LFPSO; algorithm 3.2). The concept of viewing PSO algorithms as either homogeneous or heterogeneous was discussed in section 3.1. The Generalised Local Search PSO (GLSPSO; algorithm 3.3) was introduced purely as a formalisation of this concept.

Chapter 5 used empirical evidence was to investigate the performance characteristics of gradient based algorithms, as well as to test the performance of the GDPSO and the LFPSO.

Section 5.1 showed that gradient algorithms are excellent at exploitation, evidenced by their efficacy when dealing with unimodal problems.

Section 5.2 showed that gradient algorithms can be combined with PSO. The decrease in performance of the gradient-hybrid algorithm was shown to most likely be a result of premature convergence, due to the rapid exploitation capabilities of the gradient algorithm.

Section 5.3 systematically proposed and evaluated several measures to counter the premature convergence of the GDPSO algorithm. Delayed introduction of gradient information, specifically, was shown in section 5.3.2 to be an effective means of preventing premature convergence.

The LFPSO's performance was investigated in section 5.4, showing that an unmodified LFPSO performed equally well or better than a GDPSO with delayed gradient information (section 5.4.2). This result seems to suggest that the LeapFrog algorithm is less parameter dependent than gradient descent, as well as having less of a tendency to cause the swarm to prematurely converge. Section 5.4.4 compared the performance of LFPSO against GDPSO, with each algorithm being optimally configured for each function, resulting in similar performance characteristics.

Section 5.5 tested optimised versions of LFPSO and GDPSO against the HGPSO, GCPSO and Mutating PSO.

In the comparison against HGPSO (section 5.5.1), the new gradient-hybrid algorithms performed well, often outperforming their HGPSO counterpart. LFPSO specifically was formally shown to exhibit either equal or superior performance when compared to HGPSO. The difference in performance can possibly be explained by the fact that HGPSO is a homogeneous algorithm, while the LFPSO and GDPSO are heterogeneous (discussed in section 3.1). This characteristic may allow the latter algorithms to better retain the performance characteristics of the original PSO algorithm.

Sections 5.5.3 and 5.5.4 tested the gradient hybrid algorithms GDPSO and LFPSO against GCPSO and Mutating PSO, respectively. Both GCPSO and Mutating PSO exhibited good performance on multimodal problems, with the exception of the Griewank function. The exception with Griewank made it impossible to formally confirm the hypothesis that these algorithms outperform the gradient algorithms on multimodal problems.

This thesis set out to show that gradient information can be incorporated into PSO without adversely affecting its performance on multimodal functions, while still gaining benefits on unimodal ones. This result has been clearly confirmed using empirical evidence.

The comparisons against the modified PSOs (HGPSO, GCPSO, Mutating PSO) showed that GDPSO and LFPSO were often able to outperform their counterparts on unimodal problems. Surprisingly, it was not possible to show that the gradient algorithms were inferior to GCPSO and Mutating PSO on multimodal problems. This result seems to confirm the argument for the heterogeneous construction of these new gradient-hybrid PSO algorithms.

6.2 Further Research

This section discusses some possible directions for future research.

6.2.1 Combining Gradient PSOs and Mutation

This thesis has reinforced the general notion that gradient algorithms are good exploiters, primarily suited for unimodal problems. By combining a gradient-PSO with mutation, it may be possible to construct a new PSO which is ideally suited to both unimodal and highly multimodal problems. Several different approaches might be investigated:

- Random selection of particles and dimensions for mutation.
- Mutation of the worst performing particles in the swarm after each iteration.
- The introduction of a new particle type which is concerned primarily with exploration. This particle's movements could be a completely random, or alternatively it could be given a few iterations to exploit a new area before being mutated again.

6.2.2 Advanced PSO Topologies

Section 2.4.3 discussed several PSO topologies. This thesis has been primarily concerned with the *gbest* and *lbest* topologies, but it is possible that some of the other topologies would help in preventing premature convergence by delaying the distribution of information throughout the swarm. A standard topology which has these benefits would be superior to the delayed introduction of gradient information and modification of acceleration constants used in this thesis, since it would have less dependence on reasonable parameters, which can be function specific.

6.2.3 Swarm Gradient Estimation

It can be argued that gradient information is rarely available in real-world scenarios. Section 3.4 showed that it is possible to estimate gradients relatively accurately. However, the method is computationally expensive, and does not scale well with increasing dimensions. One interesting possibility is to enable sharing of information in the swarm to estimate the gradient.

The approach here would involve using the position and fitness of other, physically nearby particles in the swarm in order to approximate the gradient at the position of any given particle. While this approach seems similar to other, spacial position aware PSOs, it is different due to the fact that the approximated gradient would not serve as a means of a neighbourhood best selector mechanism, but rather as an input into a gradient algorithm.

Bibliography

[1] D. H. Ackley. *A connectionist machine for genetic hillclimbing.* Kluwer, Boston, 1987.

[2] P. Angeline. Evolutionary optimization versus particle swarm optimization: Philosophy and performance differences. In *Proceedings of the 7th International Conference on Evolutionary Programming*, pages 601–610, 1998.

[3] T. Bäck. *Evolutionary algorithms in theory and practice.* Oxford University Press, 1996.

[4] E. Bonabeau, M. Dorigo, and G. Theraulaz. *Swarm Intelligence: From Natural to Artificial Systems.* Oxford University Press, USA, 1999.

[5] E. Bonabeau, G. Theraulaz, and J. L. Deneubourg. Quantitative study of the fixed threshold model for the regulation of division of labour in insect societies. *Royal Society of London Proceedings Series B*, 263:1565–1569, Nov 1996.

[6] B. Borowska and S. Nadolski. Particle swarm optimization: the gradient correction. *Journal of Applied Computer Science*, 17(2):7–15, 2009.

[7] M. Clerc. *Particle Swarm Optimization.* Wiley-ISTE, 2006.

[8] M. Clerc and J. Kennedy. The particle swarm - explosion, stability, and convergence in a multidimensional complex space. *Evolutionary Computation, IEEE Transactions on*, 6(1):58–73, 2002.

[9] F. Van den Bergh. *An Analysis of Particle Swarm Optimizers.* PhD thesis, University of Pretoria, Pretoria, South Arica, 2002.

[10] F. Van den Bergh and A. P. Engelbrecht. A new locally convergent particle swarm optimizer. In *Proceedings of the IEEE international conference on systems, man and cybernetics*, volume 3, pages 6–12, 2002.

[11] F. Van den Bergh and A. P. Engelbrecht. A study of particle swarm optimization particle trajectories. *Information Sciences*, 176(8):937–971, April 2006.

[12] J. E. Dennis, Jr., and J. J. Morée. Quasi-Newton methods, motivation and theory. *SIAM Review*, 19(1):46–89, 1977.

[13] L. Duwang and R. Tymerski. Comparison of simulation algorithms for accelerated determination of periodic steady state of switched networks. *IEEE Transactions on Industrual Electronics*, 47:1278–1285, 2000.

[14] R. C. Eberhart and J. Kennedy. A new optimiser using particle swarm theory. In *Proceedings of the Sixth International Symposium on Micro Machine and Human Science*, pages 39–43, 1995.

[15] R. C. Eberhart, P. K. Simpson, and R. W. Dobbins. *Computational Intelligence PC Tools*. Academic Press Professional, 1996.

[16] A. I. Edwards and A. P. Engelbrecht. Comparing different optimisation strategies for nonlinear mapping and mapping large datasets using neural networks. *SATNAC*, 2006.

[17] A. I. Edwards and A. P. Engelbrecht. Comparing particle swarm optimisation and genetic algorithms for nonlinear mapping. *The 2006 IEEE Congress on Evolutionary Computation*, pages 694–701, 2006.

[18] A. I. Edwards, A. P. Engelbrecht, and N. Franken. Nonlinear mapping using particle swarm optimisation. *The 2005 IEEE Congress on Evolutionary Computation*, 1:306–313, 2005.

[19] A. P. Engelbrecht. *Computational Intelligence: An Introduction*. Wiley, 2002.

[20] A. P. Engelbrecht. *Fundamentals of Computational Swarm Intelligence*. Wiley, 2005.

[21] A. P. Engelbrecht and A. Ismail. Training product unit neural networks. *Stability and Control: Theory and Applications*, 2:59–74, 1999.

[22] S. C. Esquivel and C. A. Coello Coello. On the use of particle swarm optimization with multimodal functions. *The 2003 Congress on Evolutionary Computation*, 2:1130–1136, 2003.

[23] R. Fletcher and C. M. Reeves. Function minimization by conjugate gradients. *The Computer Journal*, 7(2):149–154, 1964.

[24] D. Greenspan. *Discrete Models*. Addison-Wesley, Massachusetts, 1973.

[25] S. J. Huang, S. N. Koh, and H. K. Tang. Training algorithm based on Newton's method with dynamic error control. *1992 IJCNN International Joint Conference on Neural Networks*, 3:899–904, 1992.

[26] J. Kennedy. Small worlds and mega-minds: Effects of neighborhood topology on particle swarm performance. In *Proceedings of the IEEE Congress on Evolutionary Computation*, volume 3, pages 1931–1938, July 1999.

[27] J. Kennedy, R. Eberhart, and Y. Shi. *Swarm Intelligence*. Morgan Kaufmann, 2001.

[28] J. Kennedy and R. C. Eberhart. Particle swarm optimization. In *Proceedings of the IEEE Internation Conference on Neural Networks*, volume 4, pages 1942–1948, 1995.

[29] J. Kennedy and R. Mendes. Population structure and particle performance. In *In Proceedings of the IEEE Congress on Evolutionary Computation*, pages 1671–1676, 2002.

[30] T. Krink, J. S. Vesterstrøm, and J. Riget. Particle swarm optimisation with spatial particle extension. In *Proceedings of the Fourth Congress on Evolutionary Computation (CEC-2002)*, volume 2, pages 1474–1479, 2002.

[31] R. E. Larson, R. P. Hostetler, B. H. Edwards, and D. E. Heyd. *Calculus with Analytic Geometry*. Houghton Mifflin Company, 6th edition, 1998.

[32] J. J. Liang, P. N. Suganthan, and K. Deb. Novel composition test functions for numerical global optimization. In *Proceedings of the 2005 Swarm Intelligence Symposium*, pages 68–75, 2005.

[33] J. Liu, W. Xu, and J. Sun. Particle swarm optimization with mutation operator. In *The 2004 International Conference on Machine Learning and Cybernetics*, volume 4, pages 2251–2256, 2004.

[34] H. B. Mann and D. R. Whitney. On a test of whether one of two random variables is stochastically larger than the other. *Annals of Mathematical Statistics*, 18:50–60, 1947.

[35] T. Marill. Emulating the human interpretation of line-drawings as three-dimensional objects. *International Journal of Computer Vision*, 6(2):147–161, 1991.

[36] M. Matsumoto and T. Nishimura. Mersenne twister: A 623-dimensionally equidistributed uniform pseudorandom number generator. *ACM Transactions on Modeling and Computer Simulation*, 8(1):3–30, January 1998.

[37] R. Mendes, P. Cortez, M. Rocha, and J. Neves. Particle swarms for feed-forward neural network training. In *Proceedings of the International Joint Conference on Neural Networks*, pages 1895–1899, 2002.

[38] Z. Michalewicz. *Genetic Algorithms + Data Structures = Evolution Programs*. Springer-Verlag, 3rd edition, 1996.

[39] P. M. Morse and H. Feshbach. *Methods of Theoretical Physics*. McGraw-Hill, New York, 1953.

[40] M. M. Noel and T. C. Jannett. Simulation of a new hybrid particle swarm optimization algorithm. In *Proceedings of the Thirty-Sixth Southeastern Symposium on System Theory*, pages 150–153, 2004.

[41] University of Pretoria. Computational intelligence library (cilib). http://cilib.sourceforge.net/.

[42] M. G. Omran, A. Salman, and A. P. Engelbrecht. Image classification using particle swarm optimisation. In *Proceedings of the Fourth Asia-Pacific Conference on Simulated Evolution and Learning*, pages 370–374, 2002.

[43] E. Ozcan and C. K. Mohan. Analysis of a simple particle swarm optimization system. *Intelligent Engineering Systems through Artificial Neural Networks*, pages 253–258, 1998.

[44] E. Ozcan and C. K. Mohan. Particle swarm optimization: Surfing the waves. In *Proceedings of the IEEE Congress on Evolutionary Computation*, volume 3, July 1999.

[45] E. S. Peer, F. Van den Bergh, and A. P. Engelbrecht. Using neighborhoods with the guaranteed convergence pso. In *Proceedings of the IEEE Swarm Intelligence Symposium*, pages 235–242. IEEE Press, 2003.

[46] R. Poli and D. Broomhead. Exact analysis of the sampling distribution for the canonical particle swarm optimiser and its convergence during stagnation. In *Proceedings of the 9th annual conference on Genetic and evolutionary computation*, GECCO '07, pages 134–141, New York, NY, USA, 2007. ACM.

[47] R. Poli, J. Kennedy, and T. Blackwell. Particle swarm optimization; an overview. *Swarm Intelligence*, 1(1):33–57, 2007.

[48] L. Qi and J. Sun. A nonsmooth version of Newton's method. *Mathematical Programming*, 58(1–3):353–367, 1993.

[49] S. S. Shapiro and M. B. Wilk. An analysis of variance test for normality (complete samples). *Biometrika*, 52(3/4):591–611, 1965.

[50] Y. Shi and R. C. Eberhart. A modified particle swarm optimizer. In *Proceedings of the IEEE Congress on Evolutionary Computation*, pages 69–73, 1998.

[51] Y. Shi and R. C. Eberhart. Parameter selection in particle swarm optimisation. In *Proceedings of the Seventh Annual Conference on Evolutionary Programming*, pages 591–600, 1998.

[52] J. A. Snyman. A new and dynamic method for unconstrained minimization. *Applied Mathematical Modelling*, 6, December 1982.

[53] J. A. Snyman. A new convergent minimization method for functions with positive definite Hessian matrices. Technical report, Department of Applied Mathematics, University of Pretoria, 1982. Research Report UP TW 28.

[54] J. A. Snyman. An improved version of the original leapfrog dynamic method for unconstained minimization. *Applied Mathematical Modelling*, 7, June 1983.

[55] T. Sousa, A. Silva, and A. Neves. Particle swarm based data mining algorithms for classification tasks. *Parallel Computing*, 30:767–783, 2004.

[56] P. N. Suganthan. Particle swarm optimiser with neighborhood operator. In *Proceedings of the IEEE Congress on Evolutionary Computation*, volume 3, pages 1958–1962. IEEE Press, 1999.

[57] R Project Team. The r project for statistical computing. http://www.r-project.org/.

[58] B. B. Thompson, R. J. Marks, M. A. El-Sharkawi, W. J. Fox, and R. T. Miyamoto. Inversion of neural network underwater acoustic model for estimation of bottom parameters using modified particle swarm optimizer. In *Proceedings of the International Joint Conference on Neural Networks*, volume 2, pages 1301–1306, 2003.

[59] I. C. Trelea. The particle swarm optimization algorithm: convergence analysis and parameter selection. *Information Processing Letters*, 85(6):317–325, March 2003.

[60] J. S. Vesterstrøm, J. Riget, and T. Krink. Division of labor in particle swarm optimization. In *Proceedings of IEEE Congress on Evolutionary Computation*, pages 1570–1575. IEEE Press, 2002.

[61] P. Werbos. *Beyond Regression: New Tools for Prediction and Analysis in the Behavioral Sciences*. PhD thesis, Harvard University, Cambridge, MA, 1974.

[62] D. Whitley, S. Rana, J. Dzubera, and K. E. Mathias. Evaluating evolutionary algorithms. *Artificial Intelligence*, 85(1-2):245–276, 1996.

[63] A. Windisch, S. Wappler, and J. Wegener. Applying particle swarm optimization to software testing. In *Proceedings of the 9th annual conference on Genetic and evolutionary computation*, pages 1121–1128, 2007.

[64] R. Zhang, W. Zhang, and X. Zhang. A new hybrid gradient-based particle swarm optimization algorithm and its applications to control of polarization mode dispersion compensation in optical fiber communication systems. *The 2009 International Joint Conference on Computational Sciences and Optimization*, pages 1031–1033, 2009.

Appendix A

Shifting of Benchmark Functions

Section 4.2 discussed the need for, and rationale behind, the shifting of benchmark functions. All experimental results given in this thesis have been obtained using shifted functions, in order to counter the problems with traditional benchmark functions as described by Liang *et al.* [32]. The following tables contain the specific values by which the dimensions of the benchmark functions were shifted. Table (A.1) contains the shift values for the Spherical (f_1), Rosenbrock (f_2), Quartic (f_3) and Schwefel (f_4) functions. Table (A.2) contains the shift values for the remaining functions, namely Rastrigin (f_5), Ackley (f_6), Griewank (f_7) and Six Hump Camel Back (f_8) functions.

The shift amounts were obtained using the procedure described in section 4.2. The values are given here per dimension per function, up to an accuracy of 5 decimal places. The functions Schwefel and Six Hump Camel Back are exceptions to the general procedure and have shift amounts of 0, since they are already shifted. The rationale for not shifting these specific functions further was discussed in detail in section 4.2.

Table A.1: Function shifts for functions f_1, f_2, f_3 and f_4

Dimension	Spherical f_1	Rosenbrock f_2	Quartic f_3	Schwefel f_4
1	-3.13106	-0.23551	-0.90072	0.00000
2	4.45136	0.39465	1.15233	0.00000
3	1.43454	-0.89321	-1.12649	0.00000
4	4.92868	-0.93500	-0.25146	0.00000
5	-0.36802	-0.84764	-0.15765	0.00000
6	-5.04586	-0.53109	-0.85430	0.00000
7	3.87438	-0.36262	-1.13078	0.00000
8	4.00724	0.91614	1.09432	0.00000
9	2.03782	-0.65749	0.94920	0.00000
10	-0.52178	-0.88368	1.04792	0.00000
11	2.07289	-0.32709	0.05436	0.00000
12	3.15103	0.27325	1.24261	0.00000
13	4.23762	-0.52578	-0.66383	0.00000
14	-2.66532	0.18985	0.17855	0.00000
15	-1.29870	0.55900	0.30678	0.00000
16	-0.93053	0.89719	-0.69735	0.00000
17	1.09594	-0.33170	0.14283	0.00000
18	-1.59012	0.23425	-0.19033	0.00000
19	-2.33478	-0.29136	-0.25675	0.00000
20	-1.80866	0.70062	-0.88586	0.00000
21	-0.43572	-0.32650	1.20915	0.00000
22	-0.89456	-0.69322	0.30756	0.00000
23	-0.58677	0.96943	0.95485	0.00000
24	3.69869	0.11325	-0.51853	0.00000
25	4.48195	-0.09759	-0.12721	0.00000
26	2.35420	-0.31656	0.34183	0.00000
27	-4.72619	-0.55338	-0.07495	0.00000
28	2.35521	0.19870	0.18935	0.00000
29	0.84086	0.27555	1.07900	0.00000
30	2.39426	-0.73669	-0.66545	0.00000

Table A.2: Function shifts for functions f_5, f_6, f_7 and f_8

Dimension	Rastrigin f_5	Ackley f_6	Griewank f_7	Six Hump Camel Back f_8
1	-0.45841	19.42716	310.46545	0.00000
2	0.68643	-11.48837	-160.05151	0.00000
3	-3.31322	-4.61634	549.22287	0.00000
4	-3.27040	-27.04160	-210.58835	0.00000
5	-4.66849	15.94852	-458.97169	0.00000
6	-3.32058	-10.57587	104.46982	0.00000
7	2.33490	0.67070	-24.49712	0.00000
8	-4.97177	22.49589	-217.09175	0.00000
9	-4.57901	1.33919	107.91198	0.00000
10	1.00048	-0.13621	446.99340	0.00000
11	-3.32037	-26.37463	69.94157	0.00000
12	-4.76122	18.22032	-69.95599	0.00000
13	1.05141	-28.85380	-264.63363	0.00000
14	-0.43623	-16.10003	-15.86234	0.00000
15	0.07128	-11.54899	-56.99282	0.00000
16	-2.76391	-12.69936	-113.19206	0.00000
17	-1.02548	-0.79516	-310.52274	0.00000
18	-4.60372	-7.00874	-44.45474	0.00000
19	1.67487	7.24158	-165.09234	0.00000
20	4.22423	27.48397	-72.77065	0.00000
21	2.90555	17.31374	-486.70688	0.00000
22	-1.16946	0.15826	-438.76460	0.00000
23	-3.39534	23.40222	236.13066	0.00000
24	1.63388	-19.24791	-564.52950	0.00000
25	-0.48505	-25.99182	424.16727	0.00000
26	2.89194	-14.41666	-506.07972	0.00000
27	-1.71638	12.97958	292.08257	0.00000
28	1.28075	-7.33662	-452.79065	0.00000
29	-2.75574	21.79688	-242.46736	0.00000
30	-0.83711	4.29590	95.25796	0.00000

Appendix B

CILib Configuration

The Computational Intelligence Library (CILib) used to perform all experiments in this thesis is configured using XML files. While the large number of XML files and their verbosity mean that all XML files used in this thesis cannot be presented here, the configuration of the new algorithms is given below.

The GDPSO and LFPSO specific settings are given in sections B.1 and B.2, respectively. The HGPSO is also new to the CILib, and its configuration is described in section B.3. The configuration of function shifting is discussed in section B.4, and finally, a short but complete XML file is provided in section B.5. It is important to note that while the XML samples given here are accurate at the time of writing, changes due to refactoring in the CILib project may lead to changes in the configuration XML as well.

B.1 GDPSO

The following XML snippet shows a typical configuration of the GDPSO algorithm:

```
<algorithm id="GDPSO" class="PSO.GDPSO" particles="20" stepSize="0.01"
    socialAcceleration="1.496180" cognitiveAcceleration="1.496180"
    gradientVelocityUpdateCondition="0.0">
    <topology class="Entity.Topologies.GBestTopology" />
    <addStoppingCondition class="StoppingCondition.MaximumIterations"
        iterations="10000" />
</algorithm>
```

The GDPSO class extends the standard PSO implementation, and as a result all XML attributes defined for PSO are also defined for GDPSO. Specifically, these include `id`, `particles`, `socialAcceleration` and `cognitiveAcceleration`, and their meaning is unchanged. The `class` attribute specifies the use of the relevant `PSO.GDPSO` class. The attributes that are new and specific to the GDPSO are `stepSize` and `gradientVelocityUpdateCondition`. The `stepSize` setting controls the gradient descent algorithm parameter δ, while `gradientVelocityUpdateCondition` setting controls the delayed introduction of gradient information, and is expressed as a scalar $\in [0, 1]$, as described in section 3.3.1.

The `topology` and `addStoppingCondition` elements are identical in both their syntax and their use to the standard PSO.

It is worthwhile noting that in the case of complex experimental runs involving multiple functions and per-function paramaterised algorithms, it is necessary to use unique values for the `id` attribute. Typically in these cases, the `id` of the algorithm was the concatenation of the name and algorithm parameters used, which ensured uniqueness.

B.2 LFPSO

The following XML snippet shows a typical configuration of the LFPSO algorithm:

```
<algorithm id="LFPSO" class="PSO.LFPSO" particles="20"
    socialAcceleration="1.496180" cognitiveAcceleration="1.496180"
    deltaT="0.001" gradientVelocityUpdateCondition="0.0">
    <topology class="Entity.Topologies.GBestTopology" />
    <addStoppingCondition class="StoppingCondition.MaximumIterations"
        iterations="10000" />
</algorithm>
```

The configuration of the LFPSO algorithm is similar to the GDPSO algorithm. The standard attributes `id`, `particles`, `socialAcceleration` and `cognitiveAcceleration` are again present, and their meaning is unchanged. The `class` attribute specifies the use of the LFPSO class, specifically `PSO.LFPSO`. The LFPSO algorithm differs from the GDPSO in that parameter δ from GD is no longer present, and that the LeapFrog parameter Δt is used. In the XML, this setting is referred to as `deltaT`. The `gradi-`

entVelocityUpdateCondition attribute works in an identical way as with the GDPSO above.

The topology and addStoppingCondition elements are identical in syntax and use to the standard PSO.

B.3 HGPSO

The following XML snippet shows a typical configuration of the HGPSO algorithm:

```
<algorithm id="HGPSO" class="PSO.HGPSO" particles="20"
    stepSize="0.01" socialAcceleration="1.496180"
    cognitiveAcceleration="1.496180">
    <topology class="Entity.Topologies.GBestTopology" />
    addStoppingCondition class="StoppingCondition.MaximumIterations"
        iterations="10000" />
</algorithm>
```

Since both the GDPSO and the HGPSO make use of the gradient descent algorithm, the configuration of the two algorithms is almost identical. The class attribute specifies the use of the PSO.HGPSO class, and the GD parameter δ can be configured using the stepSize parameter.

B.4 Shifting of Functions

To achieve the shifting of benchmark functions as discussed in section 4.2 and Appendix A, a function decorator was used. This decorator was termed the ShiftedFunction-Decorator, and the XML snippet below shows its usage:

```
<function class="Functions.Continuous.ShiftedFunctionDecorator"
    shiftDataFile="Function Shifts\SixHumpCamelBack.txt" >
    <function class="Functions.Continuous.SixHumpCamelBack"
        domain="R(-3,3),R(-2,2)"/>
</function>
```

The shiftDataFile attribute specifies an input file, from which the shift values for each dimension are read. A single scalar value for each dimension is stored, and the values are separated by end of line characters. The exact values used in this thesis have been given in Appendix A. The second parameter to the decorator is the function to be shifted, defined in the `function` element.

B.5 A Short but Complete CILib XML Sample

A short, complete CILib XML file is given below, showing an experimental run of the LFPSO algorithm on the Six Hump Camel Back function. It has been configured with the relevant stopping condition and measurement settings for an experimental run, as used in this thesis.

```xml
<?xml version="1.0" encoding="UTF-8"?>

<!DOCTYPE simulator [
<!ATTLIST algorithm id ID #IMPLIED>
<!ATTLIST problem id ID #IMPLIED>
<!ATTLIST measurements id ID #IMPLIED>
]>

<simulator>

    <algorithms>
        <algorithm id="LFPSO" class="PSO.LFPSO" particles="20"
            socialAcceleration="1.496180"
            cognitiveAcceleration="1.496180"
            deltaT="0.001"
            gradientVelocityUpdateCondition="0.0">
            <topology class="Entity.Topologies.GBestTopology" />
                <addStoppingCondition
                    class="StoppingCondition.MaximumIterations"
                    iterations="10000" />
        </algorithm>
    </algorithms>

    <problems>
        <problem id="SixHumpCamelBack"
            class="Problem.GradientFunctionMinimisationProblem">
            <function
                class="Functions.Continuous.ShiftedFunctionDecorator"
                    shiftDataFile="Function Shifts\SixHumpCamelBack.txt">
                <function
                    class="Functions.Continuous.SixHumpCamelBack"
                    domain="R(-3,3),R(-2,2)"/>
            </function>
        </problem>
    </problems>

    <measurements id="measurements" class="Simulator.MeasurementSuite"
        samples="50" resolution="50">
        <addMeasurement class="Measurement.FitnessEvaluations" />
        <addMeasurement class="Measurement.FunctionOptimisationError" />
        <addMeasurement class="Measurement.Diversity" />
    </measurements>

    <simulations>
```

```
        <simulation>
            <algorithm idref="LFPSO"/>
            <problem idref="SixHumpCamelBack"/>
            <measurements idref="measurements" file="measurements.txt"/>
        </simulation>
    </simulations>
</simulator>
```