Niching strategies for particle swarm optimization

by

Riaan Brits

Submitted in partial fulfillment of the requirements for the degree

Master of Science

in the Faculty of Natural & Agricultural Science

University of Pretoria

Pretoria

November 2002

Niching strategies for particle swarm optimization

by

Riaan Brits

Abstract

Evolutionary algorithms and swarm intelligence techniques have been shown to successfully solve optimization problems where the goal is to find a single optimal solution. In multimodal domains where the goal is to locate multiple solutions in a single search space, these techniques fail. Niching algorithms extend existing global optimization algorithms to locate and maintain multiple solutions concurrently. This thesis develops strategies that utilize the unique characteristics of the particle swarm optimization algorithm to perform niching. Shrinking topological neighborhoods and optimization with multiple subswarms are used to identify and stably maintain niches. Solving systems of equations and multimodal functions are used to demonstrate the effectiveness of the new algorithms.

Supervisor: Prof. A. P. Engelbrecht
Co-supervisor: Dr. F. van den Bergh
Department of Computer Science
Degree: Master of Science

Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Problem Statement and Overview

Particle Swarm Optimization (PSO) is a recent, novel optimization algorithm, inspired by simulations of the social behavior of flocks of birds [49]. Numerous studies have shown the PSO to be a very effective optimization algorithm, outperforming the traditional genetic algorithm (GA). The algorithm is simple to implement, and does not depend on problem specific recombination and selection operators to achieve maximum effectiveness.

Two distinct features of the PSO algorithm's swarming behavior makes it an effective optimizer:

- each particle in a swarm retains a memory of the best solution that it has found, and
- all particles constantly move towards the best solution found by the entire swarm.

This approach allows the algorithm to quickly traverse a search space and rapidly converge. The fact that each particle in a swarm represents an *independent* candidate solution, makes the algorithm largely insensitive to multimodal optimization domains where other numerical techniques may find suboptimal solutions.

Niching techniques extend the search capabilities of population based optimization techniques to locate multiple solutions in a single search space. Numerous optimization

problems with multiple, equally acceptable solutions, such as solving systems of equations, exist. Niching techniques have been investigated in the genetic algorithm (GA) research field, with very little particle swarm optimization (PSO) based research undertaken on this subject. This thesis investigates the possible application of GA-based niching techniques to the PSO algorithm, and introduces two new unique PSO-based niching techniques that utilize the characteristics of the PSO algorithm.

1.2 Objectives

The main objective of this thesis is to study niching in the context of particle swarm optimization. In reaching this goal, the following subobjectives are identified:

- To provide an overview of existing GA–based and PSO niching techniques.
- To present and analyze the novel *nbest* and NichePSO algorithms.
- To compare the performance of the above algorithms to well-known GA niching algorithms.
- To show that the unique swarming nature of the PSO algorithm is a promising tool when investigating new niching algorithms.

1.3 Contribution

The main contributions of this thesis are:

- The conclusion that GA niching algorithms may not necessarily directly apply to the PSO algorithm.
- The introduction of a PSO niching technique, developed specifically to solve systems of linear and non-linear equations.
- The development of a PSO niching algorithm that use multiple swarms to maintain several solutions concurrently in a single search space.

1.4 Thesis Outline

Chapter 2 presents an introduction to the theory of optimization. A general evolutionary computing framework is then given, with specific focus on genetic algorithms. A complete presentation of the PSO algorithm is then given, including a number of optimizations and extensions.

Chapter 3 reviews the evolutionary motivation behind niching techniques. A number of GA niching algorithms are then discussed, followed by an existing PSO-based niching algorithm. The application of GA niching techniques to the PSO are considered.

Chapters 4 and 5 respectively present the *nbest* and NichePSO algorithms. The characteristics of each of these algorithms are analyzed and considered, and experimental results are presented to illustrate their effectiveness. Chapters 6 presents an empirical comparison of both these techniques to existing GA niching techniques.

Chapter 7 presents a summary of the research done in this thesis. A number of future research topics are identified.

An appendix presents a list of publications that followed from the presented work.

Chapter 2

Background and Literature Review

This chapter presents an overview of classical optimization, evolutionary computation and recent developments in particle swarm optimization.

2.1 Introduction

Optimization is a paradigm present in nearly every aspect of life. It describes the drive to constantly find improved ways of solving old and new problems. In the context of technological development and innovation, optimization describes the search for techniques that make better use of available resources to solve problems. Scientific and engineering applications regularly require algorithms to locate and fine-tune optimal solutions. Solving systems of equations, a comparatively simple task when only a few equations and unknowns are involved, becomes notably more complex as a problem's dimensionality grows.

Real-world optimization applications can be realized as function optimization problems. Function optimization algorithms make assumptions about what a problem looks like and are not universally applicable [11]. It is therefore justified to investigate alternative methods to solve these problems. More importantly, research should focus on identifying *efficient*, *robust* and *generally applicable* algorithms.

This chapter presents an overview of the theory behind optimization. Section 2.2 defines optimization problems and establishes a common notation used throughout this

thesis. Evolutionary computation is discussed in section 2.3, with an overview of genetic algorithms in section 2.4. Swarm intelligence is discussed in section 2.5, focusing in particular on the particle swarm optimization algorithm. Principle differences between evolutionary computing and swarm intelligence are touched on in section 2.6. Various extensions and enhancements are presented in sections 2.7 and 2.8 respectively. Section 2.9 discusses a number of PSO application areas.

2.2 Function Optimization

Function optimization is concerned with finding an optimal solution to an *objective*, or *cost* function, describing a problem. An objective function may additionally be constrained by a set of equality and/or inequality constraints. As an example, an objective function may quantify the allocation of resources, such as distribution of chemicals in a process control environment. The cost associated with the distribution of an individual chemical, as well as its interaction with other chemicals, may act as constraints on the interpretation of the optimal allocation within the process environment.

In this thesis, optimization of an objective function $f(\mathbf{x})$ over a n-dimensional space, $\mathbf{\Omega}$ is considered, where $\mathbf{x} \in \mathbf{\Omega}$, and $\mathbf{x} = [x_1, x_2, \cdots, x_n]^T$. $\mathbf{\Omega}$ may either be a real-valued space, such that $\mathbf{\Omega} \subseteq \mathbb{R}^n$, or a discrete space, where each component x_i is taken from a set of values, such as $\{0, 1\}$.

"Minimizing the objective function $f(\mathbf{x})$" is formalized as finding a *global minimizer* $\mathbf{x}^*$, subject to

$$f(\mathbf{x}^*) \leq f(\mathbf{x}), \quad \forall \mathbf{x} \in \mathbf{\Omega}. \tag{2.1}$$

If $f(\mathbf{x}^*) < f(\mathbf{x})\ \forall \mathbf{x} \in \mathbf{\Omega}$, then $\mathbf{x}^*$ is considered to be a *strict global minimizer*. Equation (2.1) defines an *unconstrained minimization problem*. If a global minimizer for a function $f(\mathbf{x})$ cannot be found, it is often acceptable to search for a *local minimizer*. A local minimizer $\mathbf{x}_L^*$, satisfies

$$f(\mathbf{x}_L^*) \leq f(\mathbf{x}), \quad \forall \mathbf{x} \in L, \quad L \subset \mathbf{\Omega}.$$

The search space limitation, $L \subset \mathbf{\Omega}$, signifies that *only* L is searched. It is possible that better solutions may exist in $\mathbf{\Omega}$, but they are not considered.

In contrast with the unconstrained optimization problem defined in (2.1), a constrained minimization problem is defined as

$$\begin{array}{lll} \text{minimizing} & f(\mathbf{x}), & \mathbf{x} \in \mathbf{\Omega} \\ \text{subject to} & g_i(\mathbf{x}) \leq 0, & i = 1, \ldots, k \\ & h_i(\mathbf{x}) = 0, & i = 1, \ldots, m. \end{array}$$

Let $\mathbf{g}(\mathbf{x}) \leq \mathbf{0}$ represent the set of k inequality constraints, $g(\mathbf{x})_i \leq 0$. The same interpretation applies to the set of m equality constraints, $\mathbf{h}(\mathbf{x}) = \mathbf{0}$. Minimization problems can be converted into maximization problems by reversing the sign of $f(\mathbf{x})$, i.e.

$$\min f(\mathbf{x}) = \max(-f(\mathbf{x})) \tag{2.2}$$

Likewise, any constraints can be adapted by reversing the sign. The rest of this thesis considers minimization problems, with the assumption that maximization problems can be converted to minimization problems using equation (2.2).

For optimization problems with multiple objectives, the objective function $f(\mathbf{x})$ may be extended to represent a set of w sub-objectives, optimized concurrently [14]. For such an objective function, vector notation for f is preferred, i.e.

$$\mathbf{f}(\mathbf{x}) = \{f(\mathbf{x}_1), f(\mathbf{x}_2), \ldots, f(\mathbf{x}_w)\}$$

The applied optimization method need not make any special provisions for such an objective function. How $\mathbf{f}(\mathbf{x})$ is evaluated depends on the underlying problem. Objective functions consisting of multiple sub-objectives are discussed in more detail in section 2.9.2. A minimization problem is considered to be unimodal when its objective function $f(\mathbf{x})$ has a single global minimum point $\mathbf{x}^*$. That is,

$$f(\mathbf{x}^*) \leq f(\mathbf{x}), \quad \forall \mathbf{x} \in \mathbf{\Omega}.$$

Figure 2.1 gives an example of an unimodal objective function, with domain $x \in [-3, 3] \subset \mathbb{R}$. Unlike a unimodal problem, which has a single minimum $\mathbf{x}^*$, a multimodal function may have multiple minima. Minima may be local or global, relative to the objective function. A *local* minimum $\mathbf{x}_L^*$ is subject to

$$f(\mathbf{x}_L^*) \leq f(\mathbf{x}), \quad \forall \mathbf{x} \in L$$

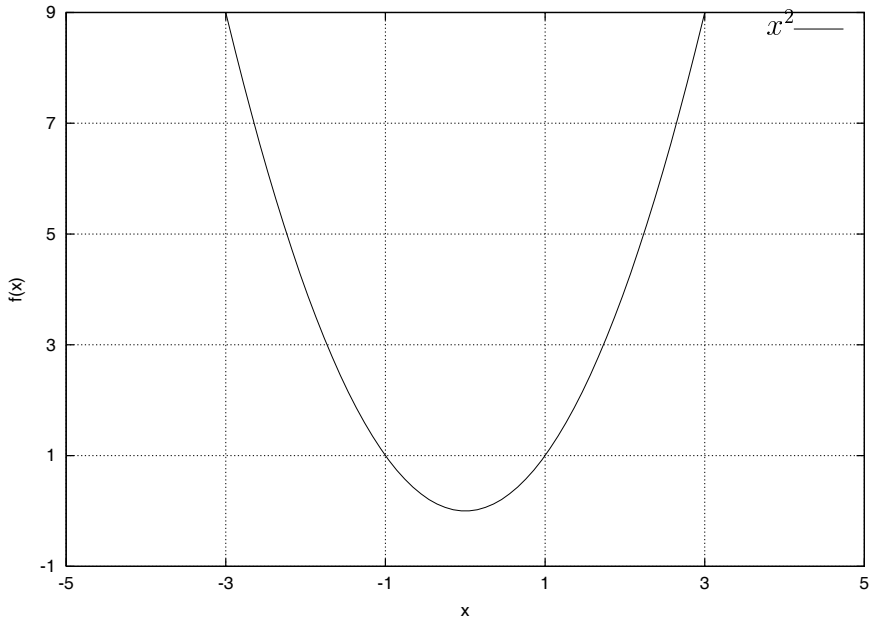


Figure 2.1: $f(x) = x^2$, a unimodal function.

where $L \subset \mathbf{\Omega}$. For a function $f(\mathbf{x})$, there may be more than one minimum. One of the minima may be a global minimum $\mathbf{x}^{**}$, with

$$f(\mathbf{x}^{**}) \leq f(\mathbf{x}_L^*)$$

for all local minima $f(\mathbf{x}_L^*)$. Figure 2.2 depicts a multimodal function with two equal minima, indicated by symbols P and Q. Classical optimization techniques consider only a single solution to an optimization problem f at any given time. Minimizing a multimodal function such as in figure 2.2 presents a problem, since a single solution may not always be the only acceptable solution. It is possible to solve multimodal problems by repeatedly applying a technique that maintains a single solution to the same search space. Such an approach is known as a *sequential* technique. A technique that locates multiple solutions concurrently, is known as a *parallel* technique.

This thesis will focus on developing evolutionary techniques that solve unconstrained optimization problems in continuous, real-valued spaces. Niching techniques are developed to solve both unimodal and multimodal optimization problems, with single or multi–objective cost functions.

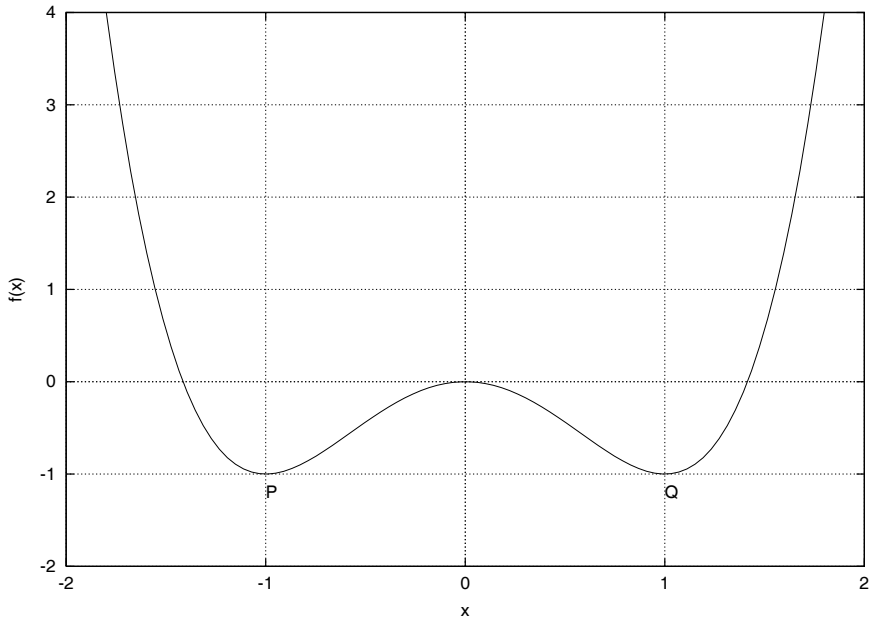


Figure 2.2: Function $f(x) = x^4 - 2x^2$ with two equal minima, P and Q.

2.3 Evolutionary Computing

Evolution is a process through which individuals in a community strive to survive in their environment. An individual's survival depends on its ability to continually adapt to make better use of the resources available in its environment and to co-exist with other individuals. Several factors may influence how evolution in an environment takes place. These factors include:

- Changes in an environment, such as the availability of resources required for survival.
- Interaction between individuals in a community of individuals residing in an environment.
- Influences of other communities.

Evolutionary computing simulates the evolution process by representing candidate solutions as a population of individuals within an environment and evolving them over time [3]. To facilitate the evolutionary process, individuals are evaluated at regular intervals,

or *generations*, using a *fitness function*. A fitness function evaluates the quality of a potential solution represented by an individual. Depending on the type of evolutionary algorithm, a population of individuals is adapted using evolutionary operators and strategies in an attempt to find an optimal solution. Evolutionary operators include mutation, reproduction, selection and competition between individuals. The adapted individuals represent a next generation. The process of creating a next generation can be repeated until an acceptable solution is found, or until a maximum number of generational steps have been taken.

Evolutionary operators manipulate individuals in an effort to produce better individuals. In an environment where resources are limited, individuals need to be genetically superior to their siblings in order to survive. The characteristics of an individual, or the candidate solution represented by it, can be interpreted in two different ways. An individual's *genotype* describes its genetic composition inherited from its parents. Genetic information may initially be random, but over a number of generations it may become ordered to represent experiences gained by its parents. Genetic information is passed on to offspring. An individual's *phenotype* is an expression of its behavior in an environment. Mayr identified two such relationships, *pleiotropy* and *polygeny* [63].

Pleiotropy describes a situation where changes to genes may affect phenotypic behavior in an unexpected way. Polygeny describes the interaction among genetic information to produce a specific phenotypic trait. In order to adapt or change the phenotypic behavior, all genetic information has to change.

Several different evolutionary computing approaches have been suggested, such as genetic algorithms, genetic programs, evolutionary programming and evolutionary strategies [3]. Niching techniques have been specifically investigated under the wing of genetic algorithms. The next section presents a short introduction to genetic algorithms.

2.4 Genetic Algorithms

Optimization with genetic algorithms (GAs) is generally regarded to have been the first evolutionary computing technique developed and applied [24]. The technique was introduced in 1975 by Holland [36]. Since the focus of this thesis falls on particle swarm

Symbol	Meaning
g	Generation index
g_{max}	Maximum number of generations
C_g	Population of individuals in generation g
$C_{g,i}$	Individual i of population C_g
$f(C_{g,i})$	Fitness of individual i at generation g
f_ϕ	Minimum fitness threshold

Table 2.1: GA symbols

optimization, the rest of this section presents only a general overview of GAs, to emphasize similarities and differences between the two paradigms. The interested reader is referred to [31] for a more complete treatment.

A genetic algorithm performs optimization by evolving a genetic representation of a problem. A population of individuals is evolved over a number of generations, using the following algorithm (refer to table 2.1 for an explanation of symbols used):

1. Set $g = 0$.
2. Initialize chromosomes of the initial population C_g.
3. Evaluate the fitness of each individual in the current population.
4. While not converged:
   (a) $g = g + 1$
   (b) Select parents from C_{g-1}
   (c) Apply crossover operators to form offspring
   (d) Apply a mutation operator to offspring
   (e) Create the new population C_g by selecting individuals from the newly created offspring and the previous population C_{g-1}.

Individuals may be initialized randomly or by using some technique relevant to the optimization problem. A population is considered to have converged either after a maximum

number of new populations have been created (when $g \geq g_{max}$), or when $f(C_{g,b}) < f_\phi$, where $\exists\ b \in C_g | f(C_{g,b}) \leq f(C_{g,i}),\ \forall\ i \in C_g$. The threshold f_ϕ represents a value that can be used to determine when an individual represents an acceptable solution. The representation of individuals, fitness functions and the GA evolutionary operators are now discussed in more detail.

2.4.1 Representation of Individuals

The traditional GA represents individuals with fixed length bit-strings, although variable length strings have also been used [32]. The significance of each individual bit varies according to the problem domain being optimized:

- An individual bit may represent a *single property* of the candidate solution represented by an individual. Properties therefore have binary values, and can only be present or absent. This scenario could occur in a process optimization problem where the presence or absence of chemical gases determine the outcome of a manufacturing step.

- A bit string can be an encoding of a *set of nominal values*. In such a case, the number of bits would be equal to the number of possible nominal values. The bit string may represent only a single nominal value, or combinations thereof.

- Bits may encode *numerical values*. An individual's genotype then has to be converted to a phenotypic representation before it can be interpreted. A 15-bit binary string representing a real-value between 0 and 1 may for example be decoded to its phenotype ρ using the formula

$$\rho = \frac{1}{2^{15}} \sum_{i=1}^{15} 2^i b_i. \tag{2.3}$$

  where b_i represents the individual bits in a bit string $\mathbf{b}$.

GAs have also been tested using real-valued representations of genetic information. Instead of using a bit-string, a vector of numerical values is optimized. Janikow and Michalewicz reported superior results when comparing numerical representations with binary representations on real-valued numerical optimization problems [45].

2.4.2 Fitness Functions

A fitness function maps an individual's genotype to a scalar value that can be interpreted to determine the quality of the candidate solution represented by the individual. Fitness functions are not only used to determine the quality of individuals, but are also used by crossover and mutation operators (discussed in section 2.4.3). Performing crossover on individuals with similar fitness may accelerate convergence in a unimodal problem. Mutation on individuals with poor fitness may lead to better solutions. These operators depend on an accurate interpretation of a candidate solution by the fitness function. In multi–objective optimization problems, the fitness function represents all the individual objectives. When dealing with constrained optimization problems, the equality and inequality constraints imposed on a search space need not be part of a fitness function.

2.4.3 Evolutionary Operators

Section 2.3 pointed out that evolutionary computing approaches utilize some form of evolutionary operator to improve the quality of individuals in a population. This section presents the following operators utilized by GAs: crossover (or recombination), selection and mutation.

Crossover

The crossover operator operates on binary or numerical genotypic representations. It produces offspring from two parent individuals by a recombination of their genetic material. The recombination process exchanges sections of the parents' genetic strings to form two new strings, representing offspring. The process of determining which parts of the genetic material to exchange, allows for the definition of several different crossover techniques, of which the most popular are listed below:

One-point crossover: One-point crossover randomly selects a position i in a string of length l, where $i \in [1, l-1]$. All positions after i are swapped.

Two-point crossover: Two-point crossover randomly selects two positions, i_1 and i_2, where $i_1, i_2 \in [1, l-1]$, subject to $i_1 < i_2$. All positions between i_1 and i_2 are

swapped.

Uniform crossover: The uniform crossover operator swaps all positions in the strings of parents with a probability p_{uc}. If $p_{uc} = 0.5$, positions are swapped with equal probability.

Mutation

The mutation operator is applied with a probability p_m. It inverts bits in the binary representation of the search space, or adds small random values to numerical chromosomes to extend the diversity of a population. The goal of this process is to allow for the representation of chromosome values and consequently positions in the search space that may not have been possible as a result of crossover/recombination operators. Unfortunately, if care is not taken, mutation may negatively affect highly fit individuals: The mutation probability p_m is therefore set to a low value.

Selection Operators

The process of selection is concerned with creating a new population of individuals C_g from a previous population C_{g-1} and newly created offspring. Random selection of individuals from these groups may lead to the loss of potentially good individuals and the inclusion of individuals that contain inferior genetic material, that may slow down the convergence process. *Elitism* introduces a selection rule that is biased towards highly fit individuals. Before a new population is selected, all available individuals are ranked based on their fitness and only the most fit subset of the population is carried over to the next generation. Alternative selection schemes include *tournament* and *roulette wheel* selection.

2.5 Particle Swarm Optimization

The particle swarm optimization (PSO) algorithm, originally introduced by Kennedy and Eberhart [49], is modeled after the social behavior of birds in a flock. PSO is a population based search process where individuals, referred to as particles, are grouped into

a swarm. Each particle in the swarm represents a candidate solution to an optimization problem. In a PSO system, each particle is "flown" through the multidimensional search space, adjusting its position in search space according to its own experience and that of neighboring particles. A particle therefore makes use of the best position encountered by itself and that of its neighbors to position itself toward an optimal solution. The effect is that particles "fly" toward a minimum, while still searching a wide area around the best known solution. The performance of each particle (i.e. the "closeness" of a particle to the global optimum) is measured using a predefined fitness function which encapsulates the characteristics of the optimization problem.

Several authors have suggested diversity improvement and convergence acceleration additions to the PSO. The algorithm's convergence behavior has also been extensively analyzed [13, 68, 69, 87]. The rest of this section presents a general mathematical abstraction of the PSO algorithm to establish a uniform notation, followed by a discussion of extensions to the algorithm.

Figure 2.3 summarizes the standard PSO algorithm. Each particle i in a swarm of particles maintains the following information:

- $\mathbf{x}_i$: The particle's *current position* in the search space;
- $\mathbf{v}_i$: Its *current velocity*;
- $\mathbf{y}_i$: The *personal best position* discovered thus far.

In all cases, $\mathbf{x}_i$ represents a position in an unconstrained, continuous, n-dimensional search space, i.e. $\mathbf{x} = [x_1, x_2, \cdots, x_n]^T$. The velocity vector $\mathbf{v}$ contains a velocity element for each dimension of the search space, i.e. $\mathbf{v} = [v_1, v_2, \cdots, v_n]^T$. Different dimensions are assumed to be independent [13, 87]. The personal best position associated with a particle i is the best position that the particle has visited thus far, i.e. a position that yielded the highest fitness value for that particle. Again, f denotes the objective function, and $f(\mathbf{x}_i)$ the fitness of particle i.

Particle positions may be initialized randomly within the search space. Some problems, such as the niching applications discussed later, benefit from the use of uniform initial particle positions. Particle velocities are initialized to be within the value range $[-v_{max}, v_{max}]$. The significance of v_{max} is discussed later in this section.

1. Initialize all particles in the swarm to random positions within the search space.
2. Initialize velocity vectors.
3. Initialize particle personal best positions to be equal to the current positions of the particles.
4. Repeat until converged:
   (a) Update the fitness of each particle i using the fitness function f and the particle's current position.
   (b) Update each particle's personal best position
   (c) Update the global best particle position
   (d) Update each particle's velocity.
   (e) Update each particle's position.

Figure 2.3: The general PSO algorithm

The personal best of a particle at a time step t is updated as:

$$\mathbf{y}_i(t+1) = \begin{cases} \mathbf{y}_i(t) & \text{if } f(\mathbf{x}_i(t+1)) \geq f(\mathbf{y}_i(t)) \\ \mathbf{x}_i(t+1) & \text{if } f(\mathbf{x}_i(t+1)) < f(\mathbf{y}_i(t)) \end{cases} \tag{2.4}$$

Two main approaches to PSO exist, namely *lbest* and *gbest*, where the difference is in the neighborhood topology used to exchange experience among particles. For the *gbest* model, the best particle is determined from the entire swarm, and all other particles flock towards this particle. If the position of the best particle is denoted by the vector $\hat{\mathbf{y}}$, then

$$\hat{\mathbf{y}}(t) = \arg\min_{1 \leq i \leq s} f(\mathbf{y}_i(t)) \tag{2.5}$$

where s is the total number of particles in the swarm. For the *lbest* model, a swarm is divided into overlapping neighborhoods of particles. For each neighborhood N_j, a best particle position is designated by $\hat{\mathbf{y}}_j$. This best particle is referred to as the *neighborhood*

best particle, defined as

$$N_j = \{\mathbf{y}_{i-l}(t), \mathbf{y}_{i-l+1}(t), \ldots, \mathbf{y}_{i-1}(t), \mathbf{y}_i(t), \tag{2.6}$$

$$\mathbf{y}_{i+1}(t), \ldots, \mathbf{y}_{i+l-1}(t), \mathbf{y}_{i+l}(t)\} \tag{2.7}$$

$$\hat{\mathbf{y}}_j(t+1) \in N_j \mid f(\hat{\mathbf{y}}_j(t+1)) = \min\{f(\mathbf{y}_i)\}, \forall \mathbf{y}_i \in N_j \tag{2.8}$$

Neighborhoods are usually determined using particles indices, although topological neighborhoods have also been used [47, 48, 51, 85]. The *gbest* PSO is a special case of *lbest* with $l = s$, where l is the number of particles per neighborhood, and s is the total number of particles in the swarm; that is, the neighborhood is the entire swarm. For each iteration of a *gbest* PSO, the j^{th}-dimension of particle i's velocity vector, $\mathbf{v}_i$, and its position vector, $\mathbf{x}_i$, is updated as follows:

$$\begin{aligned} v_{i,j}(t+1) &= v_{i,j}(t) + c_1 r_{1,j}(t)(y_{i,j}(t) - x_{i,j}(t)) + \\ &\quad c_2 r_{2,j}(t)(\hat{y}_j(t) - x_{i,j}(t)) \end{aligned} \tag{2.9}$$

$$\mathbf{x}_i(t+1) = \mathbf{x}_i(t) + \mathbf{v}_i(t+1) \tag{2.10}$$

where c_1 and c_2 are acceleration constants and $r_{1,j}(t), r_{2,j}(t) \sim U(0,1)$. For each iteration of the *lbest* PSO, the velocity update for particle i is defined to be

$$\begin{aligned} v_{i,j}(t+1) &= v_{i,j}(t) + c_1 r_{1,j}(t)(y_{i,j}(t) - x_{i,j}(t)) + \\ &\quad c_2 r_{2,j}(t)(\hat{y}_{i,j}(t) - x_{i,j}(t)) \end{aligned} \tag{2.11}$$

Upper and lower bounds are specified on $\mathbf{v}_i$ to avoid too rapid movement of particles in the search space; that is, $v_{i,j}$ is clamped to the range $[-v_{max}, v_{max}]$.

2.6 Evolutionary Computing vs. Swarm Intelligence

Sections 2.4 and 2.5 presented genetic algorithms and particle swarm optimization respectively. The GA is an evolutionary computing (EC) technique, while PSO is classified as a swarm intelligence (SI) approach. This section outlines similarities and fundamental differences between EC and SI.

Behavioral similarities between EC and SI can be categorized based on the following points:

- A group of agents, be it a population of individuals or swarm of particles, explore a search space.
- Each individual or particle represents a candidate solution to an optimization problem.
- A fitness function is used to evaluate the quality of a candidate solution.

Regardless of their similarities, EC and SI are motivated by radically different behavioral models. The following points elucidate the conceptual differences between the two paradigms:

- SI uses the behavior of a swarm of particles in a search space to optimize a problem. Particles may be simple, but their collective behavior in a swarm solves complex problems. The behavior of the swarm as a whole and individual particles is therefore tightly coupled.
- Individuals in EC algorithms, such as GAs, each perform an independent exploration of a search space. Behavioral similarity is maintained through generational recombination operators.
- SI coordinates movement of particles in a search space through *social* interaction. Social interaction shares information between particles about potential solutions. Particles in SI thus exert a direct influence on each other, i.e. a particle's candidate solution is not just as a result of the particle's exploration of the search space, but it is also determined by solutions found by other particles.
- The movement of particles in PSO is influenced by a conscience factor, known as a 'personal best', as well as the social component mentioned above. SI thus retains a memory of previous experience. EC retains no knowledge of favorable solutions and have no direct improvement strategy biased towards possible solutions (EC retains possible solutions only through recombination operators and fitness rank-based selection schemes).

From the above it follows that the *collective* efforts of a set of agents in a search space clearly differentiates the methodologies behind SI from EC.

2.7 Extensions to the PSO

Section 2.5 presented the standard PSO algorithm. Numerous authors have proposed extensions to improve on the original algorithm. The proposed extensions attempt to satisfy the following requirements:

- More effective traversal of a search space, whilst ensuring that search efforts are not duplicated.
- Accelerated convergence, without disregarding potential global solutions in multi-modal domains.

Evolutionary techniques that have been successfully applied to evolutionary algorithms and GAs, have also been successfully adapted to the PSO. This section looks into these methods. Section 2.7.1 investigates convergence acceleration techniques that allow the PSO to more efficiently locate solutions. Diversity improvements techniques are then discussed in section 2.7.2.

2.7.1 Convergence Acceleration Techniques

Convergence acceleration (CA) techniques use different methods to help the PSO algorithm solve optimization problems faster. CA techniques

- directly affect particle trajectories around solutions in a search space, and
- attempt to more effectively share information about previously discovered solutions.

The rest of this section reviews a number of well-known and more recent CA techniques.

Inertia Weight

The use of the inertia weight w was introduced by Shi and Eberhart [80] and was not present in the original paper by Kennedy and Eberhart [49]. The inertia weight controls

the influence of a particle's previous velocity, resulting in a memory effect. The velocity update equation (equation (2.9)) is redefined to be

$$\begin{aligned} v_{i,j}(t+1) &= wv_{i,j}(t) + c_1 r_{1,j}(t)(y_{i,j}(t) - x_{i,j}(t)) + \\ & \quad c_2 r_{2,j}(t)(\hat{y}_j(t) - x_{i,j}(t)) \end{aligned} \tag{2.12}$$

Decreasing w from a relatively large value to a small value over time results in a rapid initial exploration of the search space, that gradually becomes more focussed. Small w-values result in small adaptations to particle positions, effectively yielding a local search. When $w = 1$, equation (2.12) is equivalent to the original PSO velocity update in equation (2.9). Van den Bergh investigated the effect of w on the convergence properties of the PSO [87]. It was found that there exists a strong relationship between w and the acceleration coefficients, c_1 and c_2, namely that the relationship between w, c_1 and c_2 may be expressed as

$$w > \frac{1}{2}(c_1 + c_2) - 1 \tag{2.13}$$

Parameter settings that satisfy the relationship in equation (2.13) lead to convergent particle trajectories. The interested reader is referred to [87] for a thorough treatment of the subject.

Shi and Eberhart suggested using a fuzzy controller to adapt the inertia weight [81, 82]. Their controller adapts w depending on the global best solution's distance from an optimum, found by evaluating $f(\hat{\mathbf{y}})$. On unimodal functions, the fuzzy controller PSO exhibited favorable performance when compared to a PSO using a linearly decreasing inertia weight [81].

Constriction Factor

The term v_{max} is used to limit the maximum velocity that a particle can achieve. Particles with a very high velocity have distinctly divergent behavior, as their position updates are erratic, and do not focus on a solution. Clerc and Kennedy showed that v_{max} is not necessary if an alternative velocity update equation is used, namely [13]

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

where

$$\chi = \frac{2}{\left|2 - \varphi - \sqrt{\varphi^2 - 4\varphi}\right|}$$

with $\varphi = c_1 + c_2$ and $\varphi > 4$. The parameter χ 'constricts' rapid growth of the velocity vector that may lead to non-convergent particle trajectories. Clerc preceded the derivation of this model by an analysis obtained by rewriting equations (2.12) and (2.10) as difference equations, and performing an eigenvector analysis. Eberhart and Shi found that the use of the constriction factor approach in conjunction with velocity clamping leads to improved performance [22].

Evolutionary Computing Generational Operators

Angeline used an evolutionary computing selection rule to improve interaction between particles in the particle swarm [2]. Before velocity updates take place for individual particles, each particle is assigned a performance score based on a comparison to a randomly selected subset of the particle swarm. A particle scores a mark for each of the other particles in the subset that have worse fitness than itself. Particles are then ranked according to this score. The position vectors of the top half of the swarm are copied onto that of the bottom half. The personal best positions of the bottom half remain unchanged. The selection process thus resets 'poor performers' to locations within the search space that have yielded better results.

Løvbjerg *et al* used a *breeding* operator and particle subpopulations to achieve faster convergence (subpopulations are discussed in section 2.7.2) [60]. To identify breeding particles, each particle in the swarm is assigned a breeding probability, p_b. Note that particle fitness does not influence the assignment of p_b: any particle can be chosen for breeding. All particles marked for breeding are then randomly paired off until the set of marked particles is empty. New particle positions are calculated using an arithmetic crossover operator. The offspring of 'parent' particles i and j occur, at time step $t+1$, at positions

$$\mathbf{x}_i^{'}(t+1) = p_i \mathbf{x}_i(t+1) + (1 - p_i)\mathbf{x}_j(t+1)$$

and

$$\mathbf{x}_j^{'}(t+1) = p_i \mathbf{x}_j(t+1) + (1 - p_i)\mathbf{x}_i(t+1).$$

Velocity vectors are initialized to the normalized sum of the parents' velocity vectors:

$$\mathbf{v}_i^{'} = \frac{\mathbf{v}_i + \mathbf{v}_j}{\|\mathbf{v}_i + \mathbf{v}_j\|}\|\mathbf{v}_i\|$$

and

$$\mathbf{v}_j^{'} = \frac{\mathbf{v}_i + \mathbf{v}_j}{\|\mathbf{v}_i + \mathbf{v}_j\|}\|\mathbf{v}_j\|$$

respectively. Performing breeding on a global scale may avoid stagnation on local optima. Offspring particle positions are always initialized to a position within a hypercube spanned by $\mathbf{x}_i$ and $\mathbf{x}_j$. Personal best positions $\mathbf{y}_i^{'}$ and $\mathbf{y}_j^{'}$ are initialized to $\mathbf{x}_i^{'}$ and $\mathbf{x}_j^{'}$ respectively.

Objective Function Stretching

Parsopoulos *et al* introduced the "stretched" PSO (SPSO) [71]. The SPSO performs a transformation on the landscape of the original fitness (or objective) function, called *stretching*. The technique is applied as follows: A particle swarm is trained using the *gbest* algorithm. Once the PSO has identified a local minimum $\mathbf{x}^*$ by comparing particle fitnesses to a performance threshold value, the objective function is stretched so that for each point $\mathbf{x}$, where $f(\mathbf{x}) < f(\mathbf{x}^*)$, $\mathbf{x}$ is unaffected. All other points, such that $f(\mathbf{x}) \geq f(\mathbf{x}^*)$ holds, are stretched so that $\mathbf{x}^*$ becomes a local maximum. All particles are then repositioned randomly. The fitness function $f(\mathbf{x})$ is redefined to $H(\mathbf{x})$, where

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

and

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

Suggested parameter values are $\gamma_1 = 10000$, $\gamma_2 = 1$ and $\mu = 10^{-10}$. For a minimization problem, the sign$(\cdot)$ function is defined as:

$$\text{sign}(x) = \begin{cases} +1, & x > 0 \\ 0, & x = 0 \\ -1, & x < 0 \end{cases}$$

where x is a scalar value. Parsopoulos *et al* reported promising results when applying this technique to multimodal functions [71]. Objective function stretching is discussed in more detail in section 3.4.

Division of Labour

The 'division of labour' PSO (DoLPSO) attempts to improve the PSO algorithm's convergence around optima [91]. The DoLPSO method constantly monitors the activity of all particles. When a particle has not improved its candidate solution over a number of training iterations, it is re-assigned a different 'task', namely optimizing the best solution that the swarm has found thus far. This task is referred to as the *local search* task. A particle can therefore be performing one of two tasks:

Task 1: Exploring the search space using the *gbest* algorithm.

Task 2: The *local search* task.

When a particle i starts to perform the *local search* task at iteration t, its position vector is initialized to $\mathbf{x}_i(t) = \hat{\mathbf{y}}(t)$. Particle i's velocity vector is also randomly re-initialized and scaled to be no larger than the velocity of the global best particle, i.e. $\mathbf{v}_i$ is initialized so that $v_{i,j} < v_{g,j}$, $j \in [1, n]$ and g is the index of the *gbest* particle. Engaging in the *local search* task decreases swarm diversity, encouraging faster convergence. To facilitate the decision process behind 'division of labour', the following parameters are associated with a particle i:

- X_i: Particle state, i.e. which task is performed.
- θ_i: A response threshold.
- ζ_i: A particle stimulus.

When $X_i = 0$, particle i explores the search space (**Task 1**), and when $X_i = 1$, it performs the *local search* task (**Task 2**). Particle i's state X_i changes from **Task 1** to **Task 2** with a probability P per time step

$$P(X_i = 0 \rightarrow X_i = 1) = T_{\theta_i}(\zeta_i) = \frac{\zeta_i^\ell}{\theta_i^\ell + \zeta_i^\ell}.$$

The parameter ℓ controls the steepness of the response function T (the term ζ_i^ℓ thus represents the stimulus ζ_i raised to the power ℓ). The stimulus ζ_i represents the number of iterations since the last *improvement* to $f(\mathbf{x}_i)$. Therefore, when i's fitness *does not*

improve over time, ζ_i will increase. When i's fitness *does* improve over time, ζ_i will remain zero and $X_i = 0$. Vesterstrøm *et al* reported improved convergence when comparing DoLPSO to GAs, the traditional PSO and simulated annealing [91].

Multi-Phase Generalization

Al-Kazemi and Mohan suggested a 'multi-phase' extension to the PSO [1]. This technique varies the optimization goal of each particle over time, effectively forcing it to change its trajectory. The velocity update in equation (2.9) is redefined to be

$$v_{i,j}(t+1) = c_v v_{i,j}(t) + c_g \hat{y}_j(t) + c_x x_{i,j}(t) \tag{2.14}$$

The parameters c_v, c_g and c_x may assume different values, that are determined by a particle's associated *group* and *phase*. Groups and phases segment a swarm into subswarms that pursue different goals. The following parameter values were suggested for two groups, each with two phases [1]:

Phase 1, group 1: $c_v = 1$, $c_g = 1$, $c_x = -1$

Phase 1, group 2: $c_v = 1$, $c_g = -1$, $c_x = 1$

Phase 2, group 1: $c_v = 1$, $c_g = -1$, $c_x = 1$

Phase 2, group 2: $c_v = 1$, $c_g = 1$, $c_x = -1$

Particles in phase 1, group 1 move towards $\hat{\mathbf{y}}$, while particles in group 2 move away from $\hat{\mathbf{y}}$ towards $\mathbf{x}$. In phase 2, the roles are reversed. (It is not clear why the explicit specification of c_v was necessary, as it does not affect the optimization process.) The number of phases and groups is a user tunable parameter and must be known before the algorithm starts. A group of particles that are in the same phase all have identical goals. Changing phase is controlled by a *phase change frequency* (PCF). PCF indicates a number of evolutionary iterations of the PSO algorithm. Al-Kazemi and Mohan also proposed that a PCF free version of the algorithm would initiate a phase change when a pre-determined number of iterations yielded no improvement. The algorithm randomly re-initializes particle velocities to avoid converging on sub-optimal solutions after a pre-determined number of iterations, called the *velocity change variable* (VC). In quite a

departure from the original PSO, the multi-phase generalization uses a hill-climbing approach that only updates particle positions if a calculated new position presents an improvement. A particle will therefore always occupy an optimal position relative to its own history. This explains why a personal best position is not taken into consideration in equation (2.14).

Tested on both discrete and continuous problems, the multi-phase PSO outperformed the standard PSO, GAs and evolutionary programs [1].

The Guaranteed Convergence Particle Swarm Optimizer

It is decidedly difficult to determine proper values for parameters such as c_1, c_2 and w without a theoretical basis. Some authors have attempted to determine acceptable values empirically, such as in [8]. Recently, extensive analysis of particle trajectories within the PSO was done independently by Van den Bergh [87] and Clerc *et al* [13] to determine how to guarantee convergent swarm behavior. In the rest of this section, the *guaranteed convergence particle swarm optimizer* (GCPSO) is presented, which resulted from the analysis done in [87].

The *gbest* algorithm exhibits an unwanted property: when $\mathbf{x}_i = \mathbf{y}_i = \hat{\mathbf{y}}$ (for any particle i), the velocity update in equation (2.5) depends solely on the $w\mathbf{v}_i(t)$ term. When a particle approaches the global best solution, its velocity property also approaches zero, which implies that eventually all particles will stop moving. This behavior does not guarantee convergence to a global best solution, or even a local best, only to a best position found thus far [87]. Van den Bergh and Engelbrecht introduced a new algorithm, called the GCPSO, to pro-actively counter this behavior in a particle swarm and to ensure convergence [90].

The GCPSO algorithm works as follows: Let τ be the index of the global best particle. The idea of GCPSO is then to update the position of particle τ as

$$x_{\tau,j}(t+1) = \hat{y}_j(t) + wv_{\tau,j}(t) + \rho(t)(1 - 2r_{2,j}). \tag{2.15}$$

To achieve this, the velocity update of τ is defined as

$$v_{\tau,j}(t+1) = -x_{\tau,j}(t) + \hat{y}_j(t) + wv_{\tau,j}(t) + \rho(t)(1 - 2r_{2,j}). \tag{2.16}$$

In equation (2.16), the term $-\mathbf{x}_\tau$ 'resets' the particle's position to the global best position $\hat{\mathbf{y}}$, $w\mathbf{v}_\tau$ signifies a search direction, and $\rho(t)(1-2\mathbf{r}_2(t)))$ adds a random search term to the equation; $\rho(0)$ is initialized to 1.0, with $\rho(t)$ defined as

$$\rho(t+1) = \begin{cases} 2\rho(t) & \text{if } \#successes > s_c \\ 0.5\rho(t) & \text{if } \#failures > f_c \\ \rho(t) & \text{otherwise} \end{cases} \tag{2.17}$$

A 'failure' occurs when $f(\hat{\mathbf{y}}(t)) = f(\hat{\mathbf{y}}(t-1))$ and the counter variable $\#failures$ is subsequently incremented (i.e. no apparent progress has been made). A success then occurs when $f(\hat{\mathbf{y}}(t)) \neq f(\hat{\mathbf{y}}(t-1))$. The control threshold values f_c and s_c are adjusted dynamically. That is,

$$s_c(t+1) = \begin{cases} s_c(t)+1 & \text{if } \#failures(t+1) > f_c \\ s_c(t) & \text{otherwise} \end{cases} \tag{2.18}$$

$$f_c(t+1) = \begin{cases} f_c(t)+1 & \text{if } \#successes(t+1) > s_c \\ f_c(t) & \text{otherwise} \end{cases} \tag{2.19}$$

This arrangement ensures that it is harder to reach a success state when multiple failures have been encountered, and likewise, when the algorithm starts to exhibit overly confident convergent behavior, it is forced to randomly search a smaller region of the search space surrounding the global best position. Furthermore, the counter values are adapted using the equations

$$\#successes(t+1) > \#successes(t) \Rightarrow \#failures(t+1) = 0$$
$$\#failures(t+1) > \#failures(t) \Rightarrow \#successes(t+1) = 0$$

The algorithm is repeated until ρ becomes sufficiently small, or until stopping criteria are met. Stopping the algorithm when ρ reaches a lower bound is not advised, as it does not necessarily indicate that all particles have converged – other particles may still be exploring different parts of the search space.

Note that only the best particle in the swarm uses the modified updates in equations (2.15) and (2.16), the rest of the swarm uses the normal velocity and position updates defined in equations (2.9) and (2.10).

2.7.2 Swarm Diversity Enhancements

This section discusses a number of PSO diversity enhancement techniques. Diversity enhancement techniques attempt to ensure that the PSO algorithm explores as much of a search space as possible.

Subpopulations

Løvbjerg *et al*'s subpopulation scheme [60] segments the particle swarm into a number of independent particle populations. Each population maintains its own global best solution, based on the experiences of all particles that are members of the swarm. Subpopulations evolve independently, except when inter-population crossover takes place. (The crossover operator was defined as part of Løvbjerg *et al*'s breeding strategy discussed in section 2.7.1). Performing inter-population breeding facilitates a form of formalized social information exchange. Formal 'contact' between swarms ensures that all swarms are aware of the global best solution located by all the subpopulations. This information exchange may lead to faster convergence.

Parsopoulos and Vrahatis used a population based approach to solve multi–objective optimization problems [75]. Their technique, VEPSO, was motivated by the VEGA (Vector Evaluated Genetic Algorithm) approach introduced by Schaefer [79]. VEPSO optimizes a multi–objective problem, consisting of two goals, i.e. the fitness function is defined as $\mathbf{f} = \{f_1, f_2\}$. Two independent swarms each optimize one of the objective functions. Then, if swarm A optimizes function f_1 and swarm B optimizes function f_2, the *gbest* particle from A is used in the velocity update of particles in B, and vice versa. This technique is reported to efficiently and accurately solve simple multi–objective optimization problems [75].

Particle Neighborhoods

A particle subpopulation is just a different way of defining a particle neighborhood. Subpopulations

- maintain several concurrent particle swarms, each with its own global solution;
- may explicitly allow information exchange between subpopulations.

Particle neighborhoods are more dynamic. Neighborhoods

- dynamically determine 'global best' solutions that depend on the subset of particles that make up a local particle population at a certain point in time;
- may overlap, effectively implementing an implicit information exchange.

Kennedy tested a number of different particle neighborhood topologies [47]. His experiments investigated the relationship between the effect of each particle's personal experience (or conscience factor) $\mathbf{y}$, and the social exchange of information regarding a global best solution $\hat{\mathbf{y}}$. Particle neighborhoods were defined based on indices, i.e. each particle in the swarm was assigned an index and its visible neighborhood was determined accordingly. For example, if a particle i was only to consider its direct neighbors, it would consider the particles with indices $i-1$ and $i+1$, respectively. Particle indices do not assume positional similarity, i.e. particles i, $i-1$ and $i+1$ may occupy completely different positions in the search space. Among others, Kennedy tested the *cognition only* model. This model uses only a particle's personal best position in the velocity update. $\mathbf{v}_i$ is then updated as

$$v_{i,j}(t+1) = wv_{i,j}(t) + c_1 r_{1,j}(t)(y_{i,j}(t) - x_{i,j}(t)) \tag{2.20}$$

Clearly, there is no sharing of social information, and each particle effectively performs an individual search in its local area, based on its personal experience.

Suganthan tested a neighborhood operator that considers the spatial positions of particles based on inter-particle distances [85]. Particle i is considered to be in particle j's neighborhood when

$$\frac{\|\mathbf{x}_i - \mathbf{x}_j\|}{d_{max}} < \xi,$$

where d_{max} is the maximum inter-particle distance, $\xi = \frac{3t+0.6t_{max}}{t_{max}}$, and t_{max} is the maximum number of training iterations. Suganthan's approach initially favors small neighborhoods, promoting diversity. As the number of iterations increases, neighborhood size increases, until the model resembles the *gbest* algorithm when $t \to t_{max}$.

Kennedy proposed the social-stereotyping approach, a hybrid of index-based and topological neighborhoods [48]. The approach defines a number of clusters, using a

k-means clustering, on the personal best positions of all particles in a swarm. The number of clusters calculated is specified before the algorithm commences. Each cluster $\mathbb{C}$ therefore consists of a set of similar personal best positions. The centroid $\overline{\mathbb{C}}$ is defined as

$$\overline{\mathbb{C}} = |\mathbb{C}|^{-1} \sum_{\mathbf{y} \in \mathbb{C}} \mathbf{y}$$

with $|\mathbb{C}|$ the cardinality of $\mathbb{C}$. For a particle i belonging to $\mathbb{C}$, three new velocity update equations were defined, based on the *lbest* update in equation (2.11):

$$\begin{aligned} v_{i,j}(t+1) &= v_{i,j}(t) + c_1 r_{1,j}(t)(\overline{\mathbb{C}(i)}_j(t) - x_{i,j}(t)) \\ &\quad + c_2 r_{2,j}(t)(\hat{y}_j(t) - x_{i,j}(t)) \qquad (2.21) \\ v_{i,j}(t+1) &= v_{i,j}(t) + c_1 r_{1,j}(t)(y_{i,j}(t) - x_{i,j}(t)) \\ &\quad + c_2 r_{2,j}(t)(\overline{\mathbb{C}(g)}_j(t) - x_{i,j}(t)) \qquad (2.22) \\ v_{i,j}(t+1) &= v_{i,j}(t) + c_1 r_{1,j}(t)(\overline{\mathbb{C}(i)}_j(t) - x_{i,j}(t)) \\ &\quad + c_2 r_{2,j}(t)(\overline{\mathbb{C}(g)}_j(t) - x_{i,j}(t)) \qquad (2.23) \end{aligned}$$

The terms $\overline{\mathbb{C}(i)}$ and $\overline{\mathbb{C}(g)}$ respectively indicate the centroids of the clusters to which particles i and g belong, where $\mathbf{y}_g \in N_i$, and $f(\mathbf{y}_g) \leq f(\mathbf{y}_a)$, $\forall \mathbf{y}_a \in N_i$. N_i is defined as in equation (2.7). Clusters and centroids are recalculated at every iteration of the PSO algorithm. It therefore significantly increases the algorithm's computational complexity.

Kennedy and Mendes further investigated assumptions made about the *gbest* and *lbest* neighborhood architectures [51]. When considering neighborhood architectures from an information flow perspective, an architecture such as *lbest* with a neighborhood width of 1 very slowly propagates information about good solutions to other particles. On the other hand, the *gbest* algorithm immediately shares good solutions with all other particles. Kennedy and Mendes generated new architectures by varying the number of neighbors that define a particle's topological neighborhood and the 'amount of clustering'. Clustering occurs when the neighbors of a node are also neighbors of each other. It was found that the so-called *von Neumann* architecture had consistent, favorable performance. Where the *gbest*-neighborhood considers the complete swarm as a neighborhood, the von Neumann architecture can be visualized as a two-dimensional lattice, where each node's neighborhood consists of its direct neighbors: below, above and to the sides.

Self-Organized Criticality

Løvbjerg and Krink investigated the application of self-organized criticality (SOC) principles to diversity enhancement in the PSO [59]. Self-organized criticality describes a state transition in a complex system that occurs as a result of the behavior of a multitude of parameters affecting and controlling the system. No specific parameter can be identified as a controlling influence in the behavior of the system. In a physical substance such as water, a critical point occurs when it changes state from a solid to a liquid. The substance's temperature can be identified as the controlling parameter when analyzing the state-transition. When sand falls onto a surface, it forms a pile. As more sand slowly falls onto the pile, it may cause small avalanches that carries sand from the top of the pile to the bottom of the pile. In model systems the slope of the pile is independent of the rate at which sand falls. The slope is known as the critical slope.

The SOC PSO extends the definition of a particle i, to include a critical value C_i. C_i is also denoted as the criticality of the particle, and is initialized to $C_i = 0$. When particles occur in topologically similar positions in the search space, it can be said that the swarm is less diverse. The SOC PSO increases the criticality C_i of two particles when they are closer than a certain threshold value θ_{SOC} to each other. To keep criticality from building up, the C_i of a particle is reduced by a percentage value ρ_{SOC} during each iteration of the algorithm. When a particle i's C_i becomes larger than a threshold C_{MAX}, all the particles in its neighborhood's criticality values are incremented by one, and i is relocated to a different position in the search space. The particle i's criticality is re-initialized as $C_i = C_i - C_{MAX}$. Relocation of a particle effectively avoids the over-population of certain regions of the search space and promotes swarm diversity. A particle i may be relocated in one of the following ways:

- The position vector $\mathbf{x}_i$ is randomly re-initialized within the search space, and $\mathbf{y}_i = \mathbf{x}_i$.
- Particle i retains its personal best memory. Its current position vector is adapted in the direction indicated by its current velocity vector. A magnitude of C_{MAX} is added to C_i.

The traditional linearly decreasing inertia weight was also adapted to be a function of

C_i for particle i. The inertia weight of particle i, w_i, is set to

$$w_i = 0.2 + 0.1 \times C_i.$$

Particles in densely populated regions of the search space would therefore be more volatile than particles in sparsely populated regions as they would be more likely to influence each other.

Løvbjerg and Krink reported much improved results when comparing the SOC PSO to the *gbest* algorithm. Their relocation schemes do indeed help to improve swarm diversity, and the SOC inertia weight w_i leads to faster convergence.

PSO with Spatial Extension

Krink *et al* suggested the spatial extension particle swarm optimizer, SEPSO [55]. The spatial extension (SE) attempts to keep particles from clustering around potential solutions, thus improving swarm diversity and avoiding premature convergence. When a good solution is located by a swarm, particles tend to cluster around this solution, as dictated by the *gbest* velocity update equation. To avoid overcrowding on the $\hat{\mathbf{y}}$ position, a radius r is associated with each particle, that allows the algorithm to discern when two particles collide. Krink *et al* investigated three possible responses when particles collide:

1. Particles are bounced away from the collision in a random direction, retaining their original speed.

2. Particle interaction mimic realistic, physical collisions.

3. 'Velocity line bouncing', that maintains the direction of the velocity vector, but scales the speed of the particle.

The result of the above collision steps is that premature convergence is avoided. A much more diverse swarm is maintained, ensuring that more efficient exploration of the search space takes place.

Experimental results show that SEPSO perform markedly better than both a real-valued GA and the *gbest* algorithm [55].

2.8 Modifications to the PSO

A number of modifications to the PSO have been suggested that cannot be directly classified as a diversity improvement or convergence acceleration technique. This section presents a number of these techniques.

2.8.1 The Binary PSO

The basic PSO algorithm assumes a continuous, real-valued search space, where position vectors may take on any value within an allowed range. This representation is however not directly usable when optimizing a problem where the solution is represented as a bit string. Kennedy and Eberhart proposed a binary version of the PSO to cater for this need [50].

In the binary PSO, elements of position vectors may only take on values in the set $\{0, 1\}$. Velocity vectors remain real-valued. A particle position element $x_{i,j}$ is then updated using the following rule:

$$x_{i,j}(t+1) = \begin{cases} 0 & \text{if } r_{i,j}(t) \geq \sigma(v_{i,j}(t)) \\ 1 & \text{if } r_{i,j}(t) < \sigma(v_{i,j}(t)) \end{cases}$$

where $r_{i,j}(t) \sim U(0,1)$ and $\sigma(v_{i,j}(t))$ is defined as

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

The standard velocity update equation is used as in equation (2.9). Kennedy and Spears compared the binary PSO to a number of different GAs [52]. They found that the binary PSO performed exceptionally well when compared to GAs, and its performance did not suffer on high-dimensional problems.

2.8.2 Cooperative Swarms

Van den Bergh and Engelbrecht developed cooperative particle swarms to train neural networks [87, 88, 89]. In cooperative optimization, the problem representation vector $\mathbf{x}$ is segmented into sub-vectors, and each set of sub-vectors is optimized concurrently. For example, a 4-dimensional problem would be represented by a vector $\mathbf{x} = [x_1, x_2, x_3, x_4]$.

If $\mathbf{x}$ can then be segmented into two sub-vectors $\mathbf{x}' = [x_1, x_2]$ and $\mathbf{x}'' = [x_3, x_4]$ according to some problem specific rule, the sets of $\mathbf{x}'$ and $\mathbf{x}''$ vectors are optimized independently. This technique is very similar to the Cooperative Coevolutionary Genetic Algorithm (CCGA) by Potter [76]. Potter's technique optimizes each parameter in a n-dimensional problem independently, by segmenting the problem into n different populations. A complete representation of the objective vector is attained by selecting an element from each of the n populations.

Cooperative swarms have been applied to function optimization [87] as well as the training of neural networks (see section 2.9.1 on page 35).

2.8.3 Dynamic Systems

Several authors have investigated the behavior and applicability of particle swarms to dynamic, or changing environments.

Dynamic Goals

Carlisle and Dozier investigated the performance of PSO in a situation where the optimal goal position is a function that changes over time [7]. The goal position moves on a straight line over time, with variable velocity. To avoid stagnation on personal best positions $\mathbf{y}$ that may have been detected in earlier iterations of the algorithm, personal best positions are periodically re-initialized to each particle's position vector in the search space, $\mathbf{x}$. The update is referred to as *resetting* a particle.

They found that if changes in the optimal goal position are small, the swarm successfully detects these changes and updates the personal best position $\mathbf{y}_i$ accordingly. If changes in the goal position exceed the maximum velocity of the swarm, the swarm will never be able to 'keep up' with the goal position. They also found that if large changes in the goal function position trigger the resetting of particle positions, the swarm can more effectively track the changing goal. 'Large changes' were detected by evaluating the fitness of a random point in the search space. The fitness of the selected position changes proportionally to changes in the goal position.

In a further work, Carlisle and Dozier improved their original results by making the following changes [9]:

- A constriction factor approach was used for velocity vector updates, instead of an inertia weight.

- Instead of monitoring the fitness of a randomly selected position in the search space, the fitness of a particular particle is compared to that of the same position in the search space at an earlier iteration. A significant difference triggers a reset of particle positions.

The suggested changes showed to be a definite improvement.

Eberhart and Shi also tested PSO performance on dynamic goals [23]. A randomized inertia weight, of the form

$$w = 0.5 + 0.5 \times r(t)$$

where $r(t) \sim U(0.1)$, was used. Eberhart and Shi obtained improved results when comparing PSO performance to that of evolutionary strategies (ES) and evolutionary programs (EP). As pointed out in [87], it would seem that Carlisle and Dozier's resetting mechanism may not be necessary.

Hu and Eberhart introduced a 'changed-*gbest*-value' method to detect changes to the goal position [39]. This technique interprets changes to the fitness of $\hat{\mathbf{y}}$ as a change to the objective function f. If f changes, the current $\hat{\mathbf{y}}$ would no longer be optimal and needs to be re-evaluated. Hu and Eberhart also introduced the 'fixed-*gbest*-value' method [40]. This technique detects whether the best solution in a swarm $\hat{\mathbf{y}}$ stagnates on the same position over a number of training iterations. The algorithm's implementation also monitors a second global best solution $\hat{\mathbf{y}}'$. The second global best solution $\hat{\mathbf{y}}'$ has better fitness than all other particles, except the global best $\hat{\mathbf{y}}$. In summary, the algorithm can detect changes in the dynamic environment in the following ways:

- Re-evaluating $\hat{\mathbf{y}}$ will indicate a change in the quality of the candidate solution.

- Monitor $\hat{\mathbf{y}}$ and $\hat{\mathbf{y}}'$ for changes over a pre-determined number of iterations.

Once a change in the environment has been detected, the algorithm may respond in the following ways:

- No action may be taken.

- Randomize the position vectors of 10% of the particle population, while resetting the remaining particles.
- Randomly initialize $\hat{\mathbf{y}}$, and reset the remaining particles.
- Randomly initialize $\hat{\mathbf{y}}$.
- Reset all particles, while leaving $\hat{\mathbf{y}}$ unchanged.
- Randomize the position vectors of 50% of the particles, and reset the remaining particles.
- Randomize the position vectors of all particles.

The above techniques were evaluated on a single test function and needs to be further investigated before any general conclusions can be drawn on their effectiveness [40].

Noisy Functions

Parsopoulos and Vrahatis evaluated the performance of the PSO on noisy functions [73]. A noisy function can be simulated by redefining an objective function f on the parameter $\mathbf{x}$ to be:

$$f^{\sigma}(\mathbf{x}) = f(\mathbf{x})(1+\eta) \tag{2.24}$$

where $\eta \sim N(0,\sigma^2)$. Parsopoulos and Vrahatis tested objective functions by adding Gaussian noise, such as in equation (2.24), and by rotating the axes of the problem space through random angles. They found that noise did not seriously impair the PSOs ability to locate global optima, but as σ^2 becomes larger, performance becomes worse [73].

2.9 Applications of the PSO

This section describes a number of application areas where the PSO approach was used to solve specific problems. The form of the algorithm itself was not changed, only the basic definition of particle representation to suit the problem described. This section presents only a sample of the applications mentioned in available literature.

2.9.1 Training of Neural Networks

Supervised neural network training has been widely researched and several techniques such as gradient descent (GD) and scaled conjugent gradient (SCG) have been developed to train them [24, 67]. A neural network can be defined in the following way:

> *A neural network is basically a realization of a non-linear mapping from* $\mathbb{R}^I$ *to* $\mathbb{R}^K$,
>
> $$F_{NN} : \mathbb{R}^I \rightarrow \mathbb{R}^K$$
>
> *where* I *and* K *are respectively the dimension of the input and target (desired output) space. The function* F_{NN} *is usually a complex function of a set of nonlinear functions, one for each neuron in the network.*

Neural networks consist of layers of neurons. Each neuron consists of a transfer function that processes a number of inputs from a previous layer in a neural network, or external inputs. A neuron's output may serve as an input to a next layer. Successive layers in a network are connected through a set of weight values that effectively define the network. When a neural network learns to approximate a function or to classify a dataset, the network's set of weights are adapted to improve its performance. For more information on how GD and SCG learn weights, the interested reader is referred to [24]. Several authors have used the PSO algorithm to both train neural network weights, and learn characteristics of the transfer functions that define a network's neurons.

In the first paper on the PSO algorithm, Kennedy and Eberhart reported positive results when using PSO to train feed-forward networks [49]. They tested their neural networks on the XOR problem, as well as the well-known Fisher *iris* data (available from [5]). They informally remarked that networks trained with the PSO had slightly better generalization.

Eberhart and Hu used PSO to train a neural network in the medical environment [21]. The goal of the network was to distinguish between patients suffering from tremors. Tremors are a medical condition that describe uncontrollable limb movement. Apart from learning neural network weights, Eberhart and Hu used the PSO algorithm to learn the slopes of the sigmoidal transfer functions employed in the neural network's neurons. To

clarify, if the sigmoid function employed is given by

$$f_{\text{sig}}(x) = \frac{1}{1 + e^{-\lambda x}},$$

the parameter λ was learned with the network weights.

Van den Bergh used the cooperative PSO described in section 2.8.2 to train feed-forward neural networks with summation units [88] and product units [89], respectively. Ismail and Engelbrecht found that the PSO outperformed traditional neural network training techniques when training networks with product units [25, 44].

Mendes *et al* empirically studied the performance of customized particle neighborhoods when compared to a back-propagation neural network and evolutionary programming learning techniques [64]. Particle neighborhoods were considered as graph structures. The following configurations were considered:

- *Square*: Each particle's neighborhood consists of exactly four other particles.
- *Pyramid*: Particle neighborhoods are set up such that the visualized topology resembles a three-dimensional wire frame.
- *4Clusters*: Four subgraphs consisting of five particles each were formed. Each subgraph had two direct connections to its two closest neighbors and a single connection to the remaining subgraphs.

Mendes *et al* found that PSO based approaches performed better than back-propagation based methods for problems with multiple local minima. Specifically, the *Square* and *4Clusters* configurations outperformed all other techniques on classification problems.

Conradie *et al* recently presented the *Adaptive Neural Swarming* (ANS) approach [16]. ANS was developed to alleviate suboptimal performance of existing neurocontrollers in dynamic process environments. Conradie *et al* demonstrated that existing *linear* process controllers lack the ability to maintain an optimal *operating point*, due to the nonlinear nature of process environments. An operating point indicates a combination of parameter values, such as temperature and pressure, that may offer an economically acceptable performance level. Neural networks can be trained to mimic the function of neurocontrollers to maintain an operating point [18]. The ANS technique uses PSO

as a local optimization algorithm training a reinforcement neurocontroller architecture. Reinforcement neural learning uses information from a learned problem space to gauge how effective a learning process is. In the ANS PSO algorithm, each particle represents a candidate neurocontroller, which is a slightly altered replica of an existing controller. Conradie *et al* found that the utilization of PSO as optimization techniques yield much improved performance, to the extent that ANS offers a viable neurocontroller alternative [16].

2.9.2 Multi–Objective Optimization

Recently, particle swarms have been applied to multi–objective optimization (MOO) problems. Performing MOO with evolutionary algorithms is a vast research field and a complete discussion of the subject is beyond the scope of this thesis. This section presents a general formulation of MOO problems, and discusses the proposed PSO techniques to solve them.

Formulation of MOO problems

MOO techniques attempt to find a global optimum $\mathbf{x}^* = [x_1^*, x_2^*, \ldots, x_n^*]^T$ that will:

- Satisfy m inequality constraints:

$$g_i(\mathbf{x}^*) \geq 0 \quad i = 1, 2, \ldots, m$$

- Satisfy p equality constraints:

$$h_i(\mathbf{x}^*) = 0 \quad i = 1, 2, \ldots, p$$

- Optimize a k-dimensional vector function:

$$\bar{f}(\mathbf{x}^*) = [f_1(\mathbf{x}^*), f_2(\mathbf{x}^*), \ldots, f_k(\mathbf{x}^*)]^T$$

Different objective functions may have optimal solutions in locations that conflict with each other. The objective of MOO is to find a solution vector $\mathbf{x}^*$ that is acceptable in terms of $\bar{f}(\cdot)$ and represents optimal values for $\mathbf{x}^*$ [14].

The solution $\mathbf{x}^*$ is *Pareto optimal* if there exists no other solutions in the search space that would improve the quality of one of $\mathbf{x}^*$'s components, without a deterioration in any other component. To better understand this concept, let $\mathbf{x}_v = [x_{v,1}, x_{v,2}, \ldots, x_{v,n}]$ and $\mathbf{x}_u = [x_{u,1}, x_{u,2}, \ldots, x_{u,n}]$ be two vectors. Vector $\mathbf{x}_u$ *dominates* $\mathbf{x}_v$ if and only if

$$\begin{aligned} x_{u,i} &\leq x_{v,i} \quad \text{for } i = 1, \ldots, n, \text{ and} \\ x_{u,i} &< x_{v,i} \quad \text{for at least one } i. \end{aligned}$$

$\mathbf{x}^*$ is considered to be Pareto optimal if no other position in the search space dominates it.

The rest of this section presents PSO based approaches to solving multi–objective optimization problems.

MOPSO

The multiple objective particle swarm optimization (MOPSO) algorithm, introduced by Coello Coello and Lechuga [15], uses an approach motivated by the Pareto Archive Evolution Strategy (PAES) [54]. As described above, the goal of MOO techniques is to locate non-dominated solutions in the search space. MOPSO maintains a global repository of 'flight experience' where each particle is allowed to save located non-dominated solutions after each algorithm iteration. Before learning starts, MOPSO segments the problem space into a set of hypercubes, $\mathbf{H}$. The velocity update equation for a particle i is redefined to be

$$\begin{aligned} v_{i,j}(t+1) &= wv_{i,j}(t) + r_{1,j}(t)(y_{i,j}(t) - x_{i,j}(t)) + \\ &\quad r_{2,j}(t)(\mathbf{H}_{h,j}(t) - x_{i,j}(t)) \end{aligned}$$

where $\mathbf{H}_{h,j}$ represents the j^{th} dimension of a randomly selected particle in hypercube $\mathbf{H}_h$. The position repository maintains a set of favorable positions within the search space. The position repository is re-evaluated after each iteration of the algorithm. Dominated solutions in the repository may be replaced by better, *non-dominated* solutions. A limit is placed on the size of the position repository, so a heuristic is put in place to determine which particle position will have preference over others. The technique is simple: A position that exists in an area of the search space that is less populated will be given

preference over a solution occupying a position in a more densely populated region. This scheme has the added advantage of promoting swarm diversity. MOPSO performed similar to existing GA-based MOO techniques [15].

Weighted Aggregation and Populations

The *weighted aggregation* approach is a simple technique to deal with MO problems. The technique uses a linear combination of objective functions to formulate an objective function vector describing the problem. More formally, the objective function is defined as

$$\bar{f}(\mathbf{x}) = \sum_{i=1}^{k} w_i f_i(\mathbf{x})$$

Although it is not required, it is usually assumed that $\sum_{i=1}^{k} w_i = 1$.

Three different weight aggregation techniques to solve MO problems were implemented by Parsopoulos and Vrahatis using the PSO [75]:

Conventional Weighted Aggregation (CWA): CWA uses a set of fixed weights. The technique can only locate a single Pareto Optimum per algorithm run [46].

Bang-Bang Weighted Aggregation (BWA): BWA oscillates the weights associated with each of the objective functions. For a two objective function, weights are calculated as

$$w_1(t) = \text{sign}(\sin(2\pi t/\vartheta)), \quad w_2(t) = 1.0 - w_1(t)$$

The symbol t represents the index of the current iteration, and ϑ a weight change frequency.

Dynamic Weighted Aggregation (DWA): To introduce a more gradual change to weight values, the DWA approach defines, for a two objective problem, weights as

$$w_1(t) = |\sin(2\pi t/\vartheta)|, \quad w_2(t) = 1.0 - w_1(t).$$

This update forces movement on the Pareto front.

Parsopoulos and Vrahatis also introduced a technique based on the VEGA algorithm [79] that was discussed in section 2.7.2.

Dynamic Neighborhoods

Hu and Eberhart introduced three new techniques to solve multi–objective problems with the PSO approach [38]:

- A single objective is optimized while other objectives are kept constant. The technique was labelled as *one-dimensional* optimization by its authors. For example, in a two objectives problem, with objective functions f_1 and f_2, f_2 may be kept constant while optimizing f_1. Hu and Eberhart suggests keeping a 'simple' objective function constant, while optimizing a 'difficult' function. This approach is an unmotivated heuristic, and may not be generally applicable.

- *Dynamic particle neighborhoods* are used to locate multiple Pareto fronts. The technique is similar to the dynamic neighborhoods utilized in chapter 4, but was developed independently. At each velocity update step, the neighborhood term $\hat{\mathbf{y}}_i$ in a *lbest* velocity update for a particle i is replaced with a neighborhood position calculated as the best position among m neighbors. Neighbors are determined based on their proximity to i. Hu and Eberhart used $m = 2$ in their experiments.

- A particle's personal best position $\mathbf{y}$ is only updated if the improved position considered dominates the existing personal best, i.e. for particle i

$$\mathbf{y}_i(t+1) = \begin{cases} \mathbf{x}_i(t+1) & \text{if } \mathbf{x}_i(t+1) \text{ dominates } \mathbf{y}_i(t) \\ \mathbf{y}_i(t) & \text{otherwise.} \end{cases}$$

  This update limitation is similar to the hill-climbing update step taken by the multi-phase PSO [1].

Through visual inspection (i.e. plotting particle positions), Hu and Eberhart found that the above techniques extend the capabilities of the PSO to solving multi–objective optimization problems [38].

2.10 Conclusion

In this section, several different evolutionary computing and swarm intelligence techniques were discussed. Each of these techniques were originally based on the observed

behavior of some element of nature. Evolutionary computing is based on the process of evolution, and attempts to mimic it. Swarm intelligence techniques exploit the emergent collective intelligence of swarms of seemingly unintelligent particles, such as is present in the behavior of a flock of birds flying in formation.

In particular, section 2.4 reviewed simple genetic algorithms, and section 2.5 presented particle swarm optimizers. Numerous extensions and improvements to the PSO were presented in sections 2.7 and 2.8. Most notably, the GCPSO, which plays an important role in the NichePSO algorithm, was introduced in section 2.7.1. The chapter was concluded by a discussion of some of the applications of the PSO. Numerous other applications, where the PSO can be applied without any modifications exist, such as:

- Optimization of hydraulic equations [56].
- Min-max optimization [58, 83].
- Integer programming [57].
- Memetic neuro-evolution, for nonlinear process controllers [17].

The above list is by no means exhaustive. In the next chapter, niching techniques are reviewed, followed by modifications to the standard PSO to enable it to perform niching.

Chapter 3

Niching Techniques

Niching techniques maintain multiple solutions in multimodal domains, in contrast to existing evolutionary and swarm intelligence optimization techniques that have been designed to only locate single solutions. This chapter introduces well-known GA-based niching techniques. The applicability of GA techniques to PSO is considered, and a number of niching applications are presented.

3.1 Introduction

Particle swarm optimizers have proven to be useful in locating optimal solutions to optimization problems. This fact is supported by almost all papers published in this research field. The PSO technique's effectiveness can be attributed to its efficient propagation of information regarding a single, *global* solution. The original update equations for velocity and position vectors were designed to lead all particles to the same solution [49].

When attempting to optimize multimodal functions, the situation is changed. In a multimodal function, multiple positions in the function's search space may have optimal fitness. The global best solution found by a swarm of particles when using the *gbest* or *lbest* algorithms may not necessarily be the *only* possible solution. Any particle that occupies a position close to a potential global solution, or even an acceptable local optimum, that is not close to the swarm's global best in the search space, will be forced to move towards the global best position $\hat{\mathbf{y}}$. Alternatively, equally acceptable solutions

that happened not to be close to the swarm's current best at any given time, are ignored in favor of a limited collective swarm consciousness. Consequently, portions of the search space are effectively ignored in favor of a potentially limited view of the search space.

A number of evolutionary techniques have been suggested to locate multiple solutions to multimodal problems. These algorithms have been almost exclusively explored using GAs. In GA parlance, optimization techniques that locate multiple optima to multimodal function optimization problems are known as *niching*, or *speciation* techniques.

For niching, both GAs and PSOs use a population of individuals that are partitioned in some way to focus and locate different possible solutions in a single search space (note that the term individual here applies both to individuals in GAs and particles in the PSO algorithm). Each subgroup in a partitioned population or swarm, is known as a *species*. The behavioral patterns of individuals competing for the use of a resource in a subgroup and between elements in a subgroup, is known as *speciation*.

This chapter explores the evolutionary theory behind niching and speciation. Section 3.2 presents a theoretical base for niching, followed by a number of existing GA-based niching techniques in section 3.3. Section 3.4 analyses a single known PSO attempt at niching. The simple application of GA niching techniques to PSOs is discussed in section 3.5, and the chapter is concluded with a number of niching applications in section 3.6.

3.2 What is Niching?

In an environment where a large number of individuals compete for the use of available resources, behavioral patterns emerge where individuals are organized into subgroups based on their resource requirements. Horn defines niching as a

> *"form of cooperation around finite, limited resources, resulting in the lack of competition* between *such areas, and causing the formation of* species *for each niche"* [37].

Niches are thus partitions of an environment, and species are partitions of a population competing within the environment. Localization of competition is introduced by simply *sharing* resources among individuals competing for it. The terms *niche* and *species* can

be used interchangeably. As an example, a school of fish that live in a certain part of the ocean compete with each other for access to a potentially limited food supply. Food may not be available everywhere in their environment. Certain fish may learn to live in a small area around a food source, while others may learn to roam their environment and only feed when they require nourishment. If there was to be a single food source, it is a reasonable expectation that all the fish would eventually exhibit similar behavior. They would all be required to find food in the same place, and encounter the same resistance from other fish.

The social interaction and adaptation of individuals competing in an environment around multiple resources form the basis for the study of niching techniques with evolutionary optimization algorithms. In the evolutionary optimization context, Horn defines *implicit niching* as the sharing of resources, and *explicit niching* as the sharing of fitness [37].

Niching methods can be categorized as either being *sequential* or *parallel*.

Sequential niching (or temporal niching) develops niches sequentially over time. The approach can be summarized as searching for a possible solution until it is found, then removing all references to it in the search space and repeating the search until convergence criteria are met. Because of the removal of 'confusing' optima, a technique that assumes a single global solution may be used.

Parallel niching forms and maintains several different niches simultaneously. The search space is not modified. *Parallel niching* techniques thus not only depend on finding a good measure to locate possible solutions, but also need to organize individuals in a way that maintains their organization in the search space over time to populate locations around solutions.

Regardless of the way in which niches are located (i.e. in parallel or sequentially), the distribution of individuals can be formalized in a number of ways, according to their *speciation* behavior [61]:

Sympatric speciation occurs when individuals form species that coexist in the same search space, but evolve to exploit different resources (or more formally, different ecological niches). For example, different kinds of fish feed of different food sources

in the same environment. Cannibalism is explicitly excluded here, although it may be an interesting measure to consider. It may act as a deterrent in overpopulated niches.

Allopatric speciation differentiates individuals based on spatial isolation in a search space. No interspecies communication takes place, and subspecies can develop only through deviation from the available 'genetic' information. Such an event could be triggered by mutation. Here, different fish species would effectively live and play around their food sources, and not be concerned with other species living in different areas.

Parapatric speciation allows new species to form as a result of segregated species sharing a common border. Communication between the initial species may not have been encouraged or intended. As an example, new fish species may evolve based on the interaction of a small percentage of different schools of fish. The new species may have different food requirements and may eventually upset the environment's stability.

The PSO nichers presented in this thesis can be classified as using an allopatric speciation approach. Allopatric speciation will therefore be a more prevalent issue of discussion, as it defines the goals of multimodal function optimization.

The next section discusses existing niching techniques that have been introduced in the genetic algorithm and particle swarm optimization fields.

3.3 Genetic Algorithm based Niching Techniques

This section presents a number of well known niching algorithms, originally studied using GAs. GA niching is based on research originally done to maintain diverse populations. Unless indicated, each of the techniques described next assumes that a normal generational evolutionary optimization process takes place, as discussed in section 2.4.

3.3.1 Fitness Sharing

Fitness sharing is one of the earliest GA niching techniques. It was originally introduced as a population diversity maintenance technique [32]. It is a parallel, explicit niching approach. The algorithm regards each niche as a finite resource, and shares this resource among all individuals in the niche. Individuals are encouraged to populate a particular area of the search space by adapting their fitness based on the number of other individuals that populate the same area. The fitness f_i of individual u is adapted to its shared fitness:

$$f_i' = \frac{f_i}{\sum_j sh(d_{u,v})} \tag{3.1}$$

A common sharing function is:

$$sh(d) = \begin{cases} 1 - (d/\sigma_{share})^\alpha & \text{if } d < \sigma_{share} \\ 0 & \text{otherwise.} \end{cases}$$

The symbol $d_{u,v}$ represents a distance calculated between individual u and individual v. The distance measure implemented can be genotypic or phenotypic, depending on the optimization problem at hand. If the sharing function finds that $d_{u,v}$ is less than σ_{share}, it returns a value in the range $[0, 1]$ that increases as $d_{u,v}$ decreases. Therefore, the more similar u and v, the lower their individual fitnesses will become. The fitness sharing approach assumes that the number of niches can be estimated, i.e. the approximate number of niches must be known prior to the application of the algorithm. It is also assumed that niches occur at least a minimum distance $2\sigma_{share}$ from each other. The above assumptions cannot always be made: The number of niches will not always be estimatable, nor would the distance between them.

3.3.2 Dynamic Niche Sharing

Miller and Shaw introduced dynamic niche sharing as a computationally less expensive version of fitness sharing [66]. The same assumptions are made as with fitness sharing:

- Niches must occur at a minimum distance of $2\sigma_{share}$ from each other,
- and the number of optima must be known.

During the evolution of a population with the dynamic niche sharing technique, individuals will invariably start to form subspecies and populate niches. Dynamic niche sharing attempts to *classify* individuals in a population as belonging to one of the emerging niches, or to a non-niche category. Fitness calculation for individuals belonging to the non-niche category is done with the same equation that is used in the original fitness sharing technique, namely equation (3.1), in section 3.3.1. The fitness of individuals found to belong to one of the developing niches is diluted by dividing it by the size of the developing niche. Dynamically finding niches is a simple process of iterating through the population of individuals and constructing a set of non-overlapping areas in the search space. Dynamic sharing is computationally less expensive than 'normal' sharing. Miller and Shaw presented results showing that dynamic sharing has improved performance when compared to fitness sharing [66].

3.3.3 Sequential Niching

Sequential niching (SN) is a simple algorithm introduced by Beasley *et al* [4]. SN identifies multiple solutions by adapting an optimization problem's objective function's fitness landscape through the application of a *derating* function at a position where a potential solution was found [4]. A derating function is designed to lower the fitness appeal of previously located solutions. By repeated applications of the algorithm to an objective function's fitness landscape, all confusing local and global optima are removed. Sample derating functions, for a point $\mathbf{x}$ and a previous maximum $\mathbf{x}^*$ include:

$$G_1(\mathbf{x}, \mathbf{x}^*) = \begin{cases} \left(\frac{\|\mathbf{x}-\mathbf{x}^*\|}{r}\right)^{\alpha} & \text{if } \|\mathbf{x}-\mathbf{x}^*\| < r \\ 1 & \text{otherwise} \end{cases}$$

and

$$G_2(\mathbf{x}, \mathbf{x}^*) = \begin{cases} e^{\log m \frac{r-\|\mathbf{x}-\mathbf{x}^*\|}{r}} & \text{if } \|\mathbf{x}-\mathbf{x}^*\| < r \\ 1 & \text{otherwise} \end{cases}$$

where r is the radius of the derating function's effect. In G_1, α determines whether the derating function is concave ($\alpha > 1$) or convex ($\alpha < 1$). For $\alpha = 1$, G_1 is a linear function. For G_2, m determines 'concavity'. Note that $\lim_{x \to 0} \log(x) = -\infty$, hence m must always be great than 0. Smaller values for m result in a more concave derating

function. The fitness function $f(\mathbf{x})$ is then redefined to be

$$M_{t+1}(\mathbf{x}) \equiv M_t(\mathbf{x}) \times G(\mathbf{x}, \mathbf{s}_t)$$

where $M_o(\mathbf{x}) \equiv f(\mathbf{x})$ and $\mathbf{s}_t$ is the best individual found during run t of the algorithm. G can be any derating function, such as G_1 and G_2.

3.3.4 Crowding

Crowding, or the crowding factor model, as introduced by De Jong [19], was originally devised as a diversity preservation technique. Crowding was inspired by a naturally occurring phenomenon in ecologies, namely competition amongst similar individuals for limited resources. Similar individuals compete to occupy the same ecological niche, while dissimilar individuals do not compete, as they do not occupy the same ecological niche. When a niche has reached its *carrying capacity* (i.e. being occupied by the maximum number of individuals that can exist within it) older individuals are replaced by newer (younger) individuals. The carrying capacity of the niche does not change, so population size remains constant.

For a genetic algorithm, crowding is performed as follows: It is assumed that a population of GA individuals evolve over several generational steps. At each step, the crowding algorithm selects only a portion of the current generation to reproduce. The selection strategy is fitness proportionate, i.e. more fit individuals are more likely to be chosen. After the selected individuals have reproduced, individuals in the current population are replaced by their offspring. For each offspring, a random sample is taken from the current generation, and the most similar individual is replaced by the offspring individual. To cover a complete search space, the initial position of individuals should be well distributed, as the algorithm is unlikely to evaluate any part of the search space that is not within the first generation.

3.3.5 Deterministic Crowding

Deterministic crowding (DC) is based on De Jong's crowding technique (see section 3.3.4), but uses the following improvements as suggested by Mahfoud [61]:

- Mahfoud found that phenotypic similarity metrics (such as Euclidean distance) were preferred to similarity metrics based on genotypes (e.g. Hamming distance). Phenotypic metrics embody domain specific knowledge that is most useful in multimodal optimization, as several different spatial positions can contain equally optimal solutions. Not only is the quality of potential solutions important, but also their proximity to each other.

- It was shown that there exists a high probability that the most similar individuals to offspring are their parents. The replacement strategy initially proposed by De Jong was changed to compare an offspring only to its parents and not to a random sampling of the population.

- De Jong's crowding used a traditional proportional method. Individuals are selected for reproduction based on their fitness. Mahfoud suggested selecting individuals randomly, and only replacing parents with their offspring if the offspring performs better.

Since the DC algorithm is used in later chapters, it is presented in figure (3.1) (algorithm pseudo-code taken from [61], symbols as defined in table 2.1). Note the $d(\cdot)$ is a phenotypic distance function.

Probabilistic crowding, described as an *"offspring of deterministic crowding,"* was introduced by Mengshoel *et al* [65]. It is based on Mahfoud's deterministic crowding, but employs a probabilistic replacement strategy.

Where the original crowding and DC techniques replaced an individual u with v if v was more fit than u, probabilistic crowding uses the following rule: If individuals u and v are competing against each other, the probability of u winning and replacing v is given by

$$p_u = \frac{f_u}{f_u + f_v}$$

where f_u is the fitness of individual u. The core of the algorithm is therefore to use a probabilistic tournament replacement strategy. Experimental results have shown it to be both fast and effective.

Repeat for g generations:

1. Do $n/2$ times:

   (a) Select 2 parents, p_1 and p_2, randomly from C_g.

   (b) Cross p_1 and p_2, yielding c_1 and c_2.

   (c) Apply mutation/other operators, yielding $c_1^{'}$ and $c_2^{'}$.

   (d) IF $[d(p_1, c_1^{'}) + d(p_2, c_2^{'})] \leq [d(p_1, c_2^{'}) + d(p_2, c_1^{'})]$

   - If $f(c_1^{'}) > f(p_1)$ replace p_1 with $c_1^{'}$
   - If $f(c_2^{'}) > f(p_2)$ replace p_2 with $c_2^{'}$

   ELSE

   - If $f(c_2^{'}) > f(p_1)$ replace p_1 with $c_2^{'}$
   - If $f(c_1^{'}) > f(p_2)$ replace p_2 with $c_1^{'}$

Figure 3.1: Deterministic Crowding Algorithm

3.3.6 Restricted Tournament Selection

Restricted Tournament Selection (RTS), introduced by Harik [35], is similar to DC, but promotes local competition.

In RTS, the selection process is adapted in the following way:

Two individuals, u and v are randomly selected from the pool of individuals in the current generation.

Crossover and mutation operators are performed, yielding two new individuals, u^* and v^*.

The remainder of the current population is searched for individuals that are the most similar to u^* and v^*, and when found, are designated by s_u and s_v.

u^* then competes against s_u for a position in the next generation. The same happens with v^* and s_v.

Harik presented results in [35] proving that RTS successfully locates solutions to multi-modal problems.

3.3.7 Coevolutionary Shared Niching

Goldberg and Wang introduced coevolutionary shared niching (CSN) [33]. CSN locates niches by co-evolving two different populations of individuals in the same search space, in parallel. Let the two parallel populations be designated by A and B, respectively. Population A can be thought of as a normal population of candidate solutions, and it evolves as a normal population of individuals. Individuals in population B are scattered throughout the search space. Each individual in population A associates with itself a member of B that lies the closest to it using a genotypic metric. The fitness calculation of the i^{th} individual in population A, A_i, is then adapted to $f'(A_i) = \frac{f(A_i)}{|B_b|}$, where $f(\cdot)$ is the fitness function; $|B_b|$ designates the cardinality of the set of individuals associated with individual B_b and b is the index of the closest individual in population B to individual i in population A. The fitness of individuals in population B is simply the average fitness of all the individuals associated with it in population A, multiplied by B_b. Goldberg and Wang also developed the *imprint CSN* technique, that allows for the transfer of good performing individuals from the A to the B population.

CSN overcomes the limitation imposed by fixed inter-niche distances assumed in the original fitness sharing algorithm [32] and its derivate, dynamic fitness sharing [66]. The concept of a niche radius is replaced by the association made between individuals from the different populations.

3.3.8 Dynamic Niche Clustering

Dynamic Niche Clustering (DNC) is a fitness sharing based, cluster driven niching technique [28, 29]. It is distinguished from all other niching techniques by the fact that it supports 'fuzzy' clusters, i.e. clusters may overlap. This property allows the algorithm to distinguish between different peaks in a multimodal function that may lie extremely close together. In most other niching techniques, a more general inter-niche radius (such as the σ_{share} parameter in fitness sharing) would prohibit this.

The algorithm works by constructing a *nicheset*, which is a list of niches in a population. The nicheset persists over multiple generations. Initially, each individual in a population is regarded to be in its own niche. Similar niches are identified using Euclidean distance and merged. The population of individuals is then evolved over a pre-determined number of generational steps. Before selection takes place, the following process occurs:

- The midpoint of each niche in the nicheset is updated, using the formula

$$\overline{mid_u} = \overline{mid_u} + \frac{\sum_{i=1}^{n_u}(\mathbf{x}_i - \overline{mid_u}) \cdot f_i}{\sum_{i=1}^{n_u} f_i}$$

  where $\overline{mid_u}$ is the midpoint of niche u, initially set to be equal to the position of the individual from which it was constructed, as described above. n_u is the niche count, or the number of individuals in the niche, f_i is the fitness of individual i in niche u and $\mathbf{x}_i$ is the location of individual i.

- A list of inter-niche distances is calculated and sorted. Niches are then merged using a technique described in [29].

- Similar niches are merged. Each niche is associated with a minimum and maximum niche radius. If the midpoints of two niches lie within the minimum radii of each other, they are merged.

- If any niche has a population size greater than 10% of the total population, random checks are done on the niche population to ensure that all individuals are focusing on the same optima. If this is not the case, such a niche may be split into sub-niches, which will be optimized individually in further generational steps.

Using the above technique, Gan and Warwick also suggested a *niche linkage* extension to model niches of arbitrary shape [30].

3.4 PSO Niching Techniques

While research on GA niching techniques is abundant, niching with PSOs have thus far received little research attention. This section overviews a niching variation of the

objective function stretching optimization technique, as discussed in section 3.4. This approach is, to the author's knowledge, the only existing approach applicable to PSO niching.

Objective Function Stretching for Locating Multiple Global Minima

Objective function stretching, introduced in section 2.7.1, was applied as a sequential niching technique by Parsopoulos and Vrahatis [72]. The technique is partially repeated here for completeness and discussed in more detail to clarify its niching ability.

The stretching technique adapts the landscape of an optimization problem's fitness function to remove local minima. When a local solution is detected during the evolutionary learning process, the *stretching* operator is applied to remove the detected solution from the fitness landscape. Subsequent iterations of the PSO algorithm can then focus on locating solutions in other parts of the search space, assured that the detected local optima will not again lead to premature convergence.

The niching variation of the stretching technique detects potential solutions by comparing candidate solutions to a threshold value ϵ. Parsopoulos and Vrahatis suggest values such as 0.01 and 0.001 [72]. (Note that it was implicitly assumed that optimization problems were of low dimension. Their test results were only given on one and two-dimensional problems. Higher dimensional problems will most likely require larger values.) When a potential solution $\mathbf{x}^*$ is detected with this technique, the particle at $\mathbf{x}^*$ is *isolated* from the rest of the swarm. The stretching operator is then applied at $\mathbf{x}^*$, marking this position and its vicinity as undesirable by increasing the fitness. The "application of the stretching functions" means that the fitness calculation of remaining particles are adapted. If the position vector of the detected solution is given by $\mathbf{x}^*$, the fitness calculation, $f(\mathbf{x})$, for all remaining particle positions $\mathbf{x}$, is redefined to be $H(\mathbf{x})$, where:

$$H(\mathbf{x}) = G(\mathbf{x}) + \gamma_2 \frac{\text{sign}(f(\mathbf{x}) - f(\mathbf{x}^*)) + 1)}{2\tanh(\mu(G(\mathbf{x}) - G(\mathbf{x}*)))} \tag{3.2}$$

and

$$G(\mathbf{x}) = f(\mathbf{x}) + \gamma_1 \frac{\|\mathbf{x} - \mathbf{x}^*\|(\text{sign}(f(\mathbf{x}) - f(\mathbf{x}^*)) + 1)}{2} \tag{3.3}$$

The interpretation of the sign function is the same as in section 2.7.1. The transformation represented by $G(\mathbf{x})$ in equation (3.3) removes all local minima located above the

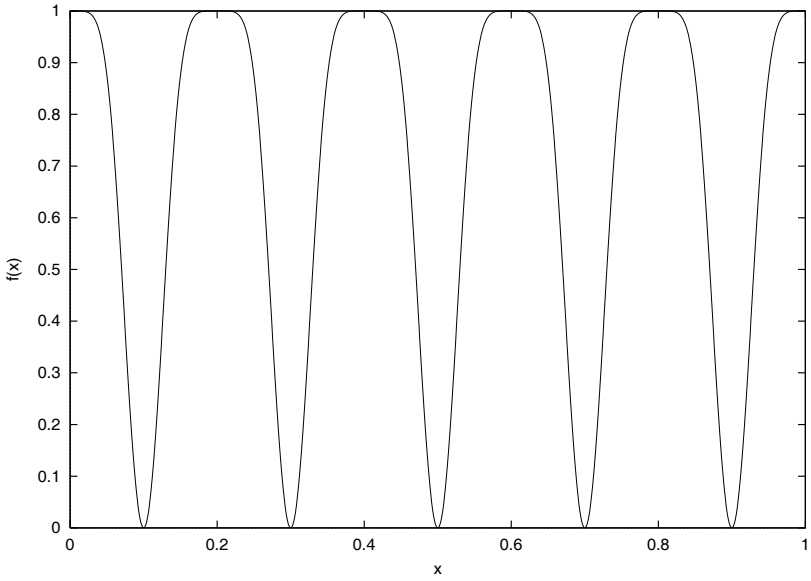


Figure 3.2: Function $F_s(x) = 1.0 - \sin^6(5\pi x)$ with 5 minima of equal fitness.

detected solution $\mathbf{x}^*$. The transformation in equation (3.2) assigns higher objective function values to positions close to $\mathbf{x}^*$. The objective function landscape below $\mathbf{x}^*$ remains unchanged. Local optima with worse fitness than at position $\mathbf{x}^*$ is thus removed from the search space.

Van den Bergh investigated the efficacy of objective function stretching as global optimization technique. It was found that the technique may alter the search space by introducing false minima [87]. This observation in part warrants a more thorough investigation of the applicability of stretching as a niching technique. The alteration of the search space by the stretching operator for niching purposes is discussed next.

In this thesis the objective function stretching technique was applied to locate all the minima of the function

$$F_s(x) = 1.0 - \sin^6(5\pi x) \tag{3.4}$$

in the domain $x \in [0, 1]$ (see figure (3.2)). The objective function f is defined to be $f(x) = F_s(x)$. A PSO was trained on the objective function f, with parameter settings $\gamma_1 = 10^4$, $\gamma_2 = 1.0$, $\mu = 10^{-10}$ and $\epsilon = 10^{-5}$. When a minimum was detected at a particle position x when $f(x) < \epsilon$, the particle was isolated and the fitness function redefined to $f(x) = H(x)$. When $x \approx 0.9$, the fitness function landscape was altered, as is shown in

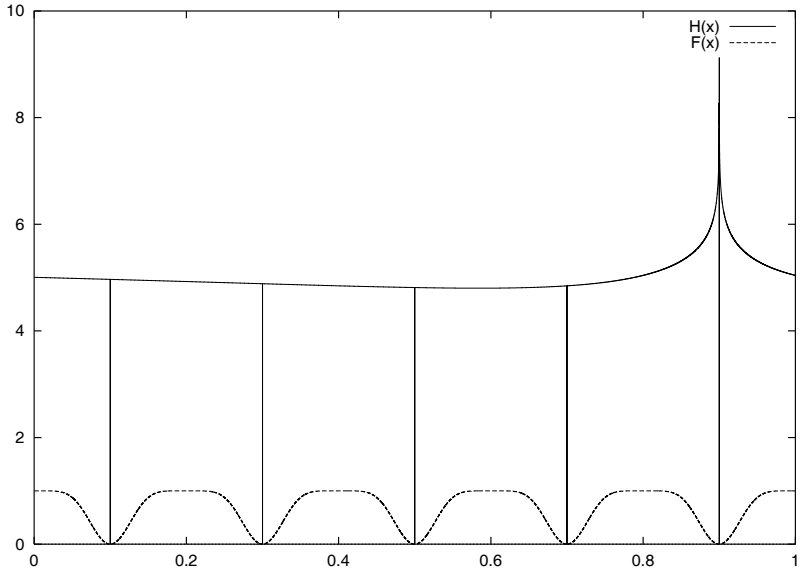


Figure 3.3: The modified objective function landscape, after the stretching functions were applied.

figure (3.3). The effect of the respective transformation functions, $H(x)$ and $G(x)$ can be pertinently seen. The effect of $G(x)$, i.e. removing all minima above the position indicated by x, clearly lifts out the positions of the remaining minima. $H(x)$ ensures that the fitness landscape around the potential solution x is marked as undesirable. The following problems are however introduced:

- If multiple acceptable minima are located close to each other, the effect of $G(x)$ may cause these alternative solutions never to be detected. The steep fitness function slope, regardless of the 'trenches' on remaining minima (see below), will keep particles from traversing towards this area of the search space. Also, if x is removed from the swarm, $\hat{\mathbf{y}}$ will be redefined to a different location that would be likely to discourage movement towards the position of x.

- The adaptation of $f(x)$ to remove all potential minima above x, introduces 'trenches' in the fitness function around remaining solutions. Although this transformation makes detecting these remaining minima visually simple, the optimization process is less likely to detect them. If $\epsilon = 10^{-5}$, as was given above, a solution

will be detected around $x \approx 0.9$ when any particle has a position in the range $[0.89999, 0.90001]$. The width of the 'trench' around 0.9 is then 0.00002. The probability of the evolutionary search process locating this position can then be calculated by dividing the width of the trench by the total width of the search space. This yields an extremely small probability of locating the minima under consideration and other minima.

- The transformation of $f(x)$ with the given values for γ_1, γ_2 and μ, introduced a new local minimum at $x \approx 0.6$. Because of the small likelihood of locating the actual minima, the optimization process is more likely to regard this position as a minimum. Tuning parameters γ_1, γ_2 and μ may remove the introduced minimum.

The above issues question the usability of objective function stretching as an effective niching technique. The results reported by Parsopoulos and Vrahatis in [72] could not be replicated.

3.5 Application of GA Niching Techniques to PSO

Given the wealth of GA based niching techniques, a natural step would be to consider the adaptation of these techniques to particle swarm optimizers. This section discusses this possibility.

By default, the PSO algorithm uses a phenotypic similarity metric in the form of a fitness function. In unimodal optimization, this approach is acceptable, since the assumption can be made that when two particles exhibit similar fitness, they are definitely approaching the same solution. The particles will also occupy similar positions in the search space. In multimodal optimization, the fitness function is still crucial in the assessment of the *quality* of solutions found by particles, but it is not capable of giving an indication of particle *similarity*, based on phenotypic behavior. Niches with similar fitness may occur in different positions in the search space. A metric such as Euclidean distance can give an acceptable indication of particle similarity, but it is not capable of doing this while considering particle quality.

GA inspired crowding techniques promote the formation of niches by maintaining sets of similar individuals. Similarity between individuals in different generations is a result

of the replacement strategy used. By maintaining individuals that are similar, multiple niche locations can be identified and maintained over several generations. Maintaining particles around a potential solution in the PSO algorithm is done quite differently. Particles are not linked over different generations or iterations of the algorithm in the same way. With PSOs, this explicit maintenance of locations around a potential solution is largely taken over by the cognition memory of its personal best solution. The PSO algorithm's rapid global search nature is achieved by the propagation of knowledge regarding global good solutions and each particle's ability to remember its personal best solution. Sharing information about a single solution focuses all search efforts on the best solution found by the swarm at any given time during the optimization process.

Removing the global best information yields particles that perform a local search, biased by the best solution found by each. This type of particle is somewhat similar to individuals in GA populations. All GA techniques that depend on the independent nature of individuals, as described above, cannot simply be applied to PSOs. Attempting to remove all references to global information effectively truncates the value of the PSO search to that of a random search evolutionary program.

Because of this fact, two new niching techniques, suited specifically to the nature of the PSO, are presented in the next chapters.

3.6 Application of Niching Techniques to Real-World Problems

Several areas, where the location of multiple solutions in a search space are beneficial, can be identified. This section gives a short overview of a number of well-known techniques.

Carroll investigated the application of a multitude of different GA techniques to the optimization of chemical oxygen-iodine lasers [10]. Chemical lasers are produced through a series of chemical reactions between gases. In particular, Carroll compared Goldberg's sharing technique [32] with a non-sharing GA. He found that sharing helped to more rapidly find an optimal power input. The interested reader is referred to [10] for a more in-depth treatment of this particular application.

Hwang and Cho successfully applied the fitness sharing technique to evolve diverse

circuit architectures [43]. Fitness sharing allowed them to design an embedded device that dynamically reconfigures its circuit architecture when necessary. The system can generate multiple architectures concurrently.

Kim and Cho used deterministic crowding (see section 3.3.5) to evolve a checkers player [53]. Their system evolved a neural network to play the game. They found that by selecting multiple neural networks from different optimal solutions found by the different niches identified by DC, game-play was improved.

To evaluate the effectiveness of the proposed PSO based niching techniques, multimodal function optimization problems are considered, as well as solving systems of unconstrained equations with multiple solutions.

3.7 Conclusion

Niching is an optimization technique inspired by natural evolution. Niching algorithms allow traditionally unimodal optimization techniques, such as GAs and PSOs, to be extended to locate multiple solutions in a search space. This chapter reviewed well-known GA-based niching techniques, as well as more recent attempts, including a PSO based algorithm. The possibility of extending GA niching techniques to PSOs was investigated. It was found that the techniques inspired by the generational model of GAs are not easily extended to the particle swarm model. The PSO model is based on a set of different assumptions and particles in a swarm are more free to traverse the search space, than individuals in a GA population. Finally, a number of existing real-world niching applications were presented.

In the next chapters, niching algorithms, designed to make use of the specifics of the PSO model, are presented.

Chapter 4

The *nbest* Particle Swarm Optimizer

A new PSO-based algorithm, *nbest*, is developed in this chapter, specifically to solve systems of unconstrained equations. It represents a first attempt at developing a PSO based nicher. The standard *gbest* PSO is adapted by redefining the fitness function in order to locate multiple solutions in one run of the algorithm. The *nbest* algorithm also introduces the concept of shrinking particle neighborhoods. Results are presented that show the new *nbest* PSO algorithm to be a promising niching algorithm.

4.1 Introduction

Many problems in science and engineering (e.g. robotics and signal processing) require solving systems of *linear* equations. When solving systems of equations (SEs) off-line with numerical methods, the goal is to find an optimal solution, without considering a time constraint. However, if such a problem needs to be solved in real-time (e.g. in a dynamic process controller) under time constraints, existing numerical methods may not scale well. Efficient numerical techniques have been developed to solve SEs, but they are not universally applicable [78]. Some systems, consisting of large numbers of equations and unknowns, can only be solved approximately by utilizing heuristic methods.

The concept of a SEs is formalized and a number of traditional algebraic approaches to solving them are discussed in section 4.1.1. Section 4.1.2 describes neural network based approaches to solving SEs, and section 4.2 investigates the necessary representation

of SEs for solving them with the PSO algorithm. Section 4.3 introduces the *nbest* PSO algorithm, and empirical results in section 4.4 show the algorithm to be effective. Section 4.5 concludes this chapter with a study of the neighborhood size parameter.

4.1.1 Systems of Equations

Optimization of systems of equations is an important task in academic and commercial environments. Finding an optimal solution to a problem can very often be simplified to solving of a set of equations describing such a problem. Such systems can be linear or nonlinear. A general formulation of a system of m *linear* equations in n unknowns is given as [26]:

$$\begin{array}{ccccccccc} a_{11}x_1 & + & a_{12}x_2 & + & \cdots & + & a_{1n}x_n & = & b_1 \\ a_{21}x_1 & + & a_{22}x_2 & + & \cdots & + & a_{2n}x_n & = & b_2 \\ & & & & \vdots & & & & \\ a_{m1}x_1 & + & a_{m2}x_2 & + & \cdots & + & a_{mn}x_n & = & b_m. \end{array} \tag{4.1}$$

In systems of *linear* equations, no component variable ($x_1, \ldots, x_n$ in system (4.1)) has a degree higher than one or lower than zero. The system can be written as a single matrix equation

$$A\mathbf{x} = \mathbf{b} \tag{4.2}$$

where both $\mathbf{x}$ and $\mathbf{b}$ represent column vectors, with i^{th} dimensions represented by x_i and b_i respectively. If an n-dimensional vector $\mathbf{s}$ can be found such that $A\mathbf{s} = \mathbf{b}$, then $\mathbf{s}$ is a solution to the system represented in equation (4.1). The symbol A represents an $m \times n$ matrix. A system, such as in (4.1), can be written as an *augmented matrix*

$$\left[\begin{array}{cccc|c} a_{11} & a_{12} & \cdots & a_{1n} & b_1 \\ a_{21} & a_{22} & \cdots & a_{2n} & b_2 \\ & & \vdots & & \vdots \\ a_{m1} & a_{m2} & \cdots & a_{mn} & b_n \end{array}\right] \tag{4.3}$$

Some of the techniques discussed below focus specifically on this formulation of SEs.

Variables in systems of nonlinear equations may have degrees greater than one, or contain transcendental functions such as trigonometric or logarithmic functions. Therefore, instead of multiplying a variable x_{rc}, where r and c respectively represent rows and

columns in the system, with a constant a_{rc}, x_{rc} is first passed through a function f_{rc}. In keeping with the format introduced in system (4.1), a system of non-linear equations can be written as

$$\begin{array}{ccccccccc} a_{11}f_{11}(x_1) & + & a_{12}f_{12}(x_2) & + & \cdots & + & a_{1n}f_{1n}(x_n) & = & b_1 \\ a_{21}f_{21}(x_1) & + & a_{22}f_{22}(x_2) & + & \cdots & + & a_{2n}f_{2n}(x_n) & = & b_2 \\ & & & & \vdots & & & & \\ a_{m1}f_{m1}(x_1) & + & a_{m2}f_{m2}(x_2) & + & \cdots & + & a_{mn}f_{mn}(x_n) & = & b_m. \end{array} \tag{4.4}$$

The functions f_{rc} may be any linear or nonlinear transformation, such as $f(x) = x$, $f(x) = x^2$, $f(x) = \ln(x)$, $f(x) = \sin(\frac{\pi}{2}x)$ or $f(x) = \sqrt{x}$ (subscripts were dropped for improved readability). The right hand side of f_{rc} generally consists of a single term.

Algebraic methods to solve systems of equations exist, ranging from relatively simple approaches for small linear systems, to computationally expensive techniques for large, non-linear systems. The following sections present an overview of a number of the simpler numerical techniques.

Graphing

Graphing is a visual approach to solving SEs. It works by simply plotting each equation in a system, such as those described in systems (4.1) or (4.4). Solution(s) can be found at positions where curves in the system intersect. Graphing's accuracy unfortunately depends on the practitioners ability to correctly draw equation graphs manually. Graphing nonlinear functions, such as transcendental functions and functions in variables of a degree higher than one, is a difficult task. If solutions are represented by fractions, accuracy will be lost because a user would be forced to guess an appropriate value. Systems of a dimension higher than three can not be easily graphed and visualized in the human brain. Typically, graphing a solution to a multidimensional problem entails elaborate decomposition of the sets of equations into different dimensions. Each decomposed graph will then be subject to the problems mentioned above. The use of graphing to solve SEs is an option if an accurate graphing package is available.

Substitution

The substitution technique, which is applicable to both linear and nonlinear SEs, finds solutions by rewriting individual equations in terms of other equations in the system. This technique is particularly useful for solving simple linear systems. The following example explains this technique.

Consider a simple linear system of equations:

$$\begin{aligned} \text{A:}\quad & 2x + 3y = 6 \\ \text{B:}\quad & x + y = 5 \end{aligned} \tag{4.5}$$

The system can be solved by rewriting equation B in system (4.5) as

$$\text{C:}\quad x = 5 - y$$

and substituting it in A, resulting in

$$\begin{aligned} \text{D:}\quad & 2(5-y) + 3y = 6 \\ \Rightarrow\quad & 10 - 2y + 3y = 6 \\ \Rightarrow\quad & y = -4 \end{aligned}$$

The calculated y value is then substituted into equation A, allowing a solution for x to be found, i.e.

$$\begin{aligned} \text{E:}\quad & 2x + 3(-4) = 6 \\ \Rightarrow\quad & 2x - 12 = 6 \\ \Rightarrow\quad & 2x = 18 \\ \Rightarrow\quad & x = 9 \end{aligned}$$

$x = 9$ and $y = -4$ then represents the system's only solution.

Gauss-Jordan Elimination

A *linear* system can be quickly solved by rewriting it in a matrix format, such as in equation (4.3), and manipulating its coefficients a_{rc} by column and row operators. This process can be demonstrated by rewriting system (4.5) in matrix notation, i.e.

$$\left[\begin{array}{cc|c} 2 & 3 & 6 \\ 1 & 1 & 5 \end{array}\right] \tag{4.6}$$

System (4.6) can be simplified and solved by applying row operators as follows:

$$\left[\begin{array}{cc|c} 0 & 1 & -4 \\ \frac{1}{3} & 0 & 3 \end{array}\right] \begin{array}{l} R_1 - 2R_2 \\ R_2 - \frac{1}{3}R_1 \end{array} \tag{4.7}$$

where R_1 and R_2 designate the top and bottom rows in the (4.6) respectively. Matrix (4.7) can be further simplified to

$$\left[\begin{array}{cc|c} 0 & 1 & -4 \\ 1 & 0 & 9 \end{array}\right] \begin{array}{l} R_1 \\ 3R_2 \end{array} \tag{4.8}$$

The process used to write SEs as matrices can be reversed to rewrite (4.8) as a SEs, yielding

$$\begin{aligned} y &= -4 \\ x &= 9, \end{aligned}$$

which is the solution to the system.

Cramer's Rule

Cramer's Rule uses matrix determinants to solve systems of *linear* equations. The rule states that for a linear system $A\mathbf{x} = \mathbf{b}$, where A is an $n \times n$ invertible matrix, each element of $\mathbf{x}$ can be calculated as

$$x_k = \frac{\det(B_k)}{\det(A)}$$

where $k = 1, \ldots, n$. The matrix B_k can be obtained from the coefficient matrix A by substituting the k^{th} column of A by $\mathbf{b}$. This process is fast for small matrices, but becomes progressively more complex as more replacements are necessary for B_k. Keeping

with the matrix example in equation (4.6), x and y can respectively be solved for:

$$x = \frac{\begin{vmatrix} 6 & 3 \\ 5 & 1 \end{vmatrix}}{\begin{vmatrix} 2 & 3 \\ 1 & 1 \end{vmatrix}} = \frac{6-15}{2-3} = \frac{-9}{-1} = 9$$

$$y = \frac{\begin{vmatrix} 2 & 6 \\ 1 & 5 \end{vmatrix}}{\begin{vmatrix} 2 & 3 \\ 1 & 1 \end{vmatrix}} = \frac{10-6}{2-3} = \frac{4}{-1} = -4$$

Matrix Inverses

If an inverse for a matrix A exists, a solution for the system $A\mathbf{x} = \mathbf{b}$ is

$$\mathbf{x} = A^{-1}\mathbf{b}. \tag{4.9}$$

The theory behind finding inverses of matrices is vast and is therefore not discussed here. The interested reader is referred to [26] for a thorough treatment. Using the formula in equation (4.9) and the inverse of equation (4.6), $\begin{bmatrix} -1 & 3 \\ 1 & -2 \end{bmatrix}$, system (4.5) can be solved by calculating

$$\begin{aligned} & \mathbf{x} = A^{-1}\mathbf{b} \\ \Rightarrow \quad & \begin{bmatrix} x \\ y \end{bmatrix} = \begin{bmatrix} -1 & 3 \\ 1 & -2 \end{bmatrix} \begin{bmatrix} 6 \\ 5 \end{bmatrix} \\ \Rightarrow \quad & = \begin{bmatrix} -6+15 \\ 6-10 \end{bmatrix} \\ \Rightarrow \quad & = \begin{bmatrix} 9 \\ -4 \end{bmatrix} \end{aligned}$$

Nonlinear Techniques

Of the above techniques, *Gauss-Jordan Elimination*, *Cramer's Rule* and *Matrix Inverses* apply only to *linear* systems of equations, subject to the determinant of the coefficient

matrix A not being zero, if it exists. Nonlinear sets of equations are harder to solve. Techniques such as the above that appear relatively simple to apply, can no longer be used. More elaborate numerical techniques exist. These include [78]:

- Newton's method
- Broyden's method
- Line searching
- Bisection
- The Secant method
- Steepest Descent

The above techniques find only approximate solutions and cannot guarantee that a complete set of solutions have been found. The efficiency of techniques such as *Newton's method*, *Broyden's method*, the *Secant method* and *steepest descent* also depend on the initial positions of the respective searching processes. A starting position far from a solution may lead to an extended search that may never converge. The type of system to be solved will determine the technique used — no technique is universally applicable.

4.1.2 Solving Linear Systems with Neural Networks

A number of authors have investigated the possibilities of using neural networks to solve systems of equations, with varying results.

Cichocki and Unbehauen implemented neural networks in circuit architectures to solve systems of linear equations [12]. Their work was motivated by the following:

- The *inversion* of large matrices is a time consuming process (see section 4.1.1). If traditional numerical approaches are utilized, the calculation of an inverted matrix in a time-critical online system may still be too slow.
- Developing simple artificial neural network models to solve a simple linear programming problem could lead to a better understanding of the problem under

consideration. The consequent development of new solution techniques could lead to improved, general methods [12].

To accommodate noise in real environments, the basic formulation of systems of linear equations in equation (4.2) was restated as

$$A\mathbf{x} = \mathbf{b}^{'} + \mathbf{r} = \mathbf{b}_{true}$$

where $\mathbf{b}^{'}$ represents real world observations made, $\mathbf{r}$ represents measurement errors and $\mathbf{b}_{true}$ represents actual values for $\mathbf{b}$ that may be unknown. A neural network, embedded in a circuit architecture, then learns as its outputs the solution vector $\mathbf{x}$. Their results showed that neural networks can successfully and very efficiently learn solution to SEs. Gabrys and Bargiela implemented Cichocki and Unbehauen's approach in a water control system [27].

Huang and Chi designed neural network architectures based on the dimensions of a SE described in equation (4.2) [42]. Equation (4.2) is rewritten as

$$A\mathbf{w} \approx \mathbf{b}$$

where $\mathbf{w} = [w_1, w_2, \ldots, w_n]^T$ represents the weight values of a feed-forward neural network. The coefficient matrix A is written as a set of row vectors,

$$A = \begin{bmatrix} \mathbf{a}_1 \\ \mathbf{a}_2 \\ \vdots \\ \mathbf{a}_n \end{bmatrix} \text{ where } \mathbf{a}_i = [a_{i1}, a_{i2}, \ldots a_{in}] .$$

Using this reformulation, equation (4.2) is rewritten as

$$\mathbf{a}_i\mathbf{x} = b_i \tag{4.10}$$

where $\mathbf{x} = [x_1, x_2, \ldots x_n]^T$. A neural network is then trained by providing sample $\mathbf{x}$ values to approximate $\mathbf{b}$ through the adaptation of $\mathbf{w}$ under the imposed constraints defined in equation (4.10). The interested reader is referred to [42] for a detailed analysis of the above technique[1].

[1] Huang and Chi also introduced a similar neural network based approach to find roots of polynomials [41].

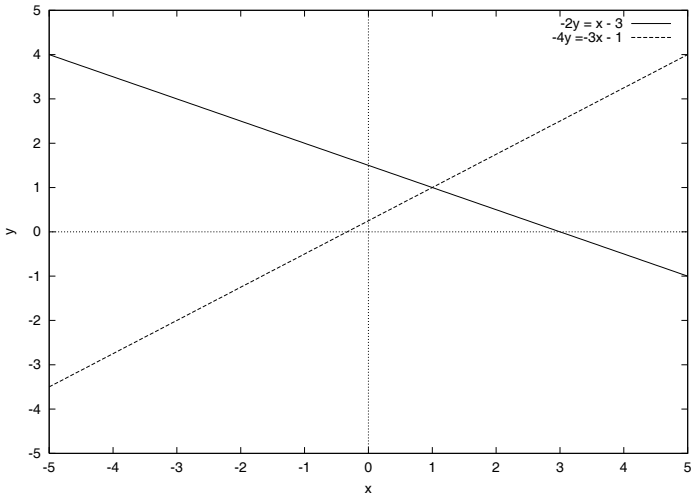


Figure 4.1: System S1, a simple system of two linear equations.

4.2 PSO and Solving SEs

This section extends the discussion of systems of equations, but considers it from a particle swarm optimization perspective. Solving SEs is restated as an optimization problem, and shortcomings of traditional PSO unimodal optimization approaches, *lbest* and *gbest*, are identified.

Similar to the neural network based approach introduced by Cichocki and Unbehauen [12], when using PSO to solve SEs, the goal is to find the solution vector $\mathbf{x}$ in the system $A\mathbf{x} = \mathbf{b}$. In a swarm of particles, each particle represents a candidate solution for each parameter in a system of equations, in this case $\mathbf{x}$. As an example, system (4.11) represents a simple system of linear equations with a single solution at the coordinates $(1, 1)$, as shown in figure 4.1:

$$\text{S1:} \quad \begin{aligned} -2y &= x - 3 \\ 4y &= 3x + 1 \end{aligned} \tag{4.11}$$

When attempting to solve system (4.11) with PSO, the goal is to find values for the unknowns (x, y). Each particle therefore represents a set of candidate values for x and y. To ascertain the quality of a (x, y) pair, the fitness function is based on the formulation of the SEs in equation (4.11).

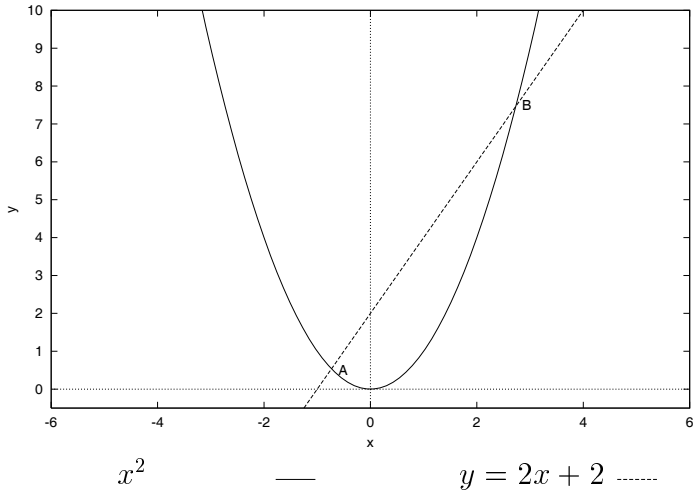


Figure 4.2: System S4, with multiple solutions.

A particle's fitness is determined by how close it is to the known solution of a SEs. The fitness function for S1 can then be defined as

$$f_{S1}(x,y) = |f_{S1,1}(x,y)| + |f_{S1,2}(x,y)| \tag{4.12}$$

where

$$\begin{aligned} f_{S1,1}(x,y) &= x + 2y - 3 = 0 \\ f_{S1,2}(x,y) &= 3x - 4y + 1 = 0 \end{aligned}$$

The objective is then to minimize $f_{S1}(x,y)$. The lower the error represented by $f_{S1}(x,y)$, the closer a swarm of particles is to the optimal solution of the system. Experimental results presented later shows that both *gbest* and *lbest* have no problem to locate the optimal solution (see table 4.2 on page 77). However, keep in mind that this optimization problem defines a single, clear goal. No local optima exist in system S1's search space, which explains the success of *lbest* and *gbest* in locating the single optimum.

When a SEs has multiple solutions, the optimization process becomes more complex. Consider the following system of equations, illustrated in figure 4.2:

$$\text{S4:} \begin{aligned} y &= x^2 \\ y &= 2x + 2 \end{aligned} \tag{4.13}$$

The fitness function $f_{S4}(x,y)$ is defined as

$$f_{S4}(x,y) = |f_{S4,1}(x,y)| + |f_{S4,2}(x,y)|.$$

where

$$\begin{aligned} f_{S4,1}(x,y) &= x^2 \\ f_{S4,2}(x,y) &= 2x + 2 \end{aligned}$$

The curves in system S4 intersect at *two* distinct positions in the search space. Both these points return equal, minimal fitness values. Attempts to find all solutions to this system with 'traditional' PSO optimization approaches, such as *gbest* and *lbest*, fail.

Both *gbest* and *lbest* implicitly assume either that the search contains but a single optimal solution, or that the goal of the search process is to locate only one solution. This behavior is expected of the *gbest* algorithm, as the position update equation (see equation (2.9) on page 16) is designed to force all particles to move to a single global position in the search space, that defines the best position located by the swarm at any given time step. The swarm's global best position can represent only one of the possible solutions. At first glance, it is expected that the *lbest* PSO will obtain more than one solution due to the formation of particle neighborhoods; that is, each *neighborhood best* will represent a solution. This is however not the case, since *lbest* propagates information about optimal positions through overlapping neighborhoods. That is, a particle is a member of multiple neighborhoods. This configuration leads to the convergence of all neighborhood best particles onto a single solution. The location of particles within a particular search space has no effect on the formation of neighborhoods: Neighborhoods are determined based on the particle *indices* only.

Given that existing standard PSO approaches clearly are not suited to the location of multiple solutions within a single search space, new techniques need to be proposed. The next section presents a new, computationally inexpensive approach to locate multiple optima, without alteration of the search space.

4.3 The *nbest* PSO

This section proposes modifications to the standard particle swarm optimizer that enables it to locate multiple solutions in a search space. First, the fitness function is extended to reward a particle when it is close to any of the possible solutions in a sys-

tem of equations[2]. A new approach to determine neighborhood best particles is then introduced. These modifications are specifically aimed at solving systems of equations.

4.3.1 'Intelligent' Fitness Function

A fitness function quantifies the quality of a potential solution [24]. The fitness function formulation given in equation (4.12) is adequate for simple systems, such as S1 and S4. In general, for a system of m equations, this approach can be written as

$$f(\mathbf{x}_i) = \sum_{k=1}^{m} f_k(\mathbf{x}_i). \tag{4.14}$$

$f_k(\mathbf{x}_i)$ represents each one of the m equations, where each equation is algebraically rewritten to be equal to zero. The fitness formulation in equation (4.14) assumes that:

- All equations intersect at a single, unique position, or that
- the fitness of a system can be determined directly from its set of equations (i.e. the number of equations is less than or equal to the number of unknowns in a linear system).

This formulation does not accurately report fitness when dealing with nonlinear systems where the number of intersection points, or solutions, depend on the number of equations and unknowns. When dealing with situations where there are *more* equations than unknowns, solving a SEs is expanded to finding all points of intersection, of all equations making up the SEs, rather than searching for points where all the equations intersect. The goal of solving a SEs with a swarm intelligence approach is to locate *all* these points of intersection.

As an example, consider system S3, as illustrated in figure 4.3:

$$\text{S3:} \quad \begin{aligned} y &= 2x - 3 \\ y &= -3x - 1 \\ y &= -x + 1 \end{aligned} \tag{4.15}$$

System S3 has three solutions. For all solutions to be located, the fitness function should

[2]Although the design of this technique was motivated by the particular need to solve SEs, given sufficient prior knowledge of a search space, it is extendable to other problems.

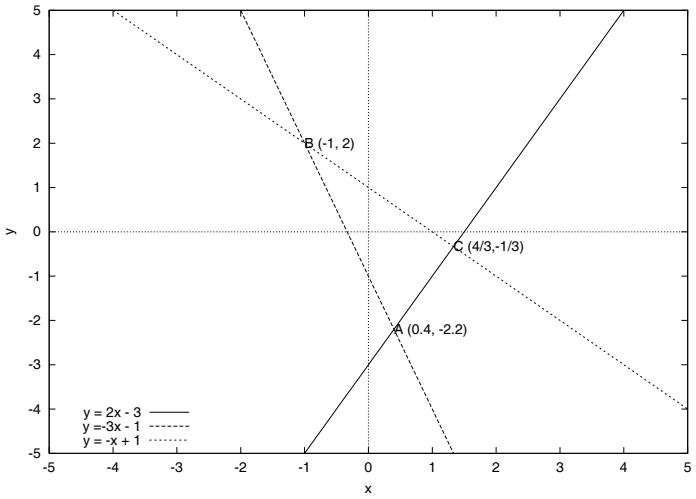


Figure 4.3: System S3: A system of linear equations with 3 solutions.

consider a particle's relative distance to each of the possible solutions. The assumption above that all the equations in the system intersect, no longer holds. A solution to the system may be found at any position where only a subset of the equations in the set of equations intersect. The 'shortest route' to convergence for a particle would then be to adapt its candidate solution towards the actual solution that it lies closest to.

Thus, to evaluate the fitness of a particle i for system S3, the fitness function is redefined to

$$f_{ABC}(\mathbf{x}_i) = \min\{f_{AB,AC}(\mathbf{x}_i), f_{BA,BC}(\mathbf{x}_i), f_{CB,CA}(\mathbf{x}_i)\} \tag{4.16}$$

where

$f_{AB,AC}(\mathbf{x}_i)$ is the fitness of particle $\mathbf{x}_i$ with respect to equations $y = 2x - 3$ and $y = -3x + 1$,

$f_{BA,BC}(\mathbf{x}_i)$ is the fitness of particle $\mathbf{x}_i$ with respect to equations $y = -3x + 1$ and $y = -x + 1$, and

$f_{CB,CA}(\mathbf{x}_i)$ is the fitness of particle $\mathbf{x}_i$ with respect to equations $y = -x + 1$ and $y = 2x - 3$.

This formulation of the fitness function *implicitly* assumes that all the lines in the system of equations actually intersect. However, to develop a general fitness function

formulation, this assumption cannot be made. When two lines do not intersect, (e.g. parallel lines or asymptotes) the result obtained when evaluating the fitness function, will be an indication of the distance between lines. If there are no intersections between lines in a SEs and therefore no solutions, particles will eventually settle on locations where lines in the system are the closest to each other, thereby still minimizing the fitness function. The fitness formulation in equation (4.16) can thus be generally applied.

The proposed reformulation of the fitness function rewards a particle for being close to *one* of a set of possible solutions. A general formulation of this fitness for a particle position $\mathbf{x}_i$, for a system of m equations is

$$f(\mathbf{x}_i) = \min_{\forall \kappa}\{f_\kappa(\mathbf{x}_i)\}$$

The symbol κ represents an element of the set of possible intersections between the m equations that define a SEs. If $< \imath, \jmath >$ represents the intersection between equations $\imath$ and $\jmath$, then the set of possible intersections, Γ, is defined as

$$\Gamma = \{< 1,1 >, \ldots, < 1,m >, < 2,1 >, \ldots, < 2,m >, \ldots, < m,1 >, \ldots < m,m >\} \tag{4.17}$$

and $\kappa \in \Gamma$. κ represents a single element in Γ and $\{\kappa\} \neq \Gamma$. $< 1,m >$ represents equations 1 and m in the SE rewritten to be equal to zero. The sets of intersecting equations represented in equation (4.17) assumes that a maximum of two of the m lines will actually intersect. Also, it is trivial that a solution to a SE does not exist where an equation represents a locus that intersects with itself. Entries such as $(1,1)$ and (m,m) are therefore to be ignored. If more than 2 lines do intersect in a SEs, the representation in equation (4.17) can be expanded to accommodate it, e.g. if three equations, e_1, e_2 and e_3 intersect, the above notation would represent it as $< e_1, e_2, e_3 >$.

This section presented a reformulation of the fitness function for a SEs. This reformulation allows the PSO to effectively locate multiple solutions in a search space. Next, the concept of a topological neighborhood is introduced to take advantage of the spatial positions of particles in a search space.

4.3.2 Topological Neighborhoods

The definition of the *lbest* PSO ensures that the algorithm spreads information about good solutions to all particles in a swarm. Standard *lbest* bases its neighborhood definition on particle indices, where each particle is assigned an unique index number that does not change over the course of the optimization process. Spatial positions therefore do not play a role when determining a particle neighborhood. This model is well-suited to unimodal optimization problems. It allows efficient sharing of a set of diverse potential solutions, while avoiding premature convergence [49]. A number of authors investigated techniques that redefine the neighborhood of a particle, to ensure eventual convergence on a global optimum in a search space. See section 2.7.2 for a discussion of these techniques. When searching for *multiple* solutions, neighborhood modifications, as well as *gbest*, are still biased towards finding a single optimum solution in the search space.

The diversity improvement techniques in section 2.7.2 all endeavor to spread information about good solutions to *all* particles in the swarm. When searching for multiple solutions, it is beneficial to restrict the sharing of social information based on a particle's proximity to a potential solution. Thus, instead of moving towards a global best solution located by the complete swarm, a particle would be better served to move towards a solution close to it in the search space. This can be achieved by defining a local, topological particle neighborhood.

To this end, the *nbest* PSO introduces a *neighborhood best* position. For a particle i, the neighborhood best $\hat{\mathbf{y}}_i$ is defined as *the center of mass of the positions of all the particles in the* topological *neighborhood of* i. Practically, the topological neighborhood is defined as the k closest particles to i, where the closest particles are found by calculating the Euclidean distance between $\mathbf{x}_i$ and all other particles in the swarm. Formally, for each particle define the set B_i, where B_i consists of the k closest particles to i at any given time step t; $\hat{\mathbf{y}}_i$ is then

$$\hat{\mathbf{y}}_i = \frac{1}{k} \sum_{h=1}^{k} \mathbf{B}_{ih} \tag{4.18}$$

where $\mathbf{B}_{ih}$ is the current position of the h^{th} particle in neighborhood B_i of particle i at time t; k is a user defined parameter. The set of particles in a neighborhood are all weighted equally.

The velocity update equation is similar to that used in the *lbest* PSO, but the neighborhood influence $\hat{\mathbf{y}}_i$ is calculated as shown in equation (4.18). The update for $v_{i,j}(t+1)$ is defined as

$$\begin{aligned} v_{i,j}(t+1) &= v_{i,j}(t) + c_1 r_{1,j}(t)(y_{i,j}(t) - x_{i,j}(t)) + \\ & \quad c_2 r_{2,j}(t)(\hat{y}_{i,j}(t) - x_{i,j}(t)) \end{aligned} \tag{4.19}$$

From equations (4.18) and (4.19), it follows that if the neighborhood size k is very large, i.e. approaching the swarm size, *nbest* approximates an algorithm similar to *gbest*, where all particles move towards a single globally defined location. The goal position $\hat{\mathbf{y}}$ will, however, represent an average particle position in the search space, conveying no information about a possible good result. In section 4.5 a study of the influence of the parameter k is presented.

4.4 Experimental Results and Discussion

This section presents empirical results obtained from the application of *gbest, lbest* and *nbest* to solve systems of *unconstrained* linear and nonlinear equations. Constrained optimization with the PSO, in the form of multi–objective problems, have been investigated by a number of authors [15, 38, 74, 75]. Paquet used PSO to solve the constrained optimization problem associated with training support vector machines [70].

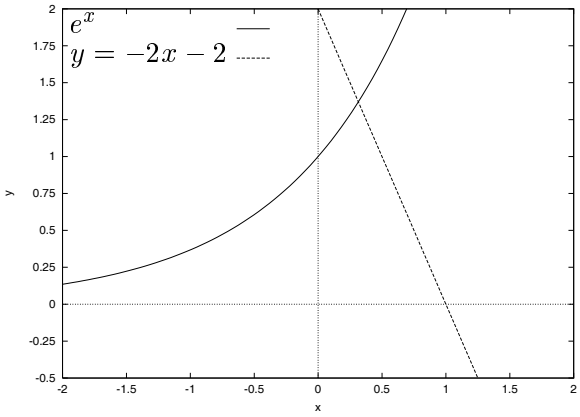
In addition to the systems defined previously, the following systems of equations are considered:

$$\begin{aligned} \text{S2: } y &= e^x \\ y &= -2x + 2 \end{aligned} \tag{4.20}$$

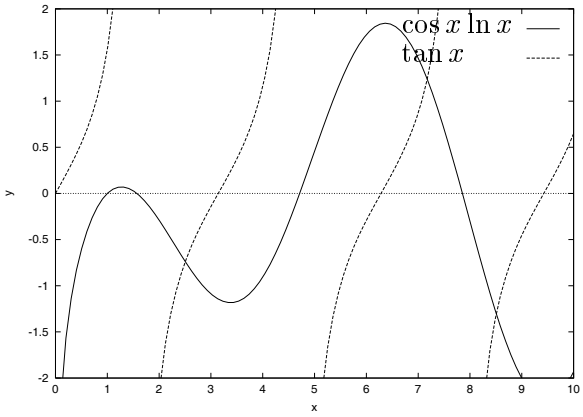
$$\begin{aligned} \text{S5: } y &= \cos x \ln x \\ y &= \tan x \end{aligned} \tag{4.21}$$

$$\begin{aligned} \text{S6: } y &= \sin x \\ x &= \tan y \end{aligned} \tag{4.22}$$

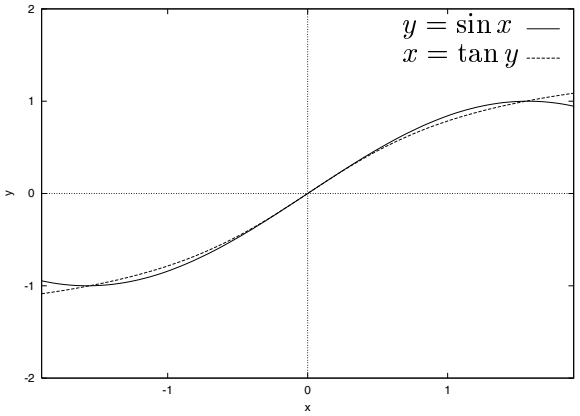
Figure 4.4 illustrates each of these systems.



(a) System S2



(b) System S5



(c) System S6

Figure 4.4: Additional Test Systems of Equations

For each set of equations, 30 simulation runs were done for each of the *lbest*, *gbest* and *nbest* algorithms. For each simulation, the inertia weight w was linearly scaled from 0.7 to 0.1 over 2000 iterations, with $c_1 = 2.0$ and $c_2 = 2.0$ kept constant.

These parameter settings require the velocity values to be clamped to the range $[-v_{max}, v_{max}]$ in order to ensure convergence [87]. The use of a linearly decreasing inertia weight promotes exploration during the earlier iterations, resulting in a thorough search of the solution space.

Table 4.1 specifies settings for v_{max}, x_{min} and x_{max}, where x_{min} and x_{max} defines the domain of each of the problems, while v_{max} is the largest velocity value that will be allowed for any dimension. These limits were chosen since all solutions are within the defined ranges. Table 4.2 summarizes the number of exact solutions found by *gbest*,

System	x_{min}	$x_{max} = v_{max}$
S1	-10.0	10.0
S2	-10.0	10.0
S3	-10.0	10.0
S4	-10.0	10.0
S5	0.1	10.0
S6	-2.0	2.0

Table 4.1: x_{min}, x_{max} and v_{max} parameter values for *nbest* experiments

lbest and *nbest* for each of the SEs. Regardless of the actual number of solutions, both the *gbest* and *lbest* algorithms always converged to a single solution, even when multiple solutions exist. This behavior of *gbest* is expected since all particles home in onto one particle, namely the global best particle of the swarm. For the *lbest* algorithm, the same happens due to the fact that neighborhoods overlap, as explained in section 4.3.2.

The *nbest* algorithm succeeded in finding all the solutions for all problems except for problem S6, shown in figure 4.4(c). For S6, none of the algorithms succeeded in locating a specific solution. In this case, a large number of points exist with fitness values very close to zero. All experiments converged to good approximate solutions close to zero (as indicated in table 4.3), and within the range $[-\pi/2, \pi/2]$. Table 4.3 lists the average fitness of the best particle for each of the three algorithms. For the *nbest* algorithm, the

given values represent the average fitness over all the solutions found by the swarm.

Problem	***gbest***	***lbest***	***nbest***	Actual #Solutions
S1	1	1	1	1
S2	1	1	1	1
S3	1	1	3	3
S4	1	1	2	2
S5	1	1	3	3
S6	*	*	*	*

Table 4.2: Solutions found by *nbest*, *gbest* and *lbest*

Problem	***gbest***	***lbest***	***nbest***
S1	6.29E-06	6.80E-06	4.52E-06
S2	6.63E-06	7.17E-06	6.60E-02
S3	7.30E-06	6.73E-06	7.08E-04
S4	8.02E-06	6.90E-06	8.60E-04
S5	7.35E-02	6.73E-06	7.15E-04
S6	2.93E-05	2.91E-02	5.13E-06

Table 4.3: *nbest* Results: Mean Best Fitness

4.5 An Analysis of the Neighborhood Size k

Existing diversity improvement techniques that modify a particle's neighborhood, share information on a global scale within a search space, ensuring that particles converge onto a single solution. The goal of *nbest* is not to increase a particle's neighborhood size over time to the complete swarm. Doing so defeats the goals of niching and speciation. Rather, neighborhoods should stably maintain multiple solutions within a search space. The influence that a particle neighborhood has in the *nbest* algorithm is controlled by the neighborhood size parameter, k. A neighborhood's size controls the proportion of social information 'communicated' by the swarm to a particular particle. To this end, this section investigates the niching capabilities of *nbest* for different k values.

Configuration	$k_{initial}$	k_{final}
D1	1	1
D2	5	5
D3	$\|S\|$	$\|S\|$
D4	$\|S\|$	1
D5	5	1

Table 4.4: Different experimental configurations of the neighborhood size parameter k

Table 4.4 describes a number of different parameter configurations used to analyze the influence of k. Configurations D1, D2 and D3 keep the value of k constant for each simulation, while D4 and D5 linearly scales k from $k_{initial}$ to k_{final} over the maximum number of allowed iterations of the algorithm. These configurations are compared on two different optimization problems. The first problem is a SEs with a single solution, for which the traditional *gbest* and *lbest* algorithms will encounter no difficulties to locate the only global solution. The second test system has multiple solutions, and cannot be solved with *gbest* and *lbest*. The systems are defined in equations (4.11) and (4.15), and are illustrated in figures 4.1 and 4.3 respectively.

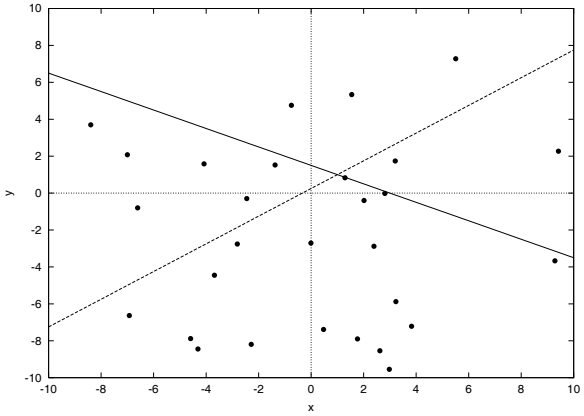
The influence of k is demonstrated by running the *nbest* algorithm with the different k values as set out in table 4.4 on the functions above. Fitness functions are defined as described in section 4.3.1. For each SEs, $c_1 = c_2 = 1.4$ and w is linearly scaled from 0.7 to 0.1 over 2000 iterations of the *nbest* algorithm. Initial particle positions are selected randomly within the range $[-10, 10]^2$.

Figure 4.5 shows the initial particle positions that were used for each problem type. The value of k is linearly scaled over the iterations of the PSO algorithm: the value of k at time step t is determined using the formula:

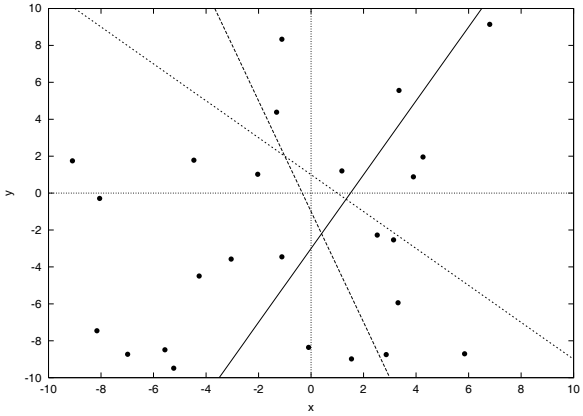
$$k_t = \left\lceil \frac{t_{\max} - t}{t_{\max}} \cdot (k_{initial} - k_{final}) + k_{final} \right\rceil$$

where $t_{\max}$ is the maximum time step and $k_{initial}$ and k_{final} is defined as in table 4.4. Next, the effects of the different k values are considered individually:

D1 When the neighborhood size is kept constant at $k = 1$ for 2000 iterations of the *nbest* algorithm, virtually no learning progress is made. For both S1 and S3,

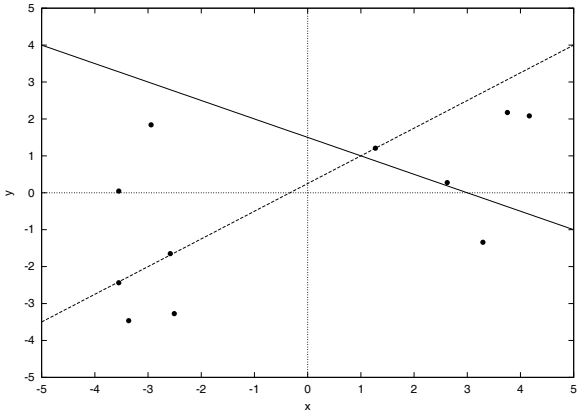


(a) Single solution: Initial particle positions

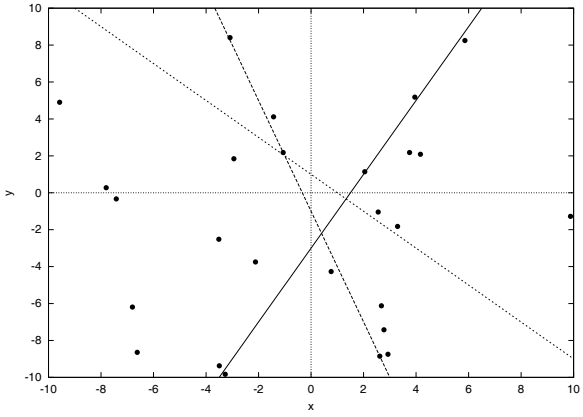


(b) Multiple solutions: Initial particle positions

Figure 4.5: Initial particle positions for analysis of neighborhood size

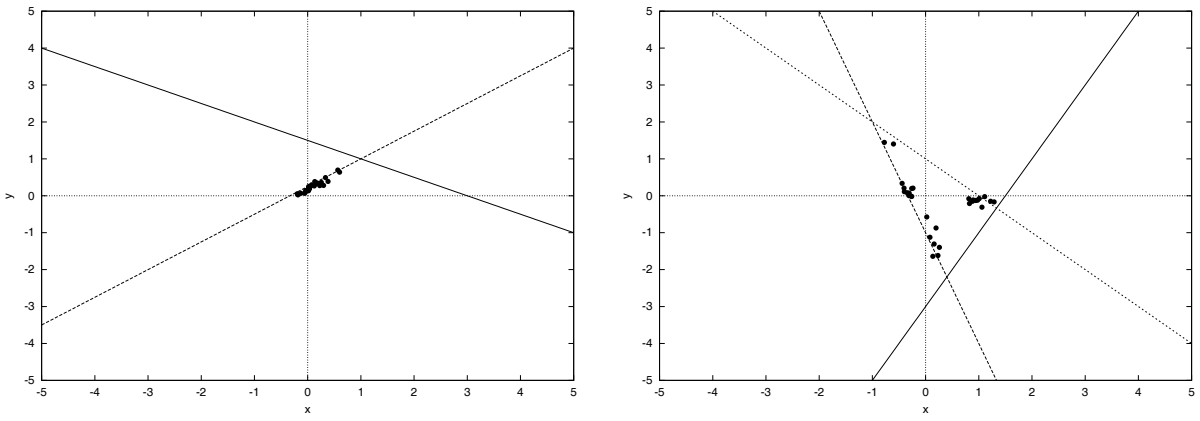


(a) Single solution



(b) Multiple solutions

Figure 4.6: D1: Particle position after 2000 iterations, with $k = 1$ kept constant

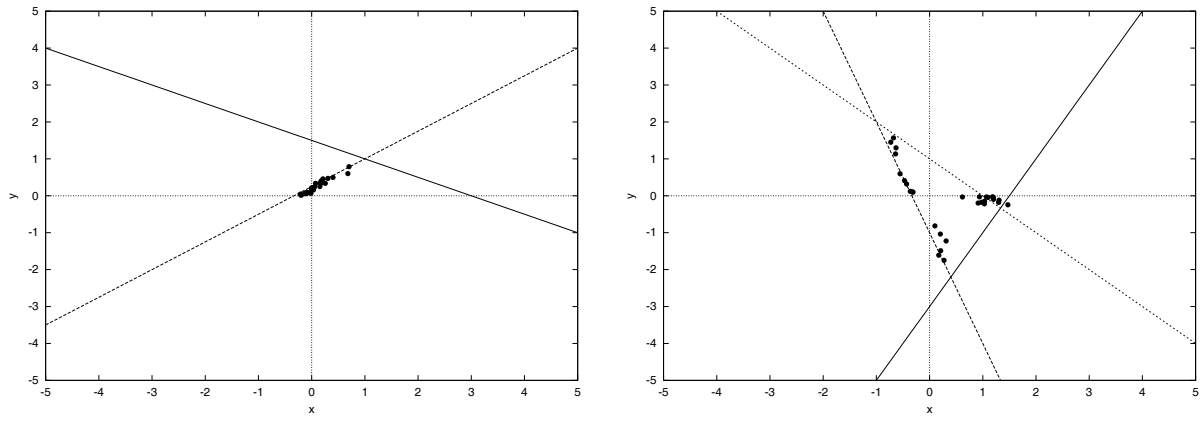


(a) Single solution (b) Multiple solutions

Figure 4.7: D3: Particle position after 2000 iterations, with $k = |S|$ kept constant

further simulations where *nbest* was left to run for 20 000 iterations with $k = 1$ also did not converge on the possible solutions. Figure 4.6 shows that particle positions still appear to be random, with only lazy movement towards possible optima. Since *nbest* uses spatial neighborhoods to calculate velocity and position updates, conceptually, when particle a finds particle b to be its closest neighbor, it is quite possible, although not necessarily guaranteed, that b will choose a as its closest neighbor. It is entirely possible that there exists a situation where a third particle c is close to b, such that $\|\mathbf{x}_b - \mathbf{x}_c\| < \|\mathbf{x}_a - \mathbf{x}_b\|$. In such a case, without the influence of any other particles and the lack of guidance such as is present in the *gbest* and *lbest* algorithms, particles a and b will be attracted to each other. Because of the limited social exchange, this arrangement will lead to the slow pursuit of the best position found between the two particles at any given iteration, possibly leading the particles to stagnate on a suboptimal position in the search space.

D2 and D3 Figures 4.7 and 4.8 show particle positions when testing the algorithm with $k = |S|$ and $k = 5$, respectively. In both cases, particles generally converge on the possible solutions. Constant neighborhoods where $k > 1$ allows greater

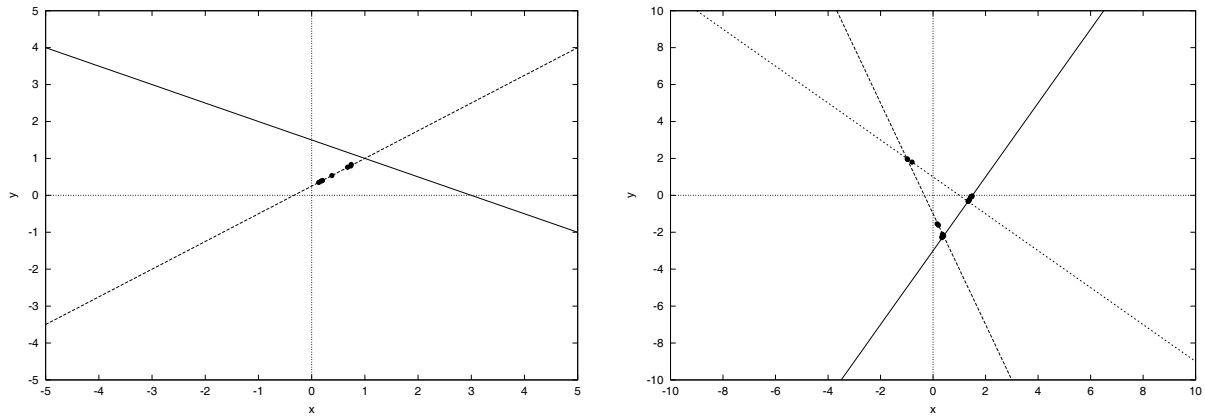


(a) Single solution (b) Multiple solutions

Figure 4.8: D2: Particle position after 2000 iterations, with $k = 5$ kept constant

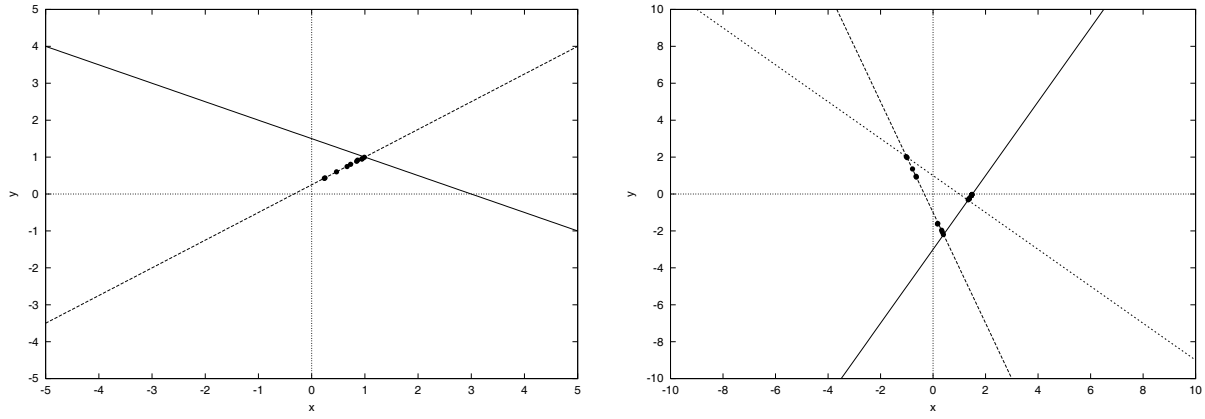
social interaction between a particle and its neighborhood, facilitating improved information exchange. The tendency of particles to settle on lines in a SEs is explained by the fact that any position on a line relatively close to a solution will have a low, and therefore attractive fitness value. An increased amount of social information is shared among particles – the extent of social interconnection remains constant throughout all iterations of the learning algorithm. Note that "extent of interconnection" simply refers to the size of a particle's neighborhood. The closest neighbors of every particle are recalculated after every velocity and position update, and the set of particles initially associated with a specific particle can change over time. If several optima occur in topologically close positions, configurations D2 and D3 will have difficulty to converge as a particle's neighborhood will be situated around several different solutions. Since neighborhoods of different particles can overlap, particles will not exhibit convergent behavior.

D4 and D5 Configurations D4 and D5 linearly scale the neighborhood size over time. These configurations ensure that a large degree of social information exchange takes place during the initial iterations of the algorithm. The social influence decreases over time until communication takes place with a single neighbor only. During the final iterations of the algorithm, the neighborhood size is the same for any particle,



(a) Single solution (b) Multiple solutions

Figure 4.9: D4: Particle position after 2000 iterations, with k linearly scaled between the swarm size and 1



(a) Single solution (b) Multiple solutions

Figure 4.10: D5: Particle position after 2000 iterations, with k linearly scaled between 5 and 1

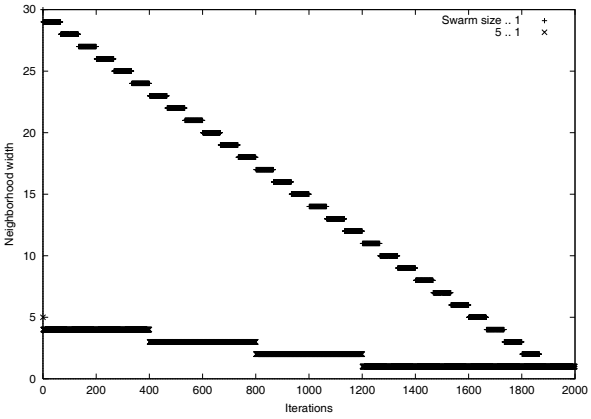


Figure 4.11: Linearly decreasing neighborhood sizes. Note that k is always a discrete value, explaining the stepwise decrease.

regardless of the initial size. Figure 4.11 shows how the neighborhood size decreases with iteration number. The greater efficiency of decreasing neighborhoods can be attributed to the decreasing influence of a spatial neighborhood on a particle over time. When a particle starts to move towards a particular solution or niche, its shrinking neighborhood will force it to move to a local solution, rather than to a global solution that is determined by the social experiences of the complete swarm.

Of the approaches tested in this section, linearly scaling the neighborhood size k over time yielded the most favorable results. Further simulations, where the neighborhood size was decreased *exponentially* over time, also yielded favorable results, but simulations did not converge as well when linear scaling was used. In this context, convergence refers to the algorithm's ability to locate and maintain multiple solutions concurrently, i.e. all particles have positions close to, or at the exact location of the solution. It seems that the initial rapid decrease in the size of the spatial neighborhood resulting in a small (i.e. a neighborhood where $k = 1$) neighborhood is less effective. The same social structure as with D1 and D2 occurs. Decreasing the particle neighborhood is virtually the exact opposite of Suganthan's *growing* particle neighborhood operator [85]. It is also noted that keeping k constant leads to some convergence only when $k > 1$. If this constraint does not hold, particles perform only a rudimentary local search, and do not exhibit any

convergent behavior.

4.6 Conclusion

This chapter presented the neighborhood particle swarm optimizer, *nbest*. It was shown that the social information exchange which forms the basis of the standard *lbest* and *gbest* algorithms keeps it from finding multiple solutions in a search space. The *nbest* PSO redefined particle neighborhoods to use spatial information to guide particles to a solution that it lies closest to. *nbest* was experimentally shown to be an effective niching technique. The influence of the neighborhood size parameter, k was investigated in section 4.5.

The next chapter presents the NichePSO optimizer, an algorithm that uses multiple subswarms to do niching.

Chapter 5

NichePSO, a Multi-Swarm Optimizer

This chapter presents a PSO technique that solves multimodal optimization problems with the concurrent optimization power of multiple swarms. The technique, NichePSO, extends the inherent unimodal nature of the standard PSO approach by growing multiple swarms from an initial particle population. The initial particle swarm is split into smaller swarms as niches are detected. Upon termination of the algorithm, each subswarm represents one of the potential solutions to the problem. Experimental results show that NichePSO can successfully locate all optima on a set of test functions. The influence of control parameters, including the relationship between the swarm size and the number of solutions (niches), as well as the scalability of the algorithm is investigated.

5.1 Introduction

This chapter presents the *niching particle swarm optimization* algorithm, NichePSO. NichePSO is aimed at locating multiple solutions to multimodal problems through the use of multiple, independent subswarms. The *nbest* optimizer presented in the previous chapter sports a drawback: It cannot properly maintain local optima. Overlap in *nbest* particle neighborhoods forces the algorithm to always prefer solutions that have better fitness. Consequently, when a local optimum occurs relatively close to another, better

optimum, the definition of a neighborhood will always prefer better solutions. The NichePSO algorithm overcomes this limitation through the use of multiple subswarms.

The use of subswarms, or *subpopulations*, as part of a population based optimization algorithm, is not a new idea. It has been applied in the GA optimization field, both as a

- diversity improvement technique [6], and as
- a premature convergence avoidance technique [84].

For the purposes of this thesis, subpopulations/subswarms imply

> A *bona-fide* segmentation of a large population of individuals/particles into smaller groupings. Each subswarm can function as a stable, individual swarm entity, evolving on its own, independent of individuals in other swarms.

The use of subswarms has been adopted early on in the development of the PSO. Løvbjerg *et al* introduced a diversity improvement technique that partitions a particle swarm into a number of different subpopulations [60]. Each subpopulation is responsible for maintaining its own best known, or *gbest*, solution. A crossover operator is used to share information about global solutions (see section 2.4.3). The crossover operator may be applied to particles from

- a single swarm, or
- particles originating from different swarms.

The algorithm selects particles on which crossover is to be performed randomly. Performing crossover between particles from the same swarm leads to better solutions and consequently faster convergence *within* a particular swarm. Crossover between particles from different swarms facilitates inter-population communication that eventually leads to a single *global* solution. Inter-swarm sharing of information is not conducive to the formation of subpopulations around different potential solutions. All swarms will eventually gravitate towards a single solution. Performing crossover in the same swarm to maintain a diverse 'local' record of good solutions reminds strongly of the goals of GA crowding techniques. Crowding techniques replace individuals in a population with similar individuals in a next generation (see section 3.3.4). It should however be noted that

the goal of the research presented by Løvbjerg *et al* was not to maintain multiple solutions and to perform niching, but to improve the quality of a global solution. Typically, this approach will be most useful in a deceptive problem domain, where particles may become trapped in suboptimal solutions. By performing crossover on randomly selected particles, particles fooled by a suboptimal solution can be moved closer to a better, or global best, solution.

In the rest of this chapter, the NichePSO algorithm is presented and motivated. A number of newly introduced niching parameters are analyzed and empirical results are presented that motivate the validity of NichePSO as a niching technique.

5.2 The Niching Particle Swarm Optimization Algorithm

The *nbest* optimizer was initially developed as a technique to find and maintain multiple points of intersection in systems of equations (SEs). Based on the reformulation of the fitness function presented in the previous chapter, points of intersection in a SEs will always have *equal*, *optimal* fitness in a search space. When local optima, i.e. points of suboptimal fitness exist, the *nbest* algorithm's neighborhood definition will keep it from locating these solutions. Although the neighborhood formulation introduces a bias towards a local optimum in a search space in the velocity update equation, neighborhoods for individual particles may still overlap. Consequently, if a suboptimal, local solution exists close to a solution that may yield a higher fitness, the neighborhood update will lead to an update, biased towards the better solution (see figure 5.1). NichePSO, through the use of subswarms and two control parameters, δ and μ, overcomes the shortcomings of the *nbest* algorithm.

NichePSO starts by uniformly distributing particles throughout the search space of an optimization problem. The initial swarm of particles is referred to as the *main* swarm. As particles traverse the search space, they invariably move towards positions that have attractive fitness. A potential solution is identified by monitoring the change in a particle's fitness over a number of training iterations. When such a solution is identified, a new subswarm is created by removing from the main swarm the particle

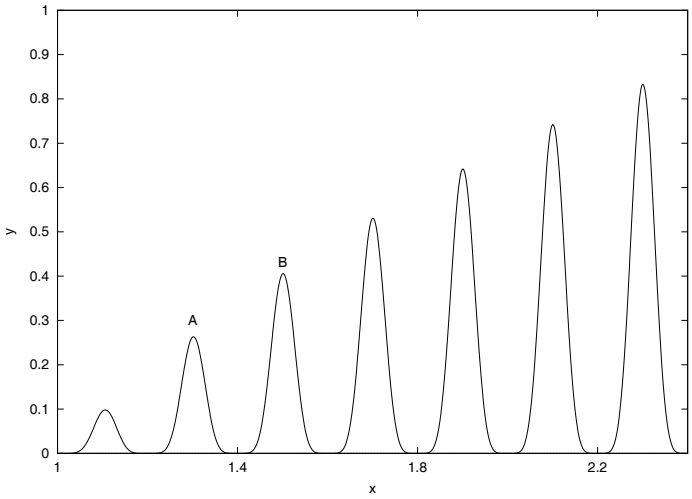


Figure 5.1: $f(x) = \ln(x)\sin^6(5\pi x)$. Note that the function consists of multiple rising peaks located close to each other. A particle close to peak A may be influenced to rather move to peak B based on the influence of its neighborhood.

that detected the potential solution and creating a subswarm from it. The main swarm thus shrinks as subswarms are grown from it. The algorithm is considered to have converged when subswarms no longer improve on the solutions that they represent. The NichePSO algorithm is summarized in figure 5.2.

In the following sections, each step of the algorithm is discussed in detail.

5.2.1 Initialization

The general location of potential solutions in a search space may not always be known in advance. Therefore, it is a good policy to distribute particles uniformly throughout the search space before learning commences. To ensure a uniform distribution, NichePSO uses *Faure*-sequences to generate initial particle positions. An efficient way of calculating Faure-sequences is given in [86]. Other pseudo-random uniform number generators, such as Sobol-sequences [77], may also be used.

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

Figure 5.2: NichePSO Algorithm

5.2.2 Main Swarm Training

In the *nbest* algorithm, overlapping particle neighborhoods discourage convergence on local optima, such as the ascending maxima shown in figure 5.1. To this end, NichePSO uses a technique that frees a particle from the influence of a neighborhood or global best term in the velocity update equation. When a particle considers only its own 'history and experiences', in the form of a personal best, it can convergence on an optimum that does not have global optimal fitness, as it is not drawn to a position in the search space that has better fitness as a result of the traversal of another particle. This search approach has been previously investigated by Kennedy [47]. It was given in equation (2.20), repeated here for clarity:

$$v_{i,j}(t+1) = wv_{i,j}(t) + c_1 r_{1,j}(t)(y_{i,j}(t) - x_{i,j}(t)) \tag{5.1}$$

Kennedy referred to update equation (5.1) as the *cognition only* model, in recognition of the fact that only a conscience factor, in the form of the personal best $\mathbf{y}_i$, is used in the update. No social information, such as the *global best* solution in the *gbest* and *lbest* algorithms, will influence position updates. This arrangement allows each particle to perform a local search.

5.2.3 Identification of Niches

A fundamental question when searching for different niches, is how to identify them. The niching algorithm proposed by Parsopoulos *et al* (see section 3.4) uses a threshold value ϵ, such that when a particle i's fitness at a position $\mathbf{x}_i$ becomes *less* than the threshold, i.e. when

$$f(\mathbf{x}_i) < \epsilon$$

for a minimization problem, the particle is removed from the swarm and labelled as a potential global solution. Immediately thereafter, the objective function's landscape is stretched to avoid additional and unnecessary exploration of this area surrounding the discovered solution. If the isolated particle's fitness is not close to a desired level, the solution can be refined by searching the surrounding function landscape with the addition of more particles. This approach proved to be effective when considering Parsopoulos *et al*'s results. The threshold parameter ϵ is however subject to fine tuning,

and locating good solutions depends strongly on the objective function's landscape and dimensionality.

To avoid the use of this tunable parameter, NichePSO uses a similar approach that monitors changes in the fitness of a particle. If a particle's fitness shows little change over a number of iterations of the learning algorithm, a *subswarm* is created with the particle and its closest topological neighbor. More formally, the standard deviation in particle i's fitness, σ_i, is tracked over a number of iterations, e_σ, where e_σ was set to 3 in the experiments conducted in section 5.3. When $\sigma_i < \delta$, a subswarm may be created with particle i and its closest neighbor. To avoid problem dependence, σ_i is normalized according to the range of the search space, commonly referred to as x_{min} and x_{max} in PSO literature. This approach can find *local* minima, for which $\sigma_i < \delta$ holds. If local minima are undesired, the fitness of a particle can be compared to a threshold to ensure that the solution meets a minimum fitness criterion.

The 'closest neighbor' to particle i's position $\mathbf{x}_i$ is simply the particle c with position $\mathbf{x}_c$, where

$$c = \arg\min_j \{\|\mathbf{x}_i - \mathbf{x}_j\|\}$$

with $1 \leq i, j \leq s$, $i \neq j$ and s is the size of the main swarm. Subswarms are optimized independent from the main swarm, in the same search space. The following sections present measures that are put into place to ensure that search efforts are not duplicated on the same solutions.

5.2.4 Absorption of Particles into a Subswarm

When a particle is still a member of the main swarm, it has no knowledge of subswarms that may have been created during the execution of the NichePSO learning algorithm. It is therefore quite likely that a particle may venture into an area of the search space that is being independently optimized by a subswarm. Such particles are merged with the corresponding subswarm, based on the following suppositions:

- Inclusion of a particle that traverses the search space of an existing subswarm may expand the diversity of the subswarm, thereby more rapidly leading to solutions with better fitness.

- An individual particle moving towards a solution on which a subswarm is working, will make much slower progress than what would have been the case had social information been available to ensure that position updates move towards the particle's *known* favorable solution.

To facilitate merging, particles are absorbed into a subswarm when they move 'into' the subswarm. That is, a particle i will be absorbed into a subswarm S_j when

$$\|\mathbf{x}_i - \hat{\mathbf{y}}_{S_j}\| \leq R_j \tag{5.2}$$

where R_j signifies the radius of subswarm S_j, and is defined as

$$R_j = \max\{\|\hat{\mathbf{y}}_{S_j} - \mathbf{x}_{S_{j,i}}\|\} \tag{5.3}$$

$\mathbf{x}_{S_{j,i}}$ represents all particles in S_j subject to $i \neq g$, $\hat{\mathbf{y}}_{S_j}$ represents the global best particle in S_j. Generally, subswarms have small radii, due to the homogeneous nature of the positions represented by their particles. Therefore, when a particle i moves into the hyper-sphere defined by a subswarm's global best particle and radius, it is unlikely that it would move away from the possible solution maintained by the subswarm. If the absorption step was absent from the algorithm, i will first have to be considered for a subswarm and successfully made part of one, before it can merge with S_j. If no other particles occur in the same portion of the search space, a subswarm containing i will never be created, and the potential solution it represents will never be considered. If i is merged with a particle in a similar situation, but that occurs in a vastly different position in the search space, the algorithm's convergence would be impaired.

5.2.5 Merging Subswarms

A subswarm is created by removing a particle that represents an acceptable candidate solution from the main swarm, as well as a particle that lies closest to it in the search space, and to group these into a subswarm. From this rule, it follows that particles in subswarms all represent similar solutions. This can lead to subswarms with radii that are very small, and even radii approximating zero. Consequently, when a particle approaches a potential solution, it may not necessarily be absorbed into a subswarm that

is already optimizing the particular solution. If the particle has an acceptable fitness, another subswarm will be created on its position in the search space. If two solutions are very similar, a single subswarm will be created to optimize both solutions. Eventually, only one of these solutions will be found. This introduces a dilemma, as multiple swarms will attempt to optimize the same solution. To alleviate this, subswarms may be merged when the hyper-space defined by their particle positions and radii intersect in the search space. When swarms are merged, the newly created swarm benefits from the extensive social information present in the parent swarms. Accordingly, superfluous local traversal of the search space is avoided. Formally, two subswarms S_{j1} and S_{j2} *intersect*, when

$$\|\hat{\mathbf{y}}_{S_{j1}} - \hat{\mathbf{y}}_{S_{j2}}\| < (R_{j1} + R_{j2}) \tag{5.4}$$

When $R_j = 0$ holds for subswarm S_j, all particles in S_j represent the same candidate solution. If this condition holds for both swarms under consideration, equation (5.4) fails to detect the presence of multiple subswarms in the same niche. Consequently, when two swarms, S_{j1} and S_{j2} do not satisfy equation (5.4), because $R_{j1} = R_{j2} = 0$[1], they can be merged when

$$\|\hat{\mathbf{y}}_{S_{j1}} - \hat{\mathbf{y}}_{S_{j2}}\| < \mu \tag{5.5}$$

As with δ, μ can be an appreciably small number, such as 10^{-3}, to ensure that two swarms are sufficiently similar. To avoid having to tune μ over the range of the search space under consideration, $\|\hat{\mathbf{y}}_{S_{j1}} - \hat{\mathbf{y}}_{S_{j2}}\|$ is normalized to the interval $[0, 1]$. S_{j1} and S_{j2} are merged by creating a new subswarm consisting of all S_{j1} and S_{j2}'s particles. The influence of different values of μ, and an upper bound on it, are discussed in section 5.4.1.

5.2.6 The GCPSO Algorithm

The GCPSO algorithm was presented and discussed in section 2.7.1. The subswarm creation technique presented in section 5.2.5 always yields swarms that initially consist of two particles. Training such a small swarm with the *gbest* algorithm, especially when

[1]Since position updates in PSO is a stochastic process, it is practically safer to consider the situation where $R_{j1} \approx 0$ and $R_{j2} \approx 0$.

its particles are topologically highly similar, may lead to swarm stagnation, forcing the subswarm to convergence on a suboptimal solution. GCPSO puts measures in place that ensure that a swarm does not stagnate. For a definition of what is meant by swarm stagnation, as well as a rigorous analysis of why GCPSO is necessary, the reader is referred to section 5.4.3.

5.2.7 Stopping Criteria

When each individual subswarm has located a solution and stably maintained it for a number of training iterations, the NichePSO may be considered to have converged. The following stopping criteria are implemented:

- Each swarm must converge on a unique solution. Typically, a subswarm is considered to have converged when its global best solution's fitness is either above or below a threshold value, depending on whether the fitness function describes a maximization or minimization problem. Fitness threshold criteria cannot however detect acceptable solutions in a multimodal fitness function where local *and* global maxima exist. Local maxima are never considered to be acceptable solutions, as their fitness do not necessarily adhere to possibly strict threshold values. Any algorithm that therefore depends solely on threshold values will fail to converge. Therefore, the change in particle positions are tracked over a number of iterations. If discernible change occurs in their positions, such as may be detected by considering their variance over a small number of training iterations, the subswarm may be considered to have converged.
- The algorithm is stopped after a maximum number of training iterations.

5.3 Experimental Results

This section presents experimental results obtained on a set of well-known multimodal functions. These functions have been extensively used in the testing of a number of GA niching techniques [4, 32, 37, 61]. Test functions are defined in section 5.3.1, and experimental results and a discussion thereof follows in section 5.3.2.

5.3.1 Test Functions

NichePSO is tested on a number of multimodal functions, where the goal is to identify all optima. These functions were originally introduced by Goldberg and Richardson to test fitness sharing [32] and have also been used by Beasley *et al* to evaluate their sequential niching algorithm [4]. Figure 5.3 illustrates functions F1 to F4, defined as:

$$F1(x) = \sin^6(5\pi x) \tag{5.6}$$

$$F2(x) = \left(e^{-2\log(2)\times\left(\frac{x-0.1}{0.8}\right)^2}\right) \times \sin^6(5\pi x) \tag{5.7}$$

$$F3(x) = \sin^6(5\pi(x^{3/4} - 0.05)) \tag{5.8}$$

$$F4(x) = \left(e^{-2\log(2)\times\left(\frac{x-0.08}{0.854}\right)^2}\right) \times \sin^6(5\pi(x^{3/4} - 0.05)) \tag{5.9}$$

Functions $F1$ and $F3$ both have 5 maxima with a function value of 1.0. In $F1$, maxima are evenly spaced, while in $F3$ maxima are unevenly spaced. In $F2$ and $F4$, local and global peaks exist at the same x-positions as in $F1$ and $F3$, but their fitness magnitudes decrease exponentially. Functions $F1$ to $F4$ are investigated in the range $x \in [0, 1]$. For each of the functions, maxima occur at the following x positions:

$F1$	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

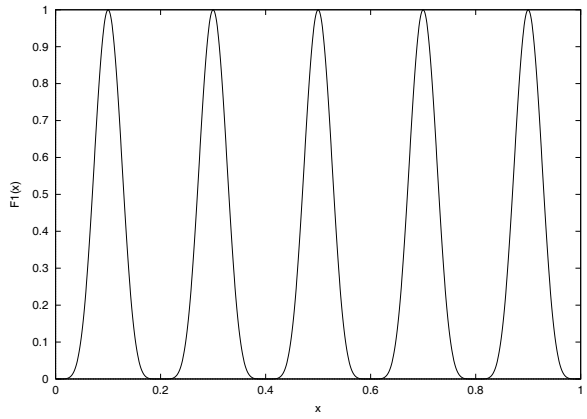
Function $F5$, the modified Himmelblau function (see figure 5.4), is defined as

$$F5(x, y) = 200 - (x^2 + y - 11)^2 - (x + y^2 - 7)^2 \tag{5.10}$$

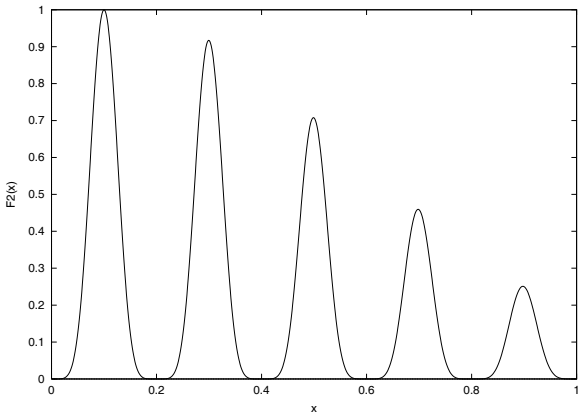
$F5$ has 4 equal maxima with $F5(x, y) = 200$. Maxima are located at $(-2.81, 3.13)$, $(3.0, 2.0)$, $(3.58, -1.85)$ and $(-3.78, -3.28)$.

5.3.2 Results

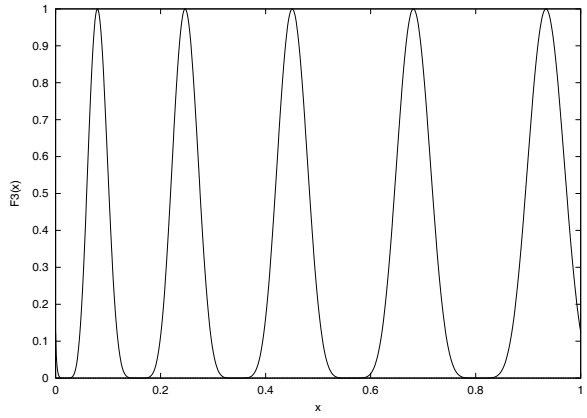
For each of the 5 test functions, 30 simulations were done with the NichePSO algorithm with $c_1 = c_2 = 1.2$. The inertia weight w was scaled linearly from 0.7 to 0.1 over a



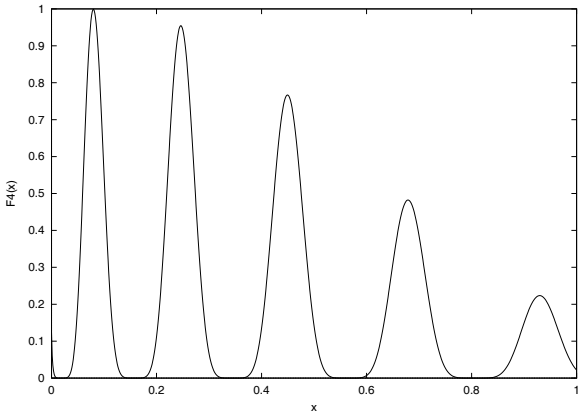
(a) Function F1



(b) Function F2



(c) Function F3



(d) Function F4

Figure 5.3: NichePSO Test Functions

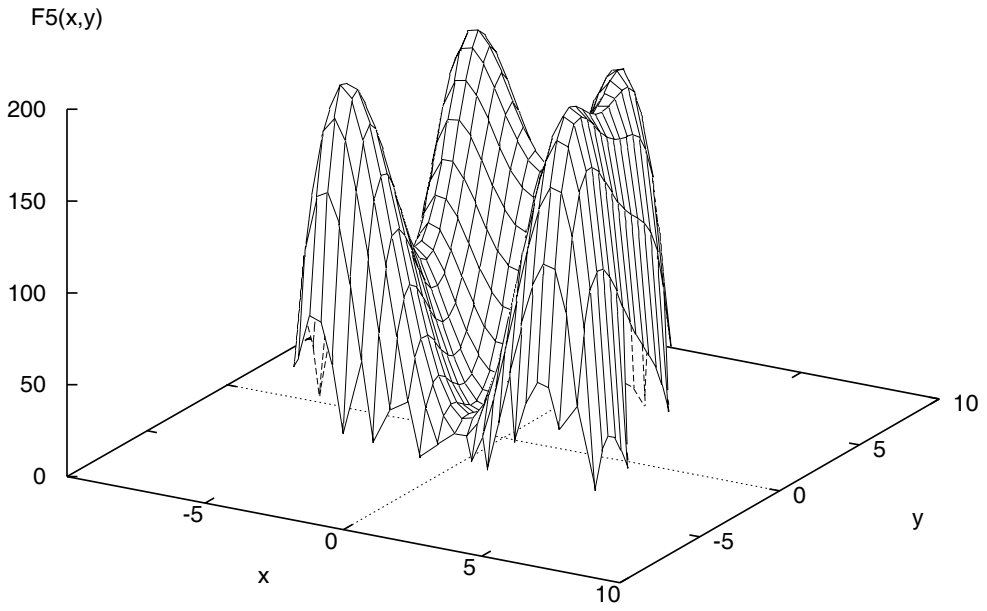


Figure 5.4: The Himmelblau function

maximum of 2000 iterations of the NichePSO algorithm. A single NichePSO iteration is defined as performing steps 2 to 8 in figure 5.2 once. A gradually decreasing inertia weight lets particles make slower velocity and position updates, at the same time ensuring that they follow convergent trajectories [87]. Table 5.1 reports NichePSO parameter settings, as well as limits on the search space ranges of each test function. $|S|$ denotes the initial number of particles in the main swarm before any niche subswarms were created; μ is the subswarm merging threshold, and δ is the subswarm creation threshold. For functions $F1$ to $F4$, a particle consists simply of a potential x value. For function $F5$, a particle represents an (x, y) position. NichePSO is evaluated according to

- *Accuracy:* Thus how close the discovered optima are to the actual solutions; and
- *Success consistency:* The proportion of the experiments that found all optima.

Function	δ	μ	$\|\mathbf{S}\|$	x_{min}	$x_{max} = v_{max}$
$F1$	0.0001	0.001	30	0.0	1.0
$F2$	0.0001	0.001	30	0.0	1.0
$F3$	0.0001	0.001	30	0.0	1.0
$F4$	0.0001	0.001	30	0.0	1.0
$F5$	0.0001	0.01	20	-5.0	5.0

Table 5.1: NichePSO Parameter Settings

The parameter values for δ and μ presented in table 5.1 have been experimentally found to be effective.

Table 5.2 reports the mean and standard deviation in fitness of all particles in all subswarms. Fitness here is simply defined as the function value $f(x)$ for each of the test problems. Only global optima are considered (suboptimal solutions in $F2$ and $F4$ are not taken into account). *%Converged* signifies the percentage of experiments that successfully located all the maxima. NichePSO successfully located all global maxima of all the functions tested. For functions $F2$ and $F4$, NichePSO located the global maximum in all cases, but did not find all local maxima for all simulations. This explains the relatively large difference in fitness between functions $F1$ and $F3$, and functions $F2$ and $F4$.

Table 5.2: Performance Results

Function	**Fitness**	**Deviation**	**NichePSO % Converged**
$F1$	$7.68E-05$	$2.20E-04$	100%
$F2$	$9.12E-02$	$6.43E-02$	93%
$F3$	$5.95E-06$	$4.86E-05$	100%
$F4$	$8.07E-02$	$6.68E-02$	93%
$F5$	$4.78E-06$	$1.03E-05$	100%

5.4 Analysis

Unimodal optimization techniques, such as the standard PSO and GAs, fail to locate multiple solutions to multimodal problems because of their inherent unimodal optimization nature. These algorithms need to be extended to facilitate niching and speciation abstractions. NichePSO is no exception – although the essence of the original PSO is retained, a number of extensions were made. The motivations and reasoning behind these extensions are presented, justified and investigated in this section. The following issues are considered:

- The algorithm's sensitivity to the niching parameters, μ and δ,
- the performance of GCPSO compared to *gbest*, and
- the relationship between the initial swarm size and the number of solutions in a multimodal fitness function.

Finally, the scalability of NichePSO on highly multimodal functions are considered.

5.4.1 Sensitivity to Changes in μ

Each subswarm created by the NichePSO algorithm can be seen as a hyper-sphere in the search space. The hyper-sphere's radius is determined by the Euclidean distance between the swarm's global best position and the particle in the swarm that lies furthest from it. Two subswarms are merged when the two conceptual hyper-spheres that they represent overlap. When all particles in a swarm have converged on a single solution, a swarm will have an effective radius of zero. In such a situation, equation (5.4) fails to allow similar

Solutions involved	**Distance between solutions**
$\|\mathbf{A} - \mathbf{B}\|$	0.714
$\|\mathbf{B} - \mathbf{C}\|$	0.622
$\|\mathbf{C} - \mathbf{D}\|$	0.675
$\|\mathbf{A} - \mathbf{D}\|$	0.493

Table 5.3: Normalized Inter-solution Distances for Function $F5$

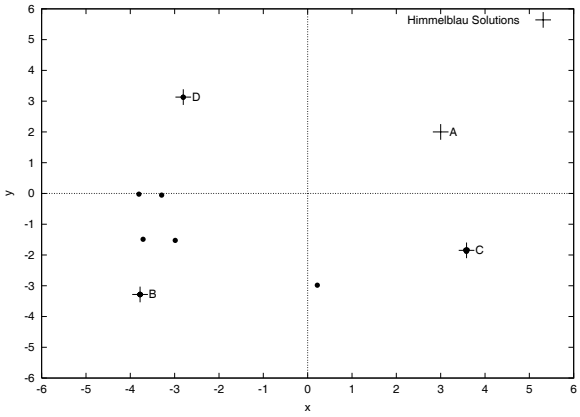
swarms to be merged. Therefore, the use of the μ parameter was introduced in equation (5.5), to allow virtually identical swarms to be merged when they occupy positions that are approximately similar. Large values of μ allow swarms that settle on different solutions, to merge. If two swarms that correctly represent different solutions are merged, the newly created swarm eventually converges on only one of the possible solutions, because of the subswarm optimization technique: The GCPSO algorithm searches for a single solution.

Figures 5.5(a), 5.5(b) and 5.5(c) illustrate the effect that different μ values have on the convergence capabilities of NichePSO, tested on function $F5$. Each particle in the swarm is represented by a '•'. The position of each solution is indicated by a '+', and labelled by a letter of the alphabet.

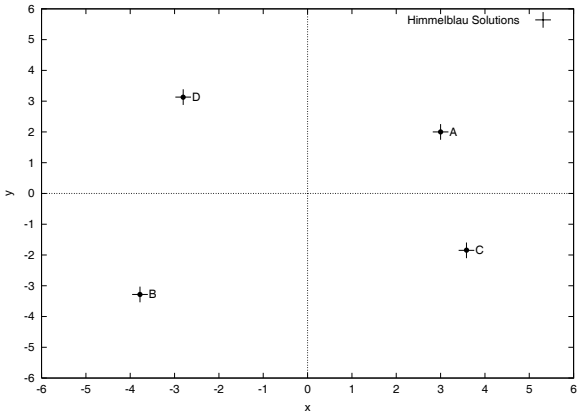
The 'goodness' of a particle's position can be determined by its proximity to the indicated solutions. Table 5.3 presents the normalized distances between the known solutions for function $F5$. The symbols correspond to those used in figures 5.5(a), 5.5(b) and 5.5(c). For $\mu = 0.5$, the NichePSO algorithm did not find all maxima for function $F5$ (see figure 5.5(a)). From table 5.3, the normalized distance between solutions A and D is 0.493. Swarms that represent solutions A and D were therefore merged, as their inter-niche solution distance was less than the threshold value μ. For μ-values less than 0.5, the algorithm successfully located all solutions (see for example figure 5.5(b)). For extremely small μ-values, NichePSO still successfully located all solutions, but not all swarms that were positioned on the same solutions were merged (see solution D in figure 5.5(c)). Swarms congregated around the solutions, but due to the 'strict' merging threshold, they could only be merged when virtually identical.

Figures 5.6(a) and 5.6(b) plot the mean number of solutions found by NichePSO for $F5$, $F1$ and $F3$ respectively, for different μ values. For both functions $F1$ and $F3$, NichePSO located all solutions when $\mu \leq 0.1$ (see figure 5.6(b)).

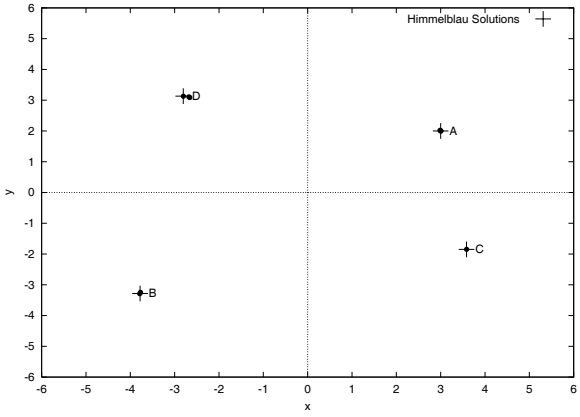
From figure 5.6, an upper bound on μ can be derived: μ should not be greater than the lowest inter-niche distance. The upper bound is similar to the assumptions made about the inter-niche distance, $2\sigma_{sh}$, in Goldberg's fitness sharing technique [32], and the niche radius r in Beasley *et al*'s sequential niching technique [4].



(a) $\mu = 0.5$

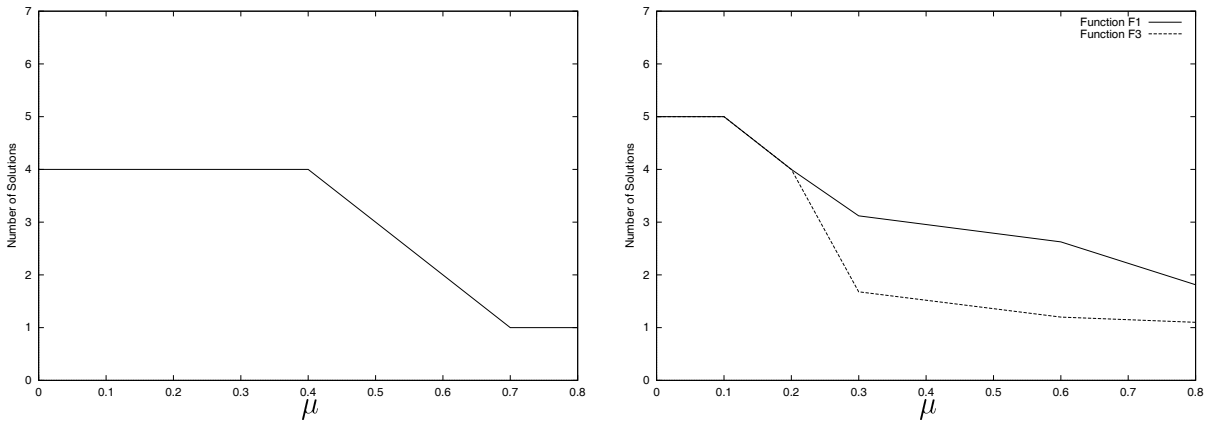


(b) $\mu = 0.4$



(c) $\mu = 0.0001$

Figure 5.5: Effect of changes in μ



(a) Number of solutions vs. μ for function $F5$

(b) Number of solutions vs. μ for functions $F1$ and $F3$

Figure 5.6: Number of solutions vs. μ for functions F1 and F3

5.4.2 Sensitivity to Changes in δ

To identify new potential solutions, the NichePSO algorithm monitors changes in the particles of the main swarm. If any particle in the main swarm exhibits very little change in its position over a number of iterations of the algorithm, the particle has approached an optimum position. The optimum may be a local or global optimum. An effective measure to detect small changes in a particle i's position is to monitor the standard deviation σ_i in particle i's fitness over a number of training iterations, e_σ. When particle i's variance in fitness becomes less than a threshold value δ, a new subswarm is created using particle i and its closest neighbor. A particle exhibits this behavior only when it is approaching a solution and has a low velocity, or when it is oscillating around a potential solution.

Different δ values were tested for functions $F1$, $F3$ and $F5$. Figure 5.7 plots the mean number of fitness function evaluations over 30 simulations for each test function against different δ settings. The number of fitness function evaluations are considered to illustrate that some δ values increase the time required for the algorithm to converge. A simulation was considered to have converged when all its subswarms had a fitness

less than 10^{-4}. For relatively large δ values ($\delta > 0.1$), NichePSO initially easily created subswarms with any particle that remotely exhibited stagnating behavior. In this context, 'stagnating behavior' indicates that a particle slowed down, and that it occupied similar positions in the search space over consecutive algorithm iterations. For small δ values ($\delta < 0.1$), particles were required to be more stationary before being considered for a subswarm. A different interpretation is that particles had to be very sure of a solution, before a subswarm was created. As indicated in figure 5.7, smaller δ values effected a slight increase in the number of fitness function evaluations required before the algorithm converged. From figure 5.7 only broad general trends can be seen. Each function appears to have an optimal δ value. For the test functions, these values are reported in table 5.4.

Figure 5.7 illustrates that NichePSO is not dependent on a finely tuned δ. Fervent subswarm creation with a 'high' δ value will be negated by the merging of similar swarms. Very small δ values ($\delta < 0.01$) leads to a minor performance penalty, but when compared to NichePSO's performance on higher δ values, the cost is low. Without exception, NichePSO successfully located *all* solutions to the test functions for all δ values used.

5.4.3 The Subswarm Optimization Technique

The NichePSO algorithm uses the GCPSO technique (refer to section 2.7.1) as subswarm optimization technique. This section compares two implementations of NichePSO: one using *gbest* and the other using GCPSO.

When considering the particle position update given in equation (2.12), it is clear that when, for a particle i, its position $\mathbf{x}_i(t)$ at time step t becomes close to its personal best position $\mathbf{y}_i(t)$ and the global best position $\hat{\mathbf{y}}(t)$, the velocity update for the next

Test Function	**Optimal δ**
F1	0.01
F3	0.1
F5	0.2

Table 5.4: Optimal δ values

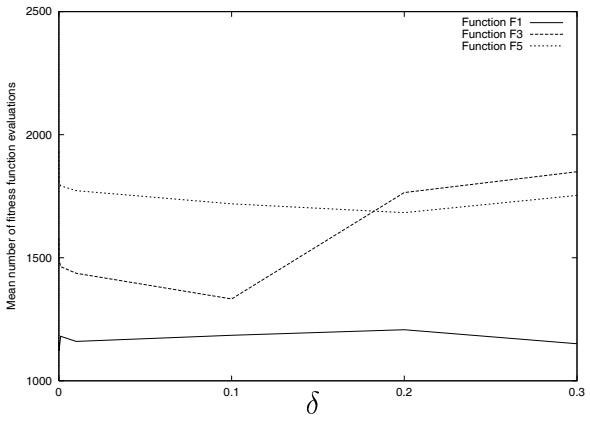


Figure 5.7: Mean number of fitness function evaluations required for different δ values

iteration of the algorithm, $\mathbf{v}_i(t+1)$ depends only on previous velocity values and the inertia weight w. A small $\mathbf{v}_i(t+1)$ dictates negligible change in a particle's position. The particle will therefore stagnate on its current position; $\hat{\mathbf{y}}(t)$ does not necessarily represent an optimum, but only the best solution found thus far by all particles in the search space. When a particle swarm consists of only two particles, as is frequently the case when subswarms are created with NichePSO, it may occur that these swarms stagnate almost immediately. The situation can be symbolically explained as follows:

With two particles, p and q, in a newly created subswarm, one of these particles, say p, immediately represents the global best position of the swarm. It also holds that the personal best position $\mathbf{y}_p$ of p is equal to the swarm's global best. This is an obvious assumption when considering that the global best position is identified as a personal best position of one of the particles in the swarm. With $\mathbf{x}_p = \mathbf{y}_p = \hat{\mathbf{y}}$, particle p's initial velocity update is then effectively reduced to

$$\mathbf{v}_p(t+1) \quad = \quad w\mathbf{v}_p(t) \tag{5.11}$$

Particle p's initial traversal of the search space therefore solely depends on its velocity vector $\mathbf{v}_p(0)$ and the value of the inertia weight w. When a subswarm is created, each particle in the new swarm not only retains its position vector, as this was the basis for its selection, but also its velocity vector. This ensures that the particle continues on its path to a local optimum. If the particle was already close to a potential solution, the magnitude of its velocity vector would be a value close to zero. Particle p will

therefore not easily move around in search space. Particle q was chosen to be the second particle in the subswarm, because it was the closest particle to p when the subswarm was created. This implies, as stated above, that $\mathbf{y}_p$ will always be considered over $\mathbf{y}_q$ for the swarm's initial global position, and consequently, that $\mathbf{x}_q$ will move towards $\mathbf{x}_p$. When $\mathbf{y}_p = \mathbf{y}_q = \hat{\mathbf{y}}$, the following conditions will occur:

- $\mathbf{x}_p \approx \mathbf{x}_q$, and
- $\mathbf{v}_p$ and $\mathbf{v}_q$ will approach zero.

Under these circumstances, no further learning takes place, and exploration of the search space is minimal. Again, it should be noted that the assumption that $\hat{\mathbf{y}}$ represents a global solution, cannot be made. To detect and avoid the described situation, the GCPSO algorithm was used. GCPSO uses adapted velocity and position update equations for the global best particle in a swarm (in this case particle p), that allows efficient local traversal of the search space. GCPSO avoids stagnation by moving the global best particle, until an optimum has been located.

Table 5.5 presents experimental results that compare the performance of GCPSO with *gbest* as subswarm optimization technique. The $\%Convergence$ column expresses an average success rate for finding *all* solutions of a test function over 30 simulations. It is clear that GCPSO was better suited towards maintaining niches than *gbest*. Experimental results obtained showed that for all functions, when using *gbest*, it frequently occurred that subswarms were formed that consisted of only two particles. Such swarms quickly stagnated on suboptimal locations. When a subswarm did not move, the probability that a particle in the main swarm would pass over it and be absorbed into the subswarm, was severely reduced. A subswarm consisting of only two particles that have stagnated on a suboptimal solution, is of no use. Results reported in table 5.5 ignored two particle swarms and reports only whether an experiment did manage to locate the actual solutions.

When several local and global optima exist in close proximity to each other in a function, GCPSO tends to be biased towards the global optima. This behavior is dictated by equations (2.16) and (2.15). GCPSO's addition of a random factor to the swarm's best particle position may place the particle closer to a global solution, hence forcing the

Table 5.5: % Convergence of experiments for GCPSO and *gbest*

Test Problem	**% Convergence: GCPSO**	**% Convergence: *gbest***
$F1$	100%	76%
$F2$	93%	66%
$F3$	100%	83%
$F4$	93%	86%
$F5$	100%	86%

subswarm to move towards the global solution that may already be well-represented by other subswarms. *gbest* does not exhibit this behavior, since the best particle's position is not modified.

5.4.4 Relationship between $|S|$ and the Number of Solutions

This section investigates the relationship between the swarm size $|S|$ and the number of optima, a, in a multimodal function.

Figures 5.8 and 5.9 present experimental results that compare the number of solutions found and the number of fitness function evaluations required for different swarm sizes. Reported results are means over 30 simulations for functions $F1$, $F3$ and $F5$. Trivially,

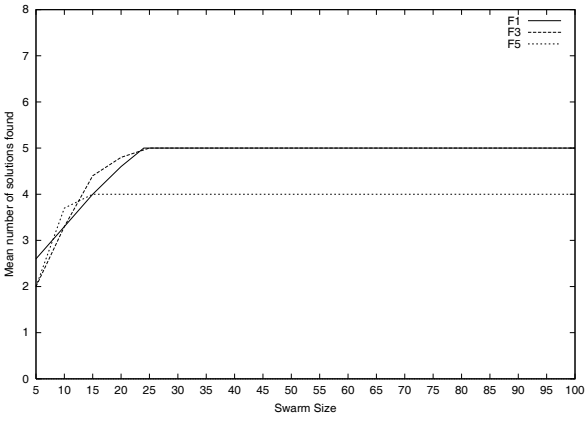


Figure 5.8: Relationship between different swarm sizes and the number of solutions located.

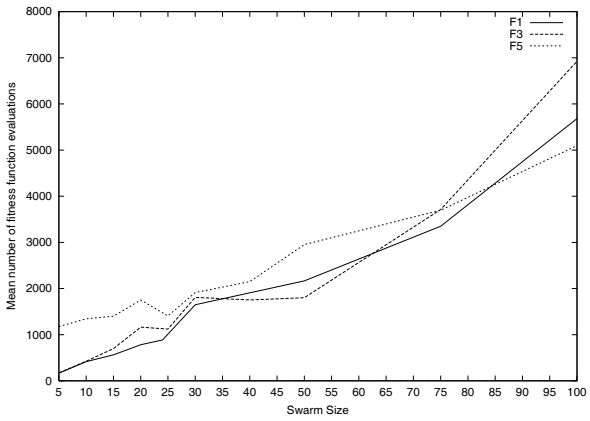


Figure 5.9: The mean number of fitness function evaluations required for different swarm sizes.

NichePSO failed to locate all solutions when $|S| < a$. When $|S| < 2a$, NichePSO also did not locate all the solutions. Since the subswarm creation technique needed two particles to create a subswarm, intuitively, $2a$ would have been expected to be a sufficient swarm size. This was however not the case. This situation can be clarified when considering the distribution of particles, and the fact that velocity vectors were initialized randomly. No 'directional-bias' is introduced by forcing velocity vectors to lead a particle into a specific direction, towards a solution. If possible solutions are not known in advance, this would not be possible. A particle could therefore be initialized close to a solution, but an initial velocity value may cause it to move away from the possible solution towards another solution, where it could eventually settle. For function $F1$, when $|S| \geq 25$, NichePSO successfully located and stably maintained all solutions. For all the tested functions, a swarm of size $|S| \geq a^2$ managed to locate all solutions. It serves to confirm the suspected relationship between the number of solutions in a multi-modal problem and the swarm size. The relationship $|S| \geq a^2$ prescribes acceptable swarm sizes for the test problems considered in this section. Results in section 5.4.5 show it is however not a general condition. Equation (5.15) in section 5.4.5 presents a more general formulation of the relationship between the number of solutions and swarm size.

Figure 5.9 shows the mean number of fitness function evaluations required for the different swarm sizes used above. As expected, the number of fitness function evaluations

steadily increases as the swarm size grows. The trend illustrates that the use of larger swarms does not necessarily benefit the optimization process:

- Convergence speed is not improved.
- Smaller swarms, subject to the condition identified above, yield the same results with lower complexity.

5.4.5 Scalability of NichePSO

The results obtained in section 5.3 showed NichePSO to effectively solve multimodal optimization problems. This section presents empirical results that investigate the scalability on NichePSO to massively multimodal domains. NichePSO was tested on the following two multimodal functions:

Griewank function:

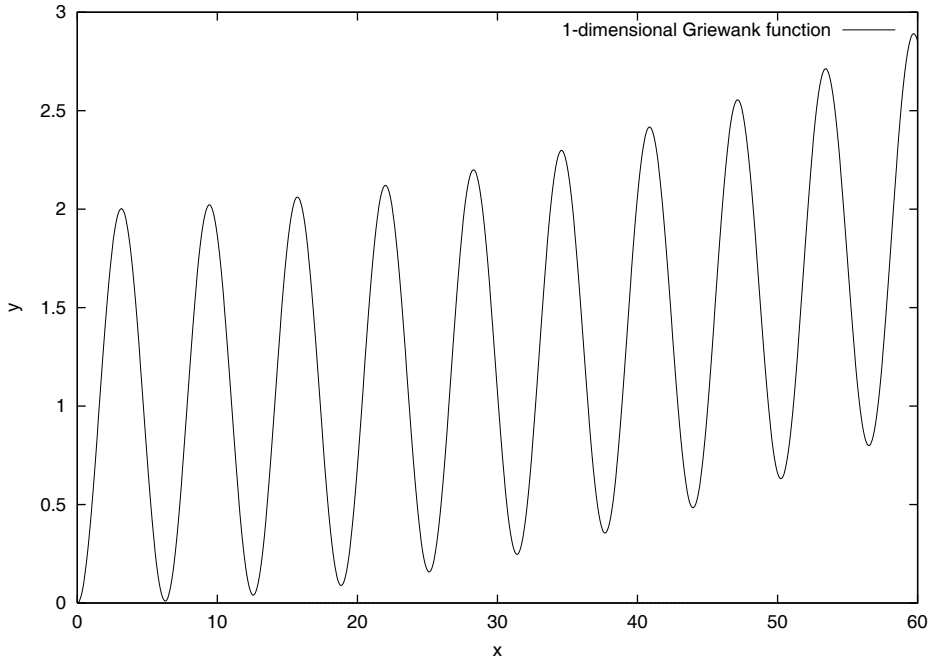
$$f(\mathbf{x}) = \left(\frac{1}{4000}\sum_{i=1}^{n} x_i^2\right) - \left(\prod_{i=1}^{n} \cos\left(\frac{x_i}{\sqrt{i}}\right)\right) + 1 \tag{5.12}$$

Rastrigin function:

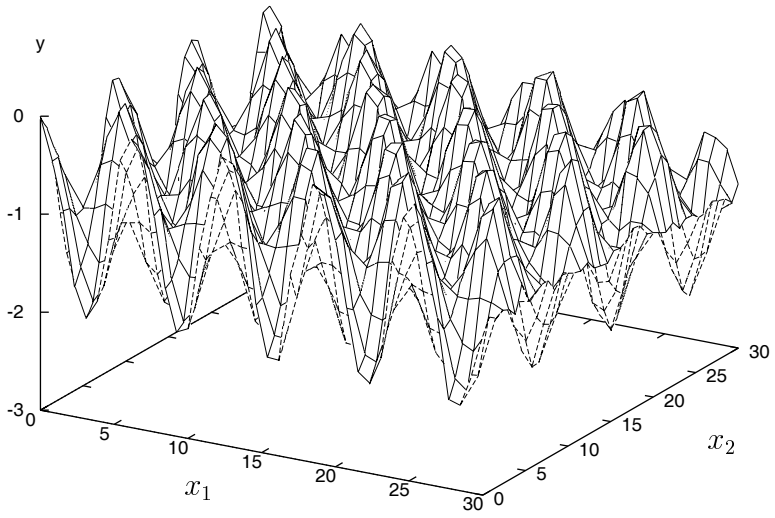
$$f(\mathbf{x}) = \sum_{i=1}^{n} \left[x_i^2 - 10\cos\left(2\pi x_i\right) + 10\right] \tag{5.13}$$

These functions are massively multimodal. Both contain a single global minimum at the origin of the n-dimensional real-valued space in which they are defined. For each of the functions, the number of minima increase exponentially, as can be seen from the one and two dimensional plots given in figures 5.10 and 5.11. Figures 5.10(b) and 5.11(b) are drawn inverted to more clearly illustrate the multimodal nature of the function surfaces. The goal of the experiments were to ascertain whether increased dimensionality and large numbers of optima degraded the performance of NichePSO.

For each of the test functions, 10 experiments were performed with the NichePSO algorithm. Initial swarms sizes used were as listed in tables 5.6 and 5.7. For all experiments, the inertia weight w was scaled linearly from 0.7 to 0.1 over a maximum of 2000 training iterations. The acceleration coefficients were set to $c_1 = c_2 = 1.2$. NichePSO

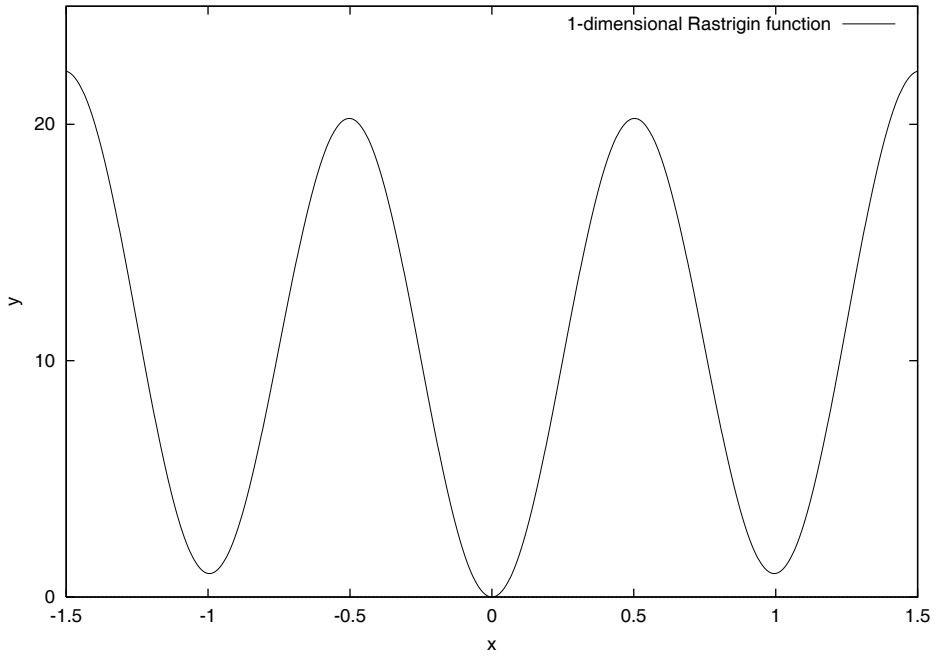


(a) One-dimensional Griewank function

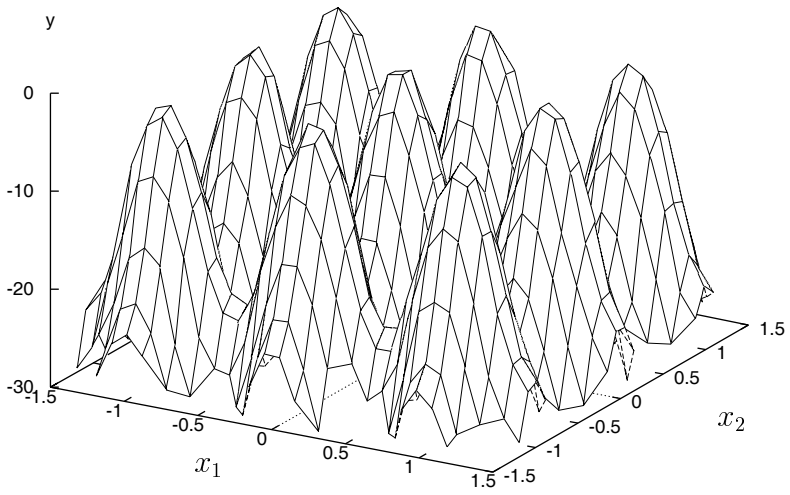


(b) Two-dimensional Griewank function

Figure 5.10: Griewank function



(a) One-dimensional Rastrigin function



(b) Two-dimensional Rastrigin function

Figure 5.11: Rastrigin function

Dimensions (n)	Number of solutions (a)	Swarm Size ($\|S\|$)	% Accuracy
1	5	20	100.00%
2	25	100	100.00%
3	625	2500	94.75%

Table 5.6: Performance on the Griewank function

Dimensions (n)	Number of solutions (a)	Swarm Size ($\|S\|$)	% Accuracy
1	3	9	100.00%
2	9	36	100.00%
3	27	108	97.45%
4	81	324	97.08%
5	243	972	92.00%

Table 5.7: Performance on the Rastrigin function

parameters were set as $\mu = 0.001$ and $\delta = 0.1$. The Griewank function was investigated in the range $[-28, 28]^n$, and the Rastrigin function in the range $[-\frac{3}{2}, \frac{3}{2}]^n$. Tables 5.6 and 5.7 present performance results of NichePSO on the two test functions.

The following observations can be made:

- Given the sharp increase in the number of optima, NichePSO gave consistent performance, with slight degradation as the number of dimensions increased. It should be taken into account that a linear increase in the number of dimensions is coupled with an *exponential* increase in the number of solutions.

- Using the exponential relationship $|S| = a^2$ suggested in section 5.4.4, is computationally very expensive. As an example, the 5-dimensional Rastrigin function with 243 solutions would require a swarm of $59,049$ particles! As shown in tables 5.6 and 5.7, the relationship between the number of solutions (a) and the swarm size ($|S|$) was kept at

$$|S| = 4a \tag{5.14}$$

  This relationship is computationally more tractable. It also shows that $|S| = a^2$ does not represent a lower bound on the relationship between swarm size and num-

ber of solutions. Consequently, the relationship between swarm size and number of solutions would more likely be expressed as

$$|S| = c \cdot a^q \tag{5.15}$$

where c is a constant, and $1 \leq q \leq 2$. Further experimentation would be required to empirically estimate ideal values for c and q.

5.5 Conclusion

Swarm intelligence algorithms such as particle swarm optimizers, present a real and viable alternative to existing numerical optimization techniques. Population based optimization techniques can rapidly search large and convoluted search spaces and less likely to be sensitive to suboptimal solutions. The standard *gbest* and *lbest* PSO approaches share information about a best solution found by the swarm or a neighborhood of particles. Sharing this information introduces a bias in the swarm's search, forcing it to converge on a single solution. When the influence of a current best solution is removed, each particle traverses the search space individually, using no expierential knowledge of its peers.

With NichePSO, a subswarm is created when a possible solution is detected at a particle's location. The subswarm is then responsible for traversing the search space in the vicinity of the potential solution to find an optimal location. This pseudo-memetic approach is called NichePSO. Experimental results obtained on a set of multimodal functions showed that NichePSO successfully located and maintained multiple optimal solutions. Several parameter optimization issues, related to NichePSO, were addressed. Suggestions were made as to potential values for tunable parameters. A scalability study carried out on highly multimodal functions, commonly used in the study of PSO algorithms, were used to demonstrate that NichePSO scales acceptably on high dimensional problems.

Chapter 6

A Comparative Analysis of Niching Techniques

This chapter presents an empirical comparison between the newly introduced *nbest* and NichePSO algorithms and two existing GA niching techniques.

6.1 Introduction

A plethora of evolutionary niching and speciation techniques are in existence. Chapter 2 presented a summary of some of the more well-known variations. All the presented techniques can be categorized as being either *sequential* or *parallel* niching techniques. Sequential niching locates niches in serialized runs of the same algorithm. Each run of the algorithm that successfully locates a niche/solution, modifies the fitness function to keep subsequent runs from unnecessarily duplicating search efforts. Parallel niching implements measures to concurrently identify and maintain niches (see section 3.2 for a more complete discussion). Both the *nbest* and NichePSO algorithms, introduced in the previous chapters, are parallel niching techniques. The *nbest* algorithm uses

- spatial particle neighborhoods and
- an alternative formulation of the fitness function

to concurrently find and maintain niches, while NichePSO uses *subswarms*.

The introduction of new algorithms in a research field necessitates a comparative analysis to determine whether they offer an advantage, or can be considered as alternatives to existing techniques. Since no existing, unique niching techniques exist in the PSO field, such a study would not be possible. The objective function stretching optimizer introduced by Parsopoulos *et al* is consciously excluded here, for reasons set out in section 3.4. Consequently, it seems appropriate to compare the new PSO techniques to well-known GA based niching techniques. It should be noted that the goal of this study was to develop new, unique PSO based solutions to niching problems. A number of authors have undertaken studies where existing evolutionary optimization approaches were re-factored for PSOs. As was found in chapter 3, this was not possible for GA niching techniques. They cannot be directly mapped to the PSO, due to differences in the behavior of the GA and PSO. These differences were analyzed in section 3.5. It was therefore necessary to develop techniques specifically suited to the dynamics of the PSO. In the rest of this chapter, a comparison is presented between PSO and GA based niching techniques.

6.2 The Algorithms

In order to form a general base for comparison, the PSO niching algorithms are compared to both a

- sequential, and a
- parallel GA niching technique.

As an example of sequential niching, Beasley *et al*'s sequential niching (SN) technique, described in section 3.3.3, is used. SN uses repeated runs of a normal generational genetic algorithm to locate multiple solutions [4]. After each run of a simple GA, the fitness function is adapted to reflect the position of the recently located solution. Subsequent runs of the GA with the modified fitness function avoid areas in the search space where solutions have already been found, forcing the optimizer to explore unknown sections of the search space. SN has been criticized, the biggest concern being that the derating

modifications made to the fitness function may conceal other optima [62]. Never the less, SN is the only sequential GA niching technique.

As a parallel niching technique, deterministic crowding (DC), presented in section 3.3.5 is used. DC is a replacement strategy that maintains several sets of similar individuals over a number of generations [61]. DC's custom selection policy only replaces offspring in a next generation when they perform better (fitness-wise) than their parents. A phenotypic similarity metric is used to quantify similarity between parents and offspring. As a parallel niching technique, DC is preferred over the more well-known fitness sharing [32], as it does not directly modify the fitness evaluation, i.e. it is not an explicit niching technique such as SN.

6.3 Experimental Setup

6.3.1 Test Problems

To test the different niching techniques, several functions presented in previous chapters were used. The following systems of equations, because of their irregular spacing of maxima, were used:

- System S3, defined in equation (4.15) on page 70, illustrated in figure 4.3.
- System S4, defined in equation (4.13) on page 68, illustrated in figure 4.2.
- System S5, defined in equation (4.21) on page 74, illustrated in figure 4.4(b).

The following multimodal functions, defined in chapter 5, are used (see page 95):

- Function F1, defined in equation (5.6), illustrated in figure 5.3(a).
- Function F2, defined in equation (5.7), illustrated in figure 5.3(b).
- Function F3, defined in equation (5.8), illustrated in figure 5.3(c).
- Function F4, defined in equation (5.9), illustrated in figure 5.3(d).
- Function F5, defined in equation (5.10), illustrated in figure 5.4.

6.3.2 Parameter Settings

For each of the *nbest*, NichePSO, SN and DC algorithms, 30 simulations were performed on each of the test problems outlined in section 6.3.1. The following sections describe GA and PSO parameter settings respectively.

GA Setup

For all experiments, populations consisting of 20 individuals were used. For Beasley *et al*'s SN algorithm, the following settings, as used in [4], were chosen:

Parameter	Value
p_c	0.9
p_m	0.01

where p_c and p_m signify the probability of whether the crossover and mutation operators are applied. A single-point crossover operator was used. As selection operator, *stochastic universal sampling* (SUS) is used, as suggested in [62]. One-dimensional problems used a 30-bit chromosome representation. For two-dimensional problems, two chromosomes of 15-bits each were used. The *halting window* approach described in [4], was used to terminate the algorithm. The approach monitors the average fitness of a population at each generation. If the average fitness has not improved on the fitness reported h generations earlier, the algorithm is terminated. For all runs of the SN algorithm, a halting window of $h = 20$ was used. Apart from this control setting, a maximum number of 2000 iterations was allowed. To determine the niche radius (see section 3.3.3), the method suggested by Beasley *et al* was used (originally suggested by Deb [20]). For a d-dimensional problem with l optima, the niche radius r was calculated as

$$r = \frac{\sqrt{d}}{2 \times \sqrt[d]{l}} \tag{6.1}$$

This technique assumes that fitness function parameters are normalized to $[0, 1]$. An exponential derating function G_e,

$$G_e(\mathbf{x}, \mathbf{x}^*) = \begin{cases} e^{\log m \frac{r - \|\mathbf{x} - \mathbf{x}^*\|}{r}} & \text{if } \|\mathbf{x} - \mathbf{x}^*\| < r \\ 1 & \text{otherwise} \end{cases}$$

was used (given in section 3.3.3, repeated here for clarity); $\mathbf{x}$ is an individual's phenotypic representation, $\mathbf{x}^*$ represents the best individual located using a particular generation's phenotype, and $\|\cdot\|$ is the Euclidean distance defined in each problem's search space.

Mahfoud's DC uses an internal selection scheme (see section 3.3.5). Crossover and mutation probabilities were set at

Parameter	Value
p_c	1.0
p_m	0.01

since the DC algorithm favors a very lower mutation probability and a high crossover probability [61]. DC also used a halting window-based termination criterion of $h = 20$, and a maximum number of generations of $g_{max} = 2000$.

PSO Setup

For all experiments, swarms consisting of 20 particles were used. A halting window approach, similar to that presented above was implemented, assuming that for NichePSO, the halting window conditions were applied to all swarms, and the main swarm was empty. The inertia weight was linearly scaled from 0.7 to 0.1, over a maximum of 2000 iterations of the respective algorithms. Coefficients c_1 and c_2 were both set to 1.2. These parameter settings allow particles to gradually decrease the magnitude of velocity and position updates, at the same time ensuring that they follow convergent trajectories [87]. Further parameters were as given in table 5.1.

6.4 Results and Discussion

Tables 6.1 and 6.2 quantify the performance of the tested niching techniques. In both tables, entries marked with a '*' indicate that experiments on the relevant test problem and algorithm combination were not carried out. Marked entries apply specifically to the situation where the *nbest* algorithm was used to find multiple solutions to problems with optima of varying fitness. If only fitness is considered on problems such as function F2 and F4 (see figures 5.7 and 5.9), topological neighborhoods of particles overlap. Particles

are therefore drawn to solutions with better fitness, and only converge on solutions with optimal fitness.

6.4.1 Computational Cost

Table 6.1 compares the computational cost of each of the niching techniques in terms of the number of fitness function evaluations required to converge. Note that in values reported in the format $a \pm b$, a refers to a mean calculated over all simulations, and b refers to the standard deviation calculated over the same values. The given results represent the actual number of fitness function evaluations that took place – distance calculations at each iteration of the respective algorithms were not factored in. This fact brings an interesting point forward: DC does not compare each individual in the population to every other population member at each generation of the algorithm: Offspring are only compared to their parents. The net effect of this is that DC requires a consistently higher number of fitness function evaluations. The following comments can be made based on the results shown in table 6.1:

- SN required fewer evaluations on the systems of equations in problems S3, S4 and S5.
- Although SN generally required a low number of fitness function evaluations, it should be taken into consideration that the basis of the SN algorithm necessitates the calculation of a derated fitness at each generation, that becomes more complex as the number of generations increases.
- DC consistently required more fitness function evaluations than the other algorithms.
- When comparing *nbest* and NichePSO, it is clear that NichePSO required a substantially smaller number of fitness function evaluations. In addition to its quota of fitness function evaluations, *nbest* also required the calculation of a matrix representing inter-particle distances, at each iteration of the algorithm.
- Both *nbest* and NichePSO consistently require less than the average number of fitness function evaluations to converge.

Problem	**SN**	**DC**	*nbest*	**NichePSO**	**Average**
S3	1840.73 ± 86.43	15087.67 ± 4197.35	8493.00 ± 413.27	2554.47 ± 228.22	6993.97
S4	1193.47 ± 96.30	17554.33 ± 4656.96	6934.12 ± 542.33	3965.50 ± 353.26	7411.86
S5	1926.60 ± 95.21	13816.00 ± 4224.50	7018.87 ± 670.09	2704.20 ± 134.76	6366.42
F1	4102.07 ± 576.58	14647.33 ± 4612.15	4769.00 ± 44.90	2371.86 ± 109.41	6472.57
F2	3504.80 ± 463.20	13052.33 ± 2506.65	*	2934.00 ± 475.44	6497.04
F3	4140.97 ± 553.93	13930.00 ± 3284.38	4789.00 ± 51.35	2403.73 ± 194.94	6315.93
F4	3464.07 ± 286.97	13929.33 ± 2996.15	*	2820.03 ± 517.09	6737.81
F5	3423.33 ± 402.13	14295.67 ± 3407.82	5007.67 ± 562.14	2151.47 ± 200.42	6219.54
Average	2949.51	14539.10	6168.61	2738.16	

Table 6.1: Average number of fitness function evaluations required to converge for each niching algorithm. Entries marked with a '*' indicate that experiments were not carried out for the relevant problem and algorithm.

- Overall, NichePSO required the least number of fitness function evaluations to converge.

6.4.2 Performance Consistency

Table 6.2 expresses as percentages the performance consistency of the tested niching techniques. 'Performance consistency' reflects each of the algorithm's ability to consistently locate all solutions to each of the optimization problems. All the tested techniques sufficiently maintained solutions sets. The following conclusions can be drawn from the results reported in table 6.2:

- Results for the SN algorithm compares well to that reported in [4]. The algorithm did however not perform as well on systems of equations, and generally performed worse than the average.
- Regardless of its higher computational requirement, DC did not generally yield superior performance.
- Although still better than SN, NichePSO performed equal to or worse than the average performance on the systems of equations in problems S3, S4 and S5.
- The *nbest* algorithm appears to have exhibited the most consistent performance over all the test problems (keeping in mind that it was not applied to F2 and F4).

6.5 Conclusion

This chapter presented an empirical comparison between the performances of the newly introduced PSO niching algorithms, and two well-known GA based techniques. Section 6.2 gave a brief overview of the algorithms. The test functions presented in section 6.3.1 were all well-known functions that have been used in this thesis and other literature for evaluating the effectiveness of niching techniques. Section 6.4 analyzed the experimental results obtained. It was established that both *nbest* and NichePSO are successful niching algorithms, and their performance, both in terms of computational complexity and performance consistency compare well to existing niching techniques.

Problem	**Sequential Niching**	**Deterministic Crowding**	*nbest*	**NichePSO**	**Average**
S3	76%	93%	100%	87%	89.00%
S4	66%	100%	100%	80%	86.50%
S5	83%	87%	100%	90%	90.00%
F1	100%	100%	93%	100%	98.25%
F2	83%	93%	*	93%	89.67%
F3	100%	90%	93%	100%	95.75%
F4	93%	90%	*	93%	92.00%
F5	86%	90%	100%	100%	94.00%
Average	85.88%	92.86%	97.67%	92.88%	

Table 6.2: The consistency with which each of the techniques managed to locate a complete set of solutions for each of the test problems. Entries marked with a '*' indicate that experiments were not carried out for the relevant problem and algorithm.

Chapter 7

Conclusion

This chapter briefly summarizes the findings and contributions of this thesis, followed by a number of ideas for further research and analysis.

7.1 Summary

This study investigated the application of the unique properties of the PSO algorithm to solve multimodal optimization problems. As a result, two new niching techniques were developed. Chapter 3 presented an overview of existing GA–based niching techniques. It was concluded that although it is not impossible to convert existing GA–based techniques for use with the PSO, the genetic representation and generational replacement used by GAs make it hard to directly implemented GA–based techniques on the PSO. Chapter 3 also tested the potential of an existing PSO global optimization algorithm, 'Stretched'-PSO (SPSO). It was found that the algorithm's performance is impaired by modifications made to the search space.

Chapter 4 introduced the *nbest* PSO niching algorithm, specifically to solve systems of equations. The use of topological neighborhoods that promote local social interaction among particles was found to be a promising extension to the PSO algorithm. Chapter 5 introduced NichePSO, a niching algorithm that uses multiple particle swarms to maintain different solutions in a single search space. In Chapter 6, the performance of both *nbest* and NichePSO was compared to two well-known GA niching techniques, *deterministic*

crowding and *sequential niching*. It was found that both of the newly introduced PSO niching algorithms were effective niching techniques, with performance comparable and better than the GA-based techniques.

7.2 Future Research and Analysis

A number of areas can be identified where the research in this thesis can be applied, or further investigated. These include the following:

nbest Neighborhood Formulation

The neighborhood formulation of the *nbest* algorithm showed to effectively find and maintain areas where very similar equations overlap. This property can be used in multi–objective optimization problems to locate and maintain Pareto fronts, as has been recently done by [38] with a similar algorithm.

Application to Global Optimization

The goal of niching techniques is to maintain diverse solutions in a search space. It remains to be seen whether this property of niching can be used to extend or enhance existing PSO diversity improvement techniques in a way that has not yet been investigated. Maintaining several solutions in different swarms and sharing information between the different swarms, may lead to improved global optimization algorithms (such an approach would seem similar to the island GA [34], and PSO subswarm techniques, such as work done by Løvbjerg *et al* [60]).

Further Investigation of NichePSO

- *Parameter Independence of NichePSO*: The current NichePSO implementation fails to correctly locate all solutions to a multimodal function if μ is greater than the inter-solution distance. The situation could be avoided by monitoring the effect of merging on swarm fitness. Ideally, swarm fitness should remain stable or improve. If particles from different potential solutions are merged, swarm fitness

will be erratic until the swarm settles on one solution. Swarms may of course not settle at all, as several potential solutions would confuse it. Alternatively, a technique similar to that of Goldberg and Wang's CSN could be utilized to remove this parameter limitation [65].

- *Swarm Sizes*: In section 5.4.4 it was found that a^2, where a is the number of optima in a multimodal function, was a conservative boundary as to the number of particles required to locate all solutions. The accuracy of this estimate warrants further investigation, specifically when applying NichePSO to multimodal functions of higher dimensions. Functions of higher dimension exist in a much larger search space. Alternative velocity vector initialization techniques could also be investigated to see whether the number of particles required per solutions can be reduced.

Ensemble Neural Networks

Section 2.9.1 investigated work presented by a number of authors where particle swarms were used to train neural networks. Ensemble architectures train a number of neural networks, either sequentially or in parallel on the same problem. Since the search space of a neural network may be highly multimodal, the use of a niching technique may be beneficial. The PSO has been shown to be an effective optimization technique for neural network training, and it seems a natural step to exploit the nature of niching algorithms and apply it to ensemble learning.

Development of Further Niching Techniques

The usefulness of existing PSO diversity improvement techniques as precursors to niching need still be investigated. The crossover operator used by Løvbjerg *et al* [60] is triggered by randomly assigning a crossover probability to particles. If this assignment is changed to use a ranking scheme, only highly fit particles will be used for crossover. Although crossover between highly fit particles in a multimodal domain is not at all beneficial, the approach is similar to deterministic crowding [61].

Section 3.2 mentioned the notion of cannibalism within a species. When using an

algorithm such as NichePSO to find multiple optima in a vastly multimodal function, a potentially large number of particles must be used to ensure consistent results. The use of a cannibalism operator, that removes particles where highly similar particles occur, may serve to simplify the interpretation of results.

Application of Niching Algorithms to Dynamic Clustering

Data clustering sorts data records into groups based on their similarity. The k-means algorithm is an example of a widely used algorithm. k-means however suffers from a major drawback: The number of clusters must be known in advance. In complex data sets, the number of clusters may not be known, or may be time dependent. For such problems, clustering algorithms that dynamically determine the number of clusters are needed. A cluster centroid represents a vector of the same dimension as data records in a data set, positioned on a location within the hyper-space defined by the data. If each particle in a swarm of particles represent a potential cluster centroid, centroids may be located by finding positions in the search space subject to the following conditions:

- The variability among data records associated with each centroid must be minimized.
- The variability among data centroids must be maximized.

Such a scheme could then dynamically determine the number of clusters in a dataset through the use of a niching algorithm such as NichePSO.

Bibliography

[1] B. Al-Kazemi and C. K. Mohan. Multi-Phase Generalization of the Particle Swarm Optimization Algorithm. In *Proceedings of the IEEE World Congress on Evolutionary Computation*, pages 489 – 494, Honolulu, Hawaii, 12 - 17 May 2002.

[2] P. J. Angeline. Using Selection to Improve Particle Swarm Optimization. In *Proceedings of the IEEE International Joint Conference on Neural Networks*, pages 84 – 89, July 1999.

[3] T. Bäck, D. B. Fogel, and T. Michalewicz, editors. *Basic Algorithms and Operators*, volume 1 of *Evolutionary Computation*. Institute of Physics Publishing, Bristol and Philidelphia, 1999.

[4] D. Beasley, D. R. Bull, and R. R. Martin. A Sequential Niching Technique for Multimodal Function Optimization. *Evolutionary Computation*, 1(2):101 – 125, 1993.

[5] C. Blake, E. Keogh, and C. J. Merz. UCI Repository of Machine Learning Databases, 2002. University of California, Irvine, Department of Information and Computer Sciences, http://www.ics.uci.edu/∼MLRepository.html.

[6] E. Cantu-Paz. A Summary of Research on Parallel Genetic Algorithms. Technical report, Genetic Algorithm Lab, Urbana, University of Illinois, Illinois, July 1995. IlliGAL Rep. 95007.

[7] A. Carlisle and G. Dozier. Adapting Particle Swarm Optimization to Dynamic Environments. In *Proceedings of the IEEE International Conference on Artificial Intelligence*, pages 429 – 434, Las Vegas, USA, 2000.

[8] A. Carlisle and G. Dozier. An Off-The-Shelf PSO. In *Proceedings of the Workshop on PSO*, Indianapolis, IN, USA, 2001. Purdue School of Engineering and Technology, IUPUI.

[9] A. Carlisle and G. Dozier. Tracking Changing Extrema with Particle Swarm Optimizer. Technical report, Auburn University, Alabama, USA, 2001. Technical Report CSSE01-08.

[10] D. L. Carroll. Genetic Algorithms and Optimizing Chemical Oxygen-Iodine Lasers. In R. Batra, C. Bert, A. Davis, R. Shapery, D Stewart, and F. Swinson, editors, *Developments in Theoretical and Applied Mechanics*, volume XVIII, pages 411 – 424. School of Engineering, University of Alabama, 1996.

[11] N. Christianini and J. Shawne-Taylor. *An Introduction to Support Vector Machines and Other Kernel-Based Learning Methods*. Cambridge University Press, 2000.

[12] A. Cichocki and R. Unbehauen. Neural Networks for Solving Systems of Linear Equations and Related Problems. *IEEE Transactions on Circuits and Systems–I: Fundamental Theory and Applications*, 39(2), February 1992.

[13] M. Clerc and J. Kennedy. The Particle Swarm – Explosion, Stability and Convergence in a Multidimensional Complex Space. *IEEE Transactions on Evolutionary Computation*, 6(1):58 – 73, February 2002.

[14] C. A. Coello Coello. An Updated Survey of Evolutionary Multiobjective Optimization Techniques: State of the Art and Future Trends. In *Proceedings of the IEEE Congress on Evolutionary Computation*, pages 3 – 13, Washington, DC, July 1999.

[15] C. A. Coello Coello and M. S. Lechuga. MOPSO: A Proposal for Multiple Objective Particle Swarm Optimization. In *Proceedings of the IEEE World Congress on Evolutionary Computation*, pages 1051 – 1056, Honolulu, Hawaii, May 2002.

[16] A. Conradie, R. Miikulainen, and C. Aldrich. Adaptive Control utilising Neural Swarming. In *Proceedings of the Genetic and Evolutionary Computation Conference*, New York, USA, 2002.

[17] A. Conradie, R. Miikulainen, and C. Aldrich. Intelligent Process Control utilising Symbiotic Memetic Neuro-Evolution. In *Proceedings of the IEEE World Congress on Evolutionary Computation*, pages 623 – 628, Honolulu, Hawaii, 12 - 17 May 2002.

[18] A. Conradie, I. Nieuwoudt, and C. Aldrich. Nonlinear Neurocontroller Development with Evolutionary Reinforcement Learning. In *9th National Meeting of SAIChe*, Secunda, South Africa, 2002.

[19] K. A. de Jong. *An Analysis of the Behavior of a Class of Genetic Adaptive Systems.* PhD thesis, Department of Computer Science, University of Michigan, Ann Arbor, Michigan, USA, 1975.

[20] K. Deb. Genetic Algorithms in Multimodal Function Optimization. Master's thesis, Department of Engineering Mathematics, University of Alabama, 1989.

[21] R. C. Eberhart and X. Hu. Human Tremor Analysis Using Particle Swarm Optimization. In *Proceedings of the IEEE World Congress on Evolutionary Computation*, pages 1927 – 1930, Washington DC, USA, July 1999.

[22] R. C. Eberhart and Y. Shi. Comparing Inertia Weights and Constriction Factors in Particle Swarm Optimization. In *Proceedings of the IEEE Congress on Evolutionary Computation*, pages 84 – 89, San Diego, USA, 2000.

[23] R. C. Eberhart and Y. Shi. Tracking and Optimizing Dynamic Systems with Particle Swarms. In *Proceedings of the IEEE Congress on Evolutionary Computation*, pages 94 – 100, Seoul, Korea, 2001.

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

[25] A. P. Engelbrecht and A. Ismail. Training Product Unit Neural Networks. *Stability and Control: Theory and Applications*, 2(1–2):59 – 74, 1999.

[26] J. B. Fraleigh and R. A. Beauregard. *Linear Algebra.* Addison Wesley Publishing Company, 3rd edition, 1995.

[27] B. Gabrys and A. Bargiela. Neural Simulation of Water Systems for Efficient State Estimation. *Proceedings of the European Simulation Multiconference*, pages 775 – 779, 1995.

[28] J. Gan and K. Warwick. A Variable Radius Niche Technique for Speciation in Genetic Algorithms. In *Proceedings of the Genetic and Evolutionary Computation Conference*, pages 96 – 103. Morgan-Kaufmann, 2000.

[29] J. Gan and K. Warwick. Dynamic Niche Clustering: A Fuzzy Variable Radius Niching Technique for Multimodal Optimization in GAs. In *Proceedings of the IEEE Congress on Evolutionary Computation*, volume I, pages 215 – 222, 2001.

[30] J. Gan and K. Warwick. Modelling Niches of Arbitrary Shape in Genetic Algorithms using Niche Linkage in the Dynamic Niche Clustering Framework. In *Proceedings of the IEEE World Congress on Evolutionary Computation*, pages 43 – 48, Honolulu, Hawaii, 12 - 17 May 2002.

[31] D. E. Goldberg. *Genetic Algorithms in Search, Optimization and Machine Learning.* Addison Wesley, Reading, MA, 1989.

[32] D. E. Goldberg and J. Richardson. Genetic Algorithm with Sharing for Multimodal Function Optimization. In *Proceedings of the Second International Conference on Genetic Algorithms*, pages 41 – 49, 1987.

[33] D. E. Goldberg and L. Wang. Adaptive Niching via Coevolutionary Sharing. Technical report, Genetic Algorithm Lab, Urbana, University of Illinois, Illinois, August 1997. IlliGAL Rep. 97007.

[34] P. Grosso. *Computer Simulations of Genetic Application: Parrallel Subcomponent Interaction in a Multilocus Model.* PhD thesis, University of Michigan, USA, 1985.

[35] G. R. Harik. Finding Multimodal Solutions Using Restricted Tournament Selection. Technical report, IlliGAL, University of Illinois at Urbana-Champaign, Urbana, Illinois, 1995.

[36] J. Holland. *Adapation in Natural and Artificial Systems.* University of Michigan Press, Ann Arbor, Michigan, USA, 1975.

[37] J. Horn. *The nature of niching: Genetic algorithms and the evolution of optimal, cooperative populations.* PhD thesis, Urbana, University of Illinois, Illinois, Genetic Algorithm Lab, 1997.

[38] X. Hu and R. Eberhart. Multiobjective Optimization using Dynamic Neighborhood Particle Swarm Optimization. In *Proceedings of the IEEE World Congress on Evolutionary Computation*, pages 1677 – 1681, Honolulu, Hawaii, 12 - 17 May 2002.

[39] X. Hu and R. C. Eberhart. Tracking Dynamic Systems with PSO: Where's the Cheese? In *Proceedings of the Workshop on Particle Swarm Optimization*, Purdue School of Engineering and Technology, Indianapolis, USA, 2001.

[40] X. Hu and R .C. Eberhart. Adaptive Particle Swarm Optimization: Detection and Response to Dynamic Systems. In *Proceedings of the IEEE World Congress on Evolutionary Computation*, pages 1666 – 1670, Honolulu, Hawaii, May 2002.

[41] D. S. Huang and Z. Chi. Neural Networks with Problem Decomposition for Finding Real Roots of Polynomials. In *Proceedings of the IEEE International Joint Conference on Neural Networks*, volume Addendum, pages 25 – 30, Washington DC, 15-19 July 2001.

[42] D. S. Huang and Z. Chi. Solving Linear Simultaneous Equations by Constraining Learning Neural Networks. In *Proceedings of the IEEE International Joint Conference on Neural Networks*, volume Addendum, pages 26 – 31, Washington DC, 15-19 July 2001.

[43] K. Hwang and S. Cho. Evolving Diverse Hardwares Using Speciated Genetic Algorithm. In *Proceedings of the IEEE World Congress on Evolutionary Computation*, pages 437–442, Honolulu, Hawaii, 12 - 17 May 2002.

[44] A. Ismail and A. P. Engelbrecht. Training Product Units in Feedforward Neural Networks using Particle Swarm Optimization. *Proceedings of the International Conference on Artificial Intelligence*, pages 36 – 40, 1999.

[45] C. Z. Janikow and Z. Michalewicz. An Experimental Comparison of Binary and Floating Point Representations in Genetic Algorithms. In *Proceedings of the 4th International Conference on Genetic Algorithms*, pages 31 – 36. Morgan Kaufmann, San Diego, USA, 1991.

[46] Y. Jin, T. Okabe, and B. Sendhoff. Dynamic Weighted Aggregation for Evolutionary Multiobjective Optimization: Why Does It Work and How? In *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO'2001)*, pages 1042 – 1049, San Francisco, USA, 2001.

[47] J. Kennedy. Small Worlds and Mega-Minds: Effects of Neighborhood Topology on Particle Swarm Performance. *Proceedings of the IEEE Congress on Evolutionary Computation*, pages 1931 – 1938, July 1999.

[48] J. Kennedy. Stereotyping: Improving Particle Swarm Performance with Cluster Analysis. In *Proceedings of the IEEE Congress on Evolutionary Computation*, pages 1507 – 1512, San Diego, USA, 2000.

[49] J. Kennedy and R. C. Eberhart. Particle Swarm Optimization. In *Proceedings of the IEEE International Conference on Neural Networks*, volume IV, pages 1942 – 1948, Perth, Australia, 1995.

[50] J. Kennedy and R. C. Eberhart. A Discrete Binary Version of the Particle Swarm Algorithm. In *Proceedings of the Conference on Systems, Man and Cybernetics*, pages 4104 – 4109, 1997.

[51] J. Kennedy and R. Mendes. Population Structure and Particle Swarm Performance. In *Proceedings of the IEEE World Congress on Evolutionary Computation*, pages 1671 – 1676, Honolulu, Hawaii, May 2002.

[52] J. Kennedy and W. M. Spears. Matching Algorithms to Problems: An Experimental Test of the Particle Swarm and some Genetic Algorithms on the Multimodal Problem Generator. In *Proceedings of the IEEE World Congress on Computational Intelligence*, pages 78 – 83, Achorage, Alaska, 1998.

[53] K. Kim and S. Cho. Evolving Speciated Checkers Players with Crowding Algorithm. In *Proceedings of the IEEE World Congress on Evolutionary Computation*, pages 407 – 412, Honolulu, Hawaii, 12 - 17 May 2002.

[54] J. D. Knowles and D. W. Corne. Approximating the Nondominated Front Using the Pareto Archived Evolution Strategy. *Evolutionary Computation*, 8(2):149 – 172, 2000.

[55] T. Krink, J. K. Vesterstrøm, and J. Riget. Particle Swarm Optimization with Spatial Particle Extension. In *Proceedings of the IEEE World Congress on Evolutionary Computation*, pages 1474 – 1479, Honolulu, Hawaii, 12 - 17 May 2002.

[56] R. A. Krohling, H. Knidel, and Y. Shi. Solving Numerical Equations of Hydraulic Problems Using Particle Swarm Optimization. In *Proceedings of the IEEE World Congress on Evolutionary Computation*, pages 1688 – 1690, Honolulu, Hawaii, 12 - 17 May 2002.

[57] E. C. Laskarski, K. E. Parsopoulos, and M. N. Vrahatis. Particle Swarm Optimization for Integer Programming. In *Proceedings of the IEEE World Congress on Evolutionary Computation*, pages 1582 – 1587, Honolulu, Hawaii, 12 - 17 May 2002.

[58] E. C. Laskarski, K. E. Parsopoulos, and M. N. Vrahatis. Particle Swarm Optimization for Minimax Problems. In *Proceedings of the IEEE World Congress on Evolutionary Computation*, pages 1576 – 1581, Honolulu, Hawaii, 12 - 17 May 2002.

[59] M. Løvbjerg and T. Krink. Extending Particle Swarm Optimizers with Self-Organized Criticality. In *Proceedings of the IEEE World Congress on Evolutionary Computation*, pages 1588 – 1593, Honolulu, Hawaii, May 2002.

[60] M. Løvbjerg, T. K. Rasmussen, and T. Krink. Hybrid Particle Swarm Optimizer with Breeding and Subpopulations. In *Proceedings of the Genetic and Evolutionary Computation Conference*, volume 1, pages 469 – 476, San Fransisco, USA, July 2001.

[61] S. W. Mahfoud. *Niching Methods for Genetic Algorithms*. PhD thesis, Genetic Algorithm Lab, University of Illinois, Illinois, 1995. IlliGAL Rep. 95001.

[62] S.W. Mahfoud. A Comparison of Parallel and Sequential Niching Methods. In *Proceedings of the Sixth International Conference on Genetic Algorithms*, pages 136 – 143, 1995.

[63] E. Mayr. *Animal Species and Evolution*. Belknap, Cambridge, MA, 1963.

[64] R. Mendes, P. Cortez, M. Rocha, and J. Neves. Particle Swarms for Feedforward Neural Network Training. In *Proceedings of the IEEE Joint Conference on Neural Networks*, pages 1895 – 1899, Honolulu, Hawaii, 12 – 17 May 2002.

[65] O. J. Mengshoel and D. E. Goldberg. Probabilistic Crowding: Deterministic Crowding with Probabilistic Replacement. In *Proceedings of the Genetic and Evolutionary Computation Conference 1999*, pages 409 – 416, San Fransisco, USA, Morgan Kaufmann, 1999.

[66] B. L. Miller and M. J. Shaw. Genetic Algorithms with Dynamic Niche Sharing for Multimodal Function Optimization. Technical report, Genetic Algorithm Lab, Urbana, University of Illinois, Illinois, December 1995. IlliGAL Rep. 95010.

[67] N. J. Nilsson. *Artificial Intelligence: A New Synthesis*. Morgan Kaufmann Publishers, Inc, 1998.

[68] E. Ozcan and C. K. Mohan. Analysis of a Simple Particle Swarm Optimization System. In *Intelligent Engineering Systems Through Artificial Neural Networks*, volume 8, pages 253 – 258, 1998.

[69] E. Ozcan and C. K. Mohan. Particle Swarm Optimization : Surfing the Waves. In *Proceedings of the International Congress on Evolutionary Computation*, pages 1939 – 1944, Washington, USA, 1999.

[70] U. Paquet. Training Support Vector Machines with Particle Swarms. Master's thesis, Department of Computer Science, University of Pretoria, Pretoria, South Africa, 2002.

[71] K. E. Parsopoulos, V. P. Plagianakos, G. D. Magoulas, and M. N. Vrahatis. Stretching Technique for Obtaining Global Minimizers Through Particle Swarm Optimiza-

tion. In *Proceedings of the Particle Swarm Optimization Workshop*, pages 22 – 29, Indianapolis, USA, 2001.

[72] K. E. Parsopoulos and M. N. Vrahatis. Modification of the Particle Swarm Optimizer for Locating all the Global Minima. In V. Kurkova, N.C. Steele, R. Neruda, and M. Karny, editors, *Artificial Neural Networks and Genetic Algorithms*, pages 324 – 327. Springer, 2001.

[73] K. E. Parsopoulos and M. N. Vrahatis. Particle Swarm Optimization in Noisy and Continuously Changing Environments. In M. H. Hamza, editor, *Artificial Intelligence and Soft Computing*, pages 289 – 294. IASTED/ACTA, Anaheim, USA, 2001.

[74] K. E. Parsopoulos and M. N. Vrahatis. Particle Swarm Optimization Method for Constrained Optimization. In P. Sincak, J. Vascak, V. Knasnicka, and J. Pospichal, editors, *Intelligent Technologies – Theory and Applications: New Trends in Intelligent Technologies*, volume 76, pages 214 – 220. IOS Press, 2002.

[75] K. E. Parsopoulos and M. N. Vrahatis. Particle Swarm Optimization Method in Multiobjective Problems. In *Proceedings of the 2002 ACM Symposium on Applied Computing (SAC 2002)*, pages 603 – 607, 2002.

[76] M. A. Potter. *The Design and Analysis of a Computational Model of Cooperative Coevolution*. PhD thesis, George Mason University, Fairfax, Virginia, USA, 1997.

[77] W. H. Press, S. A. Teukolsky, W. T. Vetterling, and B. P. Flannery. *Numerical Recipes in C: The Art of Scientific Computing*. Cambridge University Press, second edition, 1992.

[78] W. C. Rheinboldt. *Methods for Solving Systems of Nonlinear Equations*. Society for Industrial & Applied Mathematics, second edition, 1998.

[79] J. D. Schaffer. Multiple Objective Optimization with Vector Evaluated Genetic Algorithms. pages 93 – 100, 1985.

[80] Y. Shi and R. C. Eberhart. A Modified Particle Swarm Optimizer. In *Proceedings of the IEEE World Conference on Computational Intelligence*, pages 69 – 73, Anchorage, Alaska, May 1998.

[81] Y. Shi and R. C. Eberhart. Fuzzy Adaptive Particle Swarm Optimization. In *Proceedings of the 2001 Congress on Evolutionary Computation*, pages 101 – 106, Seoul, Korea, 27-30 May 2001.

[82] Y. Shi and R. C. Eberhart. Particle Swarm Optimization with Fuzzy Adaptive Inertia Weight. In *Proceedings of the Workshop on Particle Swarm Optimization*, Purdue School of Engineering and Technology, Indianapolis, USA, 2001.

[83] Y. Shi and R. A. Krohling. Co-evolutionary Particle Swarm Optimization to Solve min-max Problems. In *Proceedings of the IEEE World Congress on Evolutionary Computation*, pages 1682 – 1687, Honolulu, Hawaii, 12 - 17 May 2002.

[84] W. M. Spears. Simple Subpopulation Schemes. In *Proceedings of the Evolutionary Programming Conference*, pages 296 – 307, 1994.

[85] P. N. Suganthan. Particle Swarm Optimizer with Neighborhood Operator. *Proceedings of the IEEE Congress on Evolutionary Computation*, pages 1958 – 1961, July 1999.

[86] E. Thiémard. Economic Generation of Low-Discrepancy Sequences with a b-ary Gray Code. Department of Mathematics, Ecole Polytechnique Fédérale de Lausanne, Lausanne, Switzerland.

[87] F. van den Bergh. *An Analysis of Particle Swarm Optimizers*. PhD thesis, Department of Computer Science, University of Pretoria, Pretoria, South Africa, 2002.

[88] F. van den Bergh and A. P. Engelbrecht. Cooperative Learning in Neural Networks using Particle Swarm Optimizers. *South African Computer Journal*, 26:84 – 90, November 2000.

[89] F. van den Bergh and A. P. Engelbrecht. Training Product Unit Networks using Cooperative Particle Swarm Optimizers. In *Proceedings of the IEEE International*

Joint Conference on Neural Networks, pages 126 – 132, Washington DC, USA, July 2001.

[90] F. van den Bergh and A. P. Engelbrecht. A New Locally Convergent Particle Swarm Optimizer. *Accepted for IEEE Conference on Systems, Man and Cybernetics*, October 2002.

[91] J. S. Vesterstrøm, J. Riqet, and T. Krink. Division of Labor in Particle Swarm Optimization. In *Proceedings of the IEEE World World Congress on Evolutionary Computation*, pages 1570 – 1575, Honolulu, Hawaii, May 2002.

Appendix A

Derived Publications

This appendix lists all the papers that have been published, or are currently under review, that were derived from work done in this thesis.

1. R. Brits, A.P. Engelbrecht and F. van den Bergh. Solving Systems of Unconstrained Equations using Particle Swarm Optimization. *IEEE International Conference on Systems, Man and Cybernetics*, Hammamet, Tunisia, October 2002.

2. R. Brits, A.P. Engelbrecht and F. van den Bergh. A Niching Particle Swarm Optimizer. *Conference on Simulated Evolution and Learning*, Singapore, November 2002.

3. R. Brits, A.P. Engelbrecht and F. van den Bergh. Particle Swarm Niching. Submitted to *IEEE Transactions on Evolutionary Computation*.

4. R. Brits, A.P. Engelbrecht and F. van den Bergh. Scalability of Niche PSO. Submitted to *IEEE Swarm Intelligence Symposium 2003*.