Niching in Particle Swarm Optimization

by

Isabella Lodewina Schoeman

Submitted in partial fulfillment of the requirements for the degree
Philosophiae Doctor
in the Faculty of Engineering, Built Environment and Information Technology
University of Pretoria, Pretoria

June 2010

Niching in Particle Swarm Optimization

by

Isabella Lodewina Schoeman
E-mail: lona.schoeman@gmail.com

Abstract

Optimization forms an intrinsic part of the design and implementation of modern systems, such as industrial systems, communication networks, and the configuration of electric or electronic components. Population-based single-solution optimization algorithms such as Particle Swarm Optimization (PSO) have been shown to perform well when a number of optimal or suboptimal solutions exist. However, some problems require algorithms that locate all or most of these optimal and suboptimal solutions. Such algorithms are known as niching or speciation algorithms.

Several techniques have been proposed to extend the PSO paradigm so that multiple optima can be located and maintained within a convoluted search space. A significant number of these implementations are subswarm-based, that is, portions of the swarm are optimized separately. Niches are formed to contain these subswarms, a process that often requires user-specified parameters, as the sizes and placing of the niches are unknown. This thesis presents a new niching technique that uses the vector dot product of the social and cognitive direction vectors to determine niche boundaries. Thus, a separate niche radius is calculated for each niche, a process that requires minimal knowledge of the search space. This strategy differs from other techniques using niche radii where a niche radius is either required to be set in advance, or calculated from the distances between particles.

The development of the algorithm is traced and tested extensively using synthetic benchmark functions. Empirical results are reported using a variety of settings. An analysis of the algorithm is presented as well as a scalability study. The performance of the algorithm is also compared to that of two other well-known PSO niching algorithms. To conclude, the vector-based PSO is extended to locate and track multiple optima in dynamic environments.

Keywords: Niching, speciation, particle swarm optimization, dynamic.

Supervisor : Prof. A. P. Engelbrecht
Department : Department of Computer Science
Degree : Philosophiae Doctor

Niching in Particle Swarm Optimization

by

Isabella Lodewina Schoeman
E-mail: lona.schoeman@gmail.com

Opsomming

Optimering vorm 'n intrinsieke deel van die ontwerp en implementering van moderne stelsels soos industriële stelsels, kommunikasienetwerke en die konfigurasie van elektriese of elektroniese komponente. Populasiegebaseerde, enkel-oplossing optimeringsalgoritmes soos Partikel Swerm Optimering (PSO) het reeds getoon dat dit goeie resultate lewer indien 'n aantal optimale of suboptimale oplossings bestaan. Sommige probleme vereis egter algoritmes wat al hierdie optimale en suboptimale oplossings, of die meeste daarvan, opspoor. Sulke algoritmes staan bekend as algoritmes vir nisvorming of spesiasie.

Verskeie tegnieke is voorgestel om die PSO paradigma uit te brei sodat veelvuldige optima opgespoor en onderhou kan word in 'n soekruimte met talle hoogtepunte of pieke. 'n Betekenisvolle aantal van sulke implementasies is subswerm-gebaseerd, wat beteken optimering van gedeeltes van die swerm vind afsonderlik plaas. Nisse word gevorm om hierdie subswerms in te sluit, 'n proses wat dikwels gebruiker-gedefinieerde parameters benodig, aangesien die groottes en posisionering van die nisse nie bekend is nie. Hierdie tesis stel 'n nuwe tegniek vir nisvorming voor wat die vektor dotproduk van die sosiale en kognitiewe rigtingvektore gebruik om nisgrense te bepaal. Sodoende word 'n afsonderlike nisradius vir elke nis bereken, 'n proses wat minimale kennis van die soekruimte vereis. Hierdie strategie verskil van ander tegnieke wat nisradiusse gebruik waar 'n nisradius òf vooraf gestel word, of van die afstande tussen partikels bereken word.

Die ontwikkeling van die algoritme word nagespeur en uitvoerig getoets deur van kunsmatige toetsfunksies gebruik te maak. Empiriese resultate met 'n verskeidenheid stellings word gerapporteer. 'n Ontleding van die algoritme word aangebied sowel as 'n skalingstudie. Die prestasie van die algoritme word ook vergelyk met dié van twee ander bekende PSO nisvormingalgoritmes. Ten slotte word die vektor-gebaseerde PSO uitgebrei om veelvuldige optima in dinamiese omgewings op te spoor en te volg.

Acknowledgements

I would like to sincerely thank the following people who were instrumental in the completion of this dissertation:

- Professor Andries Engelbrecht for his enthusiasm, inspiration and assistance during the development and completion of this thesis;
- My family, friends and colleagues for their patience and support.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Many actions occurring in nature and human existence incorporate some form of optimization. Species survive by continually optimizing parameters describing their environment. Various strategies are employed to ensure the survival of the species, for example, some animals aggregate in flocks, herds and schools. Such structures serve as a defense against predators, are instrumental in locating food sources, and ensure accurate navigation on migration. To survive, individuals in a flock, herd or school are not only dependent on their own devices, but can also rely on the collective knowledge of the entire population.

Particle swarm optimization (PSO) is a population-based optimization strategy inspired by the behaviour of bird flocks that wheel and swoop in unison. Personal as well as social memory are employed to guide the movement of particles, collectively referred to as a swarm, in order to converge on a better position. PSO has been proved to be an effective, efficient and robust optimization method for solving a wide array of single-solution problems including neural network training [96] [98] and function minimization [19] [28] [48] [49]. Furthermore, the algorithm is simple to implement, and requires no gradient information which is problem-dependent and may be difficult to calculate. PSO is particularly useful when the overall optimal solution needs to be located in a problem space containing many sub-optimal solutions. The entire search space is explored, thus preventing the swarm from settling on a suboptimal solution.

While the standard PSO locates the overall best position in the search space, many problems require the location of sub-optima as well. The quality of such solutions often differ very slightly from one another or may even be similar. The user then has a choice between multiple, equally acceptable solutions in a single search space. Population-based algorithms designed

to locate multiple optima, also glean their terminology from nature. In natural ecosystems, animals survive in different ways, and different species evolve to fill each role, referred to as an ecological niche. In PSO, a species is represented by a subswarm. Algorithms designed to locate multiple optima are referred to as *niching* or *speciation* algorithms.

Originally, genetic algorithms were adapted for niching [4] [40] [56]. Since the inception of the PSO paradigm, attempts have been directed towards the development of PSO niching algorithms, and modifications to the standard PSO were put in place to facilitate niching. To this end, various strategies have been applied, for example, objective function stretching [72] where the function landscape is modified each time a position is discovered where the function has a minimum value, the species-based PSO [57] that uses a problem-dependent niche radius parameter set in advance, nBest [12] that redefines the fitness function for multiple solutions, and NichePSO [13] [14] that has a high success rate, but requires finely-tuned parameters. A number of other algorithms yielding good results have been developed [5] [41] [58], but often these algorithms are computationally complex. The need for a simple, elegant and efficient niching algorithm with minimal user-specified parameters and low complexity that would be robust enough to be used in a large variety of situations, was identified. In addition, the idea to exploit the basic characteristics of the PSO paradigm to facilitate niching, was ultimately appealing. Thus, the purpose of this thesis is the development of a niching method encompassing these characteristics as far as possible.

This thesis investigates existing niching strategies that have been developed using the particle swarm paradigm. A novel niching algorithm requiring minimal knowledge of the search space, is developed, tested, and compared with existing algorithms. The algorithm is extended to incorporate dynamic optimization problems where multiple optima are tracked in changing environments.

1.1 Motivation

Single-solution particle swarm optimizers are relatively easy to implement, and a large number of PSO variations have been developed for various problem types. However, since the number of optima in the search space, as well as the shapes and sizes of the peaks forming the function landscape are usually unknown, PSO algorithms that locate multiple optima are much more complicated. Such multimodal PSO algorithms often require prior knowledge of the search space to produce significant results. Although some parameters have to be set in advance to facilitate niching, the challenge to develop a PSO niching algorithm requiring minimal prior

knowledge of the function landscape, remained. An additional challenge comprised using and extending the principles on which the original PSO algorithm rests to facilitate niching, resulting in an efficient and elegant solution. Such a strategy should also be robust enough to operate successfully in convoluted function landscapes where the shapes, sizes, and placing of niches differ considerably.

1.2 Objectives

The main objectives of this thesis can be summarized as follows:

- To extend the essential building blocks of the basic PSO paradigm to facilitate niching.
- To develop an efficient, elegant, and robust niching algorithm that identifies candidate solutions in an unknown problem space, calculate niche boundaries, and optimize subswarms occupying the niches to yield multiple optima.
- To ensure that the algorithm requires minimal prior knowledge of the function landscape.
- To compare different niching approaches using a diverse problem set.
- To conduct a scalability study of the niching algorithm.
- To extend the niching algorithm to incorporate dynamic optimization problems.

1.3 Methodology

The main purpose of this thesis is the development of a novel strategy to induce niching in multimodal function landscapes. The assumption that the vector dot product of the social and cognitive direction vectors can be used to determine niche boundaries is explained and motivated. A technique to calculate separate niche radii and form subswarms occupying the niches, is presented. Three algorithms that have been developed consecutively, are presented. The sequential vector-based PSO identifies niches sequentially and optimizes the subswarms contained in the niches in turn. The parallel vector-based PSO also identifies niches sequentially, but subswarms occupying the niches are optimized in parallel while duplicate subswarms are merged at specific intervals. The enhanced vector-based PSO includes a strategy to prohibit unnecessary merging of niches.

All three algorithms are tested for a number of one-dimensional and two-dimensional benchmark functions with well-known characteristics, using a system-supplied random number generator as well as Sobol sequences to calculate initial particle positions. Selected benchmark functions were also used to test the sensitivity of the algorithm for a range of granularity values. An upper bound for the granularity was derived for each function.

A scalability study was undertaken relating the required swarm size to the number of optima in a specified search space. Three scalable benchmark functions were used. For each setting the function was tested with a range of swarm sizes, and optimal swarm sizes were deduced from graphs of the results. A linear relation between optimal swarm size and number of optima was established for each of the three functions.

To assess the performance of the vector-based PSO in comparison to that of two other niching algorithms, NichePSO and the species-based PSO, all three algorithms were tested on seven two-dimensional functions using the same settings. The functions were chosen to represent a range of landscapes in order to observe how specific situations impacted on the performance of the algorithms.

The vector-based PSO was extended to locate and track optima in changing environments. The moving peak benchmark [25] was used to set up a number of scenarios including spatial movement of 3 and 5 peaks across a defined two-dimensional search space, variation of the value of the function at stationary positions, peaks that are obscured during movement, and new peaks that appear at unknown positions. Each of these situations were used to test the performance of the dynamic vector-based PSO. To conclude, the influence of severity (the spatial change in the position of the optimum) on performance was tested. The movement of one peak across a landscape containing three peaks was tracked using a range of severity values to deduce an upper bound for the severity in a specific scenario.

1.4 Contributions

The main contributions of this thesis are:

- Proposing a strategy to address the problem of finding niche radii in a multimodal function landscape. Niches could be located and demarcated with minimal prior knowledge of the objective function.

- Developing a niching algorithm for particle swarm optimization that uses the above strategy and compares well to other niching paradigms.

- An empirical analysis of state-of-the-art PSO niching methods.
- Extending the niching algorithm to locate and track multiple optima in dynamic environments.

1.5 Thesis outline

Chapter 2 presents an overview of various aspects of optimization. A brief discussion of mathematical optimization is followed by different viewpoints on optimization in nature, and how it is related to the development of population-based optimization algorithms. To conclude, the main features of evolutionary computation and its various forms are discussed.

Chapter 3 describes the inception and development of the original particle swarm optimization paradigm. Several aspects concerning the improvement and refinement of the algorithm are reviewed. Particle trajectories and the concept of neighbourhoods in a swarm are briefly discussed. A classification of single-solution PSO algorithms is given followed by the description of a number of relevant strategies. Information sharing strategies and subswarm-based algorithms are emphasized.

Chapter 4 describes and elaborates on the concept of niching or speciation. Niching algorithms locate more than one optimum of an objective function in a search space. Genetic algorithm niching techniques are briefly reviewed. A number of techniques that adapt particle swarm optimization to locate and optimize functions with multiple optima are discussed in detail. These algorithms use different strategies to identify candidate solutions, estimate niche boundaries, and optimize separate subswarms to converge on different optima. Algorithms where these processes take place sequentially as well as in parallel, have been developed. In the case of parallel niching algorithms, a merging component is incorporated. Improvements and refinements of some of the algorithms are also discussed.

Chapter 5 presents the development of a new niching strategy, the vector-based particle swarm optimizer (VBPSO). The principles underlying this approach are discussed, and three consecutive versions of the algorithm are described in detail. Extensive test results obtained by applying the various versions of the algorithm to a number of one- and two-dimensional functions are presented in chapter 6. The final version of the VBPSO is analysed by empirically testing the sensitivity of the algorithm for different parameter values, its ability to scale to functions in higher dimensions, and the relationship between the initial swarm size and the algorithm's performance. Chapter 6 is concluded by presenting the results of a compara-

tive study of the performance of three diverse PSO niching algorithms, namely the VBPSO, NichePSO [13] [14] and the species-based PSO [57]. These algorithms are applied to a number of two-dimensional functions with varying characteristics.

Chapter 7 investigates the behaviour of the vector-based particle swarm optimization paradigm in a changing environment. The positions as well as the quality of multiple optima can change dynamically over time. The vector-based PSO that locates multiple optima in static environments, is adapted and extended to track multiple optima over time. In chapter 8 a number of scenarios illustrating a variety of dynamic changes is set up and tested. The chapter concludes with an investigation of the influence of severity on dynamic changes.

A summary of the developments and results obtained in this thesis, is presented in chapter 9.

Appendix A presents empirical results of tests that were run to determine specific settings used by the vector-based PSO algorithms. These settings include minimum subswarm sizes, merging interval sizes and minimum sizes of subswarms capable of tracking moving optima.

A list of publications is presented in appendix B. These papers have led to, or are derived from the work presented in this thesis.

Chapter 2

Optimization

This chapter presents an overview of various aspects of optimization. A brief discussion of mathematical optimization is followed by different viewpoints on optimization in nature and how natural phenomena relate to the development of population-based optimization algorithms. To conclude, the main features of evolutionary computation and its various forms are discussed.

2.1 Introduction

The term **optimization** is derived from the Latin root *optimus* meaning *best*. The optimum refers to the best or most favourable set of conditions. Therefore, optimization can be described as the process that has to be followed in order to obtain a set of conditions where a specific desirable feature will have the best or most effective possible value within a certain environment.

Problems for which the solution lies in finding an optimal value for some objective, are known as *optimization problems*. Typically, an optimization problem has many candidate solutions, each represented by a set of possible conditions. Depending on the characteristics and requirements of the problem, an acceptable solution should be found, which may be the best overall solution, or one of a set of feasible solutions.

Many optimization problems can be found in everyday life, ranging from navigating through rush-hour traffic to cost-effective economic policies, production scheduling in manufacturing, and process optimization in large petro-chemical plants.

Optimization is commonly used in a mathematical context, where it refers to the minimization or maximization of mathematical functions by choosing, by means of some strategy, values from within an allowed set in order to find the best solution.

However, optimization is a much broader concept. In order to survive, man had to find ways and means to overcome obstacles in a hostile environment. Strategies had to be devised to maximize the availability of food sources and favourable conditions for procreation, while minimizing any danger to the continued existence of the species. Going even further back, plant and animal species have evolved over millions of years by adapting to their environment in the most profitable manner. Features have been developed to facilitate finding food, defense against predators, and procreation of the species.

The remainder of this chapter is organized as follows: Classical mathematical optimization is briefly reviewed in section 2.2. Basic concepts, problem classification, and mathematical optimization methods are discussed. Section 2.3 covers stochastic local search algorithms, while some background on optimization in nature is given in section 2.4. The chapter is concluded by a brief summary of evolutionary algorithms in section 2.5.

2.2 Mathematical optimization

Classical mathematical optimization methods form a necessary background to any study involving some form of optimization. Formal definitions of basic concepts and maximization are given, followed by the classification of optimization problems according to specific features. A subset of the most important mathematical optimization methods, divided into derivative and non-derivative methods, are discussed.

2.2.1 Basic concepts

Mathematical optimization can be described formally as the process of

1. the *formulation* and
2. the *solution* of a constrained optimization problem of the form

$$minimize\ f(\mathbf{x}),\ \mathbf{x} = [x_1, x_2, \ldots, x_n]^\tau \in \Re^n \tag{2.1}$$

subject to the constraints:

$$g_j(\mathbf{x}) \leq 0,\ j = 1, 2, \ldots, m$$

$$h_j(\mathbf{x}) = 0,\ j = 1, 2, \ldots, r$$

where $f(\mathbf{x})$, $g_j(\mathbf{x})$ and $h_j(\mathbf{x})$ are scalar functions of the vector $\mathbf{x}$ and τ refers to some tolerance [93].

The components of equation (2.1) can be described as follows:

- The *objective function*, $f(\mathbf{x})$, represents a mathematical formulation of the problem to be optimized.
- The variables, x_i (the design variables), affects the value of the objective function.
- The inequality constraints, $h_j(x)$, and equality constraints, $g_j(x)$, restrict values that can be assigned to the variables. If no constraints or only boundary constraints are specified, the problem is referred to as an unconstrained minimization problem.

2.2.2 Maximization

Maximization of a function can be effected by modifying the standard form given in equation (2.1) as follows:

$$maximize\ f(\mathbf{x}) = minimize\ (-f(\mathbf{x})) \tag{2.2}$$

Any inequality constraints should then be modified accordingly.

Both minima and maxima can be referred to as optima, depending on how the problem is described and how the optimization process is implemented.

2.2.3 Problem classification

Optimization problems can be classified according to specific features. The choice of an optimization algorithm will, to a large extent, depend on the particular type of problem, and should be appropriate for the specific application. The following points should be taken into consideration when deciding on a strategy.

Linear versus non-linear optimization problems: For some optimization problems the objective as well as all the constraints, are linear functions [93] [98]. These problems are called *linear programming problems*. Figure 2.1 depicts a linear programming problem in two dimensions. Efficient techniques have been developed to solve these problems, one of the most famous being the simplex method proposed by Dantzig in 1947 [93].

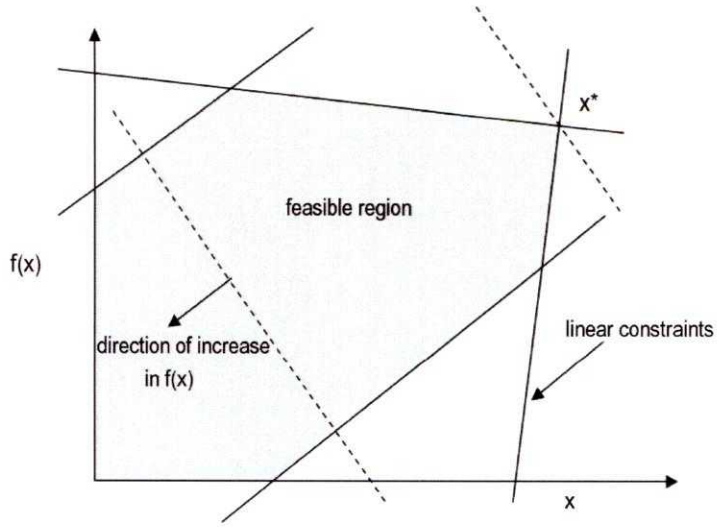


Figure 2.1: Two-dimensional linear programming problem

Non-linear optimization problems are generally more difficult to solve, and numerous techniques have been developed for this purpose.

Dimensionality of the objective function: The number of variables in the objective function may vary from one to several. Figure 2.2 and Figure 2.3 depict functions of one and two variables with single minima at $\mathbf{x}^*$.

Local and global optima: Some optimization problems are such that more than one optimum exists in the search space. The values of the objective function at these positions may differ significantly, differ very slightly, or can even be the same. The optimum with the best value is the global optimum, while the other optima are referred to as local optima. More formally, given that optima imply *minima*, local and global minimizers are defined as follows: A *local minimizer*, $\mathbf{x}^*_B$, of the region B, is defined so that

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

where $B \subset S \subseteq \Re^n$ and S denotes the search space.

A *global minimizer*, $\mathbf{x}^*$ is defined so that

$$f(\mathbf{x}^*) \leq f(\mathbf{x}), \forall \mathbf{x} \in S \tag{2.4}$$

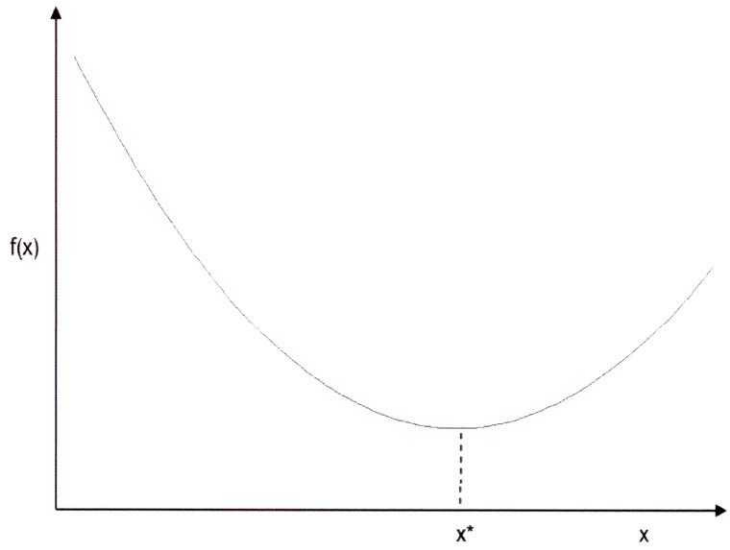


Figure 2.2: Function of single variable with minimum at x^*

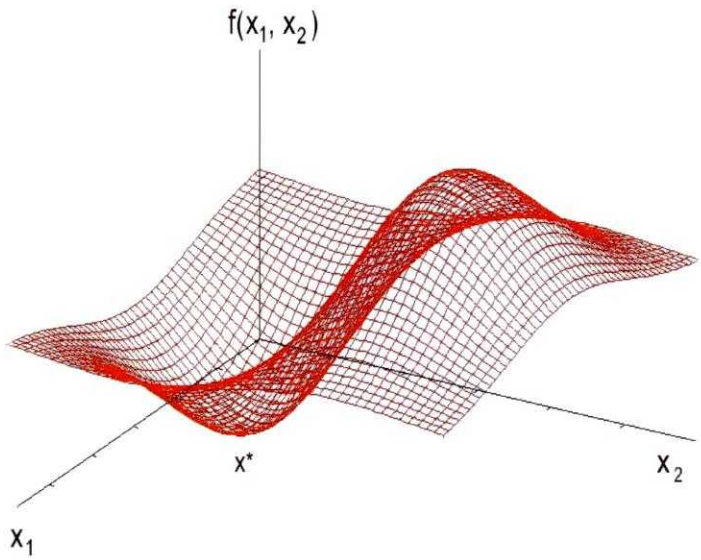


Figure 2.3: Function of two variables with minimum at x^*

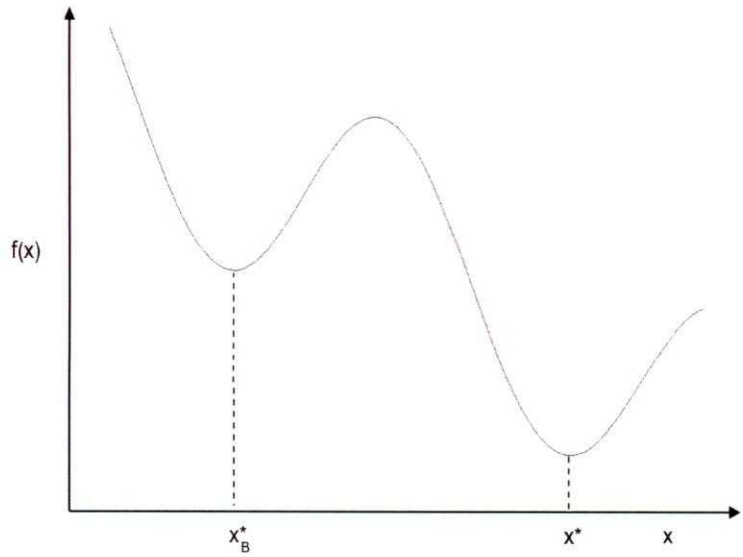


Figure 2.4: Function of a single variable with global minimum at x^* and local minimum at x^*_B.

where S is the search space.

Figure 2.4 depicts an objective function with a global minimum and one local minimum.

Constraint functions: Optimization problems can be divided into *unconstrained* and *constrained* problems. Unconstrained minimization takes place when an objective function is optimized over the entire search space. Although the boundaries of the search space are also constraints, these problems are referred to as unconstrained optimization problems.

With constrained optimization problems, one or more constraint functions are defined to restrict the region where optimization should take place. If $g(\mathbf{x})$ denotes the inequality constraint function, such that $g(\mathbf{x}) \leq 0$, the contour $g(\mathbf{x}) = 0$ divides the search space into a *feasible region* and an *infeasible region* [93] [98].

In the case of the equality constraint function, $h(\mathbf{x})$, the feasible values will only be found on the contour or surface where $h(\mathbf{x}) = 0$.

Multi-solution optimization: An objective function may, in addition to global optima, contain several local optima. As these local optima may represent alternative feasible so-

lutions, it is often necessary to find all or most solutions in a particular search space. Multi-solution optimization is also referred to as *multi-modal optimization.*

Multi-objective optimization: The simultaneous optimization of several objective functions is necessary in many real-world problems, where objectives may work against each other [33]. The result is a set of optimal solutions representing trade-offs between different, often conflicting, objectives.

Optimization in dynamic environments: Most real-world optimization has to be carried out in a dynamic environment, that is, the objective function changes over time. Such environments require algorithms that locate and track changing optima. Single or multiple solutions can be tracked, depending on the problem requirements.

This thesis focuses on multi-solution optimization, using particle swarm optimization. Algorithms are developed to locate multiple solutions in a search space. These algorithms are also adapted to track multiple optima in a dynamic environment. Throughout the thesis, only unconstrained, single-objective functions are considered.

2.2.4 Mathematical optimization methods

Numerous methods for solving optimization functions of many variables have been developed, tested and applied to practical problems [93]. It is often claimed that some optimization techniques are superior to others, a statement that is contradicted by the "No Free Lunch" (NFL) theorem of Wolpert and Macready [103]. This theorem states that all optimization techniques have the same behaviour over all $f : X \rightarrow Y$ where X and Y are finite sets. However, differences in the behaviour of optimization techniques have been observed [26]. For several optimization scenarios specific techniques have been found to be superior to general ones. It can also be reasoned that although the NFL theorem has been proved for the set of *all* functions, it does not necessarily hold for all subsets of this set [98]. Most real-world functions have some structure and compact descriptions, forming a small subset of all functions. Therefore the choice of a method depends on the specific problem and will, to a large extent, determine the efficiency and accuracy of the solution. The advent of the computer has played an important part in the development of numerical methods using an iterative approach to find better approximations to the optimum until the desired accuracy is achieved.

A primary method to find local extrema of a function is to locate points where the derivative of the function is zero, provided that the function is differentiable. An alternative method is

to evaluate the function many times and search for a local minimum. Therefore, numerical optimization methods can be divided into derivative and non-derivative methods.

Non-derivative methods

Non-derivative optimization methods are methods that do not require any derivative of the objective function. For a non-derivative optimization method to be efficient, the number of function evaluations should be reduced while an accurate solution must be reached in reasonable time. Two well-known and efficient methods are discussed below [64]:

The golden ratio method: This method is suitable for functions that are unimodal, that is, one minimum occurs in a specific interval. The method works as follows: Assuming that the initial interval is [0,1], it is divided into three subintervals [0,1-r], [1-r,r] and [r,1]. Depending on whether $f(x)$ is decreasing or increasing at the interval boundaries, a new interval, consisting of the two leftmost or the two rightmost subintervals, is formed and this smaller interval is divided into three subintervals. The process is repeated until the boundaries of the subinterval where the minimum occurs, differ less than a given error margin, in which case the minimum can be approximated.

The Nelder-Mead method: Nelder and Mead devised a simplex method to find the minimum of a function of n variables [67]. The term *simplex* describes a generalized triangle in n dimensions. For two variables, a simplex will be a triangle. Initially, function values at the three vertices are found. The search generates a sequence of triangles that are formed by replacing the worst vertex of each triangle with a new vertex in such a way that the function values at the vertices become smaller while approaching the minimum point.

Derivative methods

Derivative optimization or direct-search methods use the derivative of the objective function at certain points in the search space to direct the search for the position where the function has a minimum or maximum value.

A large number of direct-search algorithms for unconstrained minimization of functions have been developed. All such algorithms require an initial estimate of the optimum [93]. From this starting point a sequence of estimates $\mathbf{x}_0, \mathbf{x}_1, \mathbf{x}_2, \ldots$, are generated. Assuming the optimum is

a minimum point, search for a better estimate takes place in the direction of *descent*; that is, where the derivative of the function becomes smaller. The process is repeated until no further descent is possible, or if the condition for a minimum, $\nabla f(\mathbf{x}) = 0$, is sufficiently accurately satisfied in which case $x^* \approx 0$.

For smooth functions, a sub-class of the direct search methods, namely the *line search* descent methods, is suitable. The direction of descent u^{i+1} is selected at each iteration x^i such that

$$\frac{df(\mathbf{x}^i)}{d\lambda}|_{u^{i+1}} = \nabla^\tau f(\mathbf{x}^i)u^{i+1} < 0 \tag{2.5}$$

where λ indicates the minimizer, ∇ is the gradient and τ refers to some tolerance.

Within the sub-class of line search descent methods, a number of methods exist that differ in the way in which the descent directions, u^i, are chosen, as well as the way in which the line search is performed.

The *method of steepest descent* is a simple and well-known method for the minimization of a function of n variables. It was first proposed by Cauchy in 1847 [93] [64]. This method is a *first order method* because first order partial derivatives are employed to compute the search direction at each iteration. For an n-dimensional function, $f(\mathbf{x})$, partial derivatives are evaluated in turn, each time starting with the previous minimum and changing the search direction. Iteration will produce a sequence, $\{\mathbf{x}_k\}$, of points with the property $f(\mathbf{x}_0) > f(\mathbf{x}_1) > \ldots > f(\mathbf{x}_k) > \ldots$. If $\lim_{k\to\infty} \mathbf{x}_k = \mathbf{x}_B^*$ then $f(\mathbf{x}_B^*)$ will be a local minimum for f.

2.3 Stochastic local search algorithms

Derivative and non-derivative optimization methods depend, to a large extent, on a well-chosen starting point. Roughly, these methods can be classified as local methods, as the search process is initialized at some point in the search space and proceeds by iteratively moving from one location to a neighbouring one depending on local knowledge only [44]. In contrast to systematic search algorithms where the search space is traversed systematically, there is no guarantee that a local search algorithm will find a solution. Local search methods can also visit the same location more than once or get stuck in some part of the search space from which they cannot escape.

Local search algorithms may generally seem inferior to systematic algorithms, but due to time constraints, using systematic algorithms is often not viable. Ideally, search algorithms should deliver reasonably good solutions within a limited time. However, depending on the

problem, a good solution may be reached by combining local and systematic algorithms.

Local search algorithms often make use of stochastic mechanisms and are then known as *stochastic local search* algorithms. These algorithms have been proved useful for solving both NP-complete decision problems and NP-hard combinatorial problems [44]. The concept of hill-climbing, as well as algorithms such as simulated annealing [54] and tabu search [38] can be classified as using stochastic search mechanisms. In addition, population-based algorithms, finding their inspiration in the behaviour of natural populations, all use some form of stochastic search.

2.4 Optimization in nature

Optimization is not a new concept. Inhabitants of the world in which we live have evolved over millions of years into populations best suited to survive in their various environments. The amazing diversity of these populations and the myriad of ingenious ways in which they have adapted to their environment suggest some stochastic processes as well as the principle that came to be known as survival of the fittest [22]. To ensure their survival, these populations had to optimize the parameters describing their existence. Survival strategies have evolved in these populations in different ways. Some animals and insects have developed special features to protect themselves against predators and utilize their environment to their best advantage. Other creatures, especially birds and insects, form colonies, nests or swarms. It is from these, often highly specialized systems, that researchers gleaned their inspiration to develop optimization methods that model the way in which the systems work, as perceived by the researcher. The optimization system is usually more simplified than the natural system it purports to model - natural systems being much more complicated and not always well understood by man. However, this does not prevent the researcher from building an artificial system with the broad principles observed in the natural system as a starting point.

Many of these artificial systems are population-based, having found their inspiration in the behavioural patterns of populations of animals like flocks of birds, schools of fish, and termite and ant colonies. Interesting theories of group behaviour and the forces that seem to govern it, have been formulated over the years [63] [75] [80]. Some of the first studies in the behavioural patterns of animals and insects in their natural environment were undertaken in the early years of the 20th century by Eugène Marais, a South African writer, lawyer and naturalist [63]. Among others, the behaviour of termites inhabiting large termite nests, were carefully

observed and documented. Experiments suggested that the queen influences the movements of all the termites, even when they are far away from the nest. On the other hand, synchronous movements of swarms of locusts were observed where no central influence could be identified. Marais called this phenomenon the "group soul", an influence keeping the group or swarm together even if no fixed centre exists.

Different explanations for the highly coordinated movements of flocks of birds or schools of fish were put forward. Research done by Wayne Potts in 1984 showed that maneuver waves spread through a flock of birds much faster than a single bird's mean reaction time [75]. Potts called this the "chorus line hypothesis" and attributed it to an individual anticipating an approaching wave.

In 1986 Craig Reynolds showed that complex behaviour can be generated by applying simple rules. He illustrated the concept by creating "boids", computer-generated flocking organisms [80]. Three basic laws govern the movement of each boid, namely separation, alignment and cohesion. A mechanism was thus developed to create computer-generated flocks of creatures for use in films and multi-media. A current popular view is to attribute the phenomenon to emergent behaviour where the flock's movements are determined by decisions of individual birds following simple rules when responding to movements of their neighbours.

Many advantages for being part of a flock exist. A flock's collective intelligence will serve as a defense against predators, find food sources more easily and avoid being lost, while moving in groups even reduces energy expenditure as birds are able to glide more often. Therefore, it can be concluded that group behaviour can be seen as a form of optimization. The best food sources, adequate defense, and profitable energy expenditure all ensure the survival of the group as an entity, while the survival of the group, swarm, or nest takes precedence over the fate of the individual.

In general, computer systems can now be built where collections of independently acting entities exhibit collectively intelligent behaviour in an environment that these entities can sense and alter. This behaviour is known as swarm intelligence, an example of biology as a source for computing.

2.5 Computing paradigms inspired by natural optimization processes

Swarm intelligence forms only one branch of a class of computing paradigms that owe their existence to the observation and study of natural biological systems. The field of artificial intelligence (AI) hosts a number of these paradigms, including mechanisms with an ability to learn or adapt to new situations [32]. These systems are collectively known as computational intelligence (CI) which comprises a number of main paradigms: artificial neural networks (NN), evolutionary computation (EC), swarm intelligence (SI), fuzzy systems (FS), and artificial immune systems (AIS). The branch of evolutionary computation shows some characteristics similar to swarm intelligence, as both are population-based and exhibit adaptive behaviour in order to obtain better solutions to optimization problems. The development of various sub-branches of evolutionary computation preceded the advent of the discipline of swarm intelligence. Therefore, to place swarm intelligence in perspective, a brief description of evolutionary computation is presented.

The development of evolutionary algorithms was inspired by the behaviour of populations of individuals in nature. For such populations to survive, individuals of a population have to adapt to an often hostile environment. For centuries agriculturists have cultivated plants and bred domestic animals to acquire traits useful to man. During the great voyages of discovery in the 18th and 19th centuries, many new plant and animal species were discovered. Some isolated populations have developed in peculiar ways to adapt to their environments. In the absence of predators some species have even lost some of their features.

Charles Darwin, an English botanist, also joined an expedition of the "Beagle" in the capacity of a naturalist [22]. He visited the Galapagos archipelago in 1835. In the absence of humans and predators, some of the most unique life forms on earth, highly adapted to their harsh surroundings, have developed on the islands. Observations here and in other remote locations, emphasized the struggle for existence among these animals and plants. The idea that favourable variations would tend to be preserved and unfavourable ones destroyed, formed the basis of his theories about the formation of new species. Thus natural selection or "survival of the fittest" would ensure that the population having evolved over many generations, would be able to survive and reproduce.

These ideas as well as the science of genetics, inspired the development of several optimization algorithms [24] [40] [43] [55]. All of these use randomly chosen populations of individuals,

a fitness function to determine how good the solution is, and a strategy to produce offspring from the individuals.

The following techniques comprise the main sub-areas of evolutionary computation:

Genetic algorithms: Computer programs that resemble natural selection were pioneered in the late fifties and early sixties. The original genetic algorithm strategy involves a population of random strings of 1's and 0's that are generated to solve a particular problem. The measure of fitness depends on the type of problem. The process of selecting parents, recombining parents through crossover to form offspring and then selecting a new generation, is repeated until the population converges. From time to time a small fraction of strings are mutated to introduce diversity. Many modifications of the original algorithm have been tested in a variety of contexts [32], and used to solve a number of practical optimization problems [43].

Genetic programming: Genetic programming (GP) [55] is an evolutionary-based methodology to optimize a population of computer programs. The purpose of genetic programming is to find programs that perform a user-defined task. Reports on genetic programming methodology were published during the 1980's by Smith [91], Cramer [21], and Forsyth [37], while John R. Koza pioneered the solving of complex optimization problems by means of genetic programming [55].

Evolutionary programming: Evolutionary programming (EP), originally proposed by Lawrence J. Fogel in the 1960's, differs from genetic algorithms and genetic programming in that evolutionary programming evolves behavioural models instead of genetic models [35] [36]. An important difference between evolutionary programming and the other paradigms is that only mutation is used in order to introduce variation in a next population.

Evolutionary strategies: The ideas of adaptation and evolution also led to the development of evolutionary strategies (ES), developed by Rechenberg and Schwefel in the 1960's and 1970's [79] [86]. The focus of evolutionary strategies is on *evolution of evolution*. A set of strategy parameters is defined that influences the evolution process. While the population is optimized, the strategy parameters are adapted, thus optimizing the process itself.

Differential evolution: Differential evolution (DE) is a stochastic population-based optimization algorithm introduced by Storn [94] and Price [78] in 1996. DE is distinguished from other population-based techniques by the differential mutation mechanism where

the target vector is mutated by the addition of a random difference weighted vector. A trial vector is constructed and offspring is produced by crossover between a parent and the trial vector.

Cultural evolution: Cultural evolution, inspired by human social evolution, enhances the search process by including prior knowledge about the domain [81]. Culture can be defined as a system of symbolically encoded conceptual phenomena that are socially and historically transmitted within and between social groups [27]. Cultural changes occur much faster than biological evolution in natural systems, thus speeding up the search process.

2.6 Conclusion

A brief discussion of aspects of optimization was presented in this chapter. Principles from mathematical optimization that were deemed to be relevant to the objectives of this thesis were reviewed. As many modern population-based algorithms glean inspiration from adaptive natural systems, some background and interesting observations of emergent behaviour in nature were described. Finally, several evolutionary computing paradigms were briefly reviewed, as it exhibits some similarities to particle swarm optimization.

Chapter 3

Particle Swarm Optimization

This chapter describes the inception and development of the original particle swarm optimization paradigm. Several aspects concerning the improvement and refinement of the algorithm are reviewed. Particle trajectories and the concept of neighbourhoods in a swarm are briefly discussed. A classification of single-solution particle swarm optimization algorithms is given followed by the description of a number of relevant strategies. Information sharing strategies and subswarm-based algorithms are emphasized.

3.1 Introduction

In keeping with the interest in natural phenomena as the force behind the development of strategies to solve optimization problems, studies of the coordinated movements of bird flocks inspired a fascinating new optimization paradigm. In their landmark publication, *Particle Swarm Optimization* [48], James Kennedy and Russell Eberhart reason that, similar to humans, underlying collective memory could be profitable to bird flocks. Although the original intent of the study was to graphically simulate the movement of a bird flock, the realization that birds flock towards a food source by capitalizing on one another's knowledge, led to the idea that the movement of each agent is guided by two values: the agents's own previous best position as well as the best position of the entire flock. In the case of a bird flock, the objective would be to reach a food source and the best position would be the nearest position to that source.

A unique feature of this paradigm is the association of a velocity with each agent, emulating

the movement of bird flocks through space. Position changes are effected by adjusting the velocity over a number of steps or iterations. The original experiments done by Kennedy and Eberhart [48] added a random amount, weighted by a parameter of the system, to the previous velocity at each step. The velocity is adjusted in the direction of an agent's previous best position, called *pbest*, as well as the group's previous best position, called *gbest*. If the new position of an agent is better, that is, nearer to the food source, pbest is updated. If the new value for pbest proves to be better than the previous best position of the entire flock, gbest is also updated [48]. Results from these experiments showed that, with velocity parameters set to relatively small values, this simulation of a bird flock is very realistic, exhibiting much of the flock's graceful movements before the target, the food source, is reached.

3.2 Particle swarm optimization: The algorithm

Further experiments by Kennedy and Eberhart on their flock simulation yielded a simplified algorithm that could optimize multidimensional functions that proved to be difficult to optimize using conventional methods [48]. In accordance with models developed for applications in artificial life [66], the behaviour of the simulation was perceived to be more like a swarm than a flock. Also, although the initial candidate solutions or agents are only points in the search space, velocities and accelerations are more appropriate for particles. Thus the paradigm was labeled *particle swarm optimization* (PSO).

Because the concept of PSO stems from the observation of group dynamics in nature, it can be classified as a population-based algorithm [28]. Although not similar to evolutionary algorithms, both these paradigms have roots in the observation of natural phenomena. The use of an objective function for evaluating a candidate solution is practised in evolutionary algorithms as well as PSO. However, it must be emphasized that the inception of all these algorithms have only been inspired by the natural phenomena after which, to a certain extent, they have been named and do not purport to simulate nature in all its complexities.

The particle swarm optimizer maintains a population of particles in n-dimensional space. Initially, particles are randomly distributed throughout the search space, where each particle represents a potential solution to an optimization problem. If S is the size of the swarm and i denotes a specific particle, characteristics of i are represented by the following symbols [98]:

$\mathbf{x}_i$: The current *position* of the particle;

$\mathbf{v}_i$: The current *velocity* of the particle;

$\mathbf{y}_i$: The *personal best position* of the particle.

Together with each particle's position and velocity, a personal best position is maintained where the objective function yields the best fitness of all positions visited so far by that particle. Such a personal best position represents part of the memory of a particle, one of the unique characteristics of the PSO paradigm.

In 'flying' through hyperdimensional space, particles continually search for better positions. If such a position is found, the personal best is updated as follows, assuming minimization:

$$\mathbf{y}_i(t+1) = \begin{cases} \mathbf{y}_i(t) & \text{if } f(\mathbf{x}_i(t+1)) \geq f(\mathbf{y}_i(t)) \\ \mathbf{x}_i(t+1) & \text{if } f(\mathbf{x}_i(t+1)) < f(\mathbf{y}_i(t)) \end{cases} \tag{3.1}$$

Another strength of PSO is the concept of social memory. Particles benefit by capitalizing on the knowledge of their neighbours. If a better solution is discovered by a single member, the entire population will move in the direction of that solution. Simultaneously the best position of the entire swarm, or part of it, is updated. The symbol $\hat{\mathbf{y}}$ indicates the best position discovered by any member of the swarm or grouping within the swarm, defined as follows:

$$\hat{\mathbf{y}}(t) \in \{\mathbf{y}_0(t), \mathbf{y}_1(t), \ldots, \mathbf{y}_s(t)\} | \, f(\hat{\mathbf{y}}(t)) = min\{f(\mathbf{y}_0(t)), f(\mathbf{y}_1(t)), \ldots, f(\mathbf{y}_s(t))\} \tag{3.2}$$

The core of the particle swarm optimizer comprises the adjustment of the velocity associated with each particle during a single iteration. The previous velocity vector, the velocity vector towards the particle's personal best (pbest) and the velocity vector towards the entire swarm's best value (gbest), are linearly combined. The functions to be optimized are not always simple, and many suboptimal solutions may exist in the search space. To diversify the search and explore new regions in multidimensional space, random coefficients are introduced. Two independent random sequences, $\mathbf{r}_1 \sim U(0,1)^n$ and $\mathbf{r}_2 \sim U(0,1)^n$, are used. The constants, $0 < c_1, c_2 \leq 2$, called the *acceleration constants*, control the maximum values of $\mathbf{r}_1$ and $\mathbf{r}_2$. The velocity of a particle in a swarm is updated as follows:

$$v_{i,j}(t+1) = v_{i,j}(t) + c_1 r_{1,j}(t)[y_{i,j}(t) - x_{i,j}(t)] + c_2 r_{2,j}(t)[\hat{y}_j(t) - x_{i,j}(t)] \tag{3.3}$$

where $j = 1, \ldots, n$.

It is clear from equation (3.3) that the values of c_1 and c_2 can be manipulated to increase or decrease the movement in the direction of either the personal best or global best positions. Depending on the values of c_1 and c_2, a particle may also fly past the target before being pulled back in the direction of the previous best values.

The velocity is used to update the position of each particle:

$$\mathbf{x}_i(t+1) = \mathbf{x}_i(t) + \mathbf{v}_i(t+1) \tag{3.4}$$

The original PSO algorithm consists of, after initializing all particle positions, personal best positions and velocities, repeated updating of the velocity and position of each particle. Simultaneously the personal best position of each particle is updated if better values are found. The best position of the entire swarm, or neighbourhood when a subswarm is optimized, is also updated when appropriate. After a number of iterations, or when specific stopping conditions are met, the algorithm terminates and the solution is the last global best position, or the best neighbourhood best position.

3.2.1 Global and local particle swarm optimizers

Eberhart and Kennedy [28] implemented two versions of the initial particle swarm optimizer: the *global* version that keeps track of the best value, *gbest*, of the entire swarm, and the *local* version where the best value of a particle's nearest neighbours are retained and used to update the velocity of those particles [28]. The two versions are called the GBEST and LBEST models respectively. The GBEST PSO is summarized in Algorithm 1.

Algorithm 1 The GBEST particle swarm algorithm

Initialize a population of particles with random positions
  in n dimensions;
Set velocities associated with each particle to 0;
repeat
  for *each particle* $i \in [1, \mathrm{S}]$ **do**
    if $f(S.x_i) < f(S.y_i)$ **then**
      $S.y_i = S.x_i$;
    end
    if $f(S.y_i) < f(S.\hat{y})$ **then**
      $S.\hat{y} = S.y_i$;
    end
    Update velocity using equation (3.3);
    Update position using equation (3.4);
  end
until *stopping condition is true*;

The local version of the particle swarm optimizer is similar to the global version except that

particles can only access information of their nearest neighbours' best positions [28]. The best evaluation found in this group (nbest) is used together with each particle's previous best value (pbest) to calculate a new velocity and new position for each particle. Neighbourhoods consist of a predetermined number of particles adjacent to one another. The notion of adjacency is, however, based on the assumption that particle information is stored in arrays, and that the indices of particles of a neighbourhood are adjacent. Because particle positions are stochastically determined, the positions of the neighbourhood particles are not necessarily adjacent in the search space.

The purpose of this version of the particle swarm optimizer was apparently to create different groups of particles that explore different regions of the search space and so increase the diversity and the ability to optimize a function that may contain local optima. Particles within such a neighbourhood have no relationship to each other, as selection of the particles forming a neighbourhood is based on particle indices. Thus, neighbourhoods overlap and a particle may be part of more than one neighbourhood. The main differences between the GBEST PSO and LBEST PSO are [32]:

- The GBEST PSO converges faster, but swarm diversity is lost.
- As the result of improved diversity, the LBEST PSO is less likely to converge on a local minimum.

Suganthan [95] proposed a number of improvements for the standard PSO algorithm to improve the quality of the solutions as the number of iterations was increased. A variable neighbourhood operator was introduced. Starting with a neighbourhood consisting of a single particle, the size of the neighbourhood is gradually increased to include all particles. The neighbourhood is defined in two different ways: by using particle indices and by choosing a fraction of particles that are physically close to the particle for which a neighbourhood is sought. The second method uses a spatial neighbourhood and can be computationally intensive as Euclidian distances between all particles have to be calculated. Experimental results showed that the quality of the solutions improved when using dynamic neighbourhoods [95].

3.2.2 Cognition-only and social-only models

The particle swarm paradigm originates from the observation of bird flocks where individual members profit from the discoveries and previous experience of the flock in their search for

food [48]. The idea of knowledge sharing can also be found in human societies and its social structures [49].

If the PSO paradigm is viewed as simulating the ability of human societies to process knowledge, two distinct parts can be identified, a cognitive part and a social part. The term $c_1 r_{1,j}(t)(y_{i,j}(t) - x_{i,j}(t))$ is perceived to represent private thinking, as a new position in the search space is calculated using the experience of the best previous position. A particle swarm optimizer using only the cognitive component is known as the *cognition-only model* [49]. For this model particles tend to search the regions where they have been initialized, and behave as independent hill-climbers. The velocity update formula is modified as follows:

$$v_{i,j}(t+1) = v_{i,j}(t) + c_1 r_{1,j}(t)(y_{i,j}(t) - x_{i,j}(t)) \tag{3.5}$$

According to Kennedy [49] and Carlisle and Dozier [17], the cognition-only model is less successful and slower than the model with a cognitive as well as a social component, referred to as the full model. However, the cognition-only model proved to be useful when combined with other techniques to develop a niching algorithm [13] [16].

The velocity update formula for the *social-only* PSO version [49] is:

$$v_{i,j}(t+1) = v_{i,j}(t) + c_2 r_{2,j}(t)(\hat{y}_j(t) - x_{i,j}(t)) \tag{3.6}$$

For the *social-only model*, individuals have no tendency to return to positions that previously proved to be successful for themselves. All particles are attracted towards the best position in their neighbourhood so that this version converges faster [17] [49]. However, depending on the problem, there is a tendency to converge on local optima, as particles does not explore their own neighbourhoods sufficiently.

3.2.3 The inertia weight

The addition of velocity adds several unique characteristics to the particle swarm [50] [88]. Considering the metaphor of a swarm of particles flying through hyperdimensional space, the addition of velocity at every step increases the resulting velocity of a particle until it flies past the target and is pulled back towards previous personal best and neighbourhood best values. Thus the search for an optimum is diversified and other more remote regions of the search space can be explored. However, for the GBEST model it is necessary to control the velocity in order to prevent particles from leaving the search space. Therefore the velocity of a particle is limited by some value V_{max} [49]. Limiting the velocity is also referred to as *velocity clamping*, which is

implemented by clamping speed to control the global exploration of particles. If $V_{max,j}$ is the maximum allowed step size in dimension j, the speed is adjusted before particles positions are updated:

$$v_{i,j}(t+1) = \begin{cases} v'_{ij}(t+1) & \text{if } v'_{ij}(t+1)) < V_{max,j} \\ V_{max,j} & \text{if } v'_{ij}(t+1)) \geq V_{max,j} \end{cases} \tag{3.7}$$

where $v_{i,j}(t+1)$ is computed using equation (3.3).

A value for $V_{max,j}$ should be selected carefully, as the swarm will not explore the search space sufficiently if the value is too small, while a too large value might have the effect that good regions of the search space are not exploited sufficiently.

To control the velocity and to improve the performance of the particle swarm optimizer, a new parameter, the *inertia weight*, was introduced by Shi and Eberhart [88]. It was argued that a very small velocity causes particles to statistically contract to the current global optimum, resembling a local search algorithm. A larger velocity causes exploration of new regions, resulting in a global search ability. Different problems require different balances between the local search ability and global search ability. To implement this, an inertia weight w was brought into the equation as follows:

$$v_{i,j}(t+1) = wv_{i,j}(t) + c_1 r_{1,j}(t)(y_{i,j}(t) - x_{i,j}(t)) + c_2 r_{2,j}(t)(\hat{y}_j(t) - x_{i,j}(t)) \tag{3.8}$$

$$x_{i,j}(t+1) = x_{i,j}(t) + v_{i,j}(t+1) \tag{3.9}$$

where w is the inertia weight.

The inertia weight acts as a scaling factor that modifies the contribution of the previous velocity to the current update equation. In their initial empirical studies, Shi and Eberhart [88] investigated the effect of different values for $w \in [0, 1.4]$. Algorithms with an inertia weight in the range [0.8,1.2] were found to converge faster. Dynamically changing inertia values, where large inertia values decrease over time to smaller values, also improve the performance of the particle swarm optimizer significantly [88]. These results can be explained by noting that an optimization algorithm generally needs more exploration ability at the beginning of a run while exploitation ability is necessary when the optimum is approached. Further studies by Shi and Eberhart [89] concluded that the selection of the inertia parameter and maximum velocity may be problem-dependent.

The ultimate purpose of PSO is to explore the search space where the objective function is defined and, though the landscape may contain many suboptimal solutions, locate the overall or global best position. Through exploitation of the surrounding search space, the accuracy

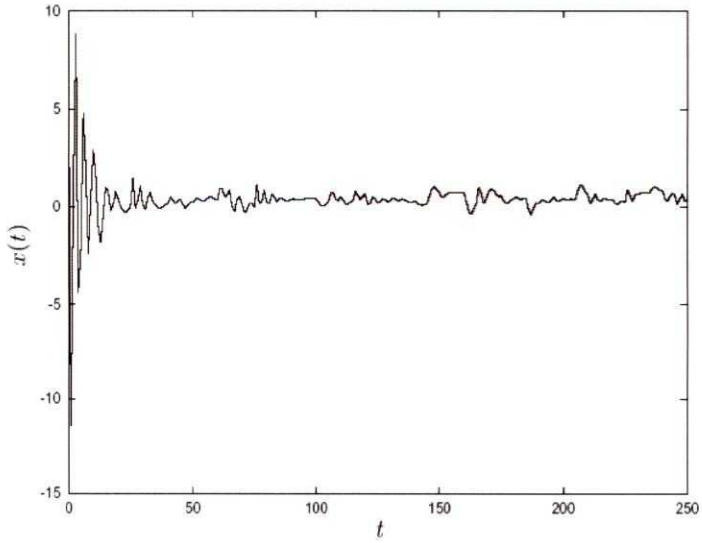


Figure 3.1: Stochastic particle trajectory, obtained using $w = 0.7$ and $c_1 = c_2 = 1.4$ [100]

of the solution is improved. The choice of control parameters such as inertia and acceleration coefficients determines whether the PSO will exhibit divergent, convergent or cyclic behaviour, and whether exploration of the search space is sufficient. Control parameter settings can not be seen in isolation. Choosing a value for w has to be made in conjunction with the selection of values for c_1 and c_2 [33]. Some studies empirically found certain parameter choices to work very well, for example Eberhart and Shi's choice of $w = 0.7298$, $c_1 = c_2 = 1.49618$ [29]. However, these choices should not be generalized as they are based on a limited sample of problems and may also be problem-dependent.

Theoretical studies of particle trajectories played a large part in gaining insight into optimal parameter choices. Ozcan and Mohan [68] conducted formal analyses showing that, in the absence of stochastic influences, the trajectory of a particle follows a sinusoidal wave. Each particle acquires a random frequency and amplitude during the search. The analyses were extended to incorporate a multidimensional and multi-particle system [69].

Particle trajectory studies done by Ozcan and Mohan [68] [69] and Clerc and Kennedy [20] use a simplified PSO system without an inertia term. Van den Bergh and Engelbrecht [100] conducted an analysis of particle trajectories with the inertia weight included. Formal proof is presented that each particle i of a *gbest* PSO converges to a stable point $\mathbf{p}_i$, that is, if $\{\mathbf{x}_i(t)\}_{t=0}^{+\infty}$

is a sequence of particle positions, it is shown that

$$\lim_{t \to +\infty} \mathbf{x}_i(t) = \mathbf{p}_i \tag{3.10}$$

Thus the trajectory of a simple particle with inertia converges to a stable point, which is a weighted average of $\mathbf{y}$ (the personal best position) and $\hat{\mathbf{y}}$ (the global best position).

A heuristic to select the best values for w, c_1 and c_2 could now be derived [100]. Van den Bergh and Engelbrecht found that, to guarantee convergence, the following relation has to be satisfied:

$$w > \frac{1}{2}(c_1 + c_2) - 1 \tag{3.11}$$

An example of the trajectory of a particle using the parameter settings $w = 0.7$ and $c_1 = c_2 = 1.4$ that satisfy the relationship given in equation (3.11) is shown in Figure 3.1 [100].

3.2.4 The constriction factor

Conditions to guarantee the convergence of the particle swarm to an equilibrium state were also studied by Clerc [19]. Clerc and Kennedy [20] proposed the incorporation of a *constriction factor* into the particle swarm update equation to balance the exploration-exploitation trade-off and prevent the velocity from growing out of bounds. Velocities have traditionally been contained by implementing a V_{max} parameter. However, the need for such a parameter is eliminated by the implementation of properly defined constriction coefficients. The effect of a constriction factor is similar to that of the incorporation of an inertia weight, but velocity clamping is not necessary. A value for the constriction factor χ was derived in terms of c_1 and c_2.

The modified velocity update equation is presented in the following equation:

$$v_{i,j}(t+1) = \chi(v_{i,j}(t) + c_1 r_{1,j}(t)(y_{i,j}(t) - x_{i,j}(t)) + c_2 r_{2,j}(t)(\hat{y}_j(t) - x_{i,j}(t))) \tag{3.12}$$

where

$$\chi = \frac{2}{|2 - \phi - \sqrt{\phi(\phi - 4)}|} \tag{3.13}$$

with

$$\phi = \phi_1 + \phi_2$$

$$\phi_1 = c_1 r_1$$

$$\phi_2 = c_2 r_2$$

The purpose of the constriction coefficient, χ, is to reduce the velocity at each time step in order to ensure convergence to a stable point [33]. Applying the constraints $\phi \geq 4$ and $\kappa \in [0, 1]$ guarantees swarm convergence, as χ evaluates to a value in the range $[0, 1]$. The value of κ determines the extent to which either exploration or exploitation dominates behaviour of the swarm. Local exploitation is favoured for $\kappa \approx 0$, resulting in fast convergence, while $\kappa \approx 1$ has the effect that convergence is slow with a high degree of exploration.

Eberhart and Shi compared the performance of particle swarm optimization using an inertia weight with the performance of a PSO with a constriction factor [29]. Empirical results showed that the constriction factor approach incorporating velocity clamping as well, performs better than the inertia weight approach. Comparing the equations representing the inertia weight approach with those representing the constriction factor approach, it can be shown that the inertia weight approach is similar to the constriction factor approach if the inertia weight w is set to χ, and c_1 and c_2 is chosen such that $\phi = c_1 + c_2$ where $\phi > 4$. Theoretical analyses by Van den Bergh and Engelbrecht [100], and Trelea [97] confirmed that careful selection of the inertia weight w as well as c_1 and c_2 result in improved performance.

3.2.5 Neighbourhoods

The discussion of two early PSO models, GBEST and LBEST in section 3.2.1 introduced the concept of a *neighbourhood*. Social structures and communication within social networks can, to a large extent, be seen as the driving force behind particle swarm behaviour. The LBEST version of the PSO as described by Eberhart and Kennedy [28] model social behaviour by making use of neighbourhoods. Overlapping neighbourhoods are created where particles of each are scattered throughout the problem space and will eventually converge to the neighbourhood best and then to the global best. Thus the diversity of the swarm is improved.

Neighbourhoods form a basis for communication within a swarm. Individuals in a neighbourhood influence one another; the more successful an individual is, the greater the influence on other members of the neighbourhood. Thus the quality of the entire neighbourhood will be enhanced. Group performance is influenced by the quality of communication within the group which is again affected by the structure of the social network [51]. In particle swarm optimization neighbourhoods may have different topologies. Particles assigned to a neighbourhood are selected according to some structure based on array indices if the swarm is implemented as an array of particles [51]. The different topologies indicate the degree of connectivity and

the amount of information interchange in a social network. Highly connected networks favour faster convergence, but the danger also exists that local minima may be reached. Sparsely connected networks converge slower, but if clustering occurs, the search space may also not be covered sufficiently [33].

Several social network structures have been empirically studied. A number of these are described below [51] [53] [65]:

The star social structure: A star structure, illustrated in Figure 3.2, describes a social structure where all particles are interconnected. Such a structure corresponds to the original GBEST PSO model. All particles communicate with one another. Information flow through the network is fast, as each individual is attracted to the best solution found so far. Populations tend to converge rapidly, but are susceptible to convergence on local optima [51].

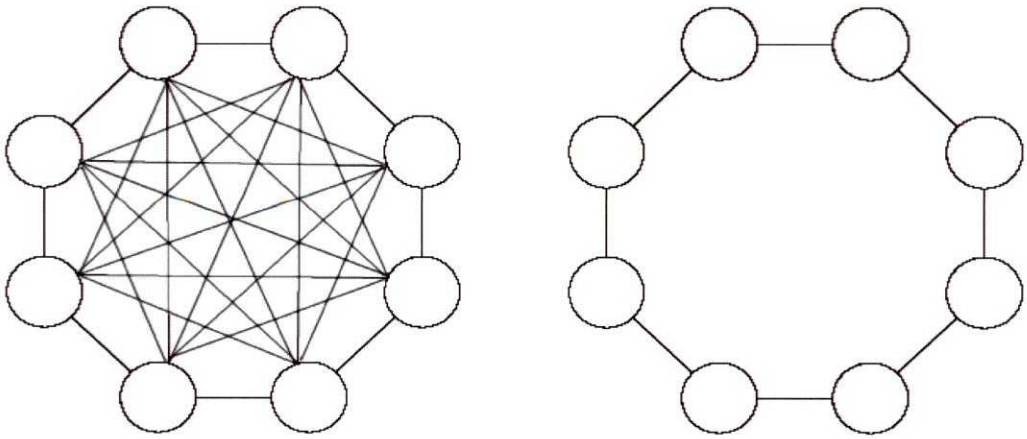
The ring social structure: The original LBEST PSO model uses a ring structure, illustrated in Figure 3.2, to form neighbourhoods. Each individual is affected by its K immediate neighbours. If $K = 2$, each particle communicates with its two adjacent neighbours. Neighbourhoods formed in such a manner will overlap, as adjacency must be understood in terms of particle indices and not physical adjacency in the search space.

The wheel social structure: For the wheel structure, illustrated in Figure 3.2, one individual is connected to all others, which are connected only to that individual. All information flow through this focal point. Too rapid conversion on local optima is prevented by the buffering effect of the focal particle, and the propagation of good solutions is slowed down.

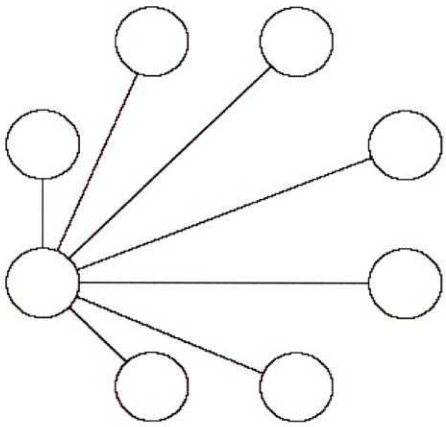
The pyramid social structure: This social structure has the shape of a three-dimensional wire-frame triangle [53]. Compared to the performance of other topologies, it produced relatively good results.

The four clusters social structure: Four clusters of particles are formed with two connections between clusters.

The Von Neumann social structure: Particles are connected in a grid structure [53]. Neighbours above, below, and on each side on a two-dimensional lattice are connected. The Von Neumann social structure was shown to perform very well in a number of empirical studies [53] [73].



(a)Star (b)Ring



(c)Wheel

Figure 3.2: Social network structures

The concept of different neighbourhoods of particles within a swarm is central to the development of a number of algorithms, referred to as *niching algorithms* to solve multi-solution optimization problems. Genetic algorithm niching techniques such as fitness sharing [40], clearing [74], the sequential niching technique of Beasley *et al.* [4] and species conserving genetic algorithms developed by Li *et al.* [56] all use some form of a *niche radius* to demarcate a portion of the swarm which is then optimized separately to converge on either the global or one of the local optima. Particle swarm niching techniques such as NichePSO [13], the species-based PSO [57] and the vector-based PSO developed in this study, also rely on a niche radius in order to divide the swarm into subswarms. These techniques are addressed in depth in chapters 4 and 5.

When neighbourhoods are created to facilitate niching, it is understood to be spatial neighbourhoods. Such neighbourhoods are defined as a subset of a swarm of particles in the same region of the problem space. Suganthan [95] proposed that neighbourhoods be formed on the basis of spatial proximity in order to improve the performance of single-solution PSO. The Euclidian distance between particles determines whether particles are close enough to one another to qualify being in the same neighbourhood. Given neighbourhoods of size n_N, Suganthan defines the neighbourhood of particle i as the n_N particles closest to particle i. The original PSO algorithm is modified to facilitate a finer grained search by employing neighbourhoods. A neighbourhood is created by calculating distances between particles and choosing a number of particles near to particle i. The best particle in the neighbourhood, *lbest*, is calculated. Neighbourhood sizes are gradually increased. Updating each neighbourhood means that each *lbest* moves in the direction of *gbest* but at a much slower rate of convergence, giving the swarm more exploratory power.

Using spatial neighbourhoods is computationally expensive as distances between particles have to be calculated at each iteration. Thus, if neighbourhoods are created to improve diversity, neighbourhoods based on particle indices are preferable.

3.3 Variations on the particle swarm optimizer

Since the inception of the particle swarm optimizer, many variations have been proposed and tested with the purpose of improving the performance of the algorithm [33]. Enhancements to the basic PSO comprise improvements to the accuracy of the solutions, the probability of locating the overall optimal value as well as the extent to which convergence could be guar-

anteed. Single-solution PSO algorithms for continuous, unconstrained problems are classified by Engelbrecht according to the technique that was implemented to improve the performance [33]:

Social-based algorithms: These algorithms use different information sharing strategies. Particles influence one another in different ways, thus changing the way in which the neighbourhood best position is calculated.

Hybrid algorithms: Aspects of evolutionary computation and ant colony optimization as well as strategies such as simulated annealing and gradient descent, are incorporated into the traditional PSO.

Sub-swarm-based: Algorithms are included where the swarm is divided into sub-swarms and manipulated in different ways in order to improve diversity and accuracy of the solutions.

Memetic algorithms: Exploration ability of PSO is improved by the incorporation of other techniques implementing local search.

Multi-start algorithms: Parts of the swarm or the entire swarm are restarted at certain stages.

Repelling methods: Particles are repelled from one another in order to improve diversity.

This study emphasizes optimization algorithms with the ability to locate multiple solutions. Such algorithms are applied to problems described by multi-modal functions in demarcated search spaces. These algorithms are referred to as *niching* or *speciation* algorithms and are described in depth in chapter 4. Some of the strategies used to facilitate niching, have their roots in techniques used in single-solution PSO algorithms. The concept of neighbourhoods as introduced in the LBEST PSO model [28], has already been discussed. A number of relevant single-solution PSO algorithms are presented in this section.

3.3.1 Information sharing strategies

The idea of neighbourhoods form a salient feature of these algorithms. The following strategies are discussed:

$$\sum_{m=1}^{n_{\mathcal{N}_i}} \frac{\mathbf{r}(t)(\mathbf{y}_m(t) - \mathbf{x}_i(t))}{n_{\mathcal{N}_i}}$$

where $n_{\mathcal{N}_i} = |\mathcal{N}_i|$, $\mathcal{N}_i$ is the set of particles in the neighbourhood of particle i and $\mathbf{r}(t) \sim U(0, c_1 + c_2)^n$. To calculate the velocity at the next time step, the above term is added to the current velocity and the result is scaled by either the inertia weight or the constriction factor. For the second approach, the contribution of each particle is scaled by a weight depending on the performance of that particle. Mendes *et al.* tested both approaches on a number of benchmark functions using different population topologies. Both approaches performed better than the standard PSO.

Fitness-distance ratio PSO

The fitness-distance ratio PSO (FDR-PSO), developed by Veeramachaneni *et al.* [102], addresses the problem of premature convergence by adjusting the velocity of each particle towards the best previous positions visited by its neighbours. Similar to the fully informed PSO [65], a particle is most likely to be influenced by more successful individuals in its neighbourhood. However, the influence of multiple other particles may cancel each other, resulting in a reduced possible benefit. The FDR-PSO algorithm counteracts this possibility by selecting only one other particle when updating each velocity dimension. Such a particle is chosen subject to the following criteria:

- It must be near to the particle being updated.
- It should have visited a position with fitness better than the particle being updated.

A simple and robust way to update each velocity dimension comprises selecting a particle that maximizes the ratio of the fitness difference to the one-dimensional distance. For dimension $j = 1, \ldots, n$, a particle referred to as *nbest* is chosen to maximize

$$FDR(i, m, j) = \frac{f(\mathbf{y}_m(t)) - f(x_i(t))}{|y_{mj}(t) - x_{ij}(t)|} \tag{3.14}$$

where FDR refers to the fitness-distance ratio. Veeramachaneni *et al.* [102] showed the FDR-PSO to perform significantly better than the original PSO and several of its variants, when tested on a number of benchmark functions.

The technique used by the FDR-PSO to quantify the influence of multiple particles on the updating equation in a neighbourhood inspired an enhanced speciation algorithm developed by Li [57]. The Fitness Euclidian-distance Ratio-based PSO (FER-PSO) is discussed in chapter 4.

3.3.2 Subswarm-based approaches

Grouping of particles into subswarms can primarily be seen as an effort to diversify the search process. In order to locate the overall optimal position, the entire search space should be explored and premature convergence inhibited. Engelbrecht [33] classifies subswarm-based PSO approaches into cooperative PSO algorithms and competitive PSO algorithms. A selection of cooperative PSO approaches is briefly discussed in this section.

Multi-phase PSO

Løvbjerg *et al.* [60] combined the particle swarm with concepts from evolutionary algorithms to develop two hybrid particle swarm optimizers. These algorithms can be classified as hybrid algorithms with the ideas of subpopulations and breeding incorporated to prevent premature convergence to suboptimal points. However, the breeding between subswarms is a form of cooperative PSO, as genetic material is exchanged between parents of different subswarms [33]. The structure of the hybrid model is illustrated in Algorithm 2.

Algorithm 2 The structure of the hybrid model

```
begin
    initialize;
    while not terminate-condition do
        evaluate;
        calculate new velocity vectors;
        move;
        breed;
    end
end
```

A breeding model as well as a subpopulation model are presented. The breeding model produces offspring by randomly selecting two parents. Offspring are produced by arithmetic crossover on the position of the parents for each dimension. Velocity of the offspring is calcu-

lated as the sum of the velocity vectors of the parents normalized to the original length of each parent velocity vector. Parent particles are then replaced by their offspring particles.

The subpopulation model extends the breeding hybrid PSO model by dividing the particles into a number of subpopulations, each having its own unique best known optimum. Breeding takes place between particles from different subswarms, which could result in an escape from a local optimum.

Al-kazemi and Mohan [1] describes an adaptation of the PSO algorithm to discrete optimization problems. Particles are divided into two subswarms of equal size. At any given time, each particle is in one of two possible phases, an attraction phase and a repulsion phase. Attraction and repulsion refer to the movement of particles towards or away from the global best position. Directions of these movements are controlled by the coefficients in the velocity update equation, which excludes the personal best position:

$$v_{ij}(t+1) = wv_{ij}(t) + c_1 x_{ij}(t) + c_2 \hat{y}_j(t) \tag{3.15}$$

Phase switching takes place either after a set number of iterations, or if no global best fitness improvement is observed in a specified number of steps. Unlike other PSO variants, particle positions are updated by incorporating a hill-climbing procedure. A particle position is permitted to change only if such a change improves fitness.

Cooperative split PSO

Stochastic optimization algorithms such as particle swarm optimizers and genetic algorithms almost always experience reduced performance with increased dimensionality. Van den Bergh and Engelbrecht introduced the cooperative split PSO (CPSO-S_K) [101] to improve the performance of the original algorithm. Multiple swarms are used to optimize different components of the solution vector cooperatively. If each subswarm represents one dimension, the number of subswarms will be equal to the dimensionality, n, of the problem. However, the function to be optimized requires an n-dimensional vector. Therefore subswarms cannot be optimized separately. To address this impediment, a *context* vector is constructed by concatenating the global best particles from each of the n swarms to form an n-dimensional vector. In swarm j, for example, the other $n-1$ components in the context vector are kept constant while the jth component of the context vector is replaced in turn by each particle from the jth swarm.

Van den Bergh and Engelbrecht reported a marked improvement in performance over the standard PSO when CPSO-S_K was tested on several benchmark optimization problems [101].

3.3.3 Memetic PSO algorithms

Empirical studies have shown that many optimization algorithms lack the ability to refine a solution once a promising region of the search space have been located. Mechanisms that have been provided to balance exploration and exploitation can be put into effect to favour exploitation in the later stages of the optimization process. In addition, special features can be incorporated to address this problem. For example, a local optimizer can be embedded between the iterations comprising the search process. Al-kazemi and Mohan [1] incorporated a hill-climbing process in their multi-phase PSO discussed earlier. To reduce computational complexity and yield similar performance, local search methods such as hill climbing can be applied only to neighbourhood best solutions.

Incorporating local search methods is also applicable to niching algorithms, especially since the subswarms and species that form an integral part of many of these algorithms consist of limited numbers of particles.

The guaranteed convergence PSO

The guaranteed convergence PSO (GCPSO) [100] incorporates a local search heurisitc related to basic hill-climbing. The algorithm forms an integral part of NichePSO, a niching PSO developed by Brits *et al.* [13] [14] [16]. NichePSO is discussed in chapter 4.

Particle swarm optimizers have a property that make them susceptible to premature convergence resulting in inaccurate solutions. Van den Bergh and Engelbrecht found that a particle may reach a state where $\mathbf{x}_i = \mathbf{y}_i = \hat{\mathbf{y}}$, meaning the particle's current position, its personal best position and the global best position is the same [100]. In such a case the velocity update will depend only on $wv_{i,j}(t)$, meaning that the previous velocity and inertia factor will be responsible for moving the particle to another position. If previous velocities are very close to zero, all particles will stop moving once they catch up with the global best position. The swarm will converge before a solution is reached. Such behaviour is referred to as *stagnation*.

A new parameter was introduced to the PSO algorithm. If τ is the index of the global best particle so that

$$\mathbf{y}_\tau = \hat{\mathbf{y}}$$

a new velocity update equation for the global best particle was suggested:

$$v_{\tau,j}(t+1) = -x_{\tau,j}(t) + \hat{y}_j(t) + wv_{\tau,j}(t) + \rho(t)(1 - 2r_{2,j}(t)) \tag{3.16}$$

The idea of the new equation is to search in a small region around the best global position, $(\hat{\mathbf{y}}_j)$, for a position with better fitness. The term $-x_{\tau,j}(t)$ resets the particle's position to $\hat{y}_j$. The search direction is the current search direction, thus $wv_{\tau,j}(t)$ is added. The term $\rho(t)(1-2r_{2,j}(t))$ generates random positions in a region with side lengths $2\rho(t)$. ρ is a scaling factor generated as follows:

$$\rho(t+1) = \begin{cases} 2\rho(t) & \text{if \# successes} > s_c \\ 0.5\rho(t) & \text{if \# failures} > f_c \\ \rho(t) & \text{otherwise} \end{cases} \tag{3.17}$$

The number of consecutive failures or successes are given by the terms #failures and #successes respectively. A failure is defined as $f(\hat{\mathbf{y}}(t)) \geq f(\hat{\mathbf{y}}(t-1))$, assuming minimization, while s_c and f_c are threshold parameters, depending on the objective function. More detail can be found in [99].

The position of the global best particle τ is updated by the following equation where the velocity is calculated by the new velocity update equation:

$$x_{\tau,j}(t+1) = \hat{y}_j(t) + wv_{\tau,j}(t) + \rho(t)(1-2r_2(t)) \tag{3.18}$$

The guaranteed convergence particle swarm optimizer (GCPSO) was tested on a number of unimodal and multimodal functions, and significantly faster convergence compared to the original PSO, was found. With smaller swarm sizes the effect was more pronounced. The algorithm can be classified as a local optimization algorithm and would be especially useful when swarms are partitioned into smaller subswarms to be optimized separately.

3.4 Conclusion

This chapter reviewed the inception and development of the particle swarm optimization paradigm. The original algorithm was discussed while an overview of the most important improvements was given. Aspects such as trajectories followed by particles as a result of the associated velocity, were touched upon. Neighbourhoods in the swarm and the forming of subswarms were emphasized, as these concepts form the basis of a number of niching strategies which will be discussed in chapter 4. A number of relevant single-solution PSO algorithms were described with the emphasis on information sharing strategies and subswarm-based algorithms.

Chapter 4

Niching

This chapter describes and elaborates on the concept of niching or speciation. Niching algorithms locate more than one optimum of an objective function in a search space. Genetic algorithm niching techniques are briefly reviewed. A number of techniques that adapt particle swarm optimization to locate and optimize functions with multiple optima are discussed in detail. These algorithms use different strategies to identify candidate solutions, estimate niche boundaries and optimize separate subswarms to converge on different optima. Algorithms where these processes take place sequentially as well as in parallel, have been developed. In the case of parallel niching algorithms, a niche merging component is incorporated. Improvements and refinements of some of the algorithms are also discussed.

4.1 Introduction

Function optimization is usually understood as the process of locating a position where the function has the best possible value. The quest for efficient and effective techniques to accomplish this goal has been described in the preceding chapters. However, for various optimization tasks more than one optimum needs to be located. In engineering design, alternative designs that perform equally well, often exist [4]. If several optima can be located, the user can choose a design by considering criteria that have not been incorporated in the objective function. Another example of multi-solution optimization is locating all the resonance points in mechanical or electrical systems [23]. Depending on the application, the designer will need to maximize or

minimize all such resonances. Solving systems of linear equations is also required in many scientific and engineering problems, for example, in robotics and signal processing. Multi-solution optimization provides an efficient technique to solve such problems.

The Oxford dictionary describes a *niche* as a shallow recess, a definition that can also be used figuratively to indicate a container of entities exhibiting some common traits. This description is used in population-based optimization strategies such as evolutionary algorithms and swarm intelligence when all optima of a multi-modal function have to be located in a demarcated search space. When a function has more than one optimum, the part of the search space where individuals have a natural tendency to converge on a specific optimal solution, is known as a niche. Therefore algorithms designed to locate multiple optima, are referred to as *niching* algorithms.

In keeping with the biological origins of the various kinds of population-based optimization algorithms, the niching metaphor also has its roots in nature [4]. In ecology, a niche describes the relational position of a species or population in its ecosystem. Such an ecological niche describes how an organism or population responds to the distribution of resources and competitors and how those same factors may be altered by the organism or population. Thus different populations survive and can exist together by utilizing the environment in different ways. Different species evolve to fill different ecological niches. Therefore niching algorithms are also referred to as *speciation* or *species-based algorithms*.

Niching has originally been studied for evolutionary algorithms, in particular genetic algorithms [4]. Recently, a number of niching algorithms for particle swarm optimization have also been developed. A new PSO niching algorithm is presented in chapter 5.

4.2 Genetic algorithm niching techniques

Similar to the development of single-solution population-based algorithms, the development of niching methods for evolutionary algorithms preceded that of particle swarm optimization. Much of the concepts and terminology used in the study of niching algorithms, stem from their use with evolutionary algorithms, particularly genetic algorithms. Therefore, a brief discussion of relevant genetic algorithm niching techniques is presented.

Fitness sharing, a strategy to promote stable sub-populations or species, is described by Goldberg and Richardson [40]. The idea comes from natural ecosystems where different species evolve to fill each ecological niche. To implement fitness sharing, it must be known if individ-

uals occupy the same niche. Although the strategy is primarily concerned with encouraging diversity, useful concepts like calculating the distances between all individuals and introducing a *niche radius*, are fundamental to many niching techniques for genetic algorithms as well as particle swarm optimization.

Crowding, originally proposed by De Jong [24], and *deterministic crowding* proposed by Mahfoud [61], are based on the competition for limited resources in a natural population. Deterministic crowding recombine pairs of individuals to produce offspring, that replace their closest parent if the fitness is better. The idea of closeness, determined here by means of the phenotypic distance function, as well as the identification of some form of similarity among individuals, can be found in a number of niching algorithms.

Pétrowski [74] proposed a clearing procedure as a niching method for genetic algorithms. The method was inspired by the sharing of limited resources within subpopulations of individuals characterized by some similarities. The population is divided into subpopulations by first sorting individuals from best to worst according to the fitness values. To determine if individuals belong to the same subpopulation, a dissimilarity measure, referred to as the *clearing radius*, σ, is used. All solutions within distance σ from the best dominant individual are then *cleared*, that is, their fitness values are set to zero. The process is repeated for the next fittest solution, until a list of dominant individuals or *winners* remain. Each of these winners represents the fittest individual in a niche.

The clearing procedure was shown to significantly improve the performance of genetic algorithms applied to multimodal optimization. The use of a radius to identify a niche and the process to determine niches also feature in later algorithms such as the species conserving genetic algorithm [56] and the species-based PSO [57].

4.2.1 A sequential niching technique

A sequential niching technique for multimodal function optimization using genetic algorithms was proposed by Beasley *et al* [4]. For this technique, maximization is assumed. Maxima are located in sequence. However, repetition of the GA does not guarantee that a different solution will be found each time. Therefore, the sequential niching technique uses knowledge gained from previous executions of the optimization algorithm to prevent the region where a solution has been found from being searched again. A PSO niching technique, objective function stretching [71] [72] shows some similarities to the sequential niching technique for genetic algorithms. Therefore, a more detailed description of the said technique is deemed to

be relevant to the study of PSO niching.

The sequential niching technique maintains two fitness functions in order to carry over knowledge from previous executions of the algorithm:

- the original fitness function or *raw fitness function*, F, and
- the modified fitness function, M.

To compute the modified fitness function, a *distance metric* is required to indicate how close two chromosomes are. Beasley *et al* uses the Euclidian distance between two points occupied by individuals in n-dimensional space. The modified fitness function, $M(\mathbf{x})$, for an individual $\mathbf{x}$, is computed by multiplying the raw fitness function, $F(\mathbf{x})$, by a *single-peak derating function*. Initially, $M_0(\mathbf{x}) \equiv F(\mathbf{x})$. The optimization algorithm is run using the modified fitness function, keeping a record of the best individual, $\hat{\mathbf{y}}$, found in the run. The modified fitness function is updated to give a depression in the region near the best individual, producing a new modified fitness function, according to

$$M_{n+1}(\mathbf{x}) = M_n(\mathbf{x}) * G(\mathbf{x}, \hat{\mathbf{y}}) \tag{4.1}$$

where $G(\mathbf{x}, \hat{\mathbf{y}})$ is a single-peak derating function.

Various derating functions can be used, for example, the *power law* and *exponential* functions, respectively defined as

$$G_p(\mathbf{x}, \hat{\mathbf{y}}) = \begin{cases} (d_{x\hat{y}}/r)^\alpha & \text{if } d_{\mathbf{x}\hat{y}} < r \\ 1 & \text{otherwise} \end{cases} \tag{4.2}$$

and

$$G_e(\mathbf{x}, \hat{\mathbf{y}}) = \begin{cases} \exp(\log m * (r - d_{\mathbf{x}\hat{y}})/r) & \text{if } d_{\mathbf{x}\hat{y}} < r \\ 1 & \text{otherwise} \end{cases} \tag{4.3}$$

The above functions assume maximization. For minimization, stretching functions are used.

The distance between $\mathbf{x}$ and $\hat{\mathbf{y}}$, as determined by the distance metric, is given by $d_{\mathbf{x}\hat{y}}$. The values of the derating functions increase as this distance increases. The curve described by the power law function can be concave, convex or linear. The power factor, α, determines how concave ($\alpha > 1$) or convex ($\alpha < 1$) the derating curve is. Curves of the exponential function are concave. In equation (4.3), m is the minimum value of the derating function, G, at which point $d_{\mathbf{x}\hat{y}} = 0$. The value of m also determines how concave the derating curve will be. Smaller values of m produce more concavity.

The niche radius, r, an estimated value depending on the inter-niche distance, requires prior knowledge of the objective function. However, Deb [23] proposed a technique to calculate r, provided that the number of maxima is known or can be estimated. Assume that a hypersphere of radius r surrounds each of the p maxima in the function, and that these hyperspheres do not overlap. The hyperspheres must completely fill the n-dimensional space. If the parameter range for each dimension is normalized to be 0 or 1, r is given by:

$$r = \frac{\sqrt{n}}{2 * \sqrt[n]{p}} \tag{4.4}$$

The purpose of the derating function is to modify the search space such that previously traversed parts of the search space are not re-explored. The derating function reduces the fitness of each individual by an amount that depends on the distance between that individual and each best individual found in previous runs. The modified fitness function now has a minimum imposed in areas where maxima were located, thus preventing the maxima to be located again. Figure 4.1 illustrates the raw fitness function, $F(\mathbf{x}) = sin^2(2\pi\mathbf{x})$, and the modified function after the raw function has been multiplied by the power law derating function with $x = 0.25$, $r = 0.25$, and $\alpha = 2$. In this example two small peaks or *false optima* remain. Larger values for α can reduce the height of these optima, but with too high values maxima of interest may be lost or the solutions may be incorrect.

After each execution of the optimization algorithm, the modified fitness function is updated by:

$$M_{n+1}(\mathbf{x}) \equiv M_n(\mathbf{x}) * G(\mathbf{x}, \hat{\mathbf{y}}) \tag{4.5}$$

Algorithm 3 describes sequential niching genetic algorithm of Beasley *et al.* [4]. A single application of this algorithm is referred to as a *sequence*, since it consists of several runs of the optimization algorithm (a GA or other search technique).

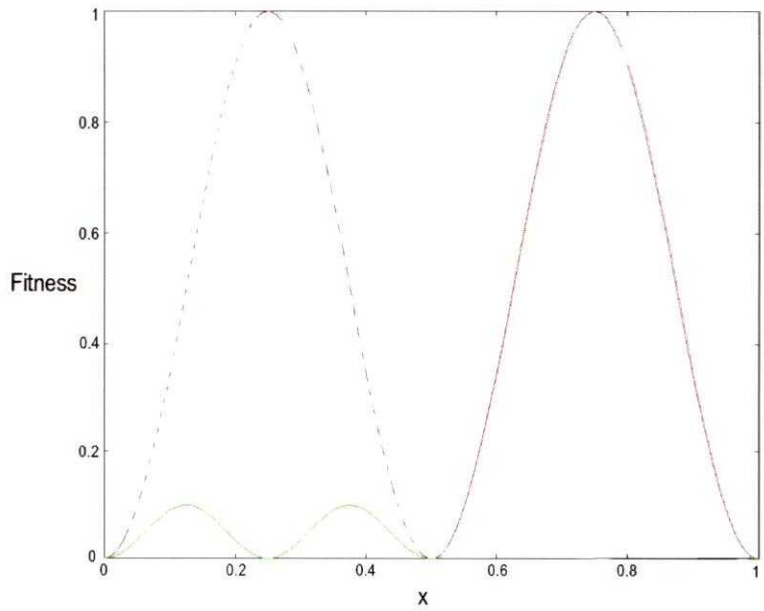


Figure 4.1: Suppression of one peak by a derating function

Algorithm 3 The sequential niching genetic algorithm

Initialize the *modified fitness function* to the initial, or *raw* fitness function;
repeat
  Run the GA to optimize the modified fitness function, keeping a record of the best individual found in the run;
  Update the modified fitness function to give a depression in the region near the best individual, producing a new modified fitness function;
  if *fitness of best individual* > *solution threshold* **then**
    display as a *solution*;
  end
until *the required number of solutions have been found*;

According to Algorithm 3, the best individual found in a run is displayed as a solution if the fitness of that individual is larger than the solution threshold. The *solution threshold* represents the lower fitness limit for maxima of interest. It is assumed that there is a known number, p, of maxima with fitnesses greater than the solution threshold. If the likely fitness values of the maxima of interest are not known, the solution threshold is set to 0 and the algorithm is terminated after the first p maxima have been located. The value of p is set beforehand.

Sequential niching proved to be a vast improvement on fitness sharing methods [4]. However, it does suffer from several drawbacks, listed below:

- Calculation of the niche radius requires the number of maxima to be known in advance. An objective function where the maxima are spread evenly throughout the search space, is assumed.
- False maxima are introduced by the derating function and may be located as solutions.
- To find q maxima, a single-solution optimization algorithm needs to be executed at least q times, which increases the computational complexity.

In order to solve the problem involving false maxima, Beasley *et al* [4] suggested using the modified fitness function to find an approximate solution. Using the original or raw fitness function, a local search method can then locate a more accurate solution.

4.2.2 A species conserving genetic algorithm

Species conserving genetic algorithms by Li *et al.* [56] evolve parallel subpopulations using species conservation. A species is a class of individuals with common characteristics. In natural ecosystems some species may become extinct due to a failure to adapt to a changing environment. However, if these species might be useful to other ecosystems or to humanity, some form of intervention could preserve a few individuals.

Li *et al.* define a species as a subset of the population containing individuals where the distance between any two individuals is less than a parameter, σ_s, the species distance. The species distance is also used to determine which individuals are worth preserving from one generation to the next. The algorithm has close ties to the species-based PSO [57] that will be discussed later in this chapter.

The notion of species is exploited by the species conserving genetic algorithm to achieve niching when optimizing multimodal functions. To locate multiple optima, individuals that are copied into the next generation have to include highly fit individuals, as well as individuals that, while not highly fit, are different enough from the current best individuals to be worth keeping. To establish which individuals have to be conserved, the population is partitioned into several species. Each species is dominated by an individual called the *species seed*. X_s denotes the set of species seeds found in generation t. To build the set X_s, each individual in generation t is considered successively in decreasing order of fitness. If X_s does not contain

any seed that is closer to that individual than half the species distance, $\sigma_s/2$, the individual becomes a new species seed and is added to X_s. The procedure for determining the species seeds is performed for every generation in a GA run.

Once all the species seeds have been found, the new population is constructed by applying the usual genetic operators, i.e. selection, crossover and mutation. Some species may not be fit enough to survive following these operations, but is nevertheless worth keeping. To enable them to survive, the species seeds in X_s are copied into the new population. Species are conserved as follows [56]:

- Mark all individuals as unprocessed.
- The new generation, $G(t+1)$, is searched for solutions belonging to the same species as each species seed $\mathbf{x} \in X_{(s)}$ identified in the previous generation.
- Species seed $\mathbf{x}$ replaces the worst of these "similar" solutions, provided $\mathbf{x}$ has better fitness and is marked as processed.
- If there are no solutions in the same species as $\mathbf{x}$ in $G(t+1)$, $\mathbf{x}$ replaces the worst unmarked solution in $G(t+1)$.
- As the species seeds are drawn from the previous generation, the number of species seeds N_s is always less than the population size, N, and therefore unmarked solutions must always exist.

Thus species seeds found in the current generation are conserved by moving them into the next generation.

Solutions to a multimodal optimization problem can, depending on the objective function, be a number of global optima or a number of dissimilar high-quality solutions. After the generation loop of the species conserving genetic algorithm is exited, these optima are identified by defining a *solution acceptance threshold*, $r_f \in [0, 1]$. Solutions are all individuals $\mathbf{x} \in X_s$ that satisfy

$$f(\mathbf{x}) \geq |(f_{max} - f_{min})|r_f \tag{4.6}$$

where f_{max} is the fitness of the most fit species seed and f_{min} is the minimum fitness in the final population.

4.3 PSO niching techniques

The purpose of the original particle swarm algorithm was to find a single optimum of an n-dimensional function which might be difficult to optimize in any other way [48], [51], [88]. Suboptimal solutions may be encountered while a PSO is in the process of converging, but niches are not formed where a subswarm can converge on such a suboptimum. One of the main reasons why niche formation is inhibited, is the influence of the social component of the velocity update equation [34]. Particles are attracted to the single best solution in a neighbourhood. Therefore, if a better solution is encountered, the entire swarm changes direction towards this more promising area in the search space. In the case of the GBEST PSO, the neighbourhood is the entire swarm, and all particles are attracted to the swarm's global best position. Therefore the GBEST PSO is incapable of niching [34]. The LBEST PSO constructs overlapping neighbourhoods consisting of a particle and its l immediate neighbours. Due to the overlapping neighbourhoods, all particles will eventually converge on the same point. Engelbrecht *et al.* [34] have empirically shown that the standard LBEST PSO is inefficient in locating and maintaining multiple solutions and cannot be used as a niching algorithm . Hence, if the problem is such that all the optimal local and global solutions in the range of values where the function is being investigated, are required, the algorithm needs to be modified.

The idea of optimizing portions of a swarm separately has been implemented in a variety of PSO algorithms, mainly to improve diversity. Eberhart and Kennedy introduced a local version of the original PSO model, called the LBEST model, where the velocity of each particle is updated depending on its own personal best value as well as the best solution in a topological neighbourhood, called lbest [28]. Particles constituting a neighbourhood are selected using particle indices. Most of the particles in these neighbourhoods will eventually converge, but convergence is slowed down, ensuring a better chance to locate a good solution. Suganthan proposed a partitioning scheme based on the spatial location of particles [95], and found that performance was improved for a number of test functions. Various other population structures and neighbourhood topologies were implemented, all with the purpose of improving the performance of the particle swarm optimizer [51], [52], [53].

The notion of neighbourhoods form a natural starting point when designing niching algorithms. A neighbourhood can be conceptualized as a collection of particles that will eventually converge on some optimum. However, to facilitate niching, spatial neighbourhoods will be more appropriate.

This section reviews a number of PSO niching strategies.

4.3.1 Objective function stretching

A function stretching technique proposed by Parsopoulos *et al.* was originally devised to overcome the problem of occasional convergence to local optima [71]. The principle of function stretching was also used by Parsopoulos and Vrahatis to modify the particle swarm optimizer in order to locate all the global minima of a function [72]. In some applications several minima exist where the minimum value of the function is similar. According to Parsopulos and Vrahatis [72], the original particle swarm will exhibit a cyclic movement over the search space in such cases, being unable to converge on one minimum. Such problems, as well as instances where a global minimum as well as some suboptimal solutions need to be located, can be solved by using the stretching technique.

The technique, which assumes minimization, exhibit some similarities to that of Beasley *et al.* [4]. Each time a minimum is discovered, the original function landscape is transformed by applying a stretching function to the original function. The PSO is prevented from returning to the previously discovered minimum, as the function landscape has changed and all local minima above the current minimum has been eliminated. Minima below the current minimum are not affected.

The following two-stage transformation to a new objective function, $H(\mathbf{x})$, is applied soon after a local minimum $\mathbf{x}^*$ of the function f has been detected:

$$G(x) = f(\mathbf{x}) + \gamma_1 \frac{\|\mathbf{x} - \mathbf{x}^*\| \cdot (sign(f(\mathbf{x}) - f(\mathbf{x}^*)) + 1)}{2} \tag{4.7}$$

$$H(\mathbf{x}) = G(\mathbf{x}) + \gamma_2 \frac{sign(f(\mathbf{x}) - f(\mathbf{x}^*)) + 1}{2\tanh(\mu(G(\mathbf{x}) - G(\mathbf{x}^*)))} \tag{4.8}$$

where γ_1, γ_2 and μ are arbitrary chosen positive constants and the sign function is defined as:

$$sign(p) = \begin{cases} -1, & \text{if } p < 0 \\ 0, & \text{if } p = 0 \\ +1, & \text{if } p > 0. \end{cases} \tag{4.9}$$

The sign function can also be approximated by

$$sign(p) \simeq \tanh(\lambda p)$$

for large values of λ.

The algorithm that uses the objective function stretching technique locate minima sequentially. Each time a minimum is discovered, the value of $\mathbf{x}^*$ is recorded. The function $f(\mathbf{x})$ is

set equal to $H(\mathbf{x})$ and all the particles in the swarm are re-initialized. Since particles have converged and will not be able to discover new minima, re-initialization is required. New random positions and personal best positions are calculated for all particles. The entire process is repeated until some stopping criterion is met. If the number of minimizers is known, the algorithm can run until all minimizers have been found. If the number of minimizers is unknown, a specific number can be requested, or the algorithm can run until the maximum number of iterations is reached.

However, Van den Bergh [98] has shown that the stretching technique introduces false local minima as well as misleading gradients. PSO exhibits a tendency to move 'down' a slope, and the function landscape of the new objective function, $H(\mathbf{x})$, will be such that the slope leads away from the minimizer. Therefore the PSO may converge to a boundary of the search space instead of the minimum.

4.3.2 The nBest PSO

Brits *et al.* developed a particle swarm optimization algorithm, *nbest*, to solve systems of unconstrained equations [12] [14]. While the standard *gbest* PSO easily solves systems of equations with one optimal solution, systems with multiple solutions require the implementation of a niching strategy. Brits *et al.* adapted the standard *gbest* PSO algorithm to locate multiple solutions in one run of the algorithm.

Using PSO requires solving systems of equations to be restated as an optimization problem. Each particle represents a candidate solution for each parameter in a system of equations. For example, in a system of two equations with two variables, x_1 and x_2, values for these variables have to be found, indicating positions where the lines intersect. A particle's fitness is determined by how close it is to a solution. Each equation of the system is converted to represent an error function for that equation. Therefore, for a system of two equations, where f_1 and f_2 are functions representing the error,

$$f(x_1, x_2) = |f_1(x_1, x_2)| + |f_2(x_1, x_2)| \tag{4.10}$$

The objective is to minimize $f(x_1, x_2)$ which is straightforward if there is only one solution. However, when the system of equations is such that it has more than one solution, a niching technique must be considered. The standard particle swarm optimizer is modified by initially extending the fitness function to reward a particle when that particle is close to any of the possible solutions in the system of equations. The fitness function for solving a system of K

equations,

$$f(\mathbf{x}_i) = \sum_{k=1}^{K} |f_k(\mathbf{x}_i)| \tag{4.11}$$

is redefined for multiple solutions. For example, if a system of linear equations consists of three equations that intersect in turn at three different positions, the fitness of particle $\mathbf{x}_i$ is calculated by equation (4.11) for each intersection, using the two linear equations of which the lines intersect. The fitness of $\mathbf{x}_i$ is the minimum of the three results. The new fitness function is then defined as:

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

An additional modification to the standard PSO model is the definition of neighbourhoods. While the *lbest* model uses topological neighbourhoods to diversify the search for better solutions and slow down convergence before locating a single optimum solution, the *nbest* model uses a different approach. An *Euclidian neighbourhood* is defined for each particle $\mathbf{x}_i$ as the n_i closest particles to $\mathbf{x}_i$. The Euclidian distances between $\mathbf{x}_i$ and other particles in the swarm are calculated to find the closest particles. The *neighbourhood best (nbest)*, $\hat{\mathbf{y}}_i$, of each Euclidian neighbourhood is defined as the center of mass of the positions of all the particles in the neighbourhood:

$$\hat{\mathbf{y}}_i = \frac{1}{k} \sum_{j=1}^{k} B_{i,j} \tag{4.13}$$

where the set B_i consists of the k closest particles to $\mathbf{x}_i$.

The number of particles in a neighbourhood is a user-defined parameter, k, that has to be chosen carefully. Considering the definition of $\hat{\mathbf{y}}_i$, k should not be too small, as a particle will blindly trail its closest neighbour. If k is too large, the algorithm will become similar to GBEST, yielding no information about a possible good result. Particles are updated as for the standard particle swarm optimizer, but the global best particle, $\hat{\mathbf{y}}$, is replaced with $\hat{\mathbf{y}}_i$. Therefore, particles move towards their neighbourhood bests, while neighbourhoods shrink over time. Eventually particles converge on optima in their regions, resulting in multi-solution optimization.

The *nbest* algorithm performed well when tested on a number of systems of linear and non-linear equations, as well as on other multimodal functions. A small inertia weight and

relatively large values of c_1 and c_2 allowed the particles to explore the search space more thoroughly, preventing premature convergence.

4.3.3 NichePSO

The *NichePSO* algorithm developed by Brits *et al.*, is a niching particle swarm optimizer where multiple solutions are located in parallel [13], [14], [16]. Multiple swarms are grown from an initial particle population. As niches are detected, subswarms are formed. Eventually each subswarm represents one of the potential solutions to the problem.

As in the case of the standard PSO, the swarm is initialized to distribute particles uniformly throughout the search space. The *NichePSO* uses *Fauré* sequences to generate initial particle positions. At this stage no niches have been detected and the entire swarm is referred to as the main swarm. All particles are trained using one iteration of the *cognition only model*. Only the cognitive component is used to update particle velocities [49]. Therefore particles move in the direction of more promising regions of the search space, facilitating subswarm formation.

Niching algorithms rely to a large extent on a good strategy to identify initial candidate solutions in the search space. Parsopoulos *et al.* identify potential solutions by using a threshold value, ϵ, where $f(\mathbf{x}_i) < \epsilon$. However, the effectiveness of this approach cannot be guaranteed where the objective function's landscape is unknown. *NichePSO* uses a similar approach to identify candidate solutions but, instead of using a threshold ϵ, the fitness of a particle is monitored by tracking its normalized standard deviation over a number of iterations. If the particle's fitness changes very little, that particle is identified as a candidate solution and a subswarm is created with the particle's closest topological neighbour.

Formally, the standard deviation, σ_i, of the fitness of particle i is tracked over a number of iterations, e_σ, set to 3 in the experiments of Brits *et al.* [14] [16]. A subswarm is created when $\sigma_i < \delta$, where δ is a small problem-dependent value. To avoid problem dependence, σ_i is normalized according to possible maximum and minimum values. The closest neighbour to particle $\mathbf{x}_i$ is particle $\mathbf{x}_k$, where c, the distance between the particles, is defined as:

$$c = \arg\min_{\mathbf{x}_k}\{\|\mathbf{x}_i - \mathbf{x}_k\|\} \tag{4.14}$$

where k is the index of any particle in the main swarm, with $k \neq i$.

Each subswarm initially consists of two particles, namely the candidate solution that acts as the neighbourhood best value, and its closest neighbour. While the main swarm is still searching the solution space, using the cognitive-only PSO for updating particle positions,

subswarms are updated separately. Initially, a subswarm consists of very few particles in a small region. To prevent swarm stagnation and premature convergence of these subswarms, the guaranteed convergence particle swarm optimization (GCPSO) algorithm [100] is used to update particle positions. GCPSO is described in Chapter 3 of this thesis.

To demarcate the subswarms that have been created, a subswarm radius is maintained and updated during iterations. Initially, the radius of each subswarm is the Euclidian distance between the candidate solution or the neighbourhood best value, and its closest neighbour, the only other particle in the subswarm and situated on the boundary of the subswarm. While the subswarms are updated, particles from the main swarm are updated at the same time, and will move towards better positions. Should such a particle move into a region already occupied by a subswarm, the particle will be absorbed into that subswarm. A particle i is absorbed into a subswarm S_j when

$$\|\mathbf{x}_i - \hat{\mathbf{y}}_{S_j}\| \leq R_j \tag{4.15}$$

where $\hat{\mathbf{y}}_{S_j}$ is the neighbourhood best position of subswarm S_j. R_j signifies the radius of subswarm S_j, and is defined as

$$R_j = \max\{\|\hat{\mathbf{y}}_{S_j} - \mathbf{x}_{S_{j,i}}\|\} \tag{4.16}$$

where $\mathbf{x}_{S_{j,i}}$ is a particle in subswarm S_j.

Existing subswarms may also attempt to optimize the same solution, given those subswarms have been created from particles near to each other. These subswarms are merged when the hyper-space defined by their particle positions and radii intersect in the search space, creating a new larger swarm. Subswarms S_{j_1} and S_{j_2} *intersect* when

$$\|\hat{\mathbf{y}}_{S_{j,1}} - \hat{\mathbf{y}}_{S_{j,2}}\| < (R_{S_{j,1}} + R_{S_{j,2}}) \tag{4.17}$$

In the case where the radii of both subswarms are zero, the subswarms are merged if

$$\|\hat{\mathbf{y}}_{S_{j,1}} - \hat{\mathbf{y}}_{S_{j,2}}\| < \mu \tag{4.18}$$

where μ is a small number, normalized to the interval [0,1] in order to make the merging process more generic.

The NichePSO algorithm is summarized in Algorithm 4 [13] [14] [16].

Algorithm 4 NichePSO Algorithm

```
Create and initialize a n-dimensional main swarm, S;
repeat
    Train the main swarm, S, for one iteration using the
      cognition-only model;
    Update the fitness of each main swarm particle, x_{S,i};
    for each subswarm S_j do
        Train subswarm particles, x_{S_j,i}, using a full model PSO
          which incorporates the social component;
        Update each particle's fitness;
        Update the swarm radius R_{S_j};
    end
    if a merging condition is satisfied then
        Merge the corresponding subswarms;
    end
    Allow subswarms to absorb any particles from the main swarm
      that moved into the subswarm;
    if a particle in the main swarm converged then
        Create a new subswarm with this particle and its closest
          neighbor;
    end
until stopping condition is true;
```

NichePSO was tested on a select group of benchmark functions. Four one-dimensional functions were investigated in the range $x \in [0, 1]$, where each function had 5 optima with differing positions and fitness of the optima. The two-dimensional modified Himmelblau function was tested in the region $x_1, x_2 \in [-5, 5]$, where the function has four equal optima [13] [14] [16]. Experimental results showed that NichePSO successfully located and maintained multiple optimal solutions. However, performance of the algorithm does depend on tunable parameters μ and δ. Merging of subswarms is controlled by the value of μ [14] [16]. A too small value of μ will impede the merging of subswarms while subswarms that are supposed to converge on separate optima, will merge when the threshold approaches the interniche distance. Therefore, if μ is too large, not all optima will be located. According to Brits *et al.* [16], μ should not be greater than the smallest interniche distance for optimal NichePSO performance.

Brits *et al.* [14] [16] investigated the sensitivity to changes in δ as well. δ is a parameter used as an indication of whether a particle could be identified as a candidate solution. Results showed that NichePSO is not dependent on a finely tuned δ. Smaller δ values effected a slight

increase in the number of fitness function evaluations, but did not influence the number of optima that was located.

Scalability of the NichePSO

The NichePSO algorithm discussed in the previous section showed good results when tested on one and two-dimensional multimodal functions. Brits *et al.* also investigated NichePSO's ability to scale to massively multimodal functions [14] [15] [16]. NichePSO was tested on the Griewank and Rastrigin functions. Both these functions have a single global minimum and local optima at regular intervals along each dimension. For a specific dimension size, the number of optima increases exponentially as the number of dimensions increases. With a corresponding increase in swarm size, NichePSO performed consistently with slight degradation as the number of dimensions increased [15].

4.3.4 The species-based PSO

The notion of different species existing together in a population was the inspiration behind a PSO for solving multimodal optimization problems [57]. In Biology a species can be described as a *group of actually or potentially interbreeding individuals who are reproductively isolated from other such groups.* The concept of species that evolve to fill a role referred to as an ecological niche, has been used in Goldberg and Richardson's fitness sharing genetic algorithm [40], the sequential niche technique of Beasley *et al.* [4], and the species conserving genetic algorithm (SCGA) [56].

Li [57] proposed the species-based PSO (SPSO) that incorporates the idea of classifying the population into groups or species. The best-fit individual, or particle in the case of PSO, in a species is referred to as the species seed. The definition of a species also includes a parameter, r_s, referred to as the species radius which is the Euclidian distance from the species seed to the boundary of the species. All particles that fall within the r_s distance from the species seed, are part of that species. Figure 4.2 illustrates how particles are assigned to species by using a species radius.

The algorithm determining the species seeds for the SCGA, introduced by Li *et al.* [56], is adopted by the species-based PSO. In PSO terminology the neighbourhood best (*lbest*), the best-fit particle in a neighbourhood, is similar to the species seed for that species. Species seeds are determined by first sorting all particles $\mathbf{x}_i$ in decreasing order of fitness. Initially the set of

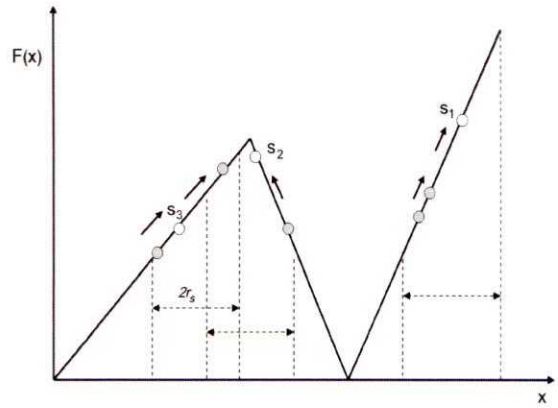


Figure 4.2: Determining the species seeds from the population at each iteration [57]

species seeds S is empty. Particles are checked in turn from best-fit to least-fit against species seeds found so far. The best-fit particle in the swarm is the first species seed and is added to S. Particles falling within a pre-determined radius of that seed, is part of that species. If a particle is found that falls outside the radius of that seed, the particle is added to S as a new seed. As S is built, subsequent particles are checked against all the species seeds in S. Only particles that do not fall within the niche radii of any of the seeds in S, are added to S as new seeds. The set of species seeds is complete once all particles have been checked.

Once the species seeds have been identified from the population, particle positions are updated using the species seeds as the neighbourhood best positions of each corresponding species. Each iteration consists of determining the set of species seeds and adjusting all particle positions once. The process is repeated for a number of iterations enabling the particles to converge on several optima in parallel.

Algorithm 5 formalizes the process to determine species seeds. The species-based PSO that incorporates the process to determine species seeds is given in Algorithm 6.

Algorithm 5 Determining the species seeds

Input: L_{sorted} - a list containing all particles $\mathbf{x}_i$ sorted in decreasing order of fitness if maximization is assumed
Output: S - a set containing particles s_j identified as species seeds

```
begin
    S = φ;
    while not reaching the end of L_sorted do
        found ← FALSE;
        for all s_j ∈ S do
            if distance d(s_j, x_i) ≤ r_s then
                found ← TRUE;
                break;
            end
        end
        if (not found) then
            let S ← S ∪ {x_i}
        end
    end
end
```

Algorithm 6 The species-based PSO

```
begin
    Create an initial population with randomly generated particles;
    repeat
        Evaluate all particle individuals in the population;
        Sort all particles in descending order of their fitness values
          i.e., from the best-fit to least-fit ones;
        Determine the species seed using Algorithm 5 for the current population;
        Assign particles identified in each species to the corresponding species seed;
        Adjust particle positions according to the PSO update equations;
    until termination condition is met;
end
```

The species-based PSO was tested on several benchmark functions [57]. The test functions used were those suggested by Beasley *et al.* [4], namely four one-dimensional functions, as the two-dimensional Himmelblau function, and the Rastrigin function in different dimensions. Tests showed that SPSO can find all the optima for all these functions in one and two dimensions reliably and with good accuracy [57]. However, one drawback of this approach is the size of

the species radius, r_s. Each objective function requires a unique species radius that has to be set in advance. Thus prior knowledge of the objective function is implied. For an unknown function landscape the algorithm can be run with varying values of the species radius. The species radius yielding the best performance is then chosen for that specific problem. Using this approach, the algorithm could still be described as robust and accurate.

However, in all these functions distances between niches do not differ much. Niche shapes are relatively symmetrical. If the shapes and sizes of niches differ a lot, niches require different niche radii. In addition, asymmetrical niche shapes require a different technique to determine the boundaries of the species. Thus the reliability of a single niche or species radius may be challenged when the function landscape is more convoluted and different niches have different niche radii.

4.3.5 The adaptive niching PSO

A niching method that removes the need to specify the niche radius in advance was proposed by Bird and Li [5]. Niching parameters are determined adaptively during a run. The method is called the *adaptive niching PSO* (ANPSO). Initially, the algorithm calculates the average distance, r, between each particle and its closest neighbour. This value is used to determine the formation of niches. ANPSO uses an undirected graph, g, with particles as nodes to keep track of the minimum distance between particles over a number of steps. At each iteration, an edge is added to g between every pair of particles that have been closer than r to one another during the last 2 steps. Niches are formed from the connected subgraphs of g, while all unconnected particles remain outside any niches. Particles can also be added to existing niches during a run. Thus the neighbourhood topology is redefined at every step. Once niches are determined, a GBEST topology is used to update particles in each niche, while a Von Neumann topology is used to update particles that have not yet been assigned to a niche. Therefore, particles that have formed a niche will tend to perform a local search around an optimum, while particles outside any niche will continue searching the whole problem space.

The ANPSO removes the need to specify a niche radius in advance, as it is calculated from the distances between particles. To prevent too many particles from converging on the same optimum, a limit is placed on the number of particles allowed in a single niche. Owing to the calculation of distances between particles, the algorithm is computationally more expensive than most other niching techniques, but, according to Bird and Li [5], the cost is offset by the time taken to tune the niche size parameter in other niching algorithms. The algorithm

performed well on a number of benchmark functions.

Algorithm 7 The adaptive niching PSO

```
begin
    Create an initial population with N randomly generated particles;
    for all particles x_i do
        Calculate distances
            d_i = min_{j≠i} ||x_i − x_j||;
        Calculate average distance
            r = (Σ_{i=1}^{N} d_i) / N;
    end
    Create an undirected graph g containing a node for each particle x_i;
    repeat
        for all particles x_i do
            if d_i < r during last 2 steps then
                Add and edge to g between x_i and nearest particle;
                if one of the particles is already assigned to a niche then
                    Add the other particle to the niche;
                else
                    Create a niche from x_i and nearest particle;
                end
            end
        end
        Update particles in each niche using GBEST;
        Update particles outside any niche using Von Neumann topology;
    until termination condition is met;
end
```

4.3.6 The fitness Euclidian-distance ratio-based PSO

Another effort to address the difficulty in pre-specifying parameters used for estimating how far apart optima in multimodal functions are, has been proposed by Li [58]. The fitness Euclidian-distance ratio-based PSO (FER-PSO) encourages the survival of fitter and closer particles. The FER-PSO was inspired by the fitness distance ratio-based PSO (FDR-PSO) of Veeramacheneni *et al.* [102]. The FDR-PSO was originally designed to locate a single global optimum. A new term based on FDR values was added to the canonical PSO velocity update equation. A canonical PSO modifies the velocity of each particle iteratively by its personal best position

and the position of the best particle from the entire swarm. As a result, each particle searches around a region defined by these two positions, and the influence from other fitter and nearer particles is increased. However, the concept of attracting each particle towards a fitter and closer point in the neighbourhood can effectively be applied to multimodal optimization a well.

FER-PSO uses two swarms, a *memory-swarm* formed by personal bests of particles, $\mathbf{y}_i$, and an *explorer-swarm* consisting of the current positions of particles, $\mathbf{x}_i$. The latter continues to explore the search space while the former acts as a stable repository where the best positions found so far are retained. Each point in the memory-swarm can be improved by moving towards its *fittest-and-closest* point. As the FER-PSO is designed for multimodal optimization, each particle is attracted towards a *fittest-and-closest* neighbourhood point, identified by computing the fitness Euclidian-distance ratio of that particle and all other particles in the swarm. The effect of such a strategy is that particles would form niches naturally around multiple optima, given that there are sufficient numbers of particles.

The fitness Euclidian-distance ratio is calculated as follows:

$$FER_{(j,i)} = \alpha \cdot \frac{f(\mathbf{y}_j) - f(\mathbf{y}_i)}{\|\mathbf{y}_j - \mathbf{y}_i\|} \tag{4.19}$$

where α is a scaling factor to ensure that neither the fitness nor the Euclidian distance becomes too dominated over one another.

For each particle with personal best $\mathbf{y}_i$, the maximum fitness Euclidian-distance ratio found will indicate the particle identified as the neighbourhood best $\hat{\mathbf{y}}$ to be used in the canonical PSO velocity update equation for particle i, namely

$$\mathbf{v}_i = \chi(\mathbf{v}_i + \mathbf{R}_1[0, \frac{\varphi_{max}}{2}] \otimes (\mathbf{y}_i - \mathbf{x}_i) + \mathbf{R}_2[0, \frac{\varphi_{max}}{2}] \otimes (\hat{\mathbf{y}} - \mathbf{x}_i)) \tag{4.20}$$

where $\mathbf{v}_i$ is the velocity of particle i, and χ is the constriction factor, used to prevent each particle from exploring too far away in the search space. $\mathbf{x}_i$ is the current position of particle i, $\mathbf{y}_i$ the personal best position of particle i, and $\hat{\mathbf{y}}$ is the best position found so far in the entire swarm. $\mathbf{R}_1[0, \frac{\varphi_{max}}{2}]$ and $\mathbf{R}_2[0, \frac{\varphi_{max}}{2}]$ are two separate functions, each returning a vector consisting of random values in the range $[0, \frac{\varphi_{max}}{2}]$, where φ_{max} is a positive constant. Point-wise vector multiplication of these functions with the difference between $\mathbf{y}_i$ and $\mathbf{x}_i$, as well as $\hat{\mathbf{y}}$ and $\mathbf{x}_i$ repectively, is indicated by $\otimes$.

Results showed that good performance were reported on some widely-used multimodal functions without the need to pre-specify niching parameters. However, the algorithm is computationally expensive due to the calculation of the fitness Euclidian-distance ratio for every pair of particles.

4.3.7 The waves of swarm particles algorithm

The waves of swarm particles (WoSP) algorithm, developed by Hendtlass [41], uses a technique that locates multiple optima by forcing the swarm to explore different promising regions of the problem space. Particles are organized into waves, each of which optimizes a single candidate solution. The practice of only evaluating a particle's performance at discrete intervals, is used to adjust particle behaviour in situations where the swarm is converging on an optimum.

To alter the behaviour of particles when they are settling close together, a short-range interaction between particles is introduced, namely a gravitational style attraction that becomes more effective when particles are near to one another. To acquire such an effect, the magnitude of the short-range force (SRF) of particle i towards particle j is set inversely proportional to some power p of the distance between the particles. This short-range force produces a velocity component, $\mathbf{v}_{i,j}$, that is represented by

$$\mathbf{v}_{i,j} = \frac{K}{d_{i,j}^{p}} \tag{4.21}$$

where $d_{i,j}$ is the distance between particles i and j, and K is a constant.

WoSP combines the short-range force with non-continuous or discrete evaluation to effect exploration of the search space. A particle's performance is evaluated at discrete time intervals. By the time of the next evaluation, particles may have passed each other and be at such a distance apart that the short-term attraction is too weak to bring them back together.

Because of the additional velocity component, $\mathbf{v}_{i,j}$, velocities decrease as the distance between particles increases. Particles will not be pulled back, but continue moving apart, exploring beyond their previous positions. Instead of converging on a single optimum, some of the neighbourhood particles are ejected with significant velocities, thus exploring other regions of the search space. Such particles are organized in a *wave*. In particles assigned to the wave, knowledge of the previous optimum is replaced by knowledge of the wave, that is, the best-fit particle of the wave becomes the neighbourhood best. Particles assigned to the wave are now attracted to the best particle in the wave. The wave converges on another optimum, starting the process again until all or most of the optima have been located.
Some aspects of the algorithm need to be clarified:

- Waves formed by ejected particles are numbered. Initially all particles belong to wave number zero. Every time a particle is ejected, it is promoted by having its wave number increased to that of the highest numbered wave. A new wave is created if necessary.

Particles are commonly ejected in pairs, and in that case a wave will have an initial size of two. To reduce the probability that particles are pulled back to the region they have left, a particle is required to move at least a user-specified distance (the *search scale*)away from the previous position before the particle may become part of a wave.

- A particle is considered as ejected when the ratio of the velocity component introduced by the short-range force to the other velocity components exceeds a user-specified value, called the *promotion factor*. As the speed of particles decrease when a wave is settling on an optimum, this ratio increases and the particle is ejected. Therefore particles are ejected when a wave is settling on an optimum.

The algorithm was tested on a number of benchmark problems. Results showed that the WoSP algorithm is able to escape from local sub-optima and continue to search for other optima.

4.4 Conclusion

This chapter described the concept of niching. Some niching strategies for genetic algorithms were reviewed, followed by a thorough discussion of a number of well-known niching algorithms developed in the field of particle swarm optimization. The algorithms are based on different principles, and various approaches are followed to locate all the solutions in functions with multiple optima.

Chapter 5 proposes the vector-based PSO, a new niching algorithm that purports to address a number of drawbacks of other niching algorithms.

Chapter 5

Vector-Based PSO

This chapter presents the development of a new niching strategy, the vector-based particle swarm optimizer (VBPSO). The principles underlying this approach are discussed and three consecutive versions of the algorithm are described in detail.

5.1 Introduction

Swarm intelligence algorithms such as particle swarm optimization (PSO) have been proved to be effective and robust for difficult optimization problems [28] [48] [49]. PSO was specifically designed to face the challenge of optimizing problems described by convoluted problem landscapes, often characterized by many sub-optimal or near-optimal solutions. The two-fold nature of the PSO algorithm containing a social and a cognitive component facilitates both the exploitation of regions with better fitness, as well as the exploration of the entire problem space. Thus the swarm is directed to where the best overall solution can be found. In both these aspects the velocity term, which is also a unique PSO feature, plays a significant role [50]. The concept of velocity is part of the metaphor of a swarm of particles flying through and exploring a hyperdimensional space. Particles move towards the target and overshoot it, but being pulled back by previous successes, oscillate around the target before eventually converging on the best solution [99]. If, in the process of exploration, better fitness is encountered, the entire swarm moves in a different direction away from a suboptimal solution. Therefore, given adequate exploration of the search space, the swarm will eventually converge on the overall

optimal position.

If a problem is such that the location of multiple optima is required, it is expected that different regions of the search space should be optimized in order to locate the different optima. Therefore, algorithms must be designed that counteract the effect of one of the essential characteristics of the particle swarm, namely redirecting the swarm away from a suboptimal solution to a solution with better fitness. Different strategies have been devised to neutralize this effect and maintain an optimal solution once it has been located, for example, by modification of the objective function [72], and using the cognition-only PSO at certain stages of the process [13].

This chapter describes a novel approach to niching where the power of the concepts underlying the original PSO is harnessed to induce niching. In addition, as in the case of all niching algorithms, the strategy should be such that multiple optima can be located and optimized with minimal prior knowledge of the landscape of the objective function and the number of optima to be found in a designated search area. While some constraints such as the boundaries of the search space and the number of initial particles obviously have to be present, another objective of this work was to limit other parameter settings to as few as possible, but still retain a robust and accurate algorithm that would yield good performances for a variety of problem landscapes.

The remainder of the chapter is organized as follows: Section 2 states the objectives of this chapter. Section 3 contains a description of vector properties and their relevance to niching, and section 4 explains how vector properties are used to determine niche boundaries. Section 5 gives a complete overview of the development of the new niching paradigm, namely vector-based particle swarm optimization (VBPSO).

5.2 Objectives of this chapter

Two steps are needed to locate multiple optimal solutions:

- Identify candidate solutions and demarcate the portion of the search space - called a niche - where an optimal solution may be found.
- Contain particles in the niche while optimizing the subswarm with the PSO method. Even one escaping particle can redirect the subswarm to a neighbouring niche and diminish the chances to find all the optimal solutions.

Niching algorithms described in the previous chapter used various strategies to find candidate solutions from which subswarms can be grown. However, a more critical issue is to determine the boundaries of the niche where the candidate solution or *neighbourhood best* was identified. Often a *niche radius* or *species radius* is pre-determined, implying prior knowledge of the objective function [39] [92]. One of the main objectives of developing a new niching algorithm is to find a niche radius for each niche. Such a niche radius will also be useful during the optimization phase to contain particles in the niche.

Another objective is to find a solution that is simple, but powerful, and where the principles driving PSO are utilized to its fullest extent. The idea of a pared-down, elegant solution to a problem has general appeal. As early as the 14th century William of Ockham, a leading figure in the golden age of Oxford scholasticism, formulated the so-called "principle of parsimony". This principle later became known as "Ockham's razor", a metaphor depicting cutting through complicated scholastic and theological arguments to reach the core of truth [42]. One of several ways in which this principle can be stated, is: "Entities are not to be multiplied beyond necessity". The principle of parsimony, originally formulated to guide the evaluation of symbolic reasoning systems, is frequently quoted in scientific disciplines. Ockham's razor has, among others, inspired the generalization of neural networks with as few as possible connections [96], and fitness evaluation based on a simplicity criterion [3]. In the case of particle swarm optimization, it can also be said that the principle of parsimony inspires a simple, cohesive solution with a limited number of parameter settings.

5.3 Using vector properties

Vectors are used extensively in PSO algorithms. In order to find a simple solution to a multimodal problem with a minimal number of pre-determined parameters, the possibility of using properties of the vectors forming part of the original PSO, was investigated. Concepts such as the tendency to move towards the personal best as well as the global best are implemented using vectors, as is the concept of a velocity associated with a particle. These vectors are manipulated to find a new position. If these vectors can also be manipulated to facilitate niching, the result will be an elegant as well as a powerful solution.

The original PSO updates the velocity vector associated with a particle as follows:

$$v_{i,j}(t+1) = w * v_{i,j}(t) + c_1 r_{1,j}(t)(y_{i,j}(t) - x_{i,j}(t)) + c_2 r_{2,j}(t)(\hat{y}_j(t) - x_{i,j}(t)) \tag{5.1}$$

where w is the inertia weight, c_1 and c_2 two positive acceleration constants, and $r_{1,j}$ and $r_{2,j}$ random values between 0 and 1.

A new particle position is calculated by adding the new velocity vector to the position vector of the particle. Equation (5.1) indicates that the velocity vector is adjusted in the aggregate direction of the sum of the cognitive and social vectors. If these two vectors point roughly in the same direction, it can be assumed that the particle's position will be adjusted towards a position with better fitness, the weighted average of the social and cognitive vectors. However, if the two vectors are pointing roughly in opposite directions, two outcomes might be expected: If only the social vector is considered, the position of the particle will most probably be adjusted nearer to that position. However, if only the cognitive vector is considered, adjustment towards a fitter position will, in most cases, mean that the particle moves away from the particle's current position, possibly in the direction of a different candidate solution. In other words, the particle follows a hill-climbing process. Figure 5.1 illustrates this scenario using a simple one-dimensional function.

When identifying niches, this knowledge can be used to identify particles that are not in the niche surrounding the current neighbourhood best position. Not all particles where both vectors point in the same direction will, of course, be moving towards the current best position, as there may be other candidate solutions between those particles and the current neighbourhood best.

Consider the inverted one-dimensional Rastrigin function,

$$F(x) = -(x^2 - 10\cos(2\pi x) + 10) \tag{5.2}$$

where $x \in [-1.5, 1.5]$. Figure 5.1 illustrates the function with three maxima in this range. Let the initial best position be at $x = 0$. The neighbourhood comprises the entire search space, thus this position is known as $\hat{y}$ or *gbest*. Assume that a number of particles are distributed along the x-axis. Each particle has an associated personal best position (y_i or $pbest_i$) where the fitness will be better than at the original particle position. Positions of two particles P_1 and P_2 are shown with vectors pointing in the direction of the personal best position as well as towards the global best position. If a particle is in a region where particles are expected to converge on the current neighbourhood best position, vectors towards the particle's personal best position as well as the neighbourhood best position point in the same direction. If the vectors point in opposite directions, it means that the position of the particle is in a region where it is not expected to converge on the current neighbourhood best position.

From the above discussion it can be deduced that the direction of the velocity vectors used

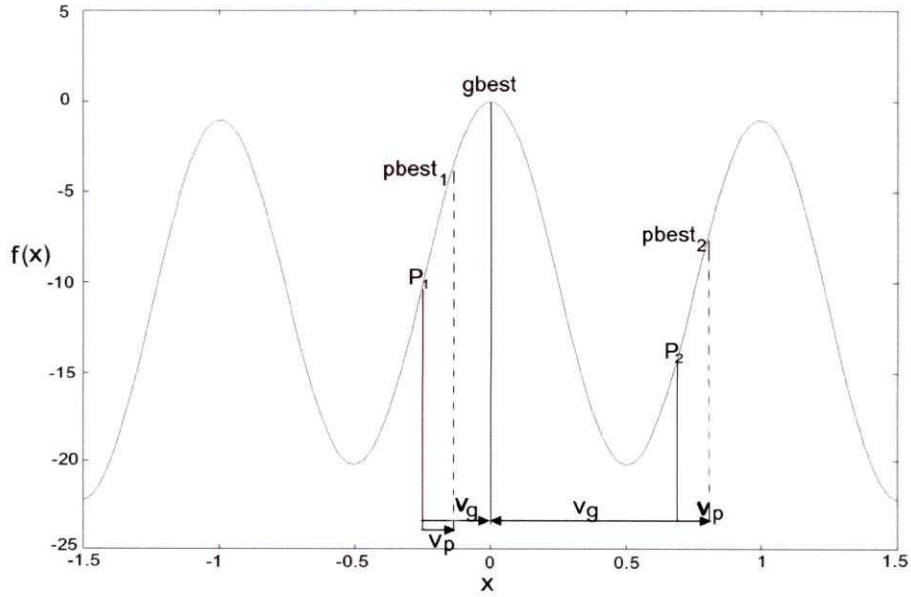


Figure 5.1: The inverted one-dimensional Rastrigin function, showing particles with associated vectors

in particle swarm optimization can indicate whether a particle will converge on an optimum or not. A method was required that combines the vectors pointing to the personal best position and the current neighbourhood best position of a particle respectively, resulting in a value indicating the inclination of the particle to move towards the current neighbourhood best position or not. The vector dot product provided a means to determine if the social and cognitive vectors point roughly in opposite directions or not. The dot product of two vectors is defined as follows [31]:

Let $\mathbf{a} = a_1\mathbf{i} + a_2\mathbf{j} + a_3\mathbf{k}$ and $\mathbf{b} = b_1\mathbf{i} + b_2\mathbf{j} + b_3\mathbf{k}$ be two vectors. The dot product (or scalar product or inner product) of $\mathbf{a}$ and $\mathbf{b}$ is the number, $\mathbf{a} \cdot \mathbf{b}$, defined by

$$\mathbf{a} \cdot \mathbf{b} = a_1b_1 + a_2b_2 + a_3b_3$$

The **angle** between two nonzero vectors, $\mathbf{a}$ and $\mathbf{b}$, is defined to be the angle, θ, where $\theta \in [0, \pi]$. The relationship between the dot product and the angle between two vectors is described by:

$$\mathbf{a} \cdot \mathbf{b} = \|\mathbf{a}\|\|\mathbf{b}\| \cos\theta$$

The value of $\cos\theta$ will be positive in the first quadrant where $\theta \in [0, \pi/2]$ and negative in the second quadrant where $\theta \in [\pi/2, \pi]$. Therefore it can be surmised that the dot product of two vectors will be positive if they point roughly in the same direction; that is, with an angle of less than 90° between them. If the vectors point roughly in opposite directions, that is, with an angle greater than 90° and less than 180° between them, the dot product will be negative. In the case of one-dimensional functions, the angle between the vectors will either be 0° or 180°.

Consider again Figure 5.1 where the best position in the entire search space is indicated by *gbest*. P_1 and P_2 are the positions of two particles distributed along the x-axis from $x = -1.5$ to $x = 1.5$. A vector points from each particle to the associated personal best position, *pbest*, situated at a position with higher fitness in the direction of the nearest suboptimal position. The sizes of these vectors are random values between 0 and a small problem-dependent value. For the purpose of illustration, assume that the size of all these vectors is 0.1.

To illustrate how dot products of particles change when situated along a range of particle positions, Table 5.1 was set up. $\mathbf{v}_p$ indicates the position vector from the particle at position x to its associated personal best position, *pbest*. $\mathbf{v}_g$ indicates the position vector from the particle at position x to the current neighbourhood best position, *gbest*. $\mathbf{v}_p \cdot \mathbf{v}_g$ is the vector dot product of these vectors. Figure 5.2 shows the regions in the search space of the inverted one-dimensional Rastrigin function where the dot products of the vectors pointing to their associated *pbest* and *gbest* positions, are positive or negative.

As a second example of the regions where positive and negative dot products of position vectors towards *pbest* and *gbest* occur, the following one-dimensional function is used:

$$F(x) = \sin^6(5\pi x) \tag{5.3}$$

For $0 \leq x \leq 0.4$, the function described by equation (5.3) has two peaks. Let *gbest* be at position $x = 0.1$. The value of $|p - x|$ is set to 0.01 except in the troughs of the function. To illustrate changes in the values of dot products in specific regions, Table 5.2 lists the dot products of particles along a range of positions. Figure 5.3 shows the regions in the search space of the function where the dot products of the vectors pointing to *pbest* and *gbest* are positive or negative.

In both these examples, note that particles in the region surrounding the current *gbest* position, yield positive dot products. At the niche boundary the dot products change to negative. However, there are other regions with positive dot products, for example in the region surrounding an adjacent niche, but not facing the current *gbest* position. These observations

Table 5.1: Dot products of a range of particle positions for the inverted one-dimensional Rastrigin function

x	$f(x)$	v_p	v_g	$v_p \cdot v_g$	x	$f(x)$	v_p	v_g	$v_p \cdot v_g$
-1.5	-22.25	0.1	1.5	0.15	0.05	-0.4919	-0.1	-0.05	0.005
-1.45	-21.6131	0.1	1.45	0.145	0.1	-1.9198	-0.1	-0.1	0.01
-1.4	-20.0502	0.1	1.4	0.14	0.15	-4.1446	-0.1	-0.15	0.015
-1.35	-17.7004	0.1	1.35	0.135	0.2	-6.9498	-0.1	-0.2	0.02
-1.3	-14.7802	0.1	1.3	0.13	0.25	-10.0625	-0.1	-0.25	0.025
-1.25	-11.5625	0.1	1.25	0.125	0.3	-13.1802	-0.1	-0.3	0.03
-1.2	-8.3498	0.1	1.2	0.12	0.35	-16.0004	-0.1	-0.35	0.035
-1.15	-5.4446	0.1	1.15	0.115	0.4	-18.2502	-0.1	-0.4	0.04
-1.1	-3.1198	0.1	1.1	0.11	0.45	-19.7131	-0.1	-0.45	0.045
-1.05	-1.5919	0.1	1.05	0.105	0.5	-20.25	0	-0.5	0
-1	-1	0	1	0	0.55	-19.8131	0.1	-0.55	-0.055
-0.95	-1.3919	-0.1	0.95	-0.095	0.6	-18.4502	0.1	-0.6	-0.06
-0.9	-2.7198	-0.1	0.9	-0.09	0.65	-16.3004	0.1	-0.65	-0.065
-0.85	-4.8446	-0.1	0.85	-0.085	0.7	-13.5802	0.1	-0.7	-0.07
-0.8	-7.5498	-0.1	0.8	-0.08	0.75	-10.5625	0.1	-0.75	-0.075
-0.75	-10.5625	-0.1	0.75	-0.075	0.8	-7.5498	0.1	-0.8	-0.08
-0.7	-13.5802	-0.1	0.7	-0.07	0.85	-4.8446	0.1	-0.85	-0.085
-0.65	-16.3004	-0.1	0.65	-0.065	0.9	-2.7198	0.1	-0.9	-0.09
-0.6	-18.4502	-0.1	0.6	-0.06	0.95	-1.3919	0.1	-0.95	-0.095
-0.55	-19.8131	-0.1	0.55	-0.055	1	-1	0	-1	0
-0.5	-20.25	0	0.5	0	1.05	-1.5919	-0.1	-1.05	0.105
-0.45	-19.7131	0.1	0.45	0.045	1.1	-3.1198	-0.1	-1.1	0.11
-0.4	-18.2502	0.1	0.4	0.04	1.15	-5.4446	-0.1	-1.15	0.115
-0.35	-16.0004	0.1	0.35	0.035	1.2	-8.3498	-0.1	-1.2	0.12
-0.3	-13.1802	0.1	0.3	0.03	1.25	-11.5625	-0.1	-1.25	0.125
-0.25	-10.0625	0.1	0.25	0.025	1.3	-14.7802	-0.1	-1.3	0.13
-0.2	-6.9498	0.1	0.2	0.02	1.35	-17.7004	-0.1	-1.35	0.135
-0.15	-4.1446	0.1	0.15	0.015	1.4	-20.0502	-0.1	-1.4	0.14
-0.1	-1.9198	0.1	0.1	0.01	1.45	-21.6131	-0.1	-1.45	0.145
-0.05	-0.4919	0.1	0.05	0.005	1.5	-22.25	0	-1.5	0
0	0	0	0	0					

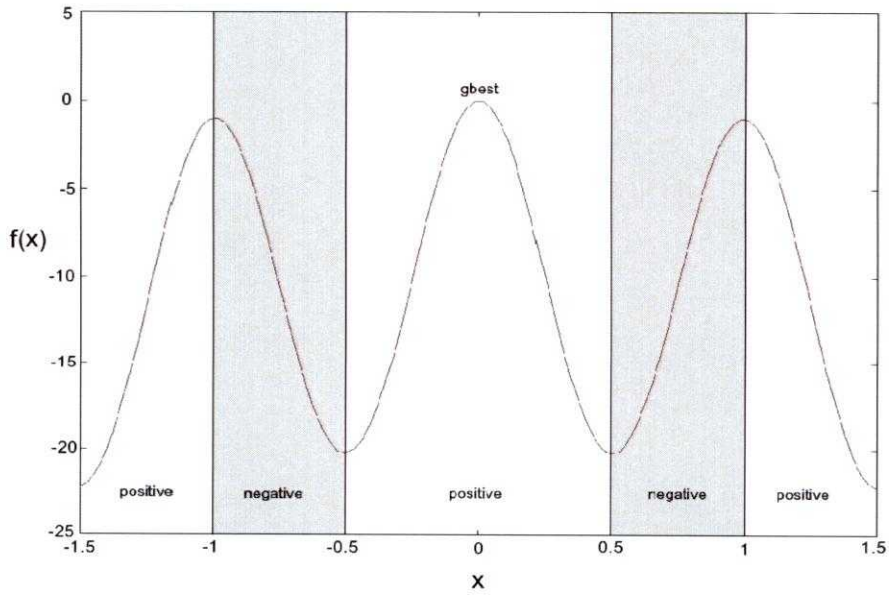


Figure 5.2: Regions of positive and negative dot products for the inverted one-dimensional Rastrigin function

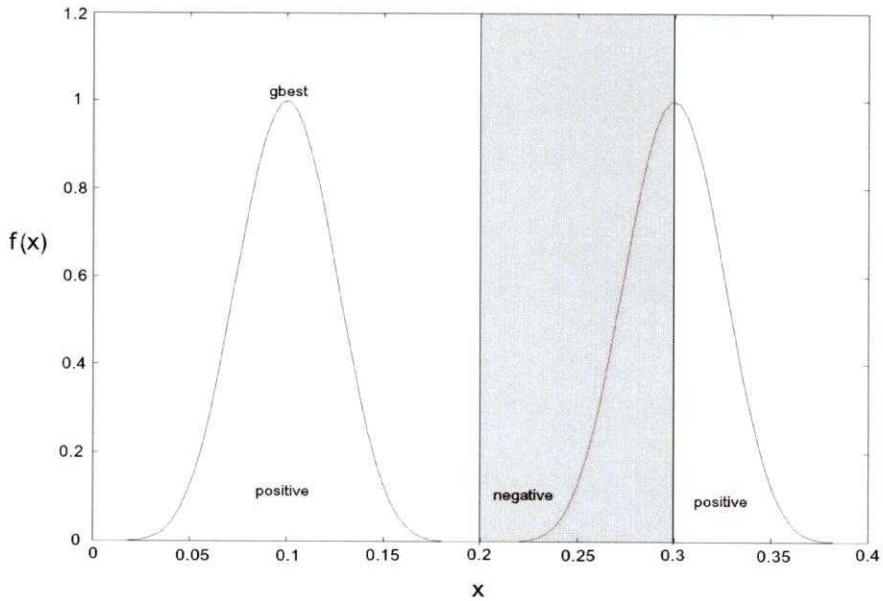


Figure 5.3: Regions of positive and negative dot products for equation (5.3)

Table 5.2: Dot products of a range of particle positions for equation (5.3)

x	$f(x)$	v_p	v_g	$v_p \cdot v_g$	x	$f(x)$	v_p	v_g	$v_p \cdot v_g$
0.01	1.4655E-05	0.01	0.09	0.0009	0.21	1.4655E-05	0.01	-0.11	-0.0011
0.02	0.0009	0.01	0.08	0.0008	0.22	0.0009	0.01	-0.12	-0.0012
0.03	0.0088	0.01	0.07	0.0007	0.23	0.0088	0.01	-0.13	-0.0013
0.04	0.0412	0.01	0.06	0.0006	0.24	0.0412	0.01	-0.14	-0.0014
0.05	0.125	0.01	0.05	0.0005	0.25	0.125	0.01	-0.15	-0.0015
0.06	0.2804	0.01	0.04	0.0004	0.26	0.2804	0.01	-0.16	-0.0016
0.07	0.5004	0.01	0.03	0.0003	0.27	0.5004	0.01	-0.17	-0.0017
0.08	0.7400	0.01	0.02	0.0002	0.28	0.7400	0.01	-0.18	-0.0018
0.09	0.9284	0.01	0.01	0.0001	0.29	0.9284	0.01	-0.19	-0.0019
0.1	1	0	0	0	0.3	1	0	-0.2	0
0.11	0.9284	-0.01	-0.01	0.0001	0.31	0.9284	-0.01	-0.21	0.0021
0.12	0.7400	-0.01	-0.02	0.0002	0.32	0.7400	-0.01	-0.22	0.0022
0.13	0.5003	-0.01	-0.03	0.0003	0.33	0.5004	-0.01	-0.23	0.0023
0.14	0.2804	-0.01	-0.04	0.0004	0.34	0.2804	-0.01	-0.24	0.0024
0.15	0.125	-0.01	-0.05	0.0005	0.35	0.125	-0.01	-0.25	0.0025
0.16	0.0412	-0.01	-0.06	0.0006	0.36	0.0412	-0.01	-0.26	0.0026
0.17	0.0088	-0.01	-0.07	0.0007	0.37	0.0088	-0.01	-0.27	0.0027
0.18	0.0009	-0.01	-0.08	0.0008	0.38	0.0009	-0.01	-0.28	0.0028
0.19	1.4655E-05	-0.01	-0.09	0.0009	0.39	1.4655E-05	-0.01	-0.29	0.0029
0.2	3.3817E-96	0	-0.1	0	0.4	2.1643E-94	0	-0.3	0

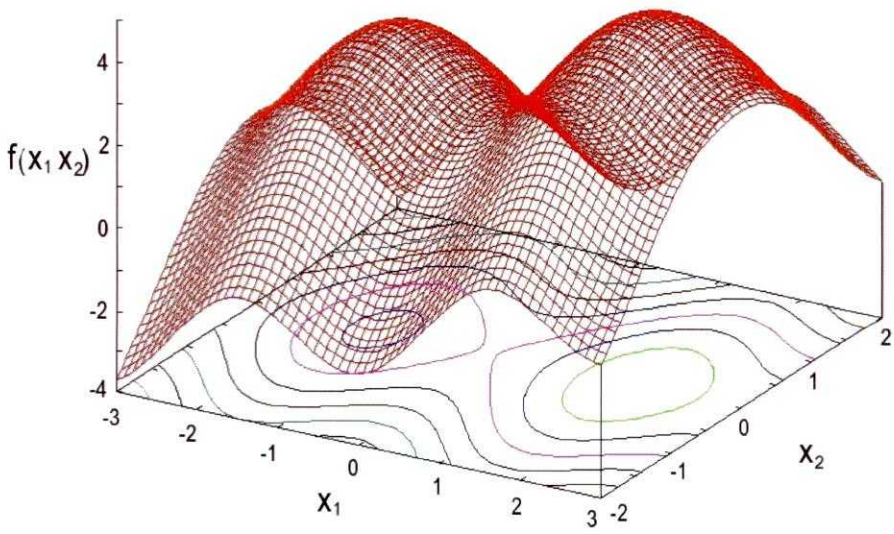


Figure 5.4: The two-dimensional Ursem F1 function

suggest that the niche boundaries of the niche surrounding the current *gbest* can be established, but that a sequential process must subsequently be followed where particles in the first niche is deactivated, a next neighbourhood best is identified, and new values for $\mathbf{v}_g$ and the dot product for each remaining particle is calculated.

To illustrate vector directions in a two-dimensional search space, consider the Ursem F1 function in two dimensions:

$$F(x_1, x_2) = sin(2x_1 - 0.5\pi) + 3cos(x_2) + 0.5x_1 \tag{5.4}$$

If the domain is infinite, this function produces a convoluted landscape of an infinite number of optima with differing heights. In the domain $x_1 \in [-2.5, 3]$ and $x_2 \in [-2, 2]$ two optima of different fitness values are present. Therefore the function has a global as well as one local optimum in the region described. The function landscape is illustrated in Figure 5.4.

Figure 5.5 shows a contour map of the two-dimensional Ursem F1 function. Two particles, P_1 and P_2, are depicted. The associated position vectors of each particle point towards the corresponding personal best position as well as *gbest*, a position at or near to the global optimum of the function.

Similar to the process followed for one-dimensional functions, vector operations can be used to calculate a position that roughly indicates the boundary between two niches. Let $\mathbf{v}_{p_1}$ be the

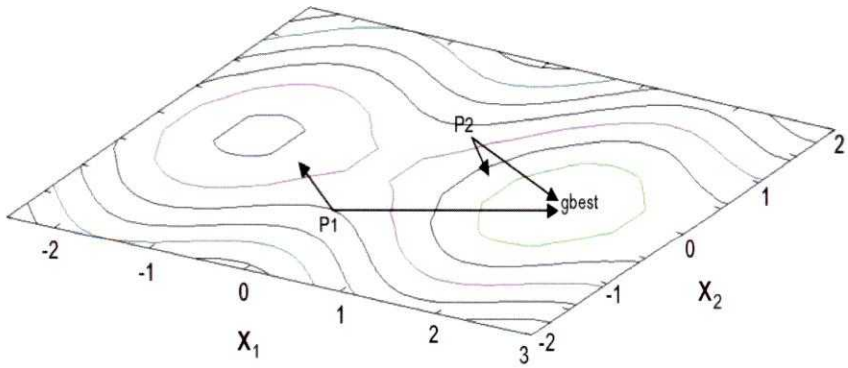


Figure 5.5: Contour map of the two-dimensional Ursem F1 function

position vector from particle P_1 to its associated personal best position, and $\mathbf{v}_{p_2}$ the velocity vector from particle P_2 to its associated personal best position. $\mathbf{v}_{g_1}$ and $\mathbf{v}_{g_2}$ are velocity vectors from P_1 and P_2 to the current neighbourhood best position, *gbest*. In this example the vector dot product $\mathbf{v}_{p_1} \cdot \mathbf{v}_{g_1}$ will be negative and the vector dot product $\mathbf{v}_{p_2} \cdot \mathbf{v}_{g_2}$ positive.

Section 5.4 describes a method using the vector dot product to establish niches and to determine their boundaries to a significant degree of certainty.

5.4 Identifying niches

As indicated earlier, the process of locating multiple optima in multimodal objective functions comprises identifying and demarcating regions in the problem space, called niches, where optima are likely to be found. Each niche is then optimized using PSO. During optimization, niches have to be maintained. As explained earlier, a particle searching outside the niche, may find a position with better fitness than the current neighbourhood best and divert an entire subswarm to converge on an adjacent niche. Therefore particles should be prevented from leaving the niche.

Typical multimodal functions have convoluted problem landscapes containing peaks or

optima with varying shapes and sizes where the niches surrounding the optima are not always symmetrical around the position of the current best-fit particle or neighbourhood best. The Euclidian distance from the neighbourhood best to the boundary of the niche is known as the *niche radius*. The concept of a niche radius is incorporated in a number of niching algorithms, in the case of genetic algorithms as well as particle swarm optimization [4] [13] [57]. Given the asymmetrical shape of many niches, niche boundaries can, at best, be approximated. Therefore a niche radius indicates an approximate region containing particles belonging to the niche. Moreover, many problem spaces contain niches of different sizes. If the same niche radius is used for all the niches in the problem space, regions demarcated by a niche radius may be smaller or larger than that of the actual niches. In addition, the objective function may be such that peaks in the function landscape are irregularly shaped. As a result, extra niches may be identified on the outskirts of the genuine niches, or genuine niches may be merged during the optimization process. Therefore it should be advantageous if a strategy can be devised where a specific niche radius is calculated for every niche in the search space.

Niching algorithms for PSO have devised different strategies to find niche radii. For example, the NichePSO algorithm initially sets each niche radius to the distance between a candidate solution and its closest neighbour, forming a subswarm with only two particles [13] [14]. During optimization, subswarms attempting to optimize the same solution merge when the hyper-space defined by their particle positions and radii intersect in the search space, creating a larger swarm with a larger niche radius. The original species-based PSO requires a niche radius to be defined by the user. This radius is the same for all niches [57]. The vector-based PSO implements a novel strategy where the vector dot-product is used to find the boundary between niches, and to compute a separate niche radius for every niche [82] [83] [84]. Therefore, one of the aspects according to which these algorithms are different from one another, is the strategy used to establish and maintain niche radii.

Finding the niche radius forms part of a larger sequential process of establishing niches, each with a neighbourhood best position from which the *niche radius*, the distance towards the boundary of the niche, is calculated. The process is summarized as follows:

Initialize the swarm: Similar to all population-based optimization algorithms, vector-based particle swarm optimizers start by initializing a swarm of particles at random positions throughout the search space. To identify candidate solutions and establish niches, personal best positions are required for each particle. An initial personal best value for each particle is established by finding a random position very near to the particle. If fitter

than the original particle, the new particle becomes the personal best. If not, the two positions are exchanged and the original particle becomes the personal best.

Identify the first candidate solution: The particle with the fittest personal best in the entire swarm becomes the first neighbourhood best. For each particle a dot product of the vector pointing to its own personal best and the vector pointing to the neighbourhood best is calculated.

Calculate the niche radius: The Euclidean distance between the current neighbourhood best particle and the closest particle with a negative dot product is now calculated. A negative dot product indicates that the fitness of the function at that point is increasing and that particle is part of an adjacent niche. This value approximates the niche radius. Only particles inside the radius having a positive dot product are identified as belonging to that niche. Niches are numbered as they are established, while the same number is assigned to particles constituting the subswarm that occupies the niche.

Establish remaining niches: The process of identifying candidate solutions and establishing niches is repeated by setting the next neighbourhood best to the fittest particle without a niche number, and calculating the corresponding niche radius. Several niches are identified in this way until all particles have been numbered. This process yields a number of neighbourhood best particles, each one being a candidate solution. A niche radius is associated with each niche, but the radii will differ depending on the size and shape of the niche.

Note that this strategy yields only a rough estimate of the niche boundaries, assuming that the function landscape is unknown and may be convoluted. However, niche radii are intrinsically inaccurate, as the niche surrounding a candidate solution is seldom completely circular or spherical. The particle representing the current neighbourhood best position is also not necessarily in the center of the niche. As the niche radius represents the shortest distance from the current neighbourhood best position to the niche boundary, a number of particles belonging to the niche will, in most cases, still be outside the boundary of the niche as defined by the niche radius. The strategy described here will group these particles together as a niche; the neighbourhood best position being the nearest particle to the genuine adjacent niche. Such niches are described as *false* or subsidiary niches. Therefore, the number of initial niches is usually larger than the number of genuine niches. False niches will yield duplicate solutions if

subswarms occupying the niches are optimized sequentially as the niches are identified, or be absorbed during the optimization phase when niches are optimized in parallel.

Some false niches may contain very few particles, often only one. Van den Bergh [98] [99] found that a single particle becomes stationary when the position of the particle is similar to the pbest and gbest positions. The velocity magnitude becomes 0 and the particle cannot move. This phenomenon is referred to as *stagnation* [99]. Therefore, subswarms consisting of single particles will not converge effectively. The vector-based PSO initializes particles with an initial value for the personal best that is different from the particle position, but for one-particle subswarms, gbest is still equal to the personal best. These particles have some initial velocity but may easily become stationary, unless such a subswarm is merged with a larger subswarm while it still has a velocity magnitude greater then zero. In such cases additional solutions that do not represent optima or suboptima, will result.

To address this issue the vector-based PSO creates additional particles at random positions in the vicinity of the neighbourhood best position of subswarms that consist of only one or two particles. Empirical tests conducted during the development of the family of vector-based algorithms have shown that, in most cases, subswarms consisting of three particles do converge effectively. A selection of these tests is presented in appendix A, section A.1. Results differ according to the landscape of a specific function and the initial distribution of particles in the search space. Therefore, the optimal size of these subswarms can, at best, be estimated. To retain the principle of parsimony, the introduction of another tunable parameter is not considered. A fixed size for these subswarms should not be too large, as an increase in the number of particles increases computational complexity. On the other hand, too small subswarms might not converge and produce solutions that do not represent good optima or suboptima. Results showed a number of extra solutions when subswarms consisted of one particle only, and a few extra solutions (for more complicated functions) when swarms sizes were extended to two particles. For subswarms consisting of three particles, no extra solutions were found for the functions tested, and it was assumed that the possibility of stagnation of particles is very low.

Once all niches have been identified, and some of the subswarms occupying the niches have been extended so that each niche contains at least three particles, these subswarms are optimized. Three different algorithms have been developed identifying niches in this manner. These are discussed in the next section.

5.5 Vector-based PSO algorithms

This section presents an overview of the development of three vector-based PSO strategies [82] [83] [84]. All three strategies use the method described in section 5.4 based on the principles discussed in section 5.3, to identify candidate solutions and calculate niche radii for the initial niches. The optimization process carried out to locate multiple optima is improved in each successive algorithm. The sequential VBPSO optimizes niches as they are identified. A number of duplicate niches are located for some optima. The parallel VBPSO incorporates a merging strategy to eliminate duplicate niches and reduce the computational complexity of the process. Niches are optimized in parallel and merged when it becomes clear that more than one subswarm converges on the same optimum. The enhanced parallel VBPSO adapts the updating process to contain particles in a niche during optimization, in order to prevent a subswarm being diverted to and merged with an adjacent subswarm.

5.5.1 The sequential vector-based PSO

The sequential VBPSO constitutes a first attempt at using the dot product to identify candidate solutions and find multiple optimal solutions for an optimization problem [82]. Niches are identified sequentially as described in section 5.4. The optimization process is incorporated in the overall sequential process, that is, subswarms are optimized as the niches are identified. Algorithm 8 presents a pseudo-code algorithm of the sequential VBPSO.
Some aspects of the algorithm are discussed in detail below:

Initialize the swarm: A specified number of particles is created at random positions throughout the search space. This algorithm stores particles as a linked list of particle objects. Provision is made for each particle to store a unique niche number, the position of the particle, its personal best position and the neighbourhood best position. Position vectors towards the personal best and neighbourhood best positions are stored, as well as the dot product of these vectors. The distance between a particle and the current neighbourhood best position is stored as the *radius*. For all particles, the *niche-id* is initially set to 0. When a particle is created, an initial personal best position is calculated. Personal best positions are required for all particles in order to calculate dot products that are used to identify niches. To find a personal best position, a random position is created in the vicinity of the particle position. For this purpose a parameter, ϵ, is introduced, which is a small value relative to the search space, acting as an upper bound to the distance

Algorithm 8 The sequential vector-based PSO

begin
  Initialize the swarm by creating N particles;
  Set niche identification number (*niche-id*) of each particle to 0;
  for *each particle* **do**
    Find a random position within the niche radius from the current neighbourhood best;
    The position with the best fitness is the personal best, $\mathbf{y}_i(t)$;
    The other position is $\mathbf{x}_i(t)$;
    Calculate the vector $\mathbf{v}_{pi}$, where
    $$\mathbf{v}_{pi}(t) = \mathbf{y}_i(t) - \mathbf{x}_i(t)$$
  end
  repeat
    Set $\hat{\mathbf{y}}(t)$ to $\mathbf{y}_i(t)$ with best fitness of particles where *niche-id* $= 0$;
    for *each particle in the swarm* **do**
      Calculate the vector $\mathbf{v}_{gi}$ where
      $$\mathbf{v}_{gi}(t) = \hat{\mathbf{y}}(t) - \mathbf{x}_i(t)$$
      Calculate the dot product δ_i:
      $$\delta_i = \mathbf{v}_{pi} \cdot \mathbf{v}_{gi}$$
      Set radius ρ_i to the distance between $\hat{\mathbf{y}}(t)$ and $\mathbf{x}_i(t)$
    end
    Set *niche radius* to distance between $\hat{\mathbf{y}}(t)$ and nearest particle with $\delta_i < 0$;
    for *each particle where* $\rho_i <$ *niche radius and* $\delta_i > 0$ **do**
      Set *niche-id* to next number;
    end
    if *particles in niche* < 3 **then**
      Create extra particles in niche so that it has at least 3 particles;
    end
    for *specified number of iterations* **do**
      for *each particle with current* niche-id **do**
        Update particle position $\mathbf{x}_i(t)$;
        Update $\mathbf{y}_i(t)$ and $\hat{\mathbf{y}}(t)$;
        Update vectors $\mathbf{v}_{pi}$ and $\mathbf{v}_{gi}$;
        Update dot product δ_i;
      end
    end
  until *no particles with* niche-id $=$ *0 remain*;
end

between the particle position and its personal best position. The position is created randomly within these bounds. If the fitness at the new position is better than that of the original position, the new position becomes the particle's personal best. If not, the values are exchanged so that the new position is the position of the particle and the original position its personal best.

Identify niches: Section 5.4 presented a general description of niche forming in principle. Algorithm 8 incorporates a formal description of the process. When the swarm is initialized, the *niche-id* of all particles is set to 0. The particle with the best fitness of its *personal best* position is identified as the current neighbourhood best, $\hat{\mathbf{y}}(t)$. For every particle in the entire swarm, the position vector from the particle's position to the current neighbourhood best position, $\mathbf{v}_{gi}$, is calculated, as well as the dot product, δ_i, of the vectors $\mathbf{v}_{gi}$ and $\mathbf{v}_{pi}$. For each particle a radius, the Euclidian distance from the particle's position to that of the current neighbourhood best, is also calculated. As explained in section 5.3, the position of a particle where δ_i has changed from positive to negative will be a rough indication of the niche boundary. Therefore, the niche radius is set to the Euclidian distance between the current neighbourhood best position and the position of the nearest particle with a negative dot product. Particles with positive dot products and radii smaller than the niche radius constitute a subswarm that can be optimized separately. As niches are identified, particles forming the subswarm in the niche are marked by setting *niche-id* to the niche number, starting with 1. While particles with *niche-id* $= 0$ remain, the process is repeated. The fittest particle with *niche-id* $= 0$ becomes the next neighbourhood best and particles are assigned the next number. When no particles with *niche-id* $= 0$ remain, all niches will be numbered, and *niche-id* indicates to which niche each particle belongs.

False niches: The shape of a niche in an unknown function landscape can not be assumed to be symmetrical around a position identified as the current neighbourhood best. Thus the *niche radius* only gives a rough estimate of the boundary of the niche. However, a number of particles belonging to the niche may still be situated outside the niche radius, especially if the niche has an irregular shape. In such cases extra or false niches form next to the niche where the true optimum will eventually be located. The particle identified as the neighbourhood best of the false niche will be the particle nearest to the adjacent niche containing the true optimum. Experimental results presented in [82] and section 6.3 confirm that a number of false niches are formed. Subswarms occupying these niches converge on optima in adjacent niches, giving rise to duplicate solutions.

Extending a subswarm: Some of the subswarms formed when niches are identified may contain very few particles. Often only one particle constitutes a subswarm and $\hat{\mathbf{y}}(t)$ will be equal to $\mathbf{y}_i(t)$. If the particle position is updated and a position is located where the fitness is better than at $\mathbf{y}_i(t)$, the positions of $\mathbf{x}_i(t)$, $\mathbf{y}_i(t)$ and $\hat{\mathbf{y}}(t)$ will be the same. Such a particle easily becomes stationary and will not converge, or converge very slowly due to the remaining momentum. False niches are especially prone to such conditions. To prevent these subswarms from becoming stationary, additional particles are added when there are less than a specific number of particles in a niche. The niche radius is an upper bound of the distance of these particles from the current neighbourhood best. While developing the sequential vector-based PSO, empirical observations showed that subswarms require at least 3 particles to prevent the particles from becoming stationary. Thus the sequential VBPSO extends subswarms with 1 or 2 particles to contain 3 particles.

Optimizing the subswarms: The sequential VBPSO optimizes subswarms as the niches they occupy, are identified. Therefore, the entire process is sequential. For a single iteration, the positions of all particles in a subswarm are updated as described in Chapter 3. For the sake of clarity, equations (5.5) and (5.6) used to update the velocity and then the particle position, are repeated:

$$\mathbf{v}_{i,j}(t+1) = w\mathbf{v}_{i,j}(t) + c_1 r_{1,j}(t)(y_{i,j}(t) - x_{i,j}(t)) + c_2 r_{2,j}(t)(\hat{\mathbf{y}}_j(t) - x_{i,j}(t)) \tag{5.5}$$

$$\mathbf{x}_i(t+1) = \mathbf{x}_i(t) + \mathbf{v}_i(t+1) \tag{5.6}$$

In the sequential VBPSO the acceleration constants, c_1 and c_2, are set to 1. That is, the maximum step size in the direction of the neighbourhood best position, referred to by c_1, is equal to c_2, the maximum step size in the direction of the personal best position. Thus the influence of the social component of the velocity update equation is equal to the influence of the cognitive component. Once niches are established, each subswarm that occupies a niche is expected to converge on one optimum. Exploration of remote areas of the search space is not required at this stage. In fact, a particle leaving a niche might divert an entire subswarm to a more promising area in the search space with the result that fewer suboptimal solutions are located. Therefore, the influence of the social component must not be too large. On the other hand, a too large cognitive component will slow down convergence. Therefore, for multimodal optimization, equal influence of the two components is appropriate. The inertia weight, w, is set to 0.8. Shi and Eberhart, who introduced the inertia weight [89], found that choosing $w \in [0.8, 1.2]$ results in faster

convergence. The choice of the settings for c_1, c_2 and w satisfies the theoretically derived relation for convergent particle trajectories as defined by Van den Bergh and Engelbrecht [100]:

$$w > \frac{1}{2}(c_1 + c_2) - 1 \tag{5.7}$$

Number of iterations: The number of iterations, that is, the number of times all particles in a subswarm are updated, can be set in advance, or the cycle can be repeated until a desired measure of accuracy is reached. However, owing to the extension of subswarms with too few particles, a specific number of iterations will not always yield the same number of function evaluations.

Duplicate solutions: The sequential VBPSO produces duplicate solutions as a result of the forming of false niches as described earlier. The algorithm contains no strategy to merge the subswarms contained in these niches. Thus, for some niches, more than one subswarm will converge on an optimum. In interpreting the results produced by the algorithm, the number of different optima will comprise the true number of niches.

The sequential VBPSO implements the strategy of using vector operations to identify niches in particle swarm optimization. However, the sequential nature of the algorithm as well as the formation of false niches produce duplicate solutions, as substantiated by experimental results presented in chapter 6. An improved algorithm is presented in the next section to eliminate duplicate solutions, thus enhancing the quality of the solutions.

5.5.2 The parallel vector-based PSO

An improved niching algorithm, the parallel VBPSO [83] is presented in this section. Yielding duplicate solutions is an undesirable characteristic of the sequential VBPSO. The parallel VBPSO incorporates a merging strategy to eliminate duplicate solutions. Niches are identified sequentially, similar to the process used by the sequential VBPSO presented in Algorithm 8. Niches are numbered and each particle is labeled with the number of the niche. Niches are then optimized in parallel, that is, all particles in all niches are updated during each iteration, while convergence is guided by each particle's personal best position as well as the neighbourhood best of the niche. The updating procedure is repeated for a specified number of iterations, interspersed with a merging procedure called repeatedly after a fixed number of the iterations have been completed. The effect that the size of these merging intervals may have on the

performance of the algorithm is discussed in detail below. Algorithm 9 presents a pseudo-code algorithm of the parallel VBPSO. Algorithm 10 presents a pseudo-code algorithm of the merging procedure.
Some aspects of the algorithm are discussed in more detail below:

Initialize swarm and identify niches: After initialization of the swarm, niches are identified similar to the process followed in the sequential vector-based algorithm described in section 5.5.1. A number of false or extra niches will also be identified. Instead of yielding duplicate solutions, the parallel vector-based PSO merges the subswarms contained in these niches while optimization takes place in parallel.

Store niche information: The parallel VBPSO does not optimize niches as they are identified. An appropriate data structure is used to store information about each numbered niche. The position and fitness of the neighbourhood best particle, the niche radius, the number of particles comprising the subswarm in the niche, as well as a boolean flag to indicate whether the neighbourhood best particle has been updated, are recorded.

Parallel optimization: The parallel VBPSO identifies niches sequentially, stores the information and then optimizes the subswarms occupying these niches, in parallel. For a single iteration, the positions of all particles in the entire swarm are updated. A single particle is updated as described in Chapter 3 and in section 5.5.1. As explained earlier, false niches that were formed around niches containing true optima, will converge towards the true niches.

The merging threshold: During the optimization process, subswarms moving towards one another in the process of converging on the same optimum, are merged once the distance between the subswarms becomes less than a certain threshold value. The distance between subswarms is measured as the Euclidian distance between current neighbourhood best positions of niches containing those subswarms. A new problem-dependent parameter, called the *granularity*, is introduced to facilitate niching. Section 6.5.2 presents the results of experiments to investigate the influence of granularity on the performance of the algorithm.

The merging procedure: The merging procedure of VBPSO is formalized in algorithm 10. A merging procedure for NichePSO has been described in chapter 4, where subswarms

Algorithm 9 The parallel vector-based PSO

begin
  Initialize the swarm by creating N particles;
  Set niche identification number (*niche-id*) of each particle to 0;
  Initialize the granularity, g;
  for *each particle* **do**
    Create a random position within the niche radius;
      The position with the best fitness is the personal best, $\mathbf{y}_i(t)$;
      The other position is $\mathbf{x}_i(t)$;
    Calculate the vector $\mathbf{v}_{pi}$, where
$$\mathbf{v}_{pi}(t) = \mathbf{y}_i(t) - \mathbf{x}_i(t)$$
  end
  repeat
    Set $\hat{\mathbf{y}}(t)$ to $\mathbf{y}_i(t)$ with best fitness of all particles with *niche-id* $= 0$;
    for *each particle in the swarm* **do**
      Calculate the vector $\mathbf{v}_{gi}$ where
$$\mathbf{v}_{gi}(t) = \hat{\mathbf{y}}(t) - \mathbf{x}_i(t)$$
      Calculate the dot product δ_i:
$$\delta_i = \mathbf{v}_{pi} \cdot \mathbf{v}_{gi}$$
      Set radius ρ_i to the distance between $\hat{\mathbf{y}}(t)$ and $\mathbf{x}_i(t)$;
    end
    Set *niche radius* to the distance between $\hat{\mathbf{y}}(t)$ and nearest particle with $\delta_i < 0$;
    for *each particle where* $\rho_i <$ *nicheradius and* $\delta_i > 0$ **do**
      Set *niche-id* to the next number;
    end
    if *particles in niche* < 3 **then**
      Create extra particles in niche so that it has at least 3 particles;
    end
    Store relevant niche information in an appropriate data structure;
  until *no particles with* niche-id $=$ *0 remain*;
  for m *times* **do**
    for k *times* **do**
      for *each particle* **do**
        Update particle position $\mathbf{x}_i(t)$;
        Update $\mathbf{y}_i(t)$ and $\hat{\mathbf{y}}(t)$;
        Update vectors $\mathbf{v}_{pi}$ and $\mathbf{v}_{gi}$;
        Update dot product δ_i;
        Update radius ρ_i;
      end
    end
    Merge niches using Algorithm 10;
  end
end

Algorithm 10 Merging procedure of the parallel VBPSO

```
begin
    for niche_i where i = 1 to n − 1 do
        for niche_j where j = i + 1 to n do
            Calculate Euclidian distance d_1 between ŷ_i(t) and ŷ_j(t);
            if d_1 < granularity then
                if ŷ_i(t) has better fitness than ŷ_j(t) then
                    for all particles in niche_j do
                        Calculate Euclidian distance d_2 to ŷ_i(t) of niche i;
                        if d_2 < granularity then
                            Set niche-id of particle to that of niche_j;
                            Update number of particles of both niches;
                        end
                    end
                else
                    for all particles in niche_i do
                        Calculate Euclidian distance d_2 to ŷ_j(t) of niche j;
                        if d_2 < granularity then
                            Set niche-id of particle to that of niche_i;
                            Update number of particles of both niches;
                        end
                    end
                end
            end
        end
    end
end
```

are merged when they intersect. Subswarms S_{j_1} and S_{j_2} *intersect* when

$$\|\hat{\mathbf{y}}_{j_1} - \hat{\mathbf{y}}_{j_2}\| < (R_{j_1} + R_{j_2}) \tag{5.8}$$

where $\hat{\mathbf{y}}_{j_1}$ and $\hat{\mathbf{y}}_{j_2}$ are the particles with the best fitness in subswarms S_1 and S_2 respectively and R_{j_1} and R_{j_2} are the radii of the respective subswarms. However, as a result of the definition of a niche radius, this approach can not be followed when merging subswarms in the parallel VBPSO. In NichePSO a niche radius is defined as the Euclidian distance from the neighbourhood best position to the boundary of a niche. Initially a subswarm consists of two particles, a candidate solution and its nearest neighbour while a number of particles remain part of the main swarm. Particles are absorbed from the main swarm and existing subswarms merged when the conditions in equation (5.8) apply. Niche radii remain small. The technique used by the family of vector-based PSO algorithms identifies niches by calculating niche boundaries as described in section 5.4. Niche radii are much larger than in the case of NichePSO and may even overlap. Therefore intersecting subswarms do not necessarily converge on the same optimum.

Merging in the parallel vector-based PSO will commence if the distance between subswarms becomes less than the *granularity*. The distance between these subswarms is measured as the distance between the neighbourhood best positions of the corresponding niches. All particles are not merged at the same time; the neighbourhood best particles are the last to merge. While the distance between two adjacent niches remains less than the *granularity* and becomes smaller, other individual particles are merged. Particles from the swarm where the fitness at the neighbourhood best position is less than that of the adjacent swarm, will be merged with the fitter swarm. An individual particle will only be merged if the distance between that particle and the neighbourhood best position of the other (fitter) subswarm becomes less than the *granularity*. Therefore the particle will be absorbed by the subswarm where the neighbourhood best has the best fitness. The process is implemented by changing the niche identification number, *niche-id*, resetting $\hat{\mathbf{y}}(t)$ and calculating new values for $\mathbf{v}_{gi}$ and δ_i. Simultaneously the number of particles is incremented in the fitter niche and decremented in the other niche. The particle at the neighbourhood best position of the less fit swarm is only absorbed once it becomes the only remaining particle in the niche.

Merging intervals: Optimization of the subswarms is interspersed by calls to the merging procedure. According to the merging procedure, particles from subswarms merge when

the distance between the neighbourhood best positions of two subswarms becomes smaller than the granularity. All particles do not merge at the same time, and several calls to the merging procedure may be required for an entire subswarm (occupying a false niche) to be absorbed by a fitter adjacent subswarm. The merging procedure can be called after each iteration of the update equation. However, to reduce computational complexity, the merging procedure can be called after a number of iterations of applying position updates. Appendix A, section A.2 presents empirical results using a selection of functions to illustrate the effect of merging intervals on the performance of the algorithm. Provided the merging procedure is called more than two or three times at intervals spread throughout the run, merging of all relevant subswarms can be expected. Exact interval sizes do not have any effect on the outcome of the algorithm. Algorithm 9 formalizes the process so that particle positions and corresponding data are updated k times, followed by one call to the merging procedure. These actions are repeated m times, giving a total of $m \times n$ iterations. The parallel VBPSO divides the total number of iterations into 10 equal intervals, i.e. if 500 iterations of the updating equations are executed, the merging procedure is called after every 50 iterations. This interval size was chosen in order to track the merging process, as presented in chapter 6.

The parallel VBPSO uses the same strategy as the sequential VBPSO to identify niches in a multimodal landscape. However, to eliminate duplicate solutions, subswarms moving towards the same optimum are merged while being optimized in parallel. Chapter 6 presents experimental results of implementations of the algorithm for a number of one- and two-dimensional functions. The next section presents an algorithm where the updating process is refined to improve results.

5.5.3 The enhanced parallel vector-based PSO

A refined version of the parallel vector-based PSO, the enhanced vector-based PSO, is presented in this section [84]. The parallel VBPSO performs well on benchmark functions where the optima are distributed regularly throughout the problem space and the fitness of these optima differ slightly or not at all. However, experimental results presented in section 6.4 show that the performance of the parallel VBPSO degrades if the problem space becomes more convoluted and the niche shapes and sizes differ considerably.

During optimization, particles may move outside the niche, but are pulled back unless the fitness of the new position is better than that of its current neighbourhood best position. In

such a situation the following problems may arise:

- Particles moving outside a niche may encounter a location where the fitness is better than that of the current neighbourhood best position and divert the entire subswarm in a different direction. Consequently, the algorithm will not locate all the optima in the search space. While this characteristic comprises one of the strengths of the original PSO, it can be seen as a weakness of using PSO for niching.
- Particles may move outside the search space and locate additional optima.
- If particles are not contained in the search space, a niche will contain fewer particles resulting in slower convergence and degradation of the accuracy of the solutions.

Algorithm 11 presents a pseudocode algorithm of the enhanced parallel vector-based PSO where the process of updating the particles in a subswarm is modified to inhibit the tendency of particles to move outside the niche.
Some aspects of the algorithm are discussed in more detail below:

Initialize swarm and identify niches: The swarm is initialized and niches identified similar to the sequential vector-based PSO and the parallel vector-based PSO.

Contain particles inside a niche: A strategy to contain particles inside a niche is included in the enhanced parallel vector-based PSO. During optimization, it is possible for a particle to leave the niche if the velocity is such that the particle overshoots the neighbourhood best position and moves out of the current niche. Such a particle may find a new position with better fitness than that of the neighbourhood best position of the original niche. In these cases the entire subswarm in the original niche will be diverted to, and merged with the new niche. To counteract this effect, each potential new position is investigated before updating that particle to determine whether it is still inside the niche. A potential new position, p, is found by adding velocity (calculated by equation (5.5)) to the previous position. If p has a better fitness than the previous position, the particle position can potentially be updated. To determine whether p is still inside the niche, a temporary particle, $\mathbf{x}_{temp}(t)$, with a corresponding personal best position is created at p. Similar to the strategy followed when particles were initially created, a random position, r, is found near p by calculating a random direction as well as a random distance between p and r. Since this distance has to be small, the granularity value forms the upper bound of the random distance. The fittest of these two positions will be the personal best position

Algorithm 11 The enhanced parallel vector-based PSO

```
begin
    Initialize the swarm by creating N particles;
    Set niche-id of each particle to 0 and initialize granularity, g;
    for each particle do
        Calculate personal best position, y_i(t), of particle at position x_i(t)
        Calculate the vector v_pi
    end
    repeat
        Set ŷ(t) to y_i(t) with best fitness of all particles with niche-id = 0;
        for each particle in the swarm do
            Calculate the vector v_gi;
            Calculate the dot product δ_i:
                δ_i = v_pi · v_gi
            Set radius ρ_i to the distance between ŷ(t) and x_i(t);
        end
        Set niche radius to distance between ŷ(t) and nearest particle with δ_i < 0;
        for each particle where ρ_i < nicheradius and δ_i > 0 do
            Set niche-id to next number;
        end
        if particles in niche < 3 then
            Create extra particles in niche so that it has at least 3 particles;
        end
        Store relevant niche information in an appropriate data structure;
    until no particles with niche-id = 0 remain;
    for m times do
        for k times do
            for each particle do
                Create temporary particle x_temp(t) and personal best y_temp(t);
                Calculate vectors v_ptemp, v_gtemp and dot product δ_temp;
                if δ_temp < 0 then
                    Retain original particle position and corresponding values;
                end
                else
                    Particle position x_i(t) = x_temp(t);
                    Update y_i(t), ŷ(t) and vectors v_pi and v_gi;
                    Update dot product δ_i and radius ρ_i;
                end
            end
        end
        Merge niches;
    end
end
```

of the particle, while the other position will be the particle position. In other words, if $fitness(p) < fitness(r)$, the temporary particle is created at position r with the personal best at position p, and vice versa. Both these positions are needed to calculate the vector $\mathbf{v}_{ptemp}$. To determine whether the particle position is inside the current niche, the vectors $\mathbf{v}_{ptemp}$ and $\mathbf{v}_{gtemp}$ as well as the dot product δ_{temp} between the two vectors, are calculated. A positive dot product means that the new particle position still forms part of the niche while a negative dot product indicates that the particle has moved outside the current niche and into an adjacent niche where the function landscape slopes away from the current niche. Although the possibility exists that a particle may move to a position far away from the current niche and still yield a positive dot product, a particle moving a small distance out of the current niche, will be identified in this manner.

Therefore, if $\delta_{temp} < 0$, the potential new position is discarded and the original particle position and the corresponding values retained. If $\delta_{temp} > 0$, the new particle position $\mathbf{x}_i(t) = \mathbf{x}_{temp}(t)$ and $\mathbf{y}_i(t)$, $\hat{\mathbf{y}}(t)$, vectors $\mathbf{v}_{pi}$ and $\mathbf{v}_{gi}$ as well as the dot product δ_i and radius ρ_i are calculated.

Merging niches: The enhanced parallel vector-based PSO also optimizes niches in parallel and merges subswarms contained in these niches. The merging strategy described by algorithm 10 is used for the enhanced parallel VBPSO as well.

The enhanced parallel VBPSO identifies niches and optimizes subswarms contained in those niches in parallel using the same basic strategy as the parallel VBPSO. The updating process is refined to prohibit particles from leaving a niche; a process incorporated to improve the performance of the parallel VBPSO.

Chapter 6 presents experimental results of implementations of the algorithm for a number of benchmark functions.

5.6 Conclusion

This chapter traced the development of the vector-based particle swarm optimizer (VBPSO) for niching through its various stages. The concept on which the strategy is based, was explained and motivated. Three algorithms were presented, namely the sequential, parallel and enhanced parallel vector-based PSO, where each algorithm represents an improvement on the previous version. Techniques used to implement the concept and refine the implementations in later

versions, were explained and motivated.

The next chapter provides an empirical evaluation of these algorithms, and compares the performance of the best VBPSO to that of other PSO-based niching algorithms.

Chapter 6

Vector-Based PSO Applied to Static Environments

This chapter presents experimental results obtained when the three vector-based PSO (VBPSO) algorithms are applied to static environments. Extensive test results obtained by applying the various versions of the algorithm to a number of benchmark functions are presented. The final version of the VBPSO is analysed by empirically testing the sensitivity of the algorithm to different parameter values, its ability to scale to highly multi-modal functions, and the relationship between the initial swarm size and the algorithm's performance. The chapter is concluded by presenting the results of a comparative study of the performance of three diverse PSO niching algorithms, namely the VBPSO, NichePSO [13] [14] and the species-based PSO [57]. These algorithms are applied to a number of two-dimensional functions with varying characteristics.

6.1 Introduction

Each of the vector-based PSO algorithms was evaluated on a number of benchmark functions. The sequential version, being the first attempt to use vector dot products to identify niches, was only evaluated on a small number of functions. This algorithm finds duplicate results if subswarms in more than one niche converge on the same optimum, and the number of duplicates increases with increasing complexity of benchmark functions. Therefore, the sequential

VBPSO was not investigated further. The two parallel versions, which are improvements of the sequential vector-based PSO, were evaluated on the same functions as the sequential version, as well as a number of additional functions. These functions were carefully selected to present a variety of landscapes, so that the performance of the parallel vector-based PSO algorithms could be evaluated in circumstances where the shapes, sizes and positions of the niches differed considerably.

The remainder of the chapter is organized as follows: Section 6.2 presents and discusses experimental and statistical procedures, gives an overview of a number of selected benchmark functions, and lists general and specific parameter settings for the experiments. Results of the sequential VBPSO are discussed in section 6.3, while section 6.4 presents results of the parallel and enhanced parallel VBPSO. These sections culminate in an identification of the best algorithm, i.e. the enhanced parallel VBPSO. A more in-depth analysis of this algorithm is done in section 6.5. A comparison of nichePSO, the species-based PSO, and the enhanced parallel VBPSO is given in section 6.6.

6.2 Experimental procedure

This section presents procedures that were followed to test the family of vector-based PSO algorithms on a number of benchmark functions. In section 6.2.1 various approaches and general settings are discussed. Section 6.2.2 presents descriptions of statistical procedures while the benchmark functions that were selected to test the VBPSO algorithms are described in section 6.2.3. Section 6.2.4 lists specific settings, namely initial swarm sizes and granularity, for the benchmark functions.

6.2.1 General procedures and settings

Two approaches used to initialize the swarm are described, and general parameter values are given and explained. For each algorithm, the functions were tested with a number of different settings. The settings for these experiments are given and explained below:

Initializing the swarm: To locate all the optima in the search space, it is essential that particles are distributed uniformly throughout the search space before niches are identified. Therefore, a good random number generator is needed to initialize the swarm. This section evaluates two approaches to initialize swarms, namely using Sobol sequences and the

random number generator supplied by the C++ compiler. The purpose of this section is to compare performances of the algorithm using these two approaches. This is the first study, to the author's knowledge, that evaluates the influence of random number generators on the performance of PSO niching methods.

Random numbers are commonly generated by means of a system-supplied number generator, for example, *rand*() in C++ and *Rand*() in Excel. Sequences of these numbers are called pseudo-random because the order of these numbers looks unpredictable. Such generators are almost always *linear congruential generators*, which generate a sequence of integers $I_1, I_2, I_3, \ldots$, each between 0 and $m-1$ (e.g. $RAND\_MAX$), by the recurrence relation

$$I_{j+1} = aI_j + c(mod\ m) \tag{6.1}$$

where m is the *modulus*, and a and c are positive integers called the *multiplier* and the *increment* respectively. A fixed sequence of numbers is generated and, depending on m, will eventually repeat itself. Successive random numbers are obtained by successive calls to the *rand*() function. In general, a sequence is initialized by a call to a function supplying a *seed* obtained from an (unpredictable) external feeder like the computer clock. The linear congruential method is very fast and easy to implement. However, it is not free of sequential correlation on successive calls, and many of the available pseudo-random number generators have flaws leading to low-order correlations [76].

Alternatively, quasi-random sequences, also called low-discrepancy sequences, can be used. The low discrepancy property is a measure of uniformity for the distribution of the points, ensuring no large gaps and no clustering of points in the d-dimensional hypercube. The Sobol sequence is one of the most popular quasi-random sequences because of its simplicity of implementation [11] [47]. An efficient implementation proposed by Antonov and Saleev [2] uses Gray codes, a binary numerical system where two successive values differ in only one bit.

The (binary-reflected) Gray code of an integer i is defined as

$$gray(i) = i \oplus \left\lfloor \frac{i}{2} \right\rfloor = (\ldots i_3 i_2 i_1)_2 \oplus (\ldots i_4 i_3 i_2)_2 \tag{6.2}$$

where $\oplus$ indicates the exclusive or operation.

Sobol points over a unit interval are generated using

$$x^n = g_1 v_1 \oplus g_2 v_2 \oplus \ldots \tag{6.3}$$

where $v_1, v_2, \ldots$ is a set of *direction numbers*. Each v_i is a binary fraction that can be written

$$v_i = m_i/2^i$$

where m_i is an odd integer with $m_i \in [0, 2^i]$.

To obtain v_i, a polynomial with coefficients from $[0, 1]$ is chosen and the coefficients are used to define a recurrence for calculating v_i. The recurrence may also be expressed in terms of m_i. To calculate points in one dimension, all m_i are set to 0.

A more detailed explanation of the implementation of Sobol's quasi-random sequence generator is presented in [11] and [47].

Initial swarm sizes: The number of particles that are created during the initialization phase depends on the test function as well as the size of the search space. Since the function landscape is unknown, an initial swarm size must be estimated. Table 6.9 in section 6.2.4 summarizes swarm sizes with which the functions have been initialized for all algorithms. These sizes showed to be sufficient to obtain good results.

Number of iterations: Separate independent experiments were conducted with the number of iterations set to 200 as well as 500, and for Sobol sequences as well as a system-supplied random number generator. Although function evaluations are counted, the number of function evaluations is not used as a stopping condition. Extending the size of some of the subswarms (discussed in section 5.5.1) may cause the swarm to be optimized with more than the number of initial particles. Therefore, the number of function evaluations may differ considerably from run to run. Using a set number of iterations ensures that each particle is updated the same number of times. Since a number of preliminary tests indicated that the algorithms located most optima after 200 iterations, the number of iterations was set to 200. To observe the effect of a larger number of iterations on performance, results were obtained with 500 iterations as well.

Parameter settings: For all three algorithms the random sequences used in the velocity update equation are scaled by constants $c_1, c_2 \in [0, 2]$. As explained in section 5.5.1, both these values were set to 1, and the inertia weight, w, was set to 0.8.

Granularity settings: For the parallel and enhanced parallel PSO algorithms merging takes place when the Euclidian distance between the neighbourhood best positions of two

niches becomes less than a merging threshold or *granularity*. The granularity is problem-dependent and has to be set in advance. According to results presented in section 6.5.2 niches merge effectively if the granularity is less than the smallest interniche distance. However, smaller values are preferable to prevent merging taking place too soon. Taking the function landscape and the expected number of optima into account, this value was set to 0.05 for the four one-dimensional functions. Granularity values for the other test functions are summarized in Table 6.9. Since the enhanced parallel VBPSO performed better than the other two algorithms, a study to determine the sensitivity of this algorithm to granularity values was undertaken. Results for various functions and the factors that influence the performance of the algorithm, are discussed in section 6.5.2.

Merging intervals: Niches are merged at intervals of 50 iterations. The intervals were chosen arbitrarily and, as discussed in section 5.5.2 and Appendix A.2, no evidence could be found that it has any effect on performance.

Number of runs: For each of the functions described in section 6.2.3, 50 independent runs were executed for each combination of number of iterations and initialization approach. Results of these experiments are presented in sections 6.3 and 6.4.

6.2.2 Statistical procedures

Since normality of data samples can not be assumed, this study made use of the non-parametric Mann-Whitney U test [87] to determine if there are statistically significant differences between different implementations or instances of the algorithms. For this purpose the Z-value, which is the normal approximation of the Mann-Whitney U test statistic, was computed to evaluate the null-hypothesis, $H_0 : \mu_a = \mu_b$, at significance levels of 0.05 and 0.01. If the null-hypothesis can not be accepted, the alternative hypothesis, $H_1 : \mu_a \neq \mu_b$, is accepted. Here a and b refer to different instances or implementations of the algorithms.

In this study, the Mann-Whitney U test is used to compare the following:

- Experiments using Sobol sequences to those using the built-in random number generator in Borland C++.
- The three versions of the vector-based PSO.

Outcomes of the hypothesis tests are presented in sections 6.3 and 6.4.

6.2.3 Test functions

A number of one-dimensional and two-dimensional functions was selected to test the different algorithms.

The following one-dimensional functions were introduced by Goldberg and Richardson to test the performance of niching genetic algorithms [40]. Beasley *et al.* used the functions to test the sequential niching technique for multimodal optimization [4], while Brits *et al.* used the same functions to test the performance of NichePSO, a parallel niching algorithm [13] [14]. These functions present different situations required to test an algorithm on one-dimensional functions, namely evenly-spaced and non-evenly spaced optima, as well as similar and differing heights of the peaks.

$$F1(x) = \sin^6(5\pi x) \tag{6.4}$$

$$F2(x) = \left(e^{-2\log(2)\cdot(\frac{x-0.1}{0.8})^2}\right) \cdot \sin^6(5\pi x) \tag{6.5}$$

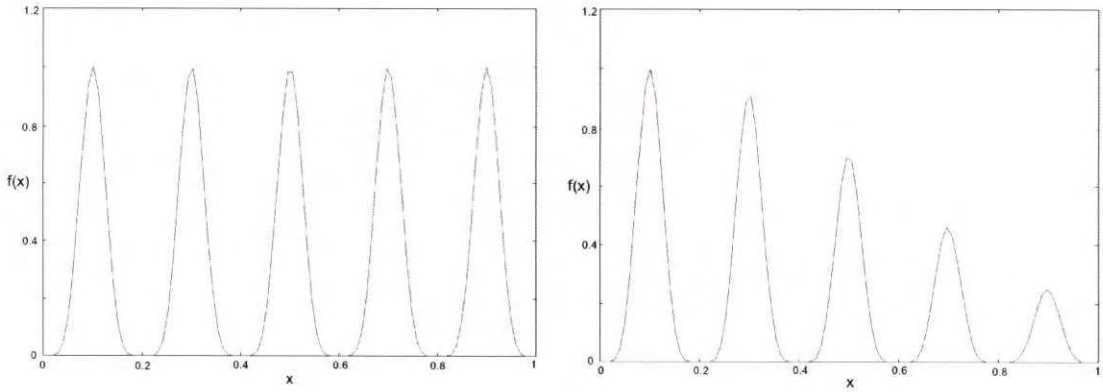
$$F3(x) = \sin^6(5\pi(x^{\frac{3}{4}} - 0.05)) \tag{6.6}$$

$$F4(x) = \left(e^{-2\log(2)\cdot(\frac{x-0.08}{0.854})^2}\right) \cdot \sin^6(5\pi(x^{\frac{3}{4}} - 0.05)) \tag{6.7}$$

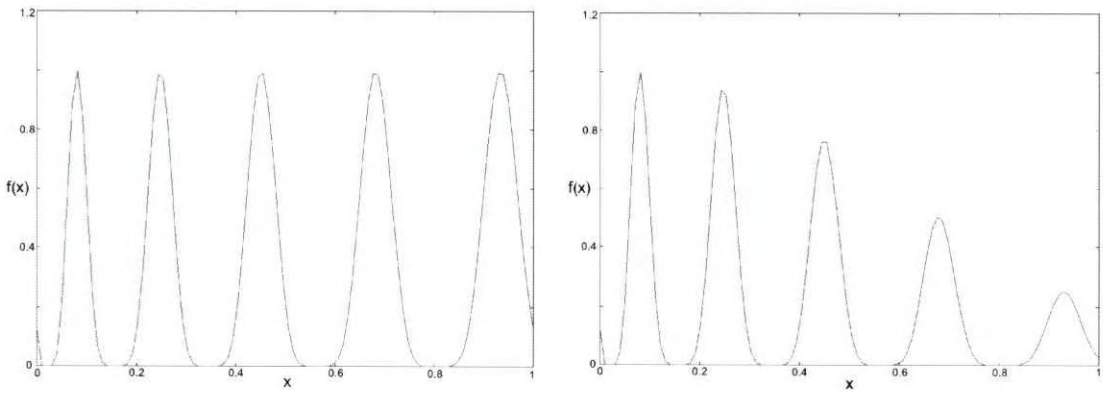
Functions $F1$ to $F4$ all have 5 maxima between 0 and 1.0, but the spacing between the maxima and the function values at the peaks differ. For each of these functions, maxima occur at positions as listed in Table 6.1. Function landscapes are illustrated in Figure 6.1.

Table 6.1: Positions of maxima of one-dimensional functions

$F1$	0.1	0.3	0.5	0.7	0.9
$F2$	0.1	0.3	0.5	0.7	0.9
$F3$	0.08	0.25	0.45	0.68	0.93
$F4$	0.08	0.25	0.45	0.68	0.93



(a) $F1$ (b) $F2$



(c) $F3$ (d) $F4$

Figure 6.1: One-dimensional functions

The following two-dimensional functions have been selected to test the vector-based PSO niching algorithms in a demarcated region of the search space. The functions that were selected include well-known benchmark functions that have been used to test other niching algorithms, as well as a number of functions specifically selected to illustrate the robustness of the vector-based algorithms when confronted with function landscapes where the shapes, sizes, and placing of the niches differ significantly.

The modified Himmelblau function: Figure 6.2 illustrates the modified Himmelblau function that is defined as

$$f(x_1, x_2) = 200 - (x_1^2 + x_2 - 11)^2 - (x_1 + x_2^2 - 7)^2 \tag{6.8}$$

The Himmelblau function, defined in two dimensions, is a popular choice for testing niching algorithms, as it has four well-defined optima with similar fitnesses. The positions of the four optima are listed in Table 6.2. Most niching algorithms, including genetic algorithms, yield good results for this function. The function is not repetitive, that is, the landscape where the optima occur, is not repeated elsewhere in the search space. The Himmelblau function was tested in the range $x_1, x_2 \in [-6, 6]$, where the optima occur.

Table 6.2: Optima of the Himmelblau function

Locations of optima		Fitness
x_1	x_2	$f(x_1, x_2)$
3	2	200
3.58	-1.85	200
-2.81	3.13	200
-3.78	-3.28	200

The Griewank function: The Griewank function is illustrated in Figure 6.3 in the range $x_1, x_2 \in [-5.0, 5.0]$ and in a larger range. The function is described by

$$f(\mathbf{x}) = -\left(\left(\frac{1}{4000}\sum_{i=1}^{n} x_i^2\right) - \left(\prod_{i=1}^{n} \cos(\frac{x_i}{\sqrt{i}})\right) + 1\right) \tag{6.9}$$

The function is also massively multimodal with one global optimum at $[0, 0]^n$, where n indicates the number of dimensions. Griewank has an infinite number of optima of

Table 6.3: Optima of the Griewank function

Locations of optima		Fitness
x_1	x_2	$f(x_1, x_2)$
3.14	4.44	-0.0074
3.14	-4.44	-0.0074
0	0	0
-3.14	4.44	-0.0074
-3.14	-4.44	-0.0074

decreasing fitness as the distance from the global optimum increases. However, the positions of the optima are not spaced as evenly as in the case of the Rastrigin function. The Griewank function is tested in the range $x_1, x_2 \in [-5.0, 5.0]$. A two-dimensional version of the Griewank function has 5 optima in this range. The positions of the optima are listed in Table 6.3.

The Rastrigin function: Figure 6.4 shows the Rastrigin function in the range $x_1, x_2 \in [-1.25, 1.25]$ and in a larger range to illustrate its massive multimodality. The function is defined as

$$f(\mathbf{x}) = -\left(\sum_{i=1}^{n} \left[x_i^2 - 10\cos(2\pi x_i) + 10\right]\right) \tag{6.10}$$

The Rastrigin function is also very popular for testing niching algorithms, as it is massively multimodal. The function has a global optimum at $[0, 0]^n$ where n indicates the number of dimensions, as well as an infinite number of optima radiating out from the global optimum. The fitness of these optima decreases as the distance from the global optimum increases. The function is also eminently suitable for testing scalability, as optima are situated at regularly spaced intervals within the problem space. To test for a finite number of optima, a search space has to be defined. For this experiment the Rastrigin function was tested in the range $x_1, x_2 \in [-1.25, 1.25]$, where the two-dimensional version has 9 optima. The positions of these optima are listed in Table 6.4.

The Ackley function: The Ackley function is defined as

$$f(\mathbf{x}) = 20 + e - 20e^{-\frac{1}{5}\sqrt{\frac{1}{n}\sum_{i=1}^{n} x_i^2}} - e^{\frac{1}{n}\sum_{i=1}^{n}\cos(2x_i)} \tag{6.11}$$

Table 6.4: Optima of the Rastrigin function

Locations of optima		Fitness
x_1	x_2	$f(x_1, x_2)$
-1.0	-1.0	-2.0
-1.0	0.0	-1.0
-1.0	1.0	-2.0
0.0	-1.0	-1.0
0.0	0.0	0.0
0.0	1.0	-1.0
1.0	-1.0	-2.0
1.0	0.0	-1.0
1.0	1.0	-2.0

Table 6.5: Optima of the Ackley function

Locations of optima		Fitness
x_1	x_2	$f(x_1, x_2)$
-1.0	-1.0	-2.0
-1.0	0.0	-1.0
-1.0	1.0	-2.0
0.0	-1.0	-1.0
0.0	0.0	0.0
0.0	1.0	-1.0
1.0	-1.0	-2.0
1.0	0.0	-1.0
1.0	1.0	-2.0

Figure 6.5 shows the function landscape in the range where it is tested, and in a larger range to illustrate the multimodal character of the function. The function also has an infinite number of optima surrounding a central global optimum. Fitnesses of the surrounding optima decrease sharply. In addition, the diameters of the niches in the function landscape become smaller as the distances from the central global optimum increase, making it quite difficult to detect niches some distance away from the center. For this experiment the Ackley function is tested in the range $x_1, x_2 \in [-1.6, 1.6]$ where the function has 9 optima. The positions of these optima are listed in Table 6.5.

Table 6.6: Optima of the Ursem F1 function

Locations of optima		Fitness
x_1	x_2	$f(x_1, x_2)$
-1.44	0	3.25
1.70	0	4.82

The Ursem F1 function: The Ursem F1 function is defined as

$$f(x_1, x_2) = -(sin(2x_1 - 0.5\pi) + 3cos(x_2) + 0.5x_1) \tag{6.12}$$

The function has a repetitive character and well-defined optima occur indefinitely, as illustrated for two ranges in Figure 6.6. However, it has no central optimum, but the fitnesses decrease linearly while the shapes of the niches remain the same. When the problem space is demarcated to contain two optima with different fitnesses, Ursem F1 is a simple two-dimensional function that should give good results for any niching algorithm. The Ursem F1 function is tested in the range $x_1 \in [-2.5, 3.0]$ and $x_2 \in [-2.0, 2.0]$, a region containing two of the optima. Table 6.6 lists the positions of these optima.

The Ursem F3 function: The Ursem F3 function is defined as

$$\begin{aligned} f(x_1, x_2) = & - (sin(2.2\pi x_1 - 0.5x_1)) \cdot \left(\frac{2 - |x_2|}{2}\right) \cdot \left(\frac{3 - |x_1|}{2}\right) \\ & - \left(sin(0.5\pi x_2^2 + 0.5\pi)\right) \cdot \left(\frac{2 - |x_2|}{2}\right) \cdot \left(\frac{2 - |x_1|}{2}\right) \end{aligned} \tag{6.13}$$

The landscape of the function contains a number of flattened peaks. Groups of four peaks with varying heights give the landscape the appearance of a mountain range. Figure 6.7 illustrates the function for small and large ranges. Groups of four peaks are repeated at specific intervals. The algorithms are tested with one such group in the range $x_1, x_2 \in [-2.0, 2.0]$. The positions of the optima in this group are listed in Table 6.7.

Table 6.7: Optima of the Ursem F3 function

Locations of optima		Fitness
x_1	x_2	$f(x_1, x_2)$
-1.72	0	1.5
-0.74	0	2.5
0.25	0	2.5
1.23	0	1.5

Table 6.8: Optima of the Six Hump Camel function

Locations of optima		Fitness
x_1	x_2	$f(x_1, x_2)$
-0.80	1.70	0.22
-0.71	0.09	1.03
-0.57	-1.61	-2.10
0.57	1.61	-2.10
0.71	-0.09	1.03
0.80	-1.70	0.22

The six hump camel function: The six hump camel function is illustrated in Figure 6.8 and defined as

$$f(x_1, x_2) = \left(4 - 2.1x_1^2 + \frac{x_1^4}{3}\right) x_1^2 + x_1 x_2 + (-4 + 4x_2^2)x_2^2 \tag{6.14}$$

Like the Himmelblau function, the six hump camel function is defined in two dimensions and is not repetitive. It has six rounded peaks of which the heights differ considerably. The niche radii also differ, making it a good test function to estimate the robustness of niching algorithms. The range $x_1 \in [1.1, 1.1]$ and $x_2 \in [-1.9, 1.9]$ contains six optima and the function is tested in this region. Table 6.8 lists the positions of these optima.

6.2.4 Initial swarm sizes and granularity

Table 6.9 lists the initial swarm sizes and granularity settings used for all experiments for which results are presented in sections 6.3 and 6.4.

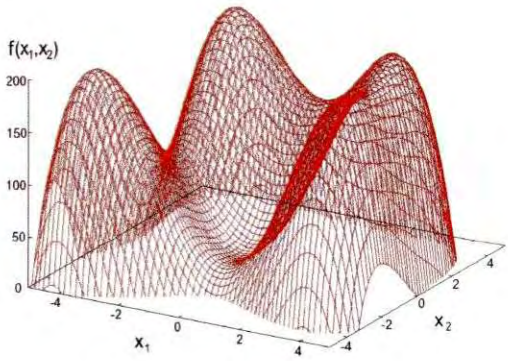
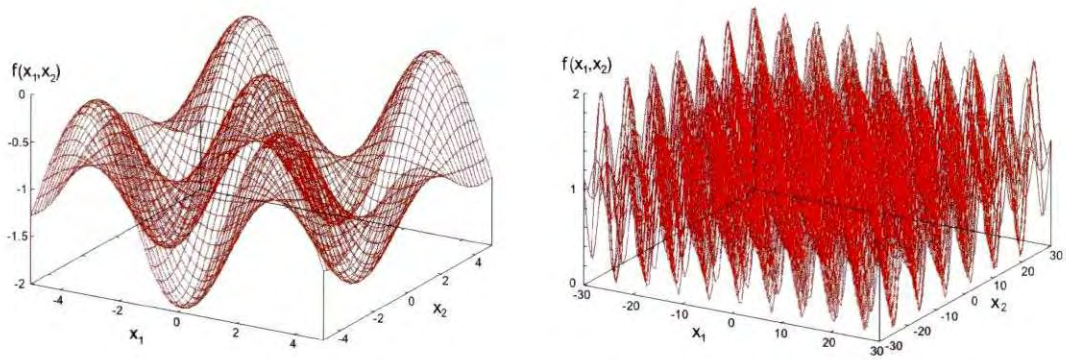
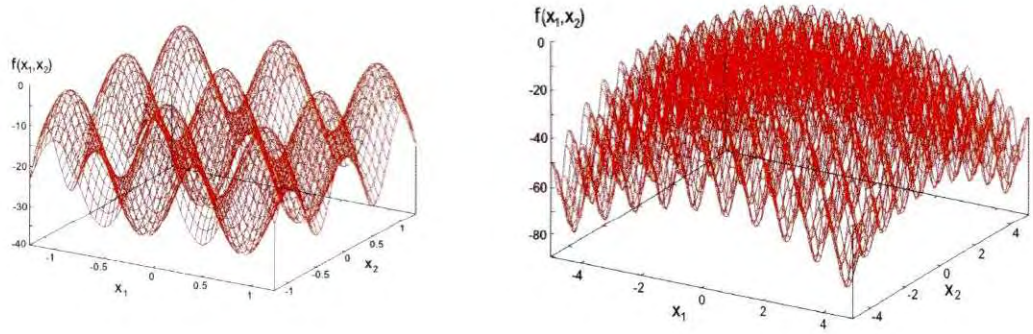


Figure 6.2: The Himmelblau function showing maxima.



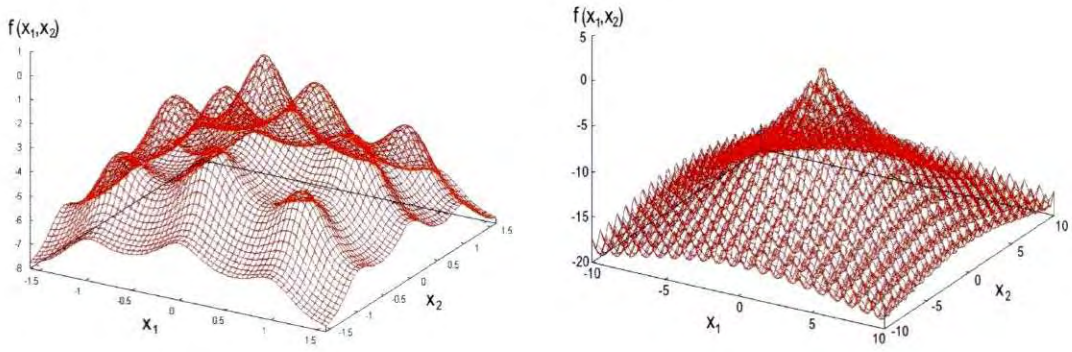
(a) (b)

Figure 6.3: The Griewank function.



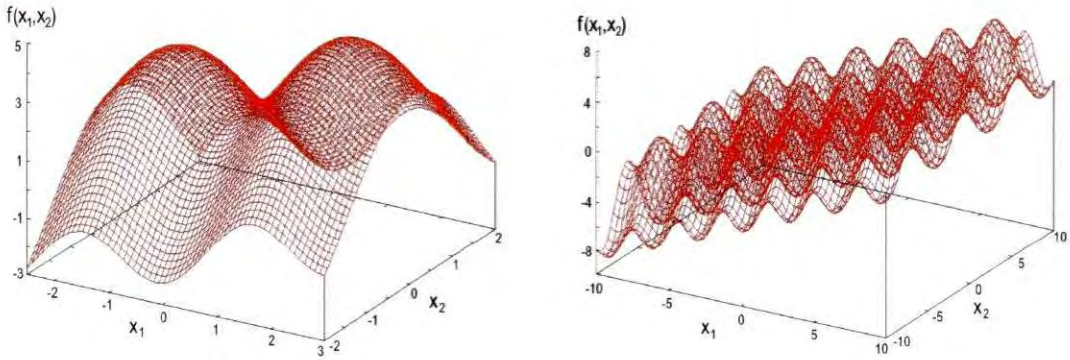
(a) (b)

Figure 6.4: The Rastrigin function.



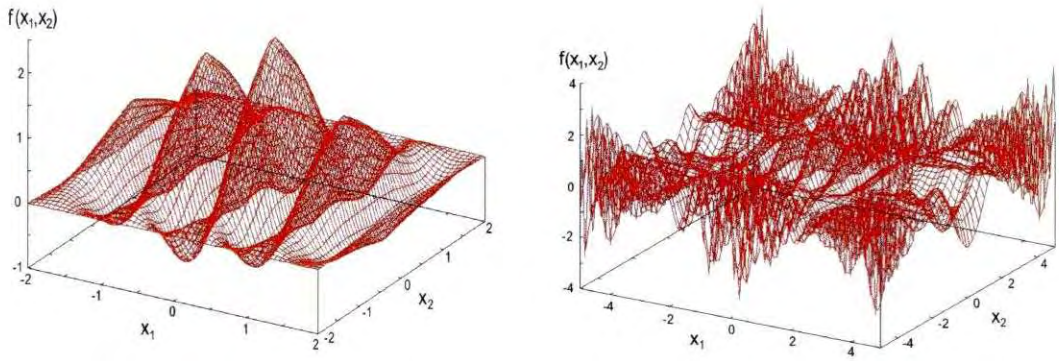
(a) (b)

Figure 6.5: The Ackley function.



(a) (b)

Figure 6.6: The Ursem F1 function.



(a) (b)

Figure 6.7: The Ursem F3 function.

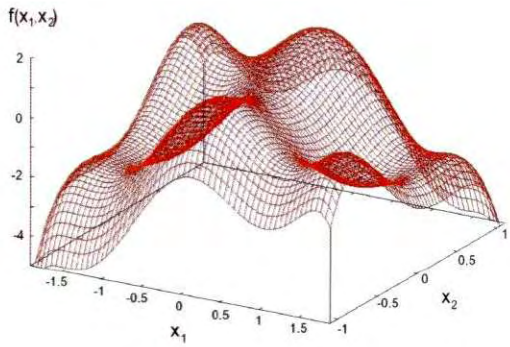


Figure 6.8: The six hump camel function.

Table 6.9: Initial swarm sizes and granularity for test functions

Function	Swarm size	Granularity
$F1$	20	0.05
$F2$	20	0.05
$F3$	20	0.05
$F4$	20	0.05
Himmelblau	30	0.5
Griewank	40	0.5
Rastrigin	60	0.1
Ackley	60	0.3
Ursem F1	30	0.5
Ursem F3	40	0.3
Six hump camel	50	0.3

6.3 Results of the sequential vector-based PSO

To analyze the performance, and investigate the effect of particle initialization on performance of the sequential VBPSO, Tables 6.10, 6.11, and 6.12 summarize the average results of 50 experiments for functions $F1$ to $F4$ and the Himmelblau function. Results have been obtained using Sobol sequences as well as a system-supplied random number generator, and for 200 as well as 500 iterations. Therefore, each function yielded four independent sets of results.

The tables list the average number of function evaluations and the average number of solutions located per configuration over 50 runs, as well as the average number of duplicate solutions. The seventh column provides the average of the derivatives of the objective function, $f'(x)$, at positions where optima are located. Derivatives were used in these experiments to indicate the quality of the solutions, since the offline error (the method of choice), calculated to 6 decimal positions, yielded a value of 0 in all cases. The success rate indicates the number of optima located as a percentage of all possible optima over 50 runs.

The averages of the function evaluations, number of solutions and number of duplicate solutions are reported together with the standard error.

Table 6.13 shows statistical data generated by the Mann-Whitney U test for the sequential vector-based PSO. The success rate of experiments using Sobol sequences (sample 1) are compared to that of experiments using the system-supplied random number generator (sample 2). The following results are listed for the one-dimensional functions $F1$ to $F4$ and the two-dimensional Himmelblau function, for 200 and 500 iterations:

- the sums of the rankings of sample 1 (ΣR_1) and sample 2 (ΣR_2).
- Z, the normal approximation of the Mann-Whitney U test statistic [87].
- the outcomes of the hypothesis tests.

Discussion

Results produced by the sequential vector-based PSO algorithm on functions $F1$ to $F4$ and the Himmelblau function showed an average success rate of above 98% when Sobol sequences were used to generate initial particle positions and above 90% for a distribution by means of a system-supplied random number generator. For all functions, the average success rate was higher when Sobol sequences were used. However, the Mann-Whitney U test (Table 6.19)

only shows a statistically significant difference between the two approaches for the sequential VBPSO with 200 iterations at a significance level of 0.05.

Results also show that, for each one-dimensional functions as well as the Himmelblau function, the average number of function evaluations recorded for experiments using Sobol sequences and those using a system-supplied random number generator differ very slightly for the same number of iterations. No trend could be observed. Averages of the derivatives of the four sets of results for each objective function were in the same order of magnitude. However, averages of derivatives for functions $F1$ and $F3$ as well as the Himmelblau function, were much smaller than the averages of derivatives for functions $F2$ and $F4$. For functions $F1$ and $F3$ distances between optima are similar, while this is not the case for functions $F2$ and $F4$. For the Himmelblau function, distances between optima differed only slightly. Therefore, the quality of the solutions depends to a large extent on the function landscape.

Although the sequential vector-based PSO performed well in the sense that it could locate all the optima in most of the runs, it produced a number of duplicate solutions. Functions $F1$ to $F4$ that have five optima each in the search space, produced an average of 1.4 duplicates per run. The Himmelblau function yielded an average of 7.8 duplicates per run while the search space only contains four optima. The conclusion was reached that the sequential VBPSO creates too many duplicates and was therefore excluded from further analysis.

Table 6.10: Sequential vector-based PSO results for one-dimensional functions $F1$ and $F2$

	Random number generator	Iterations	Average function evaluations	Average ♯ solutions	Average ♯ duplicates	Average derivative	Success rate
$F1$	System-supplied	200	5234 ± 126.2	4.76 ± 0.06	1.38 ± 0.2	3.01E-14 ± 2.25E-15	95.2%
		500	12374 ± 342.1	4.64 ± 0.07	1.18 ± 0.2	2.80E-14 ± 3.86E-15	92.8%
	Sobol sequences	200	4702 ± 119.5	5 ± 0	1.12 ± 0.21	2.91E-14 ± 2.26E-15	100%
		500	11479 ± 310.6	5 ± 0	0.9 ± 0.19	2.84E-14 ± 2.55E-15	100%
$F2$	System-supplied	200	5185 ± 125.3	4.66 ± 0.07	1.9 ± 0.26	0.11 ± 0.08	93.2%
		500	12786 ± 321.4	4.68 ± 0.1	1.78 ± 0.25	0.07 ± 0.12	93.6%
	Sobol sequences	200	4755 ± 108.7	4.96 ± 0.03	1.36 ± 0.21	0.07 ± 0.07	99.2%
		500	11811 ± 253.5	4.98 ± 0.07	1.36 ± 0.2	0.11 ± 0.07	99.6%

Table 6.11: Sequential vector-based PSO results for one-dimensional functions $F3$ and $F4$

	Random number generator	Iterations	Average function evaluations	Average ♯ solutions	Average ♯ duplicates	Average derivation	Success rate
$F3$	System-supplied	200	5144 ± 111.9	4.5 ± 0.1	1.58 ± 0.2	1.5E-04 $\pm$ 5.47E-05	90%
		500	12877 ± 290.2	4.62 ± 0.09	1.5 ± 0.22	1.4E-04 $\pm$ 5.33E-05	92.4%
	Sobol sequences	200	4617 ± 94.84	4.96 ± 0.03	0.94 ± 0.16	1.4E-04 $\pm$ 5.33E-05	99.2%
		500	11770 ± 243.5	4.94 ± 0.03	1.24 ± 0.19	1.9E-04 $\pm$ 6.51E-05	98.8%
$F4$	System-supplied	200	5014 ± 107.4	4.72 ± 0.06	1.74 ± 0.22	0.21 ± 0.022	94.4%
		500	12575 ± 315.8	4.6 ± 0.08	1.8 ± 0.25	0.2 ± 0.018	92%
	Sobol sequences	200	4755 ± 118.6	4.94 ± 0.03	1.4 ± 0.23	0.21 ± 0.21	98.8%
		500	11609 ± 262.9	4.98 ± 0.02	1.22 ± 0.19	0.21 ± 0.02	99.6%

Table 6.12: Sequential vector-based PSO results for the Himmelblau function

Random number generator	Iterations	Average evaluations	Average ♯ solutions	Average ♯ duplicates	Average $f'(x)$	Average $f'(y)$	Success rate
System-supplied	200	8764 ± 151	3.98 ± 0.02	7.78 ± 0.38	5.4E-04 $\pm$ 1.2E-03	9.1E-04 $\pm$ 2.2E-03	99.5%
	500	21438 ± 374	3.96 ± 0.03	7.5 ± 0.41	1.4E-04 $\pm$ 5.1E-04	3.1E-04 $\pm$ 5.1E-04	99%
Sobol sequences	200	8910 ± 219	4 ± 0	7.88 ± 0.53	9.8E-03 $\pm$ 0.03	7.2E-03 $\pm$ 0.02	100%
	500	21891 ± 473	3.98 ± 0.02	7.96 ± 0.47	1E-04 $\pm$ 8.55E-06	1.7E-04 $\pm$ 1.89E-05	99.5%

Table 6.13: Results of the Mann-Whitney U test. Success rates of the sequential VBPSO using a system-supplied random number generator are compared to those using Sobol sequences.

Function	Iterations	ΣR_1	ΣR_2	Significance level	Z	H_0/H_1
$F1$	200	2.83E+03	2.23E+03	0.05	-2.068	H_1
				0.01		H_0
	500	2.78E+03	2.28E+03	0.05	-1.72	H_0
				0.01		H_0
$F2$	200	2.88E+03	2.17E+03	0.05	-2.42	H_1
				0.01		H_0
	500	2.78E+03	2.27E+03	0.05	-1.73	H_0
				0.01		H_0
$F3$	200	3.00E+03	2.05E+03	0.05	-3.30	H_1
				0.01		H_0
	500	2.88E+03	2.17E+03	0.05	-2.42	H_0
				0.01		H_0
$F4$	200	2.80E+03	2.25E+03	0.05	-1.90	H_1
				0.01		H_0
	500	2.95E+03	2.10E+03	0.05	-2.94	H_0
				0.01		H_0
Himmelblau	200	2.80E+03	2.25E+03	0.05	-1.90	H_1
				0.01		H_0
	500	2.95E+03	2.10E+03	0.05	-2.94	H_0
				0.01		H_0

6.4 Results for the parallel and enhanced parallel vector-based PSO

This section presents results for the parallel and enhanced parallel vector-based PSO tested on all the benchmark functions described in section 6.2.3. Section 6.4.1 presents results where Sobol sequences as well as a system-supplied random number generator were used to initialize the swarm, and for 200 as well as 500 iterations. These results are compared to determine the best approach for initializing the swarm and the preferred number of iterations. Since both parallel algorithms incorporate a merging process, observations of the course of merging for both algorithms on all the functions are reported in section 6.4.2. To determine the best parallel algorithm, section 6.4.3 presents a comparison of the success rates of the algorithms.

6.4.1 Test results

Tables 6.14 to 6.17 present results of the performance of the parallel and enhanced parallel vector-based PSO algorithms for the one-dimensional functions $F1$ to $F4$. Tables 6.20 to 6.26 report results of the parallel algorithms for the following two-dimensional functions: Himmelblau, Griewank, Rastrigin, Ackley, Ursem $F1$, Ursem $F3$, and the six hump camel function.

All tables list the the average number of function evaluations and the average number of solutions over 50 runs together with the standard error, the average of the derivatives of the objective function, $f'(x_i)$, at positions where optima are located, and the success rate. The success rate is the total number of optima as a percentage of the total number of possible optima.

The outcome of the Mann-Whitney U test to determine statistically significant differences between experiments using Sobol sequences and a system-supplied random number generator are given in Tables 6.18, 6.19, 6.27, and 6.28. These outcomes are presented for both the parallel and enhanced parallel vector-based PSO algorithms.

Discussion

From the reported results and statistical data a number of observations were made. However, owing to the stochastic nature of PSO, results obtained from a number of independent runs of such algorithms are not deterministic. Therefore, small differences cannot form the basis of general conclusions on the performance of different algorithms.

One purpose of these experiments was to ascertain the performance of the vector-based algorithms on functions with varying landscapes with regard to the shapes, sizes, and placing of the niches. Results reported showed that the vector-based PSO algorithms performed well on functions where the shapes and distribution of the niches are relatively symmetrical, like the Himmelblau, Rastrigin, Griewank and Ursem F1 functions. The Ackley, Ursem F3 and six hump camel functions do not exhibit these characteristics, and performance degradation could be expected. However, although slightly worse, the average success rate of all functions was still above 98% for the parallel vector-based PSO and above 99% for the enhanced parallel version. Therefore, it can be concluded that both versions of the VBPSO that were tested in this section, are robust niching algorithms that performs well even in adverse circumstances.

The experiments were also designed to assess the influence of different approaches to initialization of particles. According to Table 6.35, algorithms using Sobol sequences as a random number generator had better success rates than those using system-supplied random number generators in all instances. Tables 6.27 and 6.28 report results of the Mann-Whitney U test comparing these two approaches to initialize particles. Rankings for algorithms using Sobol sequences (ΣR_1) were better than or equal to rankings for algorithms using system-supplied random number generators (ΣR_2). However, the Himmelblau, Griewank, Rastrigin and Ursem F1 functions showed no significant difference between algorithms using the two random number generators, that is, the null hypothesis was supported. Using the parallel vector-based PSO and 200 iterations, the alternative hypothesis was supported for the Ursem F3, Ackley and six hump camel functions at a significance level of 0.05, and for the six hump camel function at a significance level of 0.01. In the case of the enhanced parallel vector-based PSO, the alternative hypothesis was only supported for the 500 iterations version of the Ackley function at a significance level of 0.05. The function landscapes of these functions are less symmetrical than those of the other functions. For such function landscapes, an even distribution of particles becomes more important, which is provided by Sobol sequences.

Experiments conducted with 500 iterations showed smaller derivatives than experiments with 200 iterations. Although the number of iterations had no noticeable effect on the success rate, better quality solutions were obtained with more iterations.

From the above observations it can be concluded that algorithms using Sobol sequences to distribute particles evenly throughout the search space, perform better than those using a system-supplied random number generator. A higher number of iterations ensure better quality solutions. For this study, 500 iterations yielded an adequate level of accuracy. Hence, algorithms using Sobol sequences and executing 500 iterations will be used for all further work.

Table 6.14: Parallel vector-based PSO results for one-dimensional functions $F1$ and $F2$

	Random number generator	Iterations	Average function evaluations	Average ♯ solutions	Average derivation	Success rate
$F1$	System-supplied	200	5234 ± 129	4.86 ± 0.05	6.09E-07 $\pm$ 1.34E-06	97.2%
		500	13028 ± 335	4.74 ± 0.08	2.88E-14 $\pm$ 2.34E-15	94.8%
	Sobol sequences	200	4763 ± 82	5 ± 0	5.33E-06 $\pm$ 1.33E-05	100%
		500	12253 ± 260	5 ± 0	2.88E-14 $\pm$ 2.32E-15	100%
$F2$	System-supplied	200	5327 ± 120	4.74 ± 0.07	0.11 ± 0.07	94.8%
		500	12776 ± 318	4.88 ± 0.05	9.5E-02 $\pm$ 0.07	97.6%
	Sobol sequences	200	5181 ± 121	5 ± 0	0.10 ± 0.07	100%
		500	12696 ± 258	4.98 ± 0.02	9.7E-02 $\pm$ 0.07	99.6%

Table 6.15: Parallel vector-based PSO results for one-dimensional functions $F3$ and $F4$

	Random number generator	Iterations	Average function evaluations	Average ♯ solutions	Average derivation	Success rate
$F3$	System-supplied	200	5290 ± 112	4.94 ± 0.03	1.3E-02 $\pm$ 0.04	98.8%
		500	13249 ± 316	4.84 ± 0.05	4.8E-04 $\pm$ 7.0E-04	96.8%
	Sobol sequences	200	5100 ± 94	5 ± 0	2E-03 $\pm$ 5E-03	100%
		500	12273 ± 265	4.98 ± 0.02	1.6E-04 $\pm$ 5.59E-05	99.6%
$F4$	System-supplied	200	5400 ± 140	4.8 ± 0.06	0.2 ± 0.02	96%
		500	13078 ± 287	4.84 ± 0.05	0.21 ± 0.02	96.8%
	Sobol sequences	200	5181 ± 120	4.94 ± 0.3	0.2 ± 0.02	98.8%
		500	12314 ± 293	4.94 ± 0.03	4.5E-02 $\pm$ 0.02	98.8%

Table 6.16: Enhanced parallel vector-based PSO results for one-dimensional functions $F1$ and $F2$

	Random number generator	Iterations	Average function evaluations	Average ♯ solutions	Average derivation	Success rate
$F1$	System-supplied	200	5880 ± 138	4.8 ± 0.06	2.84E-14 $\pm$ 2.32E-15	96%
		500	13769 ± 347	4.48 ± 0.05	2.77E-14 $\pm$ 2.87E-15	96.8%
	Sobol sequences	200	5490 ± 137	5 ± 0	2.78E-14 $\pm$ 2.85E-15	100%
		500	13153 ± 281	5 ± 0	2.87E-14 $\pm$ 2.33E-15	100%
$F2$	System-supplied	200	5905 ± 163	4.84 ± 0.07	0.11 ± 0.07	96.8%
		500	14071 ± 323	4.76 ± 0.06	8.7E-02 $\pm$ 0.08	95.2%
	Sobol sequences	200	5348 ± 122	4.98 ± 0.02	9.7E-02 $\pm$ 0.07	99.6%
		500	12733 ± 242	5 ± 0	0.10 ± 0.07	100%

Table 6.17: Enhanced parallel vector-based PSO results for one-dimensional functions $F3$ and $F4$

	Random number generator	Iterations	Average function evaluations	Average ♯ solutions	Average derivation	Success rate
$F3$	System-supplied	200	5609 ± 137	4.8 ± 0.06	2.7E-04 ± 2.3E-04	96%
		500	13225 ± 310	4.7 ± 0.08	3.8E-04 ± 7.34E-05	94%
	Sobol sequences	200	5369 ± 109	5 ± 0	3.8E-04 ± 5.4E-04	100%
		500	12955 ± 291	5 ± 0	3.9E-04 ± 7.47E-05	100%
$F4$	System-supplied	200	5707 ± 140	4.74 ± 0.07	0.21 ± 0.02	94.8%
		500	13651 ± 318	4.6 ± 0.09	0.21 ± 0.02	92%
	Sobol sequences	200	5382 ± 115	4.92 ± 0.04	0.20 ± 0.02	98.4%
		500	12992 ± 258	4.96 ± 0.03	0.20 ± 0.02	99.2%

Table 6.18: Summary of ranks based on the Mann-Whitney U test for comparing success rates of algorithms using a system-supplied random number generator to those using Sobol sequences

Function	Iterations	Parallel VBPSO		Enhanced parallel VBPSO	
		ΣR_1	ΣR_2	ΣR_1	ΣR_1
$F1$	200	2.70E+03	2.35E+03	2.75E+03	2.30E+03
	500	2.78E+03	2.28E+03	2.73E+03	2.33E+03
$F2$	200	2.78E+03	2.27E+03	2.65E+03	2.40E+03
	500	2.78E+03	2.27E+03	2.83E+03	2.23E+03
$F3$	200	2.60E+03	2.45E+03	2.75E+03	2.30E+03
	500	2.70E+03	2.35E+03	2.83E+03	2.23E+03
$F4$	200	2.68E+03	2.37E+03	2.73E+03	2.32E+03
	500	2.68E+03	2.37E+03	2.88E+03	2.17E+03

Table 6.19: Results of the Mann-Whitney U test. Success rates of the parallel and enhanced parallel VBPSO algorithms using a system-supplied random number generator are compared to those using Sobol sequences

Function	Iterations	Significance level	Parallel VBPSO		Enhanced parallel VBPSO	
			Z	H_0/H_1	Z	H_0/H_1
$F1$	200	0.05	-1.21	H_0	-1.55	H_0
		0.01		H_0		H_0
	500	0.05	-1.72	H_0	-1.38	H_0
		0.01		H_0		H_0
$F2$	200	0.05	-1.73	H_0	-8.69E-01	H_0
		0.01		H_0		H_0
	500	0.05	-1.73	H_0	-2.068	H_0
		0.01		H_0		H_0
$F3$	200	0.05	-5.17E-01	H_0	-1.55	H_0
		0.01		H_0		H_0
	500	0.05	-1.21	H_0	-2.068	H_0
		0.01		H_0		H_0
$F4$	200	0.05	-1.04	H_0	-1.39	H_0
		0.01		H_0		H_0
	500	0.05	-1.04	H_0	-2.43	H_0
		0.01		H_0		H_0

Table 6.20: Parallel and enhanced parallel vector-based PSO results for the Himmelblau function

	Random number generator	Iterations	Average evaluations	Average ♯ solutions	Average $f'(x)$	Average $f'(y)$	Success rate
Parallel VBPSO	System-supplied	200	9358 ± 238	4 ± 0	2.6e-04 ± 1.4E-04	2.6E-04 ± 1E-04	100%
		500	22062 ± 432	3.96 ± 0.03	1.3E-04 ± 4.73E-05	0.02 ± 0.04	99%
	Sobol sequences	200	9328 ± 170	4 ± 0	1.3E-04 ± 2.31E-05	1.8E-04 ± 2.17E-05	100%
		500	24074 ± 389	4 ± 0	1.1E-04 ± 8.67E-05	1.6E-04 ± 1.91E-05	100%
Enhanced parallel VBPSO	System-supplied	200	10709 ± 242	3.98 ± 0.02	1.3E-04 ± 1E-04	1.9E-04 ± 1.9E-04	99.5%
		500	24483 ± 486	3.98 ± 0.02	1.86E-05 ± 2.27E-06	1.57E-05 ± 2.07E-05	99.5%
	Sobol sequences	200	11222 ± 233	4 ± 0	7.64E-05 ± 4.22E-05	5.27E-05 ± 2.53E-05	100%
		500	25310 ± 513	4 ± 0	0.018 ± 0.04	0.013 ± 0.02	100%

Table 6.21: Parallel and enhanced parallel vector-based PSO results for Griewank function

	Random number generator	Iterations	Average evaluations	Average ♯ solutions	Average $f'(x)$	Average $f'(y)$	Success rate
Parallel VBPSO	System-supplied	200	11819 ± 180	5 ± 0	3.23E-05 ± 5.53E-05	1.7E-03 ± 3.9E-03	100%
		500	28561 ± 447	5 ± 0	9.00E-05 ± 2E-04	1.79E-06 ± 1.27E-07	100%
	Sobol sequences	200	11654 ± 170	5 ± 0	2.26E-06 ± 2.82E-07	2.20E-06 ± 5.13E-07	100%
		500	29462 ± 521	5 ± 0	2.12E-06 ± 1.48E-07	1.80E-06 ± 1.26E-07	100%
Enhanced parallel VBPSO	System-supplied	200	13590 ± 229	4.98 ± 0.02	2.29E-06 ± 2.18E-07	2.01E-06 ± 2.08E-07	99.6%
		500	31512 ± 506	4.96 ± 0.03	2.10E-06 ± 1.50E-07	1.79E-06 ± 1.27E-07	99.2%
	Sobol sequences	200	13853 ± 235	5 ± 0	3.22E-06 ± 1.75E-06	2.31E-06 ± 8.74E-07	100%
		500	31678 ± 593	5 ± 0	2.11E-06 ± 1.49E-07	1.79E-06 ± 1.27E-07	100%

Table 6.22: Parallel and enhanced parallel vector-based PSO results for Rastrigin function

	Random number generator	Iterations	Average evaluations	Average ♯ solutions	Average $f'(x)$	Average $f'(y)$	Success rate
Parallel VBPSO	System-supplied	200	17166 ± 296	8.98 ± 0.02	1.2E-04 ± 2.85E-05	1.4E-04 ± 8.17E-05	99.78%
		500	43313 ± 683	8.98 ± 0.02	0.11 ± 0.34	0.016 ± 0.47	99.78%
	Sobol sequences	200	17554 ± 281	9 ± 0	0.034± 8.5E-03	8.7E-04 ± 2.3E-03	100%
		500	43644 ± 570	9 ± 0	9.58E-05 ± 9.58E-06	9.58E-05 ± 9.58E-06	100%
Enhanced parallel VBPSO	System-supplied	200	20800 ± 296	8.88 ± 0.05	1.3E-04 ± 3.16E-05	1.4E-04 ± 5.76E-05	98.67%
		500	48234 ± 604	8.88 ± 0.05	9.49E-05 ± 9.63E-06	9.53E-05 ± 9.62E-06	98.67%
	Sobol sequences	200	20973 ± 251	9 ± 0	1.2E-04 ± 1.71E-05	1.2E-04 ± 2.50E-05	100%
		500	48687 ± 634	9 ± 0	9.58E-05 ± 9.59E-06	9.58E-05 ± 9.59E-06	100%

Table 6.23: Parallel and enhanced parallel vector-based PSO results for Ackley function

	Random number generator	Iterations	Average evaluations	Average ♯ solutions	Average $f'(x)$	Average $f'(y)$	Success rate
Parallel VBPSO	System-supplied	200	20427 ± 241	8.44 ± 0.09	0.23 ± 0.09	0.22 ± 0.1	93.78%
		500	49787 ± 782	8.52 ± 0.1	0.21 ± 0.09	0.22 ± 0.09	94.67%
	Sobol sequences	200	20309 ± 272	8.76 ± 0.07	0.22 ± 0.1	0.2 ± 0.09	97.33%
		500	52383 ± 752	8.6 ± 0.08	0.21 ± 0.09	0.22 ± 0.09	95.56%
Enhanced parallel VBPSO	System-supplied	200	21594 ± 233	8.84 ± 0.06	0.19 ± 0.09	0.22 ± 0.09	96.67%
		500	51212 ± 642	8.68 ± 0.08	0.2 ± 0.09	0.2 ± 0.09	96.44%
	Sobol sequences	200	21964 ± 324	9 ± 0	0.2 ± 0.09	0.19 ± 0.09	100%
		500	51824 ± 653	9 ± 0	0.2 ± 0.09	0.2 ± 0.09	100%

Table 6.24: Parallel and enhanced parallel vector-based PSO results for Ursem $F1$ function

	Random number generator	Iterations	Average evaluations	Average ♯ solutions	Average $f'(x)$	Average $f'(y)$	Success rate
Parallel VBPSO	System-supplied	200	9734 ± 212	2 ± 0	1.45E-05 ± 2.25E-07	4.36E-07 ± 2.11E-07	100%
		500	22766 ± 551	2 ± 0	1.42E-05 ± 7.02E-08	2.22E-08 ± 2.31E-09	100%
	Sobol sequences	200	9379 ± 212	2 ± 0	1.43E-05 ± 1.68E-07	1.46E-07 ± 5.75E-08	100%
		500	23339 ± 437	2 ± 0	1.42E-05 ± 7.02E-08	2.33E-08 ± 2.27E-09	100%
Enhanced parallel VBPSO	System-supplied	200	10348 ± 215	2 ± 0	1.6E-06 ± 1.97E-07	4.5E-07 ± 1.94E-07	100%
		500	24161 ± 554	2 ± 0	1.27E-06 ± 7.02E-08	0 ± 0	100%
	Sobol sequences	200	10473 ± 231	2 ± 0	1.47E-06 ± 1.74E-07	3.9E-07 ± 4.29E-07	100%
		500	25225 ± 577	2 ± 0	1.27E-06 ± 1.27E-06	0 ± 0	100%

Table 6.25: Parallel and enhanced parallel vector-based PSO results for Ursem $F3$ function

	Random number generator	Iterations	Average evaluations	Average ♯ solutions	Average $f'(x)$	Average $f'(y)$	Success rate
Parallel VBPSO	System-supplied	200	13947 ± 210	3.86 ± 0.08	0.51 ± 0.04	0.76 ± 0.04	98.5%
		500	33752 ± 494	3.98 ± 0.02	0.51 ± 0.04	0.76 ± 0.04	99.5%
	Sobol sequences	200	14162 ± 232	4 ± 0	0.51 ± 0.04	0.76 ± 0.04	100%
		500	35955 ± 394	3.98 ± 0.02	0.51 ± 0.04	0.76 ± 0.04	99.5%
Enhanced parallel VBPSO	System-supplied	200	16168 ± 244	4 ± 0	0.51 ± 0.04	0.76 ± 0.04	100%
		500	37698 ± 600	3.98 ± 0.02	0.51 ± 0.04	0.76 ± 0.04	99.5%
	Sobol sequences	200	16474 ± 255	4 ± 0	0.51 ± 0.04	0.76 ± 0.04	100%
		500	38935 ± 578	4 ± 0	0.51 ± 0.04	0.76 ± 0.04	100%

Table 6.26: Parallel and enhanced parallel vector-based PSO results for the six hump camel function

	Random number generator	Iterations	Average evaluations	Average ♯ solutions	Average $f'(x)$	Average $f'(y)$	Success rate
Parallel VBPSO	System-supplied	200	16470 ± 247	5.12 ± 0.1	3.56E-05 ± 4.32E-06	8.98E-06 ± 1.88E-06	85.33%
		500	40371 ± 584	5.5 ± 0.08	1.2E-03 ± 2.8E-03	6.94E-06 ± 1.08E-07	91.67%
	Sobol sequences	200	16377 ± 254	5.6 ± 0.07	6.5E-03 ± 0.02	3.7E-04 ± 8.4E-04	93.33%
		500	41649 ± 640	5.56 ± 0.08	3.54E-05 ± 3.83E-06	6.97E-06 ± 1.09E-07	92.67%
Enhanced parallel VBPSO	System-supplied	200	18690 ± 234	5.68 ± 0.07	2.3E-03 ± 5.4E-03	2.3E-04 ± 5.1E-04	94.67%
		500	41418 ± 568	5.76 ± 0.06	2.41E-06 ± 2.81E-07	6.47E-06 ± 3.55E-07	96%
	Sobol sequences	200	18450 ± 234	5.86 ± 0.05	5.4E-04 ± 1.3E-03	1.3E-03 ± 3.2E-03	97.67%
		500	42825 ± 580	5.96 ± 0.03	2.42E-06 ± 2.78E-07	6.4E-06 ± 3.59E-07	99.33%

Table 6.27: Summary of ranks based on the Mann-Whitney U test for comparing success rates of algorithms using a system-supplied random number generator to those using Sobol sequences

Function	Iterations	Parallel VBPSO		Enhanced parallel VBPSO	
		ΣR_1	ΣR_2	ΣR_1	ΣR_2
Himmelblau	200	2.53E+03	2.53E+03	2.55E+03	2.50E+03
	500	2.58E+03	2.48E+03	2.55E+03	2.50E+03
Griewank	200	2.53E+03	2.53E+03	2.55E+03	2.50E+03
	500	2.53E+03	2.53E+03	2.55E+03	2.50E+03
Rastrigin	200	2.55E+03	2.50E+03	2.68E+03	2.38E+03
	500	2.55E+03	2.50E+03	2.68E+03	2.38E+03
Ursem $F1$	200	2.53E+03	2.53E+03	2.53E+03	2.53E+03
	500	2.53E+03	2.53E+03	2.53E+03	2.53E+03
Ursem $F3$	200	2.60E+03	2.45E+03	2.53E+03	2.53E+03
	500	2.53E+03	2.53E+03	2.55E+03	2.50E+03
Ackley	200	2.86E+03	2.19E+03	2.70E+03	2.35E+03
	500	2.58E+03	2.47E+03	2.88E+03	2.18E+03
Six hump camel	200	2.98E+03	2.07E+03	2.73E+03	2.32E+03
	500	2.58E+03	2.47E+03	2.78E+03	2.28E+03

Table 6.28: Results of the Mann-Whitney U test. Success rates of the parallel and enhanced parallel VBPSO algorithms using a system-supplied random number generator are compared to those using Sobol sequences

Function	Iterations	Significance level	Parallel VBPSO		Enhanced parallel VBPSO	
			Z	H_0/H_1	Z	H_0/H_1
Himmelblau	200	0.05	0.00	H_0	-1.72E-01	H_0
		0.01		H_0		H_0
	500	0.05	-3.45E-01	H_0	-1.72E-01	H_0
		0.01		H_0		H_0
Griewank	200	0.05	0.00	H_0	-1.72E-01	H_0
		0.01		H_0		H_0
	500	0.05	0.00	H_0	-3.45E-01	H_0
		0.01		H_0		H_0
Rastrigin	200	0.05	-1.72E-01	H_0	-1.034	H_0
		0.01		H_0		H_0
	500	0.05	-1.72E-01	H_0	-1.034	H_0
		0.01		H_0		H_0
Ursem $F1$	200	0.05	0.00	H_0	0.00	H_0
		0.01		H_0		H_0
	500	0.05	0.00	H_0	0.00	H_0
		0.01		H_0		H_0
Ursem $F3$	200	0.05	-5.17E-01	H_1	0.00	H_0
		0.01		H_0		H_0
	500	0.05	0.00	H_0	-1.72E-01	H_0
		0.01		H_0		H_0
Ackley	200	0.05	-2.31	H_1	-1.21	H_0
		0.01		H_0		H_0
	500	0.05	-4.00E-01	H_0	-2.41	H_1
		0.01		H_0		H_0
Six hump camel	200	0.05	-3.14	H_1	-1.40	H_0
		0.01		H_1		H_0
	500	0.05	-4.10E-01	H_0	-1.72	H_0
		0.01		H_0		H_0

6.4.2 Tracking merging of niches

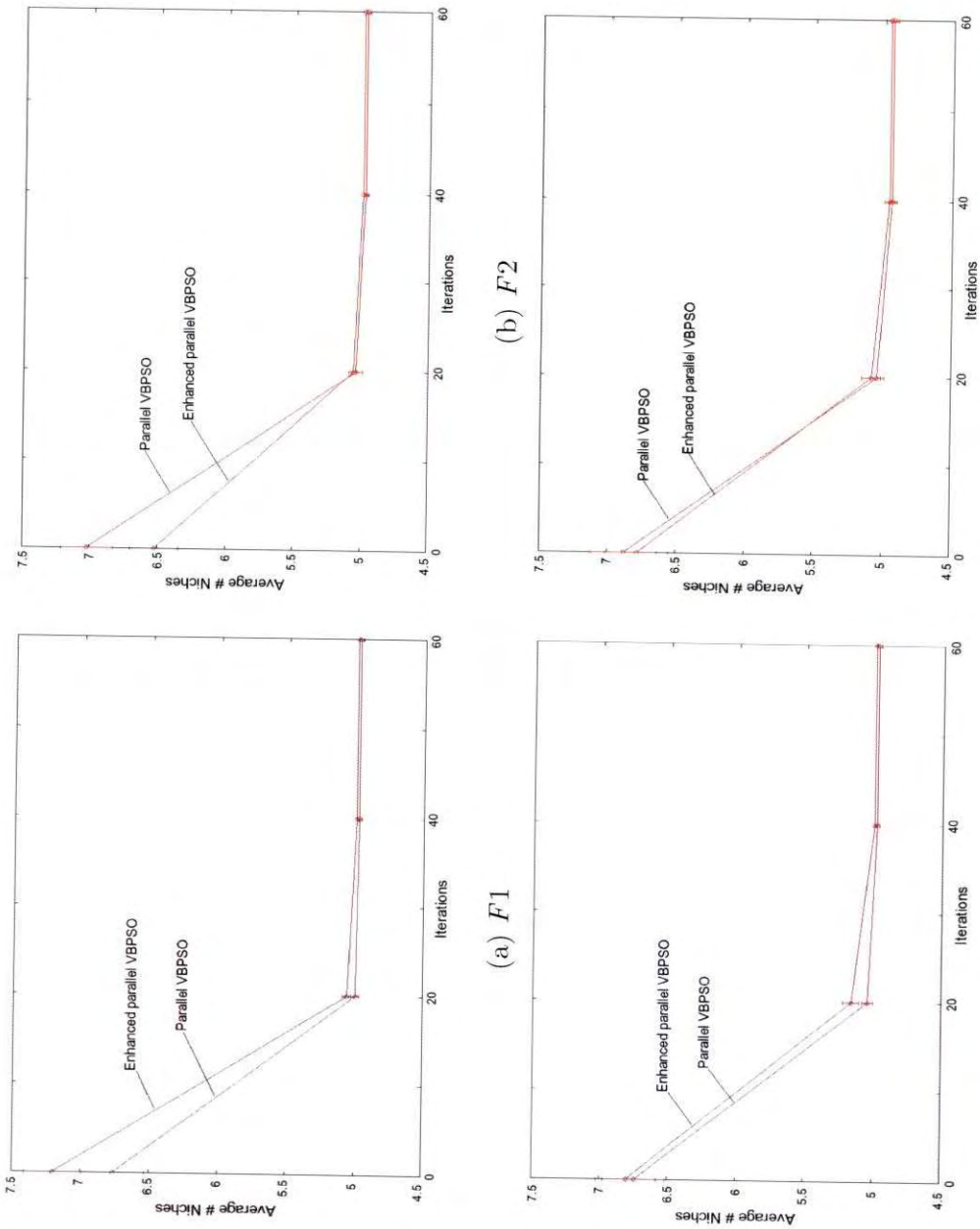
The parallel and enhanced parallel VBPSO incorporate a merging procedure to merge subswarms that converge on the same optimum. To observe the merging process, the number of niches was recorded after establishing the initial niches, as well as after every following 20 iterations. Table 6.29 summarizes these results for functions $F1$ to $F4$. Tables 6.30, 6.31, 6.32 and 6.33 present results of the niche merging process for seven two-dimensional functions, for the parallel VBPSO as well as the enhanced parallel vector-based PSO. According to the decision reached in the previous section, Sobol sequences were used to distribute particles uniformly over the search space, and the number of iterations was set to 500. Figures 6.9, 6.10 and 6.11 show graphs of the number of niches plotted against the number of iterations. For each function, graphs of the parallel and the enhanced parallel vector-based PSO are displayed on the same axes.

Results show that the final number of niches were established early during a run. For the one-dimensional functions, all niches have merged after 40 iterations, while the number of iterations differ for the two-dimensional functions. Variations in the number of initial niches can be ascribed to the stochastic nature of the PSO algorithm, as the parallel and enhanced parallel VBPSO versions identify niches in exactly the same way. Variations in merging time are in line with variations in the number of initial niches.

Tracking the number and positions of niches while vector-based PSO is converging, yielded a number of interesting observations. For functions $F1$, $F2$ and $F4$, niches merged faster when the enhanced parallel VBPSO was used, while for function $F3$, niches merged faster when the parallel version of the algorithm was used. For all two-dimensional functions, niches merged faster when the enhanced parallel VBPSO was used. For the Griewank, Ursem F1 and Rastrigin functions, the difference was negligible. The difference between the merging behaviour of the two algorithms was more pronounced in the case of the Himmelblau, Ackley, Ursem F3 and six hump camel functions where niches are more asymmetrically shaped. Therefore, the strategy used by the enhanced parallel VBPSO to contain particles in niches during optimization, resulted in faster merging, as well as a higher success rate. Two of the functions that used the parallel version of VBPSO, namely Ursem F3 and the six hump camel function, showed niches converging during the later stages of a run. These results showed that the parallel VBPSO may lose niches after all subswarms have converged to the required number of niches, while the enhanced parallel VBPSO did not.

Table 6.29: Merging of niches for one-dimensional functions $F1$ to $F4$

Iterations	Average number of niches							
	Parallel VBPSO				Enhanced parallel VBPSO			
	F1	F2	F3	F4	F1	F2	F3	F4
0	6.76 ± 0.23	7.02 ± 0.19	6.74 ± 0.23	6.88 ± 0.24	7.2 ± 0.21	6.52 ± 0.18	6.8 ± 0.22	6.78 ± 0.23
20	5 ± 0.03	5.04 ± 0.05	5.04 ± 0.02	5.04 ± 0.05	5.06 ± 0.03	5.06 ± 0.03	5.16 ± 0.06	5.08 ± 0.07
40	4.98 ± 0.02	4.98 ± 0.02	4.98 ± 0.02	4.94 ± 0.03	5 ± 0	5 ± 0	5 ± 0	4.96 ± 0.04
60	4.98 ± 0.02	4.98 ± 0.02	4.98 ± 0.02	4.94 ± 0.03	5 ± 0	5 ± 0	5 ± 0	4.96 ± 0.04



(a) $F1$ (b) $F2$ (c) $F3$ (d) $F4$

Figure 6.9: Merging of one-dimensional functions

Table 6.30: Merging of niches using the parallel VBPSO - part 1

Iterations	Average number of niches			
	Himmelblau	Griewank	Rastrigin	Ackley
0	13.68 ± 0.38	14.76 ± 0.46	23.58 ± 0.54	31.06 ± 0.63
20	9.26 ± 0.46	10.5 ± 0.26	15.94 ± 0.35	12.26 ± 0.23
40	5.44 ± 0.17	6.28 ± 0.16	10.42 ± 0.15	9.1 ± 0.11
60	4.42 ± 0.11	5.7 ± 0.12	9.36 ± 0.09	8.84 ± 0.08
80	4.16 ± 0.07	5.54 ± 0.1	9.26 ± 0.08	8.72 ± 0.08
100	4.14 ± 0.07	5.54 ± 0.1	9.12 ± 0.05	8.72 ± 0.08
120	4.08 ± 0.05	5.48 ± 0.1	9.02 ± 0.02	8.68 ± 0.07
140	4.06 ± 0.03	5.48 ± 0.1	9.02 ± 0.02	8.68 ± 0.07
160	4.06 ± 0.03	5.38 ± 0.1	9.02 ± 0.02	8.68 ± 0.07
180	4.04 ± 0.03	5.2 ± 0.07	9 ± 0	8.68 ± 0.07
200	4.04 ± 0.03	5.16 ± 0.07	9 ± 0	8.68 ± 0.07
220	4 ± 0	5.06 ± 0.04	9 ± 0	8.68 ± 0.07
240	4 ± 0	5.02 ± 0.02	9 ± 0	8.66 ± 0.07
260	4 ± 0	5 ± 0	9 ± 0	8.66 ± 0.07
280	4 ± 0	5 ± 0	9 ± 0	8.66 ± 0.07
300	4 ± 0	5 ± 0	9 ± 0	8.66 ± 0.07
320	4 ± 0	5 ± 0	9 ± 0	8.66 ± 0.07
340	4 ± 0	5 ± 0	9 ± 0	8.66 ± 0.07
360	4 ± 0	5 ± 0	9 ± 0	8.66 ± 0.07
380	4 ± 0	5 ± 0	9 ± 0	8.66 ± 0.07
400	4 ± 0	5 ± 0	9 ± 0	8.66 ± 0.07
420	4 ± 0	5 ± 0	9 ± 0	8.66 ± 0.07
440	4 ± 0	5 ± 0	9 ± 0	8.66 ± 0.07
460	4 ± 0	5 ± 0	9 ± 0	8.66 ± 0.07
480	4 ± 0	5 ± 0	9 ± 0	8.66 ± 0.07
500	4 ± 0	5 ± 0	9 ± 0	8.66 ± 0.07

Table 6.31: Merging of niches using the parallel VBPSO - part 2

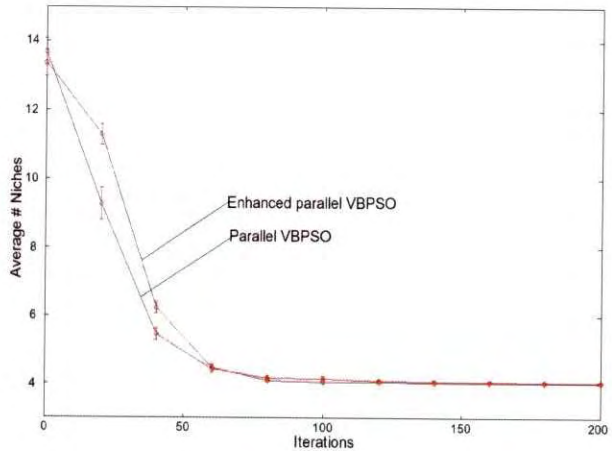
Iterations	Average number of niches		
	Ursem $F1$	Ursem $F3$	Six hump camel
0	11.24 ± 0.45	21.86 ± 0.37	24.18 ± 0.56
20	2.98 ± 0.15	11.26 ± 0.25	9.6 ± 0.18
40	2.1 ± 0.04	7.06 ± 0.22	6.3 ± 0.11
60	2 ± 0	5.52 ± 0.16	5.76 ± 0.08
80	2 ± 0	4.78 ± 0.12	5.64 ± 0.08
100	2 ± 0	4.38 ± 0.11	5.6 ± 0.08
120	2 ± 0	4.24 ± 0.07	5.6 ± 0.08
140	2 ± 0	4.16 ± 0.06	5.6 ± 0.08
160	2 ± 0	4.1 ± 0.05	5.6 ± 0.08
180	2 ± 0	4.1 ± 0.05	5.58 ± 0.08
200	2 ± 0	4.06 ± 0.03	5.58 ± 0.08
220	2 ± 0	4.04 ± 0.03	5.58 ± 0.08
240	2 ± 0	4.02 ± 0.02	5.58 ± 0.08
260	2 ± 0	4.02 ± 0.02	5.58 ± 0.08
280	2 ± 0	4.02 ± 0.02	5.58 ± 0.08
300	2 ± 0	4 ± 0	5.58 ± 0.08
320	2 ± 0	4 ± 0	5.58 ± 0.08
340	2 ± 0	4 ± 0	5.58 ± 0.08
360	2 ± 0	4 ± 0	5.58 ± 0.08
380	2 ± 0	4 ± 0	5.58 ± 0.08
400	2 ± 0	4 ± 0	5.58 ± 0.08
420	2 ± 0	3.98 ± 0.02	5.58 ± 0.08
440	2 ± 0	3.98 ± 0.02	5.58 ± 0.08
460	2 ± 0	3.98 ± 0.02	5.58 ± 0.08
480	2 ± 0	3.98 ± 0.02	5.56 ± 0.08
500	2 ± 0	3.98 ± 0.02	5.56 ± 0.08

Table 6.32: Merging of niches using the enhanced parallel VBPSO - part 1

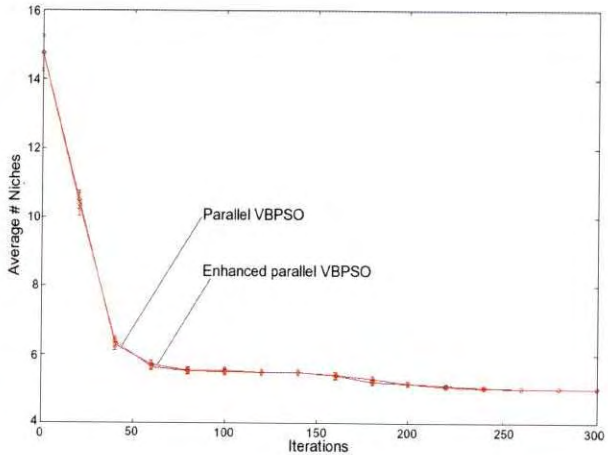
Iterations	Average number of niches			
	Himmelblau	Griewank	Rastrigin	Ackley
0	13.36 ± 0.39	14.74 ± 0.53	25.68 ± 0.52	27.56 ± 0.56
20	11.28 ± 0.3	10.36 ± 0.33	14.52 ± 0.27	11.76 ± 0.2
40	6.22 ± 0.16	6.36 ± 0.16	9.5 ± 0.1	9.14 ± 0.05
60	4.46 ± 0.09	5.64 ± 0.11	9.02 ± 0.02	9.06 ± 0.03
80	4.08 ± 0.04	5.52 ± 0.1	9 ± 0	$9. \pm 0$
100	4.04 ± 0.03	5.5 ± 0.1	9 ± 0	9 ± 0
120	4.04 ± 0.03	5.48 ± 0.1	9 ± 0	9 ± 0
140	4.02 ± 0.02	5.48 ± 0.1	9 ± 0	9 ± 0
160	4.02 ± 0.02	5.4 ± 0.1	9 ± 0	9 ± 0
180	4.02 ± 0.02	5.3 ± 0.08	9 ± 0	9 ± 0
200	4.02 ± 0.02	5.16 ± 0.07	9 ± 0	9 ± 0
220	4.02 ± 0.02	5.1 ± 0.05	9 ± 0	9 ± 0
240	4.02 ± 0.02	5.04 ± 0.04	9 ± 0	9 ± 0
260	4.02 ± 0.02	5 ± 0	9 ± 0	9 ± 0
280	4.02 ± 0.02	5 ± 0	9 ± 0	9 ± 0
300	4 ± 0	5 ± 0	9 ± 0	9 ± 0
320	4 ± 0	5 ± 0	9 ± 0	9 ± 0
340	4 ± 0	5 ± 0	9 ± 0	9 ± 0
360	4 ± 0	5 ± 0	9 ± 0	9 ± 0
380	4 ± 0	5 ± 0	9 ± 0	9 ± 0
400	4 ± 0	5 ± 0	9 ± 0	9 ± 0
420	4 ± 0	5 ± 0	9 ± 0	9 ± 0
440	4 ± 0	5 ± 0	9 ± 0	9 ± 0
460	4 ± 0	5 ± 0	9 ± 0	9 ± 0
480	4 ± 0	5 ± 0	9 ± 0	9 ± 0
500	4 ± 0	5 ± 0	9 ± 0	9 ± 0

Table 6.33: Merging of niches using the enhanced parallel VBPSO - part 2

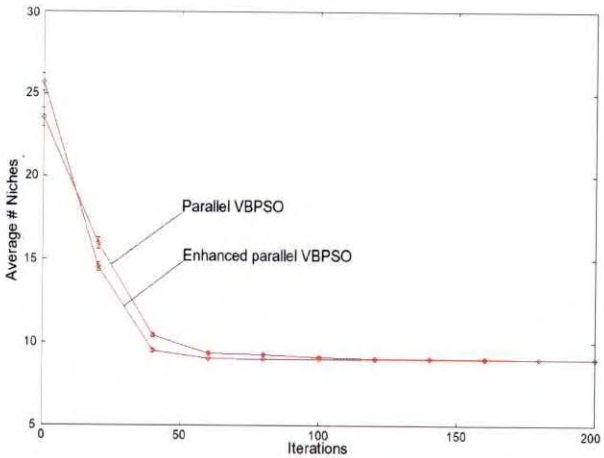
Iterations	Average number of niches		
	Ursem $F1$	Ursem $F3$	Six hump camel
0	11.54 ± 0.53	22.42 ± 0.42	22.56 ± 0.52
20	3.16 ± 0.27	11.16 ± 0.25	12.06 ± 0.31
40	2.06 ± 0.04	6.1 ± 0.17	7.22 ± 0.13
60	2 ± 0	4.44 ± 0.11	6.04 ± 0.05
80	2 ± 0	4.1 ± 0.04	5.96 ± 0.03
100	2 ± 0	4.08 ± 0.04	5.96 ± 0.03
120	2 ± 0	4.02 ± 0.02	5.96 ± 0.03
140	2 ± 0	4 ± 0	5.96 ± 0.03
160	2 ± 0	4 ± 0	5.96 ± 0.03
180	2 ± 0	4 ± 0	5.96 ± 0.03
200	2 ± 0	4 ± 0	5.96 ± 0.03
220	2 ± 0	4 ± 0	5.96 ± 0.03
240	2 ± 0	4 ± 0	5.96 ± 0.03
260	2 ± 0	4 ± 0	5.96 ± 0.03
280	2 ± 0	4 ± 0	5.96 ± 0.03
300	2 ± 0	4 ± 0	5.96 ± 0.03
320	2 ± 0	4 ± 0	5.96 ± 0.03
340	2 ± 0	4 ± 0	5.96 ± 0.03
360	2 ± 0	4 ± 0	5.96 ± 0.03
380	2 ± 0	4 ± 0	5.96 ± 0.03
400	2 ± 0	4 ± 0	5.96 ± 0.03
420	2 ± 0	4 ± 0	5.96 ± 0.03
440	2 ± 0	4 ± 0	5.96 ± 0.03
460	2 ± 0	4 ± 0	5.96 ± 0.03
480	2 ± 0	4 ± 0	5.96 ± 0.03
500	2 ± 0	4 ± 0	5.96 ± 0.03



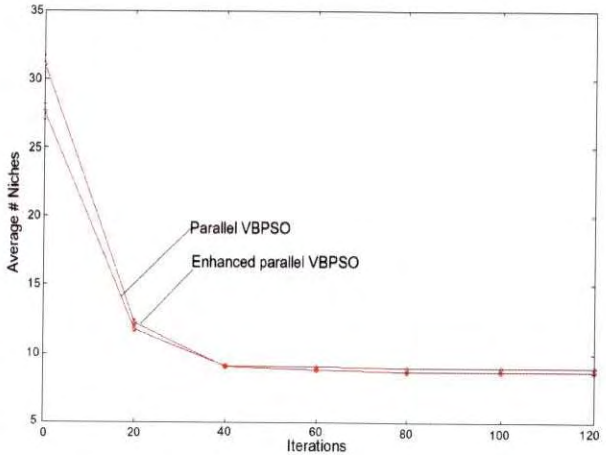
(a) Himmelblau



(b) Griewank

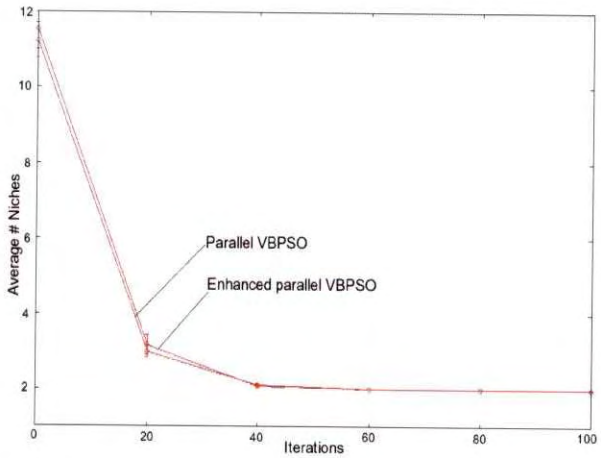


(c) Rastrigin

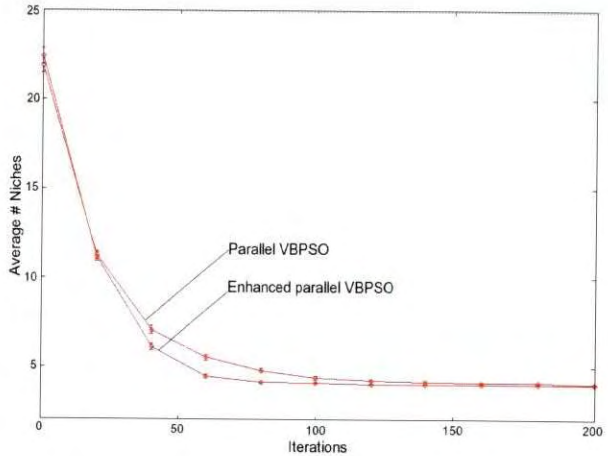


(d) Ackley

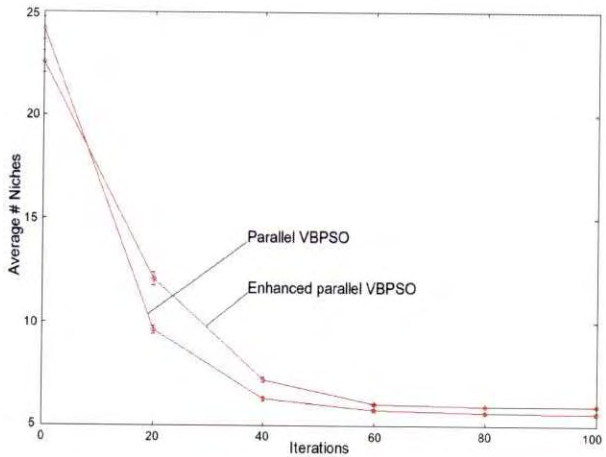
Figure 6.10: Merging of two-dimensional functions - part 1



(e) Ursem $F1$



(f) Ursem $F3$



(g) Six Hump Camel

Figure 6.11: Merging of two-dimensional functions - part 2

6.4.3 Comparing the parallel and enhanced parallel algorithms

This section compares the performance of the parallel vector-based PSO and the enhanced parallel vector-based PSO. Test results of the various functions were presented in section 6.3.1. To compare the algorithms, these results are summarized in Tables 6.34 and 6.35.

The parallel vector-based PSO was compared to the enhanced vector-based PSO for all the functions tested in section 6.4. Table 6.34 shows exactly the same outcomes for parallel and enhanced parallel VBPSO. Although individual differences occur, Table 6.35 shows that the enhanced parallel VBPSO performed better than the parallel VBPSO. Results of the Mann-Whitney U test reported in Table 6.36 show no significant difference between the performances of the algorithms for functions $F1$ to $F4$. Table 6.37 shows a significant difference between the algorithms only in the case of the Ackley and six-hump camel functions. These functions have a number of small niches that may easily merge with larger niches if particles leave the niche. The enhanced parallel VBPSO contains a strategy to prevent such occurrences and consequently performs better for such functions.

The development of the family of vector-based particle swarm optimizers culminates in the enhanced parallel vector-based PSO using Sobol sequences as a random number generator. Hence, the latter is used for all further work and will be referred to as the vector-based PSO (VBPSO).

Table 6.34: Average % optima located for functions $F1$ to $F4$

Function	Parallel VBPSO		Enhanced parallel VBPSO	
	Average ♯ solutions	Success rate	Average ♯ solutions	Success rate
$F1$	5 ± 0	100%	5 ± 0	100%
$F2$	4.98 ± 0.02	99.6%	5 ± 0	100%
$F3$	4.98 ± 0.02	99.6%	5 ± 0	100%
$F4$	4.94 ± 0.03	98.8%	4.96 ± 0.03	99.2%
Average	4.98	99.5%	4.99	99.8%

Table 6.35: Summary of VBPSO success rates for two-dimensional functions

Function	Parallel VBPSO	Enhanced parallel VBPSO
Himmelblau	100%	100%
Griewank	100%	100%
Rastrigin	100%	100%
Ackley	96.45%	100%
Ursem F1	100%	100%
Ursem F3	99.75%	100%
Six hump camel	93%	98.5%
Average	98.46%	99.79%

Table 6.36: Comparing algorithms for functions $F1$ to $F4$

Function	Significance level	Enhanced parallel vs parallel VBPSO			
		Rank 1	Rank 2	Z	H_0/H_1
$F1$	0.05	2.53E+03	2.53E+03	0.00	H_0
	0.01				H_0
$F2$	0.05	2.55E+03	2.50E+03	-1.72E-01	H_0
	0.01				H_0
$F3$	0.05	2.55E+03	2.50E+03	-1.72E-01	H_0
	0.01				H_0
$F4$	0.05	2.55E+03	2.50E+03	-1.72E-01	H_0
	0.01				H_0

Table 6.37: Comparing the enhanced parallel VBPSO to the parallel VBPSO

Function	Significance	Enhanced parallel VBPSO versus parallel VBPSO			
		ΣR_1	ΣR_2	Z	H_0/H_1
Himmelblau	0.05	2.53E+03	2.53E+03	0.00	H_0
	0.01				H_0
Griewank	0.05	2.53E+03	2.53E+03	0.00	H_0
	0.01				H_0
Rastrigin	0.05	2.53E+03	2.53E+03	0.00	H_0
	0.01				H_0
Ursem $F1$	0.05	2.53E+03	2.53E+03	0.00	H_0
	0.01				H_0
Ursem $F3$	0.05	2.55E+03	2.50E+03	-1.72E-01	H_0
	0.01				H_0
Ackley	0.05	2.98E+03	2.08E+03	-3.10E+00	H_1
	0.01				H_1
Six hump camel	0.05	3.00E+03	2.50E+03	-3.28E+00	H_1
	0.01				H_1

6.5 Analysis of the vector-based particle swarm optimizer

This section presents further results of experiments with the vector-based particle swarm optimizer. Only the enhanced version of the algorithm, using Sobol sequences to distribute initial particles evenly across the search space and executing 500 iterations, has been used, since it showed to produce the best results. This algorithm will now be referred to as the vector-based particle swarm optimizer (VBPSO).

Three aspects of the VBPSO have been investigated:

- The behaviour of the VBPSO on more complicated function landscapes.
- Sensitivity of the vector-based PSO to changes in the granularity parameter.
- Scalability of the vector-based PSO and the relationship between the initial swarm size and the number of solutions.

6.5.1 Analysis of the VBPSO on additional functions

This subsection presents results of experiments with the VBPSO on a number of additional functions in demarcated regions of the search space. These functions were specifically selected to test the behaviour of the algorithm on more complicated function landscapes. Features of each function are discussed and results for specific swarm sizes and granularity settings are presented.

Taking the size of the search space, the number of optima that can be expected, and settings for previous experiments into account, granularity settings and initial swarm sizes were estimated for these experiments. These settings were not claimed to be optimal. The sensitivity of the vector-based PSO to granularity is discussed in section 6.5.2, and the scalability of the algorithm is discussed in section 6.5.3.

The Styblinksi-Tang function:

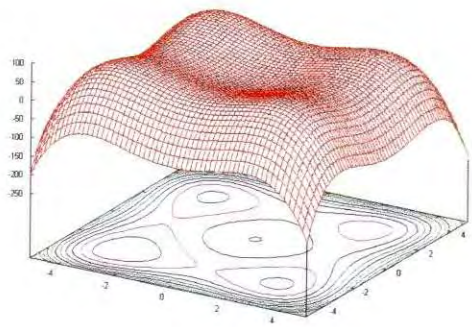


Figure 6.12: The Styblinski-Tang function showing maxima

The Styblinski-Tang function is defined as

$$f(\mathbf{x}) = \frac{1}{2}\sum_{i=1}^{2}(x_i^4 - 16x_i^2 + 5x_i) \tag{6.15}$$

and illustrated in Figure 6.12 in the range $x_1, x_2 \in [-5.0, 5.0]$. The function is defined in two dimensions with one global minimum and three local optima of which the fitness differ slightly. Niches are flat and broad, and the function has been selected to test the ability of the algorithm to optimize such niches. Positions of optima and function values at these positions are presented in Table 6.38.

The function was tested with an initial swarm size of 30 and a granularity of 0.5. Results are presented in Table 6.40. Results show a 100% success rate for 50 independent runs of the algorithm, while the derivatives indicate high quality solutions. Therefore, the function was not tested with larger swarm sizes.

Table 6.38: Optima of the Styblinksi-Tang function

Locations of optima		Fitness
x_1	x_2	$f(x_1, x_2)$
-2.9	-2.9	-78.33
2.75	-2.9	-64.2
-2.9	2.75	-64.2
2.75	2.75	-50.06

The Bird function:

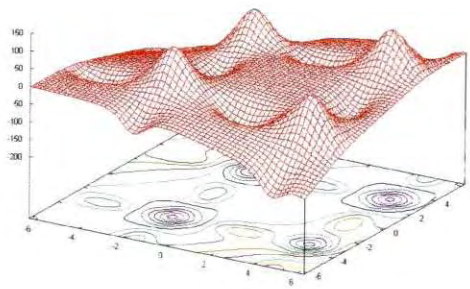


Figure 6.13: The Bird function showing maxima

The Bird function is defined as

$$f(\mathbf{x_1}, \mathbf{x_2}) = \sin(x_1)e^{(1-\cos(x_2))^2} + \cos(x_2)e^{(1-\sin(x_1))^2} + (x_1 - x_2)^2 \tag{6.16}$$

and illustrated in Figure 6.13 in the range $x_1, x_2 \in [-2\pi, 2\pi]$.

The Bird function has four well-defined minima as well as two very small depressions in the landscape. This function has been selected to test the ability of the algorithm to identify large (major solutions) as well as small niches (minor solutions) and locate the corresponding optima. Table 6.39 shows the positions and fitness of the optima. Table 6.41 presents results of the experiments, calculated separately for major and minor optima. The granularity was set to an estimated value of 30° or $\pi/6$.

To test the effect of different swarm sizes, experiments were run with initial swarm sizes of 30 and 50 particles. Results show that the algorithm had a 100% success rate for the four large niches for swarm sizes of 30 as well as 50 particles, while the minor optima were only located 76% of the time for an initial swarm size of 30 and 88% of the time when the swarm size was extended to 50 particles. These results confirm that prominent and well-defined optima are located easier, while the success rate for small niches increases for larger initial swarms sizes.

Table 6.39: Optima of the Bird function

Solution type	Locations of optima		Fitness
	x_1	x_2	$f(x_1, x_2)$
Major	-90.65	-179.35	-106.77
	269.35	180.65	-106.77
	-88.04	178.04	-87.31
	266.72	-176.72	-48.41
Minor	-321.88	-308.12	1.49
	38.12	51.88	1.49

Table 6.40: VBPSO results for Styblinksi-Tang function

♯ Particles	Granularity	Average ♯ evaluations	Average ♯ solutions	Average $f'(x)$	Average $f'(y)$	Success rate
30	0.5	24417 ± 461	4 ± 0	3.83E-06 ± 4.07E-07	3.83E-06 ± 4.07E-07	100%

Table 6.41: VBPSO results for the Bird function

♯ Particles	Granularity	Average ♯ evaluations	Optima	Average ♯ solutions	Average $f'(x)$	Average $f'(y)$	Success rate
30	$\pi/6$	29933 ± 499	Major	4 ± 0	1.44E-06 ± 1.46E-07	1.31E-06 ± 1.23E-07	100%
			Minor	1.52 ± 0.09	1.77E-08 ± 1.49E-09	1.94E-08 ± 1.77E-09	76%
50	$\pi/6$	47357 ± 644	Major	4 ± 0	1.30E-06 ± 1.19E-07	1.32E-06 ± 1.21E-07	100%
			Minor	1.76 ± 0.07	1.87E-08 ± 1.39E-09	2.68E-08 ± 1.27E-08	88%

The generalized Schwefel function 2.26:

The function is defined as:

$$f(\mathbf{x}) = -\sum_{i=1}^{2}(x_i \sin(\sqrt{|x_i|})) \tag{6.17}$$

Figure 6.14 illustrates the generalized Schwefel function 2.26 in the range $x_1, x_2 \in [-65, 100]$.

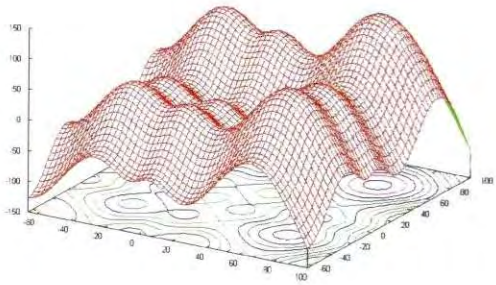


Figure 6.14: The generalized Schwefel function 2.26 showing maxima

The generalized Schwefel function 2.26 has nine optima of different heights in the given range. The four most prominent niches are situated near to the four corners of the defined search space, and are referred to as major solutions for the purposes of this experiment. The other five optima are referred to as minor solutions, as one of these is very small, while each of the other four is situated very near to a major solution, and would therefore be difficult to locate. Table 6.42 lists the positions and fitnesses of the optima.

Table 6.44 presents the results of the experiments. As in the case of the Bird function, the major solutions have a success rate of 100%, while the minor solutions are found 82.8% and 91.6% of the time for 50 and 80 particles respectively. For this function, partial derivatives could not be calculated, since the function, that contains an absolute value, is not continuous. However, all optimal positions were similar to the positions listed in Table 6.42 if compared based on two decimal positions.

Table 6.42: Optima of the generalized Schwefel function 2.26

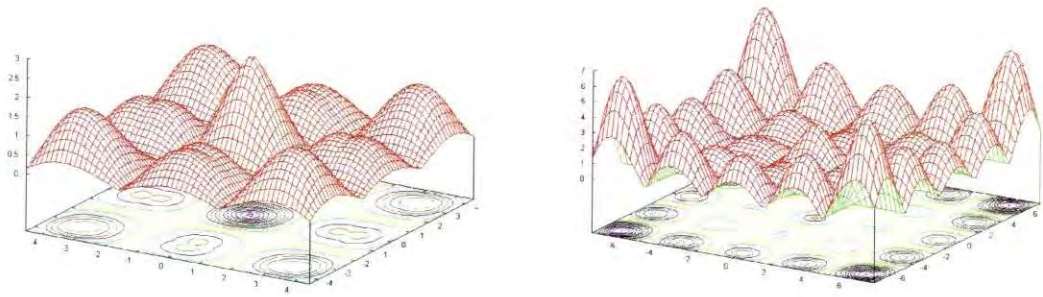
Solution type	Locations of optima		Fitness
	x_1	x_2	$f(x_1, x_2)$
Major	65.55	65.55	-127.7
	65.55	25.88	-87.72
	-25.88	65.55	-87.72
	-25.88	-25.88	-48.17
Minor	5.24	65.55	-67.58
	65.55	5.24	-67.58
	-25.88	5.24	-28.03
	5.24	-25.88	-28.03
	5.24	5.24	-7.89

The tabular holder function:

The function is defined as:

$$f(\mathbf{x_1}, \mathbf{x_2}) = -\left|\cos(x_1)\cos(x_2)e^{|1-((x_1^2+x_2^2)^{0.5})/\pi|}\right| \tag{6.18}$$

Figure 6.15 illustrates the tabular holder function in the range $x_1, x_2 \in [-4.5, 4.5]$ and in the range $x_1, x_2 \in [-7.5, 7.5]$.



(a) (b)

Figure 6.15: The tabular holder function

The tabular holder function has optima of different shapes and sizes, depending on the search space. Some of these peaks have two optima very near to one another as illustrated in Figure 6.15(a). Experiments have been run in the range $x_1, x_2 \in [-4.5, 4.5]$ to test the ability

of the algorithm to locate optima situated very near to one another. For the purpose of this experiment, larger niches are referred to as major solutions, while niches that overlap to such an extent that they have the appearance of peaks with two optima, are referred to as minor solutions. This distinction is made because of the difficulty of locating optima that are very near to one another. Table 6.43 lists these optimal positions and their corresponding fitnesses. Table 6.45 presents the average number of major and minor solutions located with an initial swarm size of 60 particles. Because the function is not continuous, partial derivatives could not be obtained, but as in the case of the Schwefel function, an average offline error of 0 was found for all results reported to two decimals. As expected, the success rate of he algorithm was higher for the major solutions, namely 98%, while the minor solutions yielded a success rate of 81.75%.

Table 6.43: Optima of the tabular holder function in the range $x_1, x_2 \in [-4.5, 4.5]$

Range	Solution type	Locations of optima		Fitness
		x_1	x_2	$f(x_1, x_2)$
$x_1, x_2 \in [-4.5, 4.5]$	Major	-3.36	-3.36	-1.59
		-3.36	3.36	-1.59
		0	0	-2.72
		3.36	-3.36	-1.59
		3.36	3.36	-1.59
	Minor	-3.45	0	-1.05
		-2.83	0	-1.05
		0	-3.45	-1.05
		0	-2.83	-1.05
		0	3.45	-1.05
		0	2.83	-1.05
		3.45	0	-1.05
		2.83	0	-1.05

Table 6.44: VBPSO results for generalized Schwefel function 2.26

♯ Particles	Granularity	Average ♯ evaluations	Average ♯ major solutions	Success rate	Average ♯ minor solutions	Success rate
50	5	51613 ± 762	4 ± 0	100%	4.14 ± 0.12	82.8%
80	5	73098 ± 865	4 ± 0	100%	4.58 ± 0.08	91.6%

Table 6.45: VBPSO results for tabular holder function

♯ Particles	Granularity	Average ♯ evaluations	Average ♯ major solutions	Success rate	Average ♯ minor solutions	Success rate
60	0.3	48762 ± 695	4.9 ± 0.04	98%	6.54 ± 0.14	81.75%

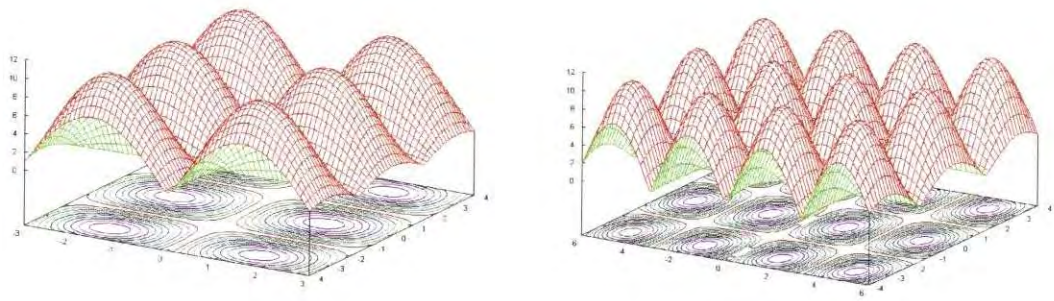
The tube holder function:

The function is defined as

$$f(\mathbf{x_1}, \mathbf{x_2}) = -\left|\sin(x_1)\cos(x_2)e^{|\cos((x_1^2+x_2^2)/200)|}\right| \tag{6.19}$$

Figure 6.16 illustrates the Tube holder function in the range $x_1 \in [-3, 3]$ and $x_2 \in [-4, 4]$, and in the range $x_1 \in [-6, 6]$ and $x_2 \in [-4, 4]$. Optima with slightly differing fitnesses are distributed evenly across the search space. The function has been selected to assess the ability of the vector-based PSO to locate optima when the size of the search space is increased. The algorithm was tested for search spaces containing 6 and 12 optima each as illustrated in Figure 6.16(a) and 6.16(b). Locations and fitnesses of these optima in the range $x_1 \in [-3, 3]$ and $x_2 \in [-4, 4]$ is listed in Table 6.46, while optimal positions and fitnesses in the range $x_1 \in [-6, 6]$ and $x_2 \in [-4, 4]$ are listed in Table 6.47. In addition, the algorithm was also tested for a search space containing 20 optima where $x_1 \in [-6, 6]$ and $x_2 \in [-8, 8]$.

Table 6.48 presents results of the experiments for each of the search spaces. An average success rate of more than 99% is reported in each case. To conclude, these experiments show that the vector-based PSO scales successfully to higher multi-modality when testing the Tube holder function.



(a) (b)

Figure 6.16: The tube holder function.

Table 6.46: Optima of the tube holder function in the range [-3,4] and [3,4]

Solution type	Locations of optima		Fitness
	x_1	x_2	$f(x_1, x_2)$
	-1.57	-3.14	-10.85
	-1.57	0	-10.87
	-1.57	3.14	-10.85
	1.57	-3.14	-10.85
	1.57	0	-10.87
	1.57	3.14	-10.85

Table 6.47: Optima of the tube holder function in the range [-3,4] and [3,4]

Solution type	Locations of optima		Fitness
	x_1	x_2	$f(x_1, x_2)$
	-1.57	-3.14	-10.85
	-1.57	0	-10.87
	-1.57	3.14	-10.85
	1.57	-3.14	-10.85
	1.57	0	-10.87
	1.57	3.14	-10.85
	-4.71	-3.14	-10.73
	-4.71	0	-10.81
	-4.71	3.14	-10.73
	4.71	-3.14	-10.73
	4.71	0	-10.81
	4.71	3.14	-10.73

Table 6.48: VBPSO results for tube holder function

Range (♯ optima)	♯ Particles	Granularity	Average ♯ evaluations	Average ♯ solutions	Average $f'(x)$	Average $f'(y)$	Success rate
$x_1 \in [-3, 3]$, $x_2 \in [-4, 4]$ (6)	30	0.5	23509 ± 472	5.98 ± 0.02	0.0038 ± 0.0003	0.0070 ± 0.0007	99.67%
$x_1 \in [-6, 6]$, $x_2 \in [-4, 4]$ (12)	50	0.5	42204 ± 606	11.96 ± 0.03	0.020 ± 0.002	0.012 ± 0.002	99.67%
$x_1 \in [-6, 6]$, $x_2 \in [-8, 8]$ (20)	80	0.5	72142 ± 678	19.94 ± 0.03	0.030 ± 0.004	0.041 ± 0.005	99.7%

6.5.2 Sensitivity of the vector-based PSO to granularity

The previous sections showed that an additional parameter, called the *granularity*, has to be introduced into the VBPSO if it becomes necessary to merge niches when a number of subswarms are optimized in parallel. One of the strengths of the VBSPO is the ability to optimize irregularly shaped niches. In such cases not all particles are included when niches are formed. The remaining particles form false or extra niches on the boundaries of existing niches. The subswarms occupying these niches will eventually merge with the subswarm having the fittest neighbourhood best value. In the experiments done so far, a value for the granularity was chosen by taking into account the size of the search space as well as the number of optima that can be expected. Intuitively it can be argued that interniche distances indicate an upper bound on the size of the granularity.

To test the influence of different granularities for a range of functions, three two-dimensional functions were used, namely the Himmelblau, Griewank and Rastrigin functions. Sets of 50 runs were conducted with the granularity set to different values. As a starting point, granularity values were used that yielded good results in previous experiments: 0.5 for Himmelblau and Griewank, and 0.1 for Rastrigin.

Results for the three functions are summarized in the following sections.

The Himmelblau function

The Himmelblau function was tested for granularity values ranging from 0.1 to 10.0. The average number of optima located over 50 runs for various values of g is summarized in Table 6.49. Figure 6.17 illustrates the variation of the number of optima located with increase in the granularity. In order to clarify the role of interniche distances in granularity settings, all interniche distances in the Himmelblau function landscape are listed in Table 6.50. Figure 6.18 presents a graphical representation of these distances.

Discussion

The graph in Figure 6.17 shows a sharp decline in the number of optima found for granularity values of between 3 and 4, 6 and 7, and between 8 and 9. These values correspond roughly to the interniche distances as given in Table 6.50 and Figure 6.18. From these results it can be inferred that the VBPSO should find all four optima of the Himmelblau function given that the granularity has a value less than the smallest interniche distance. Formulated differently, the

Table 6.49: Testing granularity for the Himmelblau function

Granularity	Average # solutions	Granularity	Average # solutions
0.1	4 ± 0	5.0	3 ± 0
0.5	4 ± 0	5.5	2.88 ± 0.05
1.0	4 ± 0	6.0	2.72 ± 0.06
1.5	3.96 ± 0.03	6.5	1.88 ± 0.06
2.0	4 ± 0	7.0	1.88 ± 0.05
2.5	4 ± 0	7.5	1.72 ± 0.06
3.0	4 ± 0	8.0	1.58 ± 0.07
3.5	3.92 ± 0.04	8.5	1.4 ± 0.07
4.0	3 ± 0	9.0	1 ± 0
4.5	2.98 ± 0.02	10.0	1 ± 0

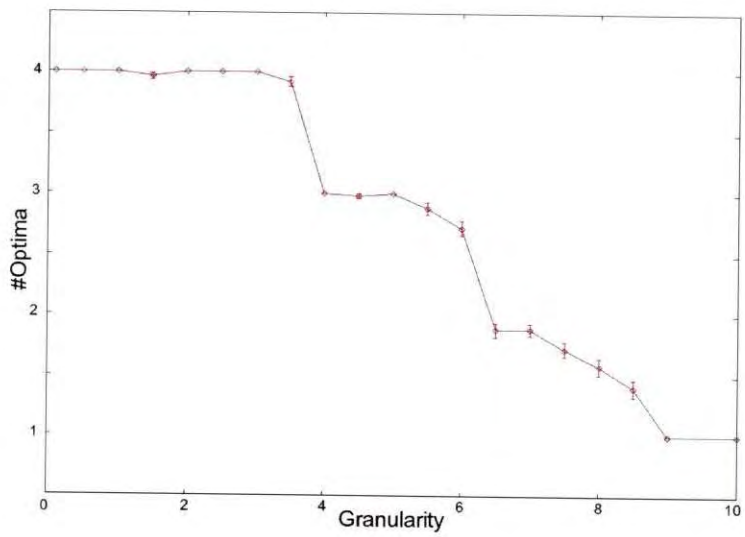


Figure 6.17: Number of optima vs. granularity for the Himmelblau function

VBPSO should find all optima where the interniche distances are larger than the granularity. However, if a function landscape and the interniche distances are unknown, it can be concluded that, for a specific granularity choice, the VBPSO should locate all optima where the smallest Euclidian distance between any two optima is less than the granularity.

Table 6.50: Interniche distances for the Himmelblau function

x_1	x_2	x_1	x_2	Interniche distance
-3.78	-3.28	-2.81	3.13	6.49
-3.78	-3.28	3.58	-1.85	7.50
-3.78	-3.28	3	2	8.59
-2.81	3.13	3.58	-1.85	8.10
-2.81	3.13	3	2	5.91
3.58	-1.85	3	2	3.89

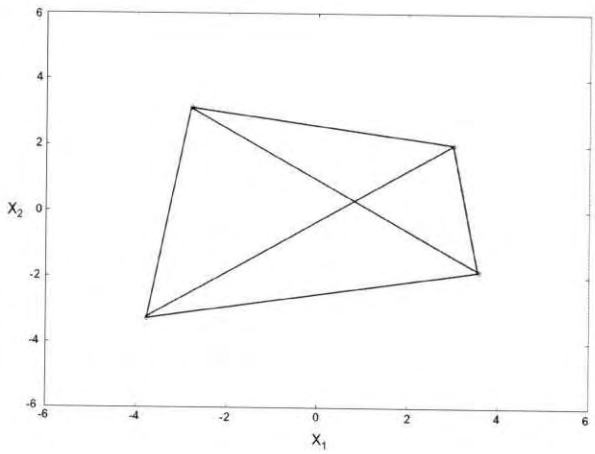


Figure 6.18: Interniche distances for the Himmelblau function

The Griewank function

The Griewank function was tested for granularity values ranging from 0.5 to 7.5. According to experiments reported in section 6.4, the Griewank function was tested with an estimated granularity value of 0.5 which yielded good results. Therefore, the granularity value was increased, starting at 0.5. The average number of optima located over 50 runs for a range of values of the granularity is summarized in Table 6.51. A graphical representation is given in Figure 6.19, which indicates the variation of the number of optima located with increase in the granularity.

Table 6.51: Testing granularity for the Griewank function

Granularity	Average # solutions	Granularity	Average # solutions
0.5	5 ± 0	6.5	1.58 ± 0.07
1.0	5 ± 0	7.0	1.46 ± 0.07
1.5	4.98 ± 0.02	7.5	1.44 ± 0.07
2.0	5 ± 0	8.0	1.52 ± 0.07
2.5	4.96 ± 0.03	8.5	1.48 ± 0.07
3.0	4.92 ± 0.04	9.0	1.6 ± 0.07
3.5	4.76 ± 0.06	9.5	1.56 ± 0.07
4.0	4.48 ± 0.07	10.0	1.64 ± 0.07
4.5	4.44 ± 0.1	10.5	1.57 ± 0.07
5.0	4.18 ± 0.1	11.0	1.5 ± 0.07
5.5	1.62 ± 0.08	11.5	1.42 ± 0.07
6.0	1.48 ± 0.07	12.0	1.38 ± 0.07

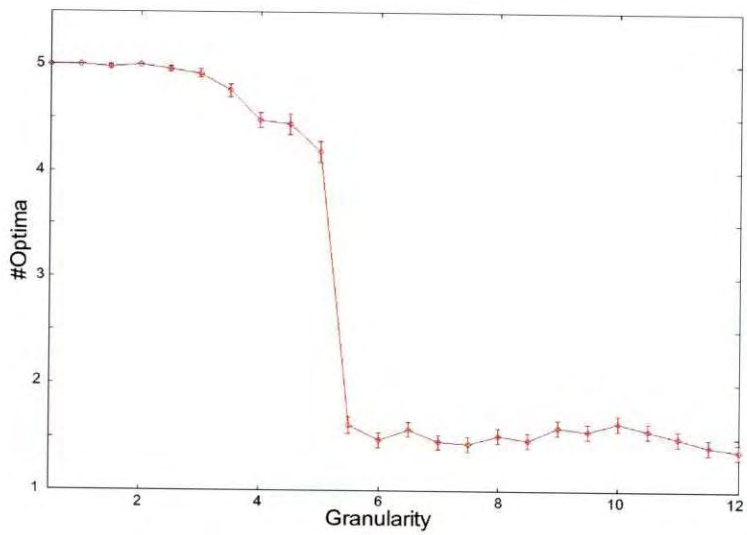


Figure 6.19: Number of optima vs. granularity for the Griewank function

Table 6.52: Interniche distances for the Griewank function

x_1	x_2	x_1	x_2	Interniche distance
0	0	-3.14	-4.44	5.44
0	0	-3.14	4.44	5.44
0	0	3.14	-4.44	5.44
0	0	3.14	4.44	5.44
-3.14	-4.44	-3.14	4.44	8.88
-3.14	-4.44	3.14	-4.44	6.28
-3.14	-4.44	3.14	4.44	10.87
3.14	-4.44	-3.14	4.44	10.87
3.14	-4.44	3.14	4.44	8.88
-3.14	4.44	3.14	4.44	6.28

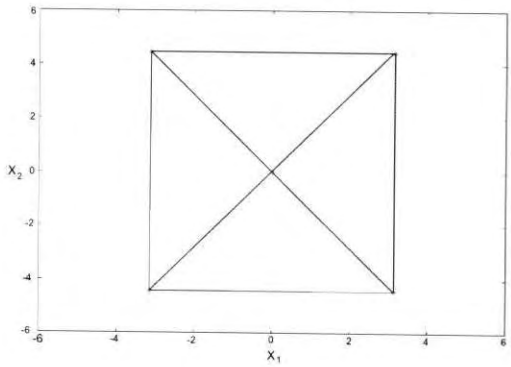


Figure 6.20: Interniche distances for the Griewank function

Discussion

Figure 6.19 shows a slight decline in the number of optima starting at a granularity value of 3, and a sharp decline for a granularity value of just more than 5. Table 6.52 and Figure 6.20 show that the interniche distances between all 5 optima and their nearest neighbours are similar to one another. For larger granularity values the number of optima found stabilizes around 1.3. When g becomes large, the algorithm found one or in some cases two optima. The above figures show that, taking the stochastic nature of PSO into account, the VBPSO locates all five optima most of the time if the granularity is less than the smallest interniche distance.

The Rastrigin function

The Rastrigin function was tested for granularity values ranging from 0.1 to 1.6. The search space is defined for $x_1 \in [-1.25, 1.25]$ and $x_2 \in [-1.25, 1.25]$, where the Rastrigin function has nine optima. The average number of optima found over 50 runs for a range of values of the granularity is summarized in Table 6.53. The variation of the number of optima found with increased granularity is illustrated in Figure 6.21.

Table 6.53: Testing granularity for the Rastrigin function

Granularity	Average # solutions	Granularity	Average # solutions
0.1	9 ± 0	0.9	7.62 ± 0.15
0.2	9 ± 0	1.0	4.14 ± 0.14
0.3	8.96 ± 0.03	1.1	3.24 ± 0.11
0.4	9 ± 0	1.2	3.32 ± 0.13
0.5	8.96 ± 0.03	1.3	3.52 ± 0.12
0.6	8.86 ± 0.05	1.4	2.88 ± 0.14
0.7	8.38 ± 0.08	1.5	1 ± 0
0.8	8.27 ± 0.10	1.6	1 ± 0

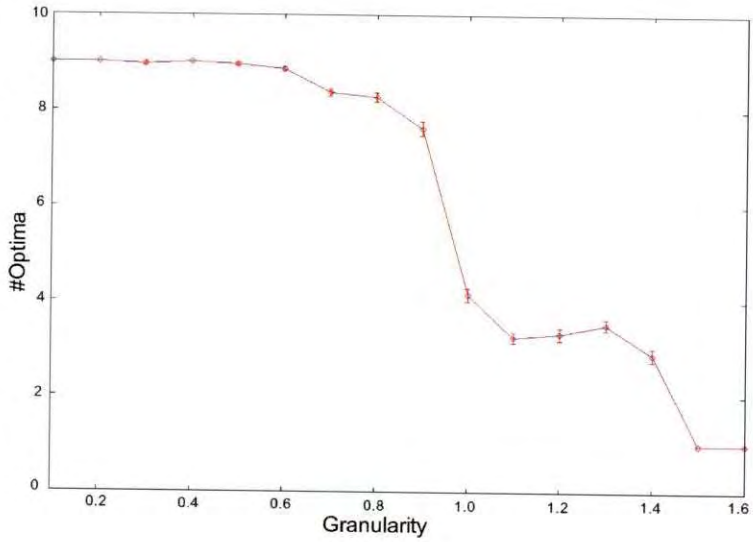


Figure 6.21: Number of optima vs. granularity for the Rastrigin function

Table 6.54: Interniche distances for the Rastrigin function.

x_1	x_2	x_1	x_2	Interniche distance
0	0	0	1	1
0	0	0	-1	1
0	0	1	0	1
0	0	-1	0	1
0	1	1	1	1
0	1	-1	1	1
0	-1	1	-1	1
0	-1	-1	-1	1
1	0	1	1	1
1	0	1	-1	1
-1	0	-1	1	1
-1	0	-1	-1	1
0	0	1	1	1.41
0	0	1	-1	1.41
0	0	-1	1	1.41
0	0	-1	-1	1.41
0	1	1	0	1.41
0	1	-1	0	1.41
0	-1	1	0	1.41
0	-1	-1	0	1.41

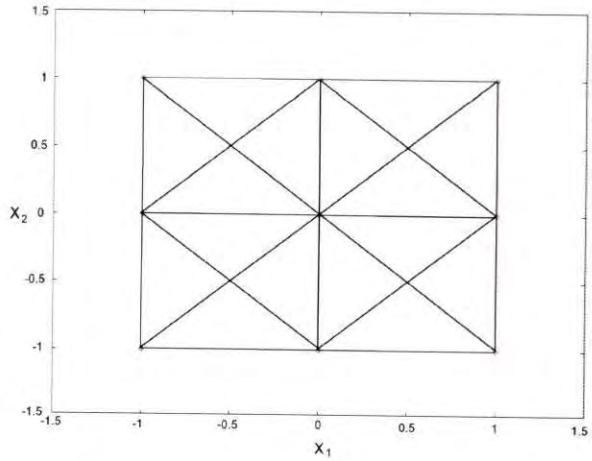


Figure 6.22: Interniche distances for the Rastrigin function

Discussion

For granularity values ranging from 0.1 to 0.9, the VBPSO finds most of the optima of the Rastrigin function, degrading slightly for higher values of the granularity. Again, the interniche distances as listed in Table 6.54 and illustrated in Figure 6.22 indicate upper boundaries for the granularity. Between granularity values of 0.9 and 1 the number of optima drops sharply, meaning that a number of adjacent niches have merged. Niches start merging when the granularity value approaches an interniche distance, which is 1 in this case. Another drop in the number of optima located occurs just before the granularity becomes 1.5. A number of interniche distances are equal to 1.41, which explains the decrease in the number of optima located. The same outcome is experienced as in the case of the previous two functions, namely that the granularity influences performance of the VBPSO algorithm in terms of the number of solutions that can be located.

Introducing a new parameter, the granularity, may be perceived as violating the principle of parsimony. However, for the VBPSO algorithm some value is required to facilitate subswarm merging. Experiments in this section showed that the granularity can be chosen from a wide range of values, provided it is smaller than the smallest interniche distance. Stated differently, the value of the granularity determines the smallest distance between niches with a reasonable expectation of being located.

6.5.3 Scalability of the vector-based PSO

Results obtained in previous sections showed that the vector-based PSO could potentially solve multi-modal optimization problems. However, a study of a niching algorithm will be incomplete if its ability to scale to massively multimodal domains is not investigated.

In the context of multimodal function optimization the concept of scalability relates to the capability of an algorithm to locate all or most of the optima when the size of the search space and the number of optima that can be found in such a space, increase. Modality can be increased by changing the boundaries of the search space, keeping the dimensionality fixed, or by increasing the dimensionality, or both. Previous experiments where swarm sizes were estimated, suggest that an increase in the number of optima requires a bigger swarm. However, success rates of algorithms locating multiple optima also depend on function landscapes. If search spaces are increased in such a way that the new landscape contains niches with shapes and sizes that differ considerably from that of the original landscape, results will not be meaningful.

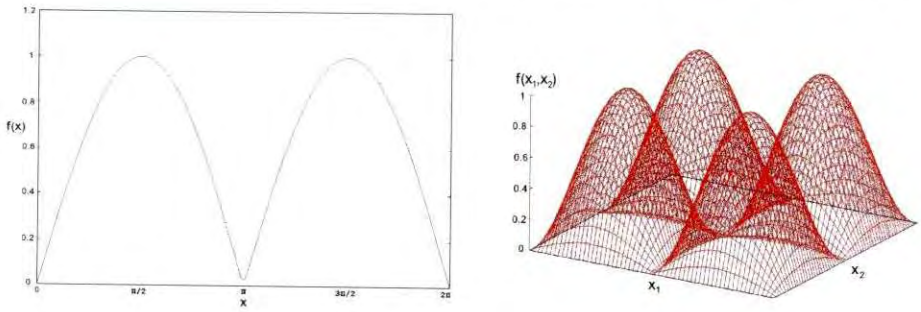
In this study, scalability is investigated by relating the required swarm size to the number of optima in a specified search space. Of the functions tested thus far, the Griewank and Rastrigin functions can be singled out as suitable for testing scalability. Descriptions of the Griewank and Rastrigin functions were presented in section 6.4. For each dimension the positions of the optima are relatively symmetrical. Search spaces can be manipulated to contain progressively higher numbers of optima. In addition, the absolute Sine function was used. The function exhibits a landscape containing regularly spaced optima with similar shapes and fitness over the entire search space. All three these benchmark functions can be described as massively multimodal, as optima are repeated indefinitely. Therefore, it is easy to visualize the function in various search spaces. The absolute Sine function is defined as:

$$f(\mathbf{x}) = \prod_{i=1}^{n} |\sin(x_i)| \tag{6.20}$$

Scalability of the VBPSO was only investigated for the three multimodal functions described above. Many multimodal functions have irregular function landscapes making it difficult to assess the positions of the optima in different search spaces. However, results obtained for these three functions do give an indication of the ability of the VBPSO to scale to higher multimodality. For this study search spaces were increased for each function to contain no more than 100 optima. Beyond that number, executing the VBPSO becomes increasingly computationally expensive. Also, it is believed that practical applications do not necessarily require niching strategies that are capable of locating such large numbers of optima.

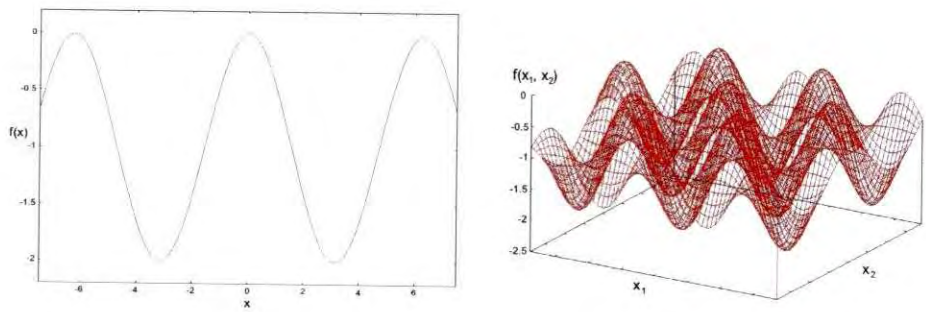
The relationship between the initial optimal swarm size, $|S|$, and the number of solutions was investigated by testing the performance of the VBPSO on a range of swarm sizes for each function in each specified search space. Thus, an optimal swarm size could be deduced for each function in each search space. For this study, search spaces were chosen which allows the optima to be easily determined through a manual process. These search spaces contain numbers of optima ranging from 2 to 96.

Figures 6.23 to 6.25 illustrate the function landscapes in some of these ranges in one and two dimensions. For clarity, the figures depict the optima as maxima, not minima.



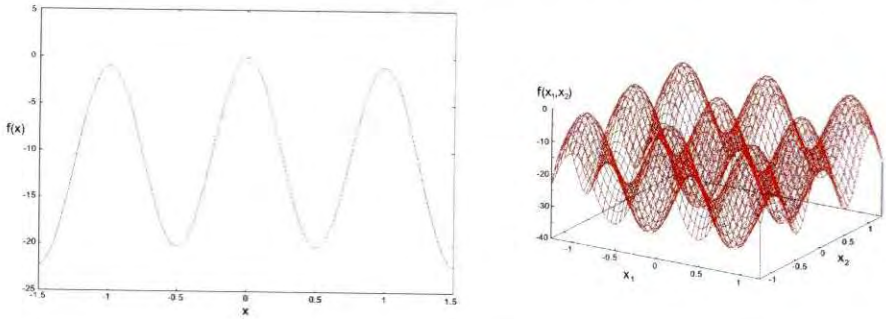
(a) One dimension (b) Two dimensions

Figure 6.23: The absolute Sine function in one and two dimensions



(a) One dimension (b) Two dimensions

Figure 6.24: The Griewank function in one and two dimensions



(a) One dimension (b) Two dimensions

Figure 6.25: The Rastrigin function in one and two dimensions

Positions of optima

Knowing the exact number of optima of a function in a designated search space is critical to support the use of a benchmark function as a test function. Similar distances between optima in the direction of the axes, as in the case of the absolute Sine and Rastrigin functions, enables easier assessment of the number of optima in a demarcated search space. However, in the case of the Griewank function, optima are not arranged symmetrically around the global optimum at $[0, 0]$, and the number and positions of the optima were not obvious. Carefully scrutinizing the search space and testing candidate positions were required to assess exact numbers of optima.

Experimental setup and results

Results of the performance of the vector-based PSO on the absolute Sine, Griewank and Rastrigin functions for a range of search spaces are presented in Tables 6.55 to 6.63 showing the average number of optima for each swarm size, as well as the success rate. Swarm sizes were increased from a small size until the algorithm produced a success rate of at least 98%. The same trend has been observed for all swarm sizes. Figures 6.26, 6.27 and 6.28 show a random selection of graphs where the average number of optima is plotted against the swarm size for the absolute Sine, Griewank and Rastrigin functions respectively.

Table 6.55: Average number of solutions versus swarm sizes for the absolute Sine function - part 1

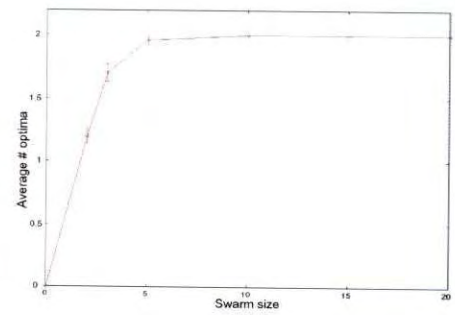
2 optima $[0, 2\pi]$			4 optima $[0, 2\pi]^2$			8 optima $[0, 2\pi]^3$			12 optima $[0, 2\pi]^2 * [0, 3\pi]$		
Swarm size	Average ♯ optima	Success rate	Swarm size	Average ♯ optima	Success rate	Swarm size	Average ♯ optima	Success rate	Swarm size	Average ♯ optima	Success rate
2	1.2 ± 0.06	60%	5	2.44 ± 0.1	61%	10	3.78 ± 0.16	47.25%	50	11.34 ± 0.1	94.5%
3	1.7 ± 0.07	85%	10	3.48 ± 0.08	87%	20	6.64 ± 0.15	83%	60	11.42 ± 0.1	95.17%
5	1.96 ± 0.03	98%	15	3.9 ± 0.04	97.5%	30	7.38 ± 0.09	92.25%	70	11.70 ± 0.07	97.5%
10	2 ± 0	100%	18	3.94 ± 0.03	98.5%	40	7.82 ± 0.06	97.75%	80	11.82 ± 0.05	98.5%
15	2 ± 0	100%	20	3.94 ± 0.03	98.5%	50	7.9 ± p.05	98.75%			
20	2 ± 0	100%	25	4 ± 0	100%	60	8 ± 0	100%			
			30	4 ± 0	100%						

Table 6.56: Average number of solutions versus swarm sizes for the absolute Sine function - part 2

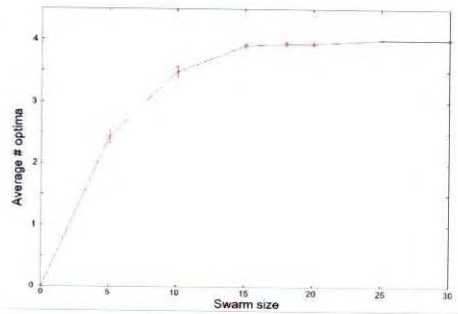
16 optima $[0, 2\pi]^4$			24 optima $[0, 2\pi]^3 * [0, 3\pi]$			32 optima $[0, 2\pi]^3 * [0, 4\pi]$		
Swarm size	Average ♯ optima	Success rate	Swarm size	Average ♯ optima	Success rate	Swarm size	Average ♯ optima	Success rate
20	7.14 ± 0.16	44.63%	20	7.74 ± 0.24	32.25%	20	7.72 ± 0.25	24.13%
40	11.92 ± 0.26	74.5%	40	14.56 ± 0.27	60.67%	40	14.76 ± 0.31	46.13%
60	14.34 ± 0.17	89.63%	60	18.36 ± 0.32	76.5%	60	19.36 ± 0.38	60.5%
80	15.24 ± 0.12	95.25%	80	21 ± 0.19	87.5%	80	23.8 ± 0.37	74.38%
100	15.56 ± 0.07	92.25%	100	22.08 ± 0.19	92%	100	26.08 ± 0.26	81.5%
120	15.84 ± 0.05	99%	120	23 ± 0.13	95.83%	120	28.36 ± 0.19	88.63%
140	15.94 ± 0.03	99.63%	140	23.28 ± 0.09	97%	140	29.4 ± 0.23	91.88%
160	16 ± 0	100%	150	23.4 ± 0.11	97.5%	160	30.48 ± 0.16	95.25%
			160	23.48 ± 0.10	97.5%	180	31.12 ± 0.10	97.25%
			170	23.64 ± 0.09	98.5%	200	31.32 ± 0.11	98.13%

Table 6.57: Average number of solutions versus swarm sizes for the absolute Sine function - part 3

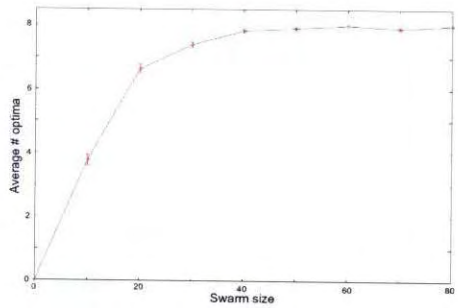
48 optima $[0, 2\pi]^2 * [0, 4\pi]^2$			64 optima $[0, 2\pi]^2 * [0, 4\pi]^2$			96 optima $[0, 2\pi] * [0, 3\pi] * [0, 4\pi]^2$		
Swarm size	Average ♯ optima	Success rate	Swarm size	Average ♯ optima	Success rate	Swarm size	Average ♯ optima	Success rate
50	±	%	100	35.16 ± 0.51	54.94%	100	39.93 ± 0.57	26.88%
100	34.08 ± 0.26	74.5%	200	53.04 ± 0.39	82.88%	200	66.6 ± 0.60	39.75%
150	±	%	240	57.72 ± 0.31	90.19%	300	81.46 ± 0.40	59.34%
200	44.12 ± 0.16	91.92%	280	60.04 ± 0.20	93.81%	400	90.56 ± 0.31	94.33%
220	44.76 ± 0.31	93.25%	300	61.36 ± 0.23	95.88%	450	93 ± 0.25	96.88%
240	46 ± 0.15	95.83%	320	61.6 ± 0.17	96.25%	500	92.84 ± 0.25	96.71%
260	46.36 ± 0.17	96.58%	340	61.76 ± 0.22	96.5%	550	93.88 ± 0.20	97.79%
280	47.04 ± 0.14	98%	360	62.48 ± 0.16	97.63%	600	94.08 ± 0.23	98%
			380	62.64 ± 0.15	97.88%			
			400	63.04 ± 0.14	98.5%			



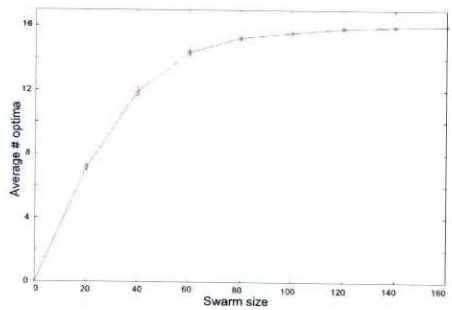
(a) 2 optima



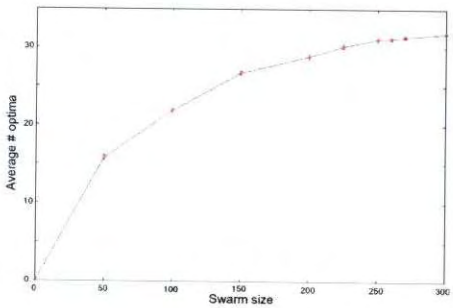
(b) 4 optima



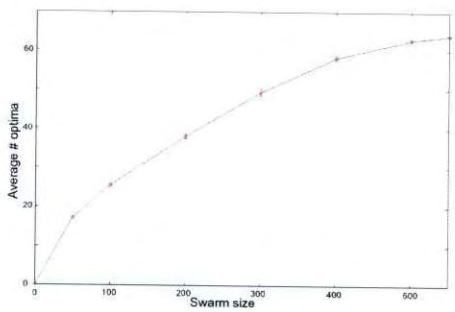
(c) 8 optima



(d) 16 optima



(e) 32 optima



(f) 64 optima

Figure 6.26: % Optima found versus swarm sizes for the absolute Sine function

Table 6.58: Average number of solutions versus swarm sizes for the Griewank function - part 1

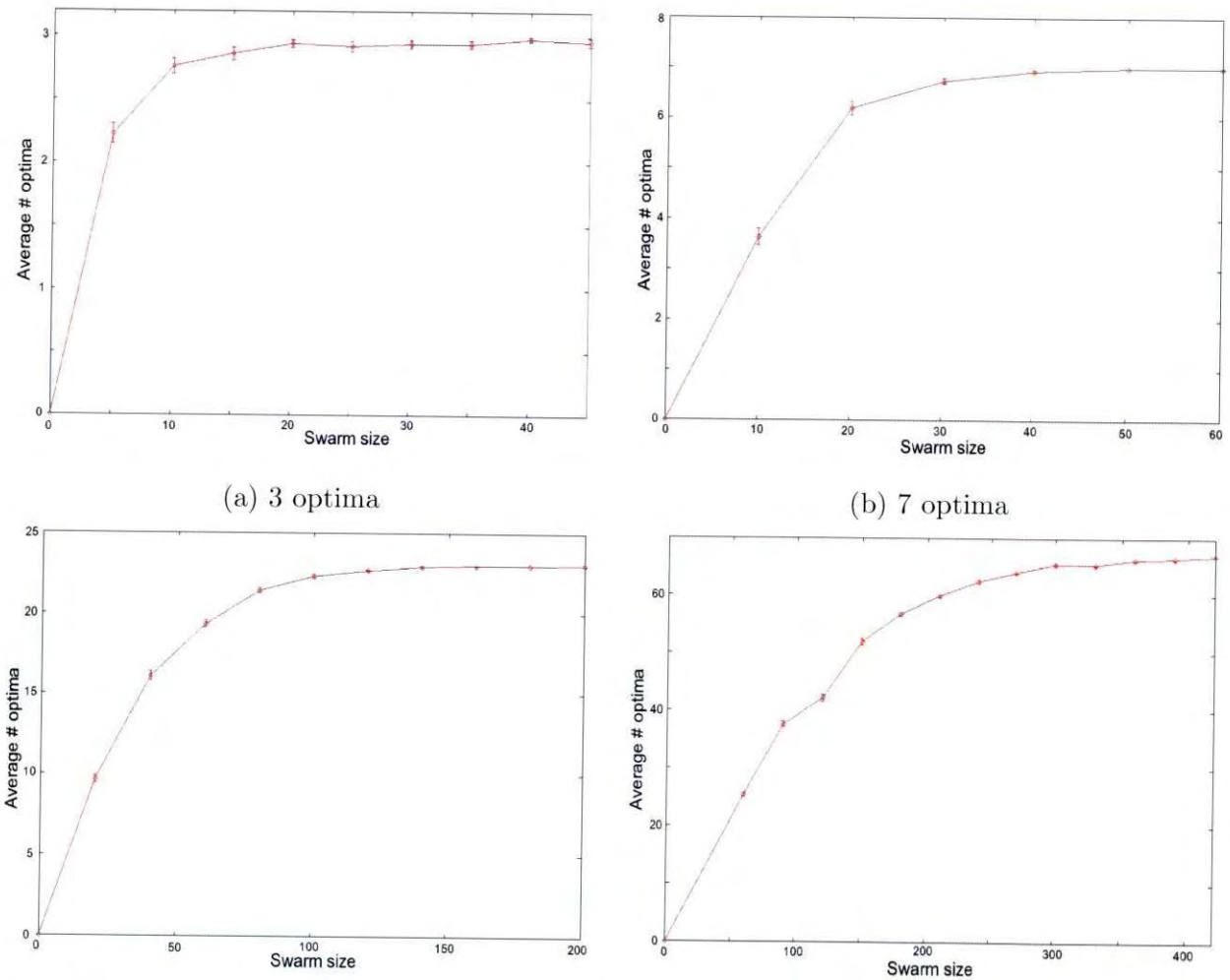
3 optima $[-7.5, 7.5]$			7 optima $[-7.5, 7.5]^2$			11 optima $[-10.5, 10.5] * [-7.5, 7.5]$			17 optima $[-10.5, 10.5]^2$		
Swarm size	Average ♯ optima	Success rate	Swarm size	Average ♯ optima	Success rate	Swarm size	Average ♯ optima	Success rate	Swarm size	Average ♯ optima	Success rate
5	2.22 ± 0.08	74%	10	3.64 ± 0.16	52%	20	7.68 ± 0.20	69.82%	20	9 ± 0.31	52.94%
10	2.76 ± 0.06	92%	20	6.2 ± 0.14	88.57%	30	9.48 ± 0.16	86.18%	30	12.88 ± 0.17	75.76%
15	2.86 ± 0.05	95.33%	30	6.74 ± 0.06	96.29%	40	10.4 ± 0.10	94.55%	40	15 ± 0.19	88.24%
20	2.94 ± 0.03	98%	40	6.94 ± 0.03	99.14%	45	10.52 ± 0.1	95.64%	20	15.88 ± 0.17	93.41%
25	2.92 ± 0.04	97.33%	50	7 ± 0	100%	50	10.96 ± 0.03	99.64%	180	16.48 ± 0.10	96.94%
30	2.94 ± 0.03	98%	60	7 ± 0	100%				210	16.76 ± 0.07	98.59%

Table 6.59: Average number of solutions versus swarm sizes for the Griewank function - part 2

23 optima $[-7.5, 7.5]^3$			31 optima $[-14.5, 14.5]^2$			39 optima $[-17.5, 17.5]^2$		
Swarm size	Average ♯ optima	Success rate	Swarm size	Average ♯ optima	Success rate	Swarm size	Average ♯ optima	Success rate
20	9.66 ± 0.23	42%	40	19.68 ± 0.24	63.48%	50	21.21 ± 0.56	54.38%
40	16.1 ± 0.29	70%	60	25.2 ± 0.31	81.29%	100	33.29 ± 0.34	85.33%
60	19.34 ± 0.22	84.09%	80	27.92 ± 0.19	90.06%	120	34.94 ± 0.29	89.64%
80	21.42 ± 0.19	93.13%	100	29.08 ± 0.15	93.81%	140	36.77 ± 0.18	94.36%
100	22.3 ± 0.21	96.96%	120	30.24 ± 0.13	97.55%	160	37.56 ± 0.16	96.31%
110	22.68 ± 0.07	98.61%	130	30.36 ± 0.09	97.94%	180	38.31 ± 0.13	98.26%
			140	30.44 ± 0.10	98.19%			

Table 6.60: Average number of solutions versus swarm sizes for the Griewank function - part 3

53 optima $[-10.5, 10.5] * [-7.5, 7.5]$			67 optima $[-7.5, 7.5]^4$			83 optima $[-17.5, 17.5] * [-10.5, 10.5] * [-7.5, 7.5]$		
Swarm size	Average ♯ optima	Success rate	Swarm size	Average ♯ optima	Success rate	Swarm size	Average ♯ optima	Success rate
100	35.54 ± 0.33	67.09%	60	25.4 ± 0.34	37.91%	100	48.46 ± 0.37	58.51%
150	44.44 ± 0.24	83.7%	90	37.64 ± 0.51	56.18%	200	72.10 ± 0.40	86.84%
200	49.42 ± 0.23	93.28%	120	42.18 ± 0.59	62.96%	250	76.82 ± 0.28	92.63%
220	51.13 ± 0.15	96.53%	150	52 ± 0.52	77.61%	300	80.54 ± 0.22	96.96%
240	50.83 ± 0.17	96%	180	56.8 ± 0.39	84.78%	320	80.98 ± 0.19	97.59%
260	51.82 ± 0.14	97.74%	210	60 ± 0.28	89.55%	340	81.08 ± 0.17	97.69%
270	52.21 ± 0.12	98.49%	240	62.8 ± 0.33	93.43%	350	82.13 ± 0.18	98.94%
			270	64.1 ± 0.12	95.67%			
			300	65.5 ± 0.13	97.76%			
			310	65.96 ± 0.10	98.44%			



(a) 3 optima

(b) 7 optima

(c) 23 optima

(d) 67 optima

Figure 6.27: % Optima versus swarm sizes for the Griewank function

Table 6.61: Average number of solutions versus swarm sizes for the Rastrigin function

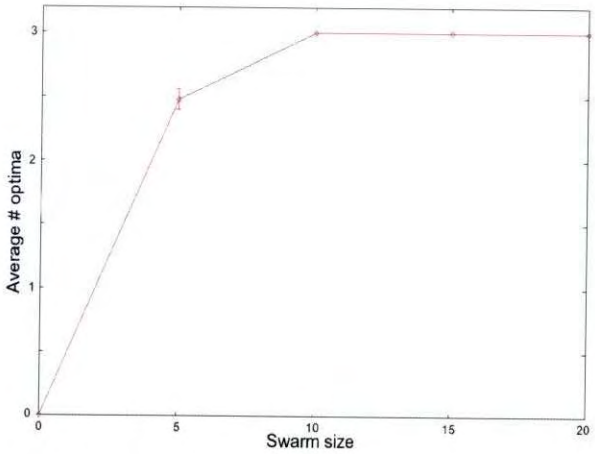
3 optima $[-1.5, 1.5]$			9 optima $[-1.5, 1.5]^2$			15 optima $[-1.5, 1.5] * [-2.5, 2.5]$			21 optima $[-1.5, 1.5] * [-3.5, 3.5]$		
Swarm size	Average ♯ optima	Success rate	Swarm size	Average ♯ optima	Success rate	Swarm size	Average ♯ optima	Success rate	Swarm size	Average ♯ optima	Success rate
5	2.48 ± 0.08	82.67%	10	5.52 ± 0.22	61.33%	20	10.12 ± 0.27	67.47%	30	14.28 ± 1.25	68%
10	3 ± 0	100%	20	8 ± 0.15	88.89%	30	12.64 ± 0.18	84.27%	40	17.92 ± 0.19	85.33%
15	3 ± 0	100%	30	8.72 ± 0.06	96.89%	40	13.96 ± 0.14	96%	60	20 ± 0.14	95.24%
20	3 ± 0	100%	40	9 ± 0	100%	50	14.56 ± 0.09	97.07%	70	20.2 ± 0.11	96.19%
			50	8.96 ± 0.03	99.56%	60	14.64 ± 0.09	97.6%	80	20.68 ± 0.08	98.48%
			60	8.98 ± 0.02	99.78%	70	14.84 ± 0.06	98.93%			
			70	9 ± 0	100%						

Table 6.62: Average number of solutions versus swarm sizes for the Rastrigin function - part 2

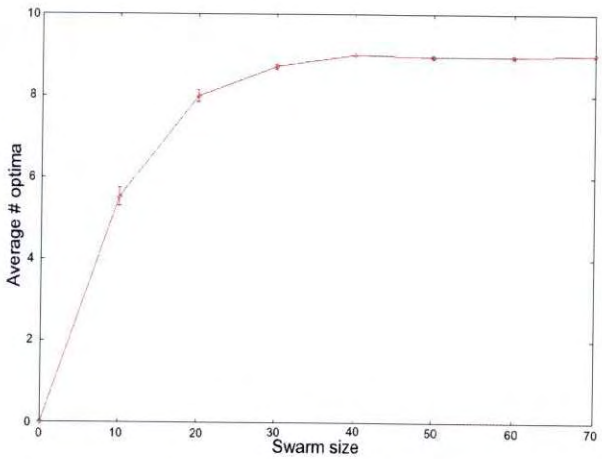
27 optima $[-1.5, 1.5]$			35 optima $[-2.5, 2.5] * [-3.5, 3.5]$			45 optima $[-1.5, 1.5]^2 * [-2.5, 2.5]$		
Swarm size	Average ♯ optima	Success rate	Swarm size	Average ♯ optima	Success rate	Swarm size	Average ♯ optima	Success rate
30	15.72 ± 0.26	58.22%	50	24.2 ± 0.36	69.14%	50	24.8 ± 0.41	55.11%
60	23.14 ± 0.21	85.7%	100	33.16 ± 0.17	94.74%	100	37.8 ± 0.27	84%
90	25.92 ± 0.15	96%	120	34.12 ± 0.10	97.49%	120	39.56 ± 0.29	87.91%
110	25.96 ± 0.16	96.15%	130	34.1 ± 0.13	97.6%	140	41.4 ± 0.20	92%
120	26.68 ± 0.08	98.81%	140	34.72 ± 0.06	99.2%	160	42.68 ± 0.15	94.84%
						180	43.44 ± 0.16	96.53%
						200	44.16 ± 0.14	98.13%

Table 6.63: Average number of solutions versus swarm sizes for the Rastrigin function - part 3

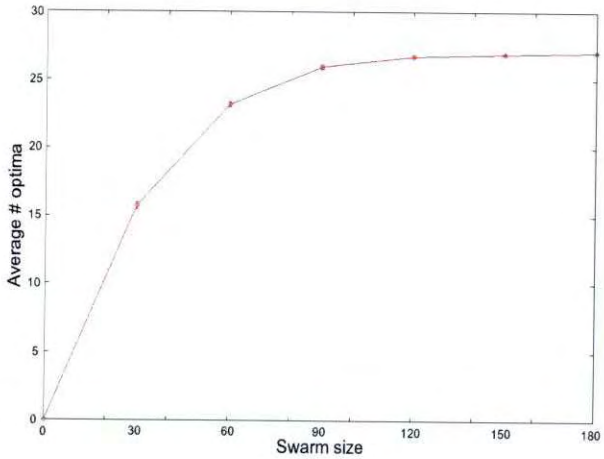
63 optima $[-1.5, 15]^2 * [-3.5, 3.5]$			75 optima $[-1.5, 1.5] * [-2.5, 2.5]^2$			81 optima $[-1.5, 1.5]^4$		
Swarm size	Average ♯ optima	Success rate	Swarm size	Average ♯ optima	Success rate	Swarm size	Average ♯ optima	Success rate
50	25.88 ± 0.36	41.08%	100	47.08 ± 0.50	62.77%	90	45.7 ± 0.60	56.42%
100	45.08 ± 0.32	71.56%	150	57.72 ± 0.45	76.96%	120	53.2 ± 0.47	65.68%
150	54.16 ± 0.32	85.97%	200	65.56 ± 0.38	87.41%	150	57.5 ± 0.44	70.99%
200	59.44 ± 0.23	94.34%	250	69.2 ± 0.30	92.27%	180	63.3 ± 0.47	78.15%
220	59.7 ± 0.22	94.7%	275	70.92 ± 0.25	94.56%	210	68.4 ± 0.40	84.44%
240	60.6 ± 0.23	96.19%	300	72.28 ± 0.22	96.37%	240	71.3 ± 0.33	88.02%
260	61.8 ± 0.16	98.10%	350	73.5 ± 0.17	98%	270	74.3 ± 0.29	91.73%
			360	73.64 ± 0.20	98.19%	300	75.8 ± 0.31	93.58%
						330	76.4 ± 0.25	94.32%
						360	78.4 ± 0.19	96.79%
						390	79 ± 0.16	97.53%
						400	79.48 ± 0.18	98.12%



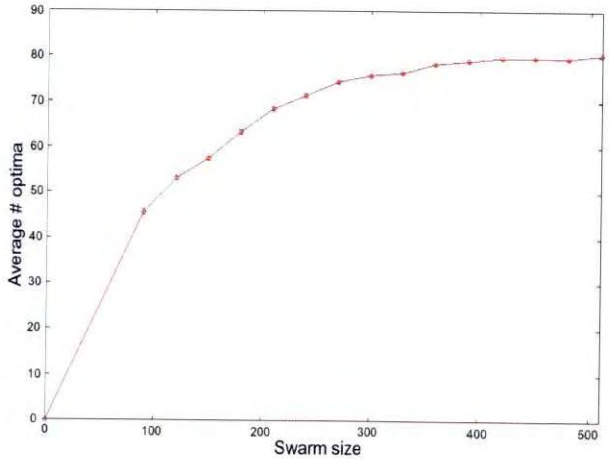
(a) 3 optima



(b) 9 optima



(c) 27 optima



(d) 81 optima

Figure 6.28: % Optima versus swarm sizes for the Rastrigin function

Table 6.64: Minimum swarm sizes required to locate 98% of possible optima for the absolute Sine function

Search space	Optima	Swarm size	R^2
$[0, 2\pi]$	2	5	
$[0, 2\pi]^2$	4	18	
$[0, 2\pi]^3$	8	50	
$[0, 2\pi]^2 * [0, 3\pi]$	12	80	
$[0, 2\pi]^4$	16	120	0.9980
$[0, 2\pi]^3 * [0, 3\pi]$	24	170	
$[0, 4\pi] * [0, 2\pi]^3$	32	210	
$[0, 4\pi] * [0, 3\pi] * [0, 2\pi]^2$	48	280	
$[0, 4\pi]^2 * [0, 2\pi]^2$	64	400	
$[0, 4\pi]^2 * [0, 3\pi] * [0, 2\pi]$	96	600	

From the results presented in this section, minimum swarm sizes required to locate at least 98% of the optima in predefined search spaces were determined for all three functions. Tables 6.64, 6.65 and 6.66 summarize the minimum swarm sizes for which an average success rate of above 98% have been reported.

For each of the functions, Figure 6.29 illustrates the optimal swarm sizes, $|S|^*$, plotted against the actual number of optima, x, that can be located in the search space. For each of the functions a linear regression was done to describe the dependence of optimal swarm size on the number of possible optima, given that $x \geq 0$. For the absolute Sine function the linear equation is

$$|S|^* = 6.26x$$

For the Griewank function the equation is

$$|S|^* = 4.44x$$

For the Rastrigin function the equation is

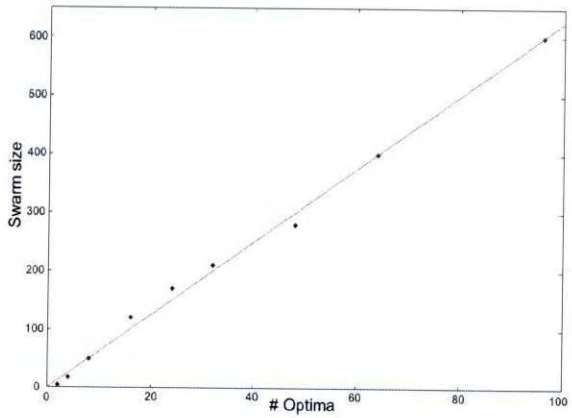
$$|S|^* = 4.56x$$

Table 6.65: Minimum swarm sizes required to locate 98% of possible optima for the Griewank function

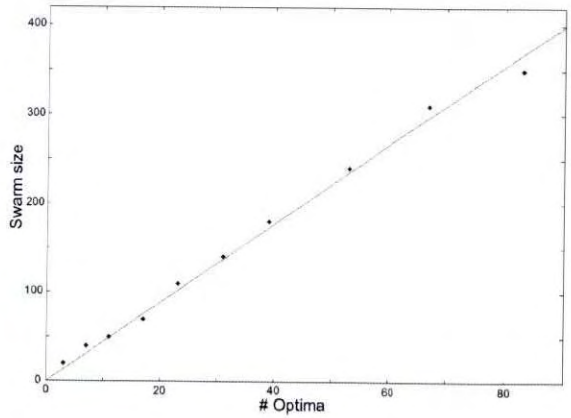
Search space	Optima	Swarm size	R^2
$[-7.5, 7.5]$	3	20	
$[-7.5, 7.5]^2$	7	40	
$[-10.5, 10.5] * [-7.5, 7.5]$	11	50	
$[-10.5, 10.5]^2$	17	70	
$[-7.5, 7.5]^3$	23	110	0.9977
$[-14.5, 14.5]^2$	31	140	
$[-17.5, 17.5]^2$	39	180	
$[-10.5, 10.5] * [-7.5, 7.5]$	53	240	
$[-7.5, 7.5]^4$	67	310	
$[-17.5, 17.5] * [-10.5, 10.5] * [-7.5, 7.5]$	83	350	

Table 6.66: Minimum swarm sizes required to locate 98% of possible optima for the Rastrigin function

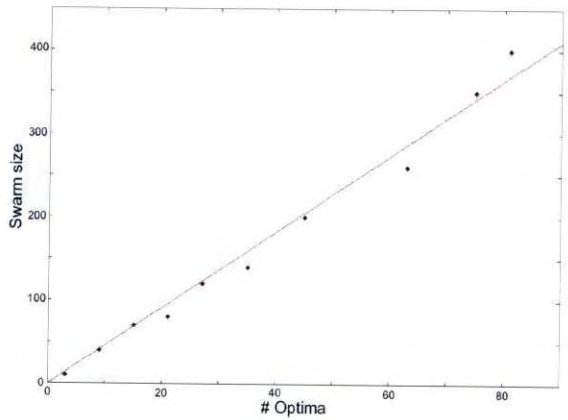
Search space	Optima	Swarm size	R^2
$[-1.5, 1.5]$	3	10	
$[-1.5, 1.5]^2$	9	40	
$[-1.5, 1.5] * [-2.5, 2.5]$	15	70	
$[-1.5, 1.5] * [-3.5, 3.5]$	21	80	
$[-1.5, 1.5]^3$	27	120	0.9985
$[-2.5, 2.5] * [-3.5, 3.5]$	35	140	
$[-1.5, 1.5]^2 * [-2.5, 2.5]$	45	200	
$[-1.5, 1.5]^2 * [-3.5, 3.5]$	63	260	
$[-1.5, 1.5] * [-2.5, 2.5]^2$	75	360	
$[-1.5, 1.5]^4$	81	400	



(a) The absolute Sine function



(b) The Griewank function



(c) The Rastrigin function

Figure 6.29: Optimal swarm sizes versus actual number of optima for three benchmark functions

Tables 6.64 to 6.66 also lists the coefficient of determination, R^2, for each regression. In all three cases $R^2 > 0.99$, indicating that the above linear functions provide a very good fit. Therefore, for all the functions considered, a linear relationship

$$|S|^* = ax + c$$

exists between the optimal swarm size and the actual number of optima where a and c are constants depending on the objective function and the search space where it is investigated. Please note that the value of c was found to be 0 for the functions considered in this study.

6.6 A comparative study of three PSO niching approaches

A number of niching methods for particle swarm optimization have been described in chapter 4. This section compares the performance of two of these methods, the species-based PSO and NichePSO, to that of the vector-based PSO. In this study the enhanced parallel vector-based PSO is used and is referred to as the vector-based PSO (VBPSO). All three niching methods optimizes subswarms in parallel, but the manner in which niches form, differs.

6.6.1 Experimental procedure

Comparison of different algorithms requires similar experimental setups. The species-based PSO and NichePSO algorithms are tested on the same seven two-dimensional functions as the vector-based PSO, using the same number of particles for each function. Given that the initial distribution of particles throughout the problem space can influence the performance of an algorithm considerably, both the species-based PSO and the vector-based PSO were implemented using Sobol sequences as a random number generator. NichePSO uses *Fauré* sequences, also yielding even distributions of initial particle positions over the search space. The number of iterations was also set to 500 for each algorithm. Average results were calculated for each algorithm.

The species-based PSO requires a niche radius to be set in advance. Each function was tested with a small range of niche radii. Results using the radius yielding the best outcomes are reported. NichePSO was tested using the CILib framework developed by the CIRG research group at the department of Computer Science, University of Pretoria (http://cilib.sourceforge.net). NichePSO requires a merging parameter, μ. In order to compare the algorithms, similar set-

tings were used as far as possible. Therefore, for each function μ was set to the granularity used in the vector-based algorithm, but it was normalized as required by NichePSO.

6.6.2 Results

Results are reported in Table 6.67. The success rate of each algorithm for each function is reported. In order to compare the algorithms, results of the vector-based PSO are repeated in the table. The granularity, niche radius or merging parameter is listed for every function, as well as the number of particles.

6.6.3 Discussion

This section discusses the behaviour of each of the algorithms when tested on functions with differing characteristics.

The species-based PSO performed well on the Himmelblau, Griewank and Rastrigin functions with a success rate of more than 90%. Each of these functions has a number of well-defined optima where the heights and interniche distances differ very little or not at all. Given the right choice of a niche radius, a good performance of such a simple, elegant and effective algorithm can be expected. However, when the search space is more convoluted and the shapes, sizes and placing of optima in the search space are less symmetrical, the niche radii differ from niche to niche and the performance degrades. These expectations are confirmed by the results where the success rate of the Ursem F1 function is 86.67%, the Ursem F3 function 73.33% and the six hump camel function 63.89%.

NichePSO performed well on a number of the functions: the Himmelblau, Griewank, Ackley and Ursem F1 functions all have success rates of more than 90%. However, the Rastrigin function has a 80% success rate in this implementation; an unexpected result given the results obtained by the original implementation where the success rate was 100% for a region where 9 optima occurred [13] [14]. The indication is that too many niches merge and that the algorithm needs some fine tuning of parameters to prevent these occurrences. Such fine tuning, which has to be adapted for each function, has not yet been incorporated into the NichePSO implementation in the CILib framework. For the Ursem F3 and six hump camel functions where the niche sizes differ considerably, the performance degrades to success rates of 35.83% and 33.33%. For the six hump camel function only the two large niches were located in all cases. Therefore it can be concluded that too many niches merge if niche sizes differ considerably.

While NichePSO manipulates subswarms in ingenious ways, the merging process is not robust enough if the objective functions become more convoluted.

For the small subset of functions that has been tested in this study, all three algorithms performed equally well on functions where the landscapes are relatively symmetrical. However, the vector-based PSO outperformed the other two algorithms on functions where the differences between the interniche distances, the fitnesses, and niche radii of the various optima become larger, for example, the Ursem F3 and six hump camel functions. Table 6.67 show larger differences in performance for these functions. These differences can be ascribed to the strategy used by VBPSO to calculate a separate niche radius for each niche, as well as the forming of false niches. The latter occurs more often when niches are not symmetrical. Different niche radii ensure that subswarms in smaller niches are not absorbed by those in larger niches, resulting in a smaller success rate. The concept of false niches address the deficiency of niche radii in general, which assumes that a niche occupies a spherical region. False niches accommodate particles inside a niche but falling outside regions demarcated by niche radii. Thus, niches of all shapes and sizes are accommodated by VBPSO, constituting a robust algorithm that performs well for convoluted and asymmetrical function landscapes.

Figure 6.30 shows a bar graph where the performance of the three niching strategies is plotted for all seven functions. The number of optima found during 50 runs is represented as a percentage of the total number of optima.

6.7 Conclusion

This chapter presented results of the vector-based PSO applied to static environments. Three versions of the algorithm, namely the sequential, parallel and enhanced parallel vector-based PSO were tested extensively on a number of one- and two-dimensional objective functions. Each implementation represents an improvement on the previous version. An analysis of the algorithm was conducted where the influence of the granularity on niching ability as well as the scalability were tested. Optimal swarm sizes were deduced for three multimodal functions in a number of dimensions, and a relationship was derived between optimal swarm sizes and the number of optima. The chapter was concluded by reporting on a comparative study of three niching algorithms that emphasized the strengths, effectiveness and robustness of the vector-base PSO.

Table 6.67: Comparing three niching algorithms

Functions	♯ Particles	Vector-based PSO		Species-based PSO		NichePSO	
		Granularity	Success rate	Niche radius	Success rate	Merging parameter μ	Success rate
Himmelblau	30	0.5	100%	3.75	99.17%	0.5	100%
Griewank	40	0.5	100%	3	100%	0.5	97.33%
Rastrigin	60	0.1	99.63%	0.6	91.11%	0.1	80%
Ackley	60	0.3	99.63%	0.6	78.51%	0.3	91.11%
Ursem F1	30	0.5	100%	1.8	86.67%	0.5	95%
Ursem F3	40	0.3	100%	0.7	73.33%	0.3	35.83%
Six hump camel	50	0.3	99.44%	0.6	63.89%	0.3	33.33%

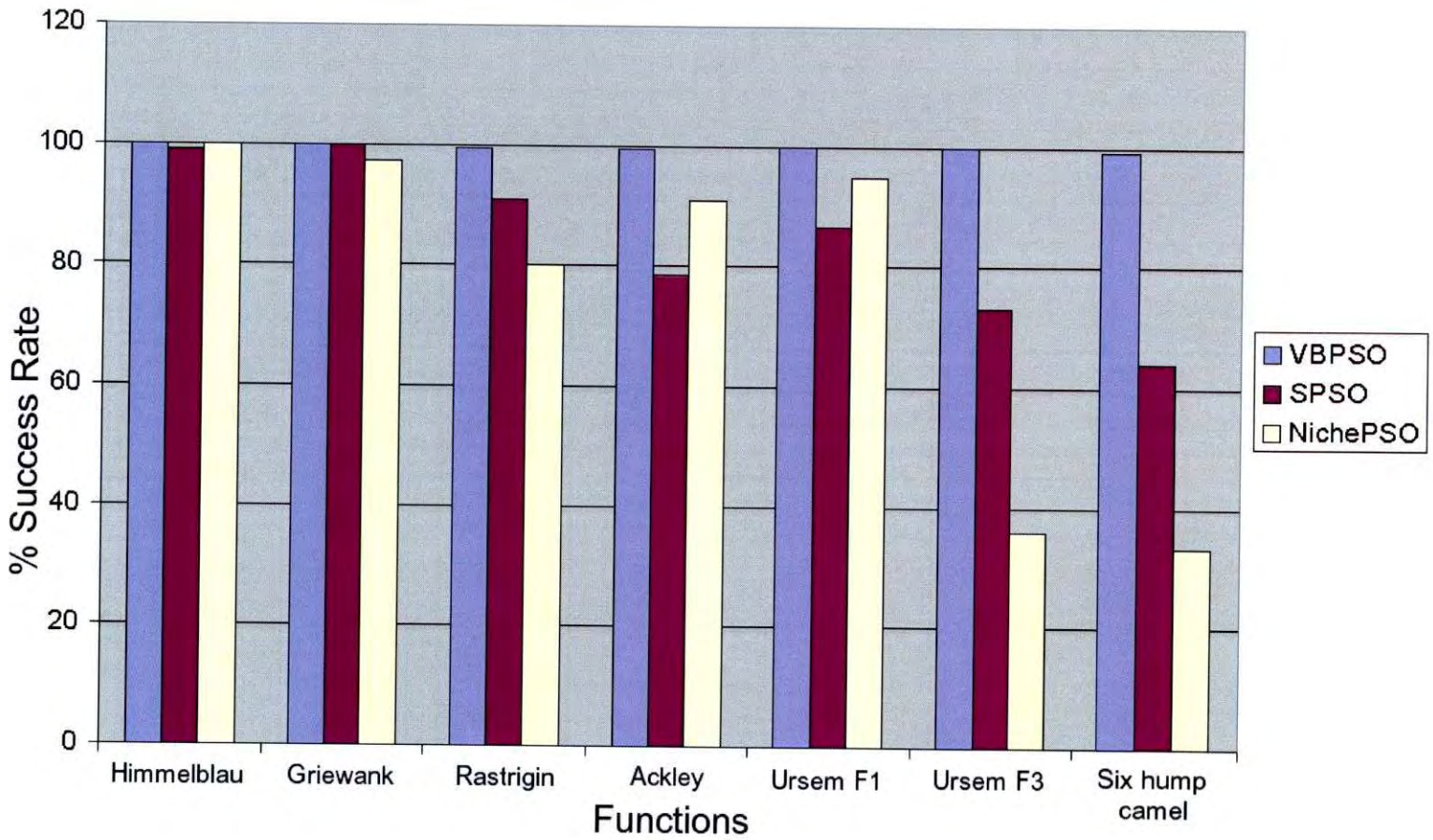


Figure 6.30: Comparing performance of three niching strategies for seven functions.

Chapter 7

Dynamic Vector-Based PSO

This chapter investigates the behaviour of the vector-based particle swarm optimization paradigm in changing environments. Multiple optima can change dynamically over time, both temporally and spatially, requiring an algorithm to track these multiple, changing optima. The vector-based PSO, originally developed to locate multiple optima in static environments, is adapted and extended to track multiple optima over time.

7.1 Introduction

PSO can be considered to be a well-established population-based optimization approach. Although PSO has been successfully applied to static problems, real-world optimization often has to be carried out in a dynamic environment; that is, the objective function changes over time. Examples of such problems include scheduling problems such as air traffic control and routing in telecommunication networks. Controlling petrochemical processes also requires frequent re-optimization to balance input and output parameters. Locations of optima and the fitness of the objective function at these positions may change, while some optimal solutions disappear altogether and others appear in new positions. To locate an optimum using particle swarm optimization, particles have to converge on a single point. However, to keep track of dynamically changing optima, some form of redistribution of particles is required in order to increase swarm diversity. This is necessary for the algorithm to continually explore the search space in order to track an optimum of which the position or fitness have changed, or to detect

the disappearance of an optimum or the appearance of a new optimum. Some algorithms such as the charged PSO [8] [9], automatically track optima in a dynamically changing environment, while others require modification of the algorithm.

Depending on the severity of the move, PSO in dynamic environments proved to be effective when locating and tracking a single optimal value [6] [7] [8] [17]. However, in a highly multi-modal environment changes in the fitness of the optima may result in a previous suboptimal solution taking on the best value. Such a situation would be easy to handle if an algorithm could locate and keep track of multiple optima.

This chapter presents an initial and explorative study of the ability of a niching PSO method to track multiple dynamically changing optima. The vector-based PSO [82], [83], [84] is adapted to locate and track optima in a changing environment. A number of scenarios is set up and used to test the performance of the dynamic vector-based PSO in dynamic environments [85].

The chapter is organized as follows: Dynamic optimization problems are defined in section 2, while changes in the environment are discussed in section 3. Research on PSO models to locate a single optimum in a dynamic environment is summarized in section 4. Section 5 presents a discussion of existing algorithms to locate and track multiple dynamic optima. Section 6 describes how the vector-based PSO is extended to locate and track optima in a dynamic environment in parallel, while section 7 concludes the chapter.

7.2 Dynamic optimization problems

A formal definition of a dynamic optimization problem is given as

$$\begin{aligned} &\textit{minimize} \quad f(\mathbf{x}, \varpi(t)), \quad \mathbf{x} = (x_1, \ldots, x_{n_x}), \qquad \varpi(t) = (\varpi_1(t), \ldots, \varpi_{n_w}) \\ &\textit{subject to} \qquad g_m(\mathbf{x}) \leq 0, m = 1, \ldots, n_g \\ &\qquad\qquad\qquad h_m(\mathbf{x}) = 0, m = n_g + 1, \ldots, n_g + n_h \\ &\qquad\qquad\qquad x_j \in dom(x_j) \end{aligned} \tag{7.1}$$

where $\varpi(t)$ is a vector of time-dependent objective function control parameters. The objective is to find

$$\mathbf{x}^*(t) = \min_x f(\mathbf{x}, \varpi(t)) \tag{7.2}$$

where $\mathbf{x}^*(t)$ is the optimum found at time step t.

Dynamic systems change over time in several ways. Changes can take place at regular or irregular time intervals, or continuously. The timescale involved is referred to as *temporal*

severity. Changes in optimal positions can also be quantified. The term *spatial severity* is used to indicate the amount of change in the position of an optimum at each step. The severity parameter, ζ, is used to indicate the change in location before the next optimization effort.

In addition to the type of dynamic environment, the way in which change occurs, can also vary. The next section presents a discussion of changes in the environment.

7.3 Changes in the environment

To a large extent the dynamics of an environment depends on the optimization problem. Optimization algorithms for dynamic environments are designed to react to changes in the environments. If changes occur at specific intervals, the mechanism the algorithm uses to improve exploration is invoked to coincide with those times. If changes do not occur at fixed intervals, those changes in the environment need to be detected automatically.

Carlisle and Dozier [17] [18] used a sentry particle to detect change in the environment. A new sentry particle is chosen at random each iteration. If the fitness of such a particle is different from that of the previous iteration, the environment has changed, although no change in the fitness of the sentry particle does not guarantee that the environment did not change. If the change is significant, a tracking strategy is triggered. An alternative method, proposed by Hu and Eberhart [45], monitors the global best position. The method was improved by monitoring the second-best position as well [46].

Response to changes in the environment can take many forms. If the environment changes while the swarm is still in the process of converging, and the spatial severity is not too large, the PSO automatically adapts and moves towards the new optimum [33]. If the swarm has converged to an equilibrium state, which is possible if the temporal severity is low, the swarm becomes stable and will not move to the new optimal position. In such cases some strategy has to be deployed to inject more diversity which will allow the swarm to move to the new optimal position. According to Blackwell [6], optimization with particle swarms has two major ingredients, the particle dynamics and the particle information network. A combination of these ingredients make PSO a robust and efficient optimizer of real-valued objective functions. However, when applying PSO to dynamic problems the algorithm must be modified for optimal results, as *diversity loss* is experienced due to convergence, while the problem of *outdated memory* occurs as a result of the environment dynamism. Diversity loss is the more serious problem, as the swarm has to re-diversify, locate the shifted optimum, and re-converge. Two strategies

can be utilized to address such a problem, namely the deployment of a re-diversification mechanism when the objective function changes, and/or some diversity measure that is maintained during the run.

The next section overviews a number of approaches to single-solution particle swarm optimization in dynamic environments.

7.4 Particle swarm models for tracking a single solution in a dynamic environment

Carlisle and Dozier [17] [18] conducted one of the first investigations into modification of the Particle Swarm Optimizer to locate changing extrema. Each particle's record of its personal best position is reset to the current particle position as the environment changes. Resetting only takes place if, after the environment change, the personal best position is worse than the current position. Thus direction and velocity decisions based on outdated information are avoided. This process is initiated by two methods: periodic resetting, based on the iteration count, and triggered resetting, based on the magnitude of the change in the environment. For the second method a sentry particle is selected at random in the search space. At each iteration the sentry particle's fitness is reevaluated; if the fitness changed by more than the *trigger value*, the personal best positions of all particles are reset.

It must be noted that such a strategy will only be effective when the swarm has not yet converged to a solution. The use of a sentry particle to detect change is also not reliable, as localized fluctuations may occur.

Eberhart and Shi experimented with tracking and optimizing a single optimum in a dynamic system [30]. Successful tracking of a 10-dimensional parabolic function is demonstrated. PSO was perceived to perform better than genetic algorithms on similar problems. However, dynamic environments can vary considerably, and many real-world systems change state frequently, thus requiring frequent re-optimization. Eberhart and Shi reason that searching from the current position works well for small environmental changes, or when the swarm has not yet reached an equilibrium state. For severe environmental changes, re-initialization of the entire swarm can be more effective as previous optimal positions will contribute very little or nothing at all to the effort. A combination of these two approaches, namely retaining the global best position and re-initializing a percentage of the particle positions, is suggested in order to retain

potentially good positions as well as increase diversity.

Blackwell and Branke identify the following problems which have to be considered when applying PSO to dynamic environments [9]:

Outdated memory: When the objective function changes, an individual particle's personal best solution (called the *particle attractor* by Blackwell and Branke) may no longer apply, and may even guide the search in the wrong direction. This problem is usually solved by resetting the position of the attractor to the current particle position, as described earlier.

Diversity loss and linear collapse: One of the principle characteristics of PSO is convergence to a stable point as proved by Van den Bergh and Engelbrecht [100], Clerc and Kennedy [20], and Trelea [97]. However, as pointed out by Van den Bergh [100], this point is not necessarily an optimum. A shrinking swarm is synonymous with loss of diversity. If the optimum moves away, but the new position is still within the collapsing swarm, there is a good chance that the swarm will successfully track the moving target [6]. If the new target is significantly far from the collapsing swarm, the low velocities of the particles will inhibit re-diversification and tracking. Van den Bergh and Engelbrecht [100], and Blackwell and Bentley [8] have found that the swarm can even oscillate around a false attractor and along a line perpendicular to the true optimum, a feature known as linear collapse.

Blackwell and Branke categorize approaches to counterbalance the effect of diversity loss as follows [9]:

Introduce diversity after the problem has changed: The entire, or part of the swarm is randomized after each function change, or at some predetermined time interval. Such an approach might have the effect that useful information such as positions with good fitness is lost.

Maintain diversity throughout the run: Blackwell and Bentley [8] have introduced the charged PSO, an attempt to maintain diversity throughout the run. A nucleus of neutral particles act like a conventional PSO exploring the neighbourhood to locate an optimal solution. In addition, a roaming swarm of "charged" particles, that is, particles that are repelled by a subswarm converging on a peak, is maintained to detect new peaks in the search space. In a later innovation, Blackwell and Branke [10] introduced the

idea of quantum particles that move to random positions around the swarm's global best position.

Maintain multiple swarms on different peaks: Blackwell and Branke [9] constructed interacting multi-swarms by extending the single population PSO as well as the charged particle swarm optimization method. A number of subswarms are maintained around local optima that have been located, while a swarm of charged particles search for new local optima, where, if discovered, new subswarms are established. Functions where the landscapes consist of several peaks and changes to the locations, heights and widths of the peaks are small, are expected to respond well to such a strategy. If peak heights change in such a way that a suboptimal solution becomes the solution with the best fitness in the search space, it is not difficult to keep track of the global optimum. A further refinement entails sustaining diversity within subswarms by incorporating the strategy used by quantum particles.

Blackwell and Branke extended the idea of multi-swarms by proposing two forms of swarm interaction, namely *exclusion* and *anti-convergence* [7]. Exclusion prevents swarms from settling on the same peak by re-initializing the lesser swarm if two swarms move too close to one another. Anti-conversion re-initializes the worst swarm in order to track down any new peak that may appear.

Although the purpose of the multiswarms that are maintained on different peaks is to track the overall optimal solution in a dynamic environment, the creation of sub-swarms *per se* is relevant to multimodal optimization. Tracking multiple dynamic optima in parallel exhibit the same advantages as the multi-swarm approach, but starts from another premise.

The next section describes how a multimodal particle swarm optimizer can be adapted for dynamic environments.

7.5 Tracking multiple optima in a dynamic environment

In multidimensional systems changes can occur in one or more dimensions, independently or simultaneously. Changes occurring in a problem space with many suboptimal solutions can also have the effect that a suboptimal solution becomes the optimal solution and vice versa. Thus, even if the severity is small, the location of the optimum may change considerably. In most cases this problem will not occur if a niching algorithm is adapted for dynamic systems,

where the objective is to keep track of all optima. The best solution can easily be selected from all the tracked optima at any stage during the optimization process. A possible exception can occur if a new optimum appears and immediately becomes the optimal solution. However, once the new optimum is detected and added to the collection of optima, the process will proceed normally.

7.5.1 A dynamic test function generator

Dynamic environments can be generated by the moving peaks benchmark (MPB). De Jong and Morrison's test problem generator [25] was devised to be used in the study of evolutionary algorithm performance in changing environments. The problem generator is also suitable to evaluate PSO performance in dynamic environments, especially if the problem space contains a number of suboptimal solutions as well. An environment in two dimensions consisting of a number of cone shapes is generated by the following equation:

$$f(x_1, x_2) = \max_{i=1,\ldots,N} \left(H_i - R_i \sqrt{(x_1 - x_{1_i})^2 + (x_2 - x_{2_i})^2} \right) \tag{7.3}$$

where the height of each cone is given by H_i, the slope by R_i and the position by (x_{1_i}, x_{2_i}). The number of optima, their positions, shapes and heights can be specified. An environment with three cones is illustrated in Figure 7.1.

7.5.2 PSO models for multiple dynamic optima

Particle swarm optimizers that locate multiple optima in parallel provide natural mechanisms for adapting to dynamic environments. The species-based PSO [57] described in Chapter 4 is eminently suitable for such a modification.

Parrott and Li [70] adapted the algorithm to track multiple peaks simultaneously. The resulting algorithm, referred to as the dynamic species-based PSO, is based on the static species-based PSO where the particle with the best fitness is identified as a species seed and a subpopulation is formed with particles within a set speciation radius, r. The process is repeated until no more particles remain. The result is a number of subpopulations, each with its own species seed which is the particle with the best fitness in the subpopulation. At each iteration of the algorithm, particles are updated and subpopulations reformed. If the function changes, species seeds will be identified at new positions, thus providing a natural way in which the positions of peaks in the landscape can be tracked. To allow particles to move towards the new

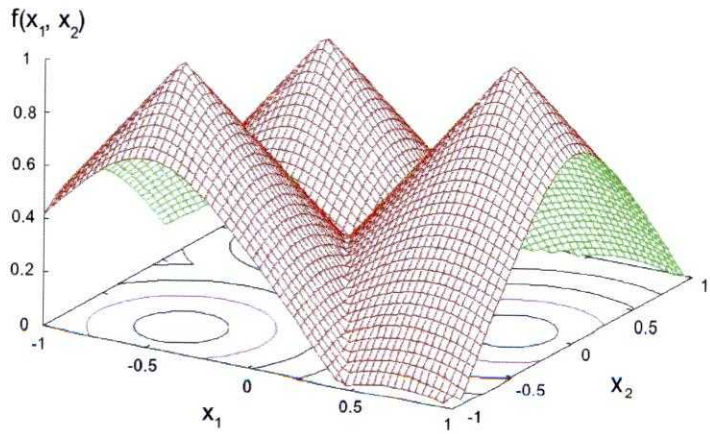


Figure 7.1: An environment with three peaks at positions $(-0.5, -0.7)$, $(-0.5, 0.5)$ and $(0.5, 0)$, generated by Morrison and De Jong's test problem generator

species seeds, the dynamic SPSO re-evaluates each particle's personal best fitness value before updating the positions of the particles. Crowding at known optima had to be prevented so that some particles could be allocated to search for new optima. Such a strategy entails introducing a maximum species population parameter. Only the best candidate members are allocated as members of a species, while redundant particles are re-initialized at random positions in the solution space.

The dynamic SPSO was extensively tested on functions generated by De Jong and Morrison's dynamic test function generator [25] [70]. Several parameters were varied and results reported. These results indicate that the algorithm can successfully track optima in a dynamic two-dimensional environment.

Li *et al.* proposed an improved dynamic species-based PSO that incorporates techniques to improve its tracking ability [59]. Some of these techniques were suggested by the multi-swarm approach of Blackwell and Branke discussed in section 6.4. Inspired by the quantum swarm model, some particles, the so-called "quantum" particles, are positioned around each peak to maintain diversity. An *anti-convergence* method to re-randomize the worst swarm when all species have converged is also adopted by the species-based PSO. The method is invoked if the average particle diversity falls below a certain threshold. Different regions of the search space can then be explored to search for new peaks. The SPSO was tested with a variety of scenarios set up by the moving peak benchmark [25]. The results suggested that SPSO can adapt well

on most test cases.

7.6 The dynamic vector-based particle swarm optimizer

This section describes how the vector-based PSO (referred to as the VBPSO) is extended to locate and track optima in a dynamic environment in parallel.

The proposed dynamic vector-based PSO uses the VBPSO to find initial multiple optima. These optima have to be tracked when the objective function changes, causing optima to move to other positions. For a tracking strategy to be efficient, all optima should be found after each objective function change with less effort than re-optimization would entail. The dynamic VBPSO endeavours to retain useful information such as previous positions of optima after each modification of the objective function, thus avoiding complete re-optimization. If changes are not too severe, the positions where the previous optima were located, are retained and act as starting points to track optima that have moved away. Thus fewer particles are required, making the process more efficient. For severe changes it could be argued that no benefit can be derived from previous optimal positions and that complete re-optimization would be preferable.

Each time the objective function is modified and the landscape changes, a tracking mechanism is invoked. In the dynamic vector-based PSO this mechanism consists of two stages. Stage 1 tracks existing optima that have moved away, or of which the fitness at one or more of the peaks have been modified. Provided the severity is not too large, optima can be tracked with relatively little effort. Stage 2 of the algorithm contains a technique to locate new optima that may have appeared.

The process to locate and track optima in a dynamic environment is described below while algorithm 12 formally presents the vector-based PSO algorithm for multiple dynamic optima.

Locate niches and optimize: After the swarm had been initialized by creating an appropriate number of particles, the static vector-based PSO as described in chapter 5, is used to locate and demarcate niches sequentially and optimize these niches in parallel. A small problem-dependent parameter, the *granularity*, described in section 5.5.2, is set in advance to facilitate niching. Thus the initial positions and fitnesses of all the optima are obtained. Faster changes in the environment may result in less accurate optimal positions.

Track existing optima: In a dynamic environment optima are tracked by finding new optimal positions each time a modification of the objective function is detected, or at specific

Algorithm 12 The dynamic vector-based PSO

```
begin
    Initialize the swarm by creating n particles;
    Locate niches and optimize using the vector-based PSO;
    while function is changing do
        if function has been modified then
            Stage 1:
            Discard all particle information except previous optimal positions;
            Create particles at previous optimal positions;
              Establish new personal best positions (pbest);
              Set each neighbourhood best position (nbest) to corresponding pbest;
              Update position vectors towards pbest, nbest and and their dot
              products;
            Create 3 additional particles near each optimal position to form small
            subswarms;
            Optimize subswarms in parallel to converge on new positions;
            if a peak has disappeared then
                Merge niches;
            end
            Stage 2:
            Discard all particle information except optimal positions found in stage 1;
            Create n particles over entire search space;
            for all existing niches do
                Find niche radii;
                Mark particles within niche radii as belonging to the corresponding
                niche;
            end
            for particles not assigned to a niche do
                repeat
                    Find particle with best pbest;
                    Set nbest of all particles to best pbest;
                    Find niche radius;
                    Mark particles within niche radius as belonging to the corresponding
                    niche;
                until all particles have been included in a niche;
                Optimize niches in parallel and merge if necessary;
            end
        end
    end
end
```

intervals if the function changes continuously. At this stage a number of particles will have converged on each previous optimal position. Redistribution of these particles throughout the search space is instrumental in tracking optima when the environment changes. However, if the locations of the optima do not change too much, only a few particles in the vicinity of each existing optimum are required to track the optima to their new positions.

The dynamic vector-based PSO employs a strategy where only the existing neighbourhood best position of each niche is retained when the objective function is modified and optima need to be tracked. All other particle information is discarded. After the objective function has changed, a new particle is created at each previous neighbourhood best position. Similar to the procedure the static vector-based PSO employs to create initial particles, a personal best position is associated with the particle at each previous neighbourhood position. A random position is found near to each particle and the personal best is set to the fittest of the two positions while the less fit position becomes the particle position. The neighbourhood best position associated with each of these particles is set equal to the personal best position of that particle. Such steps are necessary since the personal best position of each previous optimum has to be re-evaluated in a changed environment. At this stage one particle has been created at each previous neighbourhood best position so that the number of particles is equal to the previous number of optima.

To enable the algorithm to track an optimum, a small number of additional particles are created in the immediate vicinity of each particle created at a previous optimal position. These particles are initialized at random positions within a radius r from the neighbourhood best particle. The value of r is equal to the *granularity* that has been set in advance for the objective function that is being optimized. Thus a small subswarm is formed in each niche. Appendix A.3 presents empirical results showing that a subswarm consisting of the original neighbourhood best particle as well as three additional particles, giving a total of four per niche, is able to effectively track a moving optimum. New personal best positions are updated. The neighbourhood best position of each additional particle is set to the neighbourhood best position of the first particle in each niche. Vectors pointing to the personal best and neighbourhood best positions, as well as their dot product, are calculated.

To locate modified optimal positions, the subswarms thus created, are optimized in parallel. During the optimization process subswarms naturally move towards better positions.

It may happen that an optimum moves into a neighbouring niche and disappears. Re-evaluation of the personal best positions of the subswarm associated with the peak that disappeared causes that subswarm to move towards the neighbouring optimum where the two subswarms will eventually merge.

The strategy used to track existing optima is referred to as stage 1 of the algorithm in the formal description of the algorithm. New optima appearing in the landscape during modification of the objective function, are not discovered during stage 1.

Locate new optima: New optima may appear when the objective function changes. Locating these optima is not a simple matter, as the initial strategy to find niches must basically be repeated to find new candidate solutions. The dynamic VBPSO algorithm includes a mechanism to locate new optima. This mechanism is invoked after stage 1 of the algorithm and is referred to as stage 2. These stages are invoked each time the objective function is modified or at specific intervals in the process. Stage 2 of the algorithm only retains the positions of the optima located during stage 1. All other particles and their associated information are discarded. Similar to the static VBPSO that was used to locate initial optima for one run of the algorithm, new particles are created at random positions throughout the search space. The number of particles is set equal to the original number created in the first phase of the algorithm. Niches are established by using the vector dot product to calculate niche radii. However, instead of repeating the entire sequential process to find candidate solutions, niche radii of the existing optima are first calculated. Particles falling within those radii, are marked as belonging to that niche. If a new peak has appeared near to an existing optimum, the niche radius of the existing optimum will be smaller and not include particles expected to converge on the new peak. To reduce the effort of finding new optima, the subswarms thus formed, are not optimized again. The sequential process to find candidate solutions and calculate their niche radii is then resumed for the remaining particles. Only subswarms formed by these remaining particles are optimized during the optimization phase. If new optima have appeared, subswarms would have formed around those optima on which it will converge during optimization. However, if no new optima have appeared after the last modification of the objective function, particles outside the niches surrounding the existing optima will form subswarms adjacent to those niches. These subswarms reside in false niches that exhibit a tendency to move into existing niches, and will eventually converge on existing optima.

Clearly, the process to locate new optima is computationally more expensive than the

process to track existing optima. The question arises whether complete re-optimization at set intervals might not be considered as a workable alternative. The next section investigates these possibilities.

7.7 Conclusion

This chapter presented a technique to extend the vector-based particle swarm optimizer to be applicable to dynamic environments. Particle swarm models for tracking a single solution in a dynamic environment were discussed while an overview of PSO models for multiple dynamic optima was presented. The vector-based PSO for multiple dynamic optima was described and formally presented as the dynamic vector-based PSO algorithm. Demonstration of its ability to track multiple moving optima in a number of selected dynamic environments is presented in chapter 8.

Chapter 8

The Vector-Based PSO Applied to Dynamic Environments

This chapter presents results yielded by the dynamic vector-based PSO on the moving peaks benchmark function [25]. A number of scenarios illustrating a variety of dynamic changes are designed. The dynamic species-based PSO is tested for the same scenarios, and the performances of the two algorithms are compared. The chapter concludes with an investigation into the sensitivity of the dynamic vector-based PSO to changes in severity.

8.1 Introduction

To test the dynamic vector-based PSO exhaustively for all possible moving optima is a major undertaking. Ideally, new optimal positions should be located each time any change in the environment occurs. Environmental changes are brought about in different ways, often related to the problem that the algorithm tries to solve. Some tracking algorithms incorporate ways to detect changes, which may not always be necessary, as the algorithm may be informed when input parameters change. For some problems adaptation of optimal positions at specific intervals may suffice. The dynamic vector-based PSO does not detect the changes, but assume that the algorithm is triggered in some way when a change in the landscape occurs.

This chapter reports on a number of separate scenarios. Each scenario illustrates some

way in which the landscape changes. For each scenario a number of peaks was chosen and movements over the search space set up, using the moving peaks benchmark function (refer to section 7.5.1). Changes include changes in the position of the peaks, changes in the fitness, and situations where peaks disappear and appear again at different positions.

8.2 Experimental setup and results

The following settings were used for all experiments:

- Experiments are performed in a two-dimensional search space in the range $x_1, x_2 \in [-1, 1]$.
- The initial number of particles is set to 30 for all experiments.
- A uniform distribution of particles throughout the search space is ensured by using Sobol sequences as a random number generator. Sobol sequences are described in chapter 5 and are used throughout this study.
- The granularity parameter is set to 0.05 for all the experiments. The value was estimated taking into consideration the size of the search space and the number of expected peaks. According to the conclusions derived at in section 5.7.1, the granularity should be smaller than the smallest interniche distance in order to locate all optima. In this case it can be stated that the algorithm should locate all optima where the Euclidian distance separating them, is more than 0.05.
- The random sequences used in the velocity update equation, $r_1 \sim U(0, 1)$ and $r_2 \sim U(0, 1)$ are scaled by constants $c_1, c_2 \in [0, 2]$. Similar to experiments with the vector-based PSO described in chapter 5, $c_1 = c_2 = 1$, that is, the maximum step size in the direction of the neighbourhood best position, referred to by c_2, is equal to c_1, the maximum step size in the direction of the personal best position.
- The inertia weight, w, is used as a scaling factor associated with the velocity in the previous time step. As in the case of the vector-based PSO, w is set to the lower bound, 0.8.
- After each modification to the objective function that results in a changed function landscape, all previous particles are discarded and small subswarms, each consisting of a newly initialized particle at the previous optimal position, as well as three additional particles

near to that position (stage 1). During stage 2, 30 particles are initialized throughout the search space. These subswarms are optimized by repeating the updating process 500 times before the next environment change.

- The algorithm is run 50 times for each experiment.

Results are presented as follows:

Function evaluations: During each optimization phase, the number of function evaluations of the objective function was counted and recorded. Results are presented as the average number of evaluations for the first step as well as the subsequent steps of the algorithm, where a step indicates the part of the algorithm that is executed before the next objective function change. The average number of evaluations over 50 runs was calculated separately for the first step of the algorithm (when optima are located) and the subsequent steps (where optima are tracked).

Average number of optima: The average number of optima located over 50 runs, is listed for each step of the algorithm.

Offline error: The offline error, or average distances from the true optimal positions over 50 runs, is calculated and listed for each step of the algorithm. For objective functions tested in previous chapters, partial derivatives were used, but when a function contains cone-shaped peaks, the derivative is equal to the slope of the cone, except at the exact optimal position where it is 0. Therefore, for the moving peaks benchmark, the use of the offline error is a better indication of the quality of the solution.

Success rate: The success rate is the total number of optima located over 50 runs calculated as a percentage of the total number of possible optima. Again, the success rate is reported for each step of the algorithm.

In the following section a number of scenarios are described, the function landscapes at each step illustrated, and results presented.

Scenario 1

Using De Jong and Morrison's test problem generator [25] described by equation (7.3), a test function with three peaks was created. The positions of the peaks change over six steps in such a way that no two peaks merge with one another. Table A.4 lists the settings of the function

for each step, while Figure 8.1 illustrates the function landscape at each step. At each step the function landscape shows three separate peaks. Peaks are labeled $P1$ to $P3$ to correspond with columns in the table. Changes in the function landscape are described as follows:

Peak 1: The initial position of the peak is at $[-0.6, -0.8]$. The x_2-component remains the same, while the x_1-component is incremented by 0.2 at each step, resulting in a movement parallel to the x_2 axis, stopping at $[0.4, -0.8]$. The height and radius of the peak remains the same.

Peak 2: Movement of peak 2 commences at $[-0.5, 0.3]$. The x_1-component is incremented by 0.2 and the x_2-component by 0.1 at each step. The peak moves across the space towards $[0.5, 0.8]$ in the adjacent quadrant. The height and radius of the peak remains the same.

Peak 3: The starting position of peak 3 is at $[0.5, 0]$. The x_1-component is decremented by 0.1 at each step while the x_2-component does not change. The peak moves along the x_1 axes and stops at the centre of the search space at $[0, 0]$.

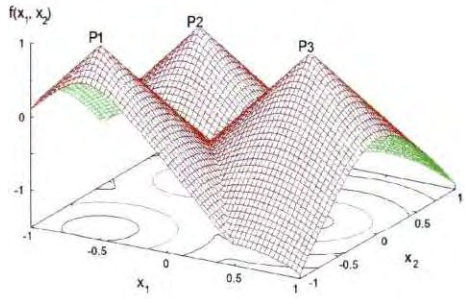
Scenario 1 illustrates simple movements of optima across the search space. Since it is assumed that no new peaks appear, only stage 1 of the algorithm is activated. Once the initial optima have been found, fewer particles are required to track the optima, depending on the number of optima. Results presented in Table 8.2 show that the average number of function evaluations during the last 5 steps of all the experiments is less than one third of the average number of evaluations required to locate the initial optima. Therefore it can be concluded that, once optima are located, the computational complexity required by the dynamic vector-based PSO to track existing optima is much less than frequent re-optimization would entail, given that the severity is not too large. According to the results, the offline error is less than 10^{-12}, indicating very good quality solutions.

Table 8.1: Scenario 1: Positions of 3 optima over 6 steps

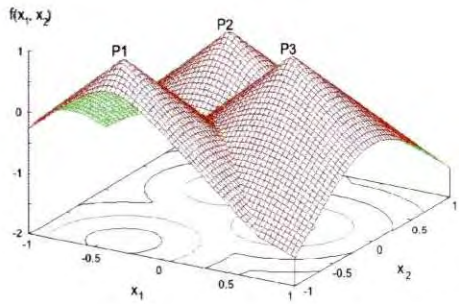
Step	Peak 1				Peak 2				Peak 3			
	x_1	x_2	H	R	x_1	x_2	H	R	x_1	x_2	H	R
1	-0.6	-0.8	1	2	-0.5	0.3	0.6	2	0.5	0	0.8	2
2	-0.4	-0.8	1	2	-0.3	0.4	0.6	2	0.4	0	0.8	2
3	-0.2	-0.8	1	2	-0.1	0.5	0.6	2	0.3	0	0.8	2
4	0	-0.8	1	2	0.1	0.6	0.6	2	0.2	0	0.8	2
5	0.2	-0.8	1	2	0.3	0.7	0.6	2	0.1	0	0.8	2
6	0.4	-0.8	1	2	0.5	0.8	0.6	2	0	0	0.8	2

Table 8.2: Scenario 1: Results

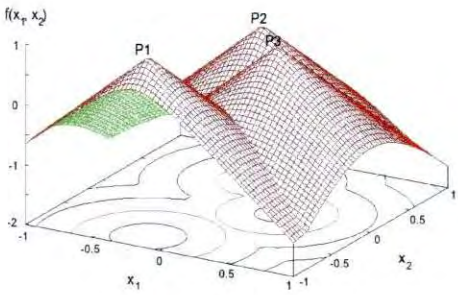
Step	Average ♯ evaluations	Average ♯ optima	Offline error	Success rate
1	25558 ± 490	3 ± 0	1.77E-13 ± 1.22E-13	100%
2	6934 ± 12	3 ± 0	7.16E-18 ± 1.57E-18	100%
3	6922 ± 7	3 ± 0	4.82E-18 ± 6.82E-19	100%
4	6927 ± 10	3 ± 0	3.47E-17 ± 5.94E-18	100%
5	6961 ± 30	3 ± 0	3.12E-18 ± 4.77E-19	100%
6	6924 ± 10	3 ± 0	1.94E-17 ± 4.24E-18	100%



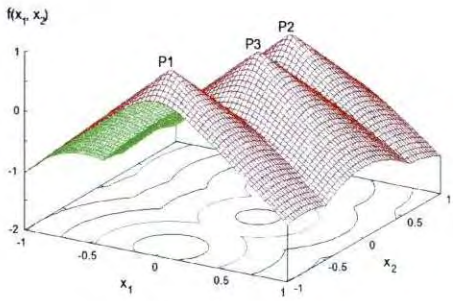
(a) Step 1



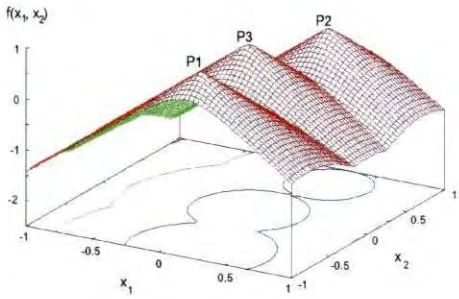
(b) Step 2



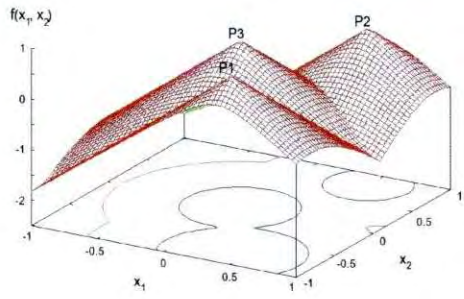
(c) Step 3



(d) Step 4

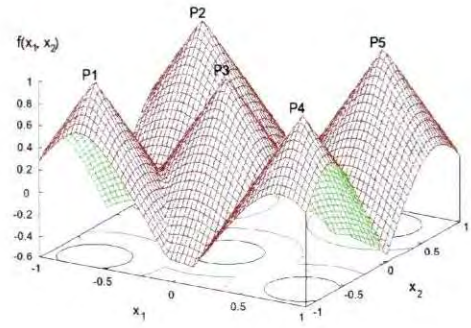


(e) Step 5

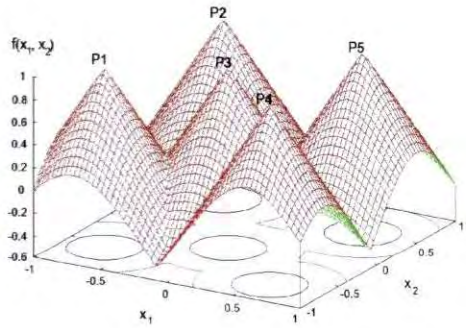


(f) Step 6

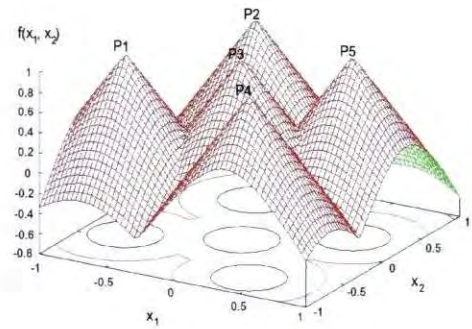
Figure 8.1: Scenario 1: Function landscapes



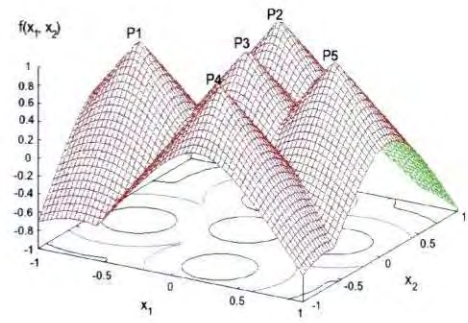
(a) Step 1



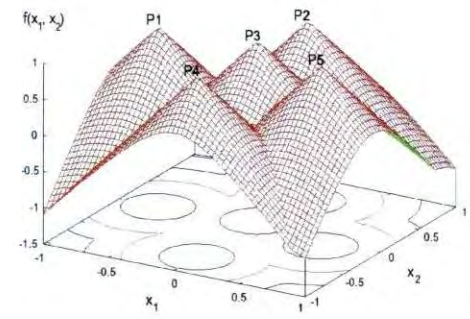
(b) Step 2



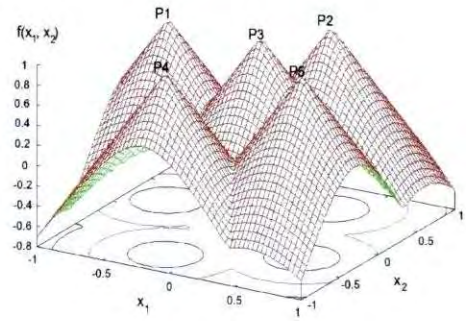
(c) Step 3



(d) Step 4

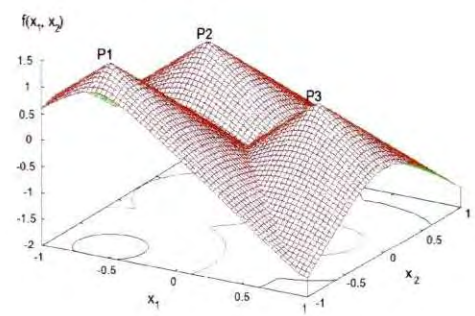


(e) Step 5

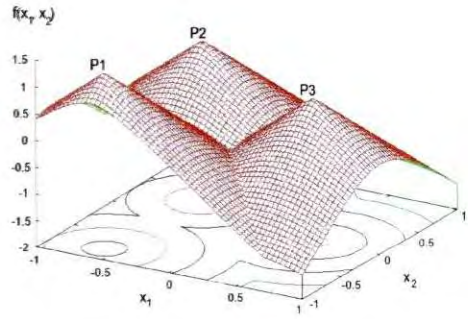


(f) Step 6

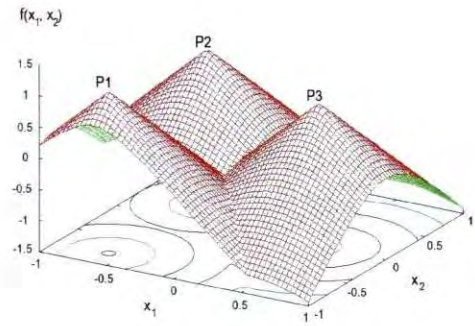
Figure 8.2: Scenario 2: Function landscapes



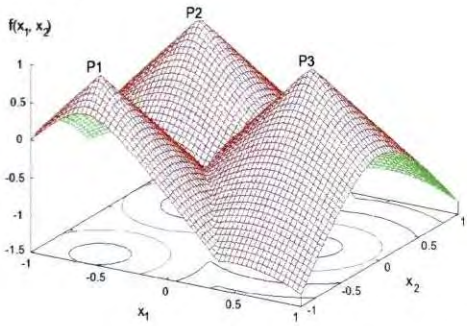
(a) Step 1



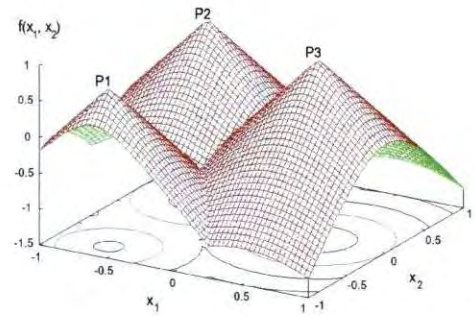
(b) Step 2



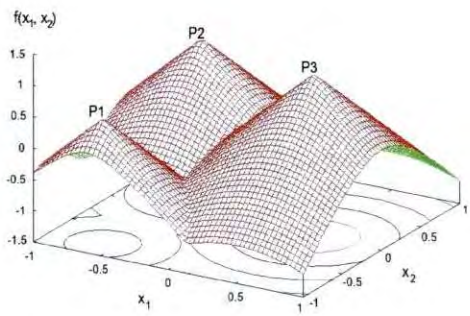
(c) Step 3



(d) Step 4

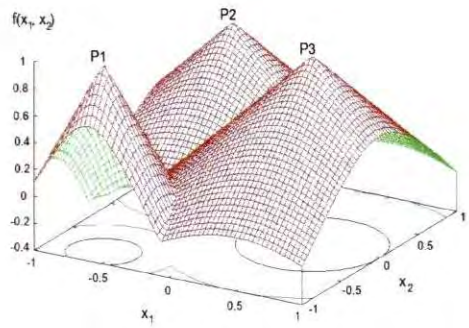


(e) Step 5

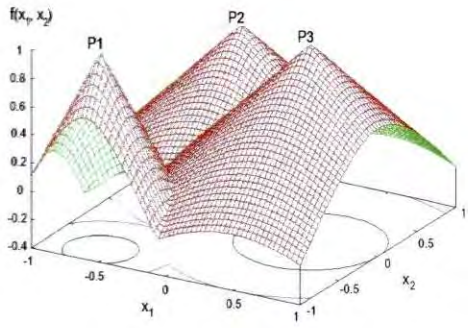


(f) Step 6

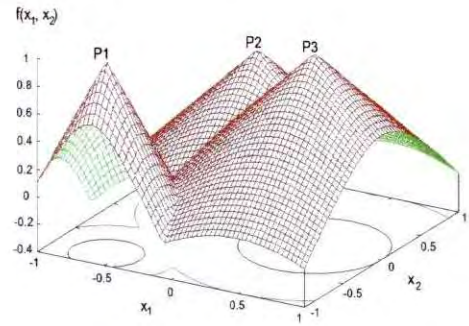
Figure 8.3: Scenario 3: Function landscapes



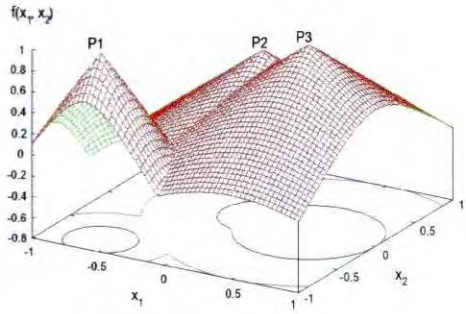
(a) Step 1



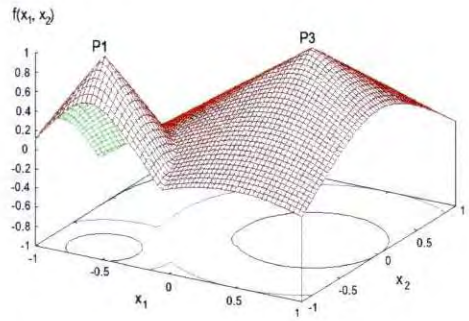
(b) Step 2



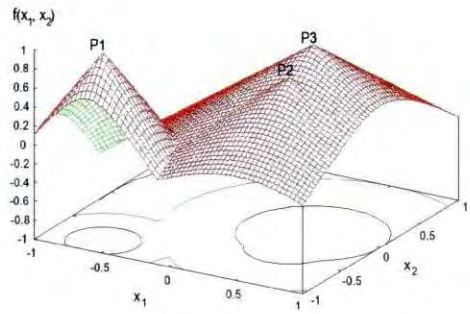
(c) Step 3



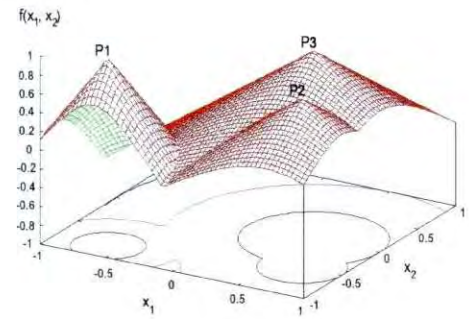
(d) Step 4



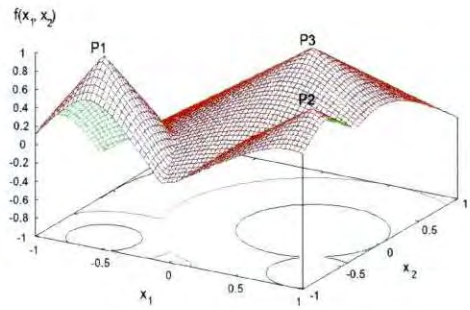
(e) Step 5



(f) Step 6



(g) Step 7



(h) Step 8

Figure 8.4: Scenario 4: Function landscapes

Scenario 2

A second test function was created where no peaks have disappeared and no new ones appear, but the number of peaks is increased to five. Changes in the objective function and therefore the positions of the optima, take place over six steps. Table 8.3 lists the settings of the function for each step. Figure 8.2 illustrates the function landscape at each step, indicating the peaks as $P1$ to $P5$. Movement of these peaks across the function landscape is described as follows:

Peak 1: The initial position of peak 1 is at $[-0.7, -0.8]$. Movement takes place parallel to the x_2-axis as the x_2-component is incremented by 0.2 at each step while the x_1-component remains the same. Movement stops at position $[-0.7, 0.2]$. The heights and radii of all peaks in this scenario remain the same.

Peak 2: Peak 2 moves parallel to the x_2-axis, from $[-0.7, 0.6]$ to $[0.3, 0.6]$, the x_1-component being incremented by 0.2 at each step.

Peak 3: Peak 3 moves along the x_1-axis, starting at position $[0, -0.3]$ to position $[0, 0.2]$. The x_2-component is incremented by 0.1.

Peak 4: This peak describes another movement parallel to the x_1-axis, but in the opposite direction from peaks 2 and 3. Movement starts at position $[0.8, -0.7]$ and stops at $[-0.3, -0.7]$. The x_1 component is decremented by 0.2 at each step.

Peak 5: Finally, peak 5 describes another movement parallel to the x_2-axis, but in the opposite direction from that of peak 1. The starting position is at $[0.6, 0.7]$ and movement stops at $[0.6, -0.3]$, meaning that the x_2-component is decremented by 0.2 at each step.

Table 8.4 presents the results. Similar to scenario 1, only stage 1 of the algorithm is used. The average number of function evaluations required to locate the initial optima, do not differ much from that of scenario 1. However, because the number of optima that are tracked, is increased to 5, the average number of function evaluations during the last 5 steps increases, but only to about half of the number required to find the initial optima. Therefore the same conclusion can be reached as in scenario 1.

Table 8.3: Scenario 2: Positions of 5 optima over 6 steps

Step	Peak 1				Peak 2				Peak 3				Peak 4				Peak 5			
	x_1	x_2	H	R	x_1	x_2	H	R	x_1	x_2	H	R	x_1	x_2	H	R	x_1	x_2	H	R
1	-0.7	-0.8	1	2	-0.7	0.6	1	2	0	-0.3	1	2	0.8	-0.7	1	2	0.6	0.7	1	2
2	-0.7	-0.6	1	2	-0.5	0.6	1	2	0	-0.2	1	2	0.6	-0.7	1	2	0.6	0.5	1	2
3	-0.7	-0.4	1	2	-0.3	0.6	1	2	0	-0.1	1	2	0.4	-0.7	1	2	0.6	0.3	1	2
4	-0.7	-0.2	1	2	-0.1	0.6	1	2	0	0	1	2	0.2	-0.7	1	2	0.6	0.1	1	2
5	-0.7	0	1	2	0.1	0.6	1	2	0	0.1	1	2	0	-0.7	1	2	0.6	-0.1	1	2
6	-0.7	0.2	1	2	0.3	0.6	1	2	0	0.2	1	2	-0.2	-0.7	1	2	0.6	-0.3	1	2

Table 8.4: Scenario 2: Results

Step	Average ♯ evaluations	Average ♯ optima	Offline error	Success rate
1	23068 ± 453	5 ± 0	8.48E-18 ± 3.30E-18	100%
2	11583 ± 11	5 ± 0	2.32E-17 ± 1.97E-17	100%
3	11578 ± 14	5 ± 0	2.81E-18 ± 5.84E-19	100%
4	11556 ± 12	5 ± 0	7.10E-18 ± 1.30E-18	100%
5	11566 ± 14	5 ± 0	2.70E-17 ± 2.88E-18	100%
6	11533 ± 13	5 ± 0	2.34E-18 ± 5.04E-19	100%

Scenario 3

Scenario 3 portrays a dynamic environment where the fitness of the optima undergo changes, but the positions stay the same. Optima are tracked over 6 steps, using only stage 1 of the algorithm, that is, no new peaks are located. Such changes may cause the global optimal position to shift from one peak to another. Table 8.5 lists the settings of the function for each step. Figure 8.3 illustrates the function landscape at each step, indicating the peaks as $P1$ to $P3$ to correspond with the column headings in Table 8.5, namely Peak 1, Peak 2 and Peak 3. Changes in the peaks are described as follows:

Peak 1: Peak 1 is positioned at $[-0.6, -0.8]$ throughout the run, but the height of the peak, H decreases from 1.5 to 0.5. The slope of the peak, R, stays the same at 2, meaning that the size of the base of the peak increases.

Peak 2: Peak 2 does not change during the run. The position remains at $[-0.5, 0.3]$, the height, H, is equal to 1 throughout, while the slope, R, remains at 2.

Peak 3: The position of peak 3 as well as the slope, R, remain at $[0.5, 0]$ and 2. The height, H, increases from 0.6 to 1.1.

Table 8.6 presents the results. The current global optima are also indicated. This scenario illustrates one of the benefits of tracking multiple optima in a multi-modal environment: If

a current suboptimal solution becomes the solution with the best fitness as the environment changes, no additional effort is required to keep track of the global optimum.

Table 8.5: Scenario 3: Three optima, stationary positions, fitness changes

Step	Peak 1				Peak 2				Peak 3			
	x_1	x_2	H	R	x_1	x_2	H	R	x_1	x_2	H	R
1	-0.6	-0.8	1.5	2	-0.5	0.3	1	2	0.5	0	0.6	2
2	-0.6	-0.8	1.3	2	-0.5	0.3	1	2	0.5	0	0.7	2
3	-0.6	-0.8	1.1	2	-0.5	0.3	1	2	0.5	0	0.8	2
4	-0.6	-0.8	0.9	2	-0.5	0.3	1	2	0.5	0	0.9	2
5	-0.6	-0.8	0.7	2	-0.5	0.3	1	2	0.5	0	1	2
6	-0.6	-0.8	0.5	2	-0.5	0.3	1	2	0.5	0	1.1	2

Scenario 4

Scenario 4 illustrates a situation where the positions of three optima change over eight steps. In the course of the movement, one of the peaks is obscured by a larger peak, but appears again when it moves out from under the higher peak. Table 8.7 shows the settings of the function for each step, and Figure 8.4 illustrates the function landscapes. Peaks are indicated as $P1$ to $P3$ to correspond to peak 1, peak 2 and peak 3 in the table. The movement of these peaks across the function landscape is described below:

Peak 1: The initial position of the peak is at $[-0.6, -0.8]$. The x_2-component remains the same, while the x_1-component is incremented by 0.2 at each step, resulting in a movement parallel to the x_2 axis, stopping at $[0.4, -0.8]$. The height and radius of the peak remains the same.

Peak 2: Movement of peak 2 commences at $[-0.5, 0.3]$. The x_1-component is incremented by 0.2 and the x_2-component by 0.1 at each step. The peak moves across the space towards $[0.5, 0.8]$ in the adjacent quadrant while the height and radius of the peak remains the

Table 8.6: Scenario 3: Results

Step	Average ♯ evaluations	Average ♯ optima	Offline error	Success rate	Global optimum
1	24699 ± 510	3 ± 0	1.37E-15 ± 1.16E-15	100%	Peak 1
2	6580 ± 9	3 ± 0	8.77E-18 ± 4.48E-18	100%	Peak 1
3	6571 ± 9	3 ± 0	5.21E-18 ± 9.27E-19	100%	Peak 1
4	6565 ± 10	3 ± 0	5.18E-18 ± 9.02E-19	100%	Peak 2
5	6553 ± 9	3 ± 0	1.35E-17 ± 8.09E-18	100%	Peak 2,3
6	6559 ± 10	3 ± 0	9.14E-18 ± 1.66E-18	100%	Peak 3

same. During the movement peak 2 moves underneath peak 3 and disappears from view, but the next step shows the peak starting to appear on the other side of the larger peak. In subsequent steps the peak can be clearly seen.

Peak 3: The starting position of peak 3 is at $[0.5, 0]$. The x_1-component is decremented by 0.1 at each step while the x_2-component does not change. The peak moves along the x_1-axis and stops at the centre of the search space at $[0, 0]$.

Scenario 4 is used in two experiments. In the first experiment, only stage 1 of the algorithm is activated. Table 8.8 presents the results, showing that the algorithm is capable of merging subswarms tracking optima which might have disappeared. Results show that only two optima are located during steps 5 and 6. In step 5 two subswarms are merged. In step 6 only two optima are tracked, which is reflected in the results showing a significantly smaller average number of function evaluations.

The second experiment using this scenario utilizes the full capacity of the dynamic vector-based PSO. Where stage 1 of the algorithm tracks existing optima, merging subswarms if necessary, stage 2 explores the problem space to detect new optima. Coverage of the entire search space is implied, requiring additional computational complexity. Three peaks are tracked over 8 steps. Table 8.9 presents the results.

Results show that 3 optima are located and tracked over the next 3 steps. One peak then disappears and over the next 2 steps only 2 optima are tracked. Step 6 shows a change in the landscape where a small peak appears in the next step. This peak is located 86.67% of the time. In the last step the peak is located in all runs of the algorithm. However, when stage 2 of the algorithm is activated, the average number of function evaluations exceeds that of the initial phase to locate optima. It would seem that the dynamic vector-based PSO is less efficient than re-optimization if a procedure to detect new optima is included.

Discussion

The four scenarios tested in this section demonstrated strengths and weaknesses of the vector-based PSO. The algorithm is capable of tracking existing optima in an accurate and efficient manner. Optimal solutions that disappear are handled in an elegant and efficient manner. Results show that tracking of optima requires less computational complexity than re-optimization. However, exploring the entire search space for new optima that may appear, is computationally more expensive that re-optimization.

Table 8.7: Scenario 4: Positions of 3 optima over 8 steps - 1 peak obscured and appears again

Step	Peak 1				Peak 2				Peak 3			
	x_1	x_2	H	R	x_1	x_2	H	R	x_1	x_2	H	R
1	-0.6	-0.8	1	2	-0.5	0.7	0.8	1	0.5	0	1	1
2	-0.6	-0.8	1	2	-0.3	0.5	0.8	1	0.5	0	1	1
3	-0.6	-0.8	1	2	-0.1	0.3	0.8	1	0.5	0	1	1
4	-0.6	-0.8	1	2	0.1	0.1	0.8	1	0.5	0	1	1
5	-0.6	-0.8	1	2	0.3	-0.1	0.8	1	0.5	0	1	1
6	-0.6	-0.8	1	2	0.5	-0.3	0.8	1	0.5	0	1	1
7	-0.6	-0.8	1	2	0.7	-0.5	0.8	1	0.5	0	1	1
8	-0.6	-0.8	1	2	0.9	-0.7	0.8	1	0.5	0	1	1

Table 8.8: Scenario 4: Results using stage 1 of the VBPSO algorithm

Step	Average ♯ evaluations	♯ optima in search space	Average ♯ optima located	Offline error	Success rate
1	23899 ± 387	3	3 ± 0	2.61E-05 ± 2.61E-05	100%
2	6659 ± 9	3	3 ± 0	1.03E-17 ± 1.90E-18	100%
3	6634 ± 5	3	3 ± 0	1.03E-17 ± 1.90E-18	100%
4	6652 ± 7	3	2.8 ± 0.06	1.09E-17 ± 2.00E-18	93.33%
5	6191 ± 134	2	2 ± 0	1.40E-17 ± 2.33E-18	100%
6	4326 ± 2	2	2 ± 0	1.40E-17 ± 2.33E-18	100%
7	4326 ± 2	3	2 ± 0	1.40E-17 ± 2.33E-18	66.67%
8	4324 ± 2	3	2 ± 0	1.40E-17 ± 2.33E-18	66.67%

Table 8.9: Scenario 4: Results using stage 1 and stage 2 of the VBPSO algorithm

Step	Average ♯ evaluations	♯ optima in search space	Average ♯ optima located	Offline error	Success rate
1	24441 ± 504	3	3 ± 0	4.19E-15 ± 4.15E-15	100%
2	31692 ± 450	3	3 ± 0	5.59E-09 ± 5.59E-09	100%
3	34043 ± 391	3	2.92 ± 0.04	1.33E-05 ± 1.22E-05	97.33%
4	33443 ± 763	3	2.48 ± 0.07	4.69E-06 ± 3.97E-06	82.67%
5	28151 ± 354	2	2 ± 0	3.71E-12 ± 3.71E-12	100%
6	28554 ± 519	2	2 ± 0	1.69E-10 ± 1.35E-10	100%
7	29914 ± 657	3	2.54 ± 0.07	7.87E-13 ± 7.84E-13	84.67%
8	31750 ± 479	3	3 ± 0	1.75E-06 ± 1.33E-06	100%

8.3 Comparing the performance of the dynamic vector-based PSO to that of the dynamic species-based PSO

This section compares the performance of the vector-based PSO to that of the dynamic species-based PSO developed by Parrott and Li [70]. The algorithm is described in chapter 7. The appeal of the species-based PSO lies in its elegance and simplicity. Since species seeds are established and subswarms reformed each time particle positions are updated, the algorithm provides a natural way in which optima can be tracked. Therefore the basic dynamic species-based PSO requires minimal modification. The version tested in this section implemented a procedure to re-evaluate each particle's personal best fitness value before updating particle positions. Introducing a maximum species population size and re-initializing redundant particles at random positions in the solution space was not attempted, as the population size is small and species radii differ from one another and over time.

8.3.1 Experimental settings and results

The dynamic species-based PSO was tested using the four scenarios described in the previous section. In all cases the initial population size was set to 30. For each experiment the algorithm was run 50 times. The dimensions of the search space, settings used by the update equation and the manner in which random positions of the initial particles are generated, are similar to the experiments conducted with the dynamic vector-based PSO. At each step the process to establish species seeds, and updating the velocity and particle positions once, were repeated 500 times. The species-based PSO uses a species radius that has to be set in advance. For each scenario the size of the species radius was estimated by running the first step of the algorithm a number of times with different species radius values and selecting the value yielding the best performance of the algorithm.

Scenario 1

A test function where the positions of 3 peaks change over 6 steps was created. A description of this scenario was given in section 6.7. Table 8.1 lists the settings of the function for each step, while Figure 8.1 shows the function landscape at each step. For these experiments the species radius was set to 0.7.

Table 8.10 presents the results. The first step has a success rate of 100% that decreases

over steps 2 to 4 to 83.33%, 76.67%, and 70%. The corresponding function landscapes as illustrated in Figure 8.1 show that distances between the optima decrease in these cases, that is, the species radii also decreases. Thus the decrease in performance can be ascribed to the fact that, in a multimodal landscape, different species have different species radii. This effect is more pronounced in dynamic environments, the degree depending on the objective function. The static setting of the species radius in the dynamic species-based PSO does not address this problem. Therefore the dynamic vector-based PSO has an advantage as each niche (corresponding to a species in the dynamic species-based PSO) is assigned a unique niche radius in the initial step of the algorithm, while smaller subswarms are used to track the optima.

Table 8.10: Scenario 1: Results for the dynamic species-based PSO

Step	Average ♯ evaluations	Average ♯ optima	Average ♯ extra species	Offline error	Success rate
1	15030 ± 0	3 ± 0	0.3 ± 0.08	5.22E-03 ± 1.2E-03	100%
2	30090 ± 0	2.5 ± 0.07	0 ± 0	5E-03 ± 1.32E-03	83.33%
3	45150 ± 0	2.3 ± 0.07	0 ± 0	5.27E-03 ± 1.3E-03	76.67%
4	60210 ± 0	2.1 ± 0.04	0.1 ± 0.04	6.11E-03 ± 1.41E-03	70%
5	75270 ± 0	3 ± 0	0 ± 0	1.28E-02 ± 1.84E-03	100%
6	90330 ± 0	3 ± 0	0.1 ± 0.04	1.86E-02 ± 2.1E-03	100%

Scenario 2

This scenario describes a similar situation to scenario 1, but the number of optima is increased to 5. This scenario was described in section 6.7. Table 8.3 lists the settings of the function for each step, while Figure 8.2 shows the function landscape at each step. For these experiments the species radius was set to 0.6.

Table 8.11 presents the results. Again, decreasing performance over the last 4 steps can be ascribed to a changing species radius. The explanation is similar to that of scenario 1,

emphasizing the fact that the dynamic VBPSO outperforms the dynamic SPSO when niche radii change in a dynamic environment.

Table 8.11: Scenario 2: Results for the dynamic species-based PSO

Step	Average ♯ evaluations	Average ♯ optima	Average ♯ extra species	Offline error	Success rate
1	15030 ± 0	5 ± 0	0 ± 0	1.51E-03 ± 1.84E-04	100%
2	30090 ± 0	5 ± 0	0 ± 0	5.74E-03 ± 8.86E-04	100%
3	45150 ± 0	4.9 ± 0.04	0 ± 0	7.88E-03 ± 1.04E-03	98%
4	60210 ± 0	4.8 ± 0.06	0.8 ± 0.09	1.50E-02 ± 1.27E-03	96%
5	75270 ± 0	4.9 ± 0.04	1.3 ± 0.14	2.17E-02 ± 1.49E-03	98%
6	90330 ± 0	4.6 ± 0.07	0.3 ± 0.07	1.82E-02 ± 1.44E-03	92%

Scenario 3

In this scenario only the heights of the peaks change over six steps. Positions of the optima remain the same. This scenario was described in section 6.7. Table 8.5 lists the settings of the function for each step, while Figure 8.3 shows the function landscape at each step. For these experiments the species radius was set to 0.7.

Table 8.12 presents the results, showing a 100% success rate in all cases. Good performance of the dynamic species-based PSO can be expected in this situation as the peaks do not move and the species radii do not change. If the niche radius setting yields good performance for the first step, the same can be expected for subsequent steps. The performance even gets better, as no extra species are identified in steps 3 to 6, meaning that, at that stage, all particles have moved into the regions surrounding the species seeds.

Table 8.12: Scenario 3: Results for the dynamic species-based PSO

Step	Average ♯ evaluations	Average ♯ optima	Average ♯ extra species	Offline error	Success rate
1	15030 ± 0	3 ± 0	0.7 ± 0.11	1.07E-02 ± 1.71E-03	100%
2	30090 ± 0	3 ± 0	0.2 ± 0.06	6.07E-03 ± 1.45E-03	100%
3	45150 ± 0	3 ± 0	0 ± 0	3.30E-03 ± 1.02E-03	100%
4	60210 ± 0	3 ± 0	0 ± 0	7.85E-04 ± 2.26E-04	100%
5	75270 ± 0	3 ± 0	0 ± 0	6.09E-04 ± 2.03E-04	100%
6	90330 ± 0	3 ± 0	0 ± 0	5.84E-04 ± 2.04E-04	100%

Scenario 4

In this scenario the problem is such that 3 optima move across the landscape over 8 steps. In the course of the movement one peak is obscured by an adjacent peak and appears again on the other side of the peak. This scenario was described in section 6.7. Table 8.7 lists the settings of the function for each step, while Figure 8.4 shows the function landscape at each step. For these experiments the species radius was set to 0.7.

Table 8.13 presents the results. Good performance is reported for the first two steps, but in the next two steps the performance declines. These results coincide with diminishing distances between two of the peaks and the fact that the species radius remains the same throughout. In steps 5 and 6 both optima are located, as well as an average of 1.1 extra optima, indicating that the dynamic species-based PSO does not successfully reflect the situation when peaks disappear. In steps 7 and 8 the peak that was obscured, surfaces again, but the success rate is similar to that of step 4. Locating new optima cannot be expected from this version of the dynamic species-based PSO, as the re-distribution of redundant particles have not been implemented. However, it can be expected of the algorithm to reflect the situation when a peak is obscured. From these results and the results reported by the dynamic vector-based PSO it can be concluded that the dynamic vector-based PSO performs better than the dynamic species-based PSO in situations where the number of peaks in the function landscape decreases.

Table 8.13: Scenario 4: Results for the dynamic species-based PSO

Step	Average ♯ evaluations	Average ♯ optima	Average ♯ extra	Offline error	Success rate
1	15030 ± 0	3 ± 0	0.7 ± 0.11	9.02E-03 ± 1.50E-03	100%
2	30090 ± 0	3 ± 0	0.2 ± 0.06	9.50E-03 ± 1.67E-03	100%
3	45150 ± 0	2.9 ± 0.04	0.2 ± 0.06	1.07E-02 ± 1.84E-03	96.67%
4	60210 ± 0	2.8 ± 0.06	0.3 ± 0.07	9.73E-03 ± 1.74E-03	93.33%
5	75270 ± 0	2 ± 0	1.1 ± 0.1	9.71E-03 ± 1.74E-03	100%
6	90330 ± 0	2 ± 0	1.1 ± 0.1	9.71E-03 ± 1.74E-03	100%
7	105390 ± 0	2.8 ± 0.06	0.3 ± 0.07	9.80E-03 ± 1.76E-03	93.33%
8	120450 ± 0	2.8 ± 0.06	0.3 ± 0.07	9.80E-03 ± 1.76E-03	93.33%

8.3.2 Discussion

From the results obtained when testing both the dynamic vector-based PSO and the dynamic species-based PSO using similar scenarios, it was concluded that the dynamic vector-based PSO perform better than the dynamic species-based PSO in cases where the positions of peaks in the landscape change. When only the heights but not the positions of the peaks change, the performance is the same.

8.4 Sensitivity of the dynamic vector-based PSO to changes in severity

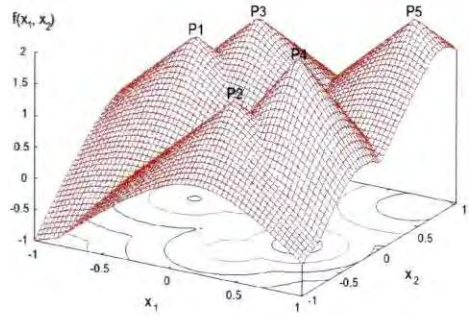
A changing problem landscape implies changes in the positions of optima and suboptima, the value of the objective function at those positions, or both. *Spatial severity*, referred to by the parameter ζ, quantifies the amount of change in the position of an optimum in problem space before the next optimization effort. In section 8.2, the performance of the dynamic vector-based

PSO was studied by setting up a number of changing environments. All of these experiments were conducted in a two-dimensional search space in the range $x_1, x_2 \in [-1.0, 1.0]$. Changes in the positions of the optima were effected by incrementing or decrementing the positions on the two axes, x_1 and x_2 by either 0, 0.1 or 0.2. In all scenarios the spatial severity, ζ, was never more than 0.28 (0.2 on both axes).

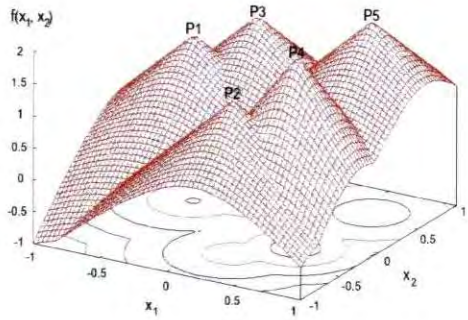
Experimental setup and results

This section investigates the influence of a range of severity values on the performance of the dynamic vector-based PSO. A function containing 5 peaks was created using the moving peaks benchmark as described by equation (7.3) [25]. One of the peaks moves diagonally through the landscape. Figure 8.5 illustrates the progression of the peak through a 5-peak landscape where the severity is 0.28. Peaks are numbered $P1$ to $P5$. Peak $P5$ progresses through the landscape while the other peaks remain stationary.

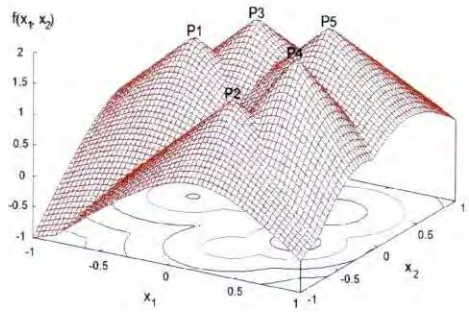
Eight experiments were conducted where the dynamic vector-based PSO tracks optima through the search space. For all experiments the starting position of peak $P5$ is at $[0.8, 0.8]$. Initially, the severity was set to 0.28 and increased by an increment of 0.1 on both axes for each experiment. For each value of ζ the algorithm was run 50 times and average results calculated. Stage 2 of the algorithm was disabled as no peaks appear or disappear. The number of steps required to move through the search space depends on the severity. Table 8.14 presents the experimental results. Figure 8.6 shows the average number of optima located, plotted against the severity.



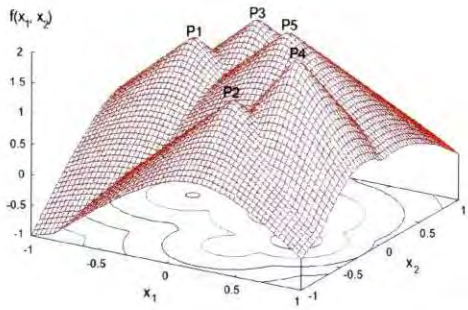
(a) Step 1



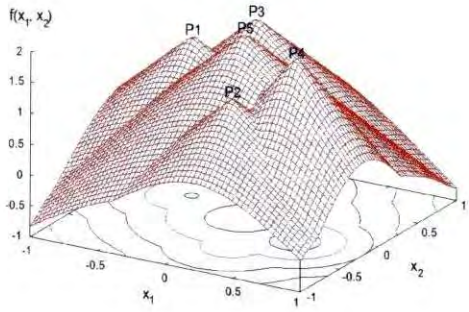
(b) Step 2



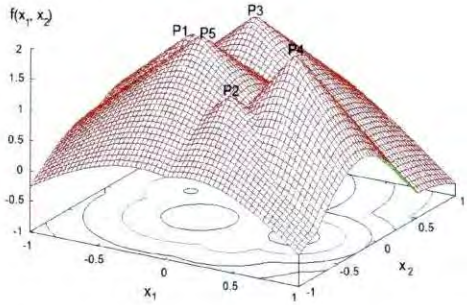
(c) Step 3



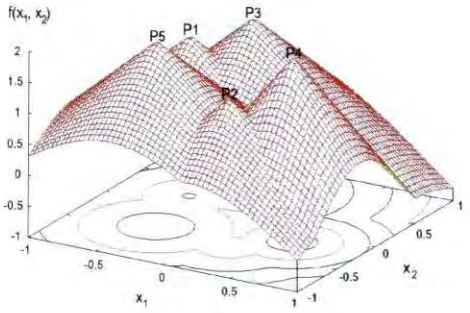
(d) Step 4



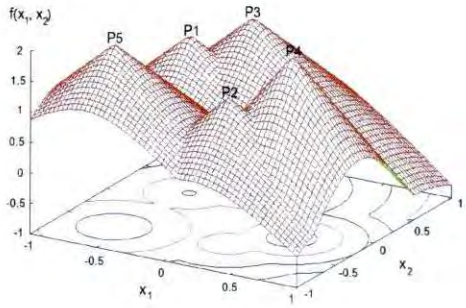
(e) Step 5



(f) Step 6



(g) Step 7



(h) Step 8

Figure 8.5: Movement of one peak through a landscape containing 5 peaks

Table 8.14: Performance of the dynamic vector-based PSO with increased severity

Experiment	x_1 increment	x_2 increment	♯ Steps	Severity	Average ♯ evaluations (step 1)	Average ♯ evaluations (step 2-8)	Average ♯ optima (step 1)	Average ♯ optima (step 2-8)	Success rate
1	0.2	0.2	9	0.28	26854 ± 384	10966 ± 6	5 ± 0	5 ± 0	100%
2	0.3	0.3	6	0.42	26033 ± 263	10854 ± 80	5 ± 0	5 ± 0	100%
3	0.4	0.4	5	0.57	27928 ± 397	10828 ± 70	5 ± 0	4.95 ± 0.03	87.5%
4	0.5	0.5	4	0.71	26415 ± 325	10238 ± 148	5 ± 0	4.27 ± 0.08	33.33%
5	0.6	0.6	3	0.85	26633 ± 295	10435 ± 134	5 ± 0	4.37 ± 0.09	25%
6	0.7	0.7	3	0.99	27521 ± 499	10190 ± 148	5 ± 0	4.35 ± 0.05	15%
7	0.8	0.8	3	1.13	26149 ± 414	10204 ± 149	5 ± 0	4.14 ± 0.05	15%
8	0.9	0.9	2	1.27	28621 ± 536	11048 ± 6	5 ±0	4.1 ± 0.04	10%

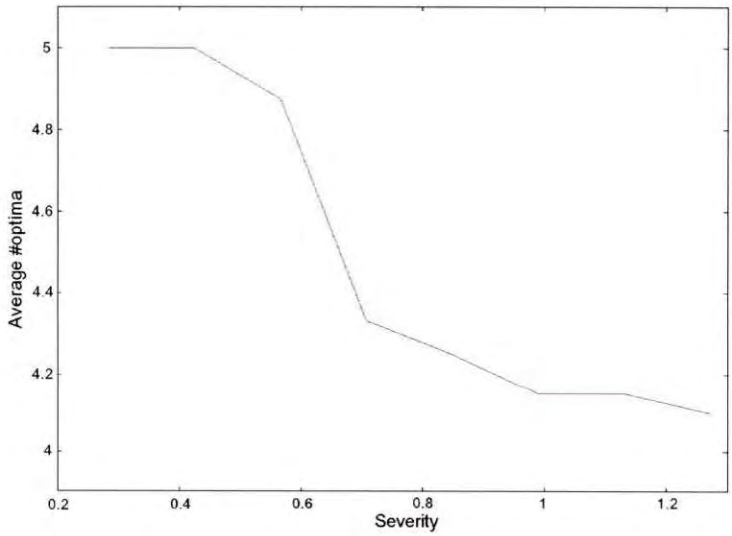


Figure 8.6: Average #optima versus severity for 5-peak function

Discussion

Intuitively, successful tracking is associated with a severity smaller than the niche radius of a peak, as the dynamic vector-based PSO retains previous optimal positions from where a small subswarm moves towards the new optimum. It stands to reason that, to track the new optimum successfully, the previous optimal position must still be on the slope leading to the new optimal position, that is, within the niche radius of the new optimum. However, niche radii cannot always be calculated accurately. In addition, some areas of the slope of a peak extends further than other areas, especially if the moving peak progresses through a landscape containing several other peaks. Results for the experiments conducted with the settings described in this section show a sharp decline in the success rate between severity values of 0.57 and 0.71 (distances between successive peaks). Careful scrutinizing of the function landscapes illustrated in Figure 8.5 reveals such severity values to be approximately equal to the upper bound of the niche radius. Therefore, a new optimum will still be tracked successfully, even if the position has moved to the boundary of the previous niche. To conclude, the dynamic vector-based PSO can be described as remarkably robust when tracking optima in a dynamic function landscape.

8.5 Conclusion

This chapter demonstrated that the parallel vector-based particle swarm optimizer can be modified to track multiple moving optima in a number of selected dynamic environments. Scenarios were chosen to represent different situations. Good results were reported when existing optima were tracked. The computational complexity required to track existing optima was much less than that of locating the optima in the first place. However, results are highly problem-dependent, and locating new peaks is not always successful. It is also computationally more expensive to search the entire problem space for these peaks, and in some cases re-optimization of the search space might have to be considered.

Comparing the dynamic vector-based PSO to the dynamic species-based PSO showed the dynamic vector-based PSO to be superior in all cases. To conclude the chapter, the sensitivity of the dynamic vector-based PSO to changes in severity was investigated. Results emphasized the dynamic vector-based PSO to be an efficient and robust algorithm.

Chapter 9

Conclusion

This chapter briefly summarizes the contributions and results of this thesis and discusses some directions for future research.

9.1 Summary

The principle focus of this thesis was directed towards the development of a PSO niching strategy that would address deficiencies and problems encountered by niching or speciation algorithms in general. The original single-solution PSO searches for one or more global optima, a task where traditional optimization methods may fail if the function landscape contains many sub-optimal solutions. One of the strengths of the single-solution PSO is the ability to direct the swarm away from a sub-optimal solution to a better solution. However, if a problem can be described by means of a multimodal function, a range of solutions is much more useful than a single solution. If the "best" solution is not viable for some reason, a "second-best" solution may suffice, especially since the value of the function at different optimal positions often differ very slightly or not at all. Niching algorithms yield such a range of solutions where each solution corresponds to an optimal position in the search space. Many niching algorithms use some strategy to divide a swarm into subswarms where each subswarm is expected to locate an optimal or suboptimal solution [13] [57]. When such subswarms are optimized, either sequentially or in parallel, multiple optima will only be located if the tendency of a subswarm to move away from a sub-optimal solution to a better solution is suppressed.

Chapters 2 and 3 presented an overview of optimization and the basic PSO paradigm.

Niching was defined in chapter 4 while a number of existing niching approaches was described and discussed. Problems with each of these approaches were identified so that particular attention could be given to these aspects in the design of a new strategy. Existing niching algorithms can be categorized as follows:

Algorithms that transform the landscape: To locate multiple optima, the problem landscape is transformed after an optimum has been located. The well-known "objective function stretching" approach of Parsopoulos and Vrahatis [71] [72] flattens or "stretches" the landscape where a minimum has been located so that the next minimum will now be at the global best position (minimization is assumed). Such an algorithm is per definition sequential. However, as described in chapter 4, false minima and misleading gradients can be introduced.

Algorithms using a niche radius: A popular approach used by niching algorithms is the division of a swarm into subswarms [13] [14] [16] [57]. Since the number, shapes, sizes and placing of niches in the function landscape are unknown, prior knowledge of the search space is often required to produce significant results. In addition to parameters like the problem range and the number of particles, some metric is needed to indicate the size and position of the region or niche in problem space where a subswarm resides. A number of algorithms use a niche radius, the distance between the current candidate solution and the boundary of the niche, to demarcate a niche. The species-based PSO sets a niche radius in advance [57], while NichePSO calculates the radius during execution of the algorithm [13] [14] [16]. An accurate niche radius, obtained by knowledge of the problem space or a number of test runs, yields good results if the shape and placing of the niches are relatively symmetrical. However, if the landscape is more convoluted with a variety of niche shapes and sizes, performance degrades. NichePSO does not need a pre-specified niche radius, but relies heavily on merging of niches, which requires fine-tuning of parameters as too many or too few niches may merge.

Algorithms relying on distances between particles: Attempts to avoid pre-specifying of a niche radius gave rise to the adaptive niching PSO (ANPSO) [5] and the fitness Euclidian-distance ratio-based PSO (FER-PSO) [58]. Good performances were reported, but both algorithms were computationally expensive due to the calculation of distances between particles (ANPSO) and ratios required by the FER-PSO.

The most imported objective of this study was to develop a new niching strategy that would

address the problems identified above. Although some parameters have to be set in advance to facilitate niching, it remained a challenge to develop a PSO niching algorithm requiring minimal prior knowledge of the function landscape. An additional challenge concerned the extension of the principles on which the original PSO algorithm rests to facilitate niching, resulting in an efficient and elegant solution. Such a strategy should also be robust enough to operate successfully in convoluted function landscapes where the shapes, sizes and placing of niches differ considerably.

The development of the vector-based PSO was described in detail in chapter 5. Vectors are used extensively in the original PSO. The position of an individual in the particle swarm is given by a position vector. Two other positions are also associated with each particle, namely the best position found so far by the particle itself (*pbest*) and the best position of the entire swarm (*gbest*). When updating particle positions, the original PSO uses vector addition to calculate new random positions. To facilitate niching, another vector operation, the vector dot product, is utilized. The technique rests on the assumption that a positive dot product indicates two vectors pointing in the same direction while a negative dot product is the result of two vectors pointing in opposite directions. The technique is explained in detail in chapter 5. Using this principle, niche boundaries can be calculated and the number of optima in an unknown search space be derived, each with its own niche radius. This technique addresses one of the major problems of niching algorithms, namely that of calculating niche radii for an unknown function landscape.

The vector-based algorithm progressed through three stages to produce an algorithm that locates niches sequentially and optimizes them in parallel. All three algorithms underwent extensive testing using one- and two-dimensional benchmark functions, presented in chapter 6. Care was taken to choose functions where the shapes and sizes of the niches differed and the positions of the optima were not distributed symmetrically throughout the search space. Thus the robustness of the algorithms could be assessed. Chapter 6 also presents a comparative study that was conducted to compare the vector-based PSO with the species-based PSO [57] and NichePSO [13] [14] [16]. The performance of these two algorithms degraded considerably for such functions. However, the vector-based PSO performed well throughout, even with multimodal functions that are perceived as difficult to optimize.

Since niches are not always symmetrical around the current best position (the *neighbourhood best*) in the niche, false niches are formed that have to be merged with genuine niches during optimization. Merging requires a problem-dependent merging parameter, the granularity, that has to be set in advance. The performance of three two-dimensional benchmark functions have

been evaluated for a range of granularity values. Results showed good performances where the granularity is less than the smallest interniche distance. Optima where the interniche distances are larger than the granularity, will still be located. Stated differently, in an unknown function landscape the vector-based PSO will locate all or most of the optima where the interniche distances are larger than the granularity.

Chapter 6 also incorporated a scalability study. A small subset of scalable multimodal functions were tested for a number of search spaces containing fewer than one hundred optima, using a range of swarm sizes. Results indicated that the increase in the optimal swarm size required to ensure that all optima in a demarcated search space are located, is smaller than expected.

The vector-based PSO was developed to locate multiple optima in a static environment. However, problems often require tracking of multiple optima that changes over time. Chapter 6 investigated the behaviour of the vector-based PSO technique in a changing environment. The vector-based PSO is adapted and extended to track multiple optima over time. A number of scenarios illustrating a variety of dynamic changes, was set up and tested. Results indicated that existing optima were tracked successfully, but locating new peaks was not always successful. It is also computationally more expensive to search the entire problem space for these peaks, and in some cases re-optimization of the search space might have to be considered. An investigation into the influence of severity on dynamic changes showed that the vector-based PSO could track optima successfully for severity values up to the upper bound of the current niche radius.

Objectives of this thesis were stated in the introduction. The extent to which these objectives have been met, is summarized as follows:

- The essential building blocks of the basic PSO paradigm was extended to facilitate niching by using another vector operation. By calculating the dot product of the two velocity vectors of a particle, niche boundaries could be determined.

- A new niching algorithm, the vector-based PSO, was developed and tested extensively. The algorithm proved to be efficient, elegant and robust. Results showed that the vector-based PSO located a very high percentage of multiple optimal solutions when tested with a variety of multimodal functions. Objective functions with asymmetrically shaped and placed functions also yielded good results, emphasizing the robustness of the algorithm. Finally, the algorithm can be described as elegant as it is small, simple, and uses vector operations, an integral part of the original PSO.

- Minimal prior knowledge of the function landscape is required as candidate solutions and niche radii are calculated. Therefore, differing niche sizes do not effect the success of the algorithm. A niching parameter, the granularity, is set beforehand. However, the granularity is not very sensitive and is an indication of the minimum distance between optima. Therefore, the user can decide how fine-grained the niching must be.
- Different niching approaches have been compared using a diverse problem set. The vector-based PSO outperformed the species-based PSO and NichePSO in cases where the function landscape was asymmetrical and convoluted.
- A scalability study of the vector-based PSO was conducted, showing that the algorithm scales well for a small set of multimodal functions in one to four dimensions.
- The vector-based PSO was extended to incorporate dynamic optimization problems.

9.2 Future work

This section briefly summarizes new directions for future research suggested by the work presented in this thesis:

- The vector-based PSO uses the dot product of two vectors to determine niche boundaries. The algorithms developed in this study utilize the dot product as positive or negative only. An investigation into the range of specific values of dot products may yield further information on the characteristics of the function landscape.
- The study of niching in dynamic environments using the vector-based PSO can be extended to include more benchmark functions, more scenarios and some practical implementations, e.g. training of neural network ensembles.
- Extend the dynamic VBPSO to automatically detect environment changes to remove its current dependence on knowledge of the change frequency.

Bibliography

[1] Al-kazemi, B., and Moham, C.K.: *Multi-phase Discrete Particle Swarm optimization* In: Proceedings of the International Workshop on Frontiers in Evolutionary Algorithms, pp. 622-625 (2002)

[2] Antonov, I.A. and Saleev, V.M.: *An economic method of computing LP_τ-sequences* Zh. vychisl. Mat. mat. **19** pp. 243-245. English translation: U.S.S.R. Comput. Math. Phys. 19 pp. 252-256. (1979)

[3] Bäck, T., Fogel, D.B., and Michalewicz, Z, eds. *Evolutionary Computation 2. 15.* Institute of Physics Publishers. (2000)

[4] Beasley, D., Bull, D.R., and Martin, R.R.: *A Sequential Niche Technique for Multimodal Function Optimization.* Evolutionary Computation, **1**(2): 101-125 (1993)

[5] Bird, S. and Li, X.: *Adaptively choosing niching parameters in a PSO* In: Proceeding of the Genetic and Evolutionary Computation Conference (GECCO 2006). pp. 3-9 Seattle, Washington, USA. (2006)

[6] Blackwell, T.: *Particle Swarm Optimization in Dynamic Environments* In: Evolutionary Computation in Dynamic and Uncertain Environments. Vol 51/2007 Springer Berlin/Heidelberg pp. 29-49 (2007)

[7] Blackwell, T.M., and Bentley, P.:*Don't Push Me! Collision-Avoiding Swarms* In: Proceedings of the IEEE Congress on Evolutionary Computation volume 2, PP. 1691-1696 (2002)

[8] Blackwell, T.M. and Bentley, P.J.: *Dynamic search with charged swarms* In: Genetic and Evolutionary Computation Conference. W.B.L. et al., Ed. Morgan Kaufmann, 19-26 (2000)

[9] Blackwell, T., Branke, J.: *Multi-Swarm Optimization in Dynamic Environments* In: Raidl, G.R. (ed.): Applications of Evolutionary Computation, Vol. 1281. Springer 489-500 (2004)

[10] Blackwell, T. and Branke, J.: *Multi-Swarms, Exclusion, and Anti-Convergence in Dynamic Environments* In: IEEE Transactions on Evolutionary Computation, **10**(4): 459-472 (August, 2006)

[11] Bratley, P. and Fox, B.L.:*Algorithm 659: Implementing Sobol's quasirandom sequence generator* ACM Trans. Math. Softw. **14**:88-100 (1988)

[12] Brits, R., Engelbrecht, A.P., and Van den Bergh, F.: *Solving systems of unconstrained equations using particle swarm optimizers* in: Proceedings of the IEEE Conference on Systems, Man and Cybernetics, pp. 102-107 (2002)

[13] Brits, R., Engelbrecht, A.P., and Van den Bergh, F.: *A niching particle swarm optimizer* In: Proceedings of the 4th Asia-Pacific Conference on Simulated Evolution and Learning. pp. 692-696 Singapore (November, 2002)

[14] Brits, R.: *Niching strategies for particle swarm optimization, MSc Thesis*, University of Pretoria (2002)

[15] Brits, R., Engelbrecht, A.P., and Van den Bergh, F.: *Scalability of NichePSO* In: Proceedings of the IEEE Swarm Intelligence Symposium, pp. 228-234, (Indianapolis, USA), (2003)

[16] Brits, R., Engelbrecht, A.P. and Van den Bergh, F.: *Locating multiple optima using particle swarm optimization* Applied Mathematics and Computation, **189**(2): 1859-1883 (June 2007)

[17] Carlisle, A., Dozier, G.: *Adapting Particle Swarm Optimization to Dynamic Environments* In: Proceedings of the International Conference on Artificial Intelligence. Las Vegas Nevada USA pp. 429-434 (2000)

[18] Carlisle, A., Dozier, G.: *Tracking Changing Extrema with Particle Swarm Optimizer* In: Proceedings of the Fifth Biennial World Automation Congress. pp. 265-270 (2002)

[19] Clerc, M.: *The Swarm & the Queen. Towards a Deterministic and Adaptive Particle Swarm Optimization* In: Proceedings of the Congress on Evolutionary Computation, pp. 1951-1957 (Washington DC, USA) IEEE Service Center, Piscataway, NJ (1999)

[20] Clerc, M. and Kennedy, J.: *The Particle Swarm - Explosion, Stability, and Convergence in a Multidimensional Complex Space*, IEEE Transactions on Evolutionary Computation, 6(1): pp. 58-73 (2002)

[21] Cramer, N.L.: *A representation for the Adaptive Generation of Sequential Programs* In: Proceedings of an International Conference on Genetic Algorithms and the Applications, Grefenstette, J.J. (ed.), Carnegie Mellon University (1985)

[22] Darwin, C.: *The Origin of Species*, P. F. Collier & Son (1909)

[23] Deb, K., and Goldberg, D.E.: *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, pp. 94-101, Atlanta, GA, (1990). The College of Engineering, Georgia Institute of Technology.

[24] De Jong, K.: *An analysis of the behavior of a class of genetic adaptive systems* Doctoral dissertation, University if Michigan, (1975)

[25] De Jong, K.A., Morrison, R.W.: *A Test Problem Generator for Non-Stationary Environment* In: Proceedings of the Congress on Evolutionary Computation. IEEE Press. pp. 2047-2053 (1999)

[26] Droste, S., Jansen, T. and Wegener, I.: *Perhaps not a free lunch but at least a free appetizer*

[27] Durham, W.: *Co-Evolution: Genes, Culture and Human Diversity*, Stanford University Press (1994)

[28] Eberhart, R.C., and Kennedy, J.: *A New Optimizer Using Particle Swarm Theory* Sixth International Symposium on Micro Machine and Human Science, pp.39-43 (Nagoya, Japan) IEEE Service Center (1995).

[29] Eberhart, R.C., and Shi, Y.: *Comparing Inertia Weights and Constriction Factors in Particle Swarm Optimization* In: Proceedings of the Congress on Evolutionary Computation, pp. 84-89 (San Diego, USA) IEEE Service Center, Piscataway, NJ (2000)

[30] Eberhart, R.C, and Shi, Y.: *Tracking and Optimizing Dynamic Systems with Particle Swarms* In: Proceedings of the 2001 Congress on Evolutionary Computation (CEC2001). pp. 94-100 (2001)

[31] Ellis, R., and Gulick, D.: *Calculus with Analytical Geometry* 5th ed., Saunders College Publishing (1994)

[32] Engelbrecht, A.P.: *Computational Intelligence - An Introduction*, John Wiley & Sons Ltd (2002)

[33] Engelbrecht, A.P.: *Fundamentals of Computational Swarm Intelligence*, John Wiley & Sons Ltd (2005)

[34] Engelbrecht, A.P., Masiye, B.S., and Pampará, G.: *Niching Ability of Basic Particle Swarm Algorithms* (2005)

[35] Fogel, D.B.: *Evolutionary Computation*, 2nd ed., IEEE Press, Piscataway, NJ (2000)

[36] Fogel, L.J., Owens, A.J. and Walsh, M.J.: *Artificial Intelligence Through Simulated Evolution* Wiley (1966)

[37] Forsyth, D.B.: *BEAGLE A Darwinian Approach to Pattern Recognition* Kybernetes, **10**:159-166 (1981)

[38] Glover, F.: *Tabu Search - Part I*, ORSA Journal on Computing **1**(3):190-206 (1989)

[39] Goldberg, D.E., Deb, K., and Horn, J.: *Massive Multimodality, Deception, and Genetic Algorithms* In: R. Männer and B. Manderick (Eds.): Parallel Problem Solving from Nature, 2, 37-46 North-Holland (1992)

[40] Goldberg, D.E., and Richardson, J.: *Genetic Algorithms with Sharing for Multimodal Function Optimization* In: Proceedings of the 2nd International Conference on Genetic Algorithms, pp. 41-49 (1997)

[41] Hendtlass, T.: *WoSP: A Multi-Optima Particle Swarm Algorithm*, In: Proceedings of the IEEE Congress on Evolutionary Computation. pp. 727-734 Edinburgh, UK. (2005)

[42] Hoffman, R., Minkin, V.I., and Carpenter, B.K.: *Ockham's razor and Chemistry.* International Journal for the Philosophy of Chemistry. 3. 3-28 (1997)

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

[44] Hoos, H.H.: *Stochastic Local Search - Methods, Models, Applications, PhD Thesis*, University of Darmstadt (1998).

[45] Hu, X. and Eberhart, R.C.: *Tracking Dynamic Systems with PSO: Where's the Cheese?*, In: Proceedings of the Workshop on Particle Swarm Optimization. pp. 80-81 (2001)

[46] Hu, X. and Eberhart, R.C.: *Adaptive Particle Swarm Optimization: Detection and Response to Dynamic Systems* In: Proceedings of the IEEE Congress on Evolutionary Computation. Vol. 2 pp. 1666-1670, (May, 2002)

[47] Joe, S. and Kuo, F.Y.: *Remark on Algorithm 659: Implementing Sobol's quasirandom sequence generator* ACM Trans. Math. Softw. **29**:49-57 (2003)

[48] Kennedy, J., and Eberhart, R.C.: *Particle Swarm Optimization* In: Proceedings of the IEEE International Conference on Neural Networks (Perth, Australia), IEEE Service Center, Piscataway, NJ, IV: 1942-1948 (1995).

[49] Kennedy, J.: *The Particle Swarm: Social Adaptation of Knowledge* In: Proceedings of the International Conference on Evolutionary Computation, pp. 303-308, Indianapolis, IN, USA (1997)

[50] Kennedy, J.: *Why Does it Need Velocity?* In: Proceedings of the IEEE Swarm Intelligence Symposium, Pasadena CA (2005)

[51] Kennedy, J.: *Small Worlds and Mega-Minds: Effects of Neighborhood Topology on Particle Swarm Performance*, In: Proceedings of the Congress on Evolutionary Computation, pp. 1931-1938 (Washington DC, USA) IEEE Service Center, Piscataway, NJ (1999)

[52] Kennedy, J.: *Stereotyping: Improving Particle Swarm Performance with Cluster Analysis* In: Proceedings of the IEEE Congress on Evolutionary Compuation vol. 2 pp. 1507-1512 (2000)

[53] Kennedy, J., and Mendes, R.: *Population Structure and Particle Swarm Performance* In: Proceedings of the IEEE Congress on Evolutionary Computation, pp. 1671-1676 IEEE Press (2002)

[54] Kirkpatrick, S., Gelatt, C.D., and Vecchi, M.P.: *Optimization by simulated annealing*, Science, **220**4598: 671-680 (1983).

[55] Koza, J.R.: *Genetic Programming: On the programming of Computers by means of Natural Selection* MIT Press (1992)

[56] Li, J.P., Balazs, M.E., Parks, G.T., and Clarkson, P.J.: *A species conserving genetic algorithm for multi-modal function optimization* In: Evolutionary Computation, **10**(3):207-234 (2002)

[57] Li, X.: *Adaptively Choosing Neighbourhood Bests using Species in a Particle Swarm Optimizer for Multimodal Function Optimization.* In: Proceedings of the Genetic and Evolutionary Computation Conference (GECCO 2004). pp. 105-116 (2004)

[58] Li, X.: *A multimodal particle swarm optimzer based on fitness Euclidian-distance ratio* In: Proceeding of the Genetic and Evolutionary Computation Conference (GECCO 2007). pp. 78-85 London, England, UK. (2007)

[59] Li, X., Branke, J. and Blackwell, T.: *Particle Swarm and Adaptation in a Dynamic Environment* In: GECCO06, Seattle, Washington, USA (2006)

[60] Løvbjerg, M., Rasmussen, T.K., Krink,T.: *Hybrid particle swarm optimizer with subpopulations and breeding* In: Proceedings of the Genetic and Evolutionary Computation Conference, vol 1, San Francisco USA pp. 469-476 (July 2001)

[61] Mahfoud,S.: *Crowding and preselection revisited* In: Reinhard Männer and Bernard Manderick, editors, Parallel Problem Solving from Nature 2, pp. 27-37, (1992)

[62] Malan, K., and Engelbrecht, A.P.:*Algorithm Comparisons and the Significance of Population Size* Proceedings of the 2008 IEEE World Congress on Computational Intelligence (WCCI 2008). Hong Kong. (2008) To be published.

[63] Marais, E.N.: *The Soul of the White Ant*, (1937).

[64] Mathews, J.H.: *Numerical Methods for Mathematics, Science and Engineering*, Prentice Hall, Inc. 2nd ed. (1992).

[65] Mendes, R., Kennedy, J., and Neves, J.: *Watch Thy Neighbour or How the Swarm can Learn from its Environment* In: Proceedings of the IEEE Swarm Intelligence Symposium, pp. 88-94 (2003)

[66] Millonas, M.M.: *Swarms, Phase Transitions, and Collective Intelligence* In: C.G.Langton, Ed., Artificial Life III. Addison Wesley, Reading, MA. (1994)

[67] Nelder, J.A. and Mead, R.: *A simplex method for function minimization* In: The Computer Journal, volume 7, pp. 308-313 (1965)

[68] Ozcan, E., and Mohan, C.: *Analysis of a simple particle swarm optimization system* In: Intelligent Engineering Systems Through Artificial Neural Networks (ANNIE'98), Vol. 8 pp. 253-258 (1998)

[69] Ozcan, E., and Mohan, C.: *Particle Swarm Optimization: Surfing the Waves* In: Proceedings of the International Congress on Evolutionary Computation, pp.1939-1944 (Washington USA) (1999)

[70] Parrott, D., Li, X.: *A Particle Swarm Model for Tracking Multiple Peaks in a Dynamic Environment using Speciation.* In: Proceedings on the 2004 Congress of Evolutionary Computation (CEC2004) pp. 98-103 (2004)

[71] Parsopoulos, K.E., Plagianakos, V.P, Magoulas, G.D. and Vrahatis, M.N.: *Stretching Techniques for Obtaining Global Minimizers through Particle Swarm Optimization.* In: Proceedings of the Particle Swarm Optimization Workshop. Indianapolis USA pp. 22-29 (2001)

[72] Parsopoulos, K.E. and Vrahatis, M.N.: *Modification of the Particle Swarm Optimizer for Locating all the Global Minima.* In: Kurkova, V., Steele, N.C., Neruda, R. and Karny, M. (eds.): Artificial Neural Networks and Genetic Algorithms, 324-327 Springer (2001)

[73] Peer, E.S., Van den Bergh, F., and Engelbrecht, A.P.: *Using Neighbourhoods with the Guaranteed Convergence PSO* In: Proceedings of the IEEE Swarm Intelligence Symposium, pp. 235-242 IEEE Press (2003)

[74] Pétrowski, A.: *A clearing procedure as a niching method for genetic algorithms* In: Proceedings of the Third IEEE International Conference on Evolutionary Computation (ICEC'96), Piscataway, NJ, IEEE Press, pp. 798-803 (1996)

[75] Potts, W.K.: *The chorus-line hypothesis of coordination in avian flocks*, Nature **24**: 344-345 (1984)

[76] Press, W.H., Teukolsky, S.A., Vetterling, W.T., and Flannery, B.P.: *Numerical Recipes in C: the Art of Scientific Computing* Cambridge University Press, second edition (1992)

[77] Fowler, F.G., and Fowler, H.W.: *The Pocket Oxford Dictionary of Current English* 5th edition Oxford University Press, London (1969)

[78] Price, K.V.: *An Introduction to Differential Evolution*, In: Corne, D., Dorigo, M., and Glover, F. (eds.) New Ideas in Optimization, McGraw-Hill, London (1999)

[79] Rechenberg, I.: *Evolutionsstrategie: Optimierung Technischer Systeme nach Prinzipien der Biologischer Evolution.* Framman-Holzboog Verlag, Stuttgart (1973)

[80] Reynolds, C.W.: *Flocks, herds and schools: a distributed behavioural model*, Computer Graphics **21**(4): 25-34 (1987)

[81] Reynolds, R.G.: *Cultural algorithms* In: Corne, D., Dorigo, M., and Glover, F., editors, New Ideas in Optimization, p. 367, McGraw-Hill (1999)

[82] Schoeman, I.L., and Engelbrecht, A.P.: *Using Vector Operations to Identify Niches for Particle Swarm Optimization* Proceedings of the IEEE Conference on Cybernetics and Intelligent Systems. pp. 361-366 Singapore (2004)

[83] Schoeman, I.L., and Engelbrecht, A.P.: *A Parallel Vector-Based Particle Swarm Optimizer* Proceedings of the International Conference on Artificial Neural Networks and Genetic Algorithms. (ICANNGA2005) Coimbra Portugal, pp. 268-271 (2005)

[84] Schoeman, I.L., and Engelbrecht, A.P.: *Containing Particles inside Niches when Optimizing Multimodal Functions* Proceedings of SAICSIT2005. White River, South Africa pp. 78-85 (2005)

[85] Schoeman, I.L., and Engelbrecht, A.P.: *Niching for Dynamic Environments Using Particle Swarm Optimization* In: Proceedings of the Sixth International Conference on Simulated Evolution and Learning (SEAL'06). pp. Hefei, China, (October 2006)

[86] Schwefel, H-P.: *Evolutionsstrategie und Numerische Optimierung.* PhD Thesis, Technical University Berlin (1975)

[87] Sheskin, D.J.:
emphHandbook of Parametric and Nonparametric Statistical Procedures, Chapman & Hall/CRC (2007)

[88] Shi, Y., and Eberhart, R.C.: *A Modified Particle Swarm Optimizer* In: IEEE International Conference of Evolutionary Computation (Anchorage, Alaska) pp. 69-73 (1998)

[89] Shi, Y. and Eberhart, R.C.: *Parameter Selection in Particle Swarm Optimization* In: Evolutionary Programming VII: Proceedings of EP 98, pp. 591-600 (1998)

[90] Singh, G., and Deb, K.: *Comparison of Multi-Modal Optimization Algorithms Based on Evolutionary Algorithms* GECCO'06, (Seattle, USA) pp. 1305-1312 (2006)

[91] Smith, S.F.: *A learning System based on Genetic Adaptive Algorithms.* PhD Thesis, University of Pittsburgh (1980)

[92] Smith, R.E., Forrest, S., and Perelson, A.S.: *Searching for diverse, cooperative populations with genetic algorithms* TCGA Report No. 92002. The University of Alabama, Department of Engineering Mechanics. (1992)

[93] Snyman, J.A.: *Practical Mathematical Optimization*, Springer (2005) pp. 1-16.

[94] Storn, R., and Price, K.: *Differential Evolution - a Simple and Efficient Heuristic for Global Optimization over Continuous Spaces*, In: Journal of Global Optimization, **11**:341-359 (1997)

[95] Suganthan, P.N.: *Particle Swarm Optimizer with Neighbourhood Operator* In: Proceedings of the Congress on Evolutionary Computation, pp. 1958-1961 (Washington DC, USA) IEEE Service center, Piscataway NJ (1999)

[96] Thodberg, H.H.: *Improving Generalization of Neural Networks through Pruning.* International Journal of Neural Systems. 1(4), 317-326 (1991)

[97] Trelea, I.C.: *The Particle Swarm Optimization Algorithm: Convergence and Parameter Selection* Information Processing Letters, **85**(6):317-325 (2003)

[98] Van den Bergh, F.: *An Analysis of Particle Swarm Optimizers* PhD Thesis, University of Pretoria (2002).

[99] Van den Bergh, F. and Engelbrecht, A.P.: *A New Locally Convergent Particle Swarm Optimiser* In: Proceedings of the IEEE International Conference on Systems, Man and Cybernetics, (Hammamet, Tunisia), October 2002.

[100] Van den Bergh, F. and Engelbrecht, A.P.: *A Study of Particle Swarm Optimization Particle Trajectories*, Information Sciences **176**:937-971 (2006)

[101] Van den Bergh, F. and Engelbrecht, A.P.: *A Cooperative Approach to Particle Swarm Optimization* IEEE Transactions on Evolutionary Computation **8**(3):225-239 (2004)

[102] Veeramachaneni, K., Peram, T., Mohan, C., and Osadciw, L.A.: *Optimization Using Particle Swarms with Near Neighbour Interactions* In: Proceedings of the Genetic and Evolutionary Computation Conference, In: Lecture Notes in Computer Science **2723**:110-121. Springer-Verlag (2003)

[103] Wolpert, D.H., and Macready, W.G.: *No Free Lunch Theorems for Optimization*, IEEE Transactions on Evolutionary Computation **1**(1), pp. 67-82 (1997)

Appendix A

Test Results

This appendix presents empirical results of tests that were run to determine specific settings used by the family of vector-based PSO algorithms. Section A.1 presents results of empirical tests to determine minimum sizes of small subswarms to prevent such swarms from stagnating. Section A.2 investigates the influence of interval sizes between calls to the merging procedure. This procedure is called several times while subswarms are updated in parallel in order to merge subswarms attempting to optimize the same optimal or suboptimal solution. Section A.3 presents results of empirical tests to find the minimum size of a subswarm capable of tracking a moving optimum. If the function landscape changes, positions of previous optima are retained and small subswarms are created around such optima. Fewer function evaluations are required resulting in a reduction in computational complexity.

A.1 Minimum subswarm sizes

To establish niche boundaries, the vector-based PSO calculates niche radii as described in section 5.4. A number of extra or false niches are formed adjacent to true niches. Some niches, especially the false niches, may contain subswarms consisting of only one or two particles. As explained in section 5.4, such subswarms may stagnate. To address this problem, these subswarms are extended to contain at least three particles.

A.1.1 Experimental procedure

A selection of two-dimensional functions with differing characteristics, namely the Himmelblau, Ursem F3 and Ackley functions, were used to investigate minimum subswarm sizes. Descriptions and illustrations of the functions were provided in section 6.2.3. The algorithms were run with minimal subswarm sizes of one, two, and three particles. The number of solutions that indicated optima was recorded as well as the number of extra or false solutions that did not indicate such positions. Each result is the average of 50 runs of the relevant algorithm. Results are presented for the parallel vector-based PSO as well as the enhanced parallel vector-based PSO.

Table A.1: Convergence of small subswarms using the parallel vector-based PSO

Function	Results					
	Subswarm size of 1		Subswarm size of 2		Subswarm size of 3	
	Success rate	Average # false solutions	Success rate	Average # false solutions	Success rate	Average # false solutions
Himmelblau	98%	5.6	99.5%	0.1	100%	0
Ursem F3	99%	5.7	100%	0.12	100%	0
Ackley (2-dim)	95.8%	2.7	98.7%	0.08	98.9%	0

A.1.2 Results and discussion

When subswarms stagnate, results show solutions that do not indicate optimal positions in the search space, since such subswarms do not converge on optimal positions. No such false solutions will result if subswarms keep moving and eventually merge to indicate true optimal positions. Thus, the number of such false solutions gives an indication of the ability of subswarms to keep moving.

Tables A.1 and A.2 show the average number of extra or false solutions for the parallel vector-based PSO and the enhanced parallel vector-based PSO. For each function and subswarm size combination that was tested, the success rate of the run (total number of solutions as a

Table A.2: Convergence of small subswarms using the enhanced parallel vector-based PSO

Function	Results					
	Subswarm size of 1		Subswarm size of 2		Subswarm size of 3	
	Success rate	Average # false solutions	Success rate	Average # false solutions	Success rate	Average # false solutions
Himmelblau	100%	0	100%	0	100%	0
Ursem F3	100%	0.8	100%	0.16	100%	0
Ackley	99.6%	0	99.6%	0	99%	0

percentage of the total number of possible solutions) and the average number of extra or false solutions were recorded. As result of the stochastic character of the VBPSO, small differences in success rates occurred. For each function, the average number of false solutions decreased until no false solutions were recorded when a subswarm contained a minimum of three particles. This lead to the conclusion that the subswarm sizes of at least three particles are sufficient to obtain good results. Note that the enhanced parallel vector-based PSO yielded fewer false solutions. This effect can be ascribed to the procedure where the particle position is updated (refer to section 5.3), which is the only difference between the two algorithms. The enhanced version tests each new position of a particle to ensure that the particle does not leave the niche. Thus, a subswarm is prevented from being diverted to, and absorbed by, a neighbouring subswarm, an effect causing a lower success rate. In addition, small subswarms are prevented from becoming still smaller, a situation where stagnation can occur more often.

The small selection of results presented here does not purport to represent an exhaustive analysis of the significance of choosing a minimum swarm size. False solutions may still be encountered, even with larger minimum swarm sizes. On the other hand, the improved performance of the enhanced version indicates that smaller minimum swarm sizes may suffice. However, keeping in mind that the introduction of another tunable parameter to indicate minimum swarm size is not an option, and too large swarm sizes increase computational complexity, the minimum swarm size was set to three for all subsequent experiments.

A.2 Merging intervals

The parallel vector-based PSO algorithms update subswarms in parallel, thereby differing from the earlier sequential version. During each iteration of the algorithm, all particles in all subswarms are updated. When the vector-based PSO initially identifies niches, a number of addtional or false niches are formed (see section 5.5). Subswarms in these niches are expected to converge on optimal or suboptimal solutions of adjacent true niches (containing an optimal solution), giving duplicate solutions. The parallel VBPSO merges these subswarms with subswarms in true niches to exclude duplicate solutions. If the merging procedure is called during the run, false niches are gradually absorbed by true niches, forming larger subswarms which converge more effectively. If the merging procedure is called only once at the end of the run, all subswarms have to be merged at once, which might be less effective. A selection of functions have been tested to observe the effect of different merging intervals. Descriptions and illustrations of the functions were provided in section 6.2. Only the enhanced parallel VBPSO was tested as the parallel and enhanced parallel algorithms use the same merging procedure. Averages of 50 runs have been reported for each setting. For each run the updating procedure is iterated 500 times, interspersed with a number of calls to the merging procedure.

Results presented in Table A.3 show that a small number of subswarms do not merge when the merging procedure is called once at the end of a run of 500 iterations. One extra call to the merging procedure (with differing intervals) improves the situation but some niches may still not merge. If the merging procedure is called more often, all niches merge. No significant difference in the success rates of the different functions was found, indicating that the algorithm is not sensitive to the exact size of the merging interval. In addition, the exact size of the interval does not exert any influence on the success of the merging process, provided that the merging procedure is called more than once during optimization and again at the end of the run. The calls are best spread evenly over the run. The vector-based PSO algorithms described in algorithms 10 and 11 calls the merging procedure at 10 equal intervals during the run. These intervals were chosen in order to trace the merging process and present it graphically. These results were presented in chapter 6.

Considering the above description, the size of the merging interval can rather be seen as a heuristic than a tunable parameter. Therefore, setting the size of the merging interval cannot be used as an argument that the principle of parsimony is violated.

Table A.3: Effect of merging intervals

Function	Results									
	Merge at end		Merge at interval 250		Merge at intervals 400, 100		Merge at 5 intervals of 100		Merge at 10 intervals of 50	
	Success rate	Average # extra solutions	Success rate	Average # extra solutions	Success rate	Average # extra solutions	Success rate	Average # extra solutions	Success rate	Average # extra solutions
Himmelblau (4 optima)	100%	0.2	100%	0	100%	0	100%	**0**	100%	0
Ursem F3 (4 optima)	100%	0.96	100%	0	100%	0	100%	**0**	100%	0
Ackley 2-dim (9 optima)	100%	1.44	99.8%	0.02	99.3%	0.06	99.8%	**0**	100%	0

A.3 Minimum swarm size for tracking optima

Section 7.6 presents an algorithm to track multiple optima in a dynamic environment. The initial stage locates optima using the VBPSO described in chapters 5 and 6. After each environment change, optima have to be tracked and new optimal positions located. To reduce computational complexity, only the optimal positions of the previous stage are retained, and small subswarms are created around those positions. This section presents tests designed to find a minimum swarm size capable of tracking optima in a dynamic environment.

A.3.1 Experimental procedure

Scenario 1 described in section 8.2 was used to observe the effect of different minimum swarm sizes on tracking capability. Three optima were tracked over six steps. For the sake of clarity, the positions of the peaks are listed in Table A.4. Experiments were conducted where the size of the small swarm created to track each optimum, were set to one (one particle at the position of each previous optimum), two, three and four. The average offline errors between the optimal positions found by the small swarms and the true optimal positions were calculated for each setting. These errors were compared to the average offline error for the initial stage of the algorithm (using a larger swarm). Errors of the same order of magnitude indicated a tracking ability of the small subswarms similar to that of the initial swarm.

Table A.4: Positions of 3 optima over 6 steps

Step	Peak 1				Peak 2				Peak 3			
x_1	x_2	H	R	x_1	x_2	H	R	x_1	x_2	H	R	
1	-0.6	-0.8	1	2	-0.5	0.3	0.6	2	0.5	0	0.8	2
2	-0.4	-0.8	1	2	-0.3	0.4	0.6	2	0.4	0	0.8	2
3	-0.2	-0.8	1	2	-0.1	0.5	0.6	2	0.3	0	0.8	2
4	0	-0.8	1	2	0.1	0.6	0.6	2	0.2	0	0.8	2
5	0.2	-0.8	1	2	0.3	0.7	0.6	2	0.1	0	0.8	2
6	0.4	-0.8	1	2	0.5	0.8	0.6	2	0	0	0.8	2

A.3.2 Results and discussion

Results of experiments conducted to test the tracking ability of small subswarms are presented in Table A.5. Averages of 50 experiments are listed. No significant difference was observed between the average offline errors of the initial swarm (consisting of 30 particles) for the different experiments. However, for stages two to six of the algorithm, the offline error decreased with an increase in the sizes of the small subswarms. For a subswarm of three particles, the offline error was marginally larger than that of the initial swarm. For a subswarm of four particles, the offline error was marginally smaller than that of the initial swarm. Therefore, given that the severity did not exceed the niche radius (refer to section 8.4), a subswarm size of three or four particles was sufficient to track moving optima in a dynamic environment. For the dynamic VBPSO, it was decided to create subswarms at previous optimal positions where each subswarm consisted of a particle at one of those positions as well as three additional particles.

Table A.5: Tracking ability of small subswarms

Subswarm size (particles)	Success rate	Offline error	
		Initial swarm	Small subswarms
1	100%	5.17E-18 ± 1.77E-18	3.48E-03 ± 1.33E-03
2	100%	2.40E-17 ± 1.17E-17	5.34E-06 ± 1.90E-06
3	100%	7.57E-18 ± 2.61E-18	9.38E-17 ± 4.67E-17
4	100%	2.62E-17 ± 1.98E-17	1.09E-17 ± 1.62E-18

Appendix B

Derived publications

This appendix lists all papers that have been published or are currently under review, that led to, or are derived from the work presented in this thesis.

Schoeman, I.L., and Engelbrecht, A.P.: *Using vector operations to identify niches for particle swarm optimization.* In: Proceedings of the IEEE Conference on Cybernetics and Intelligent Systems. pp. 361-366 Singapore (2004)

Schoeman, I.L., and Engelbrecht, A.P.: *A parallel vector-based particle swarm optimizer.* In: Proceedings of the International Conference on Artificial Neural Networks and Genetic Algorithms. (ICANNGA2005) Coimbra Portugal, pp. 268-271 (2005)

Schoeman, I.L., and Engelbrecht, A.P.: *Containing particles inside niches when optimizing multimodal functions.* In: Proceedings of SAICSIT2005. White River, South Africa pp. 78-85 (2005)

Schoeman, I.L., and Engelbrecht, A.P.: *Niching for dynamic environments using particle swarm optimization.* In: Proceedings of the Sixth International Conference on Simulated Evolution and Learning (SEAL'06). Hefei, China, (October 2006)

Schoeman, I.L., and Engelbrecht, A.P.: *A novel particle swarm niching technique based on extensive vector operations.* Natural Comuputing, Springer, 1567-7818 (print) 1572-9796 (online) (December 23, 2009)

Schoeman, I.L., and Engelbrecht, A.P.: *Scalability of the vector-based PSO.* In: Proceedings of the Congress of Evolutionary Computation (CEC2009) Trondheim, Norway, (May 2009)

Isabella Schoeman and Andries Engelbrecht: *Effect of Particle Initialization on the Performance of Particle Swarm Niching Algorithms.* Accepted as an extended abstract in ANTS 2010

Schoeman, I.L., and Engelbrecht, A.P.: *Tracking Multiple Optima in Dynamic Environments using Particle Swarm Optimization.* Under review.