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:

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

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

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

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

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

$x = 9$ 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

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

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].

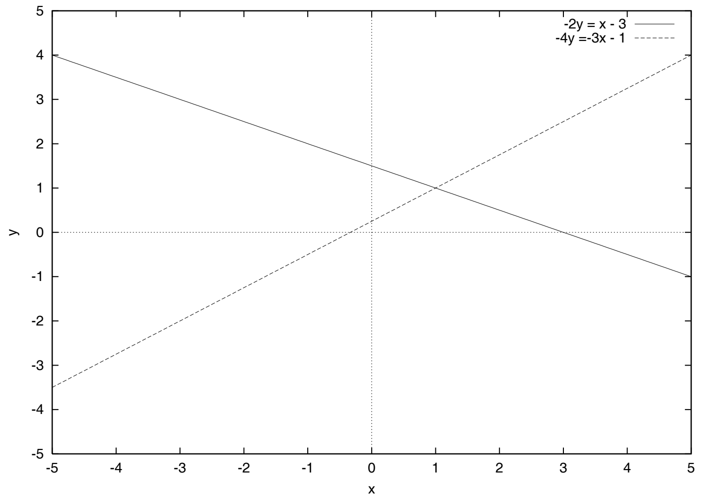


Figure 4.1: System S1, a simple system of two linear equations.

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:

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

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).

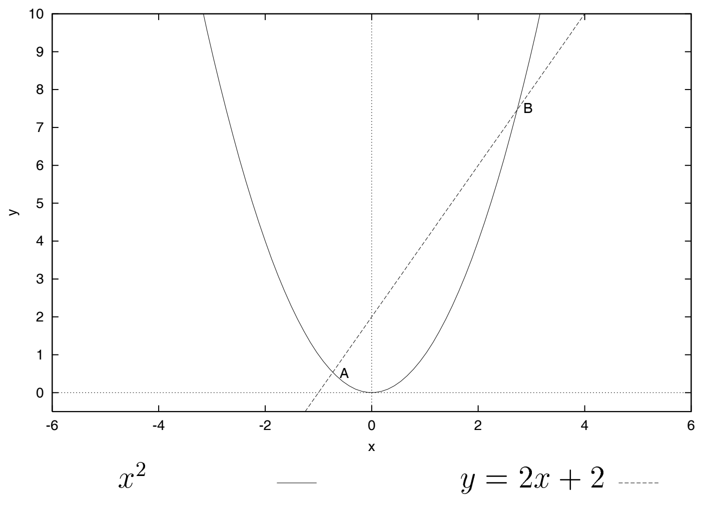


Figure 4.2: System S4, with multiple solutions.

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 system 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:

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

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.

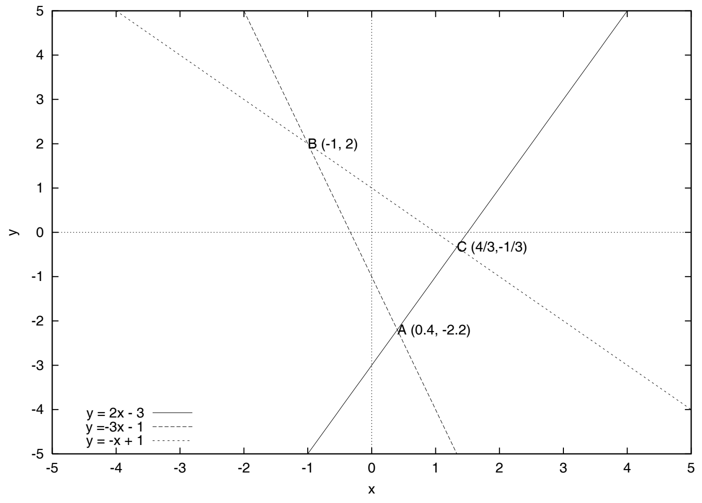


Figure 4.3: System S3: A system of linear equations with 3 solutions.

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].

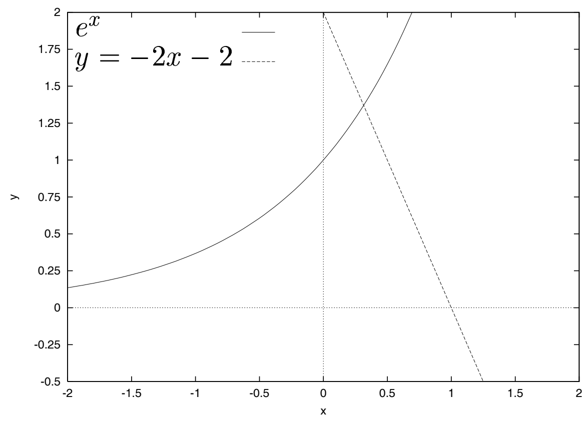
In addition to the systems defined previously, the following systems of equations are considered:

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

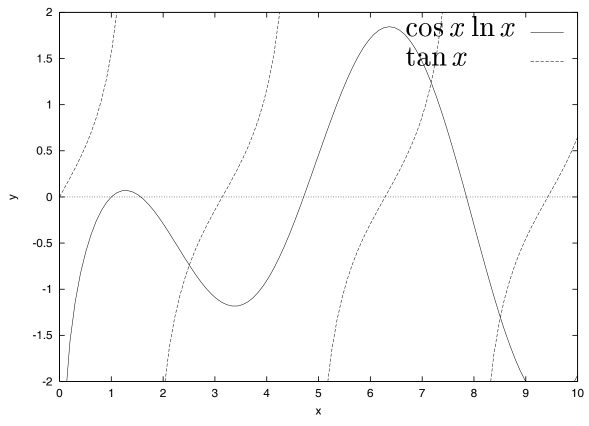
$$\begin{aligned} \text{S5: } y &= \cos x \ln x \\ y &= \tan x \end{aligned} \tag{4.21}$$

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

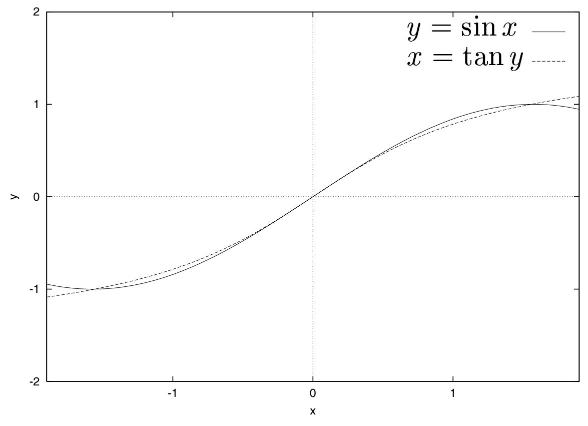
Figure 4.4 illustrates each of these systems.



(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.

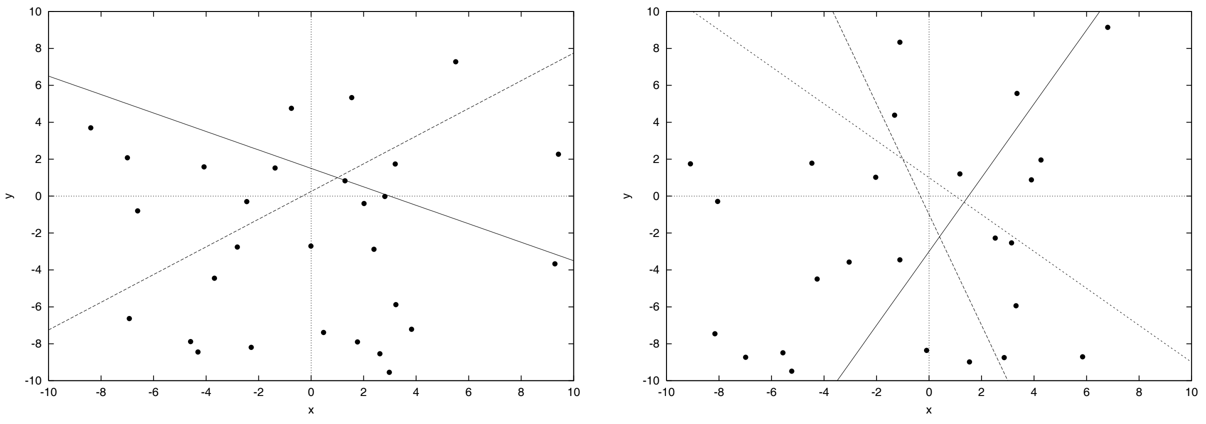
The influence of k is demonstrated by running the *nbest* algorithm with the different k values as set out in table 4.4 on the functions above. Fitness functions are defined as described in section 4.3.1. For each SEs, $c_1 = c_2 = 1.4$ and w is linearly scaled from 0.7 to 0.1 over 2000 iterations of 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:

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

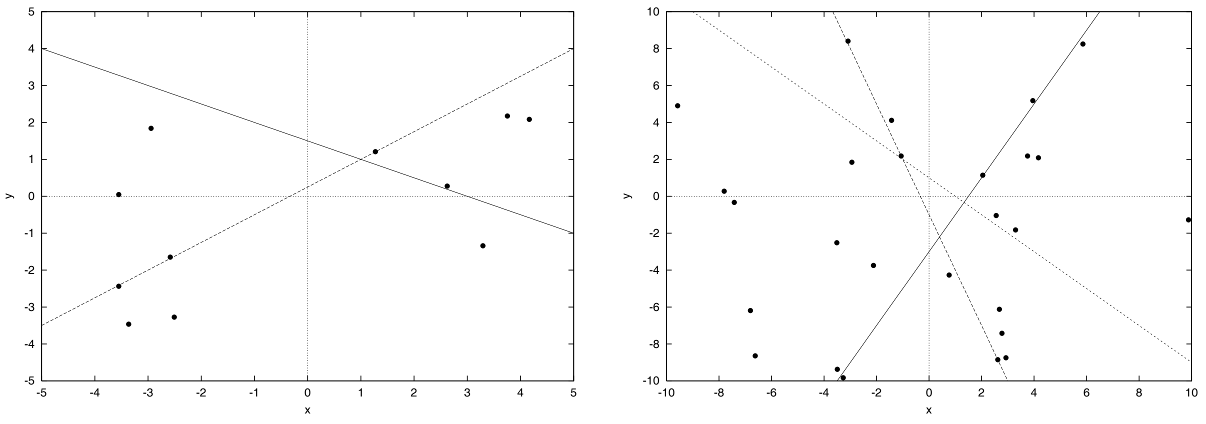
where $t_{\max}$ is the maximum time step and $k_{initial}$ and k_{final} is defined as in table 4.4. Next, the effects of the different k values are considered individually:

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

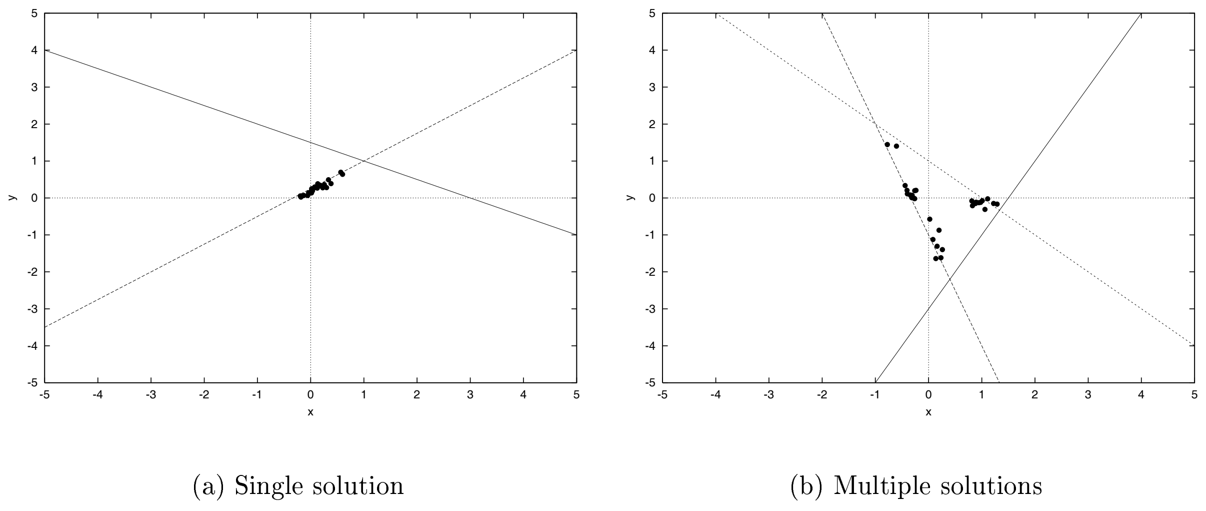
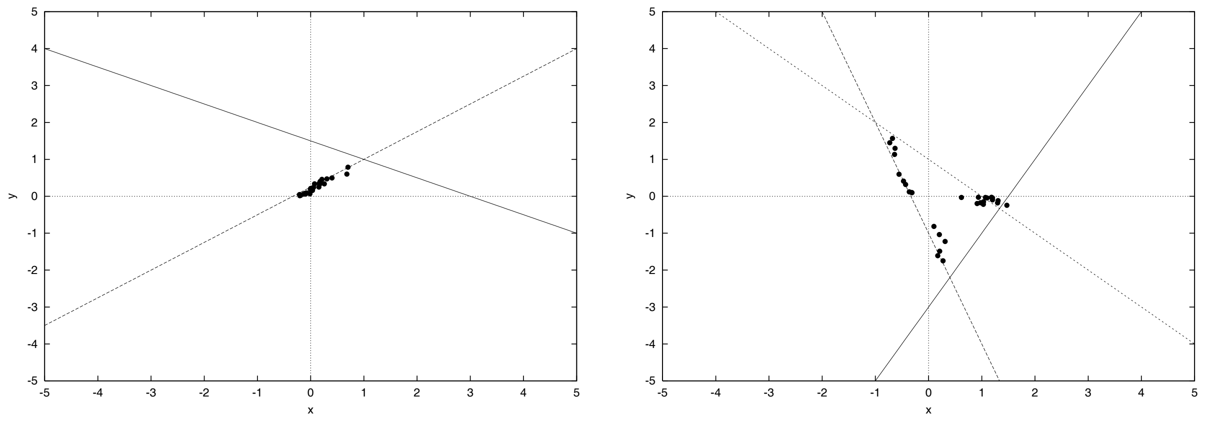


Figure 4.7: D3: Particle position after 2000 iterations, with $k = |S|$ kept constant

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

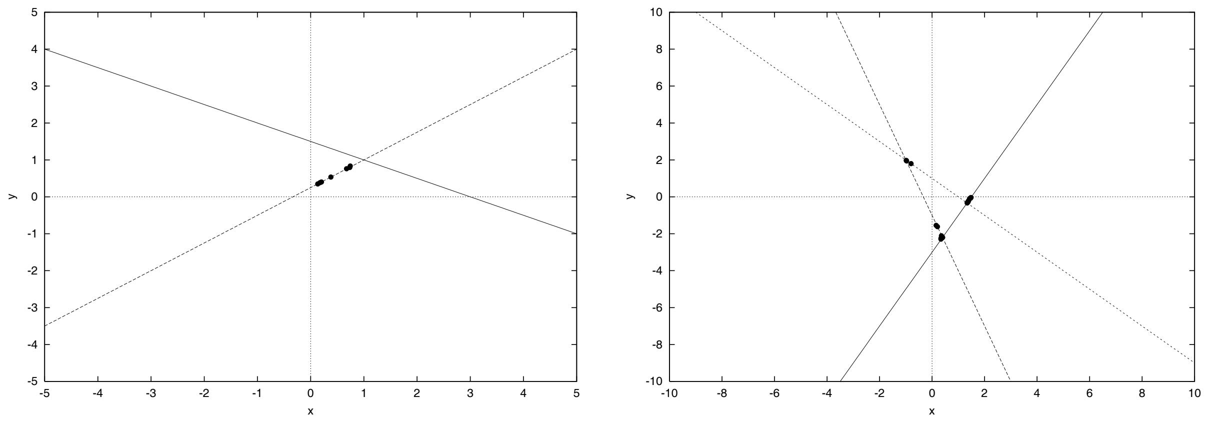


(a) Single solution (b) Multiple solutions

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

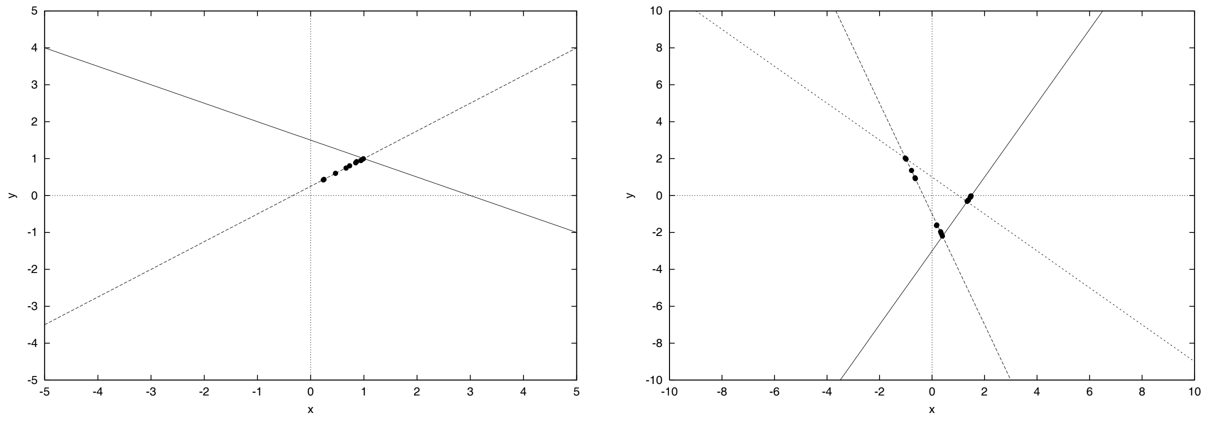
social interaction between a particle and its neighborhood, facilitating improved information exchange. The tendency of particles to settle on lines in a SEs is explained by the fact that any position on a line relatively close to a solution will have a low, and therefore attractive fitness value. An 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

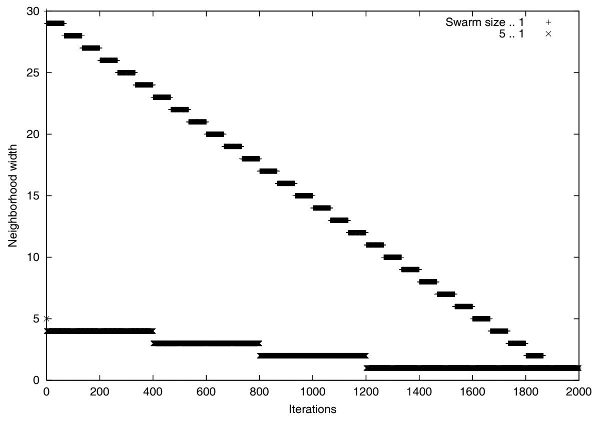


Figure 4.11: Linearly decreasing neighborhood sizes. Note that k is always a discrete value, explaining the stepwise decrease.

> 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.