Chapter 2

Background and Literature Review

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

2.1 Introduction

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

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

This chapter presents an overview of the theory behind optimization. Section 2.2 defines optimization problems and establishes a common notation used throughout this thesis. Evolutionary computation is discussed in section 2.3, with an overview of genetic algorithms in section 2.4. Swarm intelligence is discussed in section 2.5, focusing in particular on the particle swarm optimization algorithm. Principle differences between evolutionary computing and swarm intelligence are touched on in section 2.6. Various extensions and enhancements are presented in sections 2.7 and 2.8 respectively. Section 2.9 discusses a number of PSO application areas.

2.2 Function Optimization

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

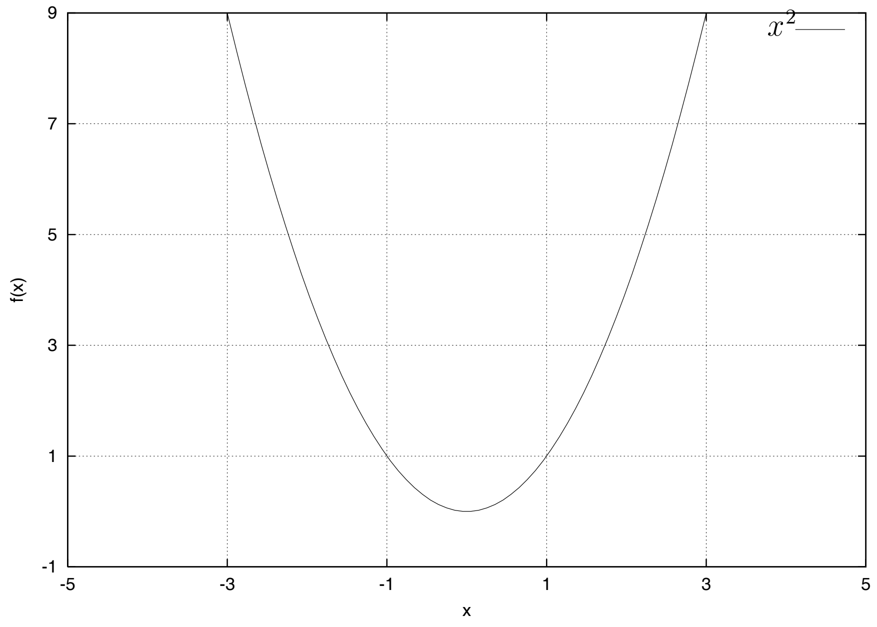

Figure 2.1: $f(x) = x^2$, a unimodal function.

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

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

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

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

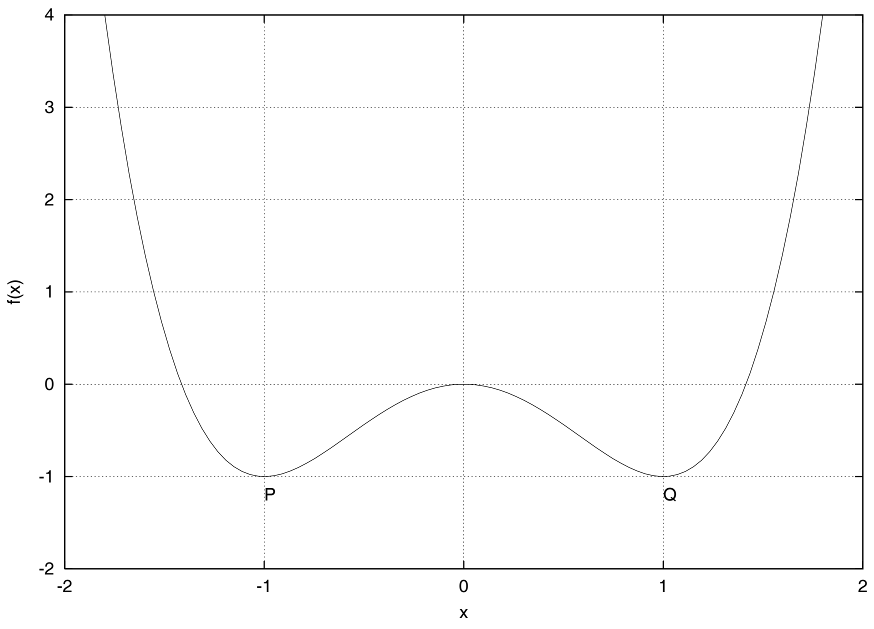

Figure 2.2: Function $f(x) = x^4 - 2x^2$ with two equal minima, P and Q.

2.3 Evolutionary Computing

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

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

Evolutionary computing simulates the evolution process by representing candidate solutions as a population of individuals within an environment and evolving them over time [3]. To facilitate the evolutionary process, individuals are evaluated at regular intervals, or *generations*, using a *fitness function.* A fitness function evaluates the quality of a potential solution represented by an individual. Depending on the type of evolutionary algorithm, a population of individuals is adapted using evolutionary operators and strategies in an attempt to find an optimal solution. Evolutionary operators include mutation, reproduction, selection and competition between individuals. The adapted individuals represent a next generation. The process of creating a next generation can be repeated until an acceptable solution is found, or until a maximum number of generational steps have been taken.

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

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

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

2.4 Genetic Algorithms

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

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

Table 2.1: GA symbols

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

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

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

Individuals may be initialized randomly or by using some technique relevant to the optimization problem. A population is considered to have converged either after a maximum number of new populations have been created (when $g \geq g_{max}$), or when $f(C_{g,b}) < f_\phi$, where $\exists\ b \in C_g | f(C_{g,b}) \leq f(C_{g,i}),\ \forall\ i \in C_g$. The threshold f_ϕ represents a value that can be used to determine when an individual represents an acceptable solution. The representation of individuals, fitness functions and the GA evolutionary operators are now discussed in more detail.

2.4.1 Representation of Individuals

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

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

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

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

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

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

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

2.4.2 Fitness Functions

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

2.4.3 Evolutionary Operators

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

Crossover

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

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

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

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

Mutation

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

Selection Operators

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

2.5 Particle Swarm Optimization

The particle swarm optimization (PSO) algorithm, originally introduced by Kennedy and Eberhart [49], is modeled after the social behavior of birds in a flock. PSO is a population based search process where individuals, referred to as particles, are grouped into a swarm. Each particle in the swarm represents a candidate solution to an optimization problem. In a PSO system, each particle is "flown" through the multidimensional search space, adjusting its position in search space according to its own experience and that of neighboring particles. A particle therefore makes use of the best position encountered by itself and that of its neighbors to position itself toward an optimal solution. The effect is that particles "fly" toward a minimum, while still searching a wide area around the best known solution. The performance of each particle (i.e. the "closeness" of a particle to the global optimum) is measured using a predefined fitness function which encapsulates the characteristics of the optimization problem.

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

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

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

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

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

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

Figure 2.3: The general PSO algorithm

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

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

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

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

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

$$\begin{aligned} N_j = \{\mathbf{y}_{i-l}(t), \mathbf{y}_{i-l+1}(t), \ldots, \mathbf{y}_{i-1}(t), \mathbf{y}_i(t), & \quad (2.6) \\ \mathbf{y}_{i+1}(t), \ldots, \mathbf{y}_{i+l-1}(t), \mathbf{y}_{i+l}(t)\} & \quad (2.7) \\ \hat{\mathbf{y}}_j(t+1) \in N_j \mid f(\hat{\mathbf{y}}_j(t+1)) = \min\{f(\mathbf{y}_i)\}, \forall \mathbf{y}_i \in N_j & \quad (2.8) \end{aligned}$$

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

$$\begin{aligned} v_{i,j}(t+1) &= v_{i,j}(t) + c_1 r_{1,j}(t)(y_{i,j}(t) - x_{i,j}(t)) + \\ & \quad c_2 r_{2,j}(t)(\hat{y}_j(t) - x_{i,j}(t)) & (2.9) \\ \mathbf{x}_i(t+1) &= \mathbf{x}_i(t) + \mathbf{v}_i(t+1) & (2.10) \end{aligned}$$

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

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

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

2.6 Evolutionary Computing vs. Swarm Intelligence

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

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

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

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

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

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

2.7 Extensions to the PSO

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

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

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

2.7.1 Convergence Acceleration Techniques

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

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

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

Inertia Weight

The use of the inertia weight w was introduced by Shi and Eberhart [80] and was not present in the original paper by Kennedy and Eberhart [49]. The inertia weight controls the influence of a particle's previous velocity, resulting in a memory effect. The velocity update equation (equation (2.9)) is redefined to be

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

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

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

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

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

Constriction Factor

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

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

where

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

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

Evolutionary Computing Generational Operators

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

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

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

and

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

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

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

and

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

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

Objective Function Stretching

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

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

and

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

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

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

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

Division of Labour

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

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

Task 2: The *local search* task.

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

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

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

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

The parameter ℓ controls the steepness of the response function T (the term ζ_i^ℓ thus represents the stimulus ζ_i raised to the power ℓ). The stimulus ζ_i represents the number of iterations since the last *improvement* to $f(\mathbf{x}_i)$. Therefore, when i's fitness *does not* improve over time, ζ_i will increase. When i's fitness *does* improve over time, ζ_i will remain zero and $X_i = 0$. Vesterstrøm *et al* reported improved convergence when comparing DoLPSO to GAs, the traditional PSO and simulated annealing [91].

Multi-Phase Generalization

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

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

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

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

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

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

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

Particles in phase 1, group 1 move towards $\hat{\mathbf{y}}$, while particles in group 2 move away from $\hat{\mathbf{y}}$ towards $\mathbf{x}$. In phase 2, the roles are reversed. (It is not clear why the explicit specification of c_v was necessary, as it does not affect the optimization process.) The number of phases and groups is a user tunable parameter and must be known before the algorithm starts. A group of particles that are in the same phase all have identical goals. Changing phase is controlled by a *phase change frequency* (PCF). PCF indicates a number of evolutionary iterations of the PSO algorithm. Al-Kazemi and Mohan also proposed that a PCF free version of the algorithm would initiate a phase change when a pre-determined number of iterations yielded no improvement. The algorithm randomly re-initializes particle velocities to avoid converging on sub-optimal solutions after a pre-determined number of iterations, called the *velocity change variable* (VC). In quite a departure from the original PSO, the multi-phase generalization uses a hill-climbing approach that only updates particle positions if a calculated new position presents an improvement. A particle will therefore always occupy an optimal position relative to its own history. This explains why a personal best position is not taken into consideration in equation (2.14).

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

The Guaranteed Convergence Particle Swarm Optimizer

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

2.7.2 Swarm Diversity Enhancements

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

Subpopulations

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

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

Particle Neighborhoods

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

- maintain several concurrent particle swarms, each with its own global solution;

- may explicitly allow information exchange between subpopulations.

Particle neighborhoods are more dynamic. Neighborhoods

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

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

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

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

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

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

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

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

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

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

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

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

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

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

Self-Organized Criticality

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

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

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

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

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

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

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

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

PSO with Spatial Extension

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

1. Particles are bounced away from the collision in a random direction, retaining their original speed.
2. Particle interaction mimic realistic, physical collisions.
3. 'Velocity line bouncing', that maintains the direction of the velocity vector, but scales the speed of the particle.

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

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

2.8 Modifications to the PSO

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

2.8.1 The Binary PSO

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

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

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

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

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

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

2.8.2 Cooperative Swarms

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

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

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

2.8.3 Dynamic Systems

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

Dynamic Goals

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

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

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

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

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

The suggested changes showed to be a definite improvement.

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

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

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

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

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

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

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

- No action may be taken.

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

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

Noisy Functions

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

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

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

2.9 Applications of the PSO

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

2.9.1 Training of Neural Networks

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

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

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

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

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

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

the parameter λ was learned with the network weights.

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

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

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

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

Conradie *et al* recently presented the *Adaptive Neural Swarming* (ANS) approach [16]. ANS was developed to alleviate suboptimal performance of existing neurocontrollers in dynamic process environments. Conradie *et al* demonstrated that existing *linear* process controllers lack the ability to maintain an optimal *operating point*, due to the nonlinear nature of process environments. An operating point indicates a combination of parameter values, such as temperature and pressure, that may offer an economically acceptable performance level. Neural networks can be trained to mimic the function of neurocontrollers to maintain an operating point [18]. The ANS technique uses PSO as a local optimization algorithm training a reinforcement neurocontroller architecture. Reinforcement neural learning uses information from a learned problem space to gauge how effective a learning process is. In the ANS PSO algorithm, each particle represents a candidate neurocontroller, which is a slightly altered replica of an existing controller. Conradie *et al* found that the utilization of PSO as optimization techniques yield much improved performance, to the extent that ANS offers a viable neurocontroller alternative [16].

2.9.2 Multi–Objective Optimization

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

Formulation of MOO problems

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

- Satisfy m inequality constraints:

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

- Satisfy p equality constraints:

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

- Optimize a k-dimensional vector function:

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

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

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

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

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

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

MOPSO

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

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

where $\mathbf{H}_{h,j}$ represents the j^{th} dimension of a randomly selected particle in hypercube $\mathbf{H}_h$. The position repository maintains a set of favorable positions within the search space. The position repository is re-evaluated after each iteration of the algorithm. Dominated solutions in the repository may be replaced by better, *non-dominated* solutions. A limit is placed on the size of the position repository, so a heuristic is put in place to determine which particle position will have preference over others. The technique is simple: A position that exists in an area of the search space that is less populated will be given preference over a solution occupying a position in a more densely populated region. This scheme has the added advantage of promoting swarm diversity. MOPSO performed similar to existing GA-based MOO techniques [15].

Weighted Aggregation and Populations

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

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

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

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

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

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

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

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

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

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

This update forces movement on the Pareto front.

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

Dynamic Neighborhoods

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

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

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

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

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

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

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

2.10 Conclusion

In this section, several different evolutionary computing and swarm intelligence techniques were discussed. Each of these techniques were originally based on the observed behavior of some element of nature. Evolutionary computing is based on the process of evolution, and attempts to mimic it. Swarm intelligence techniques exploit the emergent collective intelligence of swarms of seemingly unintelligent particles, such as is present in the behavior of a flock of birds flying in formation.

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

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

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