Chapter 2

Optimization and Optimization Methods

This chapter provides a brief overview of optimization. This is followed by a brief discussion of traditional and stochastic optimization methods. Evolutionary algorithms (with more emphasis on genetic algorithms) are then presented. This is followed by an elaborated discussion of particle swarm optimization and its various modifications. A brief overview of ant colony systems is then given.

2.1 Optimization

The objective of optimization is to seek values for a set of parameters that maximize or minimize objective functions subject to certain constraints [Rardin 1998; Van den Bergh 2002]. A choice of values for the set of parameters that satisfy all constraints is called a *feasible solution*. Feasible solutions with objective function value(s) as good as the values of any other feasible solutions are called *optimal solutions* [Rardin 1998]. An example of an optimization problem is the arrangement of the transistors in a computer chip in such a way that the resulting layout occupies the smallest area and that as few as possible components are used. Optimization techniques are used on a daily base for industrial planning, resource allocation, scheduling, decision making, etc. Furthermore, optimization techniques are widely used in many fields such as business, industry, engineering and computer science. Research in the optimization

field is very active and new optimization methods are being developed regularly [Chinneck 2000].

Optimization encompasses both maximization and minimization problems. Any maximization problem can be converted into a minimization problem by taking the negative of the objective function, and *vice versa*. Hence, the terms optimization, maximization and minimization are used interchangeably in this thesis. In general, the problems tackled in this thesis are minimization problems. Therefore, the remainder of the discussion focuses on minimization problems.

The minimization problem can be defined as follows [Pardalos *et al*. 2002]

Given $f : \boldsymbol{S} \to \Re$ where $\boldsymbol{S} \subseteq \Re^{N_d}$ and N_d is the dimension of the search space $\boldsymbol{S}$

find $\boldsymbol{x}^* \in \boldsymbol{S}$ such that $f(\boldsymbol{x}^*) \leq f(\boldsymbol{x}), \forall \boldsymbol{x} \in \boldsymbol{S}$ (2.1)

The variable $\boldsymbol{x}^*$ is called the *global minimizer* (or simply the *minimizer*) of f and $f(\boldsymbol{x}^*)$ is called the *global minimum* (or simply the *minimum*) value of f. This can be illustrated as given in Figure 2.1 where $\boldsymbol{x}^*$ is a global minimizer of f. The process of finding the global optimal solution is known as *global optimization* [Gray *et al*. 1997]. A true global optimization algorithm will find $\boldsymbol{x}^*$ regardless of the selected starting point $\boldsymbol{x}_0 \in \boldsymbol{S}$ [Van den Bergh 2002]. Global optimization problems are generally very difficult and are categorized under the class of nonlinear programming (NLP) [Gray *et al*. 1997].

Examples of global optimization problems are [Gray *et al*. 1997]:

- Combinatorial problems: where a linear or nonlinear function is defined over a finite but very large set of solutions, for example, network problems and scheduling [Pardalos *et al*. 2002]. The problems addressed in this thesis belong to this category.
- General unconstrained problems: where a nonlinear function is defined over an unconstrained set of real values.
- General constrained problems: where a nonlinear function is defined over a constrained set of real values.

Evolutionary algorithms (discussed in Sections 2.4-2.5) have been successfully applied to the above problems to find approximate solutions [Gray *et al*. 1997]. More details about global optimization can be found in Pardalos *et al*. [2002], Floudas and Pardalos [1992] and Horst *et al*. [2000].

In Figure 2.1, $\boldsymbol{x}_{\boldsymbol{B}}^{*}$ is called the *local minimizer* of region $\boldsymbol{B}$ because $f(\boldsymbol{x}_{\boldsymbol{B}}^{*})$ is the smallest value within a local neighborhood, $\boldsymbol{B}$. Mathematically speaking, the variable $\boldsymbol{x}_{\boldsymbol{B}}^{*}$ is a local minimizer of the region $\boldsymbol{B}$ if

$$f(\boldsymbol{x}_{\boldsymbol{B}}^{*}) \leq f(\boldsymbol{x}), \forall \boldsymbol{x} \in \boldsymbol{B} \tag{2.2}$$

where $\boldsymbol{B} \subset \boldsymbol{S}$. Every global minimizer is a local minimizer, but a local minimizer is not necessarily a global minimizer.

Generally, a local optimization method is guaranteed to find the local minimizer $\boldsymbol{x}_{\boldsymbol{B}}^{*}$ of the region $\boldsymbol{B}$ if a starting point $\boldsymbol{x}_0$ is used with $\boldsymbol{x}_0 \in \boldsymbol{B}$. An optimization algorithm that converges to a local minimizer, regardless of the selected starting point $\boldsymbol{x}_0 \in \boldsymbol{S}$, is called a *globally convergent* algorithm [Van den Bergh

2002]. There are many local optimization algorithms in the literature. For more detail the reader is referred to Aarts and Lenstra [2003] and Korte and Vygen [2002].

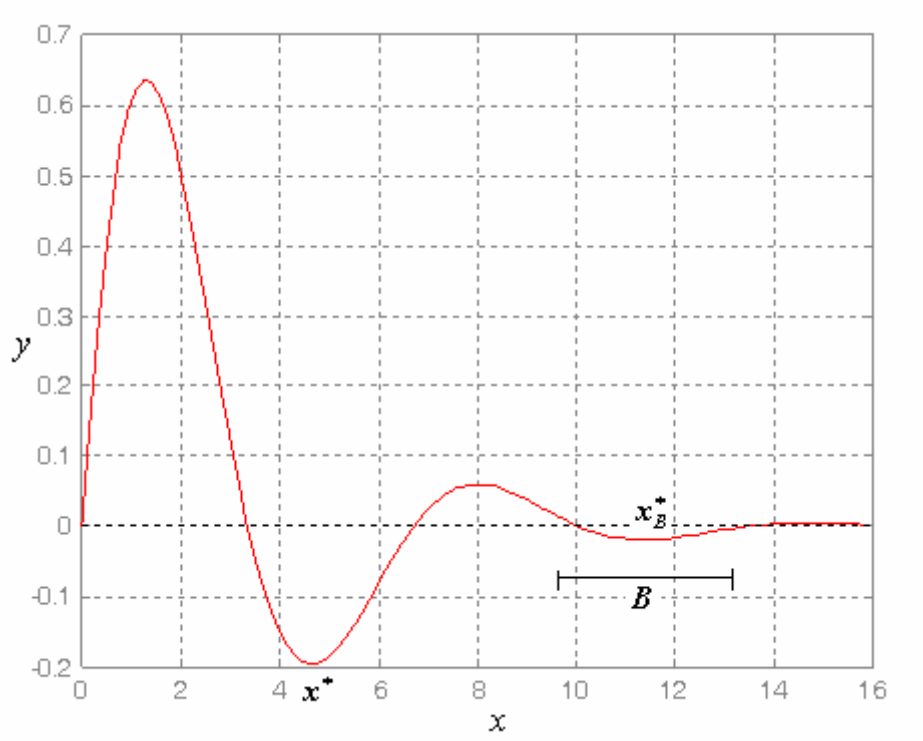


Figure 2.1: Example of a global minimizer x^* as well as a local minimizer x_B^*

2.2 Traditional Optimization Algorithms

Traditional optimization algorithms use exact methods to find the best solution. The idea is that if a problem can be solved, then the algorithm should find the global best solution. One exact method is the *brute force* (or *exhaustive*) search method where the algorithm tries every solution in the search space so that the global optimal solution is guaranteed to be found. Obviously, as the search space increases the cost of brute force algorithms increases. Therefore, brute force algorithms are not appropriate for the class of problems known as NP-hard problems. The time to exhaustively search an

NP-hard problem increases exponentially with problem size. Other exact methods include linear programming, divide and conquer and dynamic programming. More details about exact methods can be found in Michalewicz and Fogel [2000].

2.3 Stochastic Algorithms

Stochastic search algorithms are used to find near-optimal solutions for NP-hard problems in polynomial time. This is achieved by assuming that good solutions are close to each other in the search space. This assumption is valid for most real world problems [Løvberg 2002; Spall 2003]. Since the objective of a stochastic algorithm is to find a near-optimal solution, stochastic algorithms may fail to find a global optimal solution. While an exact algorithm generates a solution only after the run is completed, a stochastic algorithm can be stopped any time during the run and generate the best solution found so far [Løvberg 2002].

Stochastic search algorithms have several advantages compared to other algorithms [Venter and Sobieszczanski-Sobieski 2002]:

- Stochastic search algorithms are generally easy to implement.
- They can be used efficiently in a multiprocessor environment.
- They do not require the problem definition function to be continuous.
- They generally can find optimal or near-optimal solutions.
- They are suitable for discrete and combinatorial problems.

Three major stochastic algorithms are Hill-Climbing [Michalewicz and Fogel 2000], Simulated Annealing [Van Laarhoven and Aarts 1987] and Tabu search [Glover 1989;

Glover 1990]. In Hill-Climbing, a potential solution is randomly chosen. The algorithm then searches the neighborhood of the current solution for a better solution. If a better solution is found, then it is set as the new potential solution. This process is repeated until no more improvement can be made. Simulated annealing is similar to Hill-Climbing in the sense that a potential solution is randomly chosen. A small value is then added to the current solution to generate a new solution. If the new solution is better than the original one then the solution moves to the new location. Otherwise, the solution will move to the new location with a probability that decreases as the run progresses [Salman 1999]. Tabu search is a heuristic search algorithm where a tabu list memory of previously visited solutions is maintained in order to improve the performance of the search process. The tabu list is used to "guide the movement from one solution to the next one to avoid cycling" [Gabarro 2000], thus, avoid being trapped in a local optimum. Tabu search starts with a randomly chosen current solution. A set of test solutions are generated via moves from the current solution. The best test solution is set as the current solution if it is not in the tabu list, or if it is in the tabu list, but satisfies an aspiration criterion. A test solution satisfies an aspiration criterion if it is in the tabu list and it is the best solution found so far [Chu and Roddick 2003]. This process is repeated until a stopping criterion is satisfied.

2.4 Evolutionary Algorithms

Evolutionary algorithms (EAs) are general-purpose stochastic search methods simulating natural selection and evolution in the biological world. EAs differ from other optimization methods, such as Hill-Climbing and Simulated Annealing, in the

fact that EAs maintain a population of potential (or candidate) solutions to a problem, and not just one solution [Engelbrecht 2002; Salman 1999].

Generally, all EAs work as follows: a population of individuals is initialized where each individual represents a potential solution to the problem at hand. The quality of each solution is evaluated using a *fitness function*. A selection process is applied during each iteration of an EA in order to form a new population. The selection process is biased toward the fitter individuals to ensure that they will be part of the new population. Individuals are altered using unary transformation (mutation) and higher order transformation (crossover). This procedure is repeated until convergence is reached. The best solution found is expected to be a *near-optimum* solution [Michalewicz 1996]. A general pseudo-code for an EA is shown in Figure 2.2 [Gray *et al*. 1997].

```
Initialize the population
Evaluate the fitness of each individual in the population
Repeat
        Apply selection on the population to form a new population
        Alter the individuals in the population using evolutionary operators
        Evaluate the fitness of each individual in the population
Until some convergence criteria are satisfied
```

Figure 2.2: General pseudo-code for EAs

The unary and higher order transformations are called *evolutionary operators*. The two most frequently evolutionary operators are:

- *Mutation*, which modifies an individual by a small random change to generate a new individual [Michalewicz 1996]. This change can be done by inverting the value of a binary digit in the case of binary representations, or by adding

(or subtracting) a small number to (or from) selected values in the case of floating point representations. The main objective of mutation is to add some diversity by introducing more genetic material into the population in order to avoid being trapped in a local optimum. Generally, mutation is applied using a low probability. However, some problems (e.g. problems using floating point representations) require using mutation with high probability [Salman 1999]. A preferred strategy is to start with high probability of mutation and dreasing it over time.

- *Recombination (or Crossover)*, where parts from two (or more) individuals are combined together to generate new individuals [Michalewicz 1996]. The main objective of crossover is to explore new areas in the search space [Salman 1999].

There are four major evolutionary techniques:

- *Genetic Programming* (GP) [Koza 1992] which is used to search for the fittest program to solve a specific problem. Individuals are represented as trees and the focus is on genotypic evaluation.
- *Evolutionary Programming* (EP) [Fogel 1994] which is generally used to optimize real-valued continuous functions. EP uses selection and mutation operators; it does not use the recombination operator. The focus is on phenotypic evaluation and not on genotypic evaluation.
- *Evolutionary Strategies* (ES) [Bäck *et al*. 1991] which is used to optimize real-valued continuous functions. ES uses selection, crossover and mutation operators. ES optimizes both the population and the optimization process, by evolving strategy parameters. Hence, ES is evolution of evolution.

- *Genetic Algorithms* (GA) [Goldberg 1989] which is generally used to optimize general combinatorial problems [Gray *et al*. 1997]. The GA is a commonly used algorithm and has been used for comparison purposes in this thesis. The focus in GA is on genetic evolution using both mutation and crossover, although the original GAs developed by Holland [1962] used only crossover. Since later chapters make use of GAs, a detailed explanation of GAs is given in Section 2.5.

Due to its population-based nature, EAs can avoid being trapped in a local optimum and consequently can often find global optimal solutions. Thus, EAs can be viewed as global optimization algorithms. However, it should be noted that EAs may fail to converge to a global optimum [Gray *et al*. 1997].

EAs have successfully been applied to a wide variety of optimization problems, for example: image processing, pattern recognition, scheduling, engineering design, etc. [Gray *et al* 1997; Goldberg 1989].

2.5 Genetic Algorithms

Genetic Algorithms (GAs) are evolutionary algorithms that use selection, crossover and mutation operators. GAs were first proposed by Holland [1962; 1975] and were inspired by Darwinian evolution and Mendelian genetics [Salman 1999]. GAs follow the same algorithm presented in Figure 2.2. GAs are one of the most popular evolutionary algorithms and have been widely used to solve difficult optimization problems. GAs have been successfully applied in many areas such as pattern recognition, image processing, machine learning, etc. [Goldberg 1989]. In many cases GAs perform better than EP and ESs. However, EP and ESs usually converge better

than GAs for real valued function optimization [Weiss 2003]. Individuals in GAs are called *chromosomes*. Each chromosome consists of a string of cells called *genes*. The value of each gene is called *allele*. The major parameters of GAs are discussed in Sections 2.5.1-2.5.5. In Section 2.5.6, a brief discussion about a problem that may be encountered in GAs is discussed.

2.5.1 Solution Representation

Binary representation is often used in GAs where each gene has a value of either 0 or 1. Other presentations have been proposed, for example, floating point representations [Janikow and Michalewicz 1991], integer representations [Bramlette 1991], gray-coded representations [Whitley and Rana 1998] and matrix representation [Michalewicz 1996]. More detail about representation schemes can be found in Goldberg [1989]. Generally, non-binary representations require different evolutionary operators for each representation while uniform operators can be used with binary representation for any problem [Van den Bergh 2002]. However, according to Michalewicz [1991], floating point representations are faster, more consistent and have higher precision than binary representations.

2.5.2 Fitness Function

A key element in GAs is the selection of a fitness function that accurately quantifies the quality of candidate solutions. A good fitness function enables the chromosomes to effectively solve a specific problem. Both the fitness function and solution representation are problem dependent parameters. A poor selection of these two parameters will drastically affect the performance of GAs. One problem related to

fitness functions that may occur when GAs are used to optimize combinatorial problems is the existence of points in the search space that do not map to feasible solutions. One solution to this problem is the addition of a *penalty function* term to the original fitness function so that chromosomes representing infeasible solutions will have a low fitness score, and as such, will disappear from the population [Fletcher 2000].

2.5.3 Selection

Another key element of GAs is the selection operator which is used to select chromosomes (called *parents*) for mating in order to generate new chromosomes (called *offspring*). In addition, the selection operator can be used to select elitist individuals. The selection process is usually biased toward fitter chromosomes. Selection methods are used as mechanisms to focus the search on apparently more profitable regions in the search space [Angeline, Using Selection 1998]. Examples of well-known selection approaches are:

- *Roulette wheel selection:* Parent chromosomes are probabilistically selected based on their fitness. The fitter the chromosome, the higher the probability that it may be chosen for mating. Consider a roulette wheel where each chromosome in the population occupies a slot with slot size proportional to the chromosome's fitness [Gray *et al*. 1997]. When the wheel is randomly spun, the chromosome corresponding to the slot where the wheel stopped is selected as the first parent. This process is repeated to find the second parent. Clearly, since fitter chromosomes have larger slots, they have better chance to be chosen in the selection process [Goldberg 1989].

- *Rank selection:* Roulette wheel selection suffers from the problem that highly fit individuals may dominate in the selection process. When one or a few chromosomes have a very high fitness compared to the fitness of other chromosomes, the lower fit chromosomes will have a very slim chance to be selected for mating. This will increase selection pressure, which will cause diversity to decrease rapidly resulting in premature convergence. To reduce this problem, rank selection sorts the chromosomes according to their fitness and base selection on the rank order of the chromosomes, and not on the absolute fitness values. The worst (i.e. least fit) chromosome has rank of 1, the second worst chromosome has rank of 2, and so on. Rank selection still prefers the best chromosomes; however, there is no domination as in the case of roulette wheel selection. Hence, using this approach all chromosomes will have a good chance to be selected. However, this approach may have a slower convergence rate than the roulette wheel approach [Gray *et al*. 1997].
- *Tournament selection:* In this more commonly used approach [Goldberg 1989], a set of chromosomes are randomly chosen. The fittest chromosome from the set is then placed in a mating pool. This process is repeated until the mating pool contains a sufficient number of chromosomes to start the mating process.
- *Elitism:* In this approach, the fittest chromosome, or a user-specified number of best chromosomes, is copied into the new population. The remaining chromosomes are then chosen using any selection operator. Since the best solution is never lost, the performance of GA can significantly be improved [Gray *et al*. 1997].

2.5.4 Crossover

Crossover is "the main explorative operator in GAs" [Salman 1999]. Crossover occurs with a user-specified probability, called the *crossover probability* p_c. p_c is problem dependent with typical values in the range between 0.4 and 0.8 [Weiss 2003]. The four main crossover operators are:

- *Single point crossover:* In this approach, a position is randomly selected at which the parents are divided into two parts. The parts of the two parents are then swapped to generate two new offspring.

  Example 2.1

  Parent A: 11001**010**

  Parent B: **01110**011

  Offspring A: 11001011

  Offspring B: **01110010**

- *Two point crossover:* In this approach, two positions are randomly selected. The middle parts of the two parents are then swapped to generate two new offspring.

  Example 2.2

  Parent A: 11**0010**10

  Parent B: **01**1100**11**

  Offspring A: 11110010

  Offspring B: **01001011**

- *Uniform crossover:* In this approach, alleles are copied from either the first parent or the second parent with some probability, usually set to 0.5.

  Example 2.3

  Parent A: 1**10**010**10**

  Parent B: **0**11**100**11

  Offspring A: 11101011

  Offspring B: **01010010**

- *Arithmetic crossover:* In this approach, which is used for floating point representations, offspring is calculated as the arithmetic mean of the parents [Michalewicz 1996; Krink and Løvbjerg 2002], i.e.

$$\boldsymbol{x}_{\text{offspring A}} = r\,\boldsymbol{x}_{\text{parent A}} + (1-r)\boldsymbol{x}_{\text{parent B}} \tag{2.3}$$

$$\boldsymbol{x}_{\text{offspring B}} = r\,\boldsymbol{x}_{\text{parent B}} + (1-r)\boldsymbol{x}_{\text{parent A}} \tag{2.4}$$

  where $r \sim U(0,1)$.

2.5.5 Mutation

In GAs, mutation is considered to be a background operator, mainly used to explore new areas in the search space and to add diversity (contrary to selection and crossover which reduces diversity) to the population of chromosomes in order to prevent being trapped in a local optimum. Mutation is applied to the offspring chromosomes after crossover is performed. In a binary coded GA, mutation is done by inverting the value of each gene in the chromosome according to a user-specified probability, which is called the *mutation probability*, p_m. This probability is problem dependent. Mutation occurs infrequently both in nature and in GAs [Løvberg 2002], hence, a typical value

for p_m is 0.01 [Weiss 2003]. However, a better value for p_m is the inverse of the number of genes in a chromosome (i.e. chromosome size) [Goldberg 1989].

One mutation scheme used with floating point representations is the non-uniform mutation [Michalewicz 1996]. The j^{th} element of chromosome $\boldsymbol{x}$ is mutated as follows:

$x_j = x_j + \Delta x_j$, where

$$\Delta x_j = \begin{cases} +(Z_{\max} - x_j)(1 - r^{(1-t/t_{\max})^b}) & \text{if a random bit is 0} \\ -(x_j - Z_{\min})(1 - r^{(1-t/t_{\max})^b}) & \text{if a random bit is 1} \end{cases} \tag{2.5}$$

where $Z_{\min}$ and $Z_{\max}$ are the lower and upper bound of the search space, $r \sim U(0,1)$, t is the current iteration, $t_{\max}$ is the total number of iterations and b is a user-specified parameter determining the degree of iteration number dependency (in this thesis, b was set to 5 as suggested by Michalewicz [1996]). Thus, the amount of mutation decreases as the run progresses.

Kennedy and Spears [1998] observed that a GA using either mutation or crossover performed better than a GA using both crossover and mutation operators when applied to a set of random problems (especially for problems with a large multimodality).

2.5.6 The Premature Convergence Problem

Genetic algorithms suffer from the *premature suboptimal convergence* (simply *premature convergence* or *stagnation*) which occurs when some poor individuals attract the population - due to a local optimum or bad initialization - preventing further exploration of the search space [Dorigo *et al*. 1999]. One of the causes of this problem is that a very fit chromosome is generally sure to be selected for mating, and since offspring resemble their parents, chromosomes become too similar (i.e. population loses diversity). Hence, the population will often converge before reaching the global optimal solution, resulting in premature convergence. Premature convergence can be prevented by:

- Using subpopulations: The population of chromosomes is divided into separate subpopulations. Each subpopulation is evolved independent of the other subpopulations for a user-specified number of generations. Then, a number of chromosomes are exchanged between the subpopulations. This process helps in increasing diversity and thus preventing premature convergence.
- Re-initializing some chromosomes: A few chromosomes are re-initialized from time to time in order to add diversity to the population.
- Increase the mutation probability: As already discussed, mutation aids in exploring new areas in the search space and increases diversity. Therefore, increasing p_m will help in preventing premature convergence.

In general, any mechanism that can increase diversity will help in preventing premature convergence.

2.6 Particle Swarm Optimization

A particle swarm optimizer (PSO) is a population-based stochastic optimization algorithm modeled after the simulation of the social behavior of bird flocks [Kennedy and Eberhart 1995; Kennedy and Eberhart 2001]. PSO is similar to EAs in the sense that both approaches are population-based and each individual has a fitness function. Furthermore, the adjustments of the individuals in PSO are relatively similar to the arithmetic crossover operator used in EAs [Coello Coello and Lechuga 2002]. However, PSO is influenced by the simulation of social behavior rather than the survival of the fittest [Shi and Eberhart 2001]. Another major difference is that, in PSO, each individual benefits from its history whereas no such mechanism exists in EAs [Coello Coello and Lechuga 2002]. PSO is easy to implement and has been successfully applied to solve a wide range of optimization problems such as continuous nonlinear and discrete optimization problems [Kennedy and Eberhart 1995; Kennedy and Eberhart 2001; Eberhart and Shi, Comparison 1998].

2.6.1 The PSO Algorithm

In a PSO system, a swarm of individuals (called *particles*) fly through the search space. Each particle represents a candidate solution to the optimization problem. The position of a particle is influenced by the best position visited by itself (i.e. its own experience) and the position of the best particle in its neighborhood (i.e. the experience of neighboring particles). When the neighborhood of a particle is the entire

swarm, the best position in the neighborhood is referred to as the global best particle, and the resulting algorithm is referred to as a *gbest* PSO. When smaller neighborhoods are used, the algorithm is generally referred to as a *lbest* PSO [Shi and Eberhart, Parameter 1998]. The performance of each particle (i.e. how close the particle is from the global optimum) is measured using a fitness function that varies depending on the optimization problem.

Each particle in the swarm is represented by the following characteristics:

$\boldsymbol{x}_i$: The *current position* of the particle;

$\boldsymbol{v}_i$: The *current velocity* of the particle;

$\boldsymbol{y}_i$: The *personal best position* of the particle.

$\hat{\boldsymbol{y}}_i$: The *neighborhood best position* of the particle.

The personal best position of particle *i* is the best position (i.e. the one resulting in the best fitness value) visited by particle *i* so far. Let *f* denote the objective function. Then the personal best of a particle at time step *t* is updated as

$$\boldsymbol{y}_i(t+1) = \begin{cases} \boldsymbol{y}_i(t) & \text{if } f(\boldsymbol{x}_i(t+1)) \geq f(\boldsymbol{y}_i(t)) \\ \boldsymbol{x}_i(t+1) & \text{if } f(\boldsymbol{x}_i(t+1)) < f(\boldsymbol{y}_i(t)) \end{cases} \tag{2.6}$$

For the *gbest* model, the best particle is determined from the entire swarm by selecting the best personal best position. If the position of the global best particle is denoted by the vector $\hat{\boldsymbol{y}}$, then

$$\hat{\boldsymbol{y}}(t) \in \{\boldsymbol{y}_0, \boldsymbol{y}_1, \ldots, \boldsymbol{y}_s\} = \min\{f(\boldsymbol{y}_0(t)), f(\boldsymbol{y}_1(t)), \ldots, f(\boldsymbol{y}_s(t))\} \quad (2.7)$$

where s denotes the size of the swarm.

The velocity update step is specified for each dimension $j \in 1,\ldots,N_d$, hence, $v_{i,j}$ represents the j^{th} element of the velocity vector of the i^{th} particle. Thus the velocity of particle i is updated using the following equation:

$$v_{i,j}(t+1) = wv_{i,j}(t) + c_1 r_{1,j}(t)(y_{i,j}(t) - x_{i,j}(t)) + c_2 r_{2,j}(t)(\hat{y}_j(t) - x_{i,j}(t)) \quad (2.8)$$

where w is the inertia weight, c_1 and c_2 are the acceleration constants, and $r_{1,j}(t)$, $r_{2,j}(t) \sim U(0,1)$. Equation (2.8) consists of three components, namely

- The *inertia weight* term, w, which was first introduced by Shi and Eberhart [A modified 1998]. This term serves as a memory of previous velocities. The inertia weight controls the impact of the previous velocity: a large inertia weight favors exploration, while a small inertia weight favors exploitation [Shi and Eberhart, Parameter 1998].
- The *cognitive component*, $\boldsymbol{y}_i(t) - \boldsymbol{x}_i$, which represents the particle's own experience as to where the best solution is.
- The *social component*, $\hat{\boldsymbol{y}}(t) - \boldsymbol{x}_i(t)$, which represents the belief of the entire swarm as to where the best solution is.

According to Van den Bergh [2002], the relation between the inertia weight and acceleration constants should satisfy the following equation in order to have guaranteed convergence:

$$\frac{c_1 + c_2}{2} - 1 < w \tag{2.9}$$

Otherwise, the PSO particles may exhibit divergent of cyclic behavior. For a thorough study of the relationship between the inertia weight and acceleration constants, the reader is advised to refer to Ozcan and Mohan [1998], Clerc and Kennedy [2001], Van den Bergh [2002], Zheng *et al*. [2003], Yasuda *et al*. [2003] and Trelea [2003].

Velocity updates can also be clamped with a user defined maximum velocity, $V_{\max}$, which would prevent them from exploding, thereby causing premature convergence [Eberhart *et al*. 1996].

The position of particle i, $\boldsymbol{x}_i$, is then updated using the following equation:

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

The PSO updates the particles in the swarm using equations (2.8) and (2.10). This process is repeated until a specified number of iterations is exceeded, or velocity updates are close to zero. The quality of particles is measured using a fitness function which reflects the optimality of a particular solution. Figure 2.3 summarizes the basic PSO algorithm.

2.6.2 The *lbest* Model

For the *lbest* model, a swarm is divided into overlapping neighborhoods of particles. For each neighborhood N_i, the best particle is determined, with position $\hat{\boldsymbol{y}}_i$. This particle is referred to as the *neighborhood best* particle. Let the indices of the particles wrap around at s and the neighborhood size is l. Then the update equations are:

For each particle $i \in 1,...,s$ **do**

Randomly initialize $\boldsymbol{x}_i$

Randomly initialize $\boldsymbol{v}_i$ (or just set $\boldsymbol{v}_i$ to zero)

Set $\boldsymbol{y}_i = \boldsymbol{x}_i$

endfor

Repeat

For each particle $i \in 1,...,s$ **do**

Evaluate the fitness of particle i, $f(\boldsymbol{x}_i)$

Update $\boldsymbol{y}_i$ using equation (2.6)

Update $\hat{\boldsymbol{y}}$ using equation (2.7)

For each dimension $j \in 1,...,N_d$ **do**

Apply velocity update using equation (2.8)

endloop

Apply position update using equation (2.10)

endloop

Until some convergence criteria is satisfied

Figure 2.3: General pseudo-code for PSO

$$N_i = \{\boldsymbol{y}_{i-l}(t), \boldsymbol{y}_{i-l+1}(t), \ldots, \boldsymbol{y}_{i-1}(t), \boldsymbol{y}_i(t), \boldsymbol{y}_{i+1}(t), \ldots, \boldsymbol{y}_{i+l-1}(t), \boldsymbol{y}_{i+l}(t)\} \quad (2.11)$$

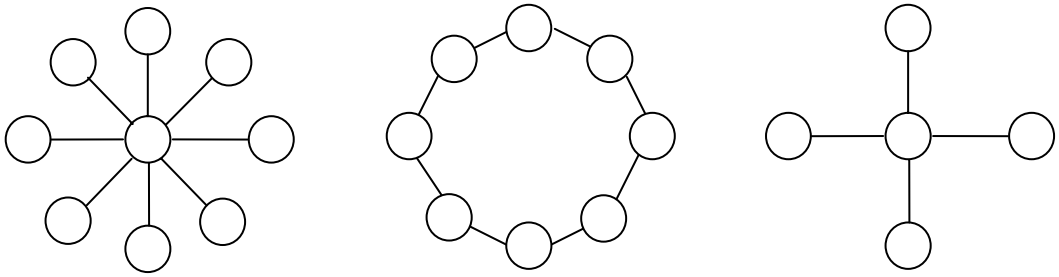
$$\hat{\boldsymbol{y}}_i(t+1) \in \{N_i \mid f(\hat{\boldsymbol{y}}_i(t+1)) = \min\{f(\boldsymbol{y}_i(t))\}, \forall \boldsymbol{y}_i \in N_i\} \quad (2.12)$$

$$v_{i,j}(t+1) = wv_{i,j}(t) + c_1 r_{1,j}(t)(y_{i,j}(t) - x_{i,j}(t)) + c_2 r_{2,j}(t)(\hat{y}_{i,j}(t) - x_{i,j}(t)) \quad (2.13)$$

The position update equation is the same as given in equation (2.10). Neighbors represent the social factor in PSO. Neighborhoods are usually determined using particle indices, however, topological neighborhoods can also be used [Suganthan 1999]. It is clear that *gbest* is a special case of *lbest* with $l = s$; that is, the neighborhood is the entire swarm. While the *lbest* approach results in a larger diversity, it is still slower than the *gbest* approach.

2.6.3 PSO Neighborhood topologies

Different neighborhood topologies have been investigated [Kennedy 1999; Kennedy and Mendes 2002]. Two common neighborhood topologies are the *star* (or *wheel*) and *ring* (or *circle*) topologies. For the star topology one particle is selected as a *hub*, which is connected to all other particles in the swarm. However, all the other particles are only connected to the hub. For the ring topology, particles are arranged in a ring. Each particle has some number of particles to its right and left as its neighborhood. Recently, Kennedy and Mendes [2002] proposed a new PSO model using a *Von Neumann* topology. For the Von Neumann topology, particles are connected using a grid network (2-dimensional lattice) where each particle is connected to its four neighbor particles (above, below, right and left particles). Figure 2.4 illustrates the different neighborhood topologies.



(a) Star topology (b) Ring Topology (c) Von Neumann Topology

Figure 2.4. A diagrammatic representation of neighborhood topologies

The choice of neighborhood topology has a profound effect on the propagation of the best solution found by the swarm. Using the *gbest* model the propagation is very fast (i.e. all the particles in the swarm will be affected by the best solution found in iteration t, immediately in iteration $t+1$). This fast propagation may result in the premature convergence problem discussed in Section 2.5.6. However, using the ring and Von Neumann topologies will slow down the convergence rate because the best solution found has to propagate through several neighborhoods before affecting all particles in the swarm. This slow propagation will enable the particles to explore more areas in the search space and thus decreases the chance of premature convergence.

2.6.4 The Binary PSO

Kennedy and Eberhart [1997] have adapted the PSO to search in binary spaces. For the binary PSO, the component values of $\boldsymbol{x}_i$, $\boldsymbol{y}_i$ and $\hat{\boldsymbol{y}}_i$ are restricted to the set {0, 1}. The velocity, $\boldsymbol{v}_i$, is interpreted as a probability to change a bit from 0 to 1, or from 1 to 0 when updating the position of particles. Therefore, the velocity vector remains continuous-valued. Since each $v_{i,j} \in \Re$, a mapping needs to be defined from $v_{i,j}$ to a

probability in the range [0, 1]. This is done by using a sigmoid function to squash velocities into a [0, 1] range. The sigmoid function is defined as

$$sig(v) = \frac{1}{1+e^{-v}} \tag{2.14}$$

The equation for updating positions (equation (2.10)) is then replaced by the probabilistic update equation [Kennedy and Eberhart 1997]:

$$x_{i,j}(t+1) = \begin{cases} 0 & \text{if } r_{3,j}(t) \geq sig(v_{i,j}(t+1)) \\ 1 & \text{if } r_{3,j}(t) < sig(v_{i,j}(t+1)) \end{cases} \tag{2.15}$$

where $r_{3,j}(t) \sim U(0,1)$.

It can be observed from equation (2.15) that if $sig(v_{i,j}) = 0$ then $x_{i,j} = 0$. This situation occurs when $v_{i,j} < -10$. Furthermore, $sig(v_{i,j})$ will saturate when $v_{i,j} > 10$ [Van den Bergh 2002]. To avoid this problem, it is suggested to set $v_{i,j} \in [-4,4]$ and to use velocity clamping with $V_{max} = 4$ [Kennedy and Eberhart 2001].

PSO has also been extended to deal with arbitrary discrete representation [Yoshida *et al*. 1999; Fukuyama and Yoshida 2001; Venter and Sobieszczanski-Sobieski 2002; Al-kazemi and Mohan 2000; Mohan and Al-Kazemi 2001]. These extensions are generally achieved by rounding $x_{i,j}$ to its closest discrete value after applying position update equation (2.10) [Venter and Sobieszczanski-Sobieski 2002].

2.6.5 PSO vs. GA

A PSO is an inherently continuous algorithm where as a GA is an inherently discrete algorithm [Venter and Sobieszczanski-Sobieski 2002]. Experiments conducted by Veeramachaneni *et al*. [2003] showed that a PSO performed better than GAs when applied on some continuous optimization problems. Furthermore, according to Robinson *et al*. [2002], a PSO performed better than GAs when applied to the design of a difficult engineering problem, namely, profiled corrugated horn antenna design [Diaz and Milligan 1996]. In addition, a binary PSO was compared with a GA by Eberhart and Shi [Comparison 1998] and Kennedy and Spears [1998]. The results showed that binary PSO is generally faster, more robust and performs better than binary GAs, especially when the dimension of a problem increases.

Hybrid approaches combining PSO and GA were proposed by Veeramachaneni *et al*. [2003] to optimize the profiled corrugated horn antenna. The hybridization works by taking the population of one algorithm when it has made no fitness improvement and using it as the starting population for the other algorithm. Two versions were proposed: GA-PSO and PSO-GA. In GA-PSO, the GA population is used to initialize the PSO population. For PSO-GA, the PSO population is used to initialize the GA population. According to Veeramachaneni *et al*. [2003], PSO-GA performed slightly better than PSO. Both PSO and PSO-GA outperformed both GA and GA-PSO.

Some of the first applications of PSO were to train Neural Networks (NNs), including NNs with product units. Results have shown that PSO is better than GA and other training algorithms [Eberhart and Shi, Evolving 1998; Van den Bergh and Engelbrecht 2000; Ismail and Engelbrecht 2000].

According to Shi and Eberhart [1998], the PSO performance is insensitive to the population size (however, the population size should not be too small). This observation was verified by Løvberg [2002] and Krink *et al*. [2002]. Consequently, PSO with smaller swarm sizes perform comparably to GAs with larger populations. Furthermore, Shi and Eberhart observed that PSO scales efficiently. This observation was verified by Løvberg [2002].

2.6.6 PSO and Constrained Optimization

Most engineering problems are constrained problems. However, the basic PSO is only defined for unconstrained problems. One way to allow the PSO to optimize constrained problems is by adding a penalty function to the original fitness function (as discussed in Section 2.5.2). In this thesis, a constant penalty function (empirically set to 10^6) is added to the original fitness function for each particle with violated constraints. More recently, a modification to the basic PSO was proposed by Venter and Sobieszczanski-Sobieski [2002] to penalize particles with violated constraints. The idea is to reset the velocity of each particle with violated constraints. Therefore, these particles will only be affected by $\boldsymbol{y}_i$ and $\hat{\boldsymbol{y}}$. According to Venter and Sobieszczanski-Sobieski [2002], this modification has a significant positive effect on the performance of PSO. Other PSO approaches dealing with constrained problems can be found in El-Gallad *et al*. [2001], Hu and Eberhart [Solving 2002], Schoofs and Naudts [2002], Parsopoulos and Vrahatis [2002], Coath *et al*. [2003] and Gaing [2003].

2.6.7 Drawbacks of PSO

PSO and other stochastic search algorithms have two major drawbacks [Løvberg 2002]. The first drawback of PSO, and other stochastic search algorithms, is that the swarm may prematurely converge (as discussed in Section 2.5.6). According to Angeline [Evolutionary 1998], although PSO finds good solutions much faster than other evolutionary algorithms, it usually can not improve the quality of the solutions as the number of iterations is increased. PSO usually suffers from premature convergence when strongly multi-modal problems are being optimized. The rationale behind this problem is that, for the *gbest* PSO, particles converge to a single point, which is on the line between the global best and the personal best positions. This point is not guaranteed to be even a local optimum. Proofs can be found in Van den Bergh [2002]. Another reason for this problem is the fast rate of information flow between particles, resulting in the creation of similar particles (with a loss in diversity) which increases the possibility of being trapped in local optima [Riget and Vesterstrøm 2002]. Several modifications of the PSO have been proposed to address this problem. Two of these modifications have already been discussed, namely, the inertia weight and the *lbest* model. Other modifications are discussed in the next section.

The second drawback is that stochastic approaches have problem-dependent performance. This dependency usually results from the parameter settings of each algorithm. Thus, using different parameter settings for one stochastic search algorithm result in high performance variances. In general, no single parameter setting exists which can be applied to all problems. This problem is magnified in PSO where modifying a PSO parameter may result in a proportionally large effect [Løvberg 2002]. For example, increasing the value of the inertia weight, w, will increase the speed of the particles resulting in more exploration (global search) and less

exploitation (local search). On the other hand, decreasing the value of *w* will decrease the speed of the particle resulting in more exploitation and less exploration. Thus finding the best value for *w* is not an easy task and it may differ from one problem to another. Therefore, it can be concluded that the PSO performance is problem-dependent.

One solution to the problem-dependent performance of PSO is to use self-adaptive parameters. In self-adaptation, the algorithm parameters are adjusted based on the feedback from the search process [Løvberg 2002]. Bäck [1992] has successfully applied self-adaptation on GAs. Self-adaptation has been successfully applied to PSO by Clerc [1999], Shi and Eberhart [2001], Hu and Eberhart [Adaptive 2002], Ratnaweera *et al*. [2003] and Tsou and MacNish [2003], Yasuda *et al*. [2003] and Zhang *et al*. [2003].

The problem-dependent performance problem can be addressed through *hybridization*. Hybridization refers to combining different approaches to benefit from the advantages of each approach [Løvberg 2002]. Hybridization has been successfully applied to PSO by Angeline [1998], Løvberg [2002], Krink and Løvbjerg [2002], Veeramachaneni *et al*. [2003], Reynolds *et al*. [2003], Higashi and Iba [2003] and Esquivel and Coello Coello [2003].

2.6.8 Improvements to PSO

The improvements presented in this section are mainly trying to address the problem of premature convergence associated with the original PSO. These improvements usually try to solve this problem by increasing the diversity of solutions in the swarm.

Constriction Factor

Clerc [1999] and Clerc and Kennedy [2001] proposed using a *constriction factor* to ensure convergence. The constriction factor can be used to choose values for w, c_1 and c_2 to ensure that the PSO converges. The modified velocity update equation is defined as follows:

$$v_{i,j}(t+1) = \chi(v_{i,j}(t) + c_1 r_{1,j}(t)(y_{i,j}(t) - x_{i,j}(t)) + c_2 r_{2,j}(t)(\hat{y}_j(t) - x_{i,j}(t))), \quad (2.16)$$

where χ is the constriction factor defined as follows:

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

and $\varphi = c_1 + c_2$, $\varphi > 4$.

Eberhart and Shi [2000] showed imperically that using both the constriction factor and velocity clamping generally improves both the performance and the convergence rate of the PSO.

Guaranteed Convergence PSO (GCPSO)

The original versions of PSO as given in Section 2.6.1, may prematurely converge when $\boldsymbol{x}_i = \boldsymbol{y}_i = \hat{\boldsymbol{y}}$, since the velocity update equation will depend only on the term $\boldsymbol{w}\boldsymbol{v}_i(t)$ [Van den Bergh and Engelbrecht 2002; Van den Bergh 2002]. To overcome this problem, a new version of PSO with guaranteed local convergence was introduced by Van den Bergh [2002], namely GCPSO. In GCPSO, the global best particle with index τ is updated using a different velocity update equation, namely

$$v_{\tau,j}(t+1) = -x_{\tau,j}(t) + \hat{y}_j(t) + wv_{\tau,j}(t) + \rho(t)(1-2r_{2,j}(t)) \tag{2.17}$$

which results in a position update equation of

$$x_{\tau,j}(t+1) = \hat{y}_j(t) + w\boldsymbol{v}_{\tau,j}(t) + \rho(t)(1-2r_{2,j}(t)) \tag{2.18}$$

The term $-\boldsymbol{x}_\tau$ resets the particle's position to the global best position $\hat{\boldsymbol{y}}$; $w\boldsymbol{v}_\tau(t)$ signifies a search direction, and $\rho(t)(1-2r_{2,j}(t))$ adds a random search term to the equation. The term $\rho(t)$ defines the area in which a better solution is searched.

The value of $\rho(0)$ is initialized to 1.0, with $\rho(t+1)$ defined as

$$\rho(t+1) = \begin{cases} 2\rho(\mathrm{t}) & \text{if } \#successes > s_c \\ 0.5\rho(\mathrm{t}) & \text{if } \#failures > f_c \\ \rho(\mathrm{t}) & \text{otherwise} \end{cases} \tag{2.19}$$

A *failure* occurs when $f(\hat{\boldsymbol{y}}(t)) \geq f(\hat{\boldsymbol{y}}(t\text{-}1))$ (in the case of a minimization problem) and the variable *#failures* is subsequently incremented (i.e. no apparent progress has been made). A *success* then occurs when $f(\hat{\boldsymbol{y}}(t)) < f(\hat{\boldsymbol{y}}(t\text{-}1))$. Van den Bergh [2002] suggests learning the control threshold values f_c and s_c dynamically. That is,

$$s_c(t+1) = \begin{cases} s_c(t)+1 & \text{if } \#failures(t+1) > f_c \\ s_c(t) & \text{otherwise} \end{cases} \tag{2.20}$$

$$f_c(t+1) = \begin{cases} f_c(t)+1 & \text{if } \#success(t+1) > s_c \\ f_c(t) & \text{otherwise} \end{cases} \tag{2.21}$$

This arrangement ensures that it is harder to reach a success state when multiple failures have been encountered. 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. For equation (2.19) to be well defined, the following rules should be implemented:

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

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

Van den Bergh suggests repeating the algorithm until ρ becomes sufficiently small, or until stopping criteria are met. Stopping the algorithm once 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.

It is important to note that, for the GCPSO algorithm, all particles except for the global best particle still follow equations (2.8) and (2.10). Only the global best particle follows the new velocity and position update equations.

According to Van den Bergh [2002] and Peer *et al*. [2003], GCPSO generally performs better than PSO when applied to benchmark problems. This improvement in performance is especially noticeable when PSO and GCPSO are applied to unimodal functions, but the performance of both algorithms was generally comparable for multi-modal functions [Van den Bergh 2002]. Furthermore, due to its fast rate of convergence, GCPSO is slightly more likely to be trapped in local optima [Van den Bergh 2002]. However, it has gauaranteed local convergence whereas the original PSO does not.

Multi-start PSO (MPSO)

Van den Bergh [2002] proposed MPSO which is an extension to GCPSO in order to make it a global search algorithm. MPSO works as follows:

1. Randomly initialize all the particles in the swarm.

2. Apply the GCPSO until convergence to a local optimum. Save the position of this local optimum.

3. Repeat Steps 1 and 2 until some stopping criteria are satisfied.

In Step 2, the GCPSO can be replaced by the original PSO. Several versions of MPSO were proposed by Van den Bergh [2002] based on the way used to determine the convergence of GCPSO. One good approach is to measure the rate of change in the objective function as follows:

$$f_{\text{ratio}} = \frac{f(\hat{\mathbf{y}}(t)) - f(\hat{\mathbf{y}}(t-1))}{f(\hat{\mathbf{y}}(t))}$$

If f_{ratio} is less than a user-specified threshold then a counter is incremented. The swarm is assumed to have converged if the counter reaches a certain threshold [Van den Bergh 2002]. According to Van den Bergh [2002], MPSO generally performed better than GCPSO in most of the tested cases. However, the performance of MPSO degrades significantly as the number of dimensions in the objective function increases [Van den Bergh 2002].

Attractive and Repulsive PSO (ARPSO)

ARPSO [Riget and Vesterstrøm 2002] alternates between two phases: attraction and repulsion based on a diversity measure. In the attraction phase, ARPSO uses PSO to allow fast information flow, as such particles attract each other and thus the diversity

reduces. It was found that 95% of fitness improvements were achieved in this phase. This observation shows the importance of low diversity in fine tuning the solution. In the repulsion phase, particles are pushed away from the best solution found so far thereby increasing diversity. Based on the experiments conducted by Riget and Vesterstrøm [2002] ARPSO outperformed PSO and GA in most of the tested cases.

Selection

A hybrid approach combining PSO with a tournament selection method was proposed by Angeline [Using Selection 1998]. Each particle is ranked based on its performance against a randomly selected group of particles. For this purpose, a particle is awarded one point for each opponent in the tournament for which the particle has a better fitnss. The population is then sorted in decending order according to the points accumulated. The bottom half of the population is then replaced by the top half. This step reduces the diversity of the population. The results showed that the hybrid approach performed better than the PSO (without w and χ) for unimodal functions. However, the hybrid approach performed worse than the PSO for functions with many local optima. Therefore, it can be concluded that although the use of a selection method improves the exploitation capability of the PSO, it reduces its exploration capability [Ven den Bergh 2002]. Hence, using a selection method with PSO may result in premature convergence.

Breeding

Løvberg *et al*. [2001] proposed a modification to PSO by using an arithmetic crossover operator (discussed in Section 2.5.4), referred to as a *breeding* operator, in order to improve the convergence rate of PSO. Each particle in the swarm is assigned

a user-defined breeding probability. Based on these probabilities, two parent particles are randomly selected to create offspring using the arithmetic crossover operator. Offspring replace the parent particles. The personal best position of each offspring particle is initialized to its current position (i.e. $\boldsymbol{y}_i = \boldsymbol{x}_i$), and its velocity is set as the sum of the two parent's velocities normalized to the original length of each parent velocity. The process is repeated until a new swarm of the same size has been generated. PSO with breeding generally performed better than the PSO when applied to multi-modal functions [Løvberg *et al*. 2001].

Mutation

Recently, Higashi and Iba [2003] proposed hybriding PSO with Gaussian mutation. Similarly, Esquivel and Coello Coello [2003] proposed hybridizing *lbest*- and *gbest*-PSO with a powerful diversity maintenance mechanism, namely, a non-uniform mutation operator discussed in section 2.5.5 to solve the premature convergence problem of PSO. According to Esquivel and Coello Coello [2003] the hybrid approach of *lbest* PSO and the non-uniform mutation operator outperformed PSO and GCPSO in all of the conducted experiments.

Dissipative PSO (DPSO)

DPSO was proposed by Xie *et al*. [2002] to add random mutation to PSO in order to prevent premature convergence. DPSO introduces negative entropy via the addition of randomness to the particles (after executing equation (2.8) and (2.10)) as follows:

If $(r_1(t) < c_v)$ **then** $v_{i,j}(t+1) = r_2(t)V_{\max}$

If $(r_3(t) < c_l)$ **then** $x_{i,j}(t+1) = R(t)$

where $r_1(t) \sim U(0,1)$, $r_2(t) \sim U(0,1)$ and $r_3(t) \sim U(0,1)$; c_v and c_v are chaotic factors in the range [0,1] and $R(t) \sim U(Z_{\min},Z_{\max})$ where $Z_{\min}$ and $Z_{\max}$ are the lower and upper bound of the search space. The results showed that DPSO performed better than PSO when applied to the benchmarks problems [Xie *et al*. 2002].

Differential Evolution PSO (DEPSO)

DEPSO [Zhang and Xie 2003] uses a differential evolution (DE) operator [Storn and Price 1997] to provide the mutations. A trait point $\ddot{\boldsymbol{y}}_i(t)$ is calculated as follows:

If $(r_1(t) < p_c$ **OR** $j = k_d)$ **then**

$$\ddot{y}_{i,j}(t) = \hat{y}_j(t) + \frac{(y_{1,j}(t) - y_{2,j}(t)) + (y_{3,j}(t) - y_{4,j}(t))}{2} \tag{2.23}$$

where $r_1(t) \sim U(0,1)$, $k_d \sim U(1,N_d)$, and $\boldsymbol{y}_1(t)$, $\boldsymbol{y}_2(t)$, $\boldsymbol{y}_3(t)$ and $\boldsymbol{y}_4(t)$ are randomly chosen from the set of personal best positions. Then, $\boldsymbol{y}_i(t) = \ddot{\boldsymbol{y}}_i(t)$, only if $f(\ddot{\boldsymbol{y}}_i(t)) < f(\boldsymbol{y}_i(t))$. The rationale behind mutating $\boldsymbol{y}_i(t)$ instead of $\boldsymbol{x}_i(t)$ is to avoid disorganization of the swarm.

DEPSO works by alternating between the original PSO and the DE operator such that equations (2.8) and (2.10) are used in the odd iterations and equation (2.23) is used is in the even iterations. According to Zhang and Xie [2003], DEPSO generally performed better than PSO, DE, GA, ES, DPSO and fuzzy-adaptive PSO when applied to the benchmark functions.

Craziness

To avoid premature convergence, Kennedy and Eberhart [1995] introduced the use of a craziness operator with PSO. However, they concluded that this operator may not be necessay. More recently, Venter and Sobieszczanski-Sobieski [2002] reintroduced the craziness operator to PSO. In each iteration, a few particles far from the center of the swarm are selected. The positions of these particles are then randomly changed while their velocities are initialized to the cognitive velocity component, i.e.

$$v_{i,j}(t+1) = c_1 r_{1,j}(t)(y_{i,j}(t) - x_{i,j}(t)) \tag{2.24}$$

According to Venter and Sobieszczanski-Sobieski [2002], the proposed craziness operator does not seem to have a big influence on the performance of PSO.

The LifeCycle Model

A self-adaptive heuristic search algorithm, called LifeCycle, was proposed by Krink and Løvbjerg [2002]. LifeCycle is a hybrid approach combing PSO, GA and Hill-Climbing approaches. The motivation for LifeCycle is to gain the benefits of PSO, GA and Hill-Climbing in one algorithm. In LifeCycle, the individuals (representing potential solutions) start as PSO particles, then depending on their performance (in searching for solutions) can change into GA individuals, or Hill-Climbers. Then, they can return back to particles. This process is repeated until convergence. The LifeCycle was compared with PSO, GA and Hill-Climbing [Krink, and Løvbjerg 2002] and has generally shown good performance when applied to the benchmark problems. However, PSO performed better than (or comparable to) the LifeCycle in three out of five benchmark problems. Another hybrid approach proposed by Veeramachaneni *et*

al. [2003] combining PSO and GA has already been discussed in Section 2.6.5. From the experimental results of Krink and Løvbjerg [2002] and Veeramachaneni *et al*. [2003], it can be observed that the original PSO performed well compared to their more complicated hybrid approaches.

Multi-Swarm (or subpopulation)

The idea of using several swarms instead of one swarm was applied to PSO by Løvberg *et al*. [2001] and Van den Bergh and Engelbrecht [2001]. The approach proposed by Løvberg *et al*. [2001] is an extension of PSO with the breeding operator discussed above. The idea is to divide the swarm into several swarms. Each swarm has its own global best particles. The only interaction between the swarms occurs when the breeding selects two particles to mate from different swarms. The results in Løvberg *et al*. [2001] showed that this approach did not improve the performance of PSO. The expected reasons are [Jensen and Kristensen 2002]:

- The authors split a swarm of 20 particles into six different swarms. Hence, each swarm contains a few particles. Swarms with few particles have little diversity and therefore little exploration power.
- No action has been taken to prevent swarms from being too similar to each other.

The above problems were addressed by Jensen and Kristensen [2002]. The modified approach works by using two swarms (each with a size of 20 particles) and keeping them away from each other either by randomly spreading the swarm (with the worst performance) over the search space or by adding a small mutation to the positions of the particles in this swarm. The approach using the mutation technique generally

performed better than PSO when applied to the benchmark problems [Jensen and Kristensen 2002]. However, one drawback of this approach is the fact that the decision of whether two swarms are too close to each other is very problem dependent [Jensen and Kristensen 2002].

Self-Organized Criticality (SOC PSO)

In order to increase the population diversity to avoid premature convergence, Løvberg and Krink [2002] extended PSO with *Self Organized Criticality* (SOC). A measure, called *criticality*, of how close particles are to each other is used to relocate the particles and thus increase the diversity of the swarm. A particle with a high criticality disperses its criticality by increasing the criticality of a user-specified number of particles, *CL*, in its neighborhood by 1. Then, the particle reduces its own criticality value by *CL*. The particle then relocates itself. Two types of relocation were investigated: the first re-initializes the particle, while the second pushes the particle with high criticality a little further in the search space. According to Løvberg and Krink [2002], the first relocation approach produced better results when applied to the tested functions. SOC PSO outperformed PSO in one case out of the four cases used in the experiments. However, adding a tenth of the criticality value of a particle to its own inertia (w was set to 0.2) results in a significant improvement of the SOC PSO compared to PSO [Løvberg and Krink 2002].

Fitness-Distance Ratio based PSO (FDR-PSO)

Recently, Veeramachaneni *et al*. [2003] proposed a major modification to the way PSO operates by adding a new term to the velocity update equation. The new term

allows each particle to move towards a particle in its neighborhood that has a better personal best position. The modified velocity update equation is defined as:

$$v_{i,j}(t+1) = wv_{i,j}(t) + \psi_1(y_{i,j}(t) - x_{i,j}(t)) + \psi_2(\hat{y}_j(t) - x_{i,j}(t)) + \psi_3(y_{\eta,j}(t) - x_{i,j}(t)) \quad (2.25)$$

where ψ_1, ψ_2 and ψ_3 are user-specified parameters and each $y_{\eta,j}(t)$ is chosen by maximizing

$$\frac{f(\boldsymbol{x}_i(t)) - f(\boldsymbol{y}_\eta(t))}{\left|y_{\eta,j}(t) - x_{i,j}(t)\right|} \quad (2.26)$$

where |.| represents the absolute value.

According to Veeramachaneni *et al*. [2003], FDR-PSO decreases the possibility of premature convergence and thus is less likely to be trapped in local optima. In addition, FDR-PSO (using $\psi_1 = \psi_2 = 1$ and $\psi_3 = 2$) outperformed PSO and several other variations of PSO, namely, ARPSO, DPSO, SOC PSO and Multi Swarm PSO [Løvberg *et al*. 2001], in different tested benchmark problems [Veeramachaneni *et al*. 2003].

2.7 Ant Systems

Another population-based stochastic approach is Ant Systems. Ant Systems were first introduced by Dorigo [1992] and Dorigo *et al*. [1991] to solve some difficult combinatorial optimization problems [Dorigo *et al*. 1999]. Ant systems were inspired by the observation of real ant colonies. In real ant colonies, ants communicate with

each other indirectly through depositing a chemical substance, called *pheromone*. Ants use, for example, pheromones to find the shortest path to food. This indirect way of communication via pheromones is called *stigmergy* [Dorigo *et al*. 1999].

Using Ant Colony Optimization (ACO), a finite size colony of artificial ants cooperate with each other via stigmergy to find quality solutions to optimization problems. Good solutions result from the cooperation of the artificial ants. ACO was applied to a wide range of optimization problems such as the traveling salesman problem, and routing and load balancing in packet switched networks with encouraging results [Dorigo *et al*. 1999]. More details about Ant Systems and their applications can be found in Bonabeau *et al*. [1999] and Dorigo and Di Caro [1999]. Ant systems and their applications are outside the scope of this thesis.

2.8 Conclusions

This chapter provided a short overview of optimization and optimization methods with a special emphasis on PSO. From the discussed methods, PSO (and GA for comparison purposes) is used in this thesis to optimize a set of problems in the field of pattern recognition and image processing. These problems are introduced in the next chapter.