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

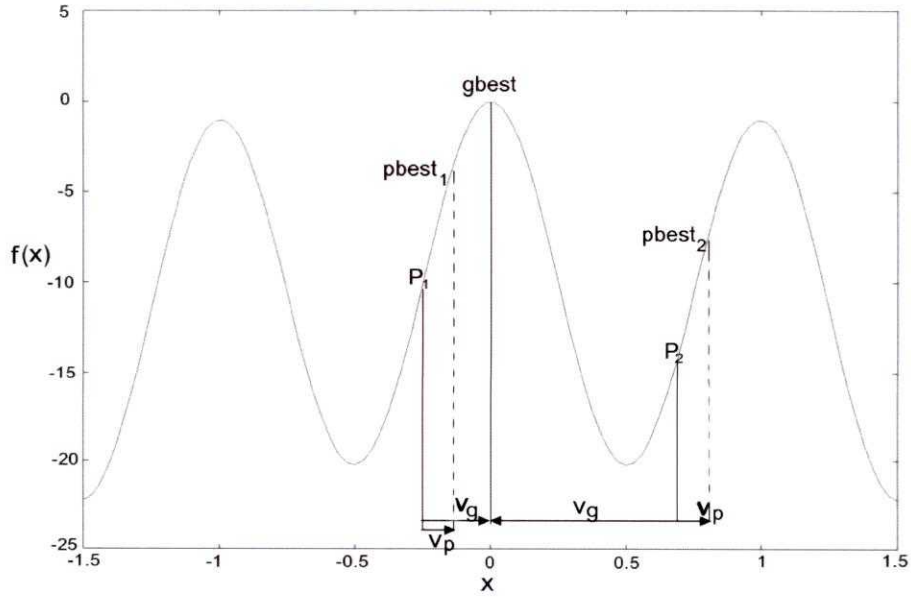

Figure 5.1: The inverted one-dimensional Rastrigin function, showing particles with associated vectors

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					

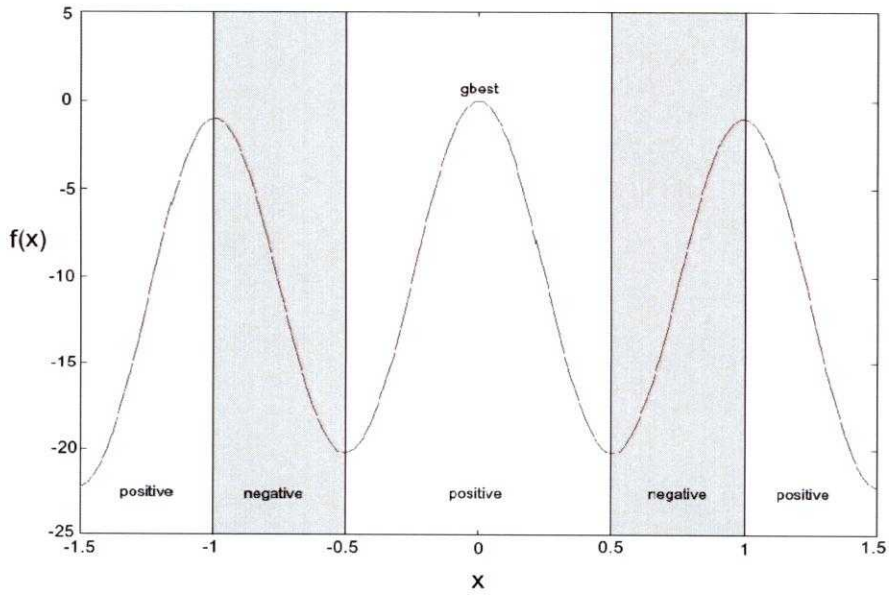

Figure 5.2: Regions of positive and negative dot products for the inverted one-dimensional Rastrigin function

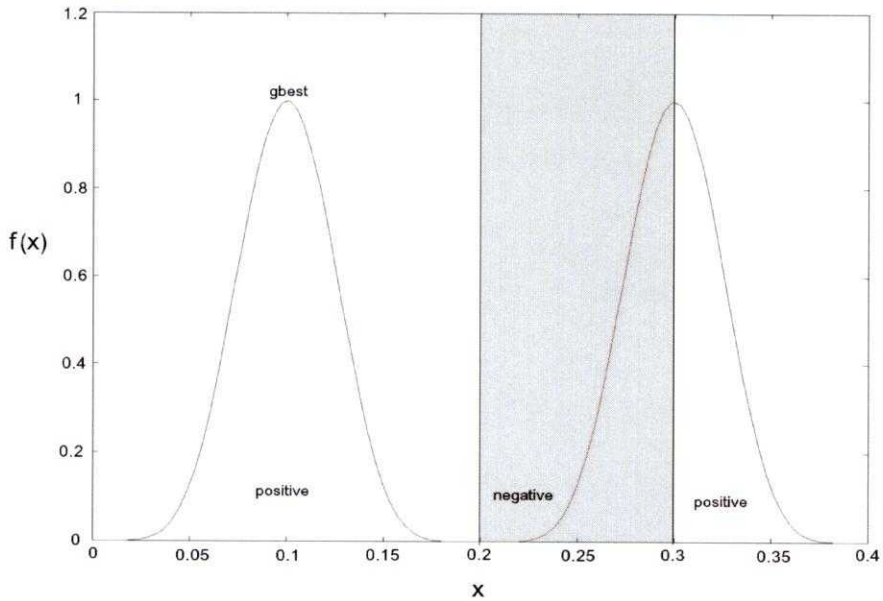

Figure 5.3: Regions of positive and negative dot products for equation (5.3)

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

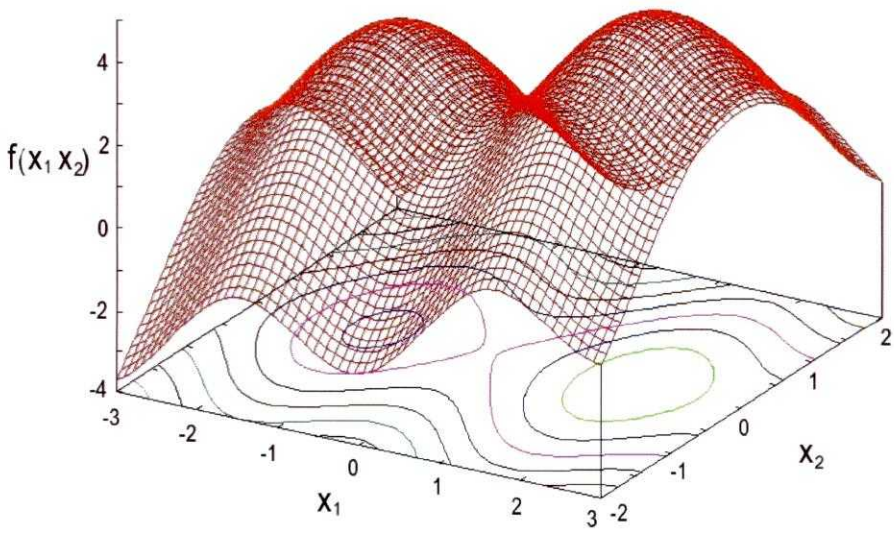

Figure 5.4: The two-dimensional Ursem F1 function

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

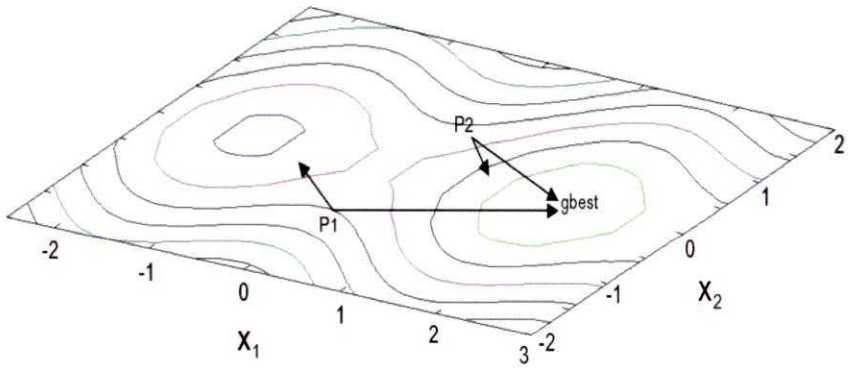

Figure 5.5: Contour map of the two-dimensional Ursem F1 function

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.