Derating NichePSO

by

Clive Naicker

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

September 2006

An electronic, hyperlinked version of this work is available online at:
`http://cirg.cs.up.ac.za/`

Derating NichePSO

by

Clive Naicker

Abstract

The search for multiple solutions is applicable to many fields (Engineering [54][67], Science [75][80][79][84][86], Economics [13][59], and others [51]). Multiple solutions allow for human judgement to select the best solution from a group of solutions that best match the search criteria. Finding multiple solutions to an optimisation problem has shown to be difficult to solve. Evolutionary computation (EC) and more recently Particle Swarm Optimisation (PSO) algorithms have been used in this field to locate and maintain multiple solutions with fair success. This thesis develops and empirically analyses a new method to find multiple solutions within a convoluted search space. The method is a hybrid of the NichePSO [14] and the sequential niche technique (SNT)[8]. The original SNT was developed using a Genetic Algorithm (GA). It included restrictions such as knowing or approximating the number of solutions that exist. A further pitfall of the SNT is that it introduces false optima after modifying the search space, thereby reducing the accuracy of the solutions. However, this can be resolved with a local search in the unmodified search space. Other sequential niching algorithms require that the search be repeated sequentially until all solutions are found without considering what was learned in previous iterations, resulting in a blind and wasteful search. The NichePSO has shown to be more accurate than GA based algorithms [14][15]. It does not require knowledge of the number of solutions in the search space prior to the search process. However, the NichePSO does not scale well for problems with many optima [16]. The method developed in this thesis, referred to as the derating NichePSO, combines SNT with the NichePSO. The main objective of the derating NichePSO is to eliminate the inaccuracy of SNT and to improve the scalability of the NichePSO. The derating

NichePSO is compared to the NichePSO, deterministic crowding [23] and the original SNT using various multimodal functions. The performance of the derating NichePSO is analysed and it is shown that the derating NichePSO is more accurate than SNT and more scalable than the NichePSO.

Keywords: multimodal, multiple, parallel, sequential, derating function

Supervisor : Prof. A. P. Engelbrecht
Department : Department of Computer Science
Degree : Magister Scientiae

Acknowledgements

I would like to express my sincere thanks to the following people for their assistance during the production of this thesis:

Prof. A. P. Engelbrecht, thank you for your patience, inspiration and motivation.
The CIlib team (`http://cirg.cs.up.ac.za/`), thank you for creating a smart set of tools.
The teams of LaTex, Wikipedia, Mathworld and Google.

To my loving Parents and two Brothers,

my close companion Yun Li,

and my dearest friend Kreesen G.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Every day life is filled with small and large goals. For example, an athlete will train every day to win a gold medal or to simply beat another competitor. Another example of a goal, is a mechanical engineer that adjusts an engine to run at its peak performance with the least possible frictional damage to the pistons of the engine. A goal of an aeronautical designer would be to improve the aerodynamics of an aeroplane thereby reducing the drag experienced in flight. In retrospect, a goal can be described as an objective or an optimum to be achieved. Problems for which an optimum to an objective is to be found, are referred to as optimisation problems.

Several methods have been developed to solve optimisation problems for systems that are described by mathematical models. The goal of an optimisation technique is to find the minimum or maximum of a function. Minimisation involves finding the lowest value of f and maximisation involves searching for the highest value of f. The optimum of a continuous function, $f(\mathbf{x})$, is defined as the point where $f'(\mathbf{x}) = 0$, where $\mathbf{x}$ is a point in the search space.

Classical numerical optimisation methods can be divided into derivative and derivative-free based methods. Derivative-based methods, such as gradient descent [65], were developed to optimise continuous functions. These derivative-based numerical methods are, however, dependent on the existence of f'. That is, f' must be known prior to the search process. Non-derivative based methods, such as the golden ratio search [65] and the Nelder-Mead method [68], are not dependent on

f'. Both derivative and derivative-free based methods are prone to errors, such as inaccuracy, incorrectly identifying the optimum, or the method may get trapped in a local optimum if more than one optimum exists in the search space. Furthermore, many real-world problems do not have a continuous objective function, in which case derivative based methods cannot be applied.

In the Computational Intelligence (CI) field, many optimisation methods have been developed, which address the above mentioned problems. These optimisation methods do not necessarily require derivative information, and usually implement mechanisms to escape local optima. This thesis focuses on two paradigms of CI, namely evolutionary computation (EC) and swarm intelligence (SI).

EC represents a group of a variety of computational models that use evolutionary processes to solve mathematical problems [89]. These computational models are often referred to as evolutionary algorithms (EAs). In essence, all EAs share the common principle of survival of the fittest [19]. An example of an EA is the genetic algorithm (GA). The GA is premised on the natural evolution of a population of individuals modelled traditionally as a string of numbers. The GA is characterised by randomly applying recombination operators as well as selection and mutation [6]. Each individual represents a position in the search space, and all individuals are evolved to optimise the function which describes the search.

SI is the study of collective behaviour in decentralised, self-organising systems. Such systems consist of a population of agents that interact with each other, which leads to a global behaviour of the swarm. Examples of natural swarm systems include, ant colonies, bird flocking, bacteria molding, and fish schooling [55]. The most popular computational models of swarm behaviour are ant algorithms [60] and particle swarm optimisation (PSO) methods [50][24]. This thesis focusses on PSO. PSO is a stochastic, population-based optimisation technique where a swarm of particles moves through an n-dimensional search space with the goal of converging on an optimum of the function to be optimised. PSO is modelled after simulation studies of bird flocks revealed that birds kept the formation of the flock even after a sudden change in direction. ACO is also a stochastic, population-based optimisation approach, developed to solve discrete, combinatorial optimisation problems. ACO

models the foraging behaviour of social ants [60].

CI algorithms have been successfully applied to locate the optimum in many different problem domains, including finance [13][59], engineering [54][67], medicine [79][84][86] and others [51]. Developing new search methods and enhancing current algorithms are essential to improving scalability, accuracy and efficiency such that difficult problem domains can be explored. This thesis focuses on problem domains with multiple optima and the objective is to discover all optima that exist in the search space. The remainder of this chapter presents the objectives, motivation and outline for the rest of the thesis.

1.1 Objectives

The objective of this thesis is to develop and evaluate a new PSO algorithm for locating and maintaining multiple optima to the same optimisation problem. Such algorithms are referred to as niching algorithms. Derating NichePSO is a new niching algorithm that is a hybrid of the sequential niching technique (SNT) [8] (refer to Chapter 3) and the NichePSO [14] (refer to Chapter 4). Further sub-objectives include a literature overview of state-of-the-art niching algorithms in EC and PSO in-order to identify problems with current techniques (in particular the NichePSO and SNT).

Brits [14] has shown that the number of particles for the NichePSO increases exponentially as the number of dimensions of the problem space increases. It is discovered that this behaviour results from the possibility that particles may unnecessarily converge on optima that have already been discovered, which reduces the ability of the NichePSO to explore the search space and therefore a larger number of particles are required. To resolve this dilemma the NichePSO is combined with the SNT. That is, the NichePSO is executed several times and in each execution the solutions found in previous executions are removed from the search space. This is achieved by modifying the search space near solutions found by subswarms and thus preventing the surrounding search area from being re-explored unnecessarily. However, as with SNT, the modifications to the search space introduces false local

optima, which are incorrectly identified as local optima [8]. The derating NichePSO minimises these errors by refining and maintaining solutions in the unmodified search space so that subswarms may converge on valid solutions.

The success of the derating NichePSO to overcome the drawbacks of the NichePSO and SNT are empirically demonstrated by comparing their scalability and accuracy. The influence of several parameters on the derating NichePSO, including the number of particles, iterations, executions, and niche radius are investigated. Finally, it is concluded that the derating NichePSO is more scalable than the NichePSO and overcomes the inaccuracy problems of the SNT.

1.2 Motivation

When people are confronted with a problem, they often seek to find more than one solution so as to compare each one and then choose the solution that best suites their situation. A simple example would be a newly wedded couple searching for a home. Natural instinct would tell the couple what they like, and common sense would tell the couple not to select the first house they find. The couple should rather explore the market and find all the houses that are of interest to them and then select the best buy.

The search for multiple solutions is applicable in many fields of study, in particular Engineering, Science and Economics. For example, an electrical engineer may be interested in the location of resonance points in a mechanical or electrical system above a particular threshold to reduce or enhance the signal [22]. Other applications that produce multiple solutions include the search for the sequence of all nucleotide base-pairs in a DNA molecule [75], finger print recognition [80], calculation of asteroid orbits [4], and stock market speculation [87]. Lastly, performing a search using a search engine such as Google (`http://www.google.com`) returns multiple results that best match the search criteria.

CI algorithms, such as GA and PSO, were designed to find only a single solution. To aid in finding multiple solutions, Mahfoud *et al* [62], De Jong *et al* [48], and Beasely *et al* [8] researched and extended the GA to find multiple optima. These

extended algorithms are generally referred to as niching algorithms, or speciation methods. Niching approaches can be divided into parallel and sequential techniques. Sequential niching methods find multiple solutions through repetitive application of the optimisation algorithm, while parallel niching methods use a single application of an optimisation algorithm to find as many optima as possible at the completion of the algorithm. Examples of parallel niching methods include crowding [48] [61], deterministic crowding (DC) [61], fitness sharing GA [36], multinational GA [93], niche clustering [32], restricted tournament selection [41] and island models [10]. Examples of sequential algorithms include the sequential niche technique (SNT) [8], objective function stretching [70], and the deflection technique [72].

Recently, Particle Swarm Optimisation (PSO) approaches have been developed to find multiple solutions. Brits *et al* [14] [16] [15] developed the *nbest* PSO and NichePSO as swarm based niching techniques. The *nbest* PSO algorithm was developed specifically to solve systems of unconstrained equations, however it can also be applied to general multimodal optimisation problems. A drawback of the *nbest* PSO algorithm is that it cannot maintain local optima [14]. The NichePSO overcame this disadvantage and has shown to be more efficient and scalable than GA based algorithms (specifically DC and SNT) [14]; which is the motivation for improving the NichePSO's performance. Other PSO based niching algorithms include the vector-based PSO [82], parrallel vector-based PSO [83], objective function stretching [70][73] and PSO with speciation [69].

The contributions of this thesis include:

- A new PSO niching method, referred to as the derating NichePSO which combines the advantages of the NichePSO and SNT.
- A formula is empirically derived to be used as a heuristic to determine the number of particles needed to locate a required number of solutions.

1.3 Outline of Thesis

The thesis is organised as follows:

- **Chapter 2** defines important concepts, including the optimisation problem, minimisation, maximisation, local and global optima, unimodal, multimodal and multi-objective optimisation (MOO). The differences and similarities between MOO and multi-modal optimisation are pointed out as well. Popular derivative and non-derivative based numerical methods used to find the local optimum of a function are discussed. A review of the standard GA and PSO algorithms are presented to serve as a platform for the remainder of the thesis. The chapter is concluded with the argument that the standard GA and PSO algorithms are not suitable for niching, and need to be adapted to enable niching

- **Chapter 3** covers various GA based niching techniques, including SNT, dynamic niche sharing, crowding, and deterministic crowding. Much focus is given to the details of how the SNT algorithm searches for multiple solutions. The pitfalls and benefits of the SNT are listed and discussed in depth.

- **Chapter 4** reviews various PSO algorithms that have been developed to locate multiple solutions, including objective function stretching, vector-based PSO, parallel vector-based PSO, *nbest* PSO, species-based PSO, and NichePSO. The pitfalls and benefits of the NichePSO are listed and discussed in detail.

- **Chapter 5** introduces the derating NichePSO. The focus of this chapter is to discuss the development of the derating NichePSO. In this discussion, it is shown how the derating NichePSO addresses the false local optima problem of the SNT and the scalability problem of NichePSO.

- **Chapter 6** evaluates DC, SNT, NichePSO and derating NichePSO on various benchmarks. The evaluation examines the scalability and accuracy of the derating NichePSO and a detailed comparison is made amongst the algorithms. It is empirically shown that the derating NichePSO has similar accuracy and is more scalable than the NichePSO. The sensitivity of the derating NichePSO towards fluctuations in the niche radius, number of particles, and the number of executions is analysed. A formula is derived for the relationship between the

number of particles, the number of executions, and the number of solutions in the search space, to serve as a guideline for selecting the swarm size.

- **Chapter 7** concludes the dissertation and summarises avenues for further research.

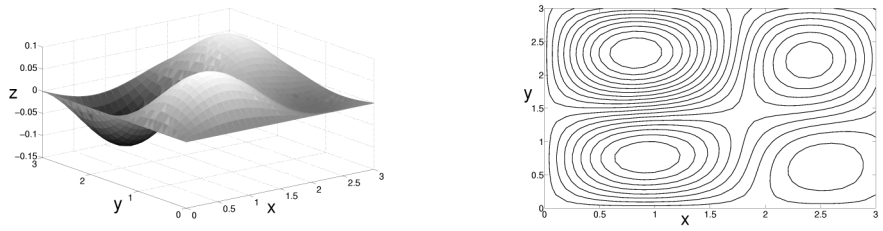
Chapter 2

Background

What is a niche? Where are niches found in the real world? How are niches found if they exist? Why are niches important? What is the relationship between niches and optima? This chapter answers these questions and defines useful niching concepts. Several numerical methods that are used to find the local optimum of a function are discussed. These methods include minimisation using derivatives (section 2.2.1), the steepest descent or gradient method (section 2.2.2), the golden ratio search (section 2.3.1), and the Nelder-Mead method (section 2.3.2). A problem with these methods is that they are not suitable for functions with more than one optimum as they are susceptible to local optima. Algorithms to optimise multimodal functions are then discussed. These algorithms include the genetic algorithms in section 2.4.1 and particle swarm optimisation in section 2.4.2. However, these algorithms, in their standard form, are not applicable to multiple solutions as discussed in section 2.5. The differences between multimodal and multi-objective functions are discussed in section 2.6. The chapter is concluded in section 2.7 which states the focus of this research and provides a short prelude to the following chapters.

2.1 Introduction

In terms of computational terminology, optimisation refers to the process of finding the location in a function's domain where the function has its maximum or minimum



(a) (b)

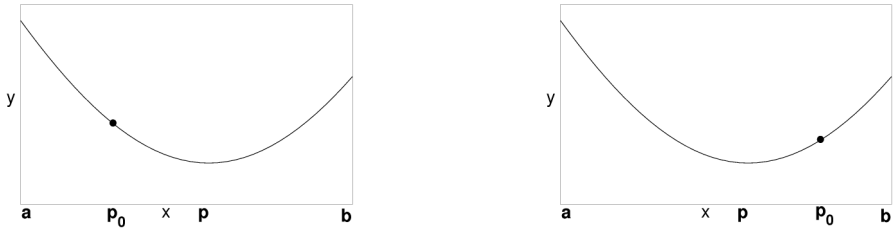
Figure 2.1: (a) The surface plot and (b) contour plot of the function defined by equation (2.1)

value. These values are referred to as the function's optima. A function with a single optimum is referred to as unimodal and a function with multiple optima is referred to as multimodal. For multimodal functions, optima can be local or global. A local minimum is defined as the point $\mathbf{p}$, if there exists a point $\mathbf{p}$ in an open interval I^n, where n is the dimension of the function f, such that $f(\mathbf{p}) \leq f(\mathbf{x})$ for all $\mathbf{x} \in I^n$, and I^n is a subset of the function's domain. If $\mathbf{p}$ does not exist, then the function has no local minimum. Similarly, the local maximum of a function is defined as the point $\mathbf{p}$, if there exists a point $\mathbf{p}$ in an open interval I^n such that $f(\mathbf{x}) \leq f(\mathbf{p})$ for all $\mathbf{x} \in I^n$. The function f is said to have a *local extremum* or *local optimum* if it has either a local maximum or local minimum value at $\mathbf{p}$. The global minimum of a function is similar to the local minimum, except that the interval I^n is defined as the entire domain of the function; that is, $\mathbf{p}$ is the global minimum if $f(\mathbf{p}) \leq f(\mathbf{x})$ for all $\mathbf{x}$ (similarly for the global maximum).

To illustrate local maxima and minima, consider figure 2.1 which is modelled by the two dimensional wave equation,

$$f(x_1, x_2) = 0.02\sin(x_1)\sin(x_2) - 0.03\sin(2x_1)\sin(x_2) + 0.04\sin(x_1)\sin(2x_2) + 0.08\sin(2x_1)\sin(2x_2) \quad (2.1)$$

Figures 2.1 (a) and (b), show that the function has a local maximum, and a global maximum, as well as a local minimum, and a global minimum. Numerical methods can be used to approximate that the local minimum is located at $f(2.5351, 0.6298) = -0.0264$ and the global minimum is located at $f(0.8278, 2.3322) = -0.1200$. The



(a) (b)

Figure 2.2: A unimodal function, f. Using $f'(x)$ to find the minimum value of the unimodal function $f(x)$ on the interval $[a, b]$ (a) If $f'(p_0) < 0$ then p lies in $[p_0, b]$ (b) If $f'(p_0) > 0$ then p lies in $[a, p_0]$

local maximum can be approximated at $f(2.3979, 2.2287) = 0.0853$ and the global maximum at $f(0.9241, 0.7640) = 0.0998$ [65]. The following sections briefly discuss a few numerical methods that are available to find the minimum of unimodal functions (the theorems and definitions have been adapted from [65]).

2.2 Derivative-based Methods

If the derivative of a one dimensional function $f(x)$ is available, then solving $f'(x) = 0$ will find all optima of the function. In some instances solving $f'(x) = 0$ is computationally complex. If this is the case, then the following search methods can be used to find the optima of a one dimensional function.

2.2.1 Minimisation Using Derivatives

This section considers a derivative method specifically for one dimensional functions. Assume that $f(x)$ is a one dimensional function that is defined over $[a, b]$, and $f'(x)$ is unimodal and defined at all points in $[a, b]$. This means that $f(x)$ has a unique minimum at point p. This method sequentially finds better approximations to p until the desired accuracy is achieved, that is $f'(p) \approx 0$.

Define p_0 as a starting point to lie in the interval (a, b). There are three possibilities [65]:

1. If $f'(p_0) < 0$ then p lies to the right of p_0 in $[p_0, b]$ (refer to figure 2.2 (a)).
2. If $f'(p_0) > 0$ then p lies to the left of p_0 in $[a, p_0]$ (refer to figure 2.2 (b)).
3. $f'(p_0) = 0$ then the optimum has been found, and $p = p_0$

Now that the direction in which p is relative to p_0 is known, the next step is to find p_1 and p_2. The values p_1 and p_2 define a new smaller interval in which p lies. The values of p_1 and p_2 are defined as:

1. $p_1 = p_0 + h$
2. $p_2 = p_0 + 2h$

If $f'(p_0) < 0$ then p lies to the right of p_0 and h should be positive. If $f'(p_0) > 0$ then p lies to the left of p_0 and h should be negative. The value of h must be chosen so that p_1 and p_2 satisfy the following criterion:

$$f(p_0) > f(p_1) \text{ and } f(p_1) < f(p_2) \tag{2.2}$$

The next step is to determine $p_{min} = p_0 + h_{min}$ such that p_{min} is a better approximation to p than p_0. The calculation of h_{min} is achieved using the Lagrange polynomial based on p_0, p_1 and p_2 [65]:

$$Q(x) = \frac{y_0(x - p_1)}{2h^2} - \frac{y_1(x - p_0)(x - p_2)}{h^2} + \frac{y_2(x - p_0)(x - p_1)}{2h^2} \tag{2.3}$$

where $y_0 = f(p_0), y_1 = f(p_1)$ and $y_2 = f(p_2)$.

$$Q'(x) = \frac{y_0(2x - p_1 - p_2)}{2h^2} - \frac{y_1(2x - p_0 - p_2)}{h^2} + \frac{y_2(2x - p_0 - p_1)}{2h^2} \tag{2.4}$$

Solving for $Q'(p_0 + h_{min}) = 0$ yields

$$0 = \left(\frac{y_0(2(p_0 + h_{min}) - p_1 - p_2)}{2h^2}\right) - \left(\frac{y_1(2(p_0 + h_{min}) - p_0 - p_2}{h^2}\right) + \left(\frac{y_2(2(p_0 + h_{min}) - p_0 - p_1)}{2h^2}\right) \tag{2.5}$$

Solving for h_{min} gives:

$$-h_{min}(2y_0-4y_1+2y_2) = y_0(2p_0-p_1-p_2)-y_1(4p_0-2p_0-2p_2)+y_2(2p_0-p_0-p_1) \quad (2.6)$$

$$h_{min} = \frac{h(4y_1 - 3y_0 - y_2)}{4y_1 - 2y_0 - 2y_2} \quad (2.7)$$

The value of $p_{min} = p_0 + h_{min}$ is a better approximation to p than p_0 (this is based on Lagrange properties and is out of the scope of this thesis). To determine a closer approximation to p, set $p_0 = p_{min}$ and recalculate p_1 and p_2 to find a new h_{min} and p_{min}. The process continues to find better approximations of p until the desired accuracy is achieved.

Finding the minima of a function using derivatives is a complex task for high dimensional search spaces, and the global minimum of the function is not guaranteed to be found if the function is multimodal [65].

2.2.2 Steepest Descent or Gradient Method

The steepest descent (or gradient method) [3] can be used to find the minimum of a function with n variables. Define an n-dimensional function as $f(\mathbf{x})$, where $\mathbf{x} = (x_1, x_2, \ldots, x_n)$. The gradient of f is then defined as a vector function:

$$\nabla f(\mathbf{x}) = (f_1, f_2, \ldots, f_n) \quad (2.8)$$

where the partial derivatives $f_k = \frac{\partial f}{\partial x_k}$ are evaluated at point $\mathbf{x}$. The vector $\nabla f(\mathbf{x})$ points locally in the direction of the greatest rate of increase of $f(\mathbf{x})$. Therefore, $-\nabla f(\mathbf{x})$ points locally in the direction of the greatest rate of decrease. Initially, a starting point $\mathbf{p}_0$ is assumed to exist in the domain. The search then continues along the line (or hyper-dimensional plane for $n > 1$) through $\mathbf{p}_0$, in the direction $\mathbf{s}_0 = -\nabla f / \|\nabla f\|$, where $\|\nabla f\|$ is the length of the vector ∇f. Point $\mathbf{p}_1$ is found where a local minimum occurs when the point $\mathbf{x}$ is constrained to lie on the line (or plane) $\mathbf{x} = \mathbf{p}_0 + t\mathbf{s}_0$. The process can be repeated to recalculate $\nabla f(\mathbf{p}_1)$ and a new search direction $\mathbf{s}_1 = -\nabla f / \|\nabla f\|$ to find a new point $\mathbf{p}_2$ which is constrained to lie on $\mathbf{x} = \mathbf{p}_1 + t\mathbf{s}_1$. A sequence of points $\{\mathbf{p}_k\}$ can be defined with property

$f(\mathbf{p}_0) > f(\mathbf{p}_1) > \ldots > f(\mathbf{p}_k) > \ldots$ Then, the limit $\lim_{k\to\infty} \mathbf{p}_k = \mathbf{p}$ and $f(\mathbf{p})$ will be a local minimum for $f(\mathbf{x})$ [65].

The gradient method is strongly dependent on the starting point, $\mathbf{p}_0$. If the function has a long, narrow valley structure and $\mathbf{p}_0$ is not close to the minimum, then many iterations are required for the minimum to be found [3]. The algorithm is guaranteed to converge on a local minimum, however, there is no guarantee that global minimum will be found [25].

2.3 Derivative-free Methods

The calculation of the derivative of a function is not always a feasible option because of the computational complexity, or in some instances the derivative does not even exist. In these cases, search methods that are not dependent on the derivative of a function need to be employed. To find an optimum usually requires many evaluations of the objective function. To reduce computation time, it is important to limit the number of function evaluations. This section discusses two non-derivative based methods that reduce the number of function evaluations: the golden ratio search and the Nelder-Mead method.

2.3.1 The Golden Ratio Search

In this discussion the author is assuming the minimisation of a one dimensional function. A precondition to using the golden ratio search is that the function $f(x)$ must be unimodal. This allows an interval to be defined such that the minimum point, p, is contained within the new interval. The golden ratio method reduces the size of the interval while still ensuring that p is in the interval. The method terminates once the interval is sufficiently small to identify p. For example, let the original interval be $[0, 1]$ and define r as, $0.5 < r < 1$ then $0 < 1 - r < 0.5$. This creates three sub-intervals: $[0, 1 - r], (1 - r, r), [r, 1]$. There are two possible actions that can be taken next:

1. Reduce the interval, $[0, 1]$, from the right to create a new interval $[0, r]$ (refer

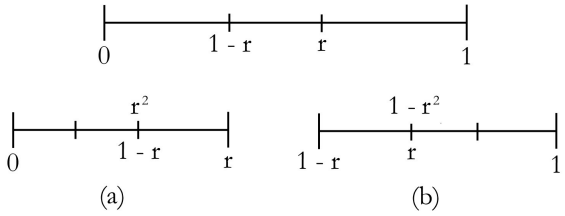


Figure 2.3: Illustration of Golden Ratio Search (a) Reduce the interval from the right and the new interval is $[0, r]$ (b) Reduce the interval from the left and the new interval is $[1 - r, 1]$

to figure 2.3(a)).

2. Reduce the interval, $[0, 1]$ from the left to create a new interval $[1 - r, 1]$ (refer to figure 2.3(b)).

A decision process (described below) is used to determine which action to take. The value of r is chosen so that the ratio $(1 - r) : r$ is the same as the ratio $r : 1$. Therefore, r can be calculated as:

$$1 - r = r^2 \tag{2.9}$$

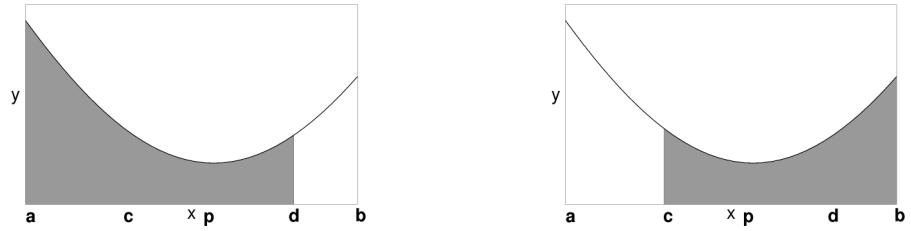
$$\Rightarrow \quad r^2 + r - 1 = 0 \tag{2.10}$$

$$\Rightarrow \quad r = (\sqrt{5} - 1)/2 \tag{2.11}$$

To decide which action to take, define two interior points, c and d, in the interval $[a, b]$, where $a = 0$ and $b = 1$, as

1. $c = a + (1 - r)(b - a)$
2. $d = a + r(b - a)$

This results in $a < b < c < d$. $f(c)$ and $f(d)$ are guaranteed to be less than $\max\{f(a), f(b)\}$ because $f(x)$ is a unimodal function. There are now two possibilities:



(a) (b)

Figure 2.4: Illustration of Golden Ration Search (a) If $f(c) \leq f(d)$ then reduce from the right and use $[a, d]$ (b) If $f(d) < f(c)$ then reduce from the left and use $[c, b]$.

1. $f(c) \leq f(d)$: the minimum must occur in the subinterval $[a, d]$. Set $b = d$ to create a new interval and continue the search in the new interval by recalculating r, c and d; that is, reduce the interval from the right (refer to figure 2.4 (a)).

2. $f(d) < f(c)$: the minimum must occur in the subinterval $[c, b]$. Set $a = c$ to create a new interval and continue the search in the new interval by recalculating r, c and d; that is, reduce the interval from the left (refer to figure 2.4 (b)).

The search is terminated once the interval width is sufficiently small to estimate the location of the local minimum, p, and $f(a) \approx f(b)$.

The golden ratio search experiences inaccuracy problems where the function is flat near the minimum as $f(a) \approx f(b)$, but the interval width is still significantly large [65]. Many function evaluations will be required if the interval is large, which makes the method computationally expensive. Furthermore, the golden ratio search does not extend well for higher dimensional problems [65]. A search method that overcomes these drawbacks is the Nelder-Mead method, discussed in the next section.

2.3.2 The Nelder-Mead Method

Nelder and Mead developed a simplex method to find the local minimum of a function that has several variables [68]. For a two variable problem, a simplex is a

triangle [96]. The method compares the function values at the vertices of the triangle. The worst vertex, defined as the vertex where the function value is the largest, is replaced with a new vertex (discussed below) and a new triangle is formed. As the search progresses, a sequence of triangles are generated and the function values of the vertices decrease. The coordinates of the minimum point are found once the size of the triangle has reduced sufficiently [65].

For a two variable problem, the simplex is a triangle defined by vertices $\mathbf{b} = (x_1, y_1)$, $\mathbf{g} = (x_2, y_2)$, and $\mathbf{w} = (x_3, y_3)$ ($\mathbf{b}$ is the best vertex, $\mathbf{g}$ is good or the second best vertex, and $\mathbf{w}$ is the worst vertex). To find the new vertex that replaces $\mathbf{w}$ the midpoint of the line $\overline{\mathbf{bg}}$ must first be calculated as,

$$\mathbf{m} = \frac{\mathbf{b} + \mathbf{g}}{2} = \left(\frac{x_1 + x_2}{2}, \frac{y_1 + y_2}{2} \right) \tag{2.12}$$

There are four options when calculating the new vertex: reflection, expansion, contraction and shrinking. Each option is tried in sequence. The current iteration is terminated when an option produces a new vector that has a function value smaller than $\mathbf{w}$.

Reflection

The function decreases along the lines $\overline{\mathbf{wg}}$ and $\overline{\mathbf{wb}}$. Thus, it is presumed that smaller values lie away from $\mathbf{w}$. A test point $\mathbf{r}$ is obtained by reflecting the triangle through the line segment $\overline{\mathbf{bg}}$. $\mathbf{r}$ is calculated as,

$$\mathbf{r} = \mathbf{m} + (\mathbf{m} - \mathbf{w}) = 2\mathbf{m} - \mathbf{w} \tag{2.13}$$

The new triangle formed is $\mathbf{bgr}$, illustrated in figure 2.5(a).

Expansion

If reflection produces a new vertex, $\mathbf{r}$, that is better than $\mathbf{b}$, then it is assumed that the function may continue to decrease farther than $\mathbf{r}$. The new vertex, $\mathbf{e}$, is calculated as,

$$\mathbf{e} = \mathbf{r} + (\mathbf{r} - \mathbf{m}) = 2\mathbf{r} - \mathbf{m} \tag{2.14}$$

The new triangle formed is **bge**, illustrated in figure 2.5(b).

Contraction

If reflection and expansion do not produce a better vertex than **b**, then two more vertices are defined, $\mathbf{c_1}$ and $\mathbf{c_2}$. $\mathbf{c_1}$ is the midpoint of line segment $\overline{\mathbf{wm}}$ and $\mathbf{c}_2$ is the midpoint of line segment $\overline{\mathbf{mr}}$. The vertex with the smallest function value (comparing $\mathbf{c_1}$ and $\mathbf{c_2}$) is called **c**. This forms the new triangle **bgc**, illustrated in figure 2.5(c).

Shrink Towards b

The final option is to calculate the midpoint **s**, between vertex **b** and **w** to form the new triangle **bsm**, illustrated in figure 2.5(d).

The above four options are tried in sequence for several iterations until the triangle formed is small enough to approximate the minimum.

The Nelder-Mead method experiences difficulty with four and higher dimensional functions having a long, curved, valley with steep sides [65]. It has been shown in [68] that this method has the problem of converging on a point other than the global minimum. This means that there is no guarantee that the global minimum will be found in a multimodal function as the search process is likely to prematurely converge on a local minimum.

2.4 Finding the Global Optimum

The numerical methods discussed above do not work well to find the global optimum if the function is multimodal. In most cases, the method assumes that the function is unimodal. A number of algorithms have been designed to locate the global optimum in an n-dimensional search space with multiple optima [65]. This thesis considers only algorithms from the field of computational intelligence, and more specifically, evolutionary computation (EC) and swarm intelligence (SI). From the EC paradigm

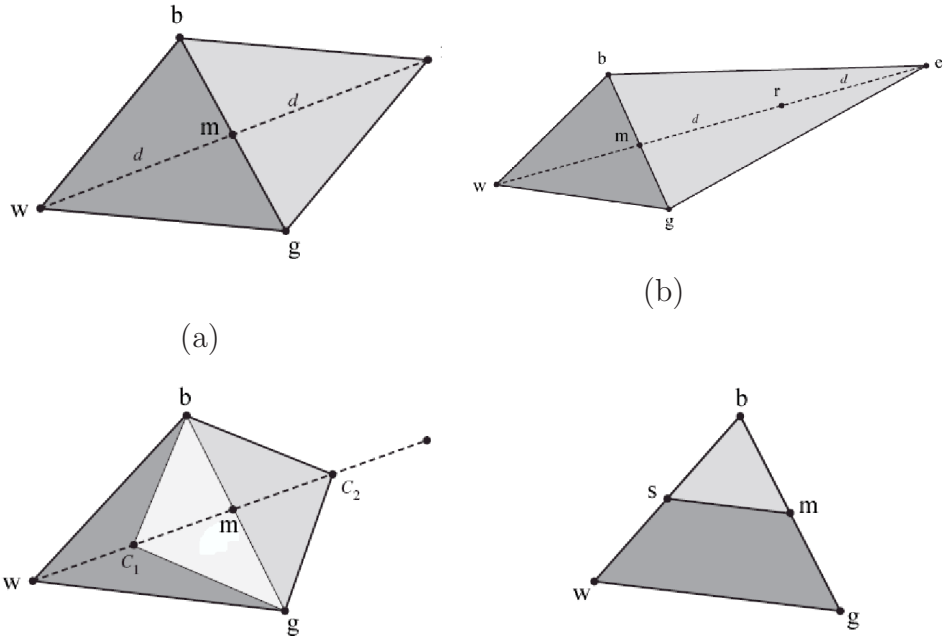


Figure 2.5: Options for calculating the new vertex in the Nelder-Mead method: (a) reflection; (b) expansion; (c) contraction; (d) shrinking [65].

only genetic algorithms are considered (section 2.4.1), and from the SI paradigm only particle swarm optimisation is considered (section 2.4.2). These algorithms do not guarantee that the global optimum will be found, but have the advantage of considering multiple points concurrently and therefore providing better exploration of the search space.

2.4.1 Genetic Algorithm

Evolutionary computation (EC) refers to any algorithm based on the concepts of evolution in nature. An algorithm within the EC paradigm is referred to as an evolutionary algorithm (EA). There are various classes of EAs, which differ in implementation but share the common principle of survival of the fittest. EA classes include: genetic algorithms [42], genetic programming [52][28], evolution strategies [5], differential evolution [74], cultural evolution [77][76], and co-evolution [1][2]. This section focuses on genetic algorithms (GAs).

Evolution is the natural ability of species to master the changing environment in which they live. It allows them to effectively compete with other species that may live in the same environment. At a biological level, the functioning and structure of an organism is determined by the genetic sequence of its deoxyribonucleic acid (DNA) which is a nucleic acid. A chromosome is a very long continuous piece of DNA, which contains many genes [56]. Genes encode proteins and other molecules that determine the structure and functioning of the organism. Altering the structure of genes in an organism changes the organism's biological structure and functioning [56]. There are two main types of cells: prokaryotic cells and eukaryotic cells. The difference between the two are their internal structure and nucleus. Each eukaryotic cell contains a single chromosome. A new cell arises when one cell divides or when two cells, like a sperm and egg cell, combine to share their DNA [56].

John Holland proposed the genetic algorithm (GA) in the 1960s [42] which is an attempt to algorithmically model genetic evolution. The model represents an individual (or chromosome) as a bit string, where each bit represents a single gene. The selection of parents to produce new chromosome(s) is determined by a method known as proportional selection; meaning that the probability of an individual to

1. Let generation $g = 0$
2. Initialise the population $\mathcal{C}_g$
3. While not converged:
   - **a.** $g = g + 1$
   - **b.** Evaluate the fitness of each individual $\mathcal{C}_{g,i} \in \mathcal{C}_g$
   - **c.** Select parents from $\mathcal{C}_{g-1}$
   - **d.** Recombine selected parents through crossover to form a set of offspring $\mathcal{O}_g$
   - **e.** Mutate offspring in $\mathcal{O}_g$
   - **f.** Select the new generation $\mathcal{C}_g$ from the previous generation $\mathcal{C}_{g-1}$ and offspring $\mathcal{O}_g$

Figure 2.6: Genetic algorithm

be selected as a parent is proportional to the fitness of the individual. A crossover operator is used to construct the new chromosome(s).

The basic GA model can be split into two different phases: selection and reproduction (or crossover). The original GA model has evolved and different representation schemes have since been used for chromosomes, various selection and reproduction methods have been developed, and a new operator known as mutation has been introduced.

The basic GA as it is used today is summarised in figure 2.6 [25]. The algorithm is iterated several times until any of the convergence criteria is satisfied. These criteria include:

1. An individual is found with an acceptable fitness,
2. the maximum number of generations has been reached, or
3. successive iterations do not produce better results.

The following subsections discuss aspects of this algorithm in more detail.

Chromosome Representation

The chromosome representation at the biological level is far too complex to be simulated in a computer environment at this time. The chromosome can be abstracted to any number of representations, depending on the characteristics of the search space and problem to be solved. For binary-valued search spaces, chromosomes are represented as a bit string containing n variables, where each variable represents a dimension in an n-dimensional search space. For a nominal-valued search space with a total of 2^D nominal values and n dimensions, each nominal value can be represented with a bit string of length D, therefore the chromosome can be represented as an $n \times D$-dimensional bit string vector. For a restricted continuous-valued search space, it is possible to use a representation similar to the nominal-valued search space or the chromosome can be represented with a vector of integer or floating-point values [45][20][25][85]. In essence, a chromosome can represent any value in the search space, provided an appropriate representation can be found.

Chromosome Fitness

Each chromosome is a vector that represents a possible solution to an optimisation problem. A fitness function is used to determine how good the solution is; that is, how close to optimal the solution is. The fitness function, F, maps the n-dimensional chromosome, $\mathbf{x}_i$, to a scalar value [25]

$$F : \mathbf{x}_i \rightarrow \Re \tag{2.15}$$

Selection

With a population of individuals distributed throughout the search space, individuals are selected for reproduction to generate new individuals. Reproduction allows the genetic material of individuals to be combined in the hope of producing new individuals that are better than their parents. A number of methods for selection have been developed, for example,

- **Random selection**: With the random selection method, a set of two indi-

viduals are randomly selected from the population. Fitness is not considered in this selection method, thus the selection method does not favour any individual.

- **Proportional selection**: This selection method favours the more fit individuals in the population. That is, the probability of selecting an individual for reproduction is directly proportional to the fitness of the individual. Roulette wheel selection, summarised in figure 2.7, is an example of proportional selection. The roulette wheel selection method may cause a few fit individuals in the population to dominate the selection procedure, which is referred to as high selection pressure. If this domination occurs early in the search process, the algorithm may prematurely converge to a poor solution.

- **Tournament selection**: This method involves the selection of a random sample of individuals from the current population. Each individual in the sample then competes with the other individuals of that sample. Individuals compete by comparing their fitness values. The individual with the best fitness value wins the tournament and is selected for the reproduction phase. Tournament selection offers the advantage that the worst individual in the population will only have a small probability of being selected for reproduction. If the sample does not include many individuals then the reproduction process will not be dominated by the fittest individuals. Tournament selection has a lower selection pressure than roulette wheel selection, which produces a more diverse population, and enhances the exploration of the search space.

Crossover

For the GA model, the crossover operator produces two offspring from two parents that are selected using a selection operator. The offspring are produced by copying values from the chromosome vector of both parents. There are a number of ways to achieve this. Three such methods are uniform, one-point crossover and two-point crossover.

1. Let $i = 1$, where i denotes the chromosome index
2. $sum = Prob(\mathbf{x}_i) = \frac{\mathcal{F}(\mathbf{x}_i)}{\sum_{i=1}^{P} \mathcal{F}(\mathbf{x}_i)}$, where $\mathcal{F}(\mathbf{x}_i)$ is the fitness of individual $\mathbf{x}_i$ and P is the size of the population.
3. Choose a uniform random number, $\xi \sim N(0, 1)$
4. While $sum < \xi$
   a. $i = i + 1$
   b. $sum = sum + Prob(\mathbf{x}_i)$
5. return $\mathbf{x}_i$ as the selected individual

Figure 2.7: Roulette wheel

- **Uniform crossover**: For the first offspring, the value of the chromosome vector at position j is determined by selecting the corresponding value from either the first parent or the second parent. That is, the value of the offspring at position j is randomly determined to be either the value of the first or second parent at position j. The second offspring is determined by selecting the chromosome value at position j from the alternate parent that the first offspring selected. That is, if the first offspring randomly selected the first parent for the chromosome value at position j, then the second offspring must select the second parent for the chromosome value at position j.
- **One-point crossover**: A random position, j, in the chromosome vector is found. For the first offspring, the first j values of the offspring is a copy of the first parent and the remaining values are a copy of the second parent. For the second offspring, the first j values of the offspring is a copy of the second parent and the remaining values are a copy of the first parent.
- **Two-point crossover**: Two random positions, j_1 and j_2, in the chromosome vector are found. For the first offspring, the first j_1 values are copies of the

first parent. The values between j_1 and j_2 are copies of the second parent. The remaining values are copies of the first parent. For the second offspring, alternate parents of the first offspring are selected. That is, the first j_1 values are copies of the second parent, the values between j_1 and j_2 are copies of the first parent, and the remain values are copies of second parent.

For continuous-valued genes, arithmetic crossover can be used. For instance, two offspring represented by the vectors $\mathbf{o}_1$ and $\mathbf{o}_2$ can be generated from two parents $\mathbf{x}_1$ and $\mathbf{x}_2$ using:

$$\mathbf{o}_{1,j} = r\mathbf{x}_{1,j} + (1.0 - r)\mathbf{x}_{2,j} \tag{2.16}$$

$$\mathbf{o}_{2,j} = (1.0 - r)\mathbf{x}_{1,j} + r\mathbf{x}_{2,j} \tag{2.17}$$

with $r \in U(0, 1)$.

Mutation

Due to random selection of the initial population, it may happen that not all gene values are covered. If only the crossover operator is used as an adaptation mechanism, then only those parts of the search space as defined by all possible combinations of initial gene values can be covered. That is, potential parts of the search space (where a solution may reside), may not even be searched. Mutation has as its objective to introduce new genetic material into a population, thereby increasing the diversity of the population. A mutation probability, known as the mutation rate (p_m), is used to control the mutation process so as not to lose good genetic material of an individual. The probability is usually large at the initial stages of the search process and then decreased as the search progresses. A large mutation rate increases the diversity of the population which aids in the prevention of premature convergence on a local optimum. As the mutation rate decreases, less genetic material is introduced into a population, which allows the population to converge on an optimum.

Drawbacks and Advantages

GAs have the ability to execute in parallel [38], which makes them well suited for optimisation problems where good results are needed quickly. However, GAs have been criticised for their trade-off in quality of the solution for the size of the population [35][34][97]. The trade-off being that a large population produces better results than a smaller population, but requires a high computational cost.

As mentioned previously, EC is a paradigm of CI. The following section discusses another paradigm of CI, namely swarm intelligence.

2.4.2 Swarm Intelligence

Bird flocks show the ability to maintain an optimal formation while flying in a three dimensional space, even after a sudden change in direction [24]. This social behaviour was the inspiration upon which Eberhart and Kennedy [50] based the development of the PSO algorithm. Instead of a bird flock, a swarm of particles move through a high-dimensional space. The position of a particle is influenced by the particle's own experience and its ability to emulate the successful behaviour of other particles. Particles form neighbourhoods on the basis of a specific social structure, which enables particles to learn from the experience of neighbouring particles.

Let $\mathbf{x}_i(t)$ denote the position of particle i in hyperspace at time t. The position of i is then changed by adding a velocity component $\mathbf{v}_i(t)$ to the current position

$$\mathbf{x}_i(t) = \mathbf{x}_i(t-1) + \mathbf{v}_i(t) \tag{2.18}$$

The velocity vector, $\mathbf{v}_i(t)$, represents the cognitive and social behaviour of the particle to imitate more successful particles. When no social information is used to update the particle's position, the velocity component only considers the particle's personal best position and current position. A particle's personal best position is the position at which the particle has found its best fitness for the duration of the search. The velocity for a particle with no social information is updated using the equation

$$\mathbf{v}_i(t) = \mathbf{v}_i(t-1) + \rho(\mathbf{y}_i - \mathbf{x}(t)) \tag{2.19}$$

where ρ is a positive random number, $\mathbf{v}_i(0)$ is randomly initialised or can be equal to a zero vector, $\mathbf{0}$, and $\mathbf{y}_i$ is the personal best position of the particle. This model is known as the cognition-only model.

When a particle belongs to a neighbourhood that spans the entire swarm, the social information used to update the position of the particle includes the position of the best particle from the entire swarm and the particle's personal best position. Adding social information, the velocity equation changes to

$$\mathbf{v}_i(t) = \mathbf{v}_i(t-1) + \rho_1(\mathbf{y}_i - \mathbf{x}_i(t)) + \rho_2(\widehat{\mathbf{y}} - \mathbf{x}_i(t)) \tag{2.20}$$

where $\rho_1 = \mathbf{r}_1(t)c_1$ and $\rho_2 = \mathbf{r}_2(t)c_2$, with c_1 and c_2 being positive acceleration constants and $\mathbf{r}_1, \mathbf{r}_2 \in U(0,1)^n$, $\widehat{\mathbf{y}}$ is the global best position and is defined as the best $\mathbf{y}_i$ of all the particles in the entire swarm. This type of model is also referred to as the global best (*gbest*) PSO using a star neighbourhood topology. The star neighbourhood topology connects each particle to every other particle in the swarm. Another model, known as the local best (*lbest*) PSO, based on a ring neighbourhood topology, divides the swarm into overlapping neighbourhoods of particles. Instead of having a best particle for the entire swarm, the *lbest* PSO keeps a best particle for each neighbourhood, represented as $\widehat{\mathbf{y}}_i$.

To control the exploration and exploitation abilities of a swarm, an inertia weight was introduced [50], as follows:

$$\mathbf{v}_i(t) = \phi\mathbf{v}_i(t-1) + \rho_1(\mathbf{y}_i - \mathbf{x}_i(t)) + \rho_2(\widehat{\mathbf{y}} - \mathbf{x}_i(t)) \tag{2.21}$$

where the inertia weight, ϕ, is used to control the influence of the particle's previous velocity on the new velocity. A large ϕ facilitates exploration, while a small ϕ causes the swarm to exploit an area of the search space. To balance exploration and exploitation, ϕ is usually initialised to a large value which is then decreased over time.

The velocity equation is sensitive to its parameter values and various heuristics have been suggested to ensure convergence. Kennedy [49] has empirically shown that

values c_1 and c_2 should be bounded such that $c_1 + c_2 \leq 4$ to prevent velocity and position vectors to grow exponentially, causing particles to diverge. Van den Bergh established that the relation between the inertia and the acceleration constants must be $\phi \geq \frac{1}{2}(c_1 - c_2) - 1$ for guaranteed convergence to an equilibrium, otherwise the PSO experiences cyclic or divergent behaviour [94]. This was also independently derived by Trelea [92]. Clerc and Kennedy derived constriction coefficients to further ensure convergence [17].

Guaranteed Convergence PSO

Van den Bergh [94] has shown that the standard PSO algorithm is not guaranteed to converge even on a local optimum. When a swarm is stabilised, it simply means that particles have converged on the best position discovered by the swarm. Assuming *gbest* PSO, this position is defined for each particle as

$$\frac{c_1\mathbf{y}_i + c_2\widehat{\mathbf{y}}}{c_1 + c_2} \tag{2.22}$$

To show that the swarm may have prematurely converged to this point, let $\mathbf{x}_i(t) = \mathbf{y}_i(t) = \widehat{\mathbf{y}}(t)$. From equation (2.21), both the social and cognitive components will be zero, and it is only the particle's inertia that contributes to any change in its position. If this condition persists for a number of iterations, then $\phi\mathbf{v}_i(t-1) \rightarrow 0$, as $\phi < 0$ in which case position updates become zero [26]. More formal proofs to show that the basic PSO does not have guaranteed convergence to a local optimum can be found in [94].

To address this problem Van den Bergh and Engelbrecht [95], and Van den Bergh [94] adapted the PSO to force the global best to change. The resulting algorithm, referred to as the guaranteed convergence PSO, changes the velocity update for the global particle to

$$\mathbf{v}_{\tau,i}(t+1) = -\mathbf{x}_{\tau,j}(t) + \mathbf{y}_j(t) + \phi\mathbf{v}_{\tau,j}(t) + \rho(t)(1 - 2r_{3,j}(t)) \tag{2.23}$$

where τ is the index of the global best particle so that

$$\mathbf{y}_\tau = \widehat{\mathbf{y}} \tag{2.24}$$

and $\rho(t)$ is a scaling factor defined below in equation (2.25) and $\mathbf{y}_j$ is the personal best position of the global best particle. The other particles in the swarm continue to use the normal velocity update equation (2.21). The $-\mathbf{x}_{\tau,j}$ resets the particle position to $\mathbf{y}_j$. The vector $\phi\mathbf{v}_j$ represents the current direction which is added to the $\mathbf{y}_j$. The term $\rho(t)(1-2r_{3,j}(t))$ generates a random value that is a sub-space of lengths $\rho(t)$. The addition of the $\rho(t)$ term causes the PSO to perform a random search in an area surrounding the global best position so that the global best is forced to change. The diameter of the search is controlled by the parameter $\rho(t)$. The value of $\rho(t)$ is adapted after each time step as follows,

$$\rho(t+1) = \begin{cases} 2\rho(t) & \text{if } success > s_c \\ 0.5 & \text{if } failures > f_c \\ \rho(t) & \text{otherwise} \end{cases} \tag{2.25}$$

where $success$ and $failures$ denote the number of consecutive successes and failures respectively. A failure is defined as $f(\mathbf{y}(t)) \geq f(\mathbf{y}(t-1))$ (assuming minimization). The parameters s_c and f_c are thresholds, where the optimal choice of the values for the parameters is dependent on the search space. Van den Bergh [94] found that $\rho(0) = 1.0$ produces acceptable results. The GCPSO algorithm has shown to be efficient and robust and it has been successfully used to locate single solutions within a search space [95][94].

2.5 Multiple Optima

The algorithms discussed thus far have been developed (in their original form) to find a single optimum. These single solution algorithms are not suitable for finding multiple solutions to the same optimisation problem. Algorithms that are designed to find multiple solutions are referred to as niching algorithms. Within these algorithms, individuals (or particles) speciate to form niches, where each niche represents a single solution.

In biological terms, a niche sustains a group of species. Different species exploit different niches in order to survive. A single niche can only sustain a certain number

of species. Species that cannot be supported by one niche have to migrate to another niche or evolve into another species to survive in the same niche. This adaptation process, known as speciation, will create a new species. Allopatric speciation occurs when a small population of a species is separated from the main population and begins to adapt to their new environment. Parapatric speciation is the formation of a new species through evolution. This type of speciation refers to species that evolve in the same niche (instead of migrating to another niche), and compete with other species that coexist in the same niche.

Niching algorithms that implement either allopatric or parapatric speciation are discussed in further detail in the following chapters.

2.6 Multi-objective Optimisation

Problems presented thus far are single objective optimisation problems for multi-modal and unimodal functions. The simultaneous optimisation of competing objective functions is referred to as multi-objective optimisation (MOO). An example of competing objective functions can be found in the field of mechanical design, where several objective functions may rate highly those designs which perform well based on maintenance, ease of use, reliability, etc. [8]. Combining all these objectives into a single function that rates the designs is usually complex. With multi-objective optimisation a set of solutions is searched for, such that the objectives cannot be all simultaneously improved. This set of solutions is referred to as the Pareto optimal set. The solutions contained by the Pareto optimal set are referred to as non-dominated, or non-inferior solutions [30]. The plot of the non-dominated solutions is referred to as the Pareto front. The Pareto front is described as either concave, convex, partially concave, partially convex, discrete or continuous. More formally, the objective of MOO is to find a set of non-dominated solutions,

$$\mathbf{x}^* = [\mathbf{x}_1^*, \mathbf{x}_2^*, ..., \mathbf{x}_n^*]^T \tag{2.26}$$

satisfying multiple objectives specified as:

- m inequality constraints: $g_j(\mathbf{x}^*) \geq 0, j = 1, 2, ..., m$

- p equality constraints: $h_j(\mathbf{x}^*) = 0, j = 1, 2, ..., p$
- Optimisation of a N-dimensional vector of optimisation functions: $\mathbf{f}(\mathbf{x}^*) = [f_1(\mathbf{x}^*), f_2(\mathbf{x}^*), ..., f_N(\mathbf{x}^*)]^T$

The optimal solution for the objective functions may be found in different locations that conflict with each other. If there are no other solutions in the search space that may improve the value of any one component without deteriorating another component, then the solution is considered to be non-dominated.

MOO can broadly be classified into two main approaches: non-Pareto and Pareto based approaches. Each of these approaches are briefly discussed in the following subsections.

2.6.1 Non-Pareto Based Approach

Non-Pareto based approaches do not directly incorporate the concept of a Pareto optimum. Examples of non-Pareto based approaches include the weight aggregated approach [44], vector evaluated genetic algorithm (VEGA) [81], lexicographic ordering [31][9], the multi-objective version of evolutionary strategies [53], and target vector approaches [40]. The following subsections discuss the commonly used weight aggregated approach and VEGA.

Weight Aggregated Approach

The weighted aggregated approach replaces the multiple objectives with a single objective, expressed as a summation of the separate objective functions, that is

$$f = \sum_{j=1}^{N} w_j f_j(\mathbf{x}) \tag{2.27}$$

where $w_j, j = 1, ..., N$ are non-negative weights. It is often assumed that $\sum_{j=1}^{N} w_j = 1$. The weights can either be fixed or dynamically changed during the optimisation process.

Conventional weighted aggregation (CWA) refers to the case where the weights are fixed. This approach only allows for a single Pareto optimal solution to be found

per search. Thus, to find the desired number of Pareto optimal solutions the search must be executed several times. Therefore, the approach has a high computational cost and requires prior knowledge of the search space to choose appropriate weight values [30][46].

The Bang-Bang weighted aggregation (BWA) approach oscillates the weights associated with each objective function. For a MOO problem with two objective functions, the weights are calculated as

$$w_1(t) = \text{sign}(\sin(2\pi t/F_{freq})), \tag{2.28}$$

$$w_2(t) = 1 - w_1 \tag{2.29}$$

where t represents the index of the current iteration and F_{freq} is the frequency at which the weights change.

Dynamic weighted aggregation (DWA) is similar to BWA, but gradually modifies the weights. DWA modifies the weights of a two objective MOO problem as

$$w_1(t) = |(\sin(2\pi t/F_{freq})|\,, \tag{2.30}$$

$$w_2(t) = 1 - w_1 \tag{2.31}$$

Vector Evaluated Genetic Algorithm

Schaffer introduced the idea of treating objective functions separately in VEGA [81]. The approach generates the next generation, $g+1$, as sub-populations from the current generation, g. For N objectives and a population of size P, N sub-populations of size P/N each is generated. The next generation is obtained by shuffling the sub-populations together to create a new population of size P. The crossover and mutation operators are applied to the new population as usual. Shuffling all sub-populations results in averaging the fitness components of each objective. Thus, the fitness of individuals corresponds to a function of the objectives [30][78]. Therefore, non-dominated individuals are assigned different fitness values [30].

2.6.2 Pareto Based Approach

Pareto based approaches involve ranking non-dominated individuals and a technique to maintain population diversity. Examples of Pareto based approaches include, pure Pareto ranking [33], multi-objective optimisation GA (MOGA) [29], non-dominated sorting genetic algorithms (NSGA) [90], and niched Pareto GA (NPGA) [43]. The following section briefly discusses the NPGA.

Niched Pareto Genetic Algorithm

Horn *et al* [43] proposed a new form of tournament selection based on Pareto dominance. The new selection procedure was introduced to prevent the GA from converging on a single solution and to find multiple solutions which are part of a Pareto optimal set. The Pareto domination tournament randomly selects two individuals from the population. The individuals selected are referred to as candidates. A random set of individuals are selected from the population for comparison. The candidates are compared to each of the individuals in the comparison set. If one candidate is dominated by the comparison set, and the other is not, then the latter wins the tournament. If neither or both are dominated by the comparison set, then equivalence class sharing is used to select a winner. The equivalence class sharing technique selects the winner as the candidate which has the least number of individuals in its niche. This sharing technique is also regarded as a form of fitness sharing. The niche radius is determined by dividing the total surface area by the population size. This creates a uniform distribution to represent the possible continuous Pareto front [43]. Individuals within the niche radius are calculated as part of the niche size. Horn *et al* [43] demonstrate that the selection operator maintains a diverse population that is able to identify the Pareto optimal set.

2.6.3 MOO and Niching Similarities

Niching and MOO have the objective to find multiple solutions, with the difference that niching traditionally applies to single objective functions. Niching techniques can however be applied to MOO as shown by the fitness sharing techniques in [29][33]

and [43].

2.7 Conclusion

This chapter discussed the concept of optimisation in terms of computational terminology. Several numerical methods that are used to find the optimum of a function, f, were discussed. The numerical methods presented can be categorised into derivative and derivative-free based methods. Derivative based methods require the existence of the derivative, f', of the objective function. The non-derivative based methods discussed in this thesis require f to be unimodal. In most cases, the derivative is not known or it cannot be guaranteed that the function is unimodal. Two popular CI methods were discussed to find the optimum for functions with multiple optima or when f' is not known, these are GAs and PSO. These CI methods have been applied to a wide array of problem domains where the global optimum is needed. However, these methods are not suitable when searching for multiple optima.

The search for multiple optima is referred to as niching. The search to optimise several objectives is known as multi-objective optimisation (MOO). Niching techniques can be applied to MOO problems, as MOO and niching both search for multiple solutions. The next chapter provides an overview of the benefits and pitfalls of niching techniques that have been developed based on GAs.

Chapter 3

GA Niching Techniques

> *Owing to this struggle for life, any variation, however slight and from whatever cause proceeding, if it be in any degree profitable to an individual of any species, in its infinitely complex relationship to other organic beings and to external nature, will tend to the preservation of that individual, and will generally be inherited by its offspring. The offspring, also, will thus have a better chance of surviving, for, of the many individuals of any species which are periodically born, but a small number can survive. I have called this principle, by which each slight variation, if useful, is preserved, by the term Natural Selection, in order to mark its relation to man's power of selection.*
> Charles Robert Darwin (12 February 1809 - 19 April 1882), Origin of Species (1859) [19].

The success of genetic algorithms (GAs) in finding the global optimum in complex, high dimensional search spaces prompted research into whether GAs can find multiple optima. This chapter presents a literature overview of GA niching techniques. The techniques can be categorised into sequential and parallel niching algorithms [64][63]. Sequential algorithms discussed include iteration-based niching (section 3.2.1), and the sequential niche technique (section 3.2.2). Parallel algorithms discussed include parallel sub-populations (section 3.3.1), fitness sharing (section

3.3.2), dynamic niche sharing (section 3.3.3), crowding (section 3.3.4), and deterministic crowding (section 3.3.5).

3.1 Introduction

Traditional GAs converge to a single optimum, even if the search space contains multiple optima. This is a result of genetic drift [48] in the population. Genetic drift refers to the inability of GAs to maintain a diverse population throughout the search process, and thus individuals converge on a single optimum. Sequential and parallel techniques are two classes of approaches for discovering multiple solutions using GAs. Sequential approaches consecutively executes a GA algorithm in an effort to discover a new optimum on each execution. Parallel approaches attempt to maintain diverse sub-populations for the duration of the search, where each sub-population represents a single solution. The following sections discuss methods that have been developed for both sequential and parallel niching techniques.

3.2 Sequential Niching Techniques

Sequential niching techniques are characterised by an optimisation process that is repeated several times. In each repetition of the optimisation process, a new optimum is searched for. The following sections discuss the iteration algorithm and sequential niche technique (SNT).

3.2.1 Iterative Search

The iterative search approach repeatedly executes an optimisation algorithm to produce multiple solutions. On each execution, the algorithm is reinitialised to start at different positions in the search space. If the optimisation process is stochastic, then there is a probability that successive executions will find different solutions. If there are q optima and a *single-solution* optimisation algorithm is used, then the minimum number of times that the algorithm has to be repeated is q. However, it is possible for iterative approaches to find the same solution more than once, and

thus there is no guarantee that it may locate all solutions irrespective of the number of times that the optimisation algorithm is repeated. To compensate for duplicate solutions found, the number of repetitions have to be increased by a factor, referred to as the *redundancy factor*, Γ. If all optima have an equal probability of being found, then Γ is given by [8]:

$$\Gamma = \sum_{m=1}^{q} \frac{1}{m} \tag{3.1}$$

Γ can be approximated [47] for $q > 3$, by:

$$\Gamma \approx \gamma + \log q \tag{3.2}$$

where $\gamma \approx 0.577$ (Eulers constant). This means, to find q unique optima the algorithm must be repeated $\Gamma \times q$. If the maxima are *not* equally likely to be found, then the value of Γ will be much higher. Iterative approaches are therefore computationally complex.

3.2.2 Sequential Niche Technique

The iteration approach does not guarantee that a new solution will be found for each repetition of the optimisation algorithm. The optimisation algorithm is repeated blindly without taking any consideration of the search area that has been explored and the solutions that have been discovered in previous repetitions. The sequential niche technique (SNT) counters these pitfalls by making use of knowledge gained about the search space from previous executions of the optimisation algorithm to each subsequent execution. The SNT finds multiple solutions by consecutively executing the optimisation algorithm multiple times, such that for each execution a new solution is searched for. The original SNT, proposed by Beasley *et al* [8], was designed using a GA as the optimisation algorithm, although any other optimisation algorithm can be used.

Knowledge gained in previous executions of the optimisation algorithm is carried over to subsequent executions by maintaining two fitness functions:

- The original fitness function, F, referred to as the *raw fitness function*, and

- the modified fitness function, M.

The modified fitness function requires a *distance metric*, which quantifies how close two chromosomes are. The modified fitness function, $M(\mathbf{x})$, for an individual, $\mathbf{x}$, is computed from the raw fitness function, $F(\mathbf{x})$, multiplied by a number of single-peak derating functions (a peak refers to the maximum of a function). Initially $M(\mathbf{x}) \equiv F(\mathbf{x})$. After an execution of the optimisation algorithm, the best individual, $\widehat{\mathbf{y}}$, is found and is used to determine a single-peak derating function, $G(\mathbf{x}, \widehat{\mathbf{y}})$. Different derating functions have been used, including the exponential derating and the power law fitness functions [8], as defined in equations (3.3) and (3.4) respectively:

$$G_e(\mathbf{x}, \widehat{\mathbf{y}}) = \begin{cases} \exp(\log m * (r - d_{\mathbf{x}\widehat{\mathbf{y}}})/r) & \text{if } d_{\mathbf{x}\widehat{\mathbf{y}}} < r \\ 1 & \text{otherwise} \end{cases} \tag{3.3}$$

$$G_p(\mathbf{x}, \widehat{\mathbf{y}}) = \begin{cases} (d_{\mathbf{x}\widehat{\mathbf{y}}}/r)^{\alpha} & \text{if } d_{\mathbf{x}\widehat{\mathbf{y}}} < r \\ 1 & \text{otherwise} \end{cases} \tag{3.4}$$

where $d_{\mathbf{x}\widehat{\mathbf{y}}}$ denotes the distance between the best individual of the current execution and another individual $\mathbf{x}$ as determined by the distance metric; r is the niche radius that is a constant defined prior to the SNT search process. The niche radius is an assumption of the inter-niche distance in the search space. Figures 3.1 (a) and 3.1 (b) illustrate several curves of the derating functions defined in equations (3.3) and (3.4) with $r = 1$. For a large α and small m the derating function increases its strength of attenuation on the raw fitness function.

The niche radius, r, requires prior knowledge of the inter-niche distance to be provided. Deb [21] proposed to calculate the niche radius for an n-dimensional search space as:

$$r = \frac{\sqrt{n}}{2 \times \sqrt[q]{n}} \tag{3.5}$$

under the following assumptions that

- there are q unique maxima each surrounded by a hypersphere of radius r,

- the hyperspheres do not overlap, and
- the hyperspheres completely fill the n-dimensional problem space.

After each execution of the optimisation algorithm, the modified fitness function is updated according to

$$M_{t+1} \equiv M_t(\mathbf{x}) * G(\mathbf{x}, \widehat{\mathbf{y}}_t) \tag{3.6}$$

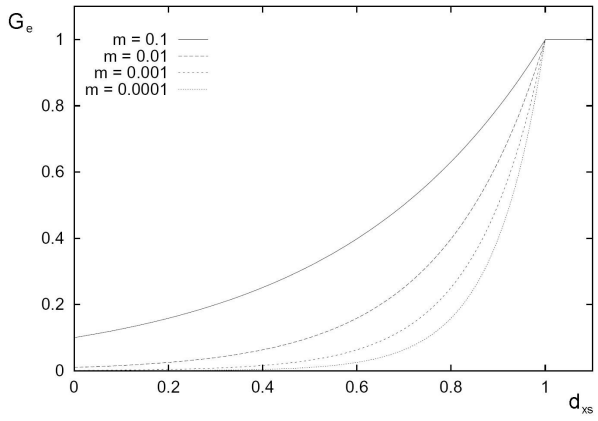
Figures 3.2 (a), (b) and (c) illustrate a possible search using SNT. In figure 3.2 (a), the first execution of the GA finds an optimum (shown as a cross) and modifies the search space accordingly to produce the modified fitness function illustrated by the broken line in figure 3.2 (b). This introduces two false peaks in the modified fitness function search space (indicated by the brocken line in figure 3.2(b)). On the second execution of the GA, one of the false peaks that were introduced in the previous execution are found. This results in modifying the search space as shown in figure 3.2 (c). Figure 3.2 (c) illustrates that the false peak is almost completely removed from the search space.

The SNT algorithm is summarised in figure 3.3, assuming a maximisation problem. To find the minima of a function, a different type of derating function must be used [73], or the problem should be changed into a maximisation problem. From figure 3.3, the solution threshold represents the lower fitness limit for accepting the candidate solution as a solution. If no information is known about the values of the maxima of interest then a value of zero could be used [8].

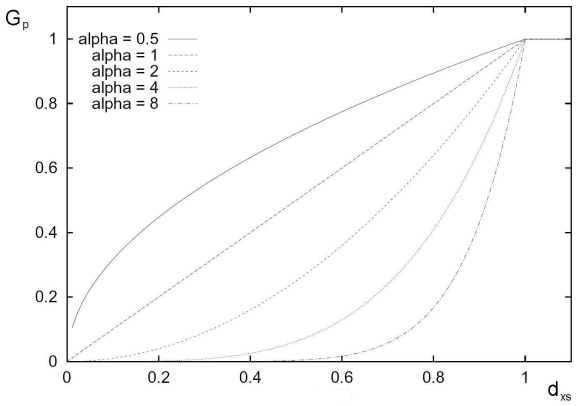
Pitfalls of SNT

SNT has the disadvantage that the number of niches has to be known, because this is used as a stopping criterion, and to calculate the niche radius. If an inappropriate niche radius is used, then the following symptoms have been experienced [8]:

- A niche radius that is too small introduces new false optima, which may be inaccurately reported as solutions in the search space.

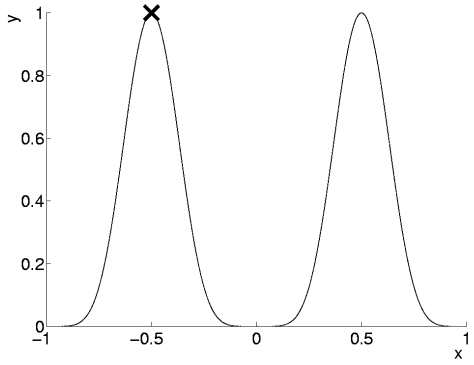


(a)

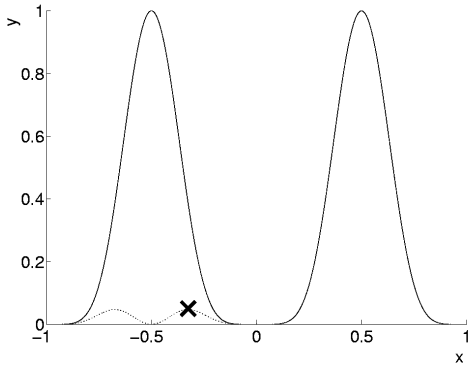


(b)

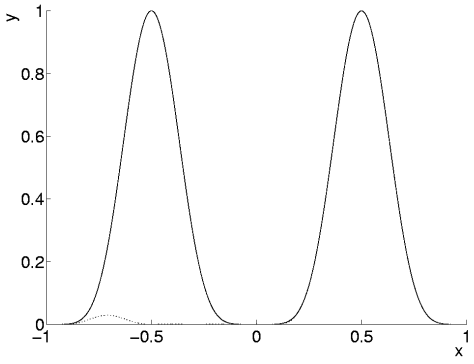
Figure 3.1: (a) Exponential derating function, equation (3.3) (b) Power law derating function, equation (3.4), where *alpha* represents α



(a)



(b)



(c)

Figure 3.2: (a) The first run of SNT finds the optimum indicated by a cross (b) The second run of SNT finds a false optimum indicated by a cross (c) The third run of the SNT almost completely removes the false optimum.

1. Equate the modified fitness to the raw fitness function (meaning that the two functions are exactly the same).
2. Run the GA (or any other search technique), using the modified fitness function, keeping a record of the best individual found in the run.
3. Update the modified fitness function to give a depression in the region near the best individual, producing a new modified fitness function.
4. If the raw fitness function of the best individual exceeds the solution threshold, display this as a solution.
5. If all solutions have not been found, or the maximum number of executions has not been exceeded return to step 2.

Figure 3.3: SNT algorithm

- A large niche radius may cause the introduction of new false peaks, as well as reducing the height and shifting the position of other peaks. The height reduction may cause a peak not to be accepted as a solution, or if the peak is recognised as a solution, then the location will be incorrect.

In order to solve the inaccuracy problems, Beasley *et al* [8] suggest to find an approximate solution using the modified fitness function and then to use a local search using the raw fitness function. Another pitfall of using SNT with a single solution optimisation algorithm is that the optimisation algorithm needs to be executed at least q times to find q unique optima, which increases the computational complexity.

3.3 Parallel Niching Techniques

Unlike sequential niching techniques, parallel niching techniques attempt to find multiple solutions in a single execution of the optimisation algorithm. This is achieved by maintaining sub-populations of high diversity to prevent genetic drift. The fol-

lowing subsections discuss several parallel niching techniques, including parallel sub-populations, fitness sharing, dynamic niche sharing, crowding, and deterministic crowding.

3.3.1 Parallel Sub-populations

Parallel sub-population strategies divide the population of individuals into multiple non-overlapping sub-populations. Each sub-population is evolved independently and in parallel with the other populations. Parallel sub-population methods are similar to the iteration approach, and has no guarantee that all optima will be found. Furthermore, it is possible that sub-populations may converge on the same optima, because there is no communication between sub-populations. If communication is allowed between the sub-populations to facilitate the spreading of good genes during crossover, the diversity of solutions will be reduced and the whole population will eventually converge to a single solution [39].

3.3.2 Fitness Sharing

Fitness sharing encourages diverse sub-populations by enforcing that each niche hold a finite number of individuals. Diversity is achieved by calculating the fitness of each individual as being inversely proportional to the number of individuals in the same niche [37]. The shared fitness of individual $\mathbf{x}_i$ is calculated as:

$$F_{sh}(\mathbf{x}_i) = \frac{F(\mathbf{x}_i)}{\sum_j^P sh(d_{ij})} \tag{3.7}$$

where $i \neq j$, P is the number of individuals in the population, and

$$sh(d) = \begin{cases} 1 - (d/\sigma_{share})^\alpha & \text{if } d < \sigma_{share} \\ 1 & \text{otherwise} \end{cases} \tag{3.8}$$

where F_{sh} is referred to as the sharing function, d_{ij} is the euclidean distance between individual $\mathbf{x}_i$ and $\mathbf{x}_j$, F is the absolute fitness of the individual (i.e. the fitness before adjustment), σ_{share} is a threshold for determining if individuals $\mathbf{x}_i$ and $\mathbf{x}_j$

are optimising the same niche, α is a constant used to regulate the sharing function (usually set to 1) [64][63]. If d_{ij} is less than σ_{share} then the individuals share their fitness based on how similar they are. The more alike the individuals are, the lower the fitness value of the individuals. In turn, during the reproduction phase of the GA, the selection operator selects individuals that are not similar to other individuals. Thus, promoting diversity in the population.

Fitness sharing is based on the following assumptions:

- prior knowledge of the number of niches must be available, as it is used as a stopping criterion for the algorithm, and
- each niche occurs a distance of at least $2\sigma_{share}$ apart from each other.

These assumptions present certain disadvantages, namely that the number of niches are generally not known, and depending on the value of σ_{share}, multiple optima may exist within $2\sigma_{share}$. Furthermore, to calculate the fitness of one individual in the population, the distance to every other individual has to be calculated. The total computational cost of calculating the fitness of all individuals in the population is therefore $O(P^2)$, where P is the population size. To find many optima, a large population is needed and thus, a high computational cost is incurred [88].

3.3.3 Dynamic Niche Sharing

Dynamic niche sharing was developed to counter the computational cost of fitness sharing [66]. The algorithm has the same assumptions as fitness sharing.

For GAs using dynamic niche sharing, individuals populate niches as the algorithm iterates [66]. Dynamic niche sharing attempts to identify q niches as they are populated. Individuals in the population are defined as belonging to a dynamic niche (i.e. individuals within σ_{share} of a dynamic niche) or non-niche category. The fitness of individuals that belong to the non-niche category are calculated using the fitness sharing function, defined in equation (3.7), whereas the fitness of individuals that belong to a dynamic niche are calculated by first determining fitness using the absolute (or raw) fitness function, and then dividing the value by the number of individuals in the niche.

Dynamic niche sharing is more efficient than standard sharing techniques. However, the same assumptions as fitness sharing are made, and therefore the algorithm suffers from the same disadvantages caused by these assumptions.

3.3.4 Crowding

Crowding [48] is based on replacing individuals in the population with similar individuals. That is, members of a niche compete with each other for the same resource. As more individuals join the niche, weaker members of that niche will be crowded and replaced by stronger members [62]. Any GA which replaces individuals to maintain a stable sub-population can be called a crowding method.

Crowding, as originally proposed by De Jong [48], replaces a fixed percentage of the population in each generation using a reproduction operator. A random sample of individuals are selected from the current population for the reproduction phase. A newly generated individual replaces an individual in the sample that is the closest to it in terms of Hamming distance. This replacement strategy reduces changes in the population distribution from one generation to the next, which increases the diversity of the population.

Studies have shown that the crowding method is a poor niching algorithm [61] because of large replacement errors. Replacement errors are defined as the replacement of an individual in a niche by another individual in another niche. Measures other than the Hamming distance, such as phenotypic distance [36] have shown to reduce replacement errors.

3.3.5 Deterministic Crowding

To eliminate replacement errors of the basic crowding algorithm, Mahfoud proposed deterministic crowding [61]. Mahfoud demonstrated that there is a high probability that individuals most similar to an offspring, are the offspring's parents. Mahfoud therefore suggests that the replacement scheme in crowding change to replacing one of the offspring's parents. This replacement strategy has a better maintenance of population diversity, as the replacement strategy finds (on average) better similarity

matches.

Deterministic crowding randomly pairs individuals in each generation, to yield $P/2$ pairs, where P is the size of the population. Each selected pair undergoes crossover, and other genetic operators (such as mutation) that are applicable during this phase, to produce two children. The two children then compete against their parents for inclusion in the population [62]. The parent-child tournament forces competition between the closest parent and child (based on phenotypic distance). That is, the only possible selection for parent-child competition can be: parent 1 against child 1 and parent 2 against child 2, or parent 1 against child 2 and parent 2 against child 1. The DC algorithm is summarised in figure 3.4, where $d(\cdot)$ is the phenotypic distance function.

3.4 Conclusion

This chapter provided an overview of popular GA niching approaches. Problems with these methods have been discussed. These problems include prior knowledge of the number of solutions (which is generally not available), and computational complexity caused by large population sizes of the GA, which is usually required for niching approaches. The next chapter discusses niching techniques for particle swarm optimisation.

Let generation $g = 0$

Repeat

For $i = 1$ to $P/2$ do

1. Select 2 parents, $\mathbf{x}_1$ and $\mathbf{x}_2$, randomly from the population C_g
2. Cross $\mathbf{x}_1$ and $\mathbf{x}_2$ yielding offspring $\mathbf{o}_1$ and $\mathbf{o}_2$.
3. Apply mutation operators to $\mathbf{o}_1$ and $\mathbf{o}_2$, yielding $\mathbf{o'}_1$ and $\mathbf{o'}_2$
4. If $[d(\mathbf{x}_1, \mathbf{o'}_1) + d(\mathbf{x}_2, \mathbf{o'}_2)] \leq [d(\mathbf{x}_1, \mathbf{o'}_2) + d(\mathbf{x}_2, \mathbf{o'}_1)]$
   a. If $f(\mathbf{o'}_1) > f(\mathbf{x}_1)$ replace $\mathbf{x}_1$ with $\mathbf{o'}_1$
   b. If $f(\mathbf{o'}_2) > f(\mathbf{x}_2)$ replace $\mathbf{x}_2$ with $\mathbf{o'}_2$

   else

   a. If $f(\mathbf{o'}_2) > f(\mathbf{x}_1)$ replace $\mathbf{x}_1$ with $\mathbf{o'}_2$
   b. If $f(\mathbf{o'}_1) > f(\mathbf{x}_2)$ replace $\mathbf{x}_2$ with $\mathbf{o'}_1$

$g = g + 1$

Until $g >$ maximum number generations

Figure 3.4: Deterministic crowding algorithm

Chapter 4

PSO Niching Techniques

This chapter provides a summary of PSO niching methods. Section 4.2 reasons about the ability of the standard PSO to locate multiple solutions. Methods that use or have modified the standard PSO algorithm to find multiple solutions are also discussed. These include objective function stretching, vector-based PSO, parallel vector-based PSO, species-based PSO, *nbest*, and NichePSO.

4.1 Introduction

Particle swarm optimisation (PSO) has shown to be successful in finding good solutions to optimisation problems [71]. However, the standard PSO cannot be directly applied to niching problems, as discussed in section 4.2, and needs to be modified to support multiple solutions. Techniques that have modified the standard PSO can be categorised into sequential and parallel methods. Objective function stretching and vector-based PSO are two sequential methods discussed in this chapter in section 4.3.1 and 4.3.3 respectively. The *nbest* PSO, NichePSO, parallel vector-based PSO, and species-based PSO are parallel methods discussed in sections 4.4.1, 4.4.2, 4.4.3, 4.4.4 respectively.

4.2 Niching Ability of *lbest* and *gbest* PSO

A recent study into the niching capabilities of PSO has shown that the *gbest* and *lbest* PSO are not suitable niching algorithms [26]. Based on formal proofs from [94][18][27] it is emphasised that each particle of a *gbest* PSO will converge on a stable point which is the weighted average of the particle's personal best and global best positions, defined as

$$\frac{c_1\mathbf{y}_i + c_2\widehat{\mathbf{y}}}{c_1 + c_2} \tag{4.1}$$

where $\mathbf{y}_i$ is the particle's personal best fitness, $\widehat{\mathbf{y}}$ is the particle's global best fitness, c_1 and c_2 are positive acceleration constants. Based on this finding it is impossible for the *gbest* PSO to locate more than one solution in a single run of the algorithm. This proof does not, however, directly apply to the *lbest* PSO. While it is the case that each particle in a neighbourhood will converge to a weighted average of the particle's personal best position and the neighbourhood best, there is no formal proof that all particles will converge on one stable point. Due to overlapping neighbourhoods, most empirical studies show that particles move towards the same point. However, Engelbrecht *et al* [26] show empirically that multiple solutions can be found by the *lbest* PSO, although only a small number of solutions have been located for the tested functions, and only when large swarm sizes were used. It should also be noted that the searches were terminated after a certain number of function evaluations has been exceeded, at which point the swarm has not necessarily reached an equilibrium state, and neighbourhoods may not have converged.

The *gbest* topology uses the social information of the entire swarm to update the position of the particles and implicitly assumes that there will be a single optimum solution. An iteration-based approach using *gbest* PSO was also reasoned to be inappropriate as there is no guarantee that different global best solutions will be found for each execution of the algorithm [27][26]. In fact there is not even a guarantee that the point to which the particles converge is a local or global optimum (formal proofs can be found in [94]).

The cognition-only model may be used to search for multiple solutions due to

its exploration abilities, since the cognition-only model is essentially performing multiple stochastic searches. It is therefore possible that multiple solutions can be found. However, an extra pre-processing step maybe required to group similar solutions into niches.

The above discussion suggests that the standard PSO algorithm is not a suitable niching technique, and needs to be modified to obtain multiple, unique solutions. The following sections summarise several techniques where the standard PSO has been modified to find multiple solutions.

4.3 Sequential Methods

This section discusses sequential niching methods using a PSO. These methods are, objective function stretching in section 4.3.1, deflection technique in section sec:DeflectionTechnique, and vector-based PSO in section 4.3.3.

4.3.1 Objective Function Stretching

Objective function stretching, developed by Parsopoulos *et al* [70][73], reduces the number of iterations required to locate a global minimum using the PSO. The approach is similar to SNT as it modifies the search space so that the PSO does not return to a previously discovered local minimum. When a local minimum has been discovered, local minima situated above it are eliminated without affecting the minima below it. The original function, $f(\mathbf{x})$, undergoes a two-stage transformation, applied after a local-minimum, $\mathbf{p}$, of function f has been located. The two transformation functions are:

$$G(\mathbf{x}) = f(\mathbf{x}) + \gamma_1 \frac{\|\mathbf{x}-\mathbf{p}\| \cdot (\text{sign}(f(\mathbf{x}) - f(\mathbf{p})) + 1)}{2} \tag{4.2}$$

$$H(\mathbf{x}) = G(\mathbf{x}) + \gamma_2 \frac{\text{sign}(f(\mathbf{x}) - f(\mathbf{p})) + 1}{2\text{tanh}(\mu(G(\mathbf{x}) - G(\mathbf{p})))} \tag{4.3}$$

where γ_1, γ_2 and μ are arbitrarily chosen positive constants, and $\|\mathbf{x}-\mathbf{p}\|$ calculates the length of the vector, and the sign$(\cdot)$ function is defined as

$$\text{sign}(x) \approx \text{logsig}(x) = \frac{2}{1 + \exp(-\lambda_1 x)} - 1 \cong \tanh(\lambda_2 x) \tag{4.4}$$

The first transformation given in equation (4.2) removes all local minima that are above $\mathbf{p}$. The second transformation stretches the neighbourhood of $\mathbf{p}$ upwards, such that higher function values are assigned to the points where local minima are above $\mathbf{p}$. In both transformations, the vector coordinates of $\mathbf{p}$ are not changed.

All other local minima that are located above the minima found by the stretching technique are removed. This does not make the stretching technique a true multimodal algorithm as it does not search for all global and local minima [94]. Furthermore, the stretching technique has the problem of introducing false local minima, similar to SNT (refer to section 3.2.2).

4.3.2 Deflection Technique

The deflection technique [72] is another method that modifies the search space. The deflection technique is similar to the SNT as it modifies the search space at all minima discovered. Unlike objective function stretching, the deflection technique is able to find both the global minimum and local minima. If the minima discovered is represented as $\widehat{\mathbf{y}}_j$ for $j = 1, \ldots, q$, then the deflection technique defines the modified fitness function, $M(\mathbf{x})$, as

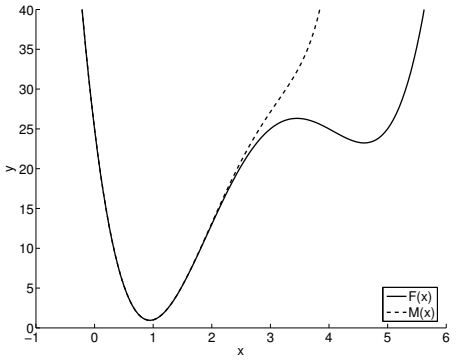
$$M(\mathbf{x}) = T_1(\mathbf{x}, \widehat{\mathbf{y}}_1, \lambda_1)^{-1} * T_2(\mathbf{x}, \widehat{\mathbf{y}}_2, \lambda_2)^{-1} \ldots T_q(\mathbf{x}, \widehat{\mathbf{y}}_q, \lambda_q) * F(\mathbf{x})^{-1} \tag{4.5}$$

where $F(\mathbf{x})$ is the original fitness function, λ_j for $j = 1, \ldots, q$ are relaxation parameters, and function T is defined as

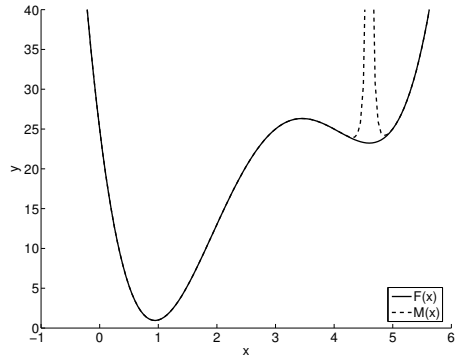
$$T(\mathbf{x}, \widehat{\mathbf{y}}_j, \lambda_j) = \tanh(\lambda_j \|\mathbf{x} - \widehat{\mathbf{y}}_j\|) \tag{4.6}$$

The relaxation parameters, λ_j, adjust the strength of attenuation on the original fitness function. If the original fitness function is defined as

$$F(\mathbf{x}) = \mathbf{x}^4 - 12\mathbf{x}^3 + 47\mathbf{x}^2 - 60\mathbf{x} + 25 \tag{4.7}$$



(a)



(b)

Figure 4.1: The deflection technique applied to the function defined in equation 4.7 at $\mathbf{x} = 4.601$ where (a) $\lambda = 1$ and (b) $\lambda = 10$.

then $\lambda = 1$ introduces large modifications to the search space (refer to figure 4.1(a)), whereas $\lambda = 10$ introduces small modifications to the search space (refer to figure 4.1(b)).

The deflection technique introduces false minima, and shifts the height of minima near the points of the search space that have been modified, which are similar problems experienced by the SNT. Furthermore, the deflection technique cannot be used on a function that has minima equal to zero, as M will also have a value of zero at these minima, resulting in no modification to the fitness function, therefore particles cannot be deflected from these minima.

4.3.3 Vector-Based PSO

The vector-based PSO proposed by Schoeman and Engelbrecht [83][82] was developed with the main objective to reduce the amount of prior knowledge required of the problem domain, such as the inter-niche distance. The vector-based PSO demarcates each niche such that it would only contain particles which would eventually converge on the neighbourhood best of that niche. This demarcation is achieved using the properties of the dot product.

The inner product of two vectors, $\mathbf{a} = a_1\mathbf{i} + a_2\mathbf{j} + a_3\mathbf{k}$ and $\mathbf{b} = b_1\mathbf{i} + b_2\mathbf{j} + b_3\mathbf{k}$, is defined as

$$\mathbf{a} \cdot \mathbf{b} = a_1 b_1 + a_2 b_2 + a_3 b_3 \tag{4.8}$$

If $\mathbf{a}$ and $\mathbf{b}$ are non-zero vectors, then the angle, θ, between $\mathbf{a}$ and $\mathbf{b}$ is calculated as

$$\mathbf{a} \cdot \mathbf{b} = \|a\| \, \|b\| \cos(\theta) \tag{4.9}$$

with $0 \leq \theta \leq \pi$. The following properties of the dot product are used in the vector-based PSO:

- If θ is in the first quadrant ($0 \leq \theta < \pi/2$) then the value of $\cos(\theta)$ will be positive.
- If θ is in the second quadrant ($\pi/2 < \theta \leq \pi$) then the value of $\cos(\theta)$ will be negative.

The vector-based PSO calculates the dot product between the particle's personal best position, $\mathbf{y}_i$, and the particle's neighbourhood best position, $\widehat{\mathbf{y}}_i$. Then

- if $\mathbf{y}_i \cdot \widehat{\mathbf{y}}_i < 0$, then the angle, θ, between the two vectors is in the second quadrant, and the particle is moving away from the neighbourhood best position, or
- if $\mathbf{y}_i \cdot \widehat{\mathbf{y}}_i > 0$, then the angle, θ, between the two vectors is in the first quadrant, and the particle is moving towards the neighbourhood best position.

This allows for a niche radius to be calculated as the distance from the particle's neighbourhood best position to the nearest particle with a negative dot product, since these two particles are converging on different optima. Particles inside the niche radius and with positive dot products are marked as belonging to that niche [83], since they are converging on the same optimum. The particles belonging to a niche are treated as a subswarm to be optimised. When the subswarm has converged on a solution, all particles within that subswarm are marked as *processed*, and maintained in the niche. Particles maintained in a niche are not allowed to move to another niche, meaning that the particles are not allowed to belong to another subswarm. The algorithm is summarised in figure 4.2 [82]. The algorithm terminates when all particles have been marked as *processed*, or the number of iterations has exceeded a maximum limit.

Vector-based PSO is a sequential algorithm as niches are optimised in turn. Schoeman *et al* [82] found that because of this sequential nature, and for non-symmetrical search spaces, more than the required number of niches are initially identified. This implies that duplicate niches are found which impedes performance. Schoeman *et al* [82] proposed the parallel vector-based PSO to eliminate duplicate niches (refer to section 4.4.3).

4.4 Parallel Methods

This section discusses parallel niching methods using PSO. These methods are, *nbest* PSO in section 4.4.1, NichePSO in section 4.4.2, parallel vector-based PSO in section 4.4.3, and species-based PSO in section 4.4.4.

4.4.1 *nbest* PSO

The first parallel PSO niching approach was developed by Brits *et al* [14]. The proposed algorithm, known as *nbest* PSO, was designed to explicitly solve systems of unconstrained equations. Brits *et al* modified the standard fitness function of a particle such that the particle is rewarded when it moves closer to any possible solution.

1. Initialise the swarm by spawning $|S|$ particles. For each particle, find another random position nearby and find the position with the best fitness (lowest value of fitness function). This position is the particle's personal best, $\mathbf{y}_i$. Calculate the vector $\mathbf{v}_{pi}$, where
   $\mathbf{v}_{pi}(t) = \mathbf{y}_i(t) - \mathbf{x}_i(t)$
2. Initialize the best position in the neighbourhood, $\widehat{\mathbf{y}}_i$, to the personal best position of that neighbourhood, where the neighbourhood is the entire swarm.
3. Calculate the vector $\mathbf{v}_{gi}$ where $\mathbf{v}_{gi}(t) = \widehat{\mathbf{y}}_i(t) - \mathbf{x}_i(t)$
4. Calculate the dot product δ of each particle: $\delta = \mathbf{v}_{pi} \cdot \mathbf{v}_{gi}$
5. Set the niche radius to the distance between $\widehat{\mathbf{y}}_i(t)$ and the nearest particle with a negative dot product.
6. For each particle within the niche radius and with a positive dot product, update the particle's position using equation (2.18), $\mathbf{y}_i(t)$, and $\widehat{\mathbf{y}}_i(t)$, and the vectors $\mathbf{v}_{pi}$ and $\mathbf{v}_{gi}$ as well as their dot product δ.
7. Update the niche radius.
8. Repeat steps 5 and 7 until stopping criteria are met.
9. Mark all particles that have converged as *processed*. Maintain particles in the current niche.
10. Repeat steps 2 to 9 for the remainder of the swarm until all particles have been marked as *processed*.

Figure 4.2: Vector-based PSO

The general formula for a system of m equations is

$$\{g_\kappa(\mathbf{x}) = c : \kappa = 1, ..., m\} \tag{4.10}$$

where m is the number of equations in the system. This system can be rewritten as

$$\{f_\kappa(\mathbf{x}) = g_\kappa(\mathbf{x}) - c = 0 : \kappa = 1, ..., m\} \tag{4.11}$$

A solution is found by minimising the set of equations (4.11). The *nbest* PSO defines a new fitness function as

$$f(\mathbf{x}_i) = \min \{f_\kappa(\mathbf{x}_i)\} \tag{4.12}$$

The neighbourhood best of particle i is determined by first defining the topological neighbourhood of particle i. The topological neighbourhood is defined as the closest k particles to particle i, where k is user defined, and the closest particles are found by calculating the Euclidean distance between particle i and all other particles in the swarm. For each particle i, a set B_i is defined, which consists of the k closest particles to particle i at any given time t. The neighbourhood best particle, $\widehat{\mathbf{y}}_i$, is then calculated as

$$\widehat{\mathbf{y}}_i(t) = \frac{1}{k} \sum_{h=1}^{k} B_{ih} \tag{4.13}$$

where B_{ih} is the current position of the h^{th} particle in the neighbourhood B_i of particle i at time t [14]. Brits *et al* have shown that linearly decreasing k over time produces the most favourable results.

The *nbest* PSO has shown to be an effective niching technique to solve systems of unconstrained equations and can be applied to multiple solutions. Brits *et al* noticed that *nbest* PSO cannot maintain local optima when local optima occur relatively close to each other, as the *nbest* PSO particle neighbourhoods force the algorithm to always prefer solutions that have better fitness [14]. The NichePSO discussed in section 4.4.2 overcomes this problem.

1. Initialise the main particle swarm.
2. Train the main swarm's particles using one iteration of the cognition only model.
3. Update the fitness of each particle in the main swarm.
4. For each subswarm:
   - **a.** Train the subswarm's particles using one iteration of the GCPSO algorithm.
   - **b.** Update each particle's fitness.
   - **c.** Update the subswarm's radius.
5. Allow subswarms to absorb any particles from the main swarm that moved into it.
6. Search the main swarm for any particles that meet the converging criterion. If any is found, create a new subswarm with this particle and its closest neighbour.
7. Repeat from 2 until stopping criteria are met.

Figure 4.3: NichePSO algorithm

4.4.2 NichePSO

The NichePSO extends PSO's unimodal nature to a multimodal technique by developing and maintaining multiple subswarms that are grown from the initial swarm, where each subswarm represents a solution or niche [14] [15]. The NichePSO begins with a single main swarm, S, which is trained using the cognition-only model (refer to section 2.4.2). The cognition-only model allows particles to explore the search space without being influenced by other particles and thereby prevents particles from prematurely converging to a single solution. Swarm S is then inspected for particles that meet a converging criterion, which indicates whether particles have located potential niches. If a particle is found that satisfies the converging criterion, then that particle and its closest neighbouring particle are removed from swarm S to form a new subswarm. Subswarms refine and maintain the solution represented by the

niche. This is achieved through training each subswarm with the GCPSO (refer to section 2.4.2). The GCPSO is used to enforce convergence through inclusion of the social component. Subswarms are merged if they overlap or if the distance between the subswarms' best particles is less than a predefined threshold value. Particles from swarm S that have moved into a subswarm are absorbed into that subswarm. At the end of the search, each subswarm represents a solution or niche. Figure 4.3 gives a brief outline of the NichePSO and more detail of the algorithm is provided in the following subsections.

Converging Criterion

When the deviation of a particles fitness over a number of iterations is less than the niche threshold, δ, then the particle and its closest neighbour are removed from the main swarm to form a new subswarm. Euclidean distance is used to find the closest neighbour of a particle.

Absorption of Particles and the Merge Threshold

Only particles from the main swarm are absorbed into an already existing subswarm. A particle from the main swarm becomes part of the subswarm if that particle falls within the area defined by the radius of the subswarm. The radius of a subswarm is defined as the distance from the subswarm's best particle's position to the particle's position that is furthest from the best particle (in the same subswarm). Thus, calculating the radius involves calculating the distance from the best particle to every other particle in the subswarm. A main swarm particle is absorbed into a subswarm due to the following reasons [14]:

1. The inclusion of a particle that traverses the search space of an existing subswarm may expand the diversity of the subswarm, thereby leading to solutions with better fitness.

2. An individual particle moving towards a solution which is being explored by a subswarm, will make slower progress than what would have been the case had the social information been available.

Subswarms are merged if it appears that they are optimising the same solution. This is determined by detecting if the subswarms intersect. Two subswarms, s_1 and s_2, intersect when

$$\|\widehat{\mathbf{y}}_1 - \widehat{\mathbf{y}}_2\| < (R_1 - R_2) \tag{4.14}$$

where $\widehat{\mathbf{y}}_1$ and $\widehat{\mathbf{y}}_2$ are the best particles found in subswarm s_1 and s_2 respectively, and $\|\cdot\|$ calculates the length of the vector, and R_1 and R_2 are the radii of the respective subswarms.

A subswarm that forms around a potential solution is likely to have a radius that approximates 0 as the subswarm converges on the solution. If there are more than one subswarm converging on the same solution with radii approaching 0, then it is possible that equation (4.14) will fail to detect multiple subswarms optimising the same solution. To resolve this problem a merge threshold, μ, is used in which case two subswarms merge when

$$\|\widehat{\mathbf{y}}_1 - \widehat{\mathbf{y}}_2\| < \mu \tag{4.15}$$

Updating the Particle Positions

When a particle is trained on the cognition-only model, its personal best solution, current position and the inertia component are used to change its position (refer to section 2.4.2). Using the cognition-only model for the main swarm results in a larger area of the search space to be explored and facilitates niche formation. While using the GCPSO for the subswarms exploits the search space so that optima can be efficiently and accurately found.

Stopping Criteria

A subswarm is considered to have converged once it has located a solution and maintained it for a number of iterations. To determine if a subswarm has maintained a solution for a number of iterations, the change in position of the best particle of a subswarm can be tracked over a number of iterations and if no noticeable deviation

is observed then the subswarm may be considered to have converged. The algorithm terminates when all subswarms have converged or if a maximum number of iterations has been exceeded.

Pitfalls

There is no restriction on where particles from the main swarm will form subswarms, nor is there a restriction on the number of particles in a subswarm. This allows for multiple subswarms to form around the same solution which will eventually be merged into one large subswarm, leading to a single solution. Particles may also be prematurely absorbed into a subswarm, which will result in much of the search space to be left unexplored. Once a subswarm has been formed, it cannot be separated into smaller subswarms and it will eventually converge to one solution. This arrangement can lead to the unnecessary traversal of previously explored areas while unexplored regions may offer alternative solutions. The consequence is an explosion in the number of particles that are needed to find solutions in higher dimensions or in large domains with many solutions [16]. Brits *et al* has empirically estimated that the relationship between the swarm size and the number of solutions to be [14][16]

$$|S| = c \cdot q^a \tag{4.16}$$

where c is a user-defined constant, $1 \leq a \leq 2$ and q is the number of solutions.

The above problem is related to the diversity of the main swarm. If the diversity of the main swarm is poor, then the number of optima that can be found decreases and the number of particles required increases. A swarm with a high diversity is able to explore the search space more effectively and requires fewer particles.

4.4.3 Parallel Vector-Based PSO

The process of identifying niches is similar to the vector-based PSO (discussed in section 4.3.3), however all particles are updated simultaneously. Subswarms that converge on the same niche are merged once the distance between the subswarms is less than a certain threshold, ϵ. This threshold is a problem dependent parameter

that has been introduced to facilitate the merging of subswarms. The distance between two subswarms is calculated as the distance between the subswarm's global best paticle's position. Only those particles in the subswarm that are nearer than ϵ to the global best of the other subswarm will be merged. Schoeman *et al* [82] showed that the parallel vector-based approach performs well on a number of one and two dimensional functions, and has a significant performance improvement over the vector-based PSO. Furthermore, duplicate niches are reduced because the parallel vector-based PSO merges subswarms when the subswarms converge on the same niche.

4.4.4 Species-Based PSO

Species-based PSO [69] forms neighbourhoods determined by the location of species seeds. A species seed is the best fit individual within a predetermined radius r_s. All particles with a Euclidean distance less than r_s to the species seed, belong to the same species. The Euclidean distance between particles $\mathbf{x}_i$ and $\mathbf{x}_j$ is calculated as

$$d(\mathbf{x}_i, \mathbf{x}_j) = \sqrt{\sum_{k=1}^{n} (x_{ik} - x_{jk})^2} \tag{4.17}$$

where $\mathbf{x}_i = (x_{i1}, x_{i2}, ..., x_{in})$ and $\mathbf{x_j} = (x_{j1}, x_{j2}, ..., x_{jn})$

Particles of the same species set the species seed as their neighbourhood best, which allows particles of the same species to converge on a similar solution. Because species are located in parallel, multiple optima can be located in parallel.

A set of species seeds, S_{seed}, is determined by first sorting the particles in descending order of fitness in a list, L_{sorted}, and initialising S_{seed} to an empty set. Each particle in L_{sorted} is marked as *unprocessed*. The next step in determining the species seeds is to get the first *unprocessed* particle from L_{sorted} and process the particle. The particle is processed by determining if there are any particles in S_{seed} that is within a Euclidean distance of r_s. If a particle is not found, then the particles is added to S_{seed}; otherwise the algorithm continues onto the next step. The particle is marked as *processed* and the next *unprocessed* particle is fetched. The process of

1. Let L_{sorted} = all particles sorted in decreasing order of fitness.
2. Let $S_{seed} = \{\}$
3. While L_{sorted} contains *unprocessed* particles do
   - **a.** Get the first *unprocessed* particle $\mathbf{x}_L$ in L_{sorted}
   - **b.** Let $found = FALSE$
   - **c.** For each particle $\mathbf{x}_S$ in S_{seed} do
     - **c.1** If $d(\mathbf{x}_S, \mathbf{x}_L) \leq r_s$ then
       - **c.1.2** $found = TRUE$
       - **c.1.3** Goto step *d*.
   - **d.** If (not $found$) then let $S_{seed} = S_{seed} \cup \{\mathbf{x}_L\}$
   - **e.** Mark $\mathbf{x}_L$ as *processed*.

Figure 4.4: Algorithm for determining species seeds

marking particles as *processed* is continued until L_{sorted} does not contain any particles that are marked as *unprocessed*. The resulting set, S_{seed}, contains particles, i, that are the fittest within a radius r_s of particles i. The particles in S_{seed} (referred to as the species seeds) form the neighbourhood best of the particles that are within the radius r_s to the species seed. The species-based PSO updates each particle's velocity according to equation (2.21) and position according to equation (2.18). The species-based PSO continues to determine the species seeds, and update the particles velocities, and positions until a maximum number of iterations is reached. Figure 4.4 summarises the algorithm for selecting the species seeds, and figure 4.5 summarises the species-based PSO algorithm.

Species seeds are determined at every iteration of the algorithm, which prevents the formation of duplicate species. Species-based PSO has shown to be effective for low dimensional problems and has been successfully applied in dynamic environments [69].

1. Generate an initial swarm with randomly generated particles.
2. Evaluate all particles in the swarm.
3. Sort all particles in descending order of their fitness values.
4. Determine the species seeds (refer to figure 4.4).
5. Assign each species seed identified as the neighbourhood best to all particles identified in the same species.
6. Adjust each particle's velocity according to equation (2.21) and each particle's positions according to equation (2.18).
7. If termination criteria is not met, go to step 2.

Figure 4.5: Species-based PSO algorithm

4.5 Conclusion

The standard PSO was shown to be inappropriate for finding multiple solutions. Several modifications to the standard PSO resulted in the development of PSO niching algorithms that can be categorised as either sequential or parallel. Objective function stretching is a sequential algorithm that modifies the search space when a minimum is found. Objective function stretching removes all other minima that are of a larger value than the minima found and then continues searching for the next minimum. Objective function stretching is not a true multimodal algorithm as it does not find all local optima. Another sequential algorithm discussed is the vector-based PSO, which dynamically classifies particles as belonging to a niche. That is, the vector-based PSO does not use a niche radius to determine if particles belong to the same niche. However, the vector-based PSO does not ensure that unique solutions are reported.

Parallel methods discussed included parallel vector-based PSO, *nbest* PSO, species-based PSO and NichePSO. Parallel vector-based PSO extends the vector-based PSO to reduce the number of duplicate solutions reported. The *nbest* PSO is used to solve systems of unconstrained linear equations, and general multimodal functions. However, *nbest* PSO does not maintain local optima when the local optima are situated relatively close to each other. Species-based PSO defines species seeds at each it-

eration of the algorithm. A species seed is the best fit particle within a predefined radius, r_s. All particles within a distance of r_s to the species seed, are grouped together to form a neighbourhood with the species seed set as the neighbourhood best. The neighbourhoods then converge on their respective optima in parallel. The NichePSO used the concepts of forming subswarms from a larger swarm (the main swarm), merging subswarms, and absorbing main swarm particles into subswarms. A subswarm is created when a particle from the main swarm begins to converge on a solution. The new subswarm consists of the converging particle and its closest neighbour. The NichePSO does not scale very well as the number of particles explodes for domains with many solutions. The next chapter presents a variation of the NichePSO to improve scalability.

Chapter 5

Derating NichePSO

GA niching techniques have shown to be adequate for problems of small dimensions, but do not scale well for highly multimodal functions or large dimensional problems. The NichePSO successfully countered the pitfalls of GA niching techniques, though it too experiences problems in large search spaces and highly multimodal functions. This chapter presents the derating NichePSO which is an attempt to improve NichePSO scalability by modifying the search space such that previously traversed parts of the search spaces are not re-explored.

5.1 Introduction

The NichePSO has shown to accurately and efficiently locate multiple solutions in a multimodal search space [14][15][16] if the search space is not highly multimodal. That is, if there are many optima to locate then the number of particles required for the NichePSO increases exponentially [14]. The exponential relationship between the number of particles and the number of solutions is mainly due to the following reasons:

- The radius of a subswarm s_1 is the distance between the best particle in s_1 and another particle in s_1 that is furthest from the best particle. If this radius is large, particles from the main swarm are easily absorbed into s_1. This effectively reduces the possible number of solutions that can be found because

diversity of the main swarm, S, is reduced by having more particles exploit an already found niche (refer to sections 4.4.2 and 6.4).

- A new subswarm is created when a particle, i_1, from the main swarm satisfies the niching criteria. The new subswarm consists of i_1 and the closest particle to i_1 in the main swarm, represented here as i_2. The criteria for selecting i_2 does not consider the direction in which i_2 is moving. That is, i_2 could be moving in the opposite direction of i_1 and possibly be converging on another niche. Furthermore, it is possible that i_2 may have a better fitness value than i_1. Because social information is used when optimising the subswarm, the velocity of i_2 will be added in the velocity update of i_1. Thus i_1 will be pulled towards i_2 and therefore losing the niche that i_1 originally identified.

- An inaccurate merge threshold implies that subswarms that satisfy the merging criteria, but are potentially optimising different niches, will intersect and subsequently be merged to produce a new subswarm. For example, if subswarm s_2 is optimising a solution located at position B and subswarm s_1 is optimising a solution located at position A, and if s_1 and s_2 are merged to form a new larger subswarm s_3, then it is possible that either the solution at position A or B is lost, or it might be the case that the newly formed subswarm s_3 will converge to an entirely different solution other than A or B, in which case both solutions are lost.

- Much of the search area is re-explored as subswarms do not communicate their knowledge about the search space to other subswarms. Consequently, more particles are required to explore larger areas of the search space.

The above points can be classified into: absorption of particles from the main swarm into a subswarm, formation of subswarms, merging of subswarms, and the exploration ability of the subswarms.

The formation of subswarms problem can be addressed by determining if two particles are moving in the same direction. However, it can only be concluded that two particles are converging on the same niche if the search space is guaranteed

to be unimodal or there exists prior knowledge of the inter-niche distance that can be used to group particles. If the search space is multimodal, then although two particles may be moving in the same direction, the particles maybe converging on different niches, and grouping the two particles in a subswarm will be incorrect. Subswarms can also be formed by grouping particles that are within a Euclidean distance of a niche radius from a particle, i, that satisfies the convergence criteria, which is similar to the species-based PSO [69] (refer to section 4.4.4). Furthermore, the niche radius from i can be determined by finding the nearest particle from i with a negative dot product, as done in the vector-based PSO [83] (refer to section 4.3.3). The niche radius found may be too large, which will incorrectly group many particles into a niche. If the niche radius is defined prior to the search process, then knowledge of the minimum inter-niche distance must be assumed, which may result in some niches not being found if the niche radius is incorrectly chosen.

The merging of subswarms problem can be addressed by adjusting the merge threshold, μ. Brits [14] analysed the sensitivity of the NichePSO to changes in μ. It was noted that μ should not be greater than the lowest inter-niche distance, which is similar to the assumptions of the inter-niche distance in fitness sharing [36] and the niche radius of the SNT [8]. However this requires prior knowledge of the search space which is not always available.

This thesis does not attempt to solve all problems related to the exponential increase in the number of particles caused by increases in dimension of the problem being optimised. Rather, the focus is on reducing the number of times the search space is re-explored. This problem can be addressed by forcing particles to explore unvisited regions of the search space. One way of achieving this objective is to improve diversity between subswarms. Diversity between particles in a swarm has been investigated by Løvberg *et al* [58] [57], and Blackwell *et al* [12][11]. Løvberg *et al* presented hybrid models of the PSO and a GA, adapting the particles to include self organised criticality and extending particles with a radius so that they can collide with other particles. Blackwell *et al* introduced an electrostatic energy to the PSO to repel particles from each other, thereby increasing diversity.

This chapter proposes a hybrid approach that uses SNT with NichePSO to improve the exploration abilities of the NichePSO algorithm.

SNT communicates information of located optima to individuals in the population by modifying the original fitness function (refer to section 3.2.2). Using the NichePSO as the search algorithm for SNT (instead of a GA) improves the exploration ability of subswarms, as modifications to the fitness function will allow subswarms to explore unvisited regions of the search space for potential solutions. As discussed in section 3.2.2, modifications to the fitness function has the disadvantage of introducing false optima and removing other optima in the search space. A possible solution to this problem, as suggested by Beasely *et al* [8], is to perform a local search using the unmodified fitness function to improve accuracy once a potential solution has been located. The derating NichePSO, proposed in this chapter, makes use of this suggestion. The search for candidate solutions is performed by the main swarm using the modified fitness function, while candidate solutions are refined by subswarms using the original, unmodified fitness function. The details of the derating NichePSO algorithm are discussed in the following sections.

5.2 The Derating NichePSO Algorithm

The derating NichePSO algorithm (summarised in figure 5.1) has two phases. The first phase forms subswarms around potential solutions or niches using the modified fitness function, M. The second phase exploits the search area using the raw fitness function, F (or unmodified fitness function), to allow subswarms to converge on true optima and to improve the fitness of the particles. Both phases are discussed in more detail in this section.

In figure 5.1, M_t is the modified fitness function at execution t of the derating NichePSO. With SNT, M_{t+1} is updated using a single vector $\widehat{\mathbf{y}}_t$ which represents the location of the best individual in the population at execution t. The modified fitness of an individual, $\mathbf{x}$, at $t+1$ is calculated as,

$$M_{t+1} \equiv M_t(\mathbf{x}) * G(\mathbf{x}, \widehat{\mathbf{y}}_t) \tag{5.1}$$

1. Let $t = 0$ and $M_t \equiv F$
2. Initialise the NichePSO.
3. Phase 1: Repeat until the main swarm has no particles or the maximum number of iterations has been exceeded:
    a. Train the main swarm's particles using one iteration of the cognition-only model in the modified search space.
    b. Update the fitness of each particle in the main swarm.
    c. Search the main swarm for any particles that meet the partitioning criteria. If any is found, create a new subswarm with this particle and its closest neighbour.
4. Phase 2: Repeat until all subswarms have converged or the maximum number of iterations has been exceeded:
    a. For each subswarm:
        i. Train the subswarm's particles using one iteration of the GCPSO algorithm in the unmodified fitness function.
        ii. Update each particle's fitness.
        iii. Update the subswarm's radius.
    b. Allow subswarms to absorb any particles from the main swarm that moved into it.
    c. Search the main swarm for any particles that meet the partitioning criteria. If any are found, create a new subswarm with this particle and its closest neighbour.
5. For each subswarm s, update the modified fitness function M_{t+1} using the location of the best particle in subswarm s.
6. $t = t + 1$
7. If no convergence goto step 2.

Figure 5.1: Derating NichePSO algorithm

The NichePSO has the ability to find multiple solutions which are represented by the best particle of each subswarm. To cater for multiple solutions found by the NichePSO, the modified fitness of particle $\mathbf{x}$ at $t+1$ is calculated as,

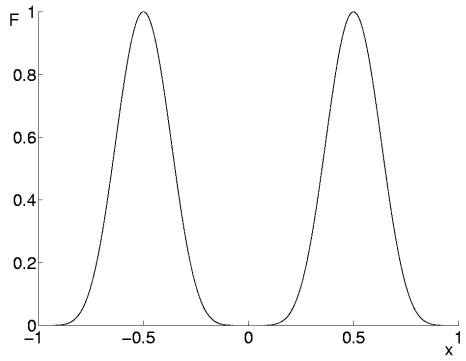
$$M_{t+1} \equiv M_t(\mathbf{x}) * G(\mathbf{x}, \widehat{\mathbf{y}}_{t,1}) * G(\mathbf{x}, \widehat{\mathbf{y}}_{t,2}) * ... * G(\mathbf{x}, \widehat{\mathbf{y}}_{t,k}) \tag{5.2}$$

where k is the number of subswarms, and $\widehat{\mathbf{y}}_{t,k}$ denotes the best particle from subswarm k at execution t of the derating NichePSO.

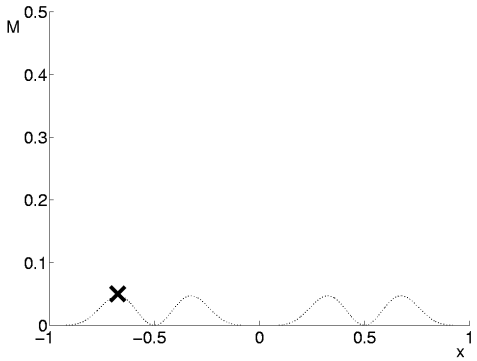
To illustrate the first and second phases, consider the raw fitness function landscape as shown in figure 5.2(a), and the modified fitness function landscape as shown in figure 5.2(b). F has two global optima at $x = -0.5$ and $x = 0.5$. Figure 5.2(b) indicates that both optima have been found, with a depression superimposed on F (to form the modified fitness function M) which prevents particles from the main swarm re-traversing that area. Note that M now has four false optima.

The first phase searches the landscape defined by M for potential optima. This search is performed solely by iterating the main swarm, which uses the cognition only model. The lack of social information in the main swarm prevents particles from prematurely converging to a single solution. Assume that, during phase 1, a false optimum at the position indicated by a cross in figure 5.2(b) is discovered by a particle in the main swarm, and that a subswarm is formed in that area. To prevent this false optimum to be exploited, the subswarm continues its search in the original landscape defined by F in phase 2, and not in the modified landscape defined by M. The newly formed subswarm does not search the landscape defined by M. Rather, the subswarm searches the landscape defined by F (this is only done in the second phase). The particles that converged to a false optimum will therefore be refined to converge to the true optimum as defined by F.

The formation of the subswarms causes the number of particles in the main swarm to decrease. It thus stands to reason that the first phase is complete when the main swarm does not contain any more particles. Alternatively a maximum limit on the number of iterations on the main swarm can be used. The maximum limit is a



a



b

Figure 5.2: (a) Raw fitness landscape, F. (b) Modified fitness landscape, M.

safety precaution to prevent lengthy iterations of the main swarm. At the end of the first phase, a number subswarms will have formed around potential optima based on the modified landscape. If no subswarms exist, then no new potential optima have been located. The algorithm terminates if this situation occurs for consecutive iterations.

The niches or potential optima that have been located in the modified search space are not accurate and are therefore required to be further refined. In the second phase, the subswarms refine potential solutions with respect to the original landscape (defined by F), using GCPSO. Applying GCPSO with respect to the original landscape allows subswarms to converge to true optima, and not false optima formed in the modified space. If there are more than one subswarm converging on the same optimum, then subswarms are merged (following the NichePSO rules for merging subswarms). Particles that may exist within the main swarm are absorbed into the enclosing subswarm (following the NichePSO rules for absorption of particles into subswarms).

At the end of the second phase, the subswarms have converged on true optima. The modified fitness function is now updated with depressions at the points of optima located by the subswarms. A new set of particles are generated for the main swarm, and the first and second phases are again executed until a stopping criterion has been satisfied.

Through modifying the search space, the particles in the main swarm are forced to explore unvisited areas of the search space, and thus preventing the re-exploration of previously traversed areas. This improves the number of times the search space is re-explored by the NichePSO.

5.3 Removing Duplicate Solutions

The derating NichePSO takes no precautions to ensure that duplicate solutions are removed in the algorithm. To remove duplicate solutions, an additional merging procedure is introduced and a history of subswarms that have converged are stored for the entire algorithm. The history of converged subswarms are stored in a set $\mathbf{X}$.

The subswarms in $\mathbf{X}$ are continuously merged at each execution of the NichePSO with the subswarms of the current execution. After the subswarms in $\mathbf{X}$ are merged with the subswarms of the current execution, the search space is modified at every solution in $\mathbf{X}$. The derating NichePSO algorithm with duplicates removed is summarised in figure 5.3.

5.4 Stopping Criteria

The derating NichePSO uses three different stopping criteria:

- Stopping criteria for the first phase: The main swarm continues to search in the modified landscape until the main swarm does not have any more particles or until a specified maximum number of iterations has been exceeded.
- Stopping criteria for the second phase: Subswarms continue to search in the unmodified or original fitness landscape until all subswarms have converged (refer to section 4.4.2) or the maximum number of iterations has been exceeded.
- Stopping criteria for the derating NichePSO: Searching is continued until all (or a specified number of) solutions are found, or until a specified maximum number of executions of the NichePSO has been exceeded.

If the number of solutions is not known, then an alternative stopping criterion is to track the solutions found, and to terminate the algorithm if no new solution is found for a number of consecutive iterations. This suggests that there are no more solutions in the search space to locate.

5.5 Conclusion

This chapter introduced the derating NichePSO. The main objective of the derating NichePSO is to reduce the number of times the NichePSO re-explores the search space. By achieving this, the large number of particles required for highly multimodal search spaces can be decreased. To improve the exploration ability of the

1. Let $t = 0$ and $M_t \equiv F$
2. Initialise the NichePSO.
3. Phase 1: Repeat until the main swarm has no particles or the maximum number of iterations has been exceeded:
   a. Train the main swarm's particles using one iteration of the cognition-only model in the modified search space.
   b. Update the fitness of each particle in the main swarm.
   c. Search the main swarm for any particles that meet the partitioning criteria. If any is found, create a new subswarm with this particle and its closest neighbour.
4. Phase 2: Repeat until all subswarms have converged or the maximum number of iterations has been exceeded:
   a. For each subswarm:
      i. Train the subswarm's particles using one iteration of the GCPSO algorithm in the unmodified fitness function.
      ii. Update each particle's fitness.
      iii. Update the subswarm's radius.
   b. Allow subswarms to absorb any particles from the main swarm that moved into it.
   c. Search the main swarm for any particles that meet the partitioning criteria. If any are found, create a new subswarm with this particle and its closest neighbour.
5. Set $\mathbf{X} = \mathbf{X} \cup \{\text{NichePSO subswarms}\}$
6. Merge subswarms in $\mathbf{X}$, according to NichePSO rules.
7. Set $M_{t+1} \equiv F$
8. For each subswarm in $\mathbf{X}$, update the modified fitness function M_{t+1} using the location of the best particle in subswarm s.
9. $t = t + 1$
10. If no convergence goto step 2.

Figure 5.3: Derating NichePSO algorithm (duplicates removed)

NichePSO, the NichePSO is combined with the SNT. The combination results in an algorithm with two phases, referred to as the derating NichePSO.

The derating NichePSO maintains two fitness functions, the modified fitness function and the unmodified fitness function. During the first phase, the main swarm searches in the modified landscape and subswarms are formed around potential solutions or niches. The first phase terminates when the main swarm is depleted of particles or when a maximum number of iterations has been exceeded. During the second phase, the subswarms search the unmodified landscape, which allows subswarms to converge on optima. The second phase completes when all subswarms have converged or when a maximum number of iterations has been exceeded. After the second phase, a depression is created in the modified landscape at the position where subswarms have converged on a solution. The depressions introduced into the modified landscape forces particles in the main swarm to explore other areas of the search space, thus improving the exploration ability of the NichePSO. To avoid duplicate solutions, a history of converged subswarms are stored in a set for the duration of the derating NichePSO algorithm. These subswarms are merged with the subswarms of the current NichePSO execution. The history set therefore contains subswarms which represent a unique set of solutions obtained by the derating NichePSO.

The following chapter presents an empirical analysis and comparison of the derating NichePSO, NichePSO, DC and SNT.

Chapter 6

Empirical Analysis

The flapping of a single butterfly's wing today produces a tiny change in the state of the atmosphere. Over a period of time, what the atmosphere actually does diverges from what it would have done. So, in a month's time, a tornado that would have devastated the Indonesian coast doesn't happen. Or maybe one that wasn't going to happen, does.
Ian Stewart, Does God Play Dice? The Mathematics of Chaos, pg. 141 [91]

This chapter presents experimental results to illustrate the efficacy of the derating NichePSO. The derating NichePSO is compared with NichePSO, SNT and DC on a benchmark of classical multimodal functions. A sensitivity analysis of control parameters is done, as well as a scalability study.

The chapter is organised as follows: Section 6.1 summarises the experimental procedure. The performance of the derating NichePSO is evaluated in section 6.2 and compared with other approaches. Section 6.4 provides a scalability study, while a parameter analysis is summarised in section 6.5. The chapter is concluded in section 6.6.

6.1 Evaluation Procedure

The performance of a search algorithm that produces a single solution can be evaluated by calculating the average accuracy over a number of simulations, where a simulation is a single execution of the algorithm until termination. The average accuracy, $\overline{Accuracy}$, can be calculated as

$$\overline{Accuracy} = \frac{\sum_{e=1}^{E} |f'(\mathbf{x}_e)|}{E} \tag{6.1}$$

where function f' is the derivative of the function f, $\mathbf{x}_e$ represents the solution for simulation e and E is the total number of simulations that the algorithm was executed. The search algorithm performs well if the value of $\overline{Accuracy}$ is approximately 0. NichePSO, SNT, DC and derating NichePSO produce multiple solutions for each simulation and it is possible that the number of solutions found in each simulation will not be the same. Furthermore, the solutions found by each algorithm may be different. Such algorithms are referred to as multi-solution or niching algorithms. The average accuracy of a multi-solution algorithm can be calculated as:

$$\begin{aligned}
\overline{M - Accuracy} &= \left(\left[\sum_{j=1}^{k_1} |f'(\mathbf{x}_{1,j})| \right] / k_1 + \left[\sum_{j=1}^{k_2} |f'(\mathbf{x}_{2,j})| \right] / k_2 + \dots \right. \\
&\quad \left. + \left[\sum_{j=1}^{k_E} |f'(\mathbf{x}_{n,j})| \right] / k_E \right) / E \\
&= \sum_{e=1}^{E} \left(\left[\sum_{j=1}^{k_e} |f'(\mathbf{x}_{e,j})| \right] / k_e \right) / E
\end{aligned} \tag{6.2}$$

where $\mathbf{x}_{e,j}$ is the j^{th} unique solution for simulation e, and k_e is the number of unique solutions found by simulation e. Equation (6.2) is the summation of the average accuracy for each simulation, for all solutions found for that simulation.

The average number of unique solutions found over E simulations is calculated as

$$\overline{Solutions} = \sum_{e=1}^{E} k_e / E \tag{6.3}$$

Table 6.1: DC parameter values

Parameter	**Value**
Number of Individuals	50
Selection Procedure	Tournament Selection
Mutation Rate	0.9 (decreased to 0)
Reproduction Probability	0.5
Number of Offspring created per iteration	10
Stopping Criteria	Maximum Iterations = 1000

Table 6.2: SNT parameter values

Parameter	**Value**
Number of Individuals	50
Selection Procedure	Tournament Selection
Mutation Rate	0.9 (decreased to 0)
Reproduction Probability	0.5
Number of Offspring created per iteration	10
Niche Radius	0.1
Alpha (α)	0.8
Stopping Criterion for GA	Maximum Iterations = 500
Stopping Criterion (sequential executions)	Maximum executions = 10

Table 6.3: NichePSO parameter values

Parameter	**Value**
Number of Particles	500
Merge Threshold	0.01
Niche Threshold	0.00001
Stopping Criterion	Maximum Iterations = 1000
c_1 and c_2	1.2
ϕ	0.7

Table 6.4: Derating NichePSO parameter values

Parameter	**Value**
Number of Particles	50
Merge Threshold	0.01
Niche Threshold	0.00001
c_1 and c_2	1.2
ϕ	0.7
Solution Threshold for Phase 2	0.000001
Stopping Criterion (sequential executions)	Maximum executions = 10
Stopping Criteria (for Phase 1)	Maximum Iterations = 500 or Main Swarm has no more particles
Stopping Criteria (for Phase 2)	Maximum Iterations = 500 or subswarm has converged.
Niche Radius	0.1
Alpha (α)	0.8

The *Success*% is another measure of performance for niching algorithms, which indicates the percentage of subswarms or individuals from the entire swarm or population that have converged on solutions over all simulations. For this study a subswarm or individual with an accuracy between $-0.1 < f'(\mathbf{x}) < 0.1$ is considered to have converged. If the *Success*% is low, then most individuals are still exploring the search space, and the best solution per subswarm has not necessarily converged yet.

The parameters used for DC, SNT, NichePSO and derating NichePSO are summarised in tables 6.1 to 6.4 respectively.

Both SNT and derating NichePSO are sequential methods which repeatedly execute the search algorithm. Each repetition is known as a run or execution. For both the SNT and derating NichePSO the corresponding search algorithm (GA for SNT, and NichePSO for derating NichePSO) is executed 10 times. However, this means that SNT can only find a maximum of 10 solutions, whereas the derating NichePSO is not purely restricted to the number of executions (refer to section 6.4 for a discussion on the maximum number of solutions that the derating NichePSO can find). For each execution of the GA, a maximum of 500 iterations was used as the stopping criterion for the GA. Similarly, for each execution of the NichePSO, a maximum of 500 iterations was used during phase 1 (or until there were no particles in the main swarm) and a further maximum of 500 iterations was used for phase 2 (or until all subswarms have converged). Subswarms converged if the standard deviation of the best particle's position in the subswarm was less than the niche threshold.

Derating NichePSO repeated the NichePSO 10 times with 50 particles for each execution, which means that a total of 500 particles were used to explore the search space. For a fair comparison, the NichePSO on its own used 500 particles as well. The merge threshold for both NichePSO and derating NichePSO was chosen such that it is less than the smallest distance between optima in all evaluation functions used. The merge threshold was kept constant at 0.01 for all search spaces to limit the amount of prior knowledge required. The niche threshold, δ, was arbitrarily chosen as Brits [14] found that the performance of the NichePSO is insensitive to

changes to δ. The only penalty paid for using a small δ (less than 0.01) is an added computational cost, as the particle will take longer to identify a niche since the particle is now required to maintain its position with very small deviation. Similar values for the acceleration constants, c_1 and c_2, and inertia, ϕ, were used for both NichePSO and derating NichePSO. The values of c_1, c_2, and ϕ were arbitrarily chosen to satisfy the heuristics, $c_1 + c_2 \leq 4$ and $\phi \geq \frac{1}{2}(c_1 - c_2) - 1$ discussed in section 2.4.2.

DC assumes prior knowledge of the height of the peaks to identify solutions. This assumption was neglected by this study, which resulted in DC producing solutions that were highly inaccurate as many of the individuals did not converge. To clearly illustrate the DC solutions it was decided that only 50 individuals would be used. The number of individuals for SNT was also kept at 50 which is equal to the number of particles for the derating NichePSO. Tournament selection was used for both SNT and DC, as it is a popular and efficient selection procedure for GAs. To improve diversity among individuals early in the search, the mutation rate was set at 0.9 and exponentially decreased to enable individuals to converge on a solution in the later stages of the search.

Using equations (6.2) and (6.3) the average accuracy, and average number of solutions were calculated for 30 simulations per evaluation function. Results reported are averages and standard deviations over the results obtained from these simulations.

The following two-dimensional functions were used for this study:

- Ackley, with 1 global maximum and many local maxima:

$$f(x_1, x_2) = 20 + e - 20 \cdot e^{-0.2\sqrt{\frac{x_1^2 + x_2^2}{2}}} - e^{\frac{\cos(2\pi x_1) + \cos(2\pi x_2)}{2}}, \tag{6.4}$$

  where $-30 \leq x_1, x_2 \leq 30$.

- Griewank, with 1 global maximum and many local maxima:

$$f(\mathbf{x}) = \frac{1}{4000}\sum_{i=1}^{2} x_i^2 - \prod_{i=1}^{2} \cos\left(\frac{x_i}{\sqrt{i}}\right) + 1, \tag{6.5}$$

where $-28 \leq x_1, x_2 \leq 28$.

- Michalewicz, with 2 global maxima and an infinite number of local maxima:

$$f(x_1, x_2) = \sin(x_1) \cdot \sin^{20}\left(\frac{x_1^2}{\pi}\right) + \sin(x_2) \cdot \sin^{20}\left(\frac{2x_2^2}{\pi}\right), \tag{6.6}$$

where $0 \leq x_1, x_2 \leq \pi$.

- Rastrigin, with 1 global maximum and many local maxima:

$$f(\mathbf{x}) = 20 + \sum_{i=1}^{2} x_i^2 - 10\cos(\pi x_i), \tag{6.7}$$

where $-10 \leq x_1, x_2 \leq 10$.

- Ursem F1 with 1 global maximum and 1 local maximum:

$$f(x, y) = \sin(2x - 0.5\pi) + 3\cos(y) + 0.5 \cdot x, \tag{6.8}$$

where $2.5 \leq x \leq 3, -2 \leq y \leq 2$.

6.2 Results

Figures 6.1 to 6.5 display the search space for each function, as well as the contour map of each search space. The contour map is a visual aid for the reader to help identify optima in the search space. For each algorithm, a total of 30 simulations

Table 6.5: Ackley results

Algorithm	$\overline{Solutions}$	**Std Dev**	$\overline{M - Accuracy}$	**Std Dev**	$Success\%$
DC	18.540	2.363	2.517	0.034	14.028
SNT	10.000	0.000	3.297	1.960	2.334
NichePSO	94.800	5.130	0.055	0.004	98.699
Derating NichePSO	241.634	2.076	0.052	7.859E-4	94.689

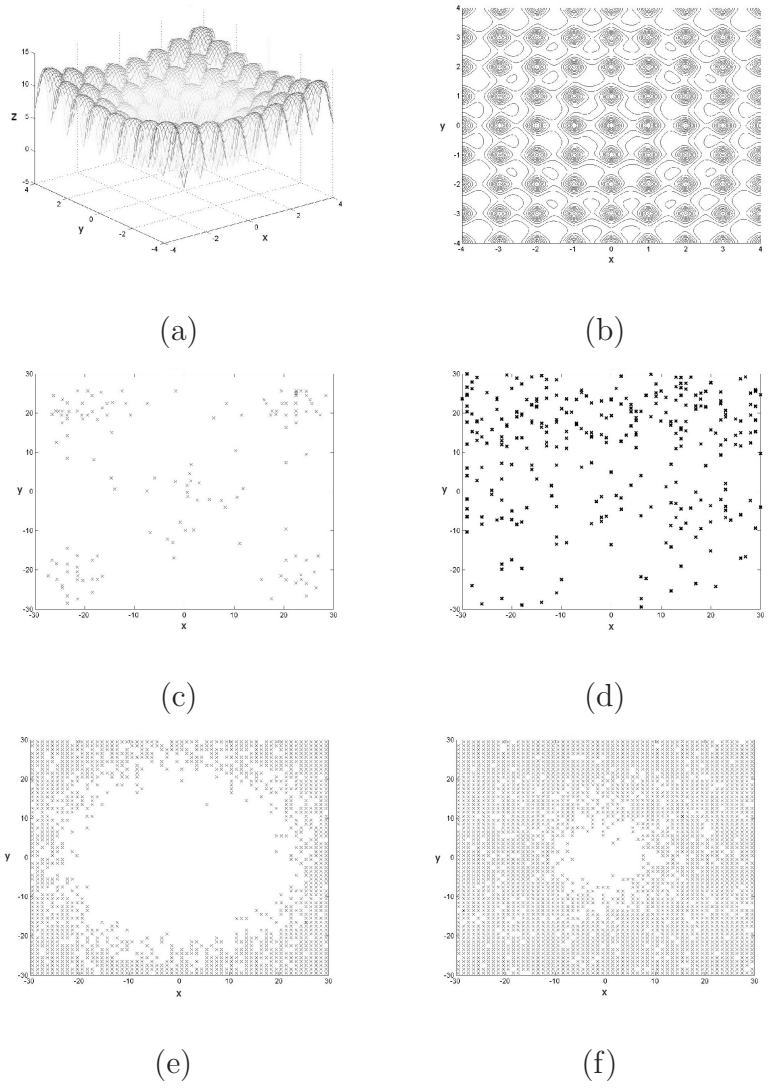
were conducted for each function and the results from each function (for the respective algorithm) were merged together to give a single set of unique solutions. These sets are plotted in separate graphs, which can be compared to the contour maps to illustrate how well the algorithms have explored the search space and how good the located solutions are. It is important to note that these figures display solutions for all 30 simulations. Two solutions were merged if the Euclidean distance between them was less than 0.1, which is larger than the merge threshold used in the NichePSO and derating NichePSO, but smaller than the inter-niche distances for the evaluation functions used. The merged solution is the average of the two solutions. The merged solution is compared to other solutions to determine if more solutions could be merged.

The Ackley function is highly multimodal and for clarity the full search space ($-30 \leq x_1, x_2 \leq 30$) is not displayed in figure 6.1. The large search space tested the exploration capabilities of the algorithms, while the large number of optima tested the scalability of the algorithms. The optima are distributed in the search space such that the worst optima are located at the centre, and better optima are located at the edges of the search space.

Table 6.5 contains the average number of peaks found, the average accuracy and the $Success\%$ for each algorithm (calculated using equation (6.2)).

The following observations can be made:

- DC found optima near the centre of the map, but much of the search space has not been explored. The accuracy is not as high compared with NichePSO and derating NichePSO and the low percentage of converged individuals show



(a) (b)

(c) (d)

(e) (f)

Figure 6.1: (a)Ackley 3D plot (b) Ackley contour map (c) DC results (d) SNT results (e) NichePSO results (f) derating NichePSO results

that many of the solutions have not converged within the maximum number of iterations. DC is unable to maintain niches in the centre of the search space as most individuals are located near good optima on the corners of the search space. That is, DC solutions concentrates on the boundaries of the search space where better solutions exist.

- SNT found the least number of optima and with the worst accuracy from all other search algorithms in this evaluation. The percentage of successfully converged individuals is extremely low, which means that individuals mostly failed to converge. Individuals in SNT are distributed throughout the search space, showing that SNT is able to diversify the population through modifying the landscape of the fitness function, at the cost of inaccuracy.

- NichePSO was unable to find most of the optima located at the centre of the search space because many of the particles have prematurely converged to better optima at the corners of the search space. That is, particles move in the direction of better optima or niches that have previously been discovered. This shows that the NichePSO is able to effectively maintain the niches found, but fails to locate other niches in the search space as particles do not know where niches have already been found. The NichePSO has found significantly more solutions and has a better average accuracy than DC and SNT. The low standard deviation in the average number of solutions implies that the NichePSO has consistently found the same number of solutions for each simulation. Furthermore, NichePSO has produced the best overall $Success\%$ and thus many particles have converged on a solutions.

- In the early stages of the derating NichePSO search process, the particles converged to better optima at the edges of the search space. However, with the ability to modify the fitness function landscape, more of the search space was explored resulting in the discovery of more niches than the NichePSO. Although the search space was modified, the derating NichePSO has produced an accuracy measure, and a $Success\%$ value that is similar to that of the NichePSO. However, the derating NichePSO has found significantly more so-

Table 6.6: Griewank results

Algorithm	$\overline{Solutions}$	**Std Dev**	$\overline{M - Accuracy}$	**Std Dev**	$Success\%$
DC	23.300	2.525	0.448	0.003	8.154
SNT	10.000	0.000	0.535	0.167	3.667
NichePSO	68.800	3.274	0.463	4.400E-4	6.153
Derating NichePSO	111.767	3.638	0.462	0.003	7.038

lutions.

The Griewank function is highly multimodal (refer to figure 6.2(a)) but to a lesser degree than the Ackley function. The fitness values of the optima increase from the centre of the map outward towards the edges, so that the best optima are found at the edges of the map.

The following observations can be made:

- The DC algorithm has not found many optima, and the average accuracy is only slightly better than the other algorithms. DC has a marginally higher percentage of successfully converged individuals than NichePSO, and derating NichePSO. However, it has not found as many solutions.

- SNT has the worst accuracy, and fails to find optima at the centre of the search space.

- Figure 6.2(e) appears as though the NichePSO has located all optima in the search space, however, keep in mind that these figures illustrate all the solutions formed over 30 simulations. Table 6.6 shows that on average the NichePSO locates only 68.8 optima of the 124 optima that exist.

- The derating NichePSO on the other hand, has located nearly all of the optima on average. The accuracy is similar to the NichePSO, which demonstrates that the modifications to the search space has not significantly effected the accuracy of the solutions.

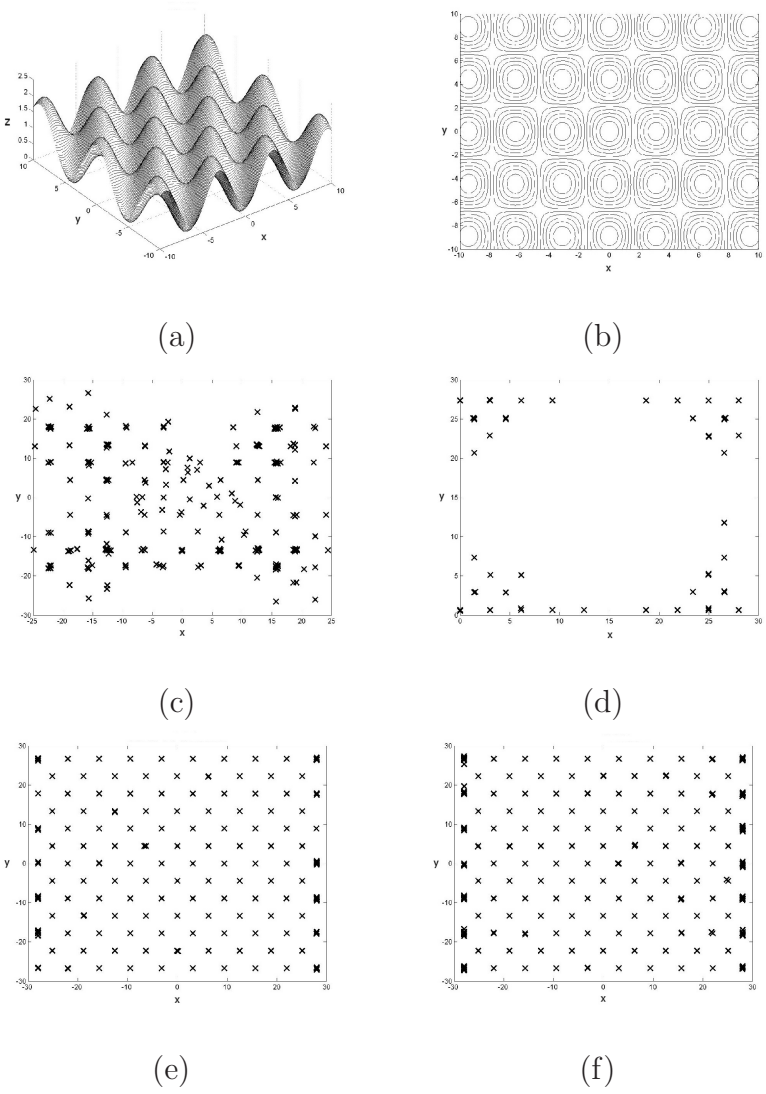


Figure 6.2: (a) Griewank 3D plot (b) Griewank contour map (c) DC results (d) SNT results (e) NichePSO results (f) derating NichePSO results

Table 6.7: Michalewicz results

Algorithm	$\overline{Solutions}$	**Std Dev**	$\overline{M - Accuracy}$	**Std Dev**	$Success\%$
DC	12.570	1.947	0.015	0.005	92.572
SNT	10.000	0.000	0.923	1.412	36.0
NichePSO	16.034	2.785	0.012	0.013	98.960
Derating NichePSO	72.967	4.874	0.005	0.004	96.710

The Michalewicz function has infinitely many local optima and one global optimum (refer to figure 6.3). This tests if the search algorithms can accurately locate the global optimum, and as many of the local optima as possible.

The following observations can be made:

- Many of the individuals in the DC algorithm have converged, but some have stagnated on a plateau in the search space (refer to figure 6.3(c)).
- SNT has the worst accuracy, which maybe due to the modifications of the fitness function and the lack of optimisation in the unmodified search space after a solution is found in the modified fitness function landscape.
- SNT, NichePSO and derating NichePSO managed to always find the global optimum. SNT managed to only locate optima on the vertical ridge, while the PSO niching approaches found local optima on all ridges. In comparison to the other algorithms, the solutions found by the derating NichePSO covers more of the ridges. The derating NichePSO is the most accurate amongst the algorithms, and has again found the most number of solutions.
- Both NichePSO and derating NichePSO accurately find the global optima. On average the NichePSO locates far less local optima than the derating NichePSO.
- For all approaches, except for the SNT, most individuals or particles converged.

The Rastrigin function is another highly multimodal function (refer to figure 6.4(a)) which has the worst local optimum at the centre of the search space ($x_1 =$

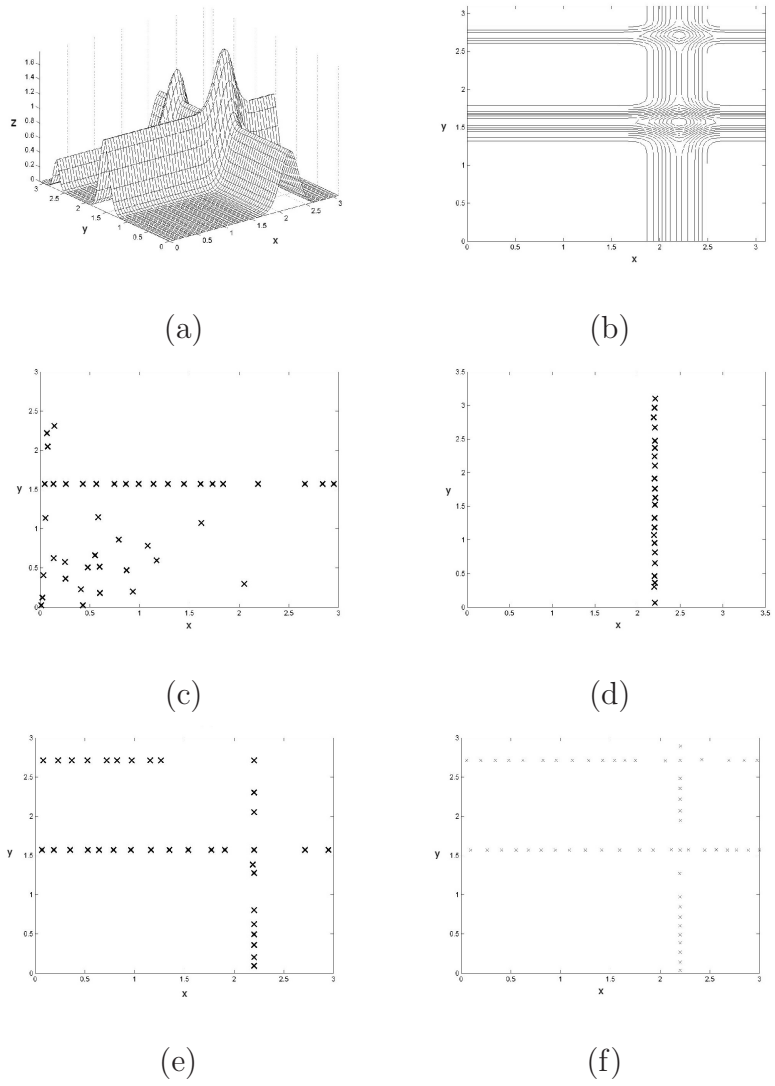


Figure 6.3: (a) Michalewicz 3D plot (b) Michalewicz contour map (c) DC results (d) SNT results (e) NichePSO results (f) derating NichePSO results

Table 6.8: Rastrigin results

Algorithm	$\overline{Solutions}$	**Std Dev**	$\overline{M - Accuracy}$	**Std Dev**	$Success\%$
DC	20.200	2.173	41.120	0.415	9.900
SNT	10.000	0.000	20.345	17.965	9.000
NichePSO	45.100	4.414	0.083	0.235	99.039
Derating NichePSO	132.000	6.140	0.183	0.007	96.135

$0, x_2 = 0$) and the best optima are located near the edges of the search space. The height of the local optima increases rapidly towards the edges of the map. The large concentration of optima in a small search space tests the ability of the search algorithms to explore the search area without a tendency to converge to the edges of the search space.

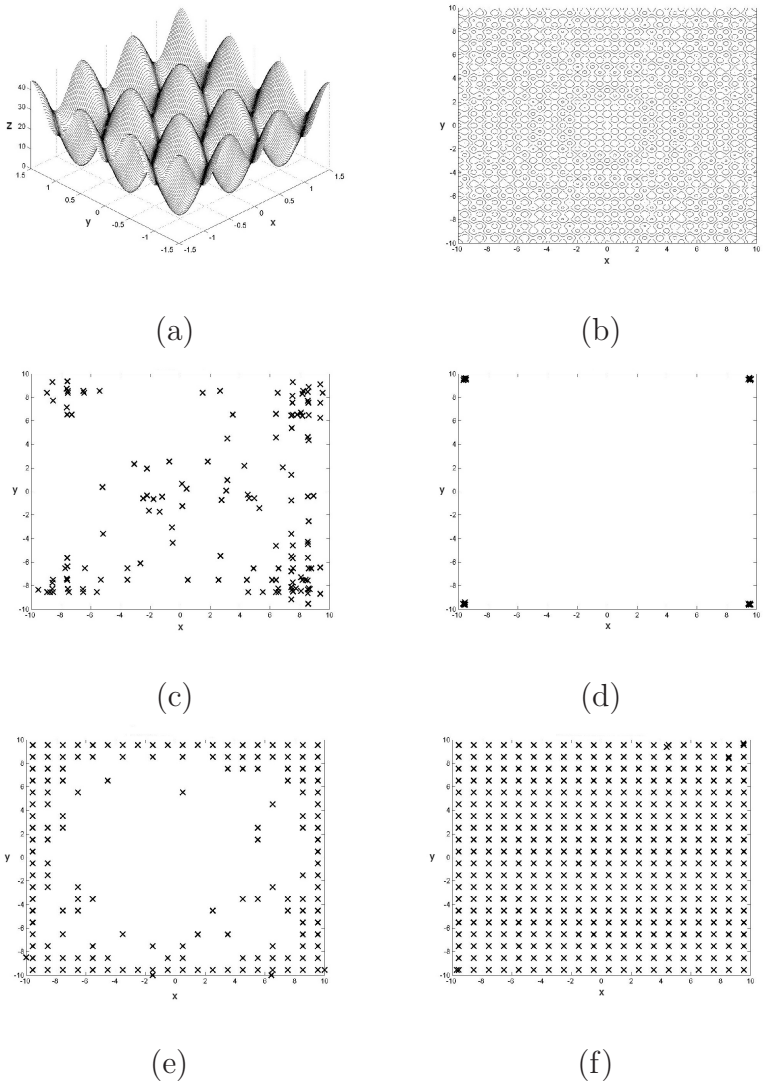
The following observations can be made:

- DC has not converged to all of the best solutions at the edges of the search space (refer to figure 6.4(c)), and the $Success\%$ value indicates that most individuals did not converge in the allowed number of generations. The average number of solutions found is extremely low compared to the NichePSO and derating NichePSO.

- SNT has converged to the global optima at the corners of the search space and has not discovered other local optima.

- NichePSO has done well in terms of accuracy and has found the global optima at the edges of the map. However, the NichePSO has failed to locate any of the niches at the centre of the map, which illustrates that the particles converged to better niches at the edges of the search space even though those niches have already been discovered.

- The derating NichePSO has proved successful in locating the global optima and other local optima in all areas of the search space. The accuracy is slightly worse than the NichePSO but the percentage of successfully converged subswarms is good. In order to improve accuracy, subswarms can be allowed to



(a) (b)

(c) (d)

(e) (f)

Figure 6.4: (a) Rastrigin 3D plot (b) Rastrigin contour map (c) DC results (d) SNT results (e) NichePSO results (f) derating NichePSO results

Table 6.9: Ursem F1 results

Algorithm	$\overline{Solutions}$	**Std Dev**	$\overline{M - Accuracy}$	**Std Dev**	$Success\%$
DC	20.767	2.117	1.976	0.017	9.791
SNT	10.000	0.000	2.139	0.970	0.380
NichePSO	1.970	0.185	2.262E-8	8.407E-4	100.00
Derating NichePSO	2.500	0.587	0.033	0.007	90.667

search in the unmodified search space for a longer period (refer to section 6.3 for a discussion on improving accuracy with derating NichePSO).

The Ursem F1 function is saddled shaped and has one global optimum and one local optimum (as can be seen in figure 6.5).

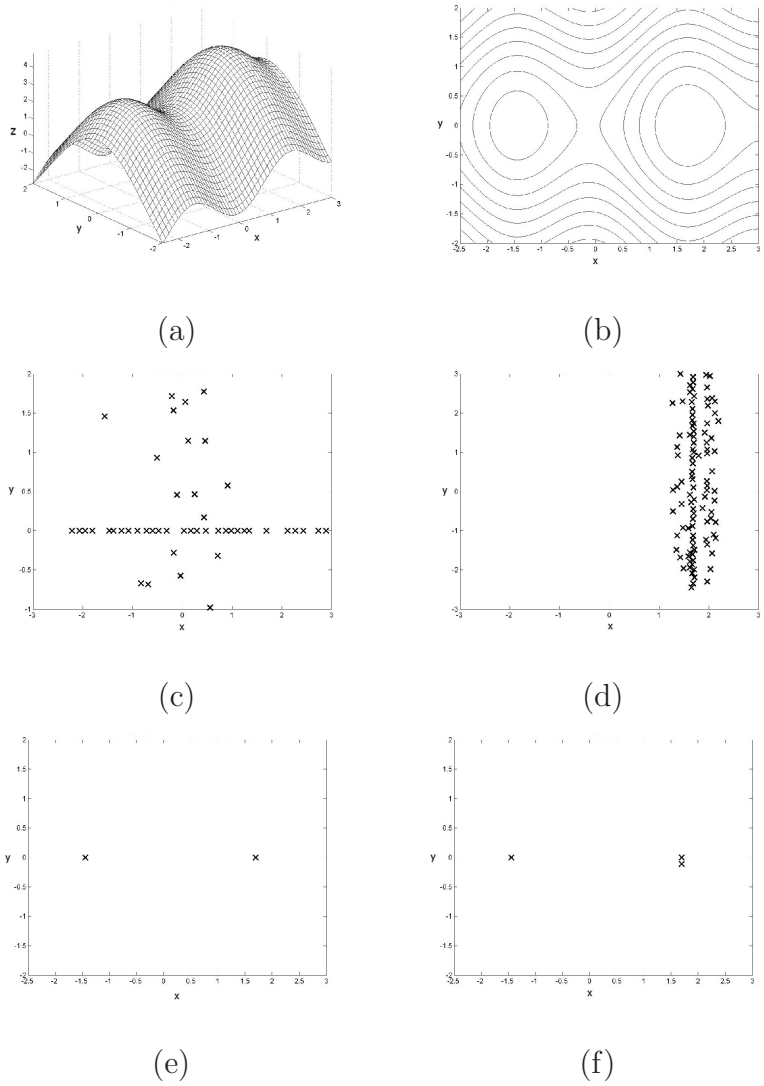
The following can be deduced:

- DC finds 20.767 solutions on average, but the $Success\%$ is 9.791% which implies that many individuals have not converged on any of the two optima. Figure 6.5 (c) further illustrates that DC does not converge on the optima, as many individuals are scattered throughout the search space.

- SNT is unable to converge to either the local or global optima. Modifications to the search space have introduced many false local optima, which have been incorrectly identified as solutions.

- NichePSO has accurately found, converged and maintained two optima for most of the simulations.

- The derating NichePSO finds 2.5 solutions on average instead of 2.0 and the percentage of successfully converged subswarms is less than that of the NichePSO's. As a result of nearly 10% of particles that have not converged, the derating NichePSO has not been able to accurately identify the correct number of optima. Further iterations in the second phase can help improve accuracy (refer to section 6.3).



(a) (b)

(c) (d)

(e) (f)

Figure 6.5: (a) Ursem F1 3D plot (b) Ursem F1 contour map (c) DC results (d) SNT results (e) NichePSO results (f) derating NichePSO results

6.3 Improving Derating NichePSO Results

For the Ackley, Griewank and Michalewicz evaluation functions, the derating NichePSO has produced more solutions than any of the other search algorithms with similar or better accuracy. For the Rastrigin evaluation function, the derating NichePSO has produced the most number of solutions, but with poor accuracy. For the Ursem F1 evaluation function, all subswarms of the derating NichePSO have not converged on optima, and therefore an incorrect number of solutions is reported. To improve the derating NichePSO's accuracy, subswarms can be allowed to search for a longer period in the unmodified search space. Tables 6.10 - 6.12 summarise the results of increasing the maximum number iterations in phase 2 for the Rastrigin, Michalewicz and Ursem F1 evaluation functions respectively.

In tables 6.10 and 6.11, the average number of optima found has decreased slightly as the percentage of successfully converged subswarms increases. The decrease in the average number of solutions is a result of more particles converging on solutions which then merge with other solutions so as to reduce the average number of solutions found. Furthermore, the average accuracy has improved as the number of iterations were increased. The Ursem F1 results in table 6.12 show that all subswarms have converged and the number of optima is correctly reported as 2 after 1000 iterations in the second phase. Thus, increasing the number of iterations in the second phase improves the accuracy of the derating NichePSO.

6.4 Scalability

This section examines the relationship between the size of the main swarm, $|S|$, and the number of executions, R, of the NichePSO. The main aim of this section is to evaluate the performance of the derating NichePSO as $|S|$ and R increase for a fixed number of solutions, q, and under increasing dimensions respectively.

NichePSO is a multi-solution algorithm, meaning that it is able to find more than one solution in a single execution of the algorithm. The derating NichePSO sequentially executes a reinitialised NichePSO for a number of times. For each new execution of the NichePSO, it is hoped that new solutions are discovered. As every

Table 6.10: Improved Rastrigin results

Iterations	$\overline{Solutions}$	**Std Dev**	$\overline{M - Accuracy}$	**Std Dev**	$Success\%$
500	131.167	6.234	0.184	0.021	95.806
1000	131.9	4.106	0.081	0.013	98.534
2000	130.267	4.042	0.063	0.019	99.641
4000	129.934	5.401	0.026	0.009	99.717

Table 6.11: Improved Michalewicz results

Iterations	$\overline{Solutions}$	**Std Dev**	$\overline{M - Accuracy}$	**Std Dev**	$Success\%$
500	70.600	5.901	0.053	0.009	96.506
1000	69.033	5.798	0.025	0.002	97.923
2000	67.167	4.537	0.011	0.001	99.404
4000	64.667	4.781	0.002	0.001	99.742

Table 6.12: Improved Ursem F1 results

Iterations	$\overline{Solutions}$	**Std Dev**	$\overline{M - Accuracy}$	**Std Dev**	$Success\%$
500	2.367	0.669	0.012	0.004	97.183
1000	2.000	0.000	0.000	0.000	100.000
2000	2.033	0.185	0.001	6.440E-4	100.000
4000	2.000	0.000	0.000	0.000	100.000

subswarm needs at least 2 particles, the maximum number of solutions that can be found by the main swarm per execution is $|S|/2$. Assuming that there is an infinite number of solutions and at least a single, unique solution is found in every execution of the NichePSO, the maximum number of solutions that can be found by the derating NichePSO is given by

$$\text{Maximum number of solutions} = R \times (|S|/2) \tag{6.9}$$

where R is the maximum number of executions of the NichePSO. From equation (6.9), three approaches can be used to improve the scalability of the derating NichePSO:

1. Increase the number of particles in the main swarm.
2. Increase the number of executions of the algorithm.
3. Increase both the number of particles and the number of executions.

To cover a larger area of the search space more particles can be used in the main swarm. A larger main swarm has the potential to produce more subswarms, which may increase the number of discovered solutions. An improvement in performance due to an increase in the number of particles depends on how particles are initialised. If particles are not well spread across the search space, then the number of particles has a marginal effect on the number of solutions found, as particles may most likely converge on a similar optimum to form a subswarm in that location. The creation of subswarms reduces the size of the main swarm, and thus the potential number of solutions that can be found decreases if many of the subswarms converge on similar optima.

Increasing the number of executions has the potential of finding a larger number of solutions, as each execution of the NichePSO may find a solution. Increasing the number executions also increases the number of possible modifications to the search space, which increases the number of false optima in the search space. Additional false peaks in the search space may result in poor accuracy if subswarms do not

converge on true optima within the maximum number of iterations in the second phase of the algorithm.

Increasing both the number particles and the number of executions may further improve the number of solutions found.

The above three approaches are examined in the following subsections for a fixed dimension, and under increasing dimension.

6.4.1 Scalability For a Fixed Dimension

This subsection's objective is to determine an equation or heuristic that can be used to verify the relationship between $|S|$, R, and a specific number of optima. The following functions were used for this objective:

- Rastrigin, equation (6.7), where $-30.0 \leq x_i \leq 30.0$, $n = 1$, and the number of optima is 60.
- Michalewicz, equation (6.6), where $0 \leq x_1, x_2 \leq \pi$, and the number of optima is infinite.
- Multimodal Function 1 (MF1)

$$f(\mathbf{x}) = \sum_{i=1}^{n} \sin^6(5\pi x_i), \tag{6.10}$$

  where $0 \leq x_i \leq 20$, $n = 1$, and the number of optima is 100.

These evaluation functions were chosen as the number of optima can be easily estimated. The parameter settings for the derating NichePSO are the same as that provided in table 6.4 except for the main swarm's number of particles and the number of executions. All average results are sampled from 30 simulations.

The results are summarised in table 6.13. Table 6.13 presents the evaluation functions, and references the respective tables (given in appendix A) for the average number of solutions found, standard deviation of the average number of solutions, average accuracy, and the standard deviation of the average accuracy. The average

number of solutions found are plotted on a meshed surface in figures 6.6, 6.7, and 6.8 for the respective evaluation functions.

Inspecting the average number of solutions found for MF1, which are presented in table A.1, and illustrated in figure 6.6, the following can be observed:

- As R and $|S|$ increase, the value of $\overline{Solutions}$ increases and plateaus at approximately the maximum number of solutions in the search space.
- The $\overline{Solutions}$ is approximately the same as either R or $|S|$ increases.
- The average accuracy in table A.3 for the corresponding $\overline{Solutions}$ in table A.1 are 0, or approximately 0, and the percentage of converged subswarms in table A.5 are 100% or approximately 100%.

Similar observations can be made for the average number of solutions found for the Rastrigin function which are presented in table A.6, and illustrated in figure 6.7. However, the Michalewicz function has an infinite number of solutions, and therefore the $\overline{Solutions}$ found by the derating NichePSO does not reach a plateau, as illustrated in figure 6.8. Furthermore, the $\overline{Solutions}$ differ for the respective evaluation functions across the corresponding values of R and $|S|$. That is, there does not seem to be a general relationship between R, $|S|$, and the number of solutions that can be found by the derating NichePSO. Rather, the relationship is dependent on the complexity of the search space and can be approximated by the equation:

$$\overline{Solutions} \approx q \times \left(\frac{e^{\frac{\lambda}{q}(R \times |S|)} - e^{-\frac{\lambda}{q}(R \times |S|)}}{e^{\frac{\lambda}{q}(R \times |S|)} + e^{-\frac{\lambda}{q}(R \times |S|)}} \right) \tag{6.11}$$

where λ represents a measure of how convoluted the search space is, q is the maximum number of solutions in the search space, R and $|S|$ are positive integer values. Equation 6.11 was empirically derived by finding the best curve to fit the results in tables A.1 and A.6. Figure 6.9 illustrates equation (6.11) for $\lambda = 0.35$ and $q = 100$, which is an approximation of the number of solutions found by the derating NichePSO for MF1. Similarly, figure 6.10 illustrates equation (6.11) for $\lambda = 0.1$

Table 6.13: Summary of results for the scalability study in a fixed dimensional problem space

Function	$\overline{Solutions}$	**Std Dev**	$\overline{M - Accuracy}$	**Std Dev**	$Success\%$
MF1	Table A.1	Table A.2	Table A.3	Table A.4	Table A.5
Rastrigin	Table A.6	Table A.7	Table A.8	Table A.9	Table A.10
Michalewicz	Table A.11	Table A.12	Table A.13	Table A.14	Table A.15

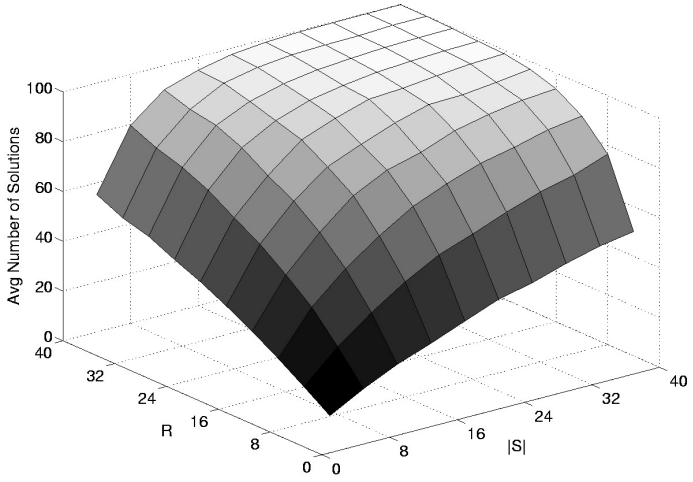


Figure 6.6: Average number of solutions for MF1 with increasing R and $|S|$

and $q = 60$, which is an approximation of the number of solutions found by the derating NichePSO for the Rastrigin function. The value of λ is used to determine the number of executions, R, and the size of the main swarm, $|S|$, required to find the maximum number of solutions, q, in the search space. For complex search domains, that is a search space with many optima, the value of λ approaches 0. If $\lambda = 0$, then no solutions can be found and equation (6.11) is a flat plane on $\overline{Solutions} = 0$ for all values of R and $|S|$. If $q = \infty$, as with the Michalewicz function, then equation (6.11) is undefined. Therefore, it is not possible to approximate $\overline{Solutions}$ when $q = \infty$.

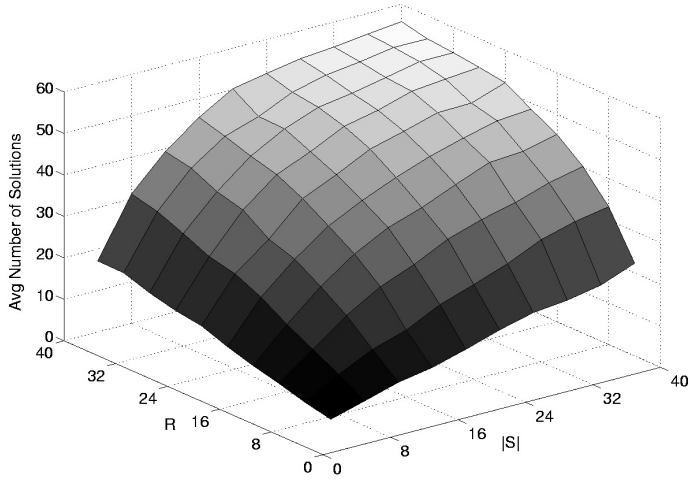


Figure 6.7: Average number of solutions for Rastrigin with increasing R and $|S|$

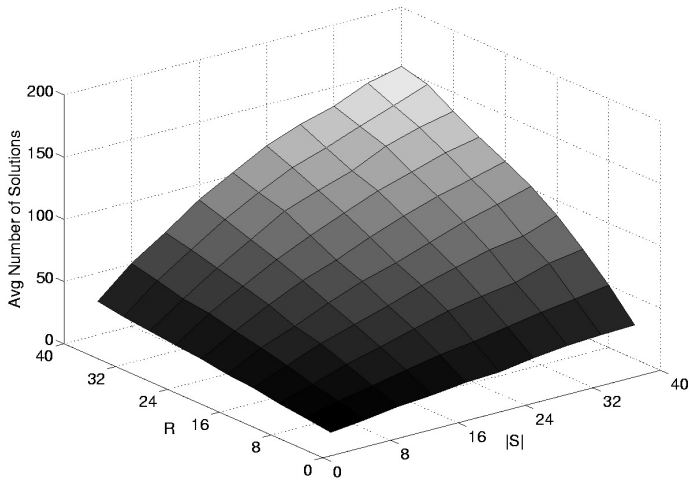


Figure 6.8: Average number of solutions for Michalewicz with increasing R and $|S|$

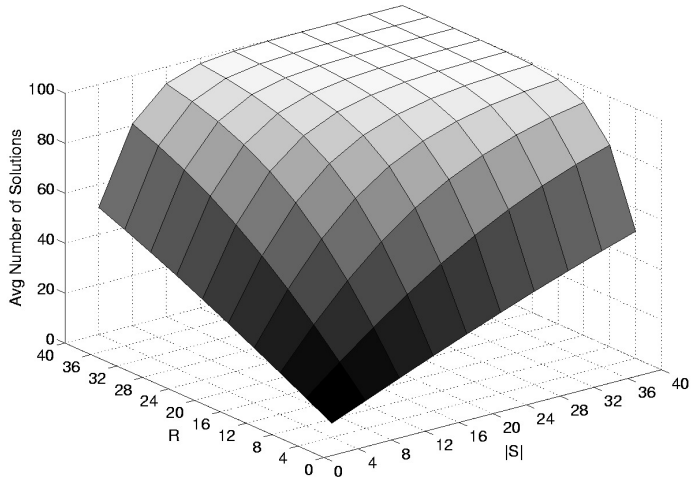


Figure 6.9: Approximation of average number of solutions for MF1 with equation (6.11), where $\lambda = 0.35$ and $q = 100$

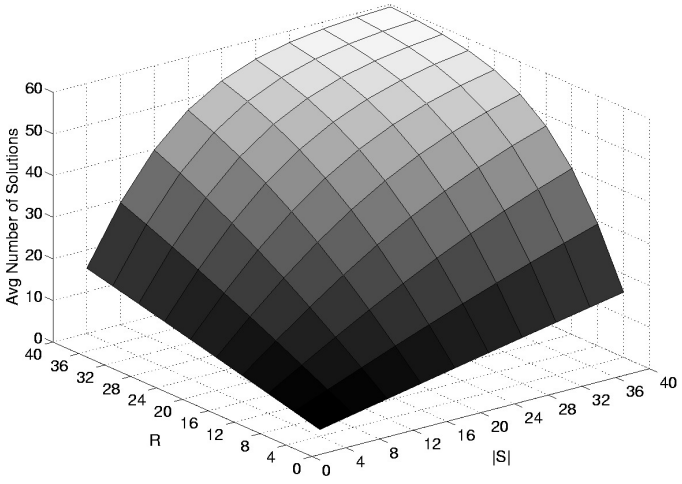


Figure 6.10: Approximation of average number of solutions for Rastrigin with equation (6.11), where $\lambda = 0.1$ and $q = 60$

Table 6.14: Maximum number of solutions for Rastrigin, Griewank, MF1 and MF3, in 2, 3 and 4 dimensional space

n	**Rastrigin**	**Griewank**	**MF1**	**MF3**
1	4	4	4	4
2	16	18	16	16
3	64	82	64	64
4	256	368	256	256

6.4.2 Scalability Under Increasing Dimensions

To analyse the scalability of the derating NichePSO under increasing dimensional problem spaces the following functions were used:

- Rastrigin, equation (6.7), where $-2.0 \leq x_i \leq 2.0$.
- Griewank, equation (6.5), where $-10.0 \leq x_i \leq 10.0$.
- Multimodal Function 1 (MF1), equation (6.10), where $0 \leq x_i \leq 0.8$.
- Multimodal Function 3 (MF3)

$$f(\mathbf{x}) = \sum_{i=1}^{n} \sin^6(5\pi x_i^{3/4}), \tag{6.12}$$

  where $0 \leq x_i \leq 0.8$.

The domain of each function was chosen such that in each dimension, all the functions have the same number of solutions (with the exception of Griewank that has more solutions than the other functions). The number of solutions for each function, for the respective dimension, is summarised in the table 6.14.

The parameter settings for the derating NichePSO are the same as that provided in table 6.4 except for the main swarm's number of particles, number of executions, and a maximum of 1000 iterations was used as the stopping criterion in the second phase in order to provide sufficient time for the refinement of solutions.

The average number of solutions found for the Rastrigin, Griewank, MF1, and MF3 functions are illustrated in figures 6.11 to 6.14. Inspecting figures 6.11 to 6.14, it can be observed that the average number of solutions found, $\overline{Solutions}$, is approximately the same whether $|S|$ or R increases. For example, the value of $\overline{Solutions}$ for $R = 8$ and $|S| = 4$ is nearly equivalent to the value of $\overline{Solutions}$ for $R = 8$ and $|S| = 4$ for all evaluation functions. It can be further observed, that $\overline{Solutions}$ plateaus near the maximum number of solutions that exist in the search space for dimensions two and three. For the fourth dimensional problem space the number of solutions that can be found by the derating NichePSO is approximately 150, whereas the number of solutions that exist in the search space is more than 250 (refer to table 6.14). However, the surface plot in figures 6.11 to 6.14 do not indicate that the derating NichePSO has reached a plateau in the fourth dimension. Increasing the number of particles, or the number of executions will therefore result in discovering further solutions.

Table 6.15: Summary of results for the scalability study in a two dimensional problem space

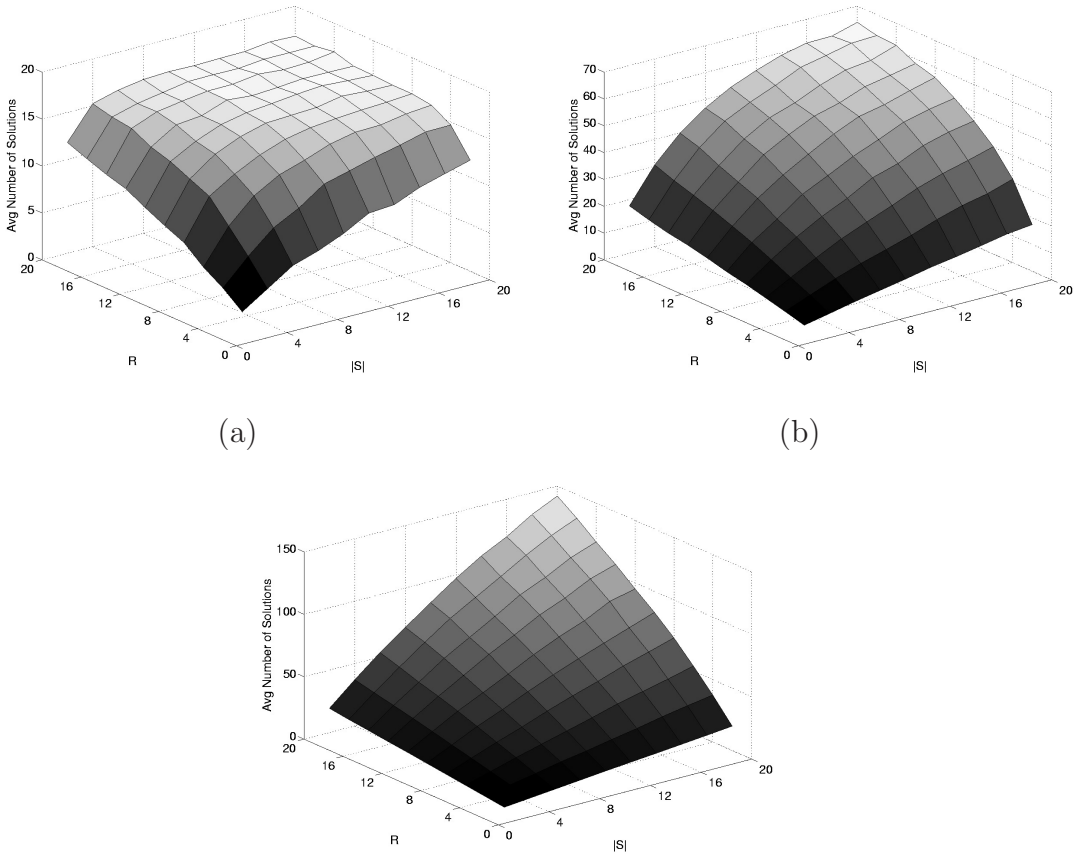
Function	$\overline{Solutions}$	**Std Dev**	$\overline{M - Accuracy}$	**Std Dev**	$Success\%$
Rastrigin	Table B.46	Table B.52	Table B.49	Table B.55	Table B.58
Griewank	Table B.1	Table B.7	Table B.4	Table B.10	Table B.13
MF1	Table B.16	Table B.22	Table B.19	Table B.25	Table B.28
MF3	Table B.31	Table B.37	Table B.34	Table B.40	Table B.43

Table 6.16: Summary of results for the scalability study in a three dimensional problem space

Function	$\overline{Solutions}$	**Std Dev**	$\overline{M - Accuracy}$	**Std Dev**	$Success\%$
Rastrigin	Table B.47	Table B.53	Table B.50	Table B.56	Table B.59
Griewank	Table B.2	Table B.8	Table B.5	Table B.11	Table B.14
MF1	Table B.17	Table B.23	Table B.20	Table B.26	Table B.29
MF3	Table B.32	Table B.38	Table B.35	Table B.41	Table B.44

Table 6.17: Summary of results for the scalability study in a four dimensional problem space

Function	$\overline{Solutions}$	**Std Dev**	$\overline{M - Accuracy}$	**Std Dev**	$Success\%$
Rastrigin	Table B.48	Table B.54	Table B.51	Table B.57	Table B.60
Griewank	Table B.3	Table B.9	Table B.6	Table B.12	Table B.15
MF1	Table B.18	Table B.24	Table B.21	Table B.27	Table B.30
MF3	Table B.33	Table B.39	Table B.36	Table B.42	Table B.45



(a) (b)

(c)

Figure 6.11: Average number of solutions found for Rastrigin in (a) 1D (b) 2D (c) 3D

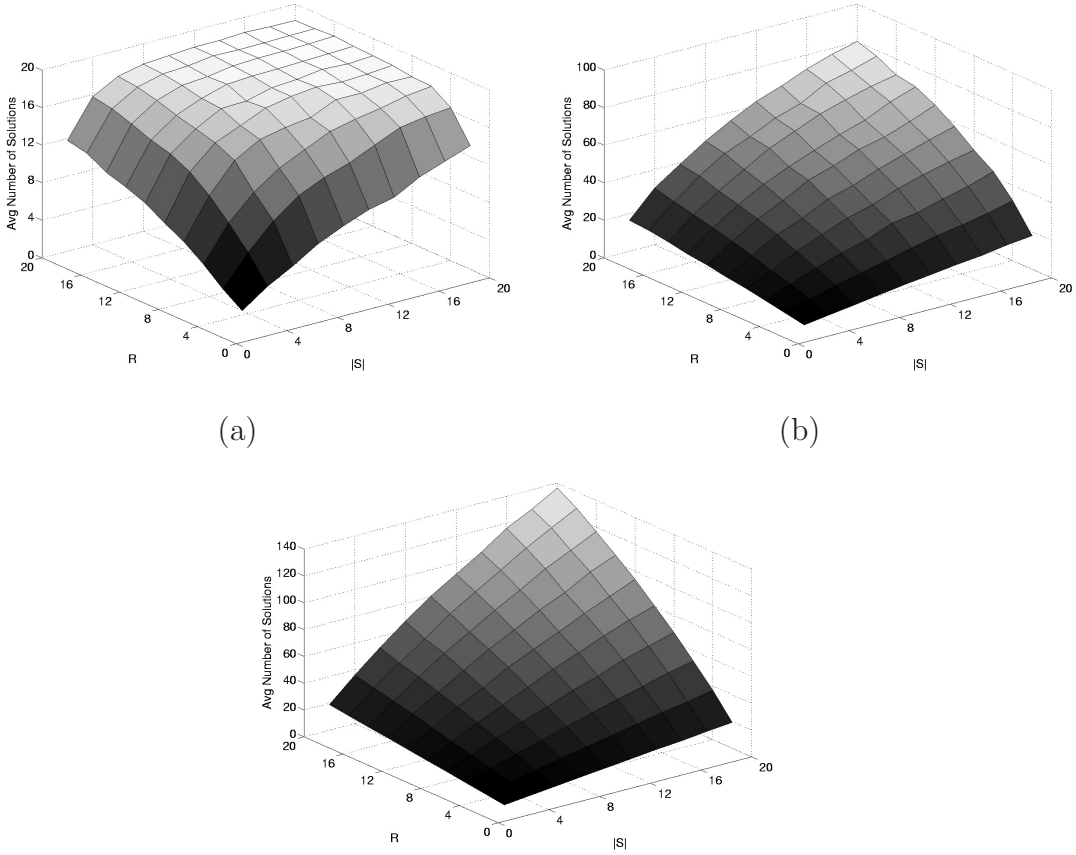


Figure 6.12: Average number of solutions found for the Griewank function in (a) 1D (b) 2D (c) 3D

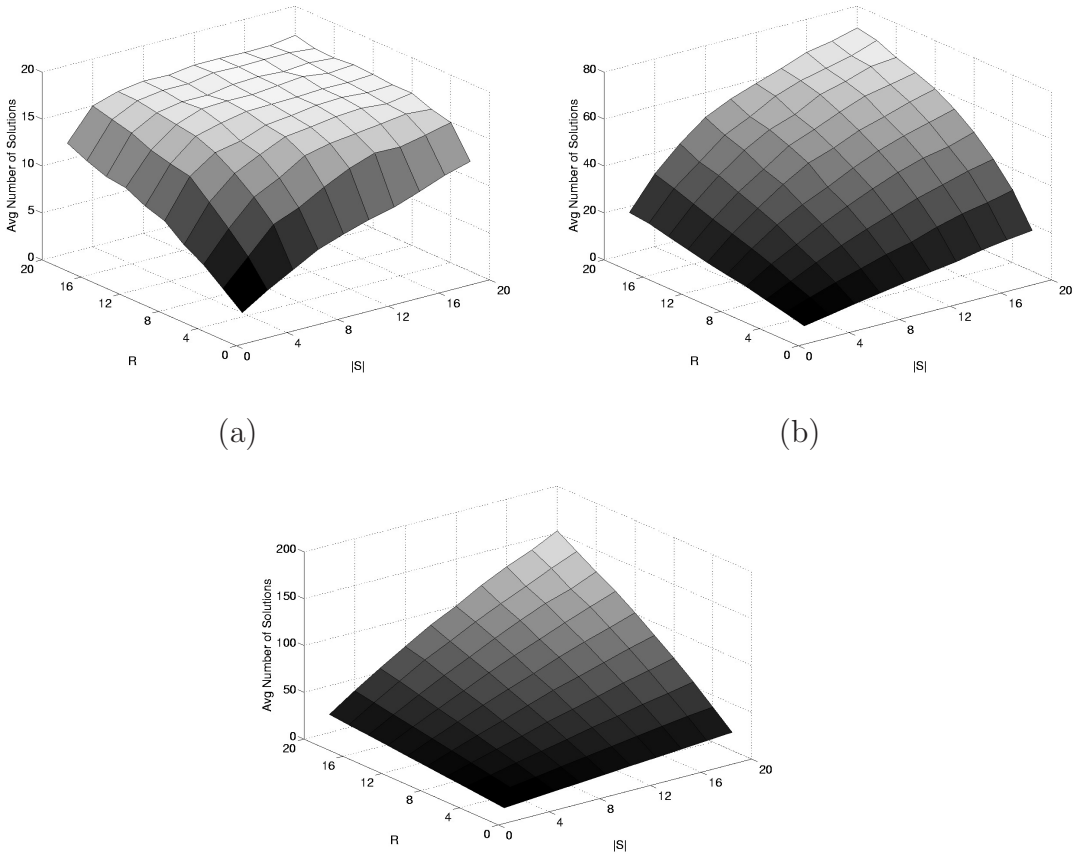
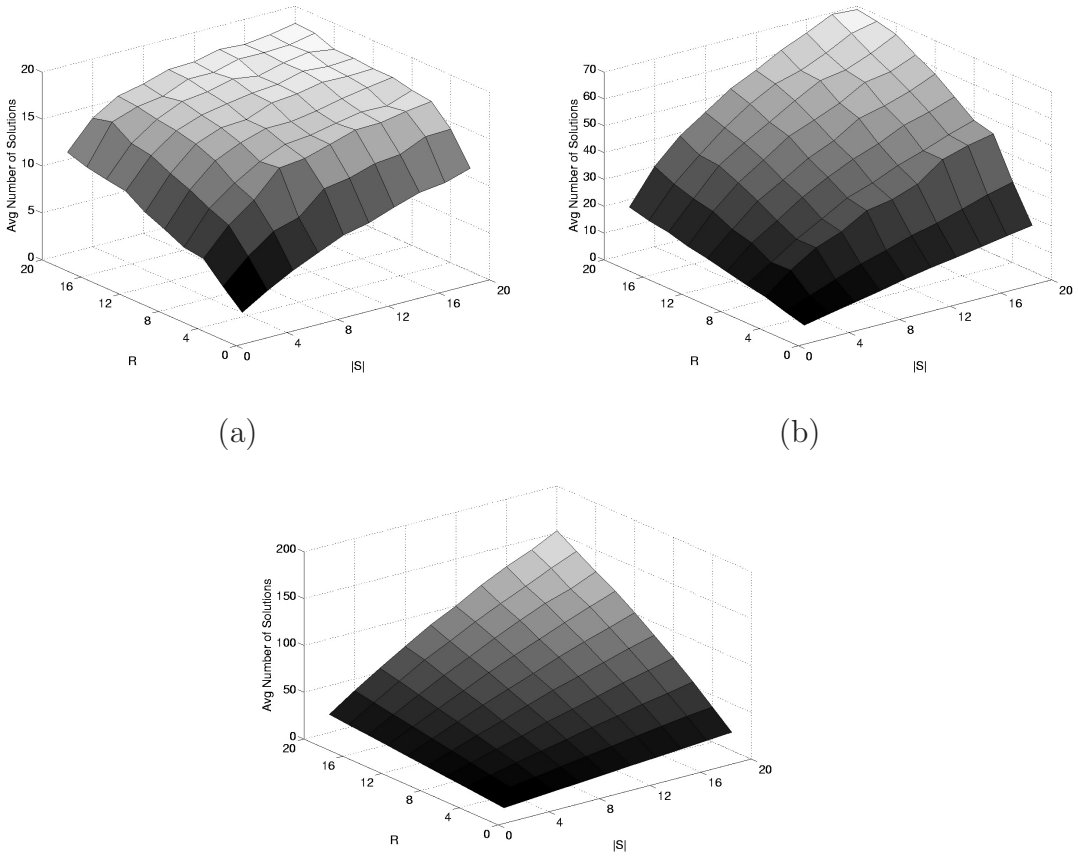


Figure 6.13: Average number of solutions found for MF1 in (a) 1D (b) 2D (c) 3D



(a) (b)

(c)

Figure 6.14: Average number of solutions found for MF3 in (a) 1D (b) 2D (c) 3D

The accuracy of the solutions and the percentage of successfully converged subswarms for MF1, MF3 and Rastrigin, as summarised in tables 6.15, 6.16, and 6.17, indicate that most solutions are consistently and accurately located. However, the percentage of successfully converged subswarms for the Griewank function is low, even though the derating NichePSO discovers nearly all the solutions in the two and three dimensional Griewank function, and a large number of solutions for the fourth dimensional Griewank problem. Increasing the maximum number of iterations in the second phase of the derating NichePSO may improve the accuracy of the solutions (as illustrated in section 6.3).

The results in tables B.1 to B.60, and figures 6.11 to 6.14, indicate that equation (6.11) can be used to approximate the average number of solutions found by the derating NichePSO (refer to section 6.4.1). A similar scalability test was performed by Brits [14] on the NichePSO where the relationship between the number of particles of the main swarm and the number of solutions in the search space was empirically derived as

$$|S| = c \cdot q^a$$

where c is a positive constant, q is the number of solutions in the search space, and $1 \leq a \leq 2$ (refer to equation (4.16)). This means, for the NichePSO to find 100 solutions, that is $q = 100$, the size of the main swarm is

$$c \cdot 100 \leq |S_N| \leq c \cdot 10000$$

where $|S_N|$ is the total number of particles required by the NichePSO. The value c in equation (4.16) and the value of λ in equation (6.11) represent the complexity of the search space, where values of λ that approximate 0 represent more convoluted search spaces. Thus, the number of particles and number of executions required for the derating NichePSO to find 100 solutions is approximately

$$20 \leq R, |S| \leq 70$$

where $0.1 \leq \lambda \leq 1.0$. The total number of particles used by the derating NichePSO is, $R \times |S|$. Therefore, the total number of particles required by the derating NichePSO to find 100 solutions is

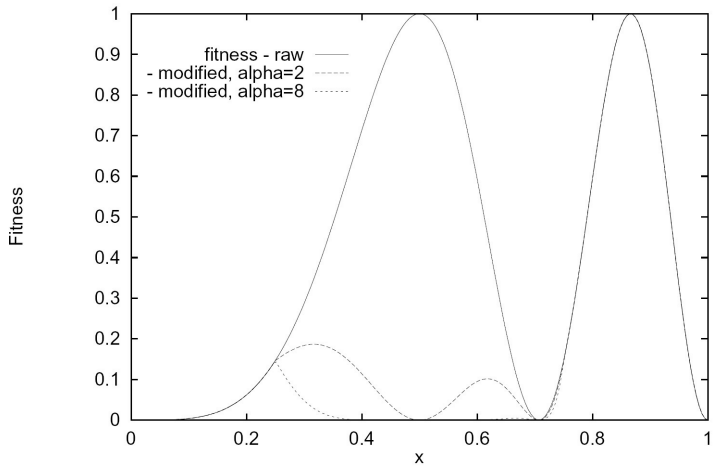
$$400 \leq |S_{DN}| \leq 4900$$

where $R = |S|$, and $|S_{DN}|$ represents the total number of particles for the derating NichePSO. Although the minimum of $|S_{DN}|$ is larger than the minimum $|S_N|$, for $0 < c \leq 1$, the maximum of $|S_{DN}|$ is lower than the maximum $|S_N|$, which shows that the derating NichePSO improves the scalability of the NichePSO.

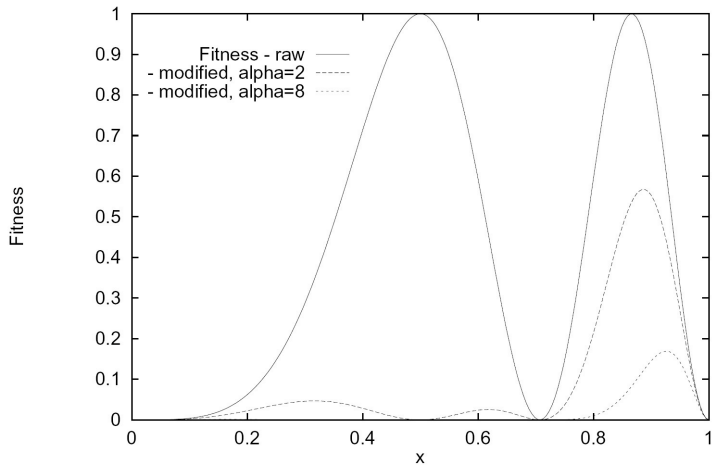
6.5 Niche Radius Sensitivity

Beasley *et al* [8] investigated problems experienced when using an inappropriate niche radius. With the assumption that maxima are not evenly distributed in the search space, Beasley *et al* found that using a small niche radius easily introduces new peaks. Figure 6.15 (a) illustrates the fitness function landscape that is modified using equation (3.4) for a small niche radius and various values of α. Figure 6.15 (a) further illustrates that two false peaks are created due to introducing the modifications. Figure 6.15 (b) illustrates the fitness function landscape after applying the derating function (3.4) for various values of α. A large niche radius still introduces new peaks and has the additional side effect of shifting the position of neighbouring peaks as well as reducing their height. Beasley *et al* suggested to increase the number of executions, which will suppress false peaks that have been introduced and to perform a local search using the raw fitness function. This section examines how sensitive the derating NichePSO is to different values of the niche radius. The test functions used in this study are equations 6.7 (where $-2.0 \leq x_i \leq 2.0$), 6.5 (where $-10.0 \leq x_i \leq 10.0$) and 6.12.

The maxima of MF3 (equation 6.12) are not evenly distributed in the search space, whereas the maxima for the Rastrigin and Griewank functions are evenly distributed (refer to figure 6.4(b)). Table 6.19 presents the results of executing the derating NichePSO with the parameters given in table 6.18. The number of particles was arbitrarily chosen as 16. Figure 6.16(a) plots the values in table 6.19, which illustrates that the derating NichePSO is not sensitive to the niche radius (with respect to average accuracy). This suggests that the subswarms are not negatively influenced by the modifications in the search space, if subswarms improve their solutions by searching in the unmodified fitness landscape. Unfortunately, the accuracy



(a)



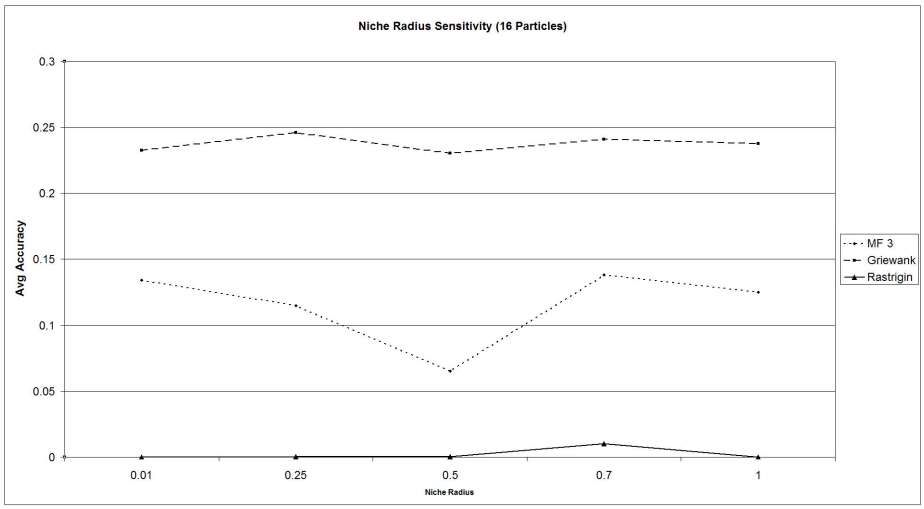
(b)

Figure 6.15: Side effects of a (a) Small niche radius (b) Large niche radius, where *alpha* represents α

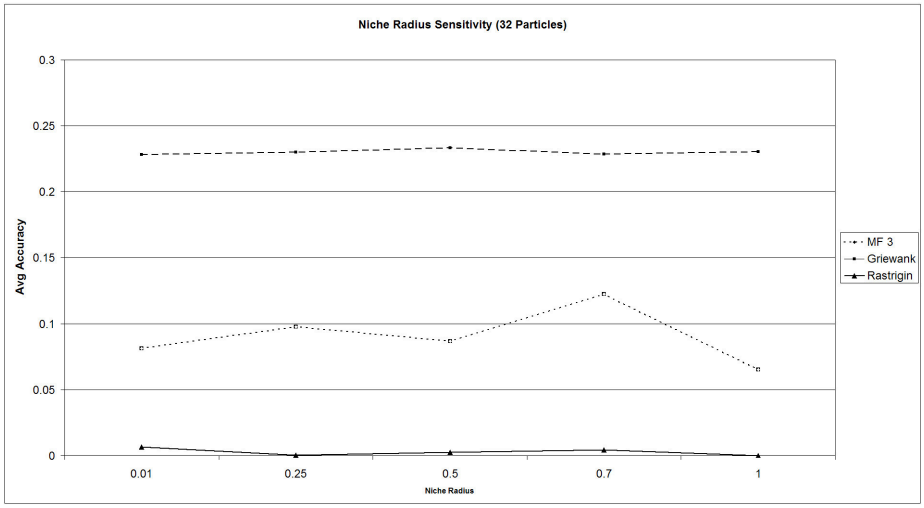
Table 6.18: Derating NichePSO parameters

Parameter	**Value**
Number of Particles	16
Merge Threshold	0.01
Niche Threshold	0.00001
c_1 and c_2	1.2
ϕ	0.7
Solution Threshold for Phase 2	0.000001
Stopping Criterion (sequential executions)	Maximum executions = 10
Stopping Criteria (for Phase 1)	Maximum Iterations = 500 or Main Swarm has no more particles
Stopping Criteria (for Phase 2)	Maximum Iterations = 5000 or subswarm has converged.
Alpha (α)	0.8

deviates for MF3. To improve accuracy, the number of particles was increased to 32. The results of this test are presented in table 6.20 and plotted in figure 6.16(b). Figure 6.16(b) illustrates that additional particles have helped to stabilise the accuracy for function MF3. Furthermore, a niche radius of 1.0 and 0.7 is clearly too large for MF3 as the domain of the function is defined in $0 \leq x_1, x_2 \leq 0.8$. Ignoring those results for a niche radius of 1.0 and 0.7, the results for MF3 now appear more stable. This demonstrates that the derating NichePSO is not as sensitive to a varying niche radius as the SNT, since the average accuracy does not deviate considerably (assuming that the niche radius is at least less than the size of the domain).



(a)



(b)

Figure 6.16: Niche radius sensitivity analysis (a) 16 Particles (b) 32 Particles

Table 6.19: Niche radius sensitivity analysis (16 Particles), with respect to average accuracy

Niche Radius	**Rastrigin**	**Griewank**	**MF 3**
0.01	0.006 ± 8.794E-5	0.228 ± 0.037	0.081 ± 0.037
0.25	0.000 ± 2.001E-4	0.230 ± 0.017	0.097 ± 0.017
0.50	0.002 ± 4.938E-4	0.233 ± 0.011	0.087 ± 0.011
0.70	0.004 ± 0.005	0.228 ± 0.016	0.122 ± 0.016
1.00	0.000 ± 7.480E-5	0.230 ± 0.016	0.065 ± 0.016

Table 6.20: Niche radius sensitivity analysis (32 Particles), with respect to average accuracy

Niche Radius	**Rastrigin**	**Griewank**	**MF 3**
0.01	0.000 ± 0.004	0.232 ± 7.133E-4	0.134 ± 0.023
0.25	0.000 ± 1.625E-4	0.246 ± 0.001	0.114 ± 0.017
0.50	0.000 ± 0.001	0.230 ± 0.001	0.065 ± 0.027
0.70	0.010 ± 0.003	0.241 ± 0.001	0.138 ± 0.009
1.00	0.000 ± 1.058E-4	0.238 ± 0.001	0.125 ± 0.014

6.6 Conclusion

Empirical results and analysis of four multimodal search algorithms, namely DC, SNT, NichePSO and derating NichePSO, were presented in this chapter. It was shown that the derating NichePSO is able to accurately locate many solutions in various search spaces. The maximum number of solutions that the derating NichePSO can find was shown to be $R \times (|S|/2)$, where R is the number of executions and $|S|$ is the number of particles in the main swarm. The derating NichePSO was observed to perform well for a fixed dimension and increasing problem dimensions. The number of solutions that can be found for a given search space was empirically estimated as given in equation (6.11). Equation (6.11) was compared to the NichePSO estimate for the number of particles required to find a certain number of

solutions (refer to equation (4.16)), and it was concluded that the derating NichePSO improves the scalability of the NichePSO. Further tests empirically analysed the derating NichePSO as the niche radius varied, and it was concluded that the derating NichePSO is not sensitive to changes in the niche radius.

Chapter 7

Conclusion

The contributions of this study are summarised in this chapter followed by thoughts on further research.

7.1 Summary

The objective and sub-objectives of this thesis included:

- Developing and evaluating a new PSO niching algorithm that is more scalable than the NichePSO.
- A literature overview of EC and PSO niching algorithms in order to identify problems with current techniques.

Chapter 2 presented and discussed what a niche is in terms of a biological system and mathematical model. In terms of a mathematical model, a niche is a solution to a system of equations or an optimum of a function. Functions with a single optimum are known as unimodal, and functions with multiple optima are known as multimodal. The chapter discussed methods that are used to find the optimum of unimodal functions and further elaborated on methods used to to find optima of multimodal functions. The methods are categorised as numerical methods and computational intelligence methods. Derivative based numerical search methods discussed included minimisation using derivatives and gradient descent. These

methods are restricted to the precondition that the function being optimised must be unimodal and must have a derivative to guarantee that the optimum can be found. If the function is multimodal, then there is no guarantee that the global optimum will be found. Non-derivative based search methods discussed included the golden ratio search and the Nelder-Mead method. The golden ratio search experiences inaccuracy if the function is flat near the optimum, and is not suitable for high dimensional problems. The Nelder-Mead method is also inaccurate with four and higher dimensional functions, and is not guaranteed to find the global optimum in a multimodal search space (as with the golden ratio search).

Two paradigms of computational intelligence methods were discussed in chapter 2: Evolutionary Computation (EC), and Swarm Intelligence (SI). In particular, the genetic algorithm (GA) was discussed as an example of EC. The genetic algorithm consists of a population of individuals which represent positions in the search space. Operators are applied to the population to generate new individuals that represent a better position in the search space. The operators discussed included crossover, mutation, and selection. GAs can execute in parallel, which makes them highly scalable. However, GAs have a trade-off between the size of the population, and the quality of the results. That is, a larger population produces better results than a smaller population. Particle swarm optimisation (PSO) and guaranteed convergence particle swarm optimisation (GCPSO) were discussed as examples of SI. PSO maintains a swarm of particles representing positions in the search space. Each particle is associated with a velocity as the step size of that particle. Two models of PSO were discussed, namely the cognitive-only model, and the social-only model. With the cognitive-only model particles move in the direction of their own best position. With the social-only model, particles move in the direction of their neighbourhood's best position. The GCPSO model forces the global best particle to change, which ensures that the swarm does not stagnate [94][95]. The GCPSO has shown to be efficient and robust in locating single solutions in multi-modal search spaces.

Chapter 2 further discussed non-Pareto and Pareto approaches to multi-objective optimisation (MOO). Both niching and MOO have the objective to find multiple

solutions, however niching traditionally applies to single objective functions.

Chapter 3 presented an overview of GA niching approaches. Several methods were discussed, including iteration, parallel sub-populations, fitness sharing, dynamic niche sharing, crowding, deterministic crowding (DC), and the sequential niche technique (SNT). In general, the chapter focused on SNT. SNT sequentially executes the GA (or any other optimisation algorithm) and modifies the search space at the location of a solution found in every execution. This improves the diversity of the GA population and prevents genetic drift of the population to converge on a single solution, but at the cost of inaccuracy due to the modifications in the search space. Beasley *et al* [8] suggests that performing a local search can improve the accuracy of solutions. The SNT finds a single solution on each execution of the GA, thus the GA needs to be executed at least q times to find q unique optima, which increases the computational complexity.

Chapter 4 discussed niching using PSO. The niching ability of *gbest* and *lbest* topologies were discussed and it was shown that the topologies are not suitable for niching. Thus, the PSO has to be modified to support multiple solutions. This led to the discussion of several PSO niching algorithms, including objective function stretching, vector-based PSO, parallel vector-based PSO, *nbest* PSO, species-based PSO and NichePSO. The main objective of chapter 4 was to examine the NichePSO. It was shown that NichePSO, although very accurate, does not scale well for highly multimodal functions. This was attributed to the merging of subswarms to form larger subswarms, which reduces the potential number of solutions that could have been discovered. To control this problem, the merge threshold must not be greater than the lowest inter-niche distance [14]. Even with this prior knowledge, particles from the main swarm and subswarms retrace the search space unnecessarily. That is, a particle may locate an already optimised niche.

Chapter 5 introduced the derating NichePSO to improve the exploration abilities of the NichePSO. The method is a combination of SNT and NichePSO. To accomplish the combination of SNT and NichePSO, the derating NichePSO uses two fitness functions: the modified fitness function and the raw (or unmodified) fitness function. The modified fitness function is the result of removing solutions from

the raw fitness function. Derating NichePSO operates in two phases: In phase 1, the main swarm searches the modified fitness function to find any potential solutions. In phase 2 the subswarms created in phase 1 refine their solutions in the unmodified search space, which improves the accuracy of the solutions. The modified fitness function is then updated by modifying the search space at the locations of the solutions found in phase 2. The search is repeated with a reinitialised NichePSO to discover other solutions. Removing solutions from the modified search space forces particles in the main swarm to explore areas of the search space that has not yet been traversed. When there are no more particles in the main swarm (or when the maximum number of iterations has been reached), phase 1 terminates, and phase 2 commences. Chapter 5 further discussed how to reduce the number of duplicate solutions reported by the derating NichePSO, as it is possible that similar solutions can be found on each execution of the NichePSO. To reduce the number of duplicate solutions, subswarms formed in previous executions are merged with the current subswarms.

Chapter 6 presented an evaluation of the derating NichePSO, DC, SNT, and NichePSO. Several commonly used multimodal functions were used as a benchmark. The results showed that the derating NichePSO found more solutions than any of the other algorithms with similar or better accuracy than the NichePSO. Furthermore, it was noted that DC and SNT performed badly for most evaluation functions. It was further shown that derating NichePSO accuracy can be improved by increasing the maximum number of iterations in phase 2.

The scalability of the derating NichePSO was examined for a fixed and increasing problem dimensions. It was found that there are three ways in which to improve the scalability of the derating NichePSO:

1. Increase the number of particles in the main swarm.
2. Increase the number of executions of the algorithm.
3. Increase both the number of particles and number of executions.

The results showed that the number of solutions which the derating NichePSO can find can be estimated by equation (6.11), which is repeated here for convenience

$$\overline{Solutions} \approx q \times \left(\frac{e^{\frac{\lambda}{q}(R \times |S|)} - e^{-\frac{\lambda}{q}(R \times |S|)}}{e^{\frac{\lambda}{q}(R \times |S|)} + e^{-\frac{\lambda}{q}(R \times |S|)}} \right)$$

where λ represents a measure of how convoluted the search space is, q is the maximum number of solutions in the search space, R is the number of executions of the NichePSO and $|S|$ is the size of the main swarm. Equation (6.11) was compared to the NichePSO's estimate of the number of particles required for q number of solutions, that is

$$|S| = c \cdot q^a$$

It was concluded that the derating NichePSO improves the scalability of the NichePSO.

The sensitivity of the derating NichePSO was analysed as the niche radius varied, and it was concluded that the derating NichePSO is not sensitive to changes in the niche radius.

7.2 Further Research

This section presents a list of potential further research avenues and improvements of the derating NichePSO:

- There maybe benefits if the niche radius is determined dynamically as most optima are not evenly distributed. The method of determining the niche radius in vector-based PSO [83][82] could be used to determine the niche radius for a specific swarm instead of having a static niche radius. It may be beneficial to increase the niche radius as more subswarms converge to the same optima, which is similar to fitness sharing [37] or dynamic niche sharing [66].

- Improving the diversity of the main swarm can improve the exploration abilities of the NichePSO. Methods such as hybrid PSO with breeding and subpopulations [58] and PSO with self organised criticality [7] may benefit the main swarm of the NichePSO. These methods, applied to the main swarm, may improve diversity and facilitate exploration.

- The diversity of the main swarm may also be improved by considering the diversity of the main swarm as another objective of the NichePSO, which means that the NichePSO is optimising two functions: one being the problem search space, and the other being a function that measures the diversity of the main swarm.
- The scalability study of the derating NichePSO in this thesis examined the performance of the algorithm for evaluation functions up to a maximum of four dimensions. Further analysis in higher dimensions and with a larger set of problems may find additional pitfalls and advantages of the derating NichePSO.
- The application of the derating NichePSO to real world problems.
- An elaborate and comprehensive empirical comparison of all PSO niching methods.

The PSO and SNT were respectively inspired by flocks of birds, and niches formed by species in the natural environment. Merging the two would produce the metaphor: various species of birds that flock in their own niche.

Appendix A

Derating NichePSO, Fixed Dimension

This appendix presents the results of the derating NichePSO for 1D MF1, 1D Rastrigin, and 2D Michalewicz. All values are rounded-up to three decimal places.

Table A.1: Average number of solutions for derating NichePSO in MF1 1D problem domain

R	$\lvert S\rvert$									
	4	8	12	16	20	24	28	32	36	40
4	7.600	14.967	21.233	26.167	31.533	36.800	39.933	44.167	47.667	49.967
8	14.667	27.233	37.433	46.300	53.633	58.467	63.733	68.333	71.633	76.533
12	21.467	38.567	50.500	60.167	68.500	74.100	78.800	81.700	85.267	87.233
16	27.767	46.967	60.500	71.767	78.533	82.100	87.300	90.300	92.533	93.600
20	33.367	54.700	68.500	78.767	84.300	89.367	93.233	93.633	95.133	96.867
24	38.533	60.767	74.167	85.033	90.167	94.333	95.200	96.700	97.933	98.500
28	42.833	67.200	80.800	88.167	92.800	95.267	97.067	98.000	98.633	98.933
32	47.633	71.800	85.200	91.300	94.867	96.967	98.067	98.933	99.300	99.533
36	50.467	75.033	87.033	94.067	96.867	98.367	98.867	99.067	99.767	99.833
40	55.200	79.533	90.333	94.667	97.500	98.867	99.500	99.700	99.833	99.900

Table A.2: Standard deviation of average number of solutions for derating NichePSO in MF1 1D problem domain

R	$\lvert S\rvert$									
	4	8	12	16	20	24	28	32	36	40
4	0.643	0.766	1.232	1.576	2.282	2.117	2.685	2.988	3.705	2.606
8	0.851	1.930	1.619	3.011	2.918	3.629	2.672	3.113	3.327	3.562
12	1.083	2.626	2.971	3.162	3.184	3.459	2.883	2.748	2.442	2.457
16	1.848	2.580	3.000	3.291	2.901	3.222	2.606	2.221	2.304	1.771
20	1.875	2.484	2.983	3.582	4.461	2.471	1.287	2.259	1.838	1.390
24	2.400	2.889	3.358	3.168	2.942	1.771	1.671	1.426	1.130	1.174
28	2.971	3.280	3.211	3.644	2.304	1.732	2.213	1.232	0.947	0.743
32	3.439	3.023	2.988	2.895	3.168	1.921	1.414	0.871	0.947	0.491
36	2.580	3.409	3.667	2.084	2.378	1.313	1.160	0.891	0.455	0.415
40	2.988	3.991	2.828	2.274	1.619	0.947	1.083	0.743	0.415	0.322

Table A.3: Average accuracy for derating NichePSO in MF1 1D problem domain

R	$\|S\|$									
	4	8	12	16	20	24	28	32	36	40
4	0.002	0.000	0.000	0.001	0.000	0.000	0.000	0.000	0.003	0.001
8	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000
12	0.000	0.001	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000
16	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000
20	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000
24	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000
28	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000
32	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000
36	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000
40	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000

Table A.4: Standard deviation of average accuracy for derating NichePSO in MF1 1D problem domain

R	$\|S\|$									
	4	8	12	16	20	24	28	32	36	40
4	0.002	0.000	0.000	0.001	0.000	0.000	0.000	0.000	0.002	0.001
8	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000
12	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000
16	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000
20	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000
24	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000
28	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000
32	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000
36	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000
40	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000

Table A.5: Percentage of successfully converged subswarms for derating NichePSO in MF1 1D problem domain

R	$\lvert S\rvert$									
	4	8	12	16	20	24	28	32	36	40
4	100	100	100	100	100	100	100	100	100	100
8	100	100	99.911	100	100	100	100	100	100	100
12	99.845	100	100	100	100	100	100	100	100	100
16	99.880	99.929	100	100	100	100	100	100	100	100
20	100	100	100	100	100	100	100	100	100	100
24	99.913	100	100	100	100	100	100	100	100	100
28	99.922	100	100	100	100	100	100	100	100	100
32	99.930	99.954	100	100	100	100	100	100	100	100
36	99.868	99.956	100	100	100	100	100	100	100	100
40	100	100	100	100	100	100	100	100	100	99.967

Table A.6: Average number of solutions for derating NichePSO in Rastrigin 1D problem domain

R	$\|S\|$									
	4	8	12	16	20	24	28	32	36	40
4	3.700	6.000	8.400	9.833	12.133	14.700	16.767	17.633	19.167	22.333
8	5.033	9.700	13.600	16.133	20.167	22.900	25.833	29.500	31.767	33.233
12	6.833	12.967	19.033	23.833	27.100	31.100	33.700	35.700	38.367	40.167
16	8.367	16.233	23.233	29.000	33.233	35.967	39.400	42.633	43.200	45.167
20	10.233	20.033	27.100	33.000	37.033	40.467	43.433	45.767	47.800	48.500
24	12.500	22.933	29.167	36.867	40.500	44.567	46.733	49.133	49.333	52.133
28	13.433	24.333	33.467	39.900	43.300	47.300	49.067	51.533	52.367	53.100
32	14.933	27.133	35.133	41.600	46.067	49.133	50.433	52.100	53.833	53.867
36	17.033	29.700	38.067	43.633	46.267	50.033	52.067	53.133	54.500	54.467
40	17.133	31.200	39.300	45.433	50.233	51.800	53.233	53.967	54.967	55.867

Table A.7: Standard deviation of average number of solutions for derating NichePSO in Rastrigin 1D problem domain

R	$\|S\|$									
	4	8	12	16	20	24	28	32	36	40
4	1.130	1.287	2.117	2.327	1.948	2.267	2.449	2.221	2.942	3.091
8	1.671	2.026	2.533	2.901	2.659	2.526	3.000	2.560	3.514	3.723
12	1.543	2.059	3.091	3.068	2.553	3.479	3.168	3.179	3.011	2.954
16	2.059	2.652	2.626	2.792	2.393	3.327	2.804	3.023	3.243	2.371
20	2.181	3.057	3.509	2.983	2.400	3.085	2.877	2.319	2.553	2.304
24	2.084	3.322	3.653	2.877	2.259	2.505	3.129	3.227	3.538	2.349
28	2.533	3.439	2.716	2.965	2.672	2.798	2.421	2.491	1.885	1.565
32	2.678	2.573	3.195	2.393	2.652	2.364	2.349	2.484	2.026	1.781
36	2.428	3.157	3.096	2.267	2.841	2.251	2.729	2.274	1.800	1.742
40	3.533	2.798	3.079	3.029	2.000	1.732	1.661	1.885	1.650	1.486

Table A.8: Average accuracy for derating NichePSO in Rastrigin 1D problem domain

R	$\|S\|$									
	4	8	12	16	20	24	28	32	36	40
4	0.001	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000
8	0.000	0.000	0.000	0.000	0.050	0.000	0.144	0.000	0.000	0.000
12	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000
16	0.000	0.000	0.000	0.003	0.000	0.000	0.000	0.000	0.000	0.000
20	0.000	0.000	0.000	0.000	0.000	0.034	0.000	0.000	0.000	0.000
24	0.000	0.000	0.134	0.000	0.025	0.008	0.037	0.000	0.000	0.000
28	0.000	0.000	0.000	0.000	0.029	0.000	0.000	0.000	0.000	0.000
32	0.001	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.016	0.000
36	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000
40	0.000	0.000	0.000	0.000	0.000	0.002	0.002	0.000	0.000	0.000

Table A.9: Standard deviation of average accuracy for derating NichePSO in Rastrigin 1D problem domain

R	$\|S\|$									
	4	8	12	16	20	24	28	32	36	40
4	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000
8	0.000	0.000	0.000	0.000	0.008	0.000	0.075	0.000	0.000	0.000
12	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000
16	0.000	0.000	0.000	0.002	0.000	0.000	0.000	0.000	0.000	0.000
20	0.000	0.000	0.000	0.000	0.000	0.017	0.000	0.000	0.000	0.000
24	0.000	0.000	0.142	0.000	0.020	0.004	0.023	0.000	0.000	0.000
28	0.000	0.000	0.000	0.000	0.008	0.000	0.000	0.000	0.000	0.000
32	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.006	0.000
36	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000
40	0.000	0.000	0.000	0.000	0.000	0.001	0.001	0.000	0.000	0.000

Table A.10: Percentage of successfully converged subswarms for derating NichePSO in Rastrigin 1D problem domain

R	$\lvert S\rvert$									
	4	8	12	16	20	24	28	32	36	40
4	100.000	100.000	100.000	100.000	100.000	100.000	100.000	100.000	100.000	100.000
8	100.000	100.000	100.000	100.000	99.835	100.000	99.871	100.000	100.000	100.000
12	100.000	100.000	100.000	100.000	100.000	100.000	100.000	100.000	100.000	100.000
16	100.000	100.000	100.000	99.885	100.000	100.000	100.000	100.000	100.000	100.000
20	100.000	100.000	100.000	100.000	100.000	99.835	100.000	100.000	100.000	100.000
24	100.000	100.000	99.886	100.000	99.918	99.925	99.929	100.000	100.000	100.000
28	100.000	100.000	100.000	100.000	99.923	100.000	100.000	100.000	100.000	100.000
32	99.777	100.000	100.000	100.000	100.000	100.000	100.000	100.000	99.938	100.000
36	100.000	100.000	100.000	100.000	100.000	100.000	100.000	100.000	100.000	100.000
40	100.000	100.000	100.000	100.000	100.000	99.936	99.937	100.000	100.000	100.000

Table A.11: Average number of solutions for derating NichePSO in Michalewicz 2D problem domain

R	$\|S\|$									
	4	8	12	16	20	24	28	32	36	40
4	4.200	7.200	11.067	13.100	16.300	17.667	21.800	24.767	26.333	27.267
8	6.833	13.133	18.533	24.100	29.267	32.833	37.767	42.400	46.033	50.167
12	9.133	17.600	26.600	33.467	41.033	48.233	53.067	57.200	65.567	69.867
16	12.633	23.633	34.067	42.333	50.467	59.667	66.967	74.267	78.300	87.800
20	14.233	28.600	38.833	50.567	60.867	68.733	78.067	87.367	94.533	102.100
24	17.500	32.867	46.900	59.333	68.800	80.200	90.200	98.967	105.800	113.000
28	19.367	37.867	52.733	67.067	79.300	90.467	101.233	109.367	117.800	123.300
32	21.633	40.333	58.167	73.933	89.867	99.433	108.133	120.867	129.000	134.900
36	23.833	45.867	63.700	80.967	95.633	108.633	117.500	127.733	137.300	147.367
40	27.100	50.600	68.333	87.600	102.433	116.933	127.167	134.700	146.300	152.667

Table A.12: Standard deviation of average number of solutions for derating NichePSO in Michalewicz 2D problem domain

R	$\|S\|$									
	4	8	12	16	20	24	28	32	36	40
4	1.352	1.629	2.034	2.646	2.620	2.773	2.907	2.994	4.206	3.886
8	1.650	2.101	2.600	2.526	3.695	3.327	3.869	3.751	4.422	3.662
12	2.259	2.959	3.869	3.787	4.537	3.620	5.278	5.442	7.037	6.220
16	1.974	3.577	3.778	4.189	3.523	5.922	5.605	5.577	4.902	5.757
20	2.435	3.572	3.296	4.749	5.099	5.965	4.976	5.487	6.838	6.009
24	2.435	3.353	4.359	4.972	5.580	5.840	6.122	6.349	6.300	7.529
28	2.930	3.639	4.518	6.226	6.876	5.596	5.669	5.654	6.003	6.734
32	4.098	5.449	5.163	5.577	4.799	7.235	6.888	6.379	7.497	6.547
36	2.566	4.418	5.586	5.654	6.926	6.217	6.438	7.511	5.922	8.826
40	3.908	4.418	6.300	6.555	5.233	8.502	8.414	5.089	5.678	9.500

Table A.13: Average accuracy for derating NichePSO in Michalewicz 2D problem domain

R	$\|S\|$									
	4	8	12	16	20	24	28	32	36	40
4	0.064	0.018	0.019	0.014	0.049	0.016	0.033	0.046	0.040	0.013
8	0.011	0.043	0.037	0.037	0.031	0.035	0.036	0.032	0.039	0.039
12	0.030	0.034	0.039	0.023	0.030	0.034	0.040	0.042	0.031	0.047
16	0.032	0.046	0.029	0.037	0.030	0.037	0.039	0.037	0.043	0.042
20	0.031	0.025	0.037	0.027	0.036	0.035	0.045	0.050	0.051	0.044
24	0.030	0.049	0.047	0.051	0.032	0.041	0.040	0.034	0.046	0.038
28	0.029	0.040	0.032	0.047	0.045	0.053	0.039	0.041	0.043	0.052
32	0.030	0.035	0.041	0.032	0.039	0.048	0.044	0.053	0.058	0.050
36	0.029	0.026	0.033	0.040	0.041	0.048	0.047	0.052	0.057	0.053
40	0.050	0.040	0.032	0.038	0.041	0.051	0.052	0.056	0.048	0.057

Table A.14: Standard deviation of average accuracy for derating NichePSO in Michalewicz 2D problem domain

R	$\|S\|$									
	4	8	12	16	20	24	28	32	36	40
4	0.111	0.019	0.020	0.023	0.060	0.021	0.033	0.063	0.043	0.027
8	0.041	0.049	0.041	0.044	0.041	0.050	0.041	0.057	0.051	0.059
12	0.042	0.046	0.064	0.036	0.037	0.044	0.054	0.045	0.045	0.052
16	0.053	0.069	0.043	0.046	0.045	0.045	0.049	0.048	0.050	0.051
20	0.050	0.037	0.044	0.041	0.045	0.052	0.057	0.060	0.064	0.051
24	0.034	0.059	0.059	0.065	0.048	0.056	0.053	0.045	0.055	0.052
28	0.040	0.043	0.040	0.057	0.051	0.060	0.051	0.053	0.051	0.068
32	0.043	0.053	0.040	0.045	0.056	0.052	0.058	0.060	0.069	0.066
36	0.051	0.042	0.043	0.049	0.055	0.059	0.061	0.062	0.071	0.070
40	0.060	0.047	0.047	0.056	0.051	0.062	0.069	0.071	0.059	0.072

Table A.15: Percentage of successfully converged subswarms for derating NichePSO in Michalewicz 2D problem domain

R	$\|S\|$									
	4	8	12	16	20	24	28	32	36	40
4	94.444	97.222	98.193	97.964	96.319	97.736	97.554	96.770	96.582	96.944
8	97.561	96.447	96.583	96.819	97.039	95.635	97.087	96.226	97.176	96.478
12	96.350	95.833	95.614	97.311	96.832	96.821	96.043	96.737	96.543	95.897
16	97.098	94.922	97.065	96.693	96.830	96.872	96.267	96.454	96.467	95.938
20	95.550	97.319	96.395	96.572	96.221	96.508	96.413	95.765	95.310	96.017
24	96.190	94.828	95.522	95.506	96.463	95.844	96.046	96.699	96.093	95.959
28	96.558	95.863	96.839	95.825	96.091	95.689	96.049	96.221	95.671	95.026
32	95.994	95.868	96.046	96.348	95.994	95.977	95.623	95.615	95.245	95.355
36	96.084	96.657	97.017	95.595	95.713	95.367	95.461	95.277	95.096	95.182
40	95.572	96.179	96.829	96.043	96.095	95.182	95.282	94.828	95.603	94.585

Appendix B

Derating NichePSO, Increasing Dimension

This appendix presents the results of the derating NichePSO for the Griewank, MF1, MF3, and Rastrigin in 2D, 3D, and 4D problem domains. All values are rounded-up to three decimal places.

Table B.1: Average number of solutions found for 2D Griewank

R	$\lvert S\rvert$									
	2	4	6	8	10	12	14	16	18	20
2	1.900	3.800	5.267	6.933	8.133	9.167	9.767	11.233	12.200	13.167
4	3.300	6.500	9.167	11.167	12.367	13.233	14.433	15.600	16.033	16.300
6	5.033	8.733	11.300	13.167	14.433	15.667	16.267	16.733	17.333	17.667
8	6.500	10.733	13.767	15.300	15.867	16.233	17.433	17.433	17.833	17.767
10	7.600	12.333	14.733	15.867	17.033	17.567	17.867	17.700	17.833	18.067
12	8.800	13.333	15.300	16.767	16.867	17.567	17.767	18.133	18.000	18.167
14	9.600	13.900	15.833	16.933	17.733	17.900	17.967	18.067	18.167	18.267
16	10.233	14.700	16.500	17.400	17.633	18.067	18.200	18.233	18.300	18.433
18	11.367	15.400	17.000	17.700	17.867	18.233	18.167	18.200	18.267	18.300
20	11.800	15.767	17.233	17.833	18.033	18.233	18.233	18.400	18.300	18.267

Table B.2: Average number of solutions found for 3D Griewank

R	$\lvert S\rvert$									
	2	4	6	8	10	12	14	16	18	20
2	1.933	3.833	5.700	7.667	9.600	11.167	12.933	14.300	16.433	17.867
4	3.833	7.533	11.333	14.533	17.533	20.767	23.900	26.000	28.767	31.967
6	5.767	11.033	16.067	20.600	24.200	29.433	31.500	35.267	38.567	42.933
8	7.700	13.700	20.400	25.833	31.067	34.733	39.800	43.667	46.933	49.800
10	9.133	17.133	24.800	29.900	36.633	40.400	45.600	49.833	54.067	57.800
12	11.000	20.100	28.433	34.867	42.167	46.267	51.500	54.533	59.300	64.567
14	12.167	22.700	31.567	38.933	44.767	49.933	54.800	58.967	65.200	69.733
16	14.167	25.133	34.200	42.167	50.000	55.600	61.267	65.233	68.833	71.500
18	15.667	27.400	37.700	46.167	51.633	57.967	62.267	69.567	72.833	76.867
20	16.500	30.233	39.333	48.100	55.567	62.200	66.967	72.433	76.833	80.133

Table B.3: Average number of solutions found for 4D Griewank

R	$\lvert S\rvert$									
	2	4	6	8	10	12	14	16	18	20
2	2.000	4.000	5.933	7.967	9.867	11.767	13.800	15.333	17.500	19.267
4	4.000	7.833	11.567	15.633	19.167	22.700	26.400	30.233	33.200	36.867
6	5.900	11.567	17.267	22.567	27.967	33.300	38.267	43.133	48.267	52.267
8	7.967	15.067	22.800	29.967	36.267	43.600	49.333	56.500	61.767	67.200
10	9.700	18.833	27.733	36.167	44.600	53.167	60.333	68.033	74.800	80.567
12	11.667	22.400	33.167	42.533	53.133	61.400	69.467	78.333	86.067	93.867
14	13.600	26.400	37.833	49.133	59.967	69.533	79.367	87.567	98.000	105.567
16	15.300	29.600	42.933	55.467	66.467	77.767	87.933	98.933	107.733	116.633
18	17.200	32.933	46.567	59.867	73.767	85.033	96.333	107.833	118.100	127.667
20	19.100	36.133	51.900	66.567	80.233	91.800	103.233	116.267	125.567	136.400

Table B.4: Average accuracy for 2D Griewank

R	$\lvert S\rvert$									
	2	4	6	8	10	12	14	16	18	20
2	0.446	0.441	0.442	0.414	0.420	0.408	0.394	0.407	0.391	0.396
4	0.471	0.376	0.411	0.406	0.402	0.398	0.401	0.387	0.385	0.392
6	0.406	0.420	0.405	0.412	0.399	0.391	0.394	0.391	0.386	0.384
8	0.399	0.410	0.400	0.398	0.394	0.394	0.388	0.387	0.385	0.386
10	0.404	0.407	0.398	0.392	0.389	0.387	0.387	0.386	0.387	0.384
12	0.416	0.398	0.396	0.396	0.387	0.386	0.387	0.387	0.386	0.382
14	0.396	0.397	0.391	0.385	0.384	0.386	0.383	0.386	0.385	0.385
16	0.393	0.394	0.392	0.386	0.388	0.383	0.385	0.383	0.383	0.384
18	0.400	0.401	0.392	0.388	0.385	0.384	0.384	0.384	0.385	0.385
20	0.415	0.395	0.388	0.387	0.386	0.385	0.386	0.385	0.386	0.387

Table B.5: Average accuracy for 3D Griewank

R	$\lvert S\rvert$									
	2	4	6	8	10	12	14	16	18	20
2	0.372	0.342	0.355	0.377	0.372	0.379	0.372	0.362	0.369	0.374
4	0.361	0.373	0.387	0.375	0.374	0.369	0.364	0.366	0.361	0.370
6	0.371	0.381	0.374	0.367	0.361	0.371	0.373	0.368	0.367	0.364
8	0.345	0.357	0.376	0.378	0.363	0.368	0.373	0.368	0.364	0.368
10	0.371	0.363	0.366	0.370	0.367	0.375	0.372	0.370	0.368	0.368
12	0.374	0.374	0.371	0.367	0.367	0.367	0.369	0.368	0.361	0.366
14	0.375	0.364	0.362	0.369	0.363	0.364	0.360	0.366	0.366	0.368
16	0.369	0.370	0.374	0.369	0.367	0.368	0.371	0.365	0.368	0.371
18	0.376	0.370	0.367	0.368	0.370	0.369	0.364	0.360	0.364	0.364
20	0.374	0.367	0.371	0.372	0.371	0.364	0.369	0.364	0.371	0.367

Table B.6: Average accuracy for 4D Griewank

R	$\lvert S\rvert$									
	2	4	6	8	10	12	14	16	18	20
2	0.330	0.308	0.314	0.327	0.326	0.325	0.325	0.319	0.322	0.326
4	0.348	0.322	0.320	0.315	0.324	0.325	0.321	0.325	0.317	0.322
6	0.307	0.321	0.329	0.321	0.325	0.321	0.327	0.322	0.321	0.317
8	0.324	0.323	0.322	0.313	0.321	0.326	0.320	0.316	0.317	0.319
10	0.325	0.313	0.321	0.317	0.325	0.318	0.319	0.321	0.317	0.320
12	0.315	0.317	0.328	0.325	0.316	0.320	0.321	0.323	0.320	0.324
14	0.319	0.323	0.315	0.320	0.319	0.318	0.320	0.317	0.319	0.317
16	0.322	0.325	0.321	0.317	0.318	0.315	0.318	0.320	0.316	0.320
18	0.317	0.323	0.329	0.320	0.319	0.322	0.321	0.318	0.319	0.321
20	0.325	0.317	0.317	0.319	0.322	0.321	0.321	0.317	0.320	0.319

Table B.7: Standard deviation of average number of solutions for 2D Griewank

R	$\lvert S\rvert$									
	2	4	6	8	10	12	14	16	18	20
2	0.322	0.415	0.788	0.871	0.910	1.130	1.486	1.114	0.965	1.189
4	0.719	0.910	1.246	1.497	1.099	1.486	1.313	1.313	1.189	1.034
6	0.616	0.983	1.099	1.543	1.083	1.067	0.947	0.871	0.851	0.851
8	0.910	1.287	1.742	1.543	1.259	1.287	0.947	0.871	0.491	0.587
10	0.910	1.587	1.145	0.983	1.218	0.743	0.587	0.670	0.616	0.525
12	1.377	1.377	1.402	1.083	0.910	0.743	0.695	0.525	0.643	0.415
14	1.145	1.671	1.130	1.083	0.587	0.616	0.616	0.371	0.616	0.455
16	1.819	1.246	1.232	0.947	0.557	0.643	0.616	0.643	0.616	0.788
18	1.326	1.017	1.050	0.670	0.643	0.587	0.557	0.491	0.643	0.616
20	1.326	1.339	0.947	0.809	0.670	0.525	0.525	0.587	0.491	0.643

Table B.8: Standard deviation of average number of solutions for 3D Griewank

R	$\lvert S\rvert$									
	2	4	6	8	10	12	14	16	18	20
2	0.263	0.415	0.557	0.491	0.587	0.851	0.743	1.130	1.050	1.259
4	0.491	0.643	0.766	0.910	1.619	1.554	1.520	2.000	1.875	2.356
6	0.525	0.809	1.619	1.232	1.956	2.259	2.334	2.435	2.546	3.129
8	0.557	1.099	1.597	1.810	1.857	2.560	2.553	2.907	3.006	3.509
10	0.743	1.742	1.732	2.484	2.471	2.626	2.197	3.124	3.778	3.306
12	0.743	1.474	2.181	2.994	3.399	3.270	2.994	3.787	4.181	3.948
14	1.000	1.885	2.546	2.983	3.629	3.533	3.091	3.987	3.557	4.275
16	1.000	2.000	2.553	2.871	3.494	3.686	4.379	4.283	4.295	4.749
18	1.189	2.259	2.442	3.200	4.499	3.970	3.869	4.426	3.577	4.433
20	1.682	1.857	2.798	3.672	3.582	2.954	3.368	4.579	4.979	6.074

Table B.9: Standard deviation of average number of solutions for 4D Griewank

R	$\lvert S\rvert$									
	2	4	6	8	10	12	14	16	18	20
2	0.000	0.000	0.263	0.186	0.371	0.695	0.415	0.719	0.871	0.788
4	0.000	0.415	0.643	0.616	0.851	1.099	1.259	0.910	0.851	1.486
6	0.322	0.643	0.830	1.287	1.402	1.650	1.875	2.407	2.464	3.140
8	0.186	0.910	1.300	1.426	1.875	1.365	1.752	2.304	2.393	3.317
10	0.491	1.189	1.390	1.921	2.000	2.282	1.848	3.337	3.057	3.074
12	0.616	0.871	1.712	2.068	2.166	2.936	3.695	3.000	3.639	4.060
14	0.643	1.017	1.629	2.491	2.659	3.107	2.798	3.184	4.251	4.480
16	0.965	1.682	2.259	2.334	3.311	3.074	3.948	4.119	3.085	4.881
18	0.766	1.509	2.626	2.924	3.259	3.728	3.557	3.499	5.082	5.116
20	0.766	1.819	2.526	3.195	3.404	3.961	5.186	4.579	4.742	5.596

Table B.10: Standard deviation of average accuracy for 2D Griewank

R	$\lvert S\rvert$									
	2	4	6	8	10	12	14	16	18	20
2	0.000	0.001	0.002	0.003	0.002	0.005	0.001	0.000	0.000	0.001
4	0.002	0.003	0.002	0.002	0.005	0.001	0.003	0.003	0.003	0.002
6	0.002	0.004	0.002	0.003	0.004	0.001	0.003	0.002	0.007	0.002
8	0.002	0.003	0.003	0.005	0.003	0.007	0.001	0.005	0.008	0.001
10	0.002	0.007	0.004	0.002	0.005	0.005	0.005	0.004	0.005	0.003
12	0.009	0.001	0.006	0.005	0.007	0.003	0.005	0.004	0.005	0.007
14	0.007	0.003	0.006	0.006	0.004	0.002	0.005	0.004	0.002	0.002
16	0.004	0.005	0.003	0.005	0.006	0.005	0.006	0.006	0.004	0.005
18	0.008	0.004	0.004	0.002	0.006	0.003	0.004	0.003	0.004	0.005
20	0.004	0.007	0.003	0.005	0.006	0.003	0.005	0.002	0.002	0.002

Table B.11: Standard deviation of average accuracy for 3D Griewank

R	$\|S\|$									
	2	4	6	8	10	12	14	16	18	20
2	0.000	0.012	0.002	0.000	0.000	0.001	0.000	0.001	0.002	0.000
4	0.005	0.000	0.002	0.002	0.001	0.002	0.001	0.000	0.005	0.001
6	0.005	0.003	0.001	0.005	0.002	0.002	0.001	0.002	0.001	0.002
8	0.005	0.002	0.002	0.001	0.001	0.001	0.002	0.001	0.002	0.001
10	0.006	0.005	0.002	0.001	0.002	0.001	0.001	0.002	0.002	0.001
12	0.004	0.004	0.003	0.002	0.002	0.001	0.001	0.002	0.002	0.002
14	0.005	0.004	0.003	0.003	0.002	0.002	0.002	0.001	0.001	0.001
16	0.007	0.002	0.001	0.002	0.001	0.002	0.001	0.003	0.001	0.001
18	0.003	0.001	0.004	0.004	0.003	0.001	0.002	0.002	0.001	0.002
20	0.007	0.001	0.001	0.001	0.002	0.001	0.001	0.003	0.003	0.001

Table B.12: Standard deviation of average accuracy for 4D Griewank

R	$\|S\|$									
	2	4	6	8	10	12	14	16	18	20
2	0.003	0.008	0.001	0.000	0.000	0.001	0.001	0.008	0.004	0.001
4	0.002	0.003	0.002	0.002	0.001	0.001	0.003	0.003	0.001	0.001
6	0.004	0.004	0.002	0.008	0.001	0.004	0.003	0.001	0.001	0.004
8	0.003	0.002	0.005	0.007	0.003	0.005	0.002	0.001	0.002	0.002
10	0.004	0.007	0.002	0.003	0.001	0.001	0.003	0.001	0.001	0.002
12	0.005	0.003	0.001	0.003	0.002	0.003	0.004	0.002	0.001	0.001
14	0.003	0.005	0.002	0.002	0.001	0.003	0.001	0.003	0.002	0.002
16	0.004	0.003	0.001	0.002	0.001	0.001	0.001	0.002	0.003	0.002
18	0.007	0.001	0.002	0.001	0.001	0.001	0.001	0.001	0.001	0.001
20	0.002	0.003	0.001	0.001	0.001	0.001	0.001	0.001	0.001	0.001

Table B.13: Percentage of converged subswarms for 2D Griewank

R	$\lvert S\rvert$									
	2	4	6	8	10	12	14	16	18	20
2	17.544	17.544	16.456	20.673	21.311	22.182	22.184	23.145	23.497	23.291
4	20.202	28.718	22.182	22.388	22.372	23.929	22.171	23.718	23.077	22.290
6	25.828	22.137	24.779	21.519	22.633	23.191	22.541	22.510	22.500	23.019
8	27.692	24.224	22.760	22.440	22.689	22.177	21.989	22.371	22.617	22.326
10	25.439	22.703	22.851	22.689	22.505	22.391	22.015	22.599	22.243	22.325
12	22.348	24.000	22.658	21.272	23.320	22.770	22.702	21.875	22.222	22.385
14	25.347	23.501	23.579	23.622	22.932	22.160	22.820	22.140	22.018	22.080
16	26.384	23.810	22.828	22.414	21.928	22.325	22.161	22.486	22.222	21.881
18	24.340	22.727	22.745	22.222	22.388	22.303	22.569	22.161	22.263	22.222
20	21.469	22.410	22.437	22.243	21.996	22.121	21.938	21.920	21.858	21.898

Table B.14: Percentage of converged subswarms for 3D Griewank

R	$\lvert S\rvert$									
	2	4	6	8	10	12	14	16	18	20
2	8.621	11.304	9.942	6.522	7.986	6.269	6.701	9.790	7.505	7.649
4	11.304	8.407	8.235	8.257	6.464	6.902	10.181	8.462	10.081	8.133
6	8.671	8.157	6.432	7.443	8.953	7.701	7.196	8.129	8.557	8.230
8	12.121	9.489	6.373	7.355	9.549	7.582	7.621	8.168	9.091	8.701
10	9.124	9.144	9.677	8.807	8.644	7.508	8.187	7.425	8.385	8.478
12	7.576	7.629	8.910	8.413	8.221	7.853	8.285	9.108	9.444	8.622
14	10.685	8.223	9.293	8.818	8.712	8.879	9.611	8.310	8.589	9.082
16	8.235	8.621	7.505	8.379	8.533	8.933	8.052	8.942	8.910	8.438
18	8.298	8.881	9.726	9.025	8.586	8.396	8.887	9.344	9.016	9.801
20	8.081	9.041	7.542	8.593	7.798	9.325	9.009	9.940	8.807	9.484

Table B.15: Percentage of converged subswarms for 4D Griewank

R	$\lvert S\rvert$									
	2	4	6	8	10	12	14	16	18	20
2	10.000	9.167	8.427	10.042	7.770	7.649	6.522	7.609	6.667	7.785
4	6.667	9.787	7.493	9.595	7.304	7.195	7.828	7.166	8.835	8.499
6	10.734	6.916	7.722	8.124	8.701	8.408	7.753	8.346	8.287	8.992
8	10.042	7.743	8.187	8.565	8.364	7.339	8.176	8.260	8.851	8.234
10	8.591	9.558	8.894	8.018	7.773	8.527	8.066	7.398	8.601	8.026
12	10.857	9.673	8.543	7.915	8.218	8.360	7.486	8.213	7.901	8.168
14	9.804	9.343	9.868	7.598	7.671	8.581	8.694	8.603	7.687	7.610
16	9.150	9.459	8.307	8.413	8.325	8.958	8.719	8.187	8.137	8.002
18	11.047	9.109	7.874	8.463	8.495	8.389	7.889	8.223	8.213	8.016
20	10.471	9.041	9.056	8.963	8.184	8.061	8.428	8.257	8.654	8.260

Table B.16: Average number of solutions found for 2D MF1

R	$\lvert S\rvert$									
	2	4	6	8	10	12	14	16	18	20
2	1.900	3.600	5.067	6.533	7.467	8.300	8.933	9.900	10.933	11.700
4	3.567	6.267	9.000	10.067	11.567	12.867	13.800	13.733	14.300	14.933
6	5.200	8.333	11.000	12.333	14.000	14.600	14.933	15.267	15.633	15.433
8	6.400	10.233	12.600	13.900	15.100	15.433	15.833	16.033	15.800	16.233
10	8.033	12.067	13.900	14.433	15.433	15.733	16.133	16.267	16.067	16.200
12	8.700	12.500	14.733	15.433	15.967	16.033	16.100	16.267	16.233	16.400
14	9.733	13.033	15.067	16.033	16.167	16.133	16.400	16.667	16.300	16.500
16	10.000	13.933	15.400	15.867	15.867	16.300	16.433	16.267	16.500	16.300
18	10.767	14.467	15.767	16.167	16.167	16.633	16.567	16.633	16.667	16.400
20	11.733	15.000	15.833	16.233	16.167	16.533	16.533	16.467	16.500	16.900

Table B.17: Average number of solutions found for 3D MF1

R	$\lvert S\rvert$									
	2	4	6	8	10	12	14	16	18	20
2	2.000	3.833	5.800	7.567	9.333	11.000	12.467	14.600	16.167	17.367
4	3.967	7.733	11.233	14.333	17.567	20.033	23.600	26.433	28.433	30.900
6	5.833	10.933	16.000	20.567	24.733	28.433	32.200	35.167	38.200	40.500
8	7.833	14.567	20.100	25.900	30.767	35.400	39.567	42.900	44.933	48.300
10	9.167	17.500	24.833	30.633	35.333	40.633	45.133	48.667	51.567	54.467
12	10.700	20.133	28.233	35.567	41.833	45.467	50.133	53.200	56.633	59.400
14	12.567	22.667	31.400	38.400	45.567	49.700	54.300	58.300	60.400	62.667
16	14.433	25.533	34.833	42.800	48.767	52.433	58.133	60.900	64.433	65.467
18	16.433	28.467	36.933	45.600	50.667	56.567	60.367	62.833	65.333	69.067
20	17.333	30.967	41.133	49.467	54.500	58.267	61.900	65.933	67.900	70.767

Table B.18: Average number of solutions found for 4D MF1

R	$\lvert S\rvert$									
	2	4	6	8	10	12	14	16	18	20
2	2.000	4.000	5.933	7.867	9.800	11.867	13.633	15.633	17.567	19.467
4	4.000	7.933	11.700	15.700	19.567	23.300	27.233	30.967	34.500	37.333
6	6.000	11.767	17.767	23.200	28.633	34.467	39.433	45.100	49.800	55.033
8	7.833	15.700	23.133	30.233	38.100	44.733	51.567	58.867	64.967	71.333
10	9.800	19.433	28.700	37.433	46.767	55.867	63.367	71.367	79.333	86.367
12	11.667	23.133	34.033	44.933	55.000	64.600	75.067	84.233	92.567	101.367
14	13.733	26.800	39.033	51.133	63.400	74.600	84.967	95.767	105.700	115.167
16	15.667	30.333	44.700	58.267	70.900	83.833	96.033	107.100	115.467	126.867
18	17.367	34.167	50.000	65.700	78.600	92.267	104.667	116.167	128.467	139.333
20	19.367	37.700	54.733	70.933	86.733	99.900	115.033	127.900	138.867	152.267

Table B.19: Average accuracy for 2D MF1

R	$\lvert S\rvert$									
	2	4	6	8	10	12	14	16	18	20
2	0.646	0.307	0.218	0.216	0.290	0.271	0.221	0.144	0.203	0.328
4	0.235	0.210	0.343	0.503	0.496	0.358	0.225	0.170	0.082	0.229
6	0.419	0.406	0.397	0.350	0.241	0.267	0.289	0.345	0.262	0.289
8	0.538	0.461	0.246	0.257	0.372	0.358	0.310	0.399	0.218	0.413
10	0.231	0.549	0.390	0.346	0.276	0.403	0.378	0.467	0.199	0.312
12	0.368	0.334	0.402	0.446	0.517	0.490	0.282	0.445	0.462	0.543
14	0.543	0.259	0.456	0.505	0.567	0.522	0.482	0.624	0.314	0.652
16	0.644	0.261	0.410	0.322	0.421	0.509	0.571	0.528	0.589	0.375
18	0.435	0.408	0.446	0.510	0.265	0.802	0.703	0.571	0.636	0.560
20	0.403	0.370	0.402	0.475	0.323	0.746	0.619	0.505	0.573	0.873

Table B.20: Average accuracy for 3D MF1

R	$\lvert S\rvert$									
	2	4	6	8	10	12	14	16	18	20
2	0.875	0.669	1.013	0.730	1.040	1.239	1.121	1.321	1.300	1.197
4	0.782	0.923	1.735	1.430	1.449	1.061	1.315	1.342	1.042	1.016
6	1.385	1.171	1.251	1.295	1.371	1.073	1.170	1.053	1.131	1.049
8	1.678	1.457	1.288	1.575	1.271	1.571	1.309	1.596	1.298	1.432
10	1.307	0.975	1.187	1.268	1.317	1.535	1.084	1.412	1.244	1.532
12	1.133	1.598	1.501	1.389	1.441	1.403	1.444	1.568	1.542	1.627
14	1.604	1.230	1.533	1.549	1.562	1.472	1.518	1.657	1.942	1.664
16	1.348	1.348	1.256	1.520	1.460	1.439	1.635	1.785	1.786	1.906
18	1.202	1.391	1.410	1.616	1.481	1.687	1.796	1.776	1.746	1.960
20	1.085	1.390	1.536	1.392	1.494	1.472	1.618	1.966	1.781	2.169

Table B.21: Average accuracy for 4D MF1

R	$\lvert S\rvert$									
	2	4	6	8	10	12	14	16	18	20
2	1.596	1.320	1.409	1.398	1.557	1.430	1.387	1.799	1.684	1.668
4	1.718	1.748	1.972	1.455	1.722	1.572	1.975	1.787	1.599	1.804
6	1.720	2.326	1.900	1.915	1.937	1.684	1.623	1.641	1.595	1.881
8	2.341	1.578	1.747	1.984	1.504	1.422	1.934	1.765	1.765	1.788
10	1.411	1.327	1.898	1.797	1.823	1.767	1.834	1.659	1.850	1.676
12	2.443	1.938	1.814	1.885	1.793	1.626	1.940	1.923	1.896	1.823
14	1.371	2.222	1.869	1.849	1.789	1.925	1.826	1.872	1.773	1.963
16	1.466	1.542	1.744	1.841	2.391	1.875	1.804	2.076	1.860	1.997
18	1.796	1.873	2.013	1.979	1.802	1.908	1.792	2.075	2.094	1.980
20	1.852	1.632	1.832	1.802	1.808	2.027	1.904	1.900	1.940	2.229

Table B.22: Standard deviation of average number of solutions for 2D MF1

R	$\lvert S\rvert$									
	2	4	6	8	10	12	14	16	18	20
2	0.322	0.587	0.830	1.017	1.114	1.067	1.050	1.160	1.174	1.352
4	0.525	1.114	0.910	1.287	1.554	1.203	1.474	1.203	1.067	1.083
6	0.891	1.067	1.083	1.218	1.050	1.174	1.174	0.983	0.670	0.788
8	0.788	1.174	1.174	1.300	0.965	0.830	0.719	0.670	0.670	0.643
10	0.928	1.259	1.246	1.174	0.743	0.743	0.643	0.643	0.371	0.491
12	1.130	1.232	1.174	0.830	0.809	0.766	0.322	0.587	0.525	0.643
14	1.114	1.300	1.083	0.670	0.809	0.525	0.587	0.851	0.491	0.830
16	1.203	1.050	0.983	0.788	0.788	0.616	0.695	0.525	0.743	0.491
18	1.017	1.050	0.788	1.000	0.415	1.034	0.643	0.719	0.719	0.587
20	1.287	0.788	0.719	0.695	0.415	0.788	0.788	0.743	0.587	0.891

Table B.23: Standard deviation of average number of solutions for 3D MF1

R	$\lvert S\rvert$									
	2	4	6	8	10	12	14	16	18	20
2	0.000	0.415	0.491	0.643	0.719	0.743	1.232	1.114	1.300	1.246
4	0.186	0.455	0.830	1.218	1.486	1.791	1.365	1.948	1.912	2.076
6	0.415	0.947	0.947	1.203	2.150	2.505	2.109	2.871	2.158	2.853
8	0.415	1.083	1.300	2.142	1.800	2.150	2.600	2.633	2.716	3.264
10	0.766	1.365	1.474	1.712	2.859	2.977	2.477	3.459	2.289	3.107
12	1.130	1.554	1.640	2.349	2.205	2.924	2.533	3.358	3.368	3.332
14	0.871	1.791	2.435	2.767	2.841	3.102	2.297	2.620	3.523	2.512
16	0.983	2.084	2.646	3.317	2.742	3.227	3.085	3.644	3.732	3.195
18	1.017	2.133	2.464	2.626	3.439	2.754	3.200	2.710	3.792	3.311
20	1.402	1.956	3.504	2.948	2.792	3.173	3.449	3.562	3.057	3.248

Table B.24: Standard deviation of average number of solutions for 4D MF1

R	$\lvert S\rvert$									
	2	4	6	8	10	12	14	16	18	20
2	0.000	0.000	0.263	0.371	0.415	0.455	0.670	0.616	0.743	0.788
4	0.000	0.263	0.491	0.557	0.695	0.719	0.830	0.891	1.203	1.629
6	0.000	0.455	0.455	1.099	1.000	1.203	1.462	1.587	1.921	1.903
8	0.415	0.557	0.910	1.287	1.692	1.800	1.857	2.068	1.712	3.168
10	0.415	0.587	1.160	1.414	1.203	1.702	2.385	2.672	2.710	2.871
12	0.557	0.947	1.218	1.232	2.334	2.560	2.639	3.173	2.349	3.200
14	0.587	0.965	1.067	1.965	2.393	2.742	3.200	3.384	3.146	3.439
16	0.557	1.130	1.848	2.449	3.102	2.895	3.102	4.140	4.299	4.698
18	0.928	1.099	1.597	2.442	2.754	3.280	3.399	3.755	4.792	4.860
20	0.719	1.377	2.017	2.213	2.652	4.247	4.917	3.389	4.094	5.960

Table B.25: Standard deviation of average accuracy for 2D MF1

R	$\|S\|$									
	2	4	6	8	10	12	14	16	18	20
2	0.160	0.039	0.013	0.006	0.011	0.053	0.001	0.012	0.000	0.030
4	0.061	0.014	0.019	0.021	0.021	0.016	0.054	0.052	0.047	0.035
6	0.102	0.036	0.048	0.020	0.035	0.031	0.038	0.049	0.046	0.038
8	0.124	0.061	0.020	0.081	0.038	0.040	0.039	0.056	0.063	0.070
10	0.062	0.087	0.043	0.039	0.039	0.035	0.063	0.040	0.039	0.033
12	0.046	0.061	0.026	0.042	0.043	0.097	0.099	0.077	0.099	0.176
14	0.116	0.045	0.065	0.079	0.084	0.101	0.079	0.148	0.044	0.188
16	0.141	0.046	0.040	0.040	0.088	0.106	0.096	0.113	0.142	0.052
18	0.071	0.072	0.089	0.098	0.035	0.166	0.102	0.155	0.095	0.144
20	0.069	0.047	0.076	0.105	0.042	0.166	0.141	0.148	0.150	0.251

Table B.26: Standard deviation of average accuracy for 3D MF1

R	$\|S\|$									
	2	4	6	8	10	12	14	16	18	20
2	0.188	0.120	0.059	0.011	0.004	0.154	0.237	0.154	0.026	0.068
4	0.061	0.141	0.179	0.062	0.156	0.019	0.028	0.065	0.038	0.168
6	0.131	0.057	0.043	0.167	0.067	0.106	0.117	0.022	0.126	0.045
8	0.351	0.212	0.184	0.055	0.194	0.159	0.080	0.036	0.181	0.170
10	0.164	0.156	0.063	0.112	0.071	0.164	0.122	0.102	0.093	0.130
12	0.069	0.192	0.119	0.156	0.087	0.072	0.157	0.127	0.161	0.237
14	0.108	0.140	0.167	0.141	0.190	0.241	0.193	0.142	0.287	0.094
16	0.128	0.076	0.199	0.159	0.087	0.189	0.215	0.180	0.171	0.298
18	0.104	0.114	0.112	0.232	0.134	0.133	0.277	0.259	0.200	0.249
20	0.517	0.068	0.226	0.123	0.119	0.181	0.135	0.209	0.280	0.393

Table B.27: Standard deviation of average accuracy for 4D MF1

R	$\|S\|$									
	2	4	6	8	10	12	14	16	18	20
2	0.156	0.049	0.275	0.078	0.019	0.232	0.068	0.030	0.049	0.107
4	0.113	0.085	0.072	0.142	0.103	0.164	0.071	0.064	0.152	0.090
6	0.156	0.198	0.168	0.143	0.148	0.088	0.168	0.095	0.030	0.060
8	0.132	0.128	0.126	0.151	0.157	0.065	0.057	0.094	0.127	0.143
10	0.175	0.126	0.072	0.137	0.109	0.037	0.078	0.150	0.046	0.087
12	0.570	0.179	0.171	0.161	0.050	0.040	0.125	0.141	0.099	0.100
14	0.260	0.181	0.071	0.187	0.096	0.031	0.113	0.060	0.220	0.113
16	0.074	0.078	0.137	0.131	0.203	0.107	0.168	0.062	0.071	0.225
18	0.246	0.219	0.055	0.055	0.118	0.107	0.224	0.039	0.048	0.142
20	0.299	0.167	0.194	0.118	0.318	0.288	0.175	0.091	0.131	0.109

Table B.28: Percentage of converged subswarms for 2D MF1

R	$\|S\|$									
	2	4	6	8	10	12	14	16	18	20
2	78.947	83.333	91.447	92.347	89.286	91.968	90.299	92.256	94.207	91.453
4	91.589	89.362	89.630	88.411	89.049	89.637	93.478	94.660	94.406	93.304
6	87.179	89.200	89.697	90.270	94.048	93.379	92.188	93.668	94.243	93.952
8	85.938	88.599	90.741	93.046	91.391	91.361	93.053	92.308	93.460	92.197
10	85.477	85.359	88.729	90.531	92.009	90.254	92.355	91.803	95.021	91.358
12	86.973	90.133	88.914	89.633	87.683	91.684	93.789	90.574	90.965	92.073
14	86.301	90.026	91.814	90.644	91.546	90.083	90.854	92.600	92.229	90.909
16	85.000	92.584	91.342	92.857	91.597	89.571	92.495	90.164	91.111	93.252
18	86.997	89.631	90.275	89.278	90.309	86.373	91.147	91.182	91.200	91.463
20	88.920	90.444	90.526	91.170	90.309	90.323	90.726	91.498	92.929	91.519

Table B.29: Percentage of converged subswarms for 3D MF1

R	$\lvert S\rvert$									
	2	4	6	8	10	12	14	16	18	20
2	88.333	87.826	88.506	90.749	86.786	87.879	85.561	87.215	86.804	87.140
4	89.076	87.931	84.273	85.814	84.820	87.188	84.887	85.876	87.573	86.947
6	83.429	86.585	84.792	86.224	87.601	86.987	86.749	88.152	86.911	87.407
8	80.851	84.668	87.231	87.001	85.807	83.992	86.268	85.859	87.834	85.162
10	85.091	85.905	86.443	84.657	85.849	85.234	87.518	85.068	86.619	84.149
12	87.227	81.457	85.714	85.473	84.223	86.070	85.505	84.085	84.285	84.456
14	83.820	86.618	85.032	84.375	83.906	84.507	85.206	83.419	82.340	83.830
16	84.758	84.987	83.732	84.424	84.211	84.997	83.200	82.923	84.066	82.536
18	85.598	85.714	84.206	83.260	84.934	83.913	82.938	82.706	83.112	82.915
20	87.115	85.038	84.360	84.164	85.382	84.153	84.491	82.508	82.327	82.242

Table B.30: Percentage of converged subswarms for 4D MF1

R	$\lvert S\rvert$									
	2	4	6	8	10	12	14	16	18	20
2	75.000	90.833	88.202	83.898	87.075	86.517	84.352	85.714	85.769	84.932
4	85.000	86.975	82.906	87.898	85.009	84.835	83.109	85.145	85.024	85.089
6	82.778	83.286	84.991	84.339	84.051	86.460	85.123	85.661	84.739	83.525
8	80.851	84.289	84.438	84.234	85.652	86.513	83.775	85.504	84.915	84.766
10	87.755	88.336	83.972	83.704	84.533	84.666	84.114	85.287	84.496	85.257
12	83.143	84.150	84.329	83.828	84.303	85.707	83.570	83.261	84.012	85.761
14	86.650	83.085	84.372	84.485	83.912	84.808	85.524	84.337	84.831	82.952
16	85.745	85.714	85.235	85.469	82.840	83.817	84.832	83.038	84.700	83.631
18	86.564	84.390	84.000	84.120	84.648	84.574	85.064	82.869	83.731	83.182
20	85.370	85.676	84.227	84.352	84.435	83.717	83.831	83.763	83.413	82.881

Table B.31: Average number of solutions found for 2D MF3

R	$\lvert S\rvert$									
	2	4	6	8	10	12	14	16	18	20.000
2	1.933	3.567	5.000	6.200	7.367	7.967	8.933	9.800	10.133	10.967
4	3.800	6.300	8.267	9.033	11.400	11.833	12.667	12.933	14.133	14.233
6	5.833	8.700	10.667	13.400	13.700	14.500	15.333	15.400	15.900	16.267
8	6.133	9.567	11.567	13.167	14.300	14.567	15.133	15.633	15.567	16.533
10	7.133	10.633	13.033	14.233	15.067	15.100	15.533	16.133	16.600	17.000
12	8.033	11.467	13.900	14.933	15.100	16.167	16.433	16.967	16.900	17.000
14	9.367	12.467	14.400	15.367	15.867	16.300	16.533	17.233	17.033	17.200
16	9.700	12.833	14.700	15.900	16.233	16.900	16.500	17.200	17.633	17.233
18	10.100	14.200	15.133	15.533	16.667	16.667	17.400	17.367	17.500	18.033
20	10.767	13.700	15.333	15.933	16.767	16.867	17.500	17.333	17.667	18.167

Table B.32: Average number of solutions found for 3D MF3

R	$\lvert S\rvert$									
	2	4	6	8	10	12	14	16	18	20
2	2.000	3.867	5.667	7.433	9.467	11.167	12.533	14.100	15.667	17.100
4	3.933	7.433	11.033	14.267	16.933	19.933	22.133	24.833	27.833	30.100
6	6.200	13.700	19.600	23.100	28.500	30.133	33.433	40.200	42.967	44.833
8	7.533	13.400	19.567	24.700	30.067	33.200	37.233	40.267	43.733	46.700
10	9.200	17.433	23.300	29.567	34.533	38.833	42.167	46.533	49.033	52.200
12	10.900	19.300	27.400	32.767	38.433	42.500	48.367	50.567	53.567	57.533
14	12.233	22.767	31.000	37.033	43.400	47.000	50.900	56.133	57.600	61.433
16	14.267	24.300	34.200	40.167	45.500	51.033	56.033	57.533	61.233	65.833
18	15.400	26.600	34.900	42.633	49.033	54.067	56.633	60.667	65.533	67.567
20	17.200	29.933	38.867	45.167	51.533	55.933	61.000	64.933	69.400	68.733

Table B.33: Average number of solutions found for 4D MF3

R	$\lvert S\rvert$									
	2	4	6	8	10	12	14	16	18	20
2	2.000	4.000	5.933	7.867	9.800	11.867	13.633	15.633	17.567	19.467
4	4.000	7.933	11.700	15.700	19.567	23.300	27.233	30.967	34.500	37.333
6	6.000	11.767	17.767	23.200	28.633	34.467	39.433	45.100	49.800	55.033
8	7.833	15.700	23.133	30.233	38.100	44.733	51.567	58.867	64.967	71.333
10	9.800	19.433	28.700	37.433	46.767	55.867	63.367	71.367	79.333	86.367
12	11.667	23.133	34.033	44.933	55.000	64.600	75.067	84.233	92.567	101.367
14	13.733	26.800	39.033	51.133	63.400	74.600	84.967	95.767	105.700	115.167
16	15.667	30.333	44.700	58.267	70.900	83.833	96.033	107.100	115.467	126.867
18	17.367	34.167	50.000	65.700	78.600	92.267	104.667	116.167	128.467	139.333
20	19.367	37.700	54.733	70.933	86.733	99.900	115.033	127.900	138.867	152.267

Table B.34: Average accuracy for 2D MF3

R	$\lvert S\rvert$									
	2	4	6	8	10	12	14	16	18	20
2	0.591	0.894	0.363	0.946	0.487	0.210	0.268	0.418	0.415	0.326
4	0.389	0.168	0.409	0.240	0.295	0.473	0.609	0.199	0.589	0.491
6	0.550	0.479	0.408	0.493	0.260	0.477	0.594	0.459	0.546	0.722
8	0.665	0.361	0.276	0.369	0.401	0.558	0.498	0.543	0.473	0.579
10	0.304	0.659	0.367	0.663	0.603	0.464	0.581	0.537	0.933	0.929
12	0.862	0.559	0.600	0.614	0.638	0.597	0.655	0.856	0.669	0.896
14	0.645	0.444	0.827	0.667	0.820	0.632	0.643	0.824	0.816	0.720
16	0.649	0.425	0.582	0.886	0.776	1.078	0.685	0.879	0.937	0.944
18	0.776	0.667	0.539	0.670	0.791	0.980	1.046	0.988	1.296	1.150
20	0.685	0.393	0.464	0.558	0.922	0.766	1.041	1.034	1.035	1.301

Table B.35: Average accuracy for 3D MF3

R	$\lvert S\rvert$									
	2	4	6	8	10	12	14	16	18	20
2	1.352	1.319	0.902	1.198	1.542	1.844	1.870	1.381	1.288	1.104
4	2.265	1.505	1.465	1.592	1.476	1.266	1.259	1.566	1.640	1.512
6	1.896	1.625	1.014	1.546	1.285	1.496	1.589	1.647	1.613	1.837
8	1.482	1.374	1.371	1.586	1.477	1.612	1.671	1.837	1.694	1.833
10	1.288	1.319	1.412	1.784	1.663	1.613	1.794	1.933	1.918	1.921
12	1.201	1.733	1.304	1.268	1.625	1.817	1.816	1.847	2.077	2.099
14	1.686	1.308	1.645	1.384	1.968	1.847	1.885	2.024	1.905	2.205
16	1.803	1.589	1.766	1.718	1.708	2.149	2.043	2.296	2.129	2.352
18	1.508	1.254	1.569	1.848	1.909	2.034	2.217	2.162	2.261	2.461
20	1.468	1.652	1.689	1.963	2.211	1.935	2.563	2.517	2.511	2.437

Table B.36: Average accuracy for 4D MF3

R	$\lvert S\rvert$									
	2	4	6	8	10	12	14	16	18	20
2	1.596	1.320	1.409	1.398	1.557	1.430	1.387	1.799	1.684	1.668
4	1.718	1.748	1.972	1.455	1.722	1.572	1.975	1.787	1.599	1.804
6	1.720	2.326	1.900	1.915	1.937	1.684	1.623	1.641	1.595	1.881
8	2.341	1.578	1.747	1.984	1.504	1.422	1.934	1.765	1.765	1.788
10	1.411	1.327	1.898	1.797	1.823	1.767	1.834	1.659	1.850	1.676
12	2.443	1.938	1.814	1.885	1.793	1.626	1.940	1.923	1.896	1.823
14	1.371	2.222	1.869	1.849	1.789	1.925	1.826	1.872	1.773	1.963
16	1.466	1.542	1.744	1.841	2.391	1.875	1.804	2.076	1.860	1.997
18	1.796	1.873	2.013	1.979	1.802	1.908	1.792	2.075	2.094	1.980
20	1.852	1.632	1.832	1.802	1.808	2.027	1.904	1.900	1.940	2.229

Table B.37: Standard deviation of average number of solutions for 2D MF3

R	$\lvert S\rvert$									
	2	4	6	8	10	12	14	16	18	20
2	0.263	0.587	0.871	0.809	1.189	1.000	1.174	1.352	1.554	1.300
4	0.557	1.000	1.232	1.326	1.145	1.497	1.246	1.287	1.414	1.339
6	1.099	1.130	0.965	1.462	1.067	1.462	1.273	1.017	0.965	1.287
8	0.871	1.390	1.390	1.474	1.450	1.339	1.203	1.218	1.339	0.830
10	1.050	1.300	1.218	1.313	1.174	1.218	1.083	0.947	1.017	1.203
12	1.300	1.203	1.326	1.083	1.000	1.218	1.174	1.218	1.067	1.050
14	1.326	1.486	1.083	1.300	1.050	1.246	0.910	1.365	1.300	1.099
16	1.326	1.273	1.352	1.300	1.438	1.402	1.017	1.130	1.300	1.017
18	1.377	1.377	1.509	1.259	1.402	1.160	1.838	1.326	1.050	1.497
20	1.365	1.189	1.273	1.050	1.259	1.390	1.174	1.130	1.300	1.608

Table B.38: Standard deviation of average number of solutions for 3D MF3

R	$\lvert S\rvert$									
	2	4	6	8	10	12	14	16	18	20
2	0.000	0.371	0.557	0.643	0.743	0.965	1.145	1.130	1.160	1.543
4	0.263	0.643	0.928	1.174	1.390	1.486	1.930	1.885	2.174	2.125
6	0.719	1.160	1.762	2.977	2.665	3.074	3.040	3.728	3.538	3.952
8	0.643	1.287	1.640	2.236	1.722	1.974	2.936	3.533	3.714	3.135
10	0.928	1.390	2.484	2.435	2.464	2.236	2.871	2.716	3.189	3.469
12	0.851	2.189	1.894	1.983	2.304	3.572	2.859	3.332	3.216	3.686
14	1.174	1.800	1.983	2.748	3.129	3.464	3.606	4.102	3.051	2.889
16	1.232	2.026	2.810	2.606	2.560	3.368	3.528	4.705	3.824	3.577
18	1.017	2.051	2.282	2.697	3.046	3.957	3.285	3.146	4.676	4.394
20	1.246	2.704	2.889	3.068	3.504	3.695	4.077	3.474	4.426	4.119

Table B.39: Standard deviation of average number of solutions for 4D MF3

R	$\lvert S\rvert$									
	2	4	6	8	10	12	14	16	18	20
2	0.000	0.000	0.263	0.371	0.415	0.455	0.670	0.616	0.743	0.788
4	0.000	0.263	0.491	0.557	0.695	0.719	0.830	0.891	1.203	1.629
6	0.000	0.455	0.455	1.099	1.000	1.203	1.462	1.587	1.921	1.903
8	0.415	0.557	0.910	1.287	1.692	1.800	1.857	2.068	1.712	3.168
10	0.415	0.587	1.160	1.414	1.203	1.702	2.385	2.672	2.710	2.871
12	0.557	0.947	1.218	1.232	2.334	2.560	2.639	3.173	2.349	3.200
14	0.587	0.965	1.067	1.965	2.393	2.742	3.200	3.384	3.146	3.439
16	0.557	1.130	1.848	2.449	3.102	2.895	3.102	4.140	4.299	4.698
18	0.928	1.099	1.597	2.442	2.754	3.280	3.399	3.755	4.792	4.860
20	0.719	1.377	2.017	2.213	2.652	4.247	4.917	3.389	4.094	5.960

Table B.40: Standard deviation of average accuracy for 2D MF3

R	$\lvert S\rvert$									
	2	4	6	8	10	12	14	16	18	20
2	0.403	0.012	0.059	0.066	0.057	0.006	0.006	0.008	0.001	0.005
4	0.023	0.026	0.017	0.010	0.025	0.056	0.044	0.041	0.044	0.106
6	0.018	0.094	0.059	0.077	0.020	0.088	0.048	0.039	0.114	0.152
8	0.138	0.079	0.058	0.069	0.079	0.089	0.166	0.085	0.115	0.124
10	0.069	0.122	0.016	0.090	0.038	0.089	0.130	0.063	0.237	0.122
12	0.140	0.102	0.073	0.139	0.060	0.040	0.096	0.143	0.107	0.217
14	0.144	0.030	0.120	0.142	0.113	0.102	0.126	0.084	0.174	0.113
16	0.102	0.026	0.063	0.082	0.066	0.176	0.136	0.180	0.229	0.184
18	0.202	0.147	0.101	0.060	0.136	0.203	0.175	0.244	0.330	0.325
20	0.107	0.122	0.045	0.083	0.153	0.173	0.216	0.240	0.243	0.281

Table B.41: Standard deviation of average accuracy for 3D MF3

R	$\lvert S\rvert$									
	2	4	6	8	10	12	14	16	18	20
2	0.033	0.116	0.000	0.025	0.009	0.078	0.110	0.055	0.020	0.025
4	0.794	0.169	0.045	0.089	0.008	0.031	0.007	0.086	0.143	0.026
6	0.278	0.093	0.024	0.173	0.094	0.117	0.100	0.035	0.155	0.141
8	0.270	0.232	0.104	0.218	0.032	0.110	0.168	0.070	0.206	0.193
10	0.172	0.078	0.120	0.194	0.073	0.131	0.139	0.128	0.257	0.234
12	0.268	0.365	0.066	0.122	0.143	0.176	0.113	0.207	0.272	0.326
14	0.164	0.081	0.144	0.169	0.054	0.151	0.225	0.176	0.218	0.269
16	0.188	0.131	0.111	0.081	0.237	0.254	0.140	0.211	0.322	0.351
18	0.109	0.048	0.079	0.129	0.197	0.230	0.232	0.274	0.212	0.331
20	0.117	0.145	0.100	0.121	0.237	0.383	0.276	0.371	0.373	0.273

Table B.42: Standard deviation of average accuracy for 4D MF3

R	$\lvert S\rvert$									
	2	4	6	8	10	12	14	16	18	20
2	0.156	0.049	0.275	0.078	0.019	0.232	0.068	0.030	0.049	0.107
4	0.113	0.085	0.072	0.142	0.103	0.164	0.071	0.064	0.152	0.090
6	0.156	0.198	0.168	0.143	0.148	0.088	0.168	0.095	0.030	0.060
8	0.132	0.128	0.126	0.151	0.157	0.065	0.057	0.094	0.127	0.143
10	0.175	0.126	0.072	0.137	0.109	0.037	0.078	0.150	0.046	0.087
12	0.570	0.179	0.171	0.161	0.050	0.040	0.125	0.141	0.099	0.100
14	0.260	0.181	0.071	0.187	0.096	0.031	0.113	0.060	0.220	0.113
16	0.074	0.078	0.137	0.131	0.203	0.107	0.168	0.062	0.071	0.225
18	0.246	0.219	0.055	0.055	0.118	0.107	0.224	0.039	0.048	0.142
20	0.299	0.167	0.194	0.118	0.318	0.288	0.175	0.091	0.131	0.109

Table B.43: Percentage of converged subswarms for 2D MF3

R	$\lvert S\rvert$									
	2	4	6	8	10	12	14	16	18	20
2	86.207	82.243	88.667	86.022	90.045	94.561	91.791	92.517	90.789	93.313
4	87.719	91.534	86.694	91.144	92.690	90.986	90.789	95.619	90.566	89.461
6	84.574	90.102	90.751	90.370	91.283	88.914	88.478	89.787	89.583	85.193
8	84.239	89.199	90.202	91.392	91.608	88.330	88.767	89.765	89.507	87.298
10	87.383	86.520	90.281	86.183	89.602	88.962	89.700	88.636	86.747	83.725
12	80.913	86.337	84.652	86.830	88.079	88.454	88.844	85.855	89.152	86.667
14	85.409	90.374	86.343	87.852	88.445	89.775	88.710	87.621	88.845	87.791
16	84.192	90.649	87.075	87.841	86.653	83.432	87.071	87.016	86.200	85.687
18	85.149	84.977	87.004	86.910	85.600	86.600	83.142	85.797	84.000	84.658
20	86.687	89.051	88.913	88.285	87.475	86.759	84.571	84.423	85.660	85.505

Table B.44: Percentage of converged subswarms for 3D MF3

R	$\lvert S\rvert$									
	2	4	6	8	10	12	14	16	18	20
2	88.333	85.345	87.059	82.511	83.803	81.791	79.787	81.797	85.106	86.160
4	81.356	80.269	83.988	83.645	84.646	85.117	83.133	83.490	82.275	82.171
6	79.545	83.971	84.122	80.978	84.925	81.639	82.491	82.003	81.664	80.769
8	80.531	82.338	84.327	83.536	83.149	82.028	81.379	82.533	80.183	80.300
10	82.971	85.277	85.122	81.172	82.915	82.060	82.767	80.516	79.946	79.374
12	83.792	82.383	84.063	84.130	82.654	80.627	80.152	80.092	79.527	77.868
14	81.744	83.602	82.366	83.438	80.031	81.773	79.895	79.513	80.671	78.242
16	80.374	81.344	80.702	81.411	81.465	79.033	79.060	76.651	78.552	76.304
18	83.117	85.088	82.139	80.923	79.470	80.395	76.045	77.473	76.348	75.234
20	83.915	81.960	80.617	79.631	78.719	79.142	75.574	74.897	74.880	75.849

Table B.45: Percentage of converged subswarms for 4D MF3

R	$\|S\|$									
	2	4	6	8	10	12	14	16	18	20
2	78.333	83.333	75.281	82.773	79.730	79.096	84.878	80.213	79.463	82.616
4	83.193	80.833	80.912	81.741	81.002	82.225	81.556	80.842	78.740	81.211
6	79.325	83.656	80.836	79.731	81.511	79.661	77.727	78.859	80.654	78.349
8	79.237	85.313	81.009	79.535	81.475	80.927	80.483	78.339	79.136	78.860
10	78.451	80.000	81.183	78.943	79.912	79.489	81.148	79.039	79.721	79.398
12	77.684	81.422	81.496	79.970	79.913	79.395	80.557	80.416	79.545	78.787
14	78.624	81.398	80.498	82.503	80.934	79.654	78.667	79.186	79.699	79.249
16	81.702	79.264	80.600	80.961	78.930	78.737	78.868	79.012	77.817	77.392
18	80.190	80.060	78.940	80.571	78.022	78.706	78.395	77.854	78.272	77.485
20	82.092	80.608	79.583	79.252	80.000	77.675	77.277	78.828	76.319	77.814

Table B.46: Average number of solutions found for 2D Rastrigin

R	$\|S\|$									
	2	4	6	8	10	12	14	16	18	20
2	2.000	3.767	5.533	6.400	7.533	9.000	9.200	10.333	11.100	11.833
4	3.633	6.300	8.633	10.033	11.900	12.600	13.333	13.933	14.633	14.667
6	4.933	8.600	11.133	12.667	13.533	14.533	15.100	15.167	15.633	15.867
8	6.633	10.867	12.533	13.867	14.833	15.367	15.633	15.833	15.967	16.167
10	7.533	11.800	13.733	14.900	15.633	15.767	15.900	16.400	16.233	16.167
12	8.533	12.733	14.200	15.267	15.833	16.100	16.400	16.600	16.433	16.300
14	9.633	13.367	15.033	15.300	16.267	16.033	16.200	16.433	16.400	16.367
16	10.233	14.433	15.500	15.933	16.300	16.333	16.700	16.467	16.567	16.967
18	11.000	14.733	15.467	15.933	16.400	16.433	16.400	16.667	16.667	16.667
20	11.800	15.233	15.967	16.400	16.533	16.467	16.533	16.567	16.900	16.800

Table B.47: Average number of solutions found for 3D Rastrigin

R	$\lvert S\rvert$									
	2	4	6	8	10	12	14	16	18	20
2	2.000	3.933	5.900	7.667	9.400	11.333	12.800	14.533	16.333	17.367
4	3.967	7.633	11.300	14.467	17.200	20.100	23.433	26.233	28.200	31.200
6	5.833	10.933	15.800	20.433	24.300	27.967	31.967	34.967	37.400	40.067
8	7.667	14.400	20.433	25.600	29.900	35.133	37.933	42.200	44.767	46.433
10	9.167	17.667	23.900	30.667	35.167	39.867	43.667	47.733	50.033	51.733
12	11.067	20.467	28.400	34.667	39.933	44.967	49.067	51.033	53.800	56.267
14	12.700	23.167	31.233	39.000	44.067	48.467	50.967	54.533	56.300	58.267
16	13.933	25.600	34.233	41.667	47.100	50.233	54.367	57.500	58.667	61.833
18	15.667	27.867	37.700	45.133	49.667	53.500	57.800	59.733	61.867	62.267
20	17.667	30.533	39.900	46.800	52.033	55.833	59.067	61.167	61.333	63.967

Table B.48: Average number of solutions found for 4D Rastrigin

R	$\lvert S\rvert$									
	2	4	6	8	10	12	14	16	18	20
2	2.000	4.000	5.933	7.933	9.967	11.800	13.600	15.567	17.500	19.533
4	3.967	7.900	11.700	15.800	19.133	23.000	26.633	30.733	34.333	37.600
6	5.933	11.733	17.567	22.933	28.300	33.667	39.433	44.100	48.800	54.000
8	7.933	15.567	22.933	29.833	37.400	44.300	50.667	57.700	63.633	69.667
10	9.833	19.233	28.767	37.000	46.133	54.300	61.900	69.100	76.133	83.200
12	11.833	23.100	34.167	43.700	53.367	63.600	72.600	80.167	90.367	95.200
14	13.567	26.733	39.133	50.633	62.200	72.367	83.233	91.533	101.567	107.900
16	15.567	30.267	43.567	57.467	68.933	80.433	92.533	101.233	111.733	119.067
18	17.333	33.600	48.833	63.400	77.233	88.767	100.233	112.100	122.600	131.167
20	19.433	37.067	53.633	68.500	83.067	97.233	109.867	120.000	132.500	142.033

Table B.49: Average accuracy for 2D Rastrigin

R	$\lvert S\rvert$									
	2	4	6	8	10	12	14	16	18	20
2	0.007	0.002	0.010	0.028	0.051	0.024	0.021	0.087	0.050	0.036
4	0.246	0.032	0.068	0.151	0.111	0.099	0.124	0.122	0.203	0.076
6	0.190	0.083	0.246	0.137	0.178	0.083	0.161	0.139	0.230	0.172
8	0.129	0.277	0.144	0.179	0.064	0.187	0.120	0.123	0.211	0.165
10	0.283	0.087	0.133	0.268	0.354	0.124	0.153	0.206	0.200	0.154
12	0.253	0.154	0.112	0.211	0.176	0.191	0.375	0.457	0.216	0.241
14	0.103	0.228	0.207	0.064	0.303	0.206	0.161	0.365	0.290	0.205
16	0.193	0.133	0.396	0.128	0.280	0.184	0.486	0.320	0.411	0.588
18	0.067	0.169	0.103	0.133	0.313	0.394	0.258	0.407	0.390	0.442
20	0.176	0.168	0.291	0.296	0.315	0.450	0.323	0.343	0.552	0.580

Table B.50: Average accuracy for 3D Rastrigin

R	$\lvert S\rvert$									
	2	4	6	8	10	12	14	16	18	20
2	0.981	0.062	0.407	0.152	0.405	0.293	0.193	0.506	0.114	0.527
4	0.237	0.458	0.302	0.792	0.409	0.187	0.343	0.263	0.244	0.243
6	0.387	0.486	0.462	0.389	0.243	0.382	0.245	0.374	0.232	0.473
8	0.390	0.378	0.458	0.191	0.379	0.243	0.196	0.335	0.360	0.477
10	0.963	0.239	0.315	0.365	0.279	0.384	0.314	0.471	0.386	0.360
12	0.289	0.407	0.234	0.332	0.328	0.456	0.393	0.277	0.642	0.431
14	0.337	0.293	0.468	0.301	0.439	0.472	0.416	0.595	0.303	0.518
16	0.511	0.426	0.214	0.475	0.379	0.474	0.608	0.536	0.465	0.594
18	0.399	0.515	0.519	0.362	0.429	0.542	0.640	0.652	0.636	0.555
20	0.437	0.576	0.405	0.600	0.421	0.433	0.702	0.557	0.553	0.690

Table B.51: Average accuracy for 4D Rastrigin

R	$\|S\|$									
	2	4	6	8	10	12	14	16	18	20
2	0.462	0.010	0.901	0.586	0.585	0.448	0.498	0.290	0.450	0.177
4	0.449	1.172	0.390	0.282	0.310	0.428	0.288	0.611	0.240	0.404
6	0.232	0.853	0.513	0.278	0.588	0.355	0.674	0.179	0.575	0.535
8	0.202	0.445	0.775	0.481	0.476	0.434	0.455	0.430	0.382	0.531
10	0.938	0.415	0.502	0.703	0.376	0.618	0.476	0.499	0.236	0.431
12	0.170	0.478	0.493	0.380	0.720	0.322	0.228	0.559	0.459	0.408
14	0.633	0.223	0.521	0.279	0.436	0.445	0.513	0.470	0.333	0.656
16	0.402	0.293	0.348	0.438	0.633	0.503	0.372	0.572	0.527	0.403
18	0.686	0.651	0.479	0.460	0.544	0.578	0.342	0.683	0.388	0.496
20	0.983	0.549	0.435	0.567	0.488	0.492	0.566	0.652	0.501	0.500

Table B.52: Standard deviation of average number of solutions for 2D Rastrigin

R	$\|S\|$									
	2	4	6	8	10	12	14	16	18	20
2	0.000	0.455	0.587	0.871	1.017	1.083	1.160	1.402	1.326	1.450
4	0.491	0.809	1.273	1.067	1.377	1.486	1.130	1.339	1.130	0.928
6	0.788	1.203	1.174	1.426	1.390	1.114	0.965	0.928	1.189	0.830
8	1.034	1.145	1.509	1.365	1.067	0.965	0.809	0.809	0.557	0.809
10	1.145	0.891	0.947	1.246	1.067	0.643	0.766	0.643	0.643	0.557
12	1.145	1.339	1.130	0.983	0.809	0.766	0.871	0.743	0.830	0.557
14	0.965	1.352	0.891	0.965	0.871	0.616	0.766	0.643	0.788	0.670
16	1.174	1.145	1.232	0.743	0.766	0.766	1.000	0.643	0.788	1.000
18	1.145	1.017	0.910	0.695	0.830	0.643	0.830	0.766	0.891	1.034
20	1.474	0.983	1.000	0.695	0.643	0.788	0.643	0.788	0.965	0.851

Table B.53: Standard deviation of average number of solutions for 3D Rastrigin

R	$\lvert S\rvert$									
	2	4	6	8	10	12	14	16	18	20
2	0.000	0.263	0.322	0.670	0.743	0.891	0.851	0.983	1.034	1.326
4	0.186	0.670	0.809	1.203	1.712	1.543	1.531	1.486	1.829	2.282
6	0.491	1.050	1.326	1.438	2.026	2.125	2.158	2.267	2.421	2.913
8	0.557	1.287	1.619	1.640	2.606	1.965	2.665	2.539	2.197	3.363
10	0.851	1.160	1.921	1.956	2.093	2.304	2.785	2.613	2.761	2.464
12	0.910	1.531	1.722	2.125	2.546	2.785	2.519	3.634	3.079	2.133
14	1.034	1.848	2.181	2.421	2.573	2.407	2.526	2.913	2.512	2.378
16	1.174	1.875	2.505	2.633	2.822	2.259	2.342	2.678	2.847	2.312
18	1.000	2.228	2.327	2.533	2.484	2.533	1.956	2.449	3.504	2.034
20	1.218	1.259	2.189	2.723	2.327	2.484	2.349	2.428	2.327	2.109

Table B.54: Standard deviation of average number of solutions for 4D Rastrigin

R	$\lvert S\rvert$									
	2	4	6	8	10	12	14	16	18	20
2	0.000	0.000	0.263	0.263	0.186	0.415	0.587	0.587	0.743	0.587
4	0.186	0.322	0.491	0.415	0.910	0.830	1.130	0.910	1.000	1.390
6	0.263	0.455	0.587	0.871	1.130	1.099	1.438	1.866	2.356	2.477
8	0.263	0.743	0.947	1.474	1.619	1.885	1.712	2.009	2.428	2.723
10	0.415	0.830	1.287	1.597	1.722	1.732	1.956	2.895	3.017	3.285
12	0.415	0.928	1.565	1.752	2.251	2.319	3.343	3.296	4.279	2.918
14	0.871	1.050	1.509	1.752	2.282	2.267	2.560	3.974	3.714	3.113
16	0.830	1.232	1.742	2.034	2.853	2.983	2.841	3.957	3.384	4.518
18	0.809	1.486	1.377	1.983	3.085	3.957	3.533	3.891	5.233	4.613
20	0.830	1.722	1.752	2.691	3.301	3.824	3.424	4.152	3.504	4.004

Table B.55: Standard deviation of average accuracy for 2D Rastrigin

R	$\|S\|$									
	2	4	6	8	10	12	14	16	18	20
2	0.000	0.000	0.001	0.003	0.034	0.008	0.001	0.030	0.001	0.004
4	0.113	0.002	0.022	0.002	0.014	0.018	0.014	0.031	0.053	0.003
6	0.031	0.029	0.032	0.036	0.046	0.026	0.051	0.056	0.051	0.048
8	0.030	0.019	0.019	0.008	0.006	0.025	0.026	0.027	0.072	0.047
10	0.069	0.039	0.034	0.046	0.048	0.034	0.050	0.032	0.048	0.051
12	0.077	0.023	0.020	0.046	0.054	0.056	0.124	0.097	0.051	0.053
14	0.041	0.087	0.040	0.014	0.038	0.068	0.026	0.083	0.090	0.034
16	0.043	0.023	0.140	0.042	0.068	0.036	0.060	0.091	0.130	0.158
18	0.019	0.039	0.028	0.025	0.088	0.104	0.077	0.139	0.081	0.091
20	0.055	0.049	0.081	0.044	0.076	0.089	0.101	0.105	0.142	0.176

Table B.56: Standard deviation of average accuracy for 3D Rastrigin

R	$\|S\|$									
	2	4	6	8	10	12	14	16	18	20
2	0.012	0.021	0.005	0.008	0.047	0.024	0.005	0.005	0.014	0.035
4	0.036	0.103	0.074	0.156	0.021	0.010	0.008	0.025	0.021	0.063
6	0.189	0.036	0.075	0.018	0.033	0.049	0.022	0.066	0.015	0.044
8	0.100	0.043	0.068	0.009	0.020	0.027	0.038	0.045	0.037	0.086
10	0.259	0.023	0.055	0.108	0.032	0.096	0.032	0.069	0.031	0.055
12	0.084	0.029	0.030	0.037	0.055	0.063	0.054	0.067	0.177	0.038
14	0.059	0.076	0.079	0.045	0.091	0.061	0.125	0.099	0.044	0.120
16	0.187	0.030	0.058	0.075	0.086	0.108	0.108	0.062	0.120	0.086
18	0.174	0.060	0.117	0.065	0.049	0.136	0.078	0.102	0.098	0.125
20	0.164	0.093	0.034	0.063	0.082	0.112	0.140	0.116	0.106	0.138

Table B.57: Standard deviation of average accuracy for 4D Rastrigin

R	$\lvert S\rvert$									
	2	4	6	8	10	12	14	16	18	20
2	0.006	0.000	0.042	0.101	0.055	0.031	0.155	0.045	0.020	0.069
4	0.260	0.065	0.033	0.017	0.025	0.053	0.037	0.129	0.113	0.107
6	0.082	0.071	0.090	0.057	0.076	0.080	0.213	0.044	0.036	0.056
8	0.085	0.086	0.253	0.094	0.179	0.104	0.043	0.065	0.025	0.069
10	0.396	0.041	0.131	0.105	0.044	0.043	0.075	0.107	0.022	0.068
12	0.092	0.033	0.112	0.080	0.136	0.059	0.026	0.047	0.065	0.079
14	0.276	0.030	0.170	0.048	0.052	0.070	0.119	0.061	0.036	0.070
16	0.072	0.055	0.033	0.049	0.084	0.019	0.023	0.047	0.119	0.054
18	0.239	0.122	0.131	0.056	0.057	0.114	0.040	0.029	0.042	0.025
20	0.613	0.046	0.036	0.049	0.097	0.043	0.077	0.089	0.041	0.068

Table B.58: Percentage of converged subswarms for 2D Rastrigin

R	$\lvert S\rvert$									
	2	4	6	8	10	12	14	16	18	20
2	96.667	99.115	99.398	97.917	97.345	98.148	98.188	99.032	98.198	98.310
4	91.743	97.354	98.842	95.681	96.919	97.884	98.250	97.847	98.405	98.182
6	92.568	97.674	95.808	97.632	97.537	97.248	97.792	97.802	97.228	97.479
8	94.472	96.626	96.809	97.596	98.202	96.746	97.868	97.895	97.077	97.113
10	96.018	97.740	98.058	95.526	95.736	98.097	97.484	96.341	96.509	96.701
12	95.313	97.906	97.887	97.162	98.105	96.894	95.732	95.181	96.349	96.115
14	95.502	97.007	97.783	96.732	95.287	97.505	96.502	96.146	95.732	96.741
16	97.720	96.074	96.774	97.280	95.706	97.143	94.810	96.356	95.171	92.927
18	96.667	97.285	98.060	96.234	95.935	96.146	96.138	95.400	94.800	94.800
20	96.328	97.374	96.242	95.528	95.363	95.344	95.363	95.372	94.083	93.849

Table B.59: Percentage of converged subswarms for 3D Rastrigin

R	$\|S\|$									
	2	4	6	8	10	12	14	16	18	20
2	98.333	97.458	97.175	98.261	96.099	97.059	98.438	96.789	97.959	97.697
4	94.958	96.943	98.230	95.392	97.093	98.010	97.155	97.078	97.754	97.650
6	97.143	96.341	95.992	98.206	97.394	96.901	96.663	97.426	97.683	97.171
8	96.087	96.528	96.574	96.745	95.875	97.723	98.243	96.919	97.096	96.770
10	95.273	97.170	97.768	97.065	96.872	96.656	96.794	96.020	96.402	97.487
12	96.386	97.068	97.300	96.923	96.912	96.664	96.943	97.322	95.663	96.090
14	96.850	96.115	95.624	97.009	97.050	96.217	97.057	96.333	96.921	96.053
16	95.694	96.615	96.981	96.480	96.886	96.549	96.383	96.290	96.364	95.364
18	96.596	96.292	96.552	96.824	96.443	95.950	95.905	95.313	95.097	95.396
20	95.849	95.742	96.491	96.510	96.605	96.716	95.880	95.749	95.870	95.362

Table B.60: Percentage of converged subswarms for 4D Rastrigin

R	$\|S\|$									
	2	4	6	8	10	12	14	16	18	20
2	98.333	99.167	96.629	96.639	98.328	97.740	97.794	98.073	97.143	98.464
4	97.479	96.203	97.436	97.046	97.909	98.116	98.248	98.156	98.252	97.961
6	96.629	95.739	96.964	98.401	97.409	97.921	97.210	98.110	97.883	97.716
8	97.899	97.645	97.093	97.318	97.861	98.044	97.697	97.978	98.062	97.512
10	96.271	98.614	96.987	97.117	98.049	97.483	97.307	97.926	98.643	97.917
12	98.310	97.258	97.561	97.788	97.064	97.799	98.209	97.505	97.565	98.004
14	97.052	98.254	96.934	98.025	97.428	97.651	97.958	97.706	97.932	97.189
16	97.645	97.797	97.781	98.492	97.147	97.638	97.839	97.498	97.584	97.844
18	96.923	96.032	97.338	97.161	96.849	97.334	97.905	97.324	97.607	97.459
20	96.913	97.392	97.203	97.129	97.833	97.360	97.633	97.306	97.484	97.630

Appendix C

List of Symbols

This appendix summarises common symbols used throughout this thesis and their definition.

Table C.1: List of symbols and their definition.

Symbol	Definition
e	Simulation index.
E	Maximum number of simulations.
f	A function.
f'	The derivative of f.
F	Original fitness function, or unmodified fitness function.
F_{sh}	Sharing fitness function.
i	Particle or individual index.
g	Index of the current generation for the genetic algorithm.
m	Number of equations in a system of equations.
M	Modified fitness function.
n	Number of dimensions in a search space.
N	Number of objectives in a multi-objective optimisation problem.
$\mathbf{p}$	Local minimum of a function.

Continued on next page

Symbol	**Definition**
p_m	Mutation rate.
P	Population size.
q	Number of optima in a search space.
R	Number of executions or runs in a sequential niching technique.
s	A subswarm.
S	The main swarm.
$\|S\|$	Swarm size.
t	Index of the current time.
$\mathbf{v}_i$	Velocity of particle i.
$\widehat{\mathbf{y}}$	Best particle's position in a swarm, or best individual's position in the population.
$\mathbf{y}_i$	Personal best position of particle i.
$\widehat{\mathbf{y}}_i$	Neighbourhood best particle's position.
$\widehat{\mathbf{y}}_j$	Global best particle's personal best position.
$\widehat{\mathbf{y}}_{t,k}$	Best particle's position in subswarm k at time t.
$\widehat{\mathbf{y}}_t$ or $\widehat{\mathbf{y}}(t)$	Global best particle's position at time t.
$\mathbf{x}$	Point in the search space.
$\mathbf{x}_i$	Position of particle or individual i.
$\mathbf{X}$	Set of subswarms created for a single simulation of the derating NichePSO.
κ	Index of a equation in a system of equations.
Γ	Redundancy factor for iteration based niching techniques.

Bibliography

[1] P. J. Angeline and J. B. Pollack. Competitive environments evolve better solutions for complex tasks. In *Proceedings of the 5th International Conference on Genetic Algorithms*, pages 264–270, 1993.

[2] P. J. Angeline and J. B. Pollack. Coevolving high-level representations. In Christopher G. Langton, editor, *Artificial Life III*, volume XVII, pages 55–71, Santa Fe, New Mexico, 1994. Addison-Wesley.

[3] G. Arfken. The method of steepest descents. In *Mathematical Methods for Physicists, 3rd edition*, pages 428–436. Academic Press, 1985.

[4] O. Arratia, A. Milani, M. E. Sansaturio, and G. Tommei. Multiple solutions for asteroid orbits: computational procedure and applications. In *Astronomy & Astrophysics manuscript no.*, July 2004.

[5] T. Bäck. Evolution strategies: An alternative evolutionary algorithm. In J. M. Alliot, E. Lutton, E. Ronald, M. Schoenauer, and D. Snyers, editors, *Artificial Evolution*, number 1063 in LNCS, pages 3–20. Springer-Verlag, 1995.

[6] T. Bäck. *Evolutionary Algorithms in Theory and Practice.* Oxford University Press, New York, 1996.

[7] P. Bak. *How Nature Works.* Springer-Verlag, 1999.

[8] D. Beasley, D. R. Bull, and R. R. Martin. A sequential niche technique for multimodal function optimization. In *Evolutionary Computation*, volume 1, pages 101–125, 1993.

[9] A. Ben-Tal. Characterization of pareto and lexicographic optimal solutions. In Fandel and Gal, editors, *Multiple Criteria Decision Making, Theory and Applications, Lecture Notes in Economics and Mathematical Systems*, volume 77, pages 1–11. Springer-Verlag, 1980.

[10] M. Bessaou, A. Pétrowski, and P. Siarry. Island model cooperating with speciation for multimodal optimization. In Hans-Paul Schwefel, Marc Schoenauer, Kalyanmoy Deb, Günter Rudolph, Xin Yao, Evelyne Lutton, and Juan Julian Merelo, editors, *Proceedings of the 6th International Conference on Parallel Problem Solving from Nature*, Paris, France, 2000. Springer-Verlag.

[11] T. M. Blackwell. Swarms in dynamic environments. In *Lecture Notes in Computer Science, Proceedings of the Genetic and Evolutionary Computation Conference*, volume 2723, pages 1–12, 2003.

[12] T. M. Blackwell and P. J. Bentley. Dynamic search with charged swarms. In *Proceedings of the Genetic and Evolutionary Computation Conference*, pages 19–26, 2002.

[13] T. Brenner. *Computational Techniques for Modelling Learning in Economics*. Springer, 1999.

[14] R. Brits. *Niching Strategies for Particle Swarm Optimization, MSc Thesis*. University of Pretoria, 2002.

[15] R. Brits, A. P. Engelbrecht, and F. van den Bergh. A niching particle swarm optimizer. In *Proceedings of the 4th Asia-Pacific Conference on Simulated Evolution and Learning*, pages 692–696, Orchid Country Club, Singapore, 2002.

[16] R. Brits, A. P. Engelbrecht, and F. van den Bergh. Scalability of Niche PSO. In *IEEE Swarm Intelligence Symposium*, pages 228–234, Indianapolis, 2003.

[17] M. Clerc and J. Kennedy. The particle swarm - explosion, stability, and convergence in a multidimensional complex space. In *IEEE Transactions on Evolutionary Computation*, volume 6, pages 58–73, 2002.

[18] M. Clerc and J. Kennedy. The particle swarm-explosion, stability, and convergence in a multidimensional complex space. In *IEEE Transactions on Evolutionary Computation*, volume 6, pages 58–73, 2002.

[19] C. Darwin. *On the Origin of Species*. John Murray, London, 1859.

[20] L. Davis. *Hybridization and Numerical Representation, The Handbook of Genetic Algorithms*. Van Nostrand Reinhold, 1991.

[21] K. Deb. *Genetic Algorithms in Multimodal Function Optimization, Masters thesis, TCGA Report No. 89002*. The University of Alabama, Department of Engineering Mechanics, 1989.

[22] K. Deb and D. E. Goldberg. Natural frequency calculation using genetic algorithms. In Sathya V. Hanagud, Manohar P. Kamat, and Charles E. Ueng, editors, *Proceedings of the 16th Southeastern Conference on Theoretical and Applied Mechanics*, pages 94–101, Atlanta, GA, 1990. The College of Engineering, Georgia Institute of Technology.

[23] K. Deb, D. E. Goldberg, and J. Horn. Massive multimodality, deception, and genetic algorithms. In R. Männer and B. Manderick, editors, *Parallel Problem Solving From Nature*, pages 37–47, Amsterdam, 1992. Elsevier Science Publishers.

[24] R. C. Eberhart and J. Kennedy. *Swarm Intelligence*. Morgan Kaufmann, 2001.

[25] A. P. Engelbrecht. *Computational Intelligence: An Introduction*. Wiley and Sons, 2002.

[26] A. P. Engelbrecht, B. S. Masiye, and G. Pampara. Niching ability of basic particle swarm optimisation algorithms. In *Proceedings of the IEEE Swarm Intelligence Symposium*, Pasadena, California, 2005.

[27] A. P. Engelbrecht and F. van den Bergh. *A Study of Particle Swarm Optimisation Particle Trajectories*. Information Sciences, 2005.

[28] L. J. Fogel, A. J. Owens, and M. J. Walsh. *Artificial Intelligence through Simulated Evolution.* John Wiley & Sons, New York, 1969.

[29] C. M. Fonseca and P. J. Fleming. Genetic algorithms for multi-objective optimization: Formulation, discussion and generalization. In Forrest, editor, *Proceedings of the 5th International Conference on Genetic Algorithms*, pages 416–423, 1993.

[30] C. M. Fonseca and P. J. Fleming. An overview of evolutionary algorithms in multiobjective optimization. *Evolutionary Computation*, 3(1):1–16, 1995.

[31] M. P. Fourman. Compaction of symbolic layout using genetic algorithm. In In J.J. Grefenstette, editor, *Proceedings of the 1st International Conference on Genetic Algorithms*, pages 141–153. Lawrence Erlbaum Associates, 1985.

[32] J. Gan and K. Warwick. A variable radius niche technique for speciation in genetic algorithms. In *Proceedings of the Genetic and Evolutionary Computation Conference*, pages 96–103. Morgan-Kaufmann, 2000.

[33] D. E. Goldberg. *Genetic Algorithms in Search, Optimization and Machine Learning.* Addison-Wesley, 1989.

[34] D. E. Goldberg. Making genetic algorithms fly: a lesson from the wright brothers. In *Advanced Technology For Developers*, volume 2, pages 1–8, February 1993.

[35] D. E. Goldberg. Using time efficiently: Genetic-evolutionary algorithms and the continuation problem. In Wolfgang Banzhaf, Jason Daida, Agoston E. Eiben, Max H. Garzon, Vasant Honavar, Mark Jakiela, and Robert E. Smith, editors, *Proceedings of the Genetic and Evolutionary Computation Conference*, volume 1, pages 212–219, Orlando, Florida, USA, 1999. Morgan Kaufmann.

[36] D. E. Goldberg and J. Richardson. Genetic algorithm with sharing for multimodal function optimization. In *Proceedings of the 2nd International Conference on Genetic Algorithms*, pages 41–49, 1997.

[37] D. E. Goldberg and L. Wang. Adaptive niching via coevolutionary sharing. In D. Quagliarella, J. Périaux, C. Poloni, and G. Winter, editors, *Genetic Algorithms and Evolution Strategy in Engineering and Computer Science*, pages 21–38, Chichester, August 1998. John Wiley and Sons.

[38] V. S. Gordon and D. Whitley. Serial and parallel genetic algorithms as function optimizers. In Stephanie Forrest, editor, *Proceedings of the 5th International Conference on Genetic Algorithms*, pages 177–183, San Mateo, 1993. Morgan Kaufman.

[39] P. B. Grosso. *Computer simulations of genetic adaptation: Parallel subcomponent interaction in a multilocus model, PhD Thesis*. University of Michigan, Ann Arbor, 1985.

[40] P. Hajela and C. Y. Lin. Genetic search strategies in multicriterion optimal design. In *Structural Optimization*, volume 4, pages 4:99–107, 1992.

[41] G. R. Harik. Finding multimodal solutions using restricted tournament selection. In Larry Eshelman, editor, *Proceedings of the 6th International Conference on Genetic Algorithms*, pages 24–31, San Francisco, 1995. Morgan Kaufmann.

[42] J. H. Holland. *Adaptation in Natural and Artificial Systems*. Ann Arbor, MI: University of Michigan Press, 1975.

[43] J. Horn and N. Nafpliotis. Multiobjective Optimization using the Niched Pareto Genetic Algorithm. In *IlliGAl Report 93005*, Urbana, Illinois, USA, 1993.

[44] W. Jakob, M. Gorges-Schleuter, and C. Blume. Application of genetic algorithms to task planning and learning. In R. Manner and B. Manderick, editors, *Parallel Problem Solving from Nature*, pages 291–300, Amsterdam, North-Holland., 1992.

[45] C. Z. Janikow and Z. Michalewicz. An experimental comparison of binary and floating point representations in genetic algorithms. In L. B. Booker R. K. Belew, editor, *Proceedings of the 4th International Conference in Genetic Algorithms*, pages 31–36. Morgan Kauffman, 1991.

[46] Y. Jin, M. Olhofer, and B. Sendhoff. Dynamic Weighted Aggregation for Evolutionary Multi-Objective Optimization: Why Does It Work and How? In Lee Spector, Erik D. Goodman, Annie Wu, W. B. Langdon, Hans-Michael Voigt, Mitsuo Gen, Sandip Sen, Marco Dorigo, Shahram Pezeshk, Max H. Garzon, and Edmund Burke, editors, *Proceedings of the Genetic and Evolutionary Computation Conference*, pages 1042–1049, San Francisco, California, 2001. Morgan Kaufmann Publishers.

[47] L. B. W. Jolley. *Summation of series*. Dover, New York, 1961.

[48] K. A. De Jong. *An Analysis of the Behaviour of a Class of Genetic Adaptive Systems, PhD Thesis*. University of Michigan, 1975.

[49] J. Kennedy. The behaviour of particles. In D. Waagen N. Saravanan, editor, *Proceedings of 7th International Conference on Evolutionary Programming*, pages 581–589, 1998.

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

[51] A. Konar. *Computational Intelligence : Principles, Techniques and Applications*. Springer, 2005.

[52] J. R. Koza. Genetic programming: A paradigm for genetically breeding populations of computer programs to solve problems. In *Technical Report STAN-CS-90-1314*. Margaret Jacks Hall, 1990.

[53] F. Kurusawe. A variant of evolution strategies for vector optimization. In H. P. Schwefel and R. Manner, editors, *Parallel Problem Solving from Nature*, pages 193–197, Berlin, Germany, 1990. Springer-Verlag.

[54] Y. P. Lin and S. M. Phoong. Perfect discrete multitone modulation with optimal transceivers. In *IEEE Transactions on Signal Processing*, volume 48, pages 1702–1711, 2000.

[55] Y. Liu and K. M. Passino. *Swarm Intelligence: Literature Overview*. The Ohio State University, 2000.

[56] H. Lodish, A. Berk, S. L. Zipursky, P. Matsudaira, D. Baltimore, and J. E. Darnell. *Molecular Cell Biology*. W. H. Freeman and Company, 41 Madison Avenue, New York, New York 10010, 1995.

[57] M. Løvbjerg and T. Krink. Extending particle swarm optimizers with self organised criticality. In *In Proceedings of the IEEE World Congress on Evolutionary Computation*, pages 1588–1593, Honolulu, Hawaii, May 2002.

[58] M. Løvbjerg, T. K. Rasmussen, and T. Krink. Hybrid particle swarm optimizer with breeding and subpopulations. In *Proceedings of the Genetic and Evolutionary Computation Conference*, volume 1, pages 469–476, 2001.

[59] F. Luna and A. Perrone, editors. *Agent-Based Methods in Economics and Finance: Simulations in Swarm*. Springer, 2002.

[60] V. Maniezzo M. Dorigo and A. Colorni. The Ant System: Optimization by a colony of cooperating agents. In *IEEE Transactions on Systems, Man, and Cybernetics Part B: Cybernetics*, volume 26, pages 29–41, February 1996.

[61] S. W. Mahfoud. Crowding and preselection revisited. In Reinhard Männer and Bernard Manderick, editors, *Parallel Problem Solving From Nature*, volume 2, pages 27–36. North-Holland, 1992.

[62] S. W. Mahfoud. Simple analytical models of genetic algorithms for multimodal function optimisation. In *Genetic Algorithm Lab*, Department of General Engineering, 117 Transportation Building, 104 South Mathews Avenue, Urbana, IL 61801-2996, 1993.

[63] S. W. Mahfoud. A comparison of parallel and sequential niching methods. In Larry Eshelman, editor, *Proceedings of the 6th International Conference on Genetic Algorithms*, pages 136–143, San Francisco, 1995. Morgan Kaufmann.

[64] S. W. Mahfoud. *Niching methods for genetic algorithms, PhD Thesis*. The University of Illinois, Urbana, Illinois, USA, 1995.

[65] J. H. Mathews and K. D. Fink. *Numerical Methods Using MATLAB, Third Edition*. Prentice Hall, Upper Saddle River, NJ 07458, 1999.

[66] B. L. Miller and Michael J. Shaw. Genetic algorithms with dynamic niche sharing for multimodal function optimization. In *International Conference on Evolutionary Computation*, pages 786–791, 1996.

[67] U. Nehmzow, T. Smithers, and J. Hallam. Location recognition in a mobile robot using self-organising feature maps. In G. Schmidt, editor, *Proceedings of the International Workshop on Information Processing in Autonomous Mobile Robots*, pages 267–277, Berlin, Heidelberg, 1991. Springer.

[68] J. A. Nelder and R. Mead. A simplex method for function minimization. In *The Computer Journal*, volume 7, pages 308–313, 1965.

[69] D. Parrot and X. Li. A particle swarm model for tracking multiple peaks in a dynamic environment using speciation. In *Proceedings of the 2004 IEEE Congress on Evolutionary Computation*, volume 6, pages 98–103, Portland, Oregon, June 2004.

[70] K. E. Parsopoulos, V. P. Plagianakos, G. D. Magoulos, and M. N. Vrahatis. Stretching technique for obtaining global minimizers through particle swarm optimization. In *Proceedings of the Particle Swarm Optimization Workshop*, pages 22–29, Indianapolis, 2001.

[71] K. E. Parsopoulos and M. Vrahatis. Recent approaches to global optimization problems through particle swarm optimization. In *Natural Computing*, volume 1, pages 235–306, 2002.

[72] K. E. Parsopoulos and M. N. Vrahatis. On the computation of all global minimizers through particle swarm optimization. In *IEEE Transactions on Evolutionary Computation*, volume 8, pages 211–224, 2004.

[73] K. E. Parsopoulos and M. N. Vrahitis. Computing periodic orbits of nondifferentiable/discontinuous mappings through particle swarm optimization. In *IEEE 2003 Swarm Intelligence Symposium*, pages 34–41, Indianapolis, USA, 2003.

[74] K. Price and R. Storn. Differential evolution. In *Dr. Dobb's Journal*, pages 18–24, 78, April 1997.

[75] Cedeño W. Vemuri V. R. and Slezak T. Multi-niche crowding in genetic algorithms and its application to the assembly of DNA restriction-fragments. In *Evolutionary Computation*, volume 2, pages 312–345, 1995.

[76] R. G. Reynolds. Cultural algorithms: Modeling of how cultures learn to solve problems. In *Third Annual Conference on Evolutionary Programming*, pages 131–139, River Edge, 1994. World Scientific.

[77] R. G. Reynolds and B. Peng. Cultural algorithms: Modeling of how cultures learn to solve problems. In *Proceedings of the 16th IEEE International Conference on Tools with Artificial Intelligence*, pages 166–172, 2004.

[78] T. Richardson, M.R. Palmer, G. Liepins, and M. Hilliard. Some guidelines for genetic algorithms with penalty functions. In J. D. Schaffer, editor, *Proceedings of the 3rd International Conference on Genetic Algorithms*, pages 191–197, George Mason University, 1989. Morgan Kaufman Publishers.

[79] R. Rymon. *Diagnostic Reasoning and Planning in Exploratory-Corrective Domains, PhD Thesis*. The University of Pennsylvania, 2005.

[80] P. Salil. *Fingerprint Classification and Matching using a Filterbank, PhD Thesis*. Michigan State University, 2001.

[81] D. Schaffer. Multiple objective optimization with vector evaluated genetic algorithms. In J. J. Grefenstette, editor, *Proceedings of the 1st International Conference on Genetic Algorithms*, pages 93–100. Lawrence Erlbaum Associates, 1985.

[82] I. L. Schoeman and A. P. Engelbrecht. Using vector operations to identify niches for particle swarm optimization. In *Proceedings of the Conference on Cybernetics and Intelligent Systems*, 2004.

[83] I. L. Schoeman and A. P. Engelbrecht. A parallel vector-based particle swarm optimiser. In *International Conference on Neural Networks and Genetic Algorithms*, Portugal, 2005.

[84] R. Setiono. Extracting rules from pruned networks for breast cancer diagnosis. In *Artificial Intelligence in Medicine*, volume 8, pages 37–51, 1996.

[85] S. Sevenster and A. P. Engelbrecht. A genetic algorithm for routing in telecommunication networks. In *IMACS Multiconference on Computational Engineering in Systems Applications, Symposium on Control, Optimisation and Supervision*, volume 2, pages 1106–1111, 1996.

[86] Y. Shahar, S. Miksch, and P. Johnson. The asgaard project: A task-specific framework for the application and critiquing of time-oriented clinical guidelines. In *Artificial Intelligence in Medicine*, volume 14, pages 29–51, 1998.

[87] R. Sikora and M. J. Shaw. A double-layered learning approach to acquiring rules for classification: Integrating genetic algorithms with similarity-based learning. In *ORSA Journal on Computing*, volume 6, pages 174–187, 1994.

[88] R. E. Smith, S. Forrest, and A. S. Perelson. Searching for diverse, cooperative populations with genetic algorithms. In *Evolutionary Computation*, volume 1, pages 127–149, The University of Alabama, Department of Engineering Mechanics, 1993.

[89] W. M. Spears, K. A. De Jong, T. Bäck, D. B. Fogel, and H. de Garis. An overview of evolutionary computation. In *European Conference on Machine Learning*, volume 667, pages 442–459, 1993.

[90] N. Srinivas and K. Deb. Multiobjective function optimization using nondominated sorting genetic algorithms. In *Evolutionary Computation*, volume 3, pages 221–248, 1995.

[91] I. Stewart. *Does God Play Dice?* Penguin, London, 1989.

[92] I. C. Trelea. The particle swarm optimization algorithm: Convergence analysis and parameter selection. In *Information Processing Letters*, volume 85, pages 317–325, 2003.

[93] R. K. Ursem. Multinational GAs: Multimodal optimization techniques in dynamic environments. In Darrell Whitley, David Goldberg, Erick Cantu-Paz, Lee Spector, Ian Parmee, and Hans-Georg Beyer, editors, *Proceedings of the Genetic and Evolutionary Computation Conference*, pages 19–26, Las Vegas, Nevada, USA, 2000. Morgan Kaufmann.

[94] F. van den Bergh. *An Analysis of Particle Swarm Optimizers, PhD Thesis*. University of Pretoria, 2002.

[95] F. van den Bergh and A. P. Engelbrecht. A new locally convergent particle swarm optimizer. In *IEEE Conference on Systems, Man, and Cybernetics*, volume 3, page 6, October 2002.

[96] E. W. Weisstein. http://mathworld.wolfram.com/simplex.html, April 2006.

[97] D. Whitley. A genetic algorithm tutorial. In *Statistics and Computing*, volume 4, pages 65–85, 1993.