Available online at www.sciencedirect.com
ScienceDirect
Applied Mathematics and Computation 189 (2007) 1859–1883
APPLIED MATHEMATICS AND COMPUTATION
www.elsevier.com/locate/amc

Locating multiple optima using particle swarm optimization

R. Brits [a], A.P. Engelbrecht [a,*], F. van den Bergh [b]

[a] *Department of Computer Science, University of Pretoria, South Africa*
[b] *Meraka Institute, CSIR, Pretoria, South Africa*

Abstract

Many scientific and engineering applications require optimization methods to find more than one solution to multi-modal optimization problems. This paper presents a new particle swarm optimization (PSO) technique to locate and refine multiple solutions to such problems. The technique, NichePSO, extends the inherent unimodal nature of the standard PSO approach by growing multiple swarms from an initial particle population. Each subswarm represents a different solution or niche; optimized individually. The outcome of the NichePSO algorithm is a set of particle swarms, each representing a unique solution. Experimental results are provided to show that NichePSO can successfully locate all optima on a small set of test functions. These results are compared with another PSO niching algorithm, *lbest* PSO, and two genetic algorithm niching approaches. The influence of control parameters is investigated, including the relationship between the swarm size and the number of solutions (niches). An initial scalability study is also done.

Keywords: Particle swarm optimization; Niching; Speciation

1. Introduction

Function optimization is the process of finding an optimal solution to an objective function describing a problem. Optimization can be either a minimization or maximization task. Optimization problems can be broadly categorized into unimodal and multi-modal problems. Unimodal problems have a single global optimum, $\mathbf{x}^*$, subject to (assuming minimization)

$$f(\mathbf{x}^*) \leqslant f(\mathbf{x}) \quad \forall \mathbf{x} \in \mathbb{R}^n,$$

where $f(\mathbf{x}) : \mathbb{R}^n \to \mathbb{R}$ is the objective function and n is the dimension of the search space. Multi-modal problems, on the other hand, have more than one optimum. These optima may all be global optima, or a mixture of global and local optima. A local optimum, $\mathbf{x}_L^*$, is subject to (assuming minimization)

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

where $L \subset \mathbb{R}^n$.

* Corresponding author.
E-mail addresses: rbrits@cs.up.ac.za (R. Brits), engel@cs.up.ac.za (A.P. Engelbrecht), fvdbergh@csir.co.za (F. van den Bergh).

0096-3003/$ - see front matter © 2007 Elsevier Inc. All rights reserved.
doi:10.1016/j.amc.2006.12.066

Many scientific and engineering optimization problems have convoluted search spaces with large numbers of optima. In the case of more than one global optimum, algorithms are needed to obtain all these solutions. It may also be beneficial to locate all, or as many as possible, local optima.

Function optimization has received extensive research attention, and several machine learning techniques such as neural networks [3,16], evolutionary algorithms [1], and swarm intelligence-based algorithms [23,12,13], have been developed and applied successfully to solve a wide range of complex optimization problems. By far the largest part of this research concentrated on developing algorithms that can locate only a single solution. However, evolutionary algorithms research has produced a number of approaches to find multiple solutions [19,27,18]. These evolutionary algorithms are generally referred to as *niching* or *speciation* algorithms. Each possible solution, known as a *niche*, is represented by a grouping of homogeneous GA individuals.

Eberhart and Kennedy recently introduced the *particle swarm optimization* (PSO) approach [22]. It is similar to evolutionary algorithms in that it evolves a group of candidate solutions. PSO, however, allows each individual to maintain a *memory* of the best solution that it has found and the best solution found in the individual's neighborhood. Each individual's traversal of the search space is then influenced by its own memory of best positions, with the individual moving towards a stochastically weighted average of these best positions [38,41]. The PSO algorithm has been shown to successfully solve a variety of unimodal optimization problems [35]. Several techniques have been proposed to improve the PSO algorithm's traversal of the search space [21,25,30,36,38,39].

Attempts have been made to solve multi-modal optimization problems with PSO, in the form of niching techniques.

Parsopoulos and Vrahatis [31] proposed an extension to a PSO convergence rate improvement technique as a sequential niching approach. The technique locates multiple optima by adapting the objective function's fitness landscape each time a new solution is located. The *nbest* PSO is a parallel niching technique used to locate multiple solutions to simple systems of equations [4,5].

This paper presents and empirically analyses the *niching particle swarm optimization* algorithm, NichePSO. NichePSO is aimed at locating multiple, optimal solutions to multi-modal optimization problems. Niches are identified by monitoring the fitness of individual particles, and growing subswarms from the initial particle swarm population. Using a global optimization strategy, subswarms then refine the solution represented by the niche.

While the focus of this paper is on niching techniques for locating multiple solutions to multi-modal optimization problems, niching techniques have also been used to solve multi-objective optimization problems. In the case of multiple objectives, the objective function consists of K (possibly conflicting) sub-objectives, where the task is to optimize these sub-objectives concurrently. The objective function, $\mathbf{f}$, is expressed as $\mathbf{f}(\mathbf{x}) = \{f_1(\mathbf{x}), f_2(\mathbf{x}), \ldots, f_K(\mathbf{x})\}$. Multi-objective optimization algorithms find good trade-offs between conflicting objectives rather than a single solution. A set of such solutions is referred to as the Pareto-optimal set. Several evolutionary algorithm approaches to multi-objective optimization have been developed. For an extensive treatment refer to [7,9]. Recently, PSO techniques have been developed for solving multi-objective problems [7,8,15,20,32].

It is significant to make a clear distinction between *multi-objective* and *multi-modal* optimization problems: Where solutions to multi-objective problems represent a combination of (possibly conflicting) solutions to each of the subgoals $f_i(\mathbf{x})$ in $\mathbf{f}$, solutions to multi-modal optimization problems represent possible values for $\mathbf{x}$ under a single objective function f. This is the main focus of our study: niching algorithms for multi-modal, single-objective problems.

The standard PSO algorithm and extensions toit as used within the NichePSO are discussed in Section 2. Section 3 presents an introduction to niching and existing niching techniques, introduced both in the fields of genetic algorithms and particle swarm optimizers. NichePSO is presented in Section 4, with experimental findings reported in Section 5. Section 5 also includes an empirical comparison between niching algorithms.

2. Particle swarm optimizers

This section presents and discusses the original PSO algorithm. Modifications and improvements to the PSO algorithm have been suggested by several authors. This section discusses PSO improvements used by

the NichePSO algorithm, as well as improvements which have influenced it. The section ends with a short discussion on the ability of the standard PSO to locate multiple solutions.

2.1. The standard PSO

Particle swarm optimizers are optimization algorithms modeled after the social behavior of birds in a flock [22]. PSO is a population based search process where individuals, referred to as particles, are grouped into a swarm. Each particle in a swarm represents a candidate solution to the 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 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.

Each particle i maintain a current position, $\mathbf{x}_i$, current velocity, $\mathbf{v}_i$, and personal best position, $\mathbf{y}_i$. For the purposes of this paper, $\mathbf{x}_i$ represents a position in an unconstrained, continuous search space. 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. If f denotes the objective function to be minimized, then the personal best of a particle at a time step t is updated as

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

Different PSO models have been developed based on the neighborhood topology particles use to exchange information about the search space [24]. In the *gbest* model, which is used in this paper, 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

$$\hat{\mathbf{y}}(t) \in \{\mathbf{y}_0, \mathbf{y}_1, \ldots, \mathbf{y}_s\} | f(\hat{\mathbf{y}}(t)) = \min\{f(\mathbf{y}_0(t)), f(\mathbf{y}_1(t)), \ldots, f(\mathbf{y}_s(t))\}, \tag{2}$$

where s is the total number of particles in the swarm. For each iteration of a *gbest* PSO, the jth-dimension of particle i's velocity vector, $\mathbf{v}_i$, and its position vector, $\mathbf{x}_i$, is updated as follows:

$$v_{i,j}(t+1) = wv_{i,j}(t) + c_1 r_{1,j}(t)(y_{i,j}(t) - x_{i,j}(t)) + c_2 r_{2,j}(t)(\hat{y}_{i,j}(t) - x_{i,j}(t)), \tag{3}$$

$$\mathbf{x}_i(t+1) = \mathbf{x}_i(t) + \mathbf{v}_i(t+1), \tag{4}$$

where w is the inertia weight, c_1 and c_2 are the acceleration constants and $r_{1,j}(t), r_{2,j}(t) \sim U(0,1)$. Upper and lower bounds are usually 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 $[-v_{\max,j}, v_{\max,j}]$. The inertia weight, w, was introduced by Shi and Eberhart [34] to control the influence of the velocity vector on a particle's position. Decreasing w from a relatively large value to a small value over time, results in rapid initial exploration of the search space, and facilitates later exploration. Small w-values result in small adaptations to particle positions, effectively yielding a local search.

The PSO algorithm performs repeated applications of the update equations until a specified number of iterations has been exceeded, or until velocity updates are close to zero.

The reader is referred to [38,41] for a study of the relationship between the inertia weight and the acceleration constants in order to select values that will ensure convergent behavior.

2.2. The guaranteed convergence particle swarm optimizer

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 Eq. (3) depends only on the $w\mathbf{v}_i(t)$ term. When a particle approaches the global best solution, its velocity approaches zero, implying 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 [38,41]. Van den Bergh et al. introduced a new algorithm, called the Guaranteed Convergence PSO (GCPSO) [38,40],

to pro-actively counteract this behavior in a particle swarm. Let τ be the index of the global best particle. The velocity and position updates for the global best particle are then redefined to be

$$v_{\tau,j}(t+1) = -x_{\tau,j}(t) + \hat{y}_j(t) + wv_{\tau,j}(t) + \rho(t)(1 - 2r_{2,j}), \quad (5)$$

$$x_{\tau,j}(t+1) = \hat{y}_j(t) + wv_{\tau,j}(t) + \rho(t)(1 - 2r_{2,j}). \quad (6)$$

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. The parameter $\rho(t)$ is dynamically adapted to control the size of the bounding box around $\hat{y}$ within which a local search is conducted to force a change in the value of $\hat{y}$, thereby preventing the above problem [38,40].

2.3. Niching ability of standard PSO

This section provides a short summary of [14], where it is shown that the standard *gbest* PSO cannot locate multiple solutions, and that the standard *lbest* PSO is inefficient in doing so. For the *gbest* PSO, Van den Bergh and Engelbrecht [38,41], and Clerc and Kennedy [6] formally proved that all particles converge on a single attractor, which is a weighted average of the global best position and the personal best position of the particle. These formal proofs clearly show that the *gbest* PSO cannot locate, in the same execution of the algorithm, more than one solution. If the *gbest* PSO is executed a number of times, each time from different initial conditions, it may be the case that more than one unique solution is found. However, there is no such guarantee. In fact, there is not even a guarantee that the solution found is a local optimum, as proven in [38].

The proof of convergence to the weighted average of personal best and global best positions have been done with respect to *gbest* PSO, and cannot be directly applied to *lbest* PSO. Based on this proof, the only thing that can be said is that for each neighborhood, a particle converges to a weighted average of its personal best and neighborhood best solutions. Since neighborhoods are constructed based on particles indices with some overlap between neighborhoods, it is intuitively expected that the neighborhoods (and therefore all particles), will eventually converge on the same point. However, no such proof exist. Engelbrecht et al. [14] empirically showed that, for the *lbest* PSO, particles do form subgroups, which can be perceived as niches. Even if these subgroups can be considered as niches, it was shown in [14] that only a small number of solutions are found, and only when large swarm sizes are used. The conclusion from [14] is that the standard *lbest* PSO is inefficient in locating multiple solutions. For sake of completeness, Section 5 summarizes the results from [14] in comparison with the results of NichePSO.

The inability of *gbest* PSO and the inefficiency of *lbest* PSO in locating multiple solutions motivates research in the modification of the standard PSO to promote niching within a single swarm.

3. Niching techniques

A number of algorithms have been suggested to find multiple solutions to multi-modal optimization problems using genetic algorithms, and to a lesser extend, PSOs. In GA parlance, optimization techniques that locate multiple optima in multi-modal function optimization problems are known as *niching* techniques. Both GAs and PSOs use a population of 'agents' (individuals, particles) partitioned in some way to focus on and locate different possible solutions in a single search space. Each subgroup in the partitioned population is known as a species. The behavioral pattern of individuals competing for the use of a resource in a subgroup and between elements in a subgroup, is known as *speciation*.

Section 3.1 defines niching, while the remainder of this section summarizes only those niching methods from which NichePSO borrowed concepts, and those methods used in the empirical analysis done for this paper.

3.1. 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 compe-*

tition between such areas, and causing the formation of species for each niche'' [19]. 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 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 in an environment around multiple resources form the basis for the study of niching techniques with evolutionary optimization algorithms. In the evolutionary context, Horn defines *implicit niching* as the sharing of resources, and *explicit niching* as the sharing of fitness.

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

- *Sequential niching* (or temporal niching) techniques develop niches sequentially over time. As niches are discovered, the search space of a problem is adapted to repel other individuals from traversing the area around the recently located solution. The search is repetitively applied to the adapted search space in order to focus on unexplored areas [2].
- *Parallel niching* forms and maintains several different niches simultaneously. The search space is not modified. Parallel niching techniques therefore 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 [27,17,19,29].

Regardless of the way in which niches are found (i.e. in parallel or sequentially), the distribution of individuals can be formalized in a number of ways, according to their speciation behavior [27]:

- *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).
- *Allopatric speciation* differentiates between 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.
- *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.

The PSO niching approach presented in Section 4 may 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 multi-modal function optimization.

3.2. Fitness sharing

Fitness sharing is one of the earliest GA niching techniques, originally introduced as a population diversity maintenance technique [17]. 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 i is adapted to its shared fitness:

$$f_i' = \frac{f_i}{\sum_j \mathrm{sh}(d_{i,j})}.$$

A common sharing function is

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

The symbol d represents a distance calculated between individuals i and j. The distance measure may be genotypic or phenotypic, depending on the optimization problem at hand. If the sharing function finds that $d_{i,j}$ is less than σ_{share}, it returns a value in the range $[0,1]$, which increases as $d_{i,j}$ decreases. The more similar i and j, the lower their individual fitnesses will become. Sharing assumes that the number of niches can be estimated, i.e. it must be known prior to the application of the algorithm how many niches there are. It is also assumed that niches occur at least a minimum distance, $2\sigma_{\text{share}}$, from each other.

3.3. Sequential niching

Sequential niching (SN), introduced by Beasley et al. [2], 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. A derating function is designed to lower the fitness appeal of previously located solutions. By repeatedly running the algorithm, all optima are removed from the fitness landscape. Sample derating functions, for 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

$$G_2(\mathbf{x},\mathbf{x}^*) = \begin{cases} e^{\log m \frac{r-\|\mathbf{x}-\mathbf{x}^*\|}{r}} & \text{if } \|\mathbf{x}-\mathbf{x}^*\| < r \\ 1 & \text{otherwise,} \end{cases} \tag{7}$$

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'. Noting that $\lim_{x\to 0}\log(x) = -\infty$, m must always be larger than 0. Smaller values for m result in a more concave derating function. The fitness function $f(\mathbf{x})$ is then redefined to be

$$M_{n+1}(\mathbf{x}) \equiv M_n(\mathbf{x}) \times G(\mathbf{x},\mathbf{s}_n),$$

where $M_0(\mathbf{x}) \equiv f(\mathbf{x})$ and $\mathbf{s}_n$ is the best individual found during run n of the algorithm. G can be any derating function, such as G_1 and G_2.

3.4. Crowding

Crowding (or the crowding factor model), as introduced by de Jong [10], was originally devised as a diversity preservation technique. Crowding is 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 the population size will remain 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.

Deterministic crowding (DC) is based on de Jong's crowding technique, but with the following improvements as suggested by Mahfoud [27]:

- Phenotypic similarity measures are used instead of genotypic measures. Phenotypic metrics embody domain specific knowledge that is most useful in multi-modal optimization, as several different spatial positions can contain equally optimal solutions.

– It was shown that there exists a high probability that the most similar individuals to an offspring are its parents. Therefore, DC compares an offspring only to its parents and not to a random sample of the population.
– Random selection is used to select individuals for reproduction. Offspring replace parents only if the offspring perform better than the parents.

3.5. Objective function stretching

Objective function stretching [30] was applied as a sequential PSO niching technique [31], similar to that of Beasley et al. [2]. A particle swarm is trained using the *gbest* algorithm. Once the PSO has identified a local minimum $f(\mathbf{x}^*)$, through comparing particle fitnesses to a performance threshold value, the objective function is *stretched* such that for each point $\mathbf{x}$, where $f(\mathbf{x}) < f(\mathbf{x}^*)$, $\mathbf{x}$ is unaffected. All other points, such that $f(\mathbf{x}) \geqslant f(\mathbf{x}^*)$ 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 as $H(\mathbf{x})$, 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

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

For a minimization problem, the sign(·) 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. Stretching of $f(\mathbf{x})$ to $H(\mathbf{x})$ ensures that subsequent iterations of the PSO algorithm does not focus on previously located solutions.

Although effective in global optimization and to a lesser extent in multi-modal optimization, the stretching technique introduces the following problems:

- If multiple acceptable solutions are located close to each other, the effect of $G(\mathbf{x})$ may cause these alternative solutions never to be detected.
- The adaptation of $f(\mathbf{x})$ close to $\mathbf{x}$ leads to the introduction of 'trenches' in f around remaining potential solutions with fitness similar to that of $\mathbf{x}$ [4].
- For some combinations of γ_1, $\gamma_2\mu$, new local optima are introduced [38].

3.6. The nbest particle swarm optimizer

The *nbest* PSO was developed to solve unconstrained systems of equations (SEs) [5]. Solving SEs here specifically refer to the process of finding points of intersection between the individual equations making up a SEs. Standard PSO techniques, such as *gbest* and *lbest* can quickly and accurately solve SEs that have a single solution. However, it is frequently the case that complex SEs have multiple solutions. Such problems can be classified as multi-modal problems. Solutions to the SEs all have optimal fitness, and the fitness of any other potential, non-optimal solution, depends on its proximity to one of the optimal solutions.

For a potential solution $\mathbf{x}_i$, the fitness function for solving a SE can in general be written as

$$f(\mathbf{x}_i) = \sum_{k=1}^{K} |z_k(\mathbf{x}_i)|,$$

where $z_k(\mathbf{x}_i)$ represents each one of the K equations in the SE; each equation is algebraically rewritten to be equal to zero. While this fitness function works well when an unique solution exists, it fails for multiple solutions. For example, consider the following SE:

$$y = 2x - 3,$$
$$y = -3x - 1,$$
$$y = -x + 1.$$

The SE has three solutions (i.e. there are three intersections between the solutions). To find them all, the fitness function should consider a particle's relative distance to a possible solution. Thus, to evaluate the fitness of $\mathbf{x}$ for this SE, the fitness function is redefined as

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

where

$f_{\mathrm{AB,AC}}(\mathbf{x})$ is the fitness of particle $\mathbf{x}$ with respect to equations $y = 2x - 3$ and $y = -3x + 1$,
$f_{\mathrm{BA,BC}}(\mathbf{x})$ is the fitness of particle $\mathbf{x}$ with respect to equations $y = -3x + 1$ and $y = -x + 1$,
$f_{\mathrm{CB,CA}}(\mathbf{x})$ is the fitness of particle $\mathbf{x}$ 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, and rewards a particle for being close to *one* of the solutions, and does not penalize it if the particle is still far from the global best particle.

If there are no intersections between lines in a system of equations, and therefore no solution to the system of equations, particles will eventually settle on locations where lines in the system are the closest to each other.

The *lbest* model's neighborhood definition is motivated by the fact that it tries to promote the spread of information regarding good solutions to all particles, without considering a particle's current position. When searching for a single optimum solution, this model is appropriate as it allows for efficient evaluation of the search space while avoiding premature convergence. However, when searching for multiple possible solutions, *lbest* is biased towards finding a single optimum solution in the search space due to overlapping neighborhoods. It has been experimentally shown that it is possible for index-based particle neighborhoods to emerge for relatively high iterations of the *lbest* algorithm (see results presented in Section 5.5). The propagation of information about a local best solution within an index-based neighborhood, where neighborhoods are unique to particles, also hampers convergence. Neighborhoods constantly change as particles do not pursue a common goal. *nbest* PSO however attempts to compensate for this deficiency.

For *nbest* PSO, a neighborhood best, $\hat{\mathbf{y}}_i$, is defined for each particle, $\mathbf{x}_i$, as the center of mass of the positions of all the particles in the *topological* neighborhood of $\mathbf{x}_i$. The topological neighborhood is defined as the n_i closest particles to $\mathbf{x}_i$, 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 $\mathbf{B}_i$, where $\mathbf{B}_i$ consists of the k closest particles to $\mathbf{x}_i$ at any given time step t; $\hat{\mathbf{y}}_i$ is then

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

where B_{ij} is the current position of the jth particle in neighborhood B_i of particle $\mathbf{x}_i$ at time t; k is a user defined parameter. Considering the above formulation of $\hat{\mathbf{y}}_i$, k should not be too small, as it will force a particle to blindly trail its closest neighbor. Also, if k is too large, it would yield an algorithm similar to *gbest*, but where the goal position would be an average particle position conveying no information about a possible good result. Particle updates are done as for normal *gbest* PSO, but with the difference that the position of the global best particle, $\hat{\mathrm{y}}$, is replaced with $\hat{\mathbf{y}}_i$, which represents the average of the best positions of particles in the topological neighborhood of particle $\mathbf{x}_i$.

Experimental results showed that *nbest* successfully locates multiple solutions to SEs [5,4].

4. The niching particle swarm optimization algorithm

The *NichePSO* algorithm is a PSO based niching technique. Niches are identified by monitoring changes in the fitness of individual particles in a swarm. The initial swarm is known as the 'main' swarm. When a particle representing a candidate solution is found, a subswarm is created with this particle and its closest neighbor. Subswarms are grown from the 'main' swarm. When all particles from the main swarm have been depleted, no further subswarms can be created. Subswarms that occupy regions in the search space that may represent the same niche are merged. To avoid this problem, any particle that occupies a position that can be considered to fall within an existing swarm (discussed next) is also added to the swarm.

Fig. 1 summarizes the NichePSO algorithm. A number of issues relating to the algorithm are now discussed:

Initialization: The general location of potential solutions in a search space may not always be known in advance. It is therefore a good policy to distribute particles uniformly throughout the search space before learning commences. To ensure a uniform distribution, *Faure*-sequences are used to generate initial particle positions (as described in [37]). *Faure*-sequences are distributed with high uniformity within a n-dimensional unit cube. Other pseudo-random uniform number generators, such as Sobol-sequences [33], may also be used.

Main swarm training: In the *nbest* algorithm, overlapping particle neighborhoods discourage convergence on local optima [4]. 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 converge on a local optimum as it is not drawn to a position in the search space that has better fitness as a result of the traversal of other particles. This search approach has been previously investigated [21]. Kennedy referred to it 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.

Identification of niches: A fundamental question when searching for different niches is how to identify them. Parsopoulos et al. (see Section 3.5) used a threshold value ϵ such that when $f(\mathbf{x}_i) < \epsilon$ for particle i, the particle is removed from the swarm and labelled as a potential global solution. The objective function's landscape is then stretched to keep other particles from exploring this area in the search space. If the isolated particle's fitness is not close to a desired level, the solution can be refined by searching the surrounding

(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.

Fig. 1. NichePSO algorithm.

function landscape with the addition of more particles. This approach proves to be effective when considering Parsopoulos et al.'s results. This threshold parameter ϵ is however subject to fine tuning, and locating good solutions with it 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 showed very little change over a small 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 our experiments. When $\sigma_i < \delta$, a subswarm may be created with $\mathbf{x}_i$. The threshold δ is a much more intuitive parameter than ϵ. To avoid problem dependence, σ_i is normalized according to $x_{\min}$ and $x_{\max}$. It is possible that this approach can find *local* minima, satisfying $\sigma_i < \delta$. 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 $\mathbf{x}_i$ is simply the particle $\mathbf{x}_k$ where

$$\mathbf{x}_k = \arg\min_{\mathbf{x}_k}\{\|\mathbf{x}_i - \mathbf{x}_k\|\},$$

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

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:

- Including a particle traversing 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

$$\|\mathbf{x}_i - \hat{\mathbf{y}}_{S_j}\| \leqslant R_j, \tag{8}$$

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

$$R_j = \max\{\|\hat{\mathbf{y}}_{S_j} - \mathbf{x}_{S_{j,i}}\|\}. \tag{9}$$

$\mathbf{x}_{S_{j,i}}$ 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 which occurs in a vastly different position in the search space, the algorithm's convergence would be impaired.

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 already optimizing the particular solution. If the particle has an acceptable fitness, another subswarm will be created at 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 maintained. This introduces a dilemma, as multiple swarms will attempt to optimize the same solution. To alleviate this situation, 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_{j_1} and S_{j_2} *intersect*, when

$$\|\hat{\mathbf{y}}_{S_{j_1}} - \hat{\mathbf{y}}_{S_{j_2}}\| < (R_{j_1} + R_{j_2}). \tag{10}$$

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, Eq. (10) fails to detect the presence of multiple subswarms in the same niche. Consequently, when two swarms, S_{j_1} and S_{j_2} do not satisfy Eq. (10), because $R_{j_1} = R_{j_2} = 0$,[1] the subswarms can be merged when

$$\|\hat{\mathbf{y}}_{S_{j_1}} - \hat{\mathbf{y}}_{S_{j_2}}\| < \mu. \tag{11}$$

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, $\|\hat{\mathbf{y}}_{S_{j_1}} - \hat{\mathbf{y}}_{S_{j_2}}\|$ is normalized to the interval $[0, 1]$. S_{j_1} and S_{j_2} are merged by creating a new subswarm consisting of all S_{j_1} and S_{j_2}'s particles. The influence of different μ values are discussed in Section 5.3.1. An upper bound on μ is empirically derived.

The GCPSO algorithm (*refer to* Section 2.2):
Subswarms created by the NichePSO algorithm initially always consist of two particles. Utilizing the *gbest* algorithm, especially when these particles are topologically similar, may lead to swarm stagnation, forcing the subswarm to converge on a sub-optimal solution. GCPSO puts measures in place to ensure that a swarm does not stagnate. Performance differences between these two algorithms are discussed in Section 5.3.3.

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. 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 can not, however, detect acceptable solutions in a multi-modal 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 no significant 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 may also be stopped after a maximum number of training iterations.

5. Experimental results

This section presents results obtained by applying NichePSO to find multiple solutions to a set of multi-modal problems. The performance of NichePSO is compared to that of *nbest* PSO, *lbest* PSO, sequential niching and deterministic crowding. The influence of different control parameter values for NichePSO is also investigated, and an initial scalability study is included.

5.1. Test functions

NichePSO is tested on a number of multi-modal functions, where the goal is to identify all optima. These functions were originally introduced by Goldberg and Richardson to test fitness sharing [17] and have also been used by Beasley et al. to evaluate the sequential niching algorithm [2]. Functions F1–F4 are defined as (refer to Fig. 2):

[1] Since position updates in PSO is a stochastic process, it is practically safer to consider the situation where $R_{j_1} \approx 0$ and $R_{j_2} \approx 0$.

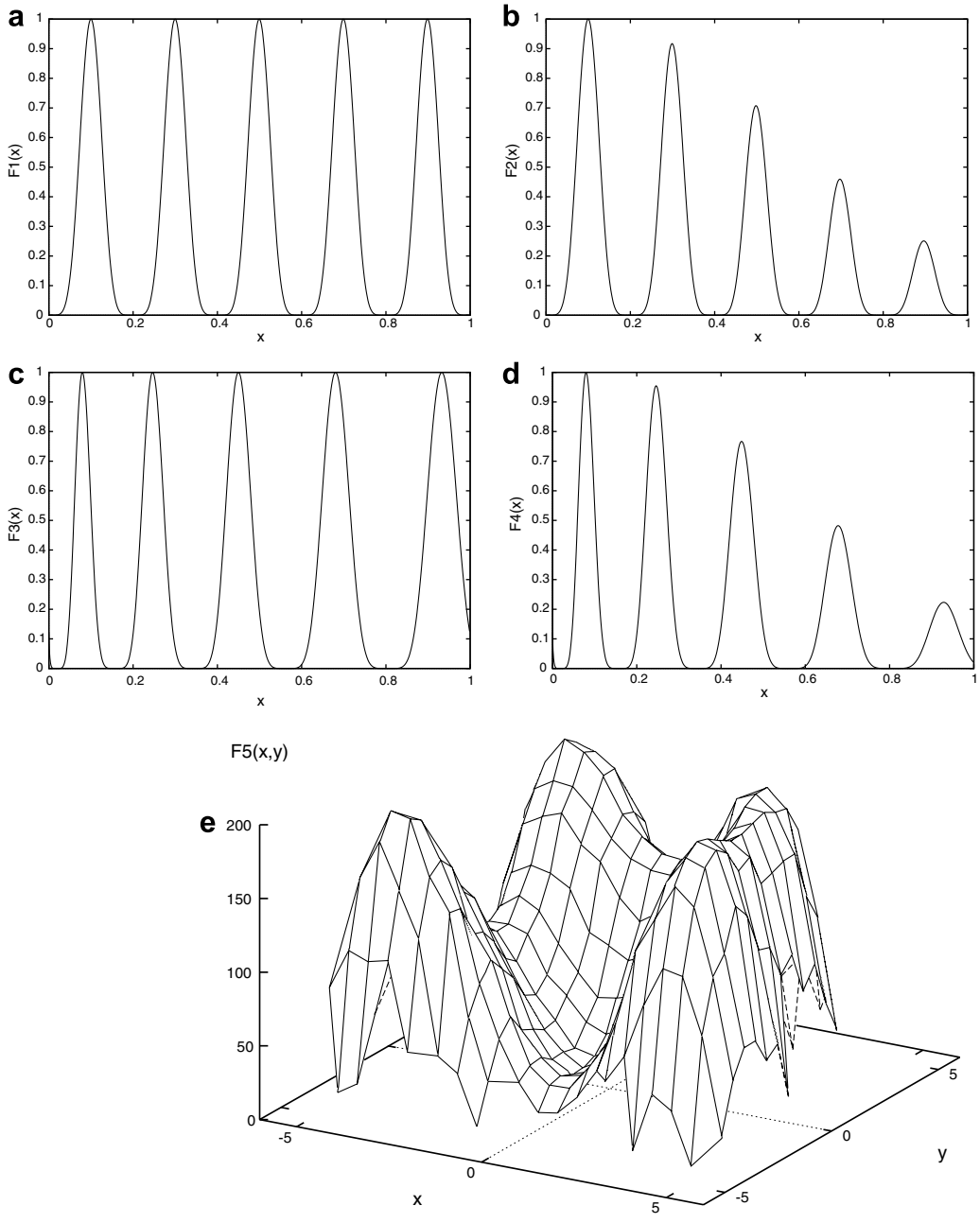

Fig. 2. Test functions: (a) function F1, (b) function F2, (c) function F3, (d) function F4 and (e) function F5.

$$\mathrm{F1}(x) = \sin^6(5\pi x),$$
$$\mathrm{F2}(x) = \left(e^{-2\log(2)\times\left(\frac{x-0.1}{0.8}\right)^2}\right) \times \sin^6(5\pi x),$$
$$F3(x) = \sin^6(5\pi(x^{3/4} - 0.05)),$$
$$F4(x) = \left(e^{-2\log(2)\times\left(\frac{x-0.08}{0.854}\right)^2}\right) \times \sin^6(5\pi(x^{3/4} - 0.05)).$$

Functions F1 and F3 both have five 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–F4 are investigated in the range $x \in [0, 1]$. For F1 and F2, maxima are located at $x = 0.1, x = 0.3, x = 0.5, x = 0.7$ and $x = 0.9$. For F3 and F4, maxima are located at $x = 0.08, x = 0.25, x = 0.45, x = 0.68$ and $x = 0.93$.

Function F5, the modified Himmelblau function (see Fig. 2(e)), is defined as

$$\mathrm{F5}(x, y) = 200 - (x^2 + y - 11)^2 - (x + y^2 - 7)^2. \tag{12}$$

F5 has four equal maxima with $\mathrm{F5}(x, y) = 200$. Maxima are located at $A = (3.0, 2.0), B = (-3.78, -3.28)$, $C = (3.58, -1.85)$ and $D = (-2.81, 3.13)$.

5.2. Preliminary results

For each of the five test functions, 30 experiments were done with the NichePSO algorithm with $c_1 = c_2 = 1.2$. The inertia weight w was scaled linearly from 0.7 to 0.2 over a maximum of 2000 iterations of the algorithm. These parameters were chosen to ensure convergent trajectories [41]. A single NichePSO iteration is defined as performing steps 2–8 in Fig. 1 once. Table 1 reports further parameter settings, where $|S|$ denotes the initial number of particles in the main swarm before any niche subswarms are created, μ is the subswarm merging threshold, and δ is the subswarm creation threshold. A maximum velocity equal to the maximum range, $x_{\max}$, was used. 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:* how close the discovered optima are to the actual solutions; and
- *Success consistency:* the proportion of the experiments that found all optima.

The parameter values for δ and μ presented in Table 1 have been experimentally found to be effective.

Table 2 reports the mean and standard deviation of fitness of all particles in all subswarms. *%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 experiments. This explains the relatively large difference in fitness between functions F1 and F3, and functions F2 and F4. Beasley et al. reported similar results for functions F1–F4, and only found all maxima in 76% of the experiments done for F5 [2].

Table 1
Parameter settings

Function	δ	μ	$\lvert S\rvert$	$x_{\min}$	$x_{\max}$
F1	10^{-4}	10^{-3}	30	0.0	1.0
F2	10^{-4}	10^{-3}	30	0.0	1.0
F3	10^{-4}	10^{-3}	30	0.0	1.0
F4	10^{-4}	10^{-3}	30	0.0	1.0
F5	10^{-4}	10^{-3}	20	−5.0	5.0

Table 2
Performance results

Function	Fitness	Deviation	% 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

5.3. Discussion

Unimodal optimization techniques, such as the standard PSO and GAs, fail to efficiently locate multiple solutions to multi-modal 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* when used in step 4(a) of the algorithm given in Fig. 1,
- the relationship between the initial swarm size and the number of solutions in a multi-modal fitness function, and
- scalability of the algorithm on highly multi-modal functions.

5.3.1. Sensitivity to changes in μ

Each subswarm created by the NichePSO algorithm can be seen as a hyper-sphere in a 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 represented by them 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, Eq. (10) fails to allow similar swarms to be merged. Therefore, the use of the μ parameter was introduced in Eq. (11). μ allows virtually identical swarms to be merged when they occupy positions that are almost equal. Large μ values 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, global solution.

In Table 3, the normalized distances between the known solutions for function F5 are given. Symbols A–D correspond to the solutions listed in Section 5.1. For $\mu = 0.5$, the NichePSO algorithm failed to locate all the maxima for function F5. The normalized distance between solutions A and D is 0.493. Swarms that represented 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. For extremely small μ-values, NichePSO still successfully located all solutions, but not all swarms positioned around the same solutions were merged. Swarms congregated around the solutions, but due to the 'strict' merging threshold, they could only be merged when virtually identical. Fig. 6a and b plot the mean number of solutions found by NichePSO for F5, F1 and F3 respectively, for different μ values. For function F1, NichePSO located all solutions when $\mu < 0.2$ (see Fig. 6b). For F3, all solutions were only located for $\mu \leqslant 0.1$. These results can easily be verified by inspecting the inter-solution distances for functions F1 and F3.

From Fig. 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 [17], and the niche radius r in Beasley et al.'s SN technique [2].

5.3.2. Sensitivity to changes in δ

To identify new potential solutions, the NichePSO algorithm monitors changes in the main particle swarm. If any particle exhibits very little change in its position over a number of iterations of the algorithm, the par-

Table 3
Normalized inter-solution distances for function F5

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

ticle may have 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 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.

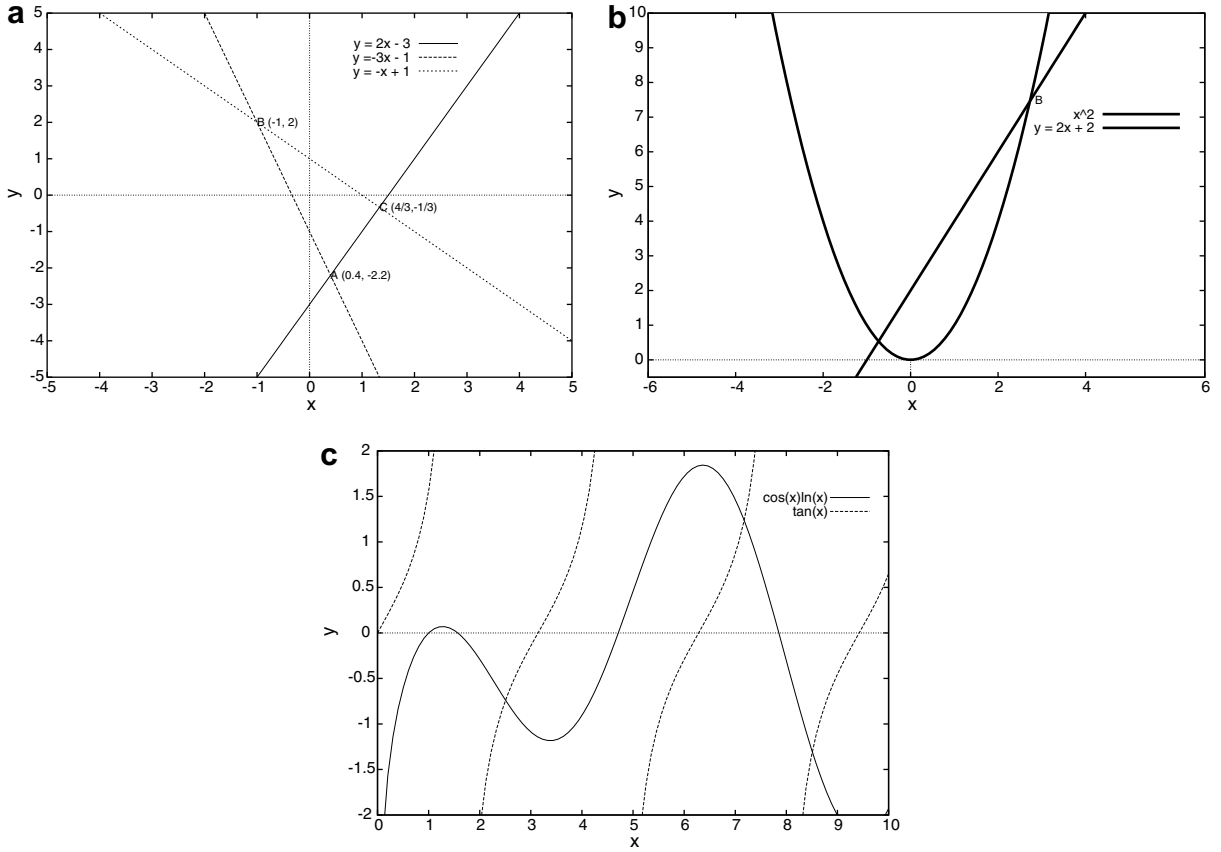

Fig. 3. Example systems of equations: (a) S1, (b) S2 and (c) S3.

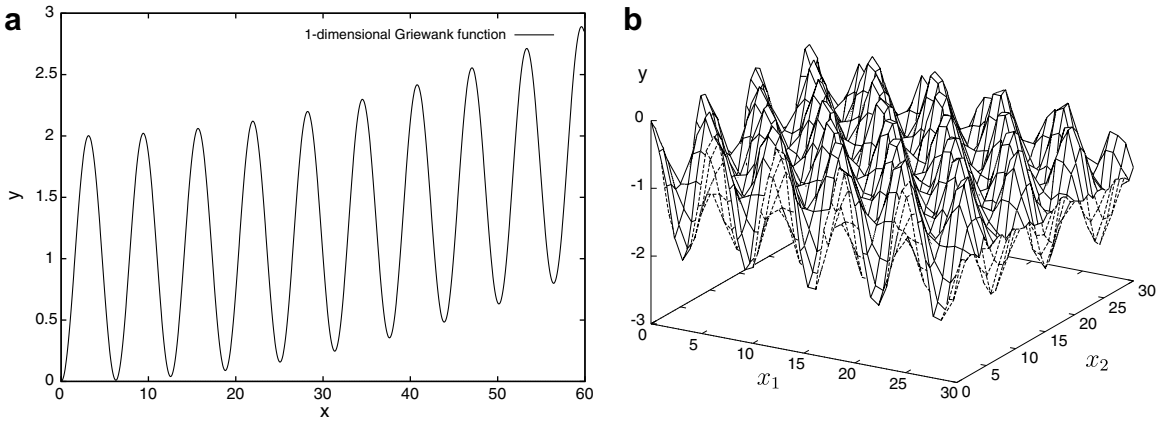

Fig. 4. Griewank function: (a) one-dimensional and (b) two-dimensional.

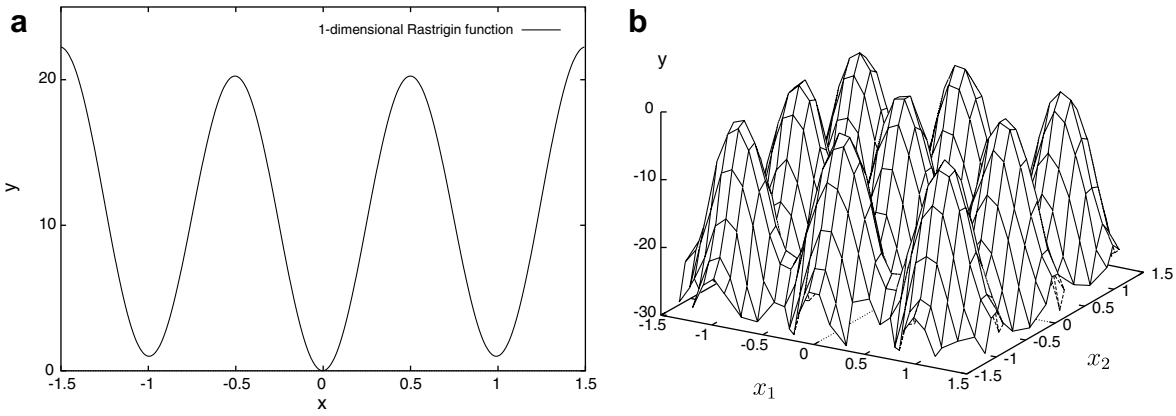

Fig. 5. Rastrigin function: (a) one-dimensional and (b) two-dimensional.

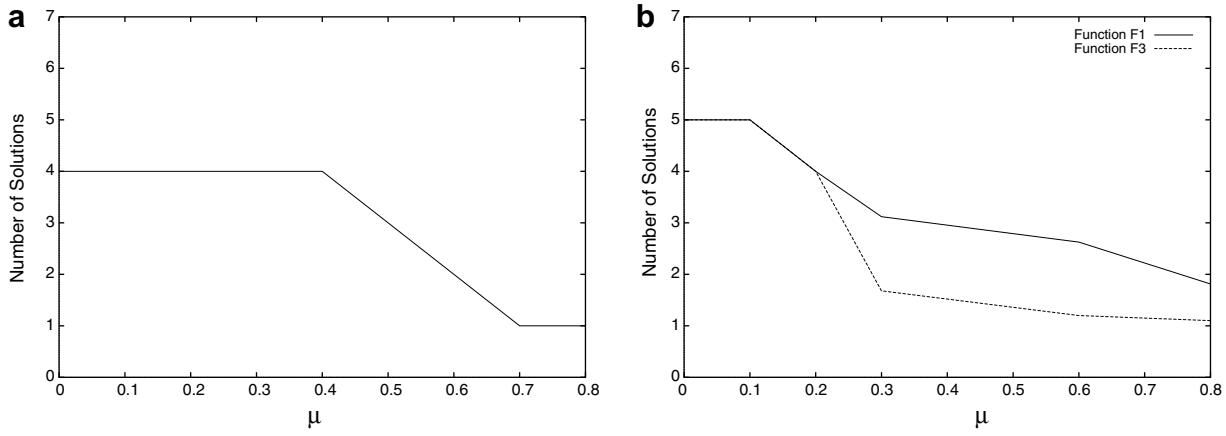

Fig. 6. Effect of changes in μ: (a) number of solutions vs. μ for function F5 and (b) number of solutions vs. μ for function F1 and F3.

Different δ values were tested for functions F1, F3 and F5. Fig. 7 plots the mean number of fitness function evaluations over 30 experiments against different δ settings. An experiment 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 stagnation behavior. In this context, 'stagnation 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 Fig. 7, smaller δ values effected a slight increase in the number of fitness function evaluations required before the algorithm converged.

Fig. 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.

5.3.3. The subswarm optimization technique

The NichePSO algorithm uses the GCPSO technique (refer to Section 2.2) as subswarm optimization technique. This section elucidates the use of GCPSO where *gbest* would be expected to perform adequately.

When considering the particle velocity update given in Eq. (3), 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

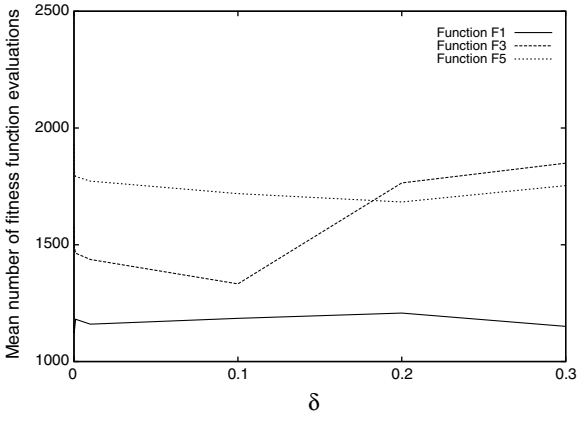

Fig. 7. Mean number of fitness function evaluations required for different δ values.

velocity update for the next 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 almost immediately stagnate. The situation can be 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 particle 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

$$\mathbf{v}_p(t+1) = w\mathbf{v}_p(t). \tag{13}$$

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, 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, as 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. The same argument can be applied to the *lbest* PSO, but with reference to the neighborhood best instead of the global best. It should be noted that this behavior will be exhibited only when the problem occurs for all the neighborhoods. To detect and avoid the described situation, the GCPSO algorithm is employed. GCPSO uses adapted velocity and position update equations for the global best particle in a swarm (in our case particle p), that allows efficient local traversal of the search space.

Table 4 presents experimental results that compare the performance of GCPSO with *gbest* PSO and *lbest* PSO as subswarm optimization technique. For the *lbest* PSO, all parameters are the same as for the *gbest* PSO. Neighborhoods of size 2 have been used. *% Convergenced* is the average success rate for finding *all* solutions of

Table 4
% 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

a test function over 30 experiments. It is clear that GCPSO was better suited towards maintaining niches than *gbest* PSO and *lbest* PSO. Experimental results obtained showed that for all functions, when using *gbest* PSO, it frequently occurred that subswarms were formed that consisted of only two particles. Such swarms quickly stagnated on suboptimal locations. The same problem was observed with *lbest* PSO. The *lbest* PSO performed consistently better than the *gbest* PSO, except for function F5.

When several local and global optima exist in close proximity to each other in a function, GCPSO would tend to be biased towards the global optima. This was not the case for the *gbest* PSO and *lbest* PSO.

5.3.4. 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 multi-modal function.

Fig. 8a and b 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, 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, specifically towards a solution. If possible solutions are not known in advance, such a bias 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 $\|S\| \geqslant 20$, NichePSO successfully located and stably maintained all solutions. For all the tested functions, a swarm of size $\|S\| \geqslant a^2$ managed to locate all solutions. Fig. 8b shows the mean number of fitness function evaluations required for the different swarm sizes used above.

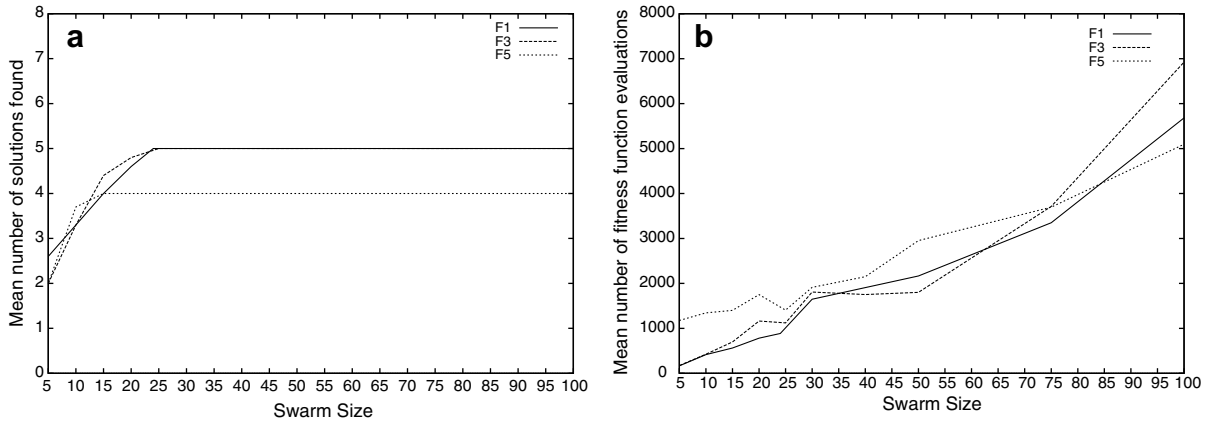

Fig. 8. Performance under different swarm sizes: (a) relationship between different swarm sizes and the number of solutions locates and (b) the mean number of fitness function evaluations required for different swarm sizes.

5.4. Comparison with other niching approaches

This section compares the performances of NichePSO, *nbest* PSO, *lbest* PSO, sequential niching (SN) and deterministic crowding (DC) using the same parameter settings for all algorithms. For this purpose, functions F1–F5 are used, as well as the following systems of equations (as illustrated in Fig. 3):

$$\begin{aligned} \text{S1}: \ & y = 2x - 3, \\ & y = -3x - 1, \\ & y = -x + 1, \end{aligned} \tag{14}$$

$$\begin{aligned} \text{S2}: \ & y = x^2, \\ & y = 2x + 2, \end{aligned} \tag{15}$$

$$\begin{aligned} \text{S3}: \ & y = \cos x \ln x, \\ & y = \tan x. \end{aligned} \tag{16}$$

For each of the *nbest* PSO, *lbest*, NichePSO, SN and DC algorithms, 30 simulations were performed on each of the systems of equations above. The following sections describe GA and PSO parameter settings respectively.

5.4.1. GA setup

For all experiments, populations consisting of 20 individuals were used. For Beasley et al.'s SN algorithm, the probabilities of crossover and mutation were set to 0.9 and 0.01 respectively [2]. mutation operators are applied. A single-point crossover operator was used. As selection operator, *stochastic universal sampling* (SUS) were used, as suggested in [26]. 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 [2], 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 (refer to Section 3.3), the method suggested by Beasley et al. was used (originally suggested by Deb [11]). For a d-dimensional problem with l optima, the niche radius r was calculated as

$$r = \frac{\sqrt{d}}{2 \times \sqrt[d]{l}}. \tag{17}$$

This technique assumes that fitness function parameters are normalized to $[0, 1]$. The exponential derating function in Eq. (7) was used.

Mahfoud's DC uses an internal selection scheme (see Section 3.4). Crossover and mutation probabilities were set at 1.0 and 0.01 respectively, since the DC algorithm favors a low mutation probability and a high crossover probability [27]. DC also used a halting window-based termination criterion of $h = 20$, and a maximum number of generations of $g_{\max} = 2000$.

The PSO implementations and NichePSO used the same values for control parameters as used in the previous sections and summarized in Table 1 (except that 20 particles have been used for all functions).

5.5. Results and discussion

Tables 5 and 6 summarize 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* PSO 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, topological neighborhoods of particles overlap. Particles are therefore drawn to solutions with better fitness, and only converge on solutions with optimal fitness.

Table 5
Average number of fitness function evaluations required to converge for each niching algorithm

Problem	SN	DC	*nbest*	NichePSO	Average
S1	1841 ± 86	15088 ± 4197	8493 ± 413	2554 ± 228	6994
S2	1193 ± 96	17554 ± 4657	6934 ± 542	3966 ± 353	7412
S3	1927 ± 95	13816 ± 4225	7019 ± 670	2704 ± 135	6366
F1	4102 ± 577	14647 ± 4612	4769 ± 45	2372 ± 109	6473
F2	3505 ± 463	13052 ± 2507	*	2934 ± 475	6497
F3	4141 ± 554	13930 ± 3284	4789 ± 51	2404 ± 195	6316
F4	3464 ± 287	13929 ± 2996	*	2820 ± 517	6738
F5	3423 ± 402	14296 ± 3408	5008 ± 562	2151 ± 200	6220
Average	2950	14539	6169	2738	

Entries marked with a '*' indicate that experiments were not carried out for the relevant problem and algorithm.

Table 6
The consistency with which each of the techniques managed to locate a complete set of solutions for each of the test problems

Problem	SN (%)	DC (%)	*nbest* (%)	*lbest* (%)	NichePSO (%)	Average (%)
S1	76	93	100	0.00	87	89.00
S2	66	100	100	0.00	80	86.50
S3	83	87	100	0.00	90	90.00
F1	100	100	93	0.00	100	98.25
F2	83	93	*	0.00	93	89.67
F3	100	90	93	0.00	100	95.75
F4	93	90	*	0.00	93	92.00
F5	86	90	100	0.00%	100	94.00
Average	85.88	92.86	97.67	0	92.88	

Entries marked with a '*' indicate that experiments were not carried out for the relevant problem and algorithm.

5.5.1. Computational cost

Table 5 compares the computational cost of each of the niching techniques in terms of the number of fitness function evaluations required to converge. For 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. 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 5:

- SN required fewer evaluations on the systems of equations in problems S1, S2 and S3.
- 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, which becomes more complex as the number of generations increases.
- DC consistently required more fitness function evaluations than the other algorithms.
- When comparing *nbest* PSO 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* PSO also required the calculation of an inter-particle distance matrix at each iteration of the algorithm.
- Both *nbest* PSO and NichePSO consistently require less than the average number of fitness function evaluations to converge.
- Overall, NichePSO required the least number of fitness function evaluations to converge.
- Results for *lbest* PSO were not included as the algorithm failed to locate a complete set of solutions for any of the test problems using the same set of parameters (see Table 6).

5.5.2. Performance consistency

Table 6 expresses as percentages the performance consistency of the tested niching techniques. 'Performance consistency' reflects each algorithm's ability to consistently locate all solutions to each of the optimization problems for the given parameters. All the tested techniques sufficiently maintained solutions sets. The following conclusions can be drawn from the results reported in Table 6:

- Results for the SN algorithm compare well to that reported in [2]. 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 S1, S2 and S3.
- The *nbest* PSO 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).

For the given set of test problems and control parameters, the *lbest* PSO was not successful in finding a complete set of solutions, although the algorithm did manage to locate some of the solutions, as shown in Table 7. These results prompted an experimental investigation into the potential of *lbest* PSO using more favorable parameters. A set of 30 experiments were performed using *lbest* PSO on function F1, with swarms consisting of 100 particles over 100,000 iterations of the algorithm. Settings for w, c_1 and c_2 were as above. Table 8 show that *lbest* PSO was only successful in locating all solutions 13.33% of the time. In order for *lbest* PSO to locate a complete set of solutions for a multi-modal problem, particles should be re-arranged during the execution of the algorithm ensuring that index-based neighborhoods reflect the topological arrangement of solutions within an optimization problem's search space. This trend was not apparent for results reported in Table 6, but did emerge for larger swarms. It is however significant that regardless of the high number of iterations and particles, the *lbest* PSO was less successful than the investigated niching techniques. This finding is confirmed by Engelbrecht et al. in a separate study of the niching ability of basic PSO [14].

Table 7
lbest PSO performance on test functions

Function	Solutions	% Experiments converged on # solutions				
		1 Sol (%)	2 Sols (%)	3 Sols (%)	4 Sols (%)	5 Sols (%)
F1	5	6.67	53.33	33.33	6.67	0.00
F2	5	60.00	20.00	20.00	0.00	0.00
F3	5	0.00	33.33	60.00	6.67	0.00
F4	5	20.00	33.33	40.00	0.00	0.00
F5	4	66.67	26.67	0.00	0.00	–
S1	3	80.00	20.00	0.00	–	–
S2	2	100.00	0.00	–	–	–
S3	3	80.00	20.00	0.00	–	–

Table 8
lbest performance with large swarms

# Solutions	Located by # experiments	
1	0	0.00%
2	0	0.00%
3	13	43.33%
4	13	43.33%
5	4	13.33%
	30	100.00%

5.6. Scalability of NichePSO

The results obtained in Section 5.2 showed NichePSO to effectively solve multi-modal optimization problems. This section presents empirical results that investigate NichePSO's ability to scale to massively multi-modal domains.

NichePSO was tested on the following multi-modal functions:

Griewank function:

$$f(\mathbf{x}) = \left(\frac{1}{4000}\sum_{i=1}^{n} x_i^2\right) - \left(\prod_{i=1}^{n} \cos\left(\frac{x_i}{\sqrt{i}}\right)\right) + 1. \tag{18}$$

Rastrigin function:

$$f(\mathbf{x}) = \sum_{i=1}^{n} [x_i^2 - 10\cos(2\pi x_i) + 10]. \tag{19}$$

These functions are massively multi-modal. 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 increases exponentially, as can be seen from the one and two dimensional plots given in Figs. 4 and 5. Figs. 4b and 5b are drawn inverted to more clearly illustrate the multi-modal 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 9 and 10. 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 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 9 and 10 present performance results of NichePSO on the two test functions.

The following observations could 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.

Table 9
Scalability performance on the Griewank function

Dimensions (n)	Number of solutions (a)	Swarm size ($\|S\|$)	% Accuracy		
			NichePSO (%)	SN (%)	DC (%)
1	5	20	100.00	100.00	90.00
2	25	100	100.00	68.00	66.60
3	625	2500	94.75	41.00	56.6

Table 10
Scalability performance on the Rastrigin function

Dimensions (n)	Number of solutions (a)	Swarm size ($\|S\|$)	% Accuracy		
			NichePSO (%)	SN (%)	DC (%)
1	3	9	100.00	100.00	100.00
2	9	36	100.00	78.00	67.00
3	27	108	97.45	66.70	61.10
4	81	324	97.08	58.00	54.20
5	243	972	92.00	*	*

- Using the exponential relationship $\|S\| = a^2$ suggested in Section 5.3.4, 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 9 and 10, the relationship between the number of solutions, a, and the swarm size, $|S|$, was kept at
$$|S| = 4a. \tag{20}$$
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 number of solutions. Consequently, the relationship between swarm size and number of solutions would more likely be expressed as
$$|S| = c \cdot a^q, \tag{21}$$
where c is a constant, and $1 \leqslant q \leqslant 2$. Further experimentation would be required to empirically estimate ideal values for c and q.

6. Conclusions and future work

Population-based search methods 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 are not susceptible 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. When a possible solution is detected at a particle's location, a subswarm is created. The subswarm then optimizes the potential solution. This pseudo-memetic approach is called NichePSO. Experimental results obtained on a set of multi-modal functions showed that NichePSO successfully located and maintained multiple optimal solutions. Several parameter issues, related to NichePSO, were addressed. Suggestions were made as to potential values for tunable parameters.

Future work on NichePSO will include:

- *Parameter independence:* The current NichePSO implementation fails to correctly locate all solutions to a multi-modal function if μ is greater than the inter-swarm distance. This 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 [28].
- *Swarm sizes:* In Section 5.3.4 it was found that a^2, where a is the number of optima in a multi-modal function, was an acceptable estimate as to the number of particles required to locate all solutions. The accuracy of this estimate warrants further investigation, specifically when applying NichePSO to multi-modal functions of higher dimensions. Alternative velocity vector initialization techniques could also be investigated to see whether the number of particles per solutions can be reduced.
- *Ensemble neural networks:* Ensemble architectures train a number of neural networks, either sequentially or in parallel on the same problem. Since the search space associated with a neural network may be highly multi-modal, the use of a niching technique may be beneficial. It seems a natural step to exploit the nature of niching algorithms and applying it to ensemble learning, given that the PSO algorithm has been shown to be an effective optimization technique for neural networks.
- *Multi-objective optimization:* Future studies will investigate the applicability of the NichePSO to multi-objective optimization problems.

References

[1] T. Bäck, D.B. Foge, A. Michalewicz (Eds.), Handbook of Evolutionary Computation, IOP Publishers and Oxford University Press, 1997.

[2] D. Beasley, D.R. Bull, R.R. Martin, A sequential niching technique for multimodal function optimization, Evolutionary Computation 1 (2) (1993) 101–125.
[3] C.M. Bishop, Neural Networks for Pattern Recognition, Oxford University Press, 1995.
[4] R. Brits, Niching Strategies for Particle Swarm Optimization. Master's thesis, Department of Computer Science, University of Pretoria, Pretoria, South Africa, November 2002.
[5] R. Brits, A.P. Engelbrecht, F. van den Bergh, Solving systems of unconstrained equations using particle swarm optimizers, in: Proceedings of the IEEE Conference on Systems, Man and Cybernetics, October 2002, pp. 102 – 107.
[6] M. Clerc, J. Kennedy, The particle swarm – explosion, stability and convergence in a multidimensional complex space, IEEE Transactions on Evolutionary Computation 6 (1) (2002) 58–73, February.
[7] 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, Washington, DC, July 1999, pp. 3–13.
[8] C.A. Coello Coello, M.S. Lechuga, MOPSO: a proposal for multiple objective particle swarm optimization, in: Proceedings of the IEEE World Congress on Evolutionary Computation, Honolulu, Hawaii, May 2002, pp. 1051–1056.
[9] C.A. Coello Coello, D.A. van Veldhuizen, G.B. Lamont, Evolutionary Algorithms for Solving Multi-objective Problems, Kluwer Academic Publishers, 2003.
[10] K.A. de Jong, An Analysis of the Behavior of a Class of Genetic Adaptive Systems. Ph.D. thesis, Department of Computer Science, University of Michigan, Ann Arbor, Michigan, USA, 1975.
[11] K. Deb, Genetic Algorithms in Multimodal Function Optimization. Master's thesis, Department of Engineering Mathematics, University of Alabama, 1989.
[12] M. Dorigo, T. Stützle, Ant Colony Optimization, MIT Press, 2004.
[13] A.P. Engelbrecht, Fundamentals of Computational Swarm Intelligence, John Wiley & Sons, 2005.
[14] A.P. Engelbrecht, B.S. Masiye, G. Pampara, Niching ability of basic particle swarm optimization algorithms, in: Proceedings of the IEEE Swarm Intelligence Symposium, 2005.
[15] J.E. Fieldsend, S. Singh, A multi-objective algorithm based upon particle swarm optimisation, and efficient data structure and turbulence, in: The 2002 UK Workshop on Computational Intelligence, 2002, pp. 33–44.
[16] E. Fiesler, R. Beale (Eds.), Handbook of Neural Computation, IOP Publishers and Oxford University Press, 1996.
[17] D.E. Goldberg, J. Richardson, Genetic algorithm with sharing for multimodal function optimization, in: Proceedings of the Second International Conference on Genetic Algorithms, 1987, pp. 41–49.
[18] D.E. Goldberg, L. Wang, Adaptive Niching via Coevolutionary Sharing, technical report, Genetic Algorithm Lab, Urbana, University of Illinois, Illinois, August 1997. IlliGAL Rep. 97007.
[19] J. Horn, The nature of niching: Genetic algorithms and the evolution of optimal, cooperative populations. Ph.D. thesis, Urbana, University of Illinois, Illinois, Genetic Algorithm Lab, 1997.
[20] X. Hu, R. Eberhart, Multiobjective optimization using dynamic neighborhood particle swarm optimization, in: Proceedings of the IEEE World Congress on Evolutionary Computation, Honolulu, Hawaii, 12–17 May 2002, pp. 1677–1681.
[21] J. Kennedy, Small worlds and mega-minds: effects of neighborhood topology on particle swarm performance, in: Proceedings of the IEEE Congress on Evolutionary Computation, July 1999, pp. 1931–1938.
[22] J. Kennedy, R.C. Eberhart, Particle swarm optimization, in: Proceedings of the IEEE International Conference on Neural Networks, vol. IV, Perth, Australia, 1995, pp. 1942–1948.
[23] J. Kennedy, R.C. Eberhart, Swarm Intelligence, Morgan Kaufman, 2001.
[24] J. Kennedy, R. Mendes, Population structure and particle swarm performance, in: Proceedings of the IEEE World Congress on Evolutionary Computation, Honolulu, Hawaii, May 2002, pp. 1671–1676.
[25] M. Løvbjerg, T.K. Rasmussen, T. Krink, Hybrid particle swarm optimizer with breeding and subpopulations, in: Proceedings of the Genetic and Evolutionary Computation Conference, vol. 1, San Fransisco, USA, July 2001, pp. 469–476.
[26] S.W. Mahfoud, A comparison of parallel and sequential niching methods, in: Proceedings of the Sixth International Conference on Genetic Algorithms, 1995, pp. 136–143.
[27] S.W. Mahfoud, Niching Methods for Genetic Algorithms. Ph.D. thesis, Genetic Algorithm Lab, University of Illinois, Illinois, 1995. IlliGAL Rep. 95001.
[28] O.J. Mengshoel, D.E. Goldberg, Probabilistic crowding: deterministic crowding with probabilistic replacement, in: Proceedings of the Genetic and Evolutionary Computation Conference 1999, San Fransisco, USA, Morgan Kaufmann, 1999, pp. 409–416.
[29] B.L. Miller, 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.
[30] K.E. Parsopoulos, V.P. Plagianakos, G.D. Magoulas, M.N. Vrahatis, Stretching technique for obtaining global minimizers through particle swarm optimization, in: Proceedings of the Particle Swarm Optimization Workshop, Indianapolis, USA, 2001, pp. 22–29.
[31] K.E. Parsopoulos, M.N. Vrahatis, Modification of the particle swarm optimizer for locating all the global minima, in: V. Kurkova, N.C. Steele, R. Neruda, M. Karny (Eds.), Artificial Neural Networks and Genetic Algorithms, Springer, 2001, pp. 324–327.
[32] K.E. Parsopoulos, M.N. Vrahatis, Particle swarm optimization method in multiobjective problems, in: Proceedings of the 2002 ACM Symposium on Applied Computing (SAC 2002), 2002, pp. 603–607.
[33] W.H. Press, S.A. Teukolsky, W.T. Vetterling, B.P. Flannery, Numerical Recipes in C: The Art of Scientific Computing, second ed., Cambridge University Press, 1992.
[34] Y. Shi, R.C. Eberhart, A modified particle swarm optimizer, in: Proceedings of the IEEE World Conference on Computational Intelligence, Anchorage, Alaska, May 1998, pp. 69–73.

[35] Y. Shi, R.C. Eberhart, An empirical study of particle swarm optimization, in: Proceedings of the IEEE Congress on Evolutionary Computation, Piscataway, NJ, 1999, pp. 1945–1960.
[36] P.N. Suganthan, Particle swarm optimizer with neighborhood operator, in: Proceedings of the IEEE Congress on Evolutionary Computation, July 1999, pp. 1958–1961.
[37] E. Thiémard, Economic Generation of Low-Discrepancy Sequences with a b-ary Gray Code, Department of Mathematics, Ecole Polytechnique Fédérale de Lausanne, Lausanne, Switzerland.
[38] F. van den Bergh, *An Analysis of Particle Swarm Optimizers*. Ph.D. thesis, Department of Computer Science, University of Pretoria, Pretoria, South Africa, 2002.
[39] F. van den Bergh, A.P. Engelbrecht, Cooperative learning in neural networks using particle swarm optimizers, South African Computer Journal 26 (2000) 84–90, November.
[40] F. van den Bergh, A.P. Engelbrecht, A new locally convergent particle swarm optimizer, in: Proceedings of the IEEE Conference on Systems, Man and Cybernetics, October 2002, pp. 96–101.
[41] F. van den Bergh, A.P. Engelbrecht, A study of particle swarm optimization particle trajectories, Information Science 176 (8) (2006) 937–971.