Particle Swarm Optimization Methods for Pattern Recognition and Image Processing

by

Mahamed G. H. Omran

Submitted in partial fulfillment of the requirements for the degree Philosophiae Doctor in the Faculty of Engineering, Built Environment and Information Technology

University of Pretoria

Pretoria

November 2004

Particle Swarm Optimization Methods for Pattern Recognition and Image Processing

by
Mahamed G. H. Omran

Abstract

Pattern recognition has as its objective to classify objects into different categories and classes. It is a fundamental component of artificial intelligence and computer vision. This thesis investigates the application of an efficient optimization method, known as Particle Swarm Optimization (PSO), to the field of pattern recognition and image processing. First a clustering method that is based on PSO is proposed. The application of the proposed clustering algorithm to the problem of unsupervised classification and segmentation of images is investigated. A new automatic image generation tool tailored specifically for the verification and comparison of various unsupervised image classification algorithms is then developed. A dynamic clustering algorithm which automatically determines the "optimum" number of clusters and simultaneously clusters the data set with minimal user interference is then developed. Finally, PSO-based approaches are proposed to tackle the color image quantization and spectral unmixing problems. In all the proposed approaches, the influence of PSO parameters on the performance of the proposed algorithms is evaluated.

Key terms: Clustering, Color Image Quantization, Dynamic Clustering, Image Processing, Image Segmentation, Optimization Methods, Particle Swarm Optimization, Pattern Recognition, Spectral Unmixing, Unsupervised Image Classification.

Thesis supervisor: Prof. A. P. Engelbrecht

Thesis co-supervisor: Dr. Ayed Salman
Department of Computer Engineering, Kuwait University, Kuwait

Department of Computer Science

Degree: Philosophiae Doctor

"Obstacles are those frightening things you see when you take your eyes off your goal."

Henry Ford

"You will recognize your own path when you come upon it, because you will suddenly have all the energy and imagination you will ever need."

Jerry Gillies

Acknowledgments

I address my sincere gratitude to God as whenever I faced any difficulty I used to pray to God to help me and He always was there protecting and saving me.

Then, I would like to express my warm thanks to Professor Andries Engelbrecht, who spared no effort in supporting me with all care and patience. I enjoyed working with him, making every moment I spent in the research work as enjoyable as can be imagined.

I would like also to thank my co-supervisor Dr. Ayed Salman from Kuwait University for his continuous guidance, encouragement and patience throughout the PhD journey. I will never forget the long hours we spent together discussing various ideas and methods.

Last but not least, I would love to thank my family for their support and care, especially my mother Aysha and my father Ghassib. May God bless and protect them both hoping that God will help me to repay them part of what they really deserve. I also thank my two sisters Ala'a and Esra'a for their help in preparing this thesis.

Contents

Chapter 1

Introduction....1

1.1 Motivation....1
1.2 Objectives....2
1.3 Methodology....3
1.4 Contributions....4
1.5 Thesis Outline....5

Chapter 2

Optimization and Optimization Methods....7

2.1 Optimization....7
2.2 Traditional Optimization Algorithms....10
2.3 Stochastic Algorithms....11
2.4 Evolutionary Algorithms....12
2.5 Genetic Algorithms....15
2.5.1 Solution Representation....16
2.5.2 Fitness Function....16
2.5.3 Selection....17
2.5.4 Crossover....19
2.5.5 Mutation....20
2.5.6 The Premature Convergence Problem....22
2.6 Particle Swarm Optimization....23
2.6.1 The PSO Algorithm....23
2.6.2 The *lbest* Model....26
2.6.3 PSO Neighborhood topologies....28
2.6.4 The Binary PSO....29
2.6.5 PSO vs. GA....31
2.6.6 PSO and Constrained Optimization....32
2.6.7 Drawbacks of PSO....33
2.6.8 Improvements to PSO....34
2.7 Ant Systems....45
2.8 Conclusions....46

Chapter 3

Problem Definiton....47

3.1 The Clustering Problem....47
3.1.1 Definitions....48
3.1.2 Similarity Measures....49
3.1.3 Clustering Techniques....51
3.1.4 Clustering Validation Techniques....64
3.1.5 Determining the Number of Clusters....69
3.1.6 Clustering using Self-Organizing Maps....75

3.1.7 Clustering using Stochastic Algorithms....78
3.1.8 Unsupervised Image Classification....82
3.2 Image Segmentation using Clustering....83
3.2.1 Thresholding Techniques....84
3.2.2 Edge-based Techniques....84
3.2.3 Region growing Techniques....85
3.2.4 Clustering Techniques....85
3.3 Color Image Quantization....89
3.3.1 Pre-clustering approaches....91
3.3.2 Post-clustering approaches....94
3.4 Spectral Unmixing....97
3.4.1 Linear Pixel Unmixing (or Linear Mixture Modeling)....98
3.4.2 Selection of the End-Members....100
3.5 Conclusions....103

Chapter 4

A PSO-based Clustering Algorithm with Application to Unsupervised Image Classification....104

4.1 PSO-Based Clustering Algorithm....104
4.1.1 Measure of Quality....104
4.1.2 PSO-Based Clustering Algorithm....105
4.1.3 A Fast Implementation....107
4.2 Experimental Results....108
4.2.1 *gbest* PSO versus K-Means....111
4.2.2 Improved Fitness Function....114
4.2.3 *gbest* PSO versus GCPSO....115
4.2.4 Influence of PSO Parameters....116
4.2.5 *gbest* PSO versus *state-of-the-art* clustering algorithms....122
4.2.6 Different Versions of PSO....126
4.2.7 A Non-parametric Fitness Function....128
4.2.8 Multispectral Imagery Data....129
4.2.9 PSO for Data Clustering....134
4.3 Conclusions....134

Chapter 5

SIGT: Synthetic Image Generation Tool for Clustering Algorithms....136

5.1 Need for Benchmarks....136
5.2 SIGT: Synthetic Image Generation Tool....138
5.2.1 Synthetic Image Generator....139
5.2.2 Clustering Verification Unit....141
5.3 Experimental Results....144
5.4 Conclusions....146

Chapter 6

Dynamic Clustering using Particle Swarm Optimization with Application to Unsupervised Image Classification 153

6.1 The Dynamic Clustering using PSO (DCPSO) Algorithm 153
6.1.1 Validity Index 158
6.1.2 Time Complexity 158
6.2 Experimental results 159
6.2.1 Synthetic images 162
6.2.2 Natural images 163
6.2.3 Comparison with GA and RS 166
6.2.4 Swarm Size 167
6.2.5 The Termination Criteria 168
6.2.6 p_{ini} and N_c 171
6.2.7 Comparison of *gbest*-, *lbest*- and *lbest-to-gbest*-PSO 173
6.2.8 Multispectral Imagery Data 174
6.3 Conclusions 175

Chapter 7

Applications 177

7.1 A PSO-based Color Image Quantization Algorithm 177
7.1.1 The PSO-based Color Image Quantization (PSO-CIQ) Algorithm 178
7.1.2 Experimental Results 181
7.2 A PSO-based End-Member Selection Method for Spectral Unmixing of Multispectral Satellite Images 192
7.2.1 The PSO-based End-Member Selection (PSO-EMS) Algorithm 192
7.2.2 Experimental Results 195
7.3 Conclusions 207

Chapter 8

Conclusion 208

8.1 Summary 208
8.2 Future Research 210

Bibliography 213

Appendix A

Definition of Terms and Symbols 238

Appendix B

Derived Publications 239

List of Figures

Figure 2.1: Example of a global minimizer x^* as well as a local minimizer x_B^* 10
Figure 2.2: General pseudo-code for EAs.....13
Figure 2.3: General pseudo-code for PSO 27
Figure 2.4. A diagrammatic representation of neighborhood topologies 29
Figure 3.1: General pseudo-code for SOM.....76
Figure 3.2: Rectangular Lattice arrangement of neighborhoods 77
Figure 3.3: General simulated annealing based clustering algorithm.....78
Figure 3.4: General pseudo-code for GA-based clustering algorithm.....80
Figure 4.1: The PSO clustering algorithm 107
Figure 4.2: Data set consisting of synthetic, MRI and LANDSAT images.....110
Figure 4.3: PSO Performance on Synthetic Image 112
Figure 4.4: The Segmented Synthetic Images 113
Figure 4.5: The Segmented MRI Images.....113
Figure 4.6: The Segmented Lake Tahoe Images 113
Figure 4.7: Effect of swarm size on synthetic image.....118
Figure 4.8: Effect of swarm size on MRI image.....119
Figure 4.9: The Landsat MSS test images of Lake Tahoe.....132
Figure 4.10: The Thematic Maps for Lake Tahoe Image Set 133
Figure 5.1: The synthetic image generator algorithm.....140
Figure 5.2: The clustering verification algorithm 143
Figure 6.1: The DCPSO algorithm 156
Figure 6.2: Natural Images 160
Figure 6.3: 6-Clusters thematic map obtained using DCPSO.....175
Figure 7.1: The PSO-CIQ algorithm.....180
Figure 7.2: Quantization results for the Lenna image using PSO-CIQ 184
Figure 7.3: Quantization results for the peppers image using PSO-CIQ.....185
Figure 7.4: Quantization results for the jet image using PSO-CIQ 186
Figure 7.5: Quantization results for the mandrill image using PSO-CIQ.....187
Figure 7.6: The PSO-EMS algorithm 195
Figure 7.7: AVHRR Image of UK, Size: 847x1009 , 5 bands, 10-bits per pixel 199
Figure 7.8: Species concentration maps resulting from the application of ISO-UNMIX to unmix the Lake Tahoe test image set.....200

Figure 7.9: Species concentration maps resulting from the application of PSO-EMS to unmix the Lake Tahoe test image set.....201
Figure 7.10: Species concentration maps resulting from the application of ISO-UNMIX to unmix the UK test image set 202
Figure 7.11: Species concentration maps resulting from the application of PSO-EMS to unmix the UK test image set.....203

List of Tables

Table 4.1: Comparison between K-means and PSO 112
Table 4.2: 2-component versus 3-component fitness function 115
Table 4.3: PSO versus GCPSO 116
Table 4.4: Effect of inertia weight on the synthetic image 120
Table 4.5: Effect of inertia weight on the MRI image 120
Table 4.6: Effect of acceleration coefficients on the synthetic image 121
Table 4.7: Effect of acceleration coefficients on the MRI image 121
Table 4.8: Effect of sub-objective weight values on synthetic image 123
Table 4.9: Effect of sub-objective weight values on MRI image 124
Table 4.10: Comparison between K-means, FCM, KHM, H2, GA and PSO for fitness function defined in equation (4.6) 125
Table 4.11: Comparison of different PSO versions 127
Table 4.12: Comparison between K-means, FCM, KHM, H2, GA and PSO for fitness function defined in equation (4.7) 130
Table 4.13: Comparison between different non-parametric fitness function 131
Table 4.14: Comparison between K-means, *gbest* PSO and *lbest-to-gbest* PSO when applied to multispectral image set 131
Table 5.1: Synthetic image details and classification accuracy 148
Table 5.2: Synthetic images, Histograms and Thematic Maps 149
Table 5.2 (*continued*). 150
Table 5.2 (*continued*). 151
Table 5.2 (*continued*). 152
Table 6.1: Additional synthetic images used along with the corresponding histograms 161
Table 6.2: Experiments on synthetic images 163
Table 6.3: Experiments on natural images 164
Table 6.4: Samples of segmented images resulting from DCPSO using V 165
Table 6.4: Samples of segmented images resulting from DCPSO using V (*continued*) 166
Table 6.5: Comparison of PSO-, GA- and RS- versions of the proposed approach .. 167
Table 6.6: Comparison of PSO- and GA- versions of the proposed approach using a swarm size $s = 20$ 168

Table 6.7: Effect of termination criterion TC_1 on the DCPSO using a swarm size s = 20 and TC_2= 2 170
Table 6.8: Effect of termination criterion TC_2 on the DCPSO using a swarm size s = 20 and TC_1= 50 171
Table 6.9: Effect of p_{ini} on the DCPSO using a swarm size s = 20 172
Table 6.10: Effect of N_c on the DCPSO using a swarm size s = 20 173
Table 6.11: Comparison of *gbest*-, *lbest*- and *lbest-to-gbest*- PSO versions of DCPSO using V (s = 20) 174
Table 7.1: Comparison between SOM, GCMA and PSO-CIQ 183
Table 7.2: Effect of V_{max} on the performance of PSO-CIQ using Lenna image (16 colors) 188
Table 7.3: Effect of the swarm size on the performance of PSO-CIQ using Lenna image (16 colors) 189
Table 7.4: Effect of the number of PSO iterations on the performance of PSO-CIQ using Lenna image (16 colors) 189
Table 7.5: Effect of p_{kmeans} on the performance of PSO-CIQ using Lenna image (16 colors) 190
Table 7.6: Effect of the number of K-means iterations on the performance of PSO-CIQ using Lenna image (16 colors) 191
Table 7.7: Comparison of *gbest*-, *lbest*- and *lbest-to-gbest*- PSO versions of PSO-CIQ using Lenna image (16 colors) 191
Table 7.8: Comparison between ISO-UNMIX and PSO-EMS 198
Table 7.9: Effect of V_{max} on the performance of PSO-EMS using Lake Tahoe image set 198
Table 7.10: Effect of the swarm size on the performance of PSO-EMS using Lake Tahoe image set 204
Table 7.11: Effect of the number of PSO iterations on the performance of PSO-EMS using Lake Tahoe image set 204
Table 7.12: Effect of p_{kmeans} on the performance of PSO-EMS using Lake Tahoe image set 205
Table 7.13: Effect of the number of K-means iterations on the performance of PSO-EMS using Lake Tahoe image set 206
Table 7.14: Comparison of *gbest*-, *lbest*- and *lbest-to-gbest*- PSO versions of PSO-EMS using Lake Tahoe image set 206

Chapter 1

Introduction

As humans, it is easy (even for a child) to recognize letters, objects, numbers, voices of friends, etc. However, making a computer solve these types of problems is a very difficult task. Pattern recognition is the science with the objective to classify objects into different categories and classes. It is a fundamental component of artificial intelligence and computer vision. Pattern recognition methods are used in various areas such as science, engineering, business, medicine, etc. Interest in pattern recognition is fast growing in order to deal with the prohibitive amount of information we encounter in our daily life. Automation is desperately needed to handle this information explosion. This thesis investigates the application of an efficient optimization method, known as Particle Swarm Optimization, to the field of pattern recognition and image processing. PSOs solve optimization problems by simulating the social behavior of bird flocks.

1.1 Motivation

There are many difficult problems in the field of pattern recognition and image processing. These problems are the focus of much active research in order to find efficient approaches to address them. However, the outcome of the research is still unsatisfactory.

Local search approaches were generally used to solve difficult problems in the field of pattern recognition and image processing. However, the selected set of

problems in this thesis are NP-hard and combinatorial. Hence, evolutionary algorithms are generally more suitable to solve these difficult problems because they are population-based stochastic approaches. Thus, evolutionary algorithms can avoid being trapped in a local optimum and can often find a global optimal solution. A PSO is a population-based stochastic optimization algorithm modeled after the simulation of the social behavior of bird flocks. PSO is easy to implement and has been successfully applied to solve a wide range of optimization problems [Hu 2004]. Thus, due to its simplicity and efficiency in navigating large search spaces for optimal solutions, PSOs are used in this research to develop efficient, robust and flexible algorithms to solve a selective set of difficult problems in the field of pattern recognition and image processing. Out of these problems, data clustering is elaborately tackled in this thesis specifically image data. The motivation for the focus on data clustering is the fact that data clustering is an important process in pattern recognition and machine learning. Actually, clustering is a primary goal of pattern recognition. Furthermore, it is a central process in Artificial Intelligence. In addition, clustering algorithms are used in many applications, such as image segmentation, vector and color image quantization, spectral unmixing, data mining, compression, etc. Therefore, finding an efficient clustering algorithm is very important for researchers in many different disciplines.

1.2 Objectives

The primary objectives of this thesis can be summarized as follows:

- To show that the PSO can be successfully used to solve difficult problems in pattern recognition and image processing.

- To develop an efficient clustering algorithm based on PSO.
- To develop a tool that can aid researchers in the unsupervised image classification field to test their algorithms, compare different clustering algorithms and generate benchmarks.
- To develop an efficient dynamic clustering algorithm that can find the "optimum" number of clusters in a data set with minimum user interference.
- To develop a PSO-based approach to tackle the color image quantization problem.
- To develop an efficient end-members selection method based on PSO for spectral unmixing of multispectral imagery data.

1.3 Methodology

Algorithms proposed in this thesis are first presented and discussed. Experimental results were then generally obtained using various synthetic images with well-known characteristics in order to show the accuracy and efficiency of the proposed algorithms.

In addition, natural images from different areas such as medical images and remotely sensed satellite images were also used to show the wide applicability of the proposed approaches.

The results of *state-of-the-art* algorithms when applied to the same test images were also reported to show the relative performance of the proposed approaches when compared to other well-known approaches.

For the task of unsupervised image classification, attempts were made to find the best values for the PSO parameters.

Due to the stochastic nature of the proposed algorithms, all the presented results are averages and standard deviations over several simulations. However, due to the computational expensive nature of the simulations, results were generally taken over 10 or 20 runs.

1.4 Contributions

The main contributions of this thesis are:

- The development of an efficient clustering algorithm based on the PSO that performs better than *state-of-the-art* clustering algorithms when applied to the problem of unsupervised image classification.
- The development of a simple tool for synthetic image generation and verification. This tool can be used as a preliminary test to compare different unsupervised image classification algorithms. In addition, it can be used to generate a set of benchmark images that can be used by the researchers in the field of unsupervised image classification.
- The development of an efficient dynamic clustering algorithm based on the PSO that is able to simultaneously cluster a data set and find the "optimum" number of clusters in the data set.
- The development of an efficient color image quantization algorithm based on the PSO which is capable of generating high quality quantized images.
- The development of an efficient end-members selection method for spectral unmixing of multispectral satellite imagery data which is based on the PSO. The efficiency of the algorithm is demonstrated by applying it to test imagery from various platforms.

1.5 Thesis Outline

Chapter 2 briefly reviews the subject of optimization. This is followed by a brief discussion of traditional and stochastic optimization methods. Evolutionary Algorithms (EAs) (with more emphasis on Genetic Algorithms (GAs)) are then presented. This is followed by an elaborated discussion of particle swarm optimization and its various modifications. PSO is a model from the swarm intelligence paradigm. Therefore in order to provide a complete coverage of swarm intelligence background, a brief overview of another swarm intelligence model, Ant Colony Systems, is given.

Chapter 3 reviews the problems addressed in this thesis in sufficient detail. First the clustering problem is defined and different clustering concepts and approaches are presented. This is followed by defining image segmentation in addition to presenting various image segmentation methods. A survey of color image quantization and its approaches is then presented. This is followed by a brief introduction to spectral unmixing.

Chapter 4 presents a clustering method that is based on PSO. The algorithm finds the centroids of a user specified number of clusters, where each cluster groups together similar patterns. The application of the proposed clustering algorithm to the problem of unsupervised classification and segmentation of images is investigated. To illustrate its wide applicability, the proposed algorithm is then applied to synthetic, MRI and satellite images.

Chapter 5 presents a new automatic image generation tool tailored specifically for the verification and comparison of different unsupervised image classification

algorithms. The usefulness of the tool is demonstrated in this chapter with reference to the well-known K-means clustering algorithm and the PSO-based clustering algorithm proposed in the chapter 4.

Chapter 6 presents a new dynamic clustering approach based on PSO. This approach is applied to unsupervised image classification. The proposed approach automatically determines the "optimum" number of clusters and simultaneously clusters the data set with minimal user interference. The proposed approach is then applied to synthetic, natural and multispectral images. A genetic algorithm and a random search version of dynamic clustering are presented and compared to the particle swarm version.

Chapter 7 presents PSO-based approaches to tackle the color image quantization and spectral unmixing problems. The proposed approaches are then applied on different image sets to show their applicability and they are compared with other *state-of-the-art* approaches.

Chapter 8 highlights the conclusions of this thesis and discusses directions for future research.

The appendices present a definition of frequently used terms and symbols and a list of publications derived from the work introduced in this thesis.

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].

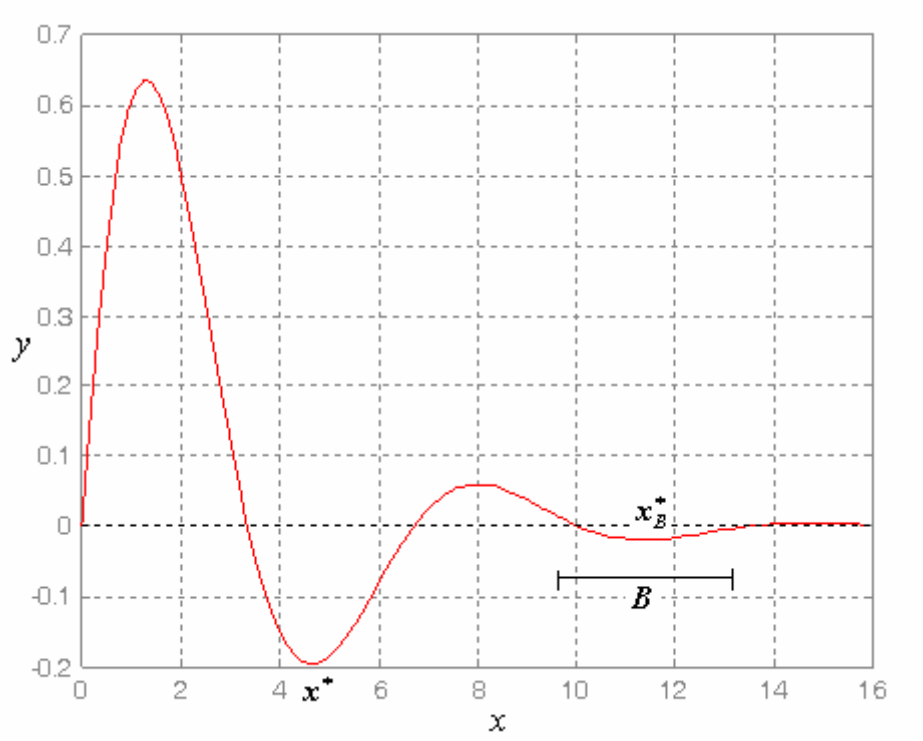


Figure 2.1: Example of a global minimizer x^* as well as a local minimizer x_B^*

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

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

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

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

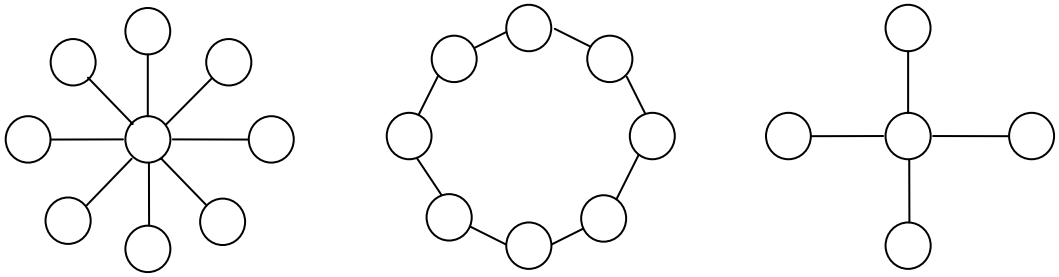
$$\hat{\boldsymbol{y}}_i(t+1) \in \{N_i \mid f(\hat{\boldsymbol{y}}_i(t+1)) = \min\{f(\boldsymbol{y}_i(t))\}, \forall \boldsymbol{y}_i \in N_i\} \quad (2.12)$$

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

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

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

which results in a position update equation of

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

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

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

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,

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

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

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.

Chapter 3

Problem Definition

This chapter reviews the problems addressed in this thesis in sufficient detail. First the clustering problem is defined and different clustering concepts and approaches are discussed. This is followed by defining image segmentation in addition to presenting various image segmentation methods. A survey of color image quantization and approaches to quantization are then presented. This is followed by a brief introduction to spectral unmixing.

3.1 The Clustering Problem

Data clustering is the process of identifying natural groupings or clusters within multidimensional data based on some similarity measure (e.g. Euclidean distance) [Jain *et al*. 1999; Jain *et al*. 2000]. It is an important process in pattern recognition and machine learning [Hamerly and Elkan 2002]. Furthermore, data clustering is a central process in Artificial Intelligence (AI) [Hamerly 2003]. Clustering algorithms are used in many applications, such as image segmentation [Coleman and Andrews 1979; Jain and Dubes 1988; Turi 2001], vector and color image quantization [Kaukoranta *et al*. 1998; Baek *et al*. 1998; Xiang 1997], data mining [Judd *et al*. 1998], compression [Abbas and Fahmy 1994], machine learning [Carpineto and Romano 1996], etc. A cluster is usually identified by a cluster center (or *centroid*) [Lee and Antonsson 2000]. Data clustering is a difficult problem in unsupervised pattern recognition as the clusters in data may have different shapes and sizes [Jain *et al*. 2000].

3.1.1 Definitions

The following terms are used in this thesis:

- A *pattern* (or *feature vector*), $\boldsymbol{z}$, is a single object or data point used by the clustering algorithm [Jain *et al*. 1999].
- A *feature* (or *attribute*) is an individual component of a pattern [Jain *et al*. 1999].
- A *cluster* is a set of similar patterns, and patterns from different clusters are not similar [Everitt 1974].
- *Hard* (or *Crisp*) clustering algorithms assign each pattern to one and only one cluster.
- *Fuzzy* clustering algorithms assign each pattern to each cluster with some degree of membership.
- A *distance measure* is a metric used to evaluate the similarity of patterns [Jain *et al*. 1999].

The clustering problem can be formally defined as follows (Veenman *et al*. 2003):

Given a data set $\boldsymbol{Z} = \{\boldsymbol{z}_1, \boldsymbol{z}_2, \ldots, \boldsymbol{z}_p, \ldots, \boldsymbol{z}_{N_p}\}$ where $\mathbf{z}_p$ is a pattern in the N_d-dimensional feature space, and N_p is the number of patterns in $\boldsymbol{Z}$, then the clustering of $\boldsymbol{Z}$ is the partitioning of $\boldsymbol{Z}$ into K clusters $\{\boldsymbol{C}_1, \boldsymbol{C}_2, \ldots, \boldsymbol{C}_K\}$ satisfying the following conditions:

- Each pattern should be assigned to a cluster, i.e.

  $\cup_{k=1}^{K} \boldsymbol{C}_k = \boldsymbol{Z}$

- Each cluster has at least one pattern assigned to it, i.e.

$\boldsymbol{C}_k \neq \phi, \quad k = 1,\ldots,K$

- Each pattern is assigned to one and only one cluster (in case of hard clustering only), i.e.

  $\boldsymbol{C}_k \cap \boldsymbol{C}_{kk} = \phi \quad \text{where } k \neq kk$

3.1.2 Similarity Measures

As previously mentioned, clustering is the process of identifying natural groupings or clusters within multidimensional data based on some similarity measure. Hence, similarity measures are fundamental components in most clustering algorithms [Jain *et al*. 1999].

The most popular way to evaluate a similarity measure is the use of distance measures. The most widely used distance measure is the Euclidean distance defined as

$$d(\boldsymbol{z}_u, \boldsymbol{z}_w) = \sqrt{\sum_{j=1}^{N_d} (z_{u,j} - z_{w,j})^2} = \left\| \boldsymbol{z}_u - \boldsymbol{z}_w \right\| \tag{3.1}$$

Euclidean distance is a special case (when $\alpha = 2$) of the Minkowski metric [Jain *et al*. 1999] defined as

$$d^{\alpha}(\boldsymbol{z}_u, \boldsymbol{z}_w) = \left(\sum_{j=1}^{N_d} (z_{u,j} - z_{w,j})^{\alpha}\right)^{1/\alpha} = \left\| \boldsymbol{z}_u - \boldsymbol{z}_w \right\|^{\alpha} \tag{3.2}$$

When $\alpha = 1$, the measure is referred to as the Manhattan distance [Hamerly 2003].

Clustering data of high dimensionality using the Minkowski metric is usually not efficient because the distance between the patterns increases with increase in dimensionality. Hence, the concepts of near and far become weaker [Hamerly 2003]. Furthermore, for the Minkowski metric, the largest-scaled feature tends to dominate the other features. This can be solved by normalizing the features to a common range [Jain *et al*. 1999]. One way to do this is by using the cosine distance (or vector dot product) which is the sum of the product of each component from two vectors defined as

$$< \boldsymbol{z}_u , \boldsymbol{z}_w > = \frac{\sum_{j=1}^{N_d} z_{u,j} z_{w,j}}{\left\| \boldsymbol{z}_u \right\| \left\| \boldsymbol{z}_w \right\|} \tag{3.3}$$

where $< \boldsymbol{z}_u , \boldsymbol{z}_w > \in [-1,1]$.

The cosine distance is actually not a distance but rather a similarity metric. In other words, the cosine distance measures the difference in the angle between two vectors not the difference in the magnitude between two vectors. The cosine distance is suitable for clustering data of high dimensionality [Hamerly 2003].

Another distance measure is the Mahalanobis distance defined as

$$d_{\mathrm{M}}(\boldsymbol{z}_u , \boldsymbol{z}_w) = (\boldsymbol{z}_u - \boldsymbol{z}_w) \Sigma^{-1} (\boldsymbol{z}_u - \boldsymbol{z}_w)^{\mathrm{T}} \tag{3.4}$$

where Σ is the covariance matrix of the patterns. The Mahalanobis distance gives different features different weights based on their variances and pairwise linear

correlations. Thus, this metric implicitly assumes that the densities of the classes are multivariate Gaussian [Jain *et al*. 1999].

3.1.3 Clustering Techniques

Most clustering algorithms are based on two popular techniques known as *hierarchical* and *partitional* clustering [Frigui and Krishnapuram 1999; Leung *et al*. 2000]. In the following, an overview of both techniques is presented with an elaborate discussion of popular hierarchical and partitional clustering algorithms.

3.1.3.1 Hierarchical Clustering Techniques

Algorithms in this category generate a cluster tree (or *dendrogram*) by using heuristic splitting or merging techniques [Hamerly 2003]. A cluster tree is defined as "a tree showing a sequence of clustering with each clustering being a partition of the data set" [Leung *et al*. 2000]. Algorithms that use splitting to generate the cluster tree are called *divisive*. On the other hand, the more popular algorithms that use merging to generate the cluster tree are called *agglomerative*. Divisive hierarchical algorithms start with all the patterns assigned to a single cluster. Then, splitting is applied to a cluster in each stage until each cluster consists of one pattern. Contrary to divisive hierarchical algorithms, agglomerative hierarchical algorithms start with each pattern assigned to one cluster. Then, the two most similar clusters are merged together. This step is repeated until all the patterns are assigned to a single cluster [Turi 2001]. Several agglomerative hierarchical algorithms were proposed in the literature which differ in the way that the two most similar clusters are calculated. The two most popular

agglomerative hierarchical algorithms are the *single link* [Sneath and Sokal 1973] and *complete link* [Anderberg 1973] algorithms. Single link algorithms merge the clusters whose distance between their closest patterns is the smallest. Complete link algorithms, on the other hand, merge the clusters whose distance between their most distant patterns is the smallest [Turi 2001]. In general, complete link algorithms generate compact clusters while single link algorithms generate elongated clusters. Thus, complete link algorithms are generally more useful than single link algorithms [Jain *et al*. 1999]. Another less popular agglomerative hierarchical algorithm is the *centroid* method [Anderberg 1973]. The centroid algorithm merges the clusters whose distance between their centroids is the smallest. One disadvantage of the centroid algorithm is that the characteristic of a very small cluster is lost when merged with a very large cluster [Turi 2001]. More details about traditional hierarchical clustering techniques can be found in Everitt [1974].

Recently, a hierarchical clustering approach to simulate the human visual system by modeling the blurring effect of lateral retinal interconnections based on scale space theory has been proposed by Leung *et al*. [2000]. The following paragraph provides the reader with a good idea about this approach as described by Leung *et al*. [2000]:

> "In this approach, a data set is considered as an image with each light point located at a datum position. As we blur this image, smaller light blobs merge into larger ones until the whole image becomes one light blob at a low level of resolution. By identifying each blob with a cluster, the blurring process generates a family of clustering along the hierarchy."

According to Leung *et al*. [2000], this approach has several advantages, including:

- it is not sensitive to initialization,
- it is robust in the presence of noise in the data set, and
- it generates clustering that is similar to that perceived by human eyes.

In general, hierarchical clustering techniques have the following advantages [Frigui and Krishnapuram 1999]:

- the number of clusters need not to be specified *a priori*, and
- they are independent of the initial conditions.

However, hierarchical clustering techniques generally suffer from the following drawbacks:

- They are computationally expensive (time complexity is $O(N_p^2 \log N_p)$ and space complexity is $O(N_p^2)$ [Turi 2001]). Hence, they are not suitable for very large data sets.
- They are static, i.e. patterns assigned to a cluster cannot move to another cluster.
- They may fail to separate overlapping clusters due to a lack of information about the global shape or size of the clusters.

3.1.3.2 Partitional Clustering Techniques

Partitional clustering algorithms divide the data set into a specified number of clusters. These algorithms try to minimize certain criteria (e.g. a square error function) and can therefore be treated as optimization problems. However, these optimization

problems are generally NP-hard and combinatorial [Leung *et al*. 2000]. The advantages of hierarchical algorithms are the disadvantages of the partitional algorithms and *vice versa*. Because of their advantages, partitional clustering techniques are more popular than hierarchical techniques in pattern recognition [Jain *et al*. 2000], hence, this thesis concentrates on partitional techniques.

Partitional clustering algorithms are generally iterative algorithms that converge to local optima [Hamerly and Elkan 2002]. Employing the general form of iterative clustering used by Hamerly and Elkan [2002], the steps of an iterative clustering algorithm are:

1. Randomly initialize the K cluster centroids
2. **Repeat**
   (a) **For** each pattern, $\boldsymbol{z}_p$, in the data set **do**
      Compute its membership $u(\boldsymbol{m}_k \mid \boldsymbol{z}_p)$ to each centroid $\boldsymbol{m}_k$ and its weight $w(\boldsymbol{z}_p)$
   endloop

   (b) Recalculate the K cluster centroids, using

$$\boldsymbol{m}_k = \frac{\sum_{\forall \boldsymbol{z}_p} u(\boldsymbol{m}_k \mid \boldsymbol{z}_p)\, w(\boldsymbol{z}_p)\boldsymbol{z}_p}{\sum_{\forall \boldsymbol{z}_p} u(\boldsymbol{m}_k \mid \boldsymbol{z}_p)\, w(\boldsymbol{z}_p)} \tag{3.5}$$

   until a stopping criterion is satisfied.

In the above algorithm, $u(\boldsymbol{m}_k \mid \boldsymbol{z}_p)$ is the membership function which quantifies the membership of pattern $\boldsymbol{z}_p$ to cluster k. The membership function, $u(\boldsymbol{m}_k \mid \boldsymbol{z}_p)$, must satisfy the following constraints:

1) $u(\boldsymbol{m}_k \mid \boldsymbol{z}_p) \geq 0,\ p = 1,\ldots, N_p$ and $k = 1,\ldots, K$

2) $\sum_{k=1}^{K} u(\boldsymbol{m}_k \mid \boldsymbol{z}_p) = 1,\ p = 1,\ldots, N_p$

Crisp clustering algorithms use a *hard* membership function (i.e. $u(\boldsymbol{m}_k \mid \boldsymbol{z}_p) \in \{0,1\}$), while fuzzy clustering algorithms use a *soft* member function (i.e. $u(\boldsymbol{m}_k \mid \boldsymbol{z}_p) \in [0,1]$) [Hamerly and Elkan 2002].

The weight function, $w(\boldsymbol{z}_p)$, in equation (3.5) defines how much influence pattern $\boldsymbol{z}_p$ has in recomputing the centroids in the next iteration, where $w(\boldsymbol{z}_p) > 0$ [Hamerly and Elkan 2002]. The weight function was proposed by Zhang [2000].

Different stopping criteria can be used in an iterative clustering algorithm, for example:

- stop when the change in centroid values are smaller than a user-specified value,
- stop when the quantization error is small enough, or
- stop when a maximum number of iterations has been exceeded.

In the following, popular iterative clustering algorithms are described by defining the membership and weight functions in equation (3.5).

The K-means Algorithm

The most widely used partitional algorithm is the iterative K-means approach [Forgy 1965]. The objective function that the K-means optimizes is

$$J_{\text{K-means}} = \sum_{k=1}^{K} \sum_{\forall \boldsymbol{z}_p \in \boldsymbol{C}_k} d^2(\boldsymbol{z}_p, \boldsymbol{m}_k) \tag{3.6}$$

Hence, the K-means algorithm minimizes the intra-cluster distance [Hamerly and Elkan 2002]. The K-means algorithm starts with *K* centroids (initial values for the centroids are randomly selected or derived from *a priori* information). Then, each pattern in the data set is assigned to the closest cluster (i.e. closest centroid). Finally, the centroids are recalculated according to the associated patterns. This process is repeated until convergence is achieved.

The membership and weight functions for K-means are defined as

$$u(\boldsymbol{m}_k \mid \boldsymbol{z}_p) = \begin{cases} 1 & \text{if } d^2(\boldsymbol{z}_p, \boldsymbol{m}_k) = \arg\min_k \left\{ d^2(\boldsymbol{z}_p, \boldsymbol{m}_k) \right\} \\ 0 & \text{otherwise} \end{cases} \tag{3.7}$$

$$w(\boldsymbol{z}_p) = 1 \tag{3.8}$$

Hence, K-means has a hard membership function. Furthermore, K-means has a constant weight function, thus, all patterns have equal importance [Hamerly and Elkan 2002].

The K-means algorithm has the following main advantages [Turi 2001]:

- it is very easy to implement, and
- its time complexity is O(N_p) making it suitable for very large data sets.

However, the K-means algorithm has the following drawbacks [Davies 1997]:

- the algorithm is data-dependent,

- it is a greedy algorithm that depends on the initial conditions, which may cause the algorithm to converge to suboptimal solutions, and
- the user needs to specify the number of clusters in advance.

The K-medoids algorithm is similar to K-means with one major difference, namely, the centroids are taken from the data itself [Hamerly 2003]. The objective of K-medoids is to find the most centrally located patterns within the clusters [Halkidi *et al.* 2001]. These patterns are called *medoids*. Finding a single medoid requires $O(N_p^2)$. Hence, K-medoids is not suitable for moderately large data sets.

The Fuzzy C-means Algorithm

A fuzzy version of K-means, called Fuzzy C-means (FCM) (sometimes called fuzzy K-means), was proposed by Bezdek [1980; 1981]. FCM is based on a fuzzy extension of the least-square error criterion. The advantage of FCM over K-means is that FCM assigns each pattern to each cluster with some degree of membership (i.e. fuzzy clustering). This is more suitable for real applications where there are some overlaps between the clusters in the data set. The objective function that the FCM optimizes is

$$J_{\text{FCM}} = \sum_{k=1}^{K}\sum_{p=1}^{N_p} u_{k,p}^{q} d^2(\boldsymbol{z}_p, \boldsymbol{m}_k) \tag{3.9}$$

where q is the fuzziness exponent, with $q \geq 1$. Increasing the value of q will make the algorithm more fuzzy; $u_{k,p}$ is the membership value for the p^{th} pattern in the k^{th} cluster satisfying the following constraints:

1) $u_{k,p} \geq 0$, $p = 1,\ldots, N_p$ and $k = 1,\ldots, K$

2) $\sum_{k=1}^{K} u_{k,p} = 1,\ p = 1,\ldots, N_p$

The membership and weight functions for FCM are defined as [Hamerly and Elkan 2002]

$$u(\boldsymbol{m}_k \mid \boldsymbol{z}_p) = \frac{\left\|\boldsymbol{z}_p - \boldsymbol{m}_k\right\|^{-2/(q-1)}}{\sum_{k=1}^{K}\left\|\boldsymbol{z}_p - \boldsymbol{m}_k\right\|^{-2/(q-1)}} \tag{3.10}$$

$$w(\boldsymbol{z}_p) = 1 \tag{3.11}$$

Hence, FCM has a soft membership function and a constant weight function. In general, FCM performs better than K-means [Hamerly 2003] and it is less affected by the presence of uncertainty in the data [Liew *et al*. 2000]. However, as in K-means it requires the user to specify the number of clusters in the data set. In addition, it may converge to local optima [Jain *et al*. 1999].

Krishnapuram and Keller [1993; 1996] proposed a possibilistic clustering algorithm, called *possibilistic* C-means. Possibilistic clustering is similar to fuzzy clustering; the main difference is that in possibilistic clustering the membership values may not sum to one [Turi 2001]. Possibilistic C-means works well in the presence of noise in the data set. However, it has several drawbacks, namely [Turi 2001],

- it is likely to generate coincident clusters,
- it requires the user to specify the number of clusters in advance,
- it converges to local optima, and
- it depends on initial conditions.

The Gaussian Expectation-Maximization Algorithm

Another popular clustering algorithm is the Expectation-Maximization (EM) algorithm [McLachlan and Krishnan 1997; Rendner and Walker 1984; Bishop 1995]. EM is used for parameter estimation in the presence of some unknown data [Hamerly 2003]. EM partitions the data set into clusters by determining a mixture of Gaussians fitting the data set. Each Gaussian has a mean and covariance matrix [Alldrin *et al*. 2003]. The objective function that the EM optimizes as defined by Hamerly and Elkan [2002] is

$$J_{\mathrm{EM}} = -\sum_{p=1}^{N_p} \log\left(\sum_{k=1}^{K} p(\boldsymbol{z}_p \mid \boldsymbol{m}_k)\, p(\boldsymbol{m}_k)\right) \tag{3.12}$$

where $p(\boldsymbol{z}_p \mid \boldsymbol{m}_k)$ is the probability of $\boldsymbol{z}_p$ given that it is generated by a Gaussian distribution with centroid $\boldsymbol{m}_k$, and $p(\boldsymbol{m}_k)$ is the prior probability of centroid $\boldsymbol{m}_k$.

The membership and weight functions for EM are defined as [Hamerly and Elkan 2002]

$$u(\boldsymbol{m}_k \mid \boldsymbol{z}_p) = \frac{p(\boldsymbol{z}_p \mid \boldsymbol{m}_k)\, p(\boldsymbol{m}_k)}{p(\boldsymbol{z}_p)} \tag{3.13}$$

$$w(\boldsymbol{z}_p) = 1 \tag{3.14}$$

Hence, EM has a soft membership function and a constant weight function. The algorithm starts with an initial estimate of the parameters. Then, an *expectation* step is applied where the known data values are used to compute the expected values of the unknown data [Hamerly 2003]. This is followed by a *maximization* step where the

known and expected values of the data are used to generate a new estimate of the parameters. The expectation and maximization steps are repeated until convergence.

Results from Veenman *et al*. [2002] and Hamerly [2003] showed that K-means performs comparably to EM. Furthermore, Aldrin *et al*. [2003] stated that EM fails on high-dimensional data sets due to numerical precision problems. They also observed that Gaussians often collapsed to delta functions [Alldrin *et al*. 2003]. In addition, EM depends on the initial estimate of the parameters [Hamerly 2003; Turi 2001] and it requires the user to specify the number of clusters in advance. Moreover, EM assumes that the density of each cluster is Gaussian which may not always be true [Ng *et al*. 2001].

The K-harmonic Means Algorithm

Recently, Zhang and colleagues [1999; 2000] proposed a novel algorithm called K-harmonic means (KHM), with promising results. In KHM, the harmonic mean of the distance of each cluster center to every pattern is computed. The cluster centroids are then updated accordingly. The objective function that the KHM optimizes is

$$J_{\text{KHM}} = \sum_{p=1}^{N_p} \frac{K}{\sum_{k=1}^{K} \frac{1}{\left\| \boldsymbol{z}_p - \boldsymbol{m}_k \right\|^{\alpha}}} \tag{3.15}$$

where α is a user-specified parameter, typically $\alpha \geq 2$.

The membership and weight functions for KHM are [Hamerly and Elkan 2002]

$$u(\boldsymbol{m}_k \mid \boldsymbol{z}_p) = \frac{\left\|\boldsymbol{z}_p - \boldsymbol{m}_k\right\|^{-\alpha-2}}{\sum_{k=1}^{K}\left\|\boldsymbol{z}_p - \boldsymbol{m}_k\right\|^{-\alpha-2}} \tag{3.16}$$

$$w(\boldsymbol{z}_p) = \frac{\sum_{k=1}^{K}\left\|\boldsymbol{z}_p - \boldsymbol{m}_k\right\|^{-\alpha-2}}{\left(\sum_{k=1}^{K}\left\|\boldsymbol{z}_p - \boldsymbol{m}_k\right\|^{-\alpha}\right)^2} \tag{3.17}$$

Hence, KHM has a soft membership function and a varying weight function. KHM assigns higher weights for patterns that are far from all the centroids to help the centroids in covering the data [Hamerly and Elkan 2002].

Contrary to K-means, KHM is less sensitive to initial conditions and does not have the problem of collapsing Gaussians exhibited by EM [Alldrin *et al*. 2003]. Experiments conducted by Zhang *et al*. [1999], Zhang [2000] and Hamerly and Elkan [2002] showed that KHM outperformed K-means, FCM (according to Hamerly and Elkan [2002]) and EM.

Hybrid 2

Hamerly and Elkan [2002] proposed a variation of KHM, called Hybrid 2 (H2), which uses the soft membership function of KHM (i.e. equation (3.16)) and the constant weight function of K-means (i.e. equation (3.8)). Hamerly and Elkan [2002] showed that H2 outperformed K-means, FCM and EM. However, KHM, in general, performed slightly better than H2.

K-means, FCM, EM, KHM and H2 are linear time algorithms (i.e. their time complexity is O(N_p)) making them suitable for very large data sets. According to

Hamerly [2003], FCM, KHM and H2 - all use soft membership functions - are the best available clustering algorithms.

Non-iterative Partitional Algorithms

Another category of unsupervised partitional algorithms includes the non-iterative algorithms. The most widely used non-iterative algorithm is MacQueen's K-means algorithm [MacQueen 1967]. This algorithm works in two phases: the first phase finds the centroids of the clusters, and the second clusters the patterns. Competitive Learning (CL) updates the centroids sequentially by moving the closest centroid toward the pattern being classified [Scheunders, A Comparison 1997]. These algorithms suffer the drawback of being dependent on the order in which the data points are presented. To overcome this problem, data points are presented in a random order [Davies 1997]. In general, iterative algorithms are more effective than non-iterative algorithms, since they are less dependent on the order in which data points are presented.

3.1.3.3 Other Clustering Techniques

Another type of clustering algorithms includes the *Nearest Neighbor* clustering algorithm proposed by Lu and Fu [1978]. For each unclassified pattern, the algorithm finds the nearest classified pattern whose distance from the unclassified pattern is less than a pre-specified threshold. The unclassified pattern is then assigned to the cluster of the classified pattern. This process is repeated until all the patterns become classified or no further assignments can occur [Jain *et al*. 1999].

Recently, a new type of clustering algorithms called *spectral* clustering algorithms [Ng *et al*. 2001; Bach and Jordan 2003] has been proposed by computer vision researchers and graph theorists. Spectral clustering is based on spectral graph theory [Chung 1997] where a graph representing the data (the graph is analogous to a matrix of the distance between the patterns in the data set) is searched by the spectral clustering algorithm for globally optimal cuts [Hamerly 2003]. One major advantage of spectral clustering is that it can generate arbitrary-shaped clusters. However, spectral clustering suffers from two major drawbacks [Hamerly 2003]:

- It is computationally expensive (its time complexity is $O(N_p^3 + N_d N_p^2)$). Hence, they are not suitable for moderately large data sets.
- It requires the user to specify a kernel width parameter which has a profound effect on the result of the spectral clustering algorithm. Choosing a good value for this parameter is usually difficult.

The *mean shift* algorithm [Comaniciu and Meer 2002] also automatically finds the number of clusters in a data set and can work with arbitrary shaped clusters. The mean shift algorithm starts with a number of kernel estimators in the input space. These estimators are then repeatedly moved towards areas of higher density. When all the kernels reached stability, all the kernels that are near to each other are grouped together. The data is then segmented based on where each kernel started.

The mean shift algorithm has the following problems, [Hamerly 2003]:

- it has to find a way to group kernels and patterns, and

- as in spectral clustering, the mean shift algorithm requires the user to specify a kernel width parameter which has a profound effect on the result of the algorithm.

3.1.4 Clustering Validation Techniques

The main objective of cluster validation is to evaluate clustering results in order to find the best partitiong of a data set [Halkidi *et al*. 2001]. Hence, cluster validity approaches are used to quantitatively evaluate the result of a clustering algorithm [Halkidi *et al*. 2001]. These approaches have representative indices, called *validity indices*. The traditional approach to determine the "optimum" number of clusters is to run the algorithm repetitively using different input values and to select the partitioning of data resulting in the best validity measure [Halkidi and Vazirgiannis 2001].

Two criteria that have been widely considered sufficient in measuring the quality of data partitioning, are [Halkidi *et al*. 2001]

- *Compactness*: patterns in one cluster should be similar to each other and different from patterns in other clusters. The variance of patterns in a cluster gives an indication of compactness.
- *Separation*: clusters should be well-separated from each other. The Euclidean distance between cluster centroids gives an indication of cluster separation.

There are several validity indices; a thorough survey of validity indices can be found in Halkidi *et al*. [2001]. In the following, some representative indices are discussed.

Dunn [1974] proposed a well known cluster validity index that identifies compact and well separated clusters. The main goal of Dunn's index is to maximize

inter-cluster distances (i.e. separation) while minimizing intra-cluster distances (i.e. increase compactness). The Dunn index is defined as

$$D = \min_{k=1,\ldots,K} \left\{ \min_{kk=k+1,\ldots,K} \left(\frac{dist(\boldsymbol{C}_k, \boldsymbol{C}_{kk})}{\max_{a=1,\ldots,K} diam(\boldsymbol{C}_a)} \right) \right\} \tag{3.18}$$

where $dist(\boldsymbol{C}_k, \boldsymbol{C}_{kk})$ is the dissimilarity function between two clusters $\boldsymbol{C}_k$ and $\boldsymbol{C}_{kk}$ defined as

$$dist(\boldsymbol{C}_k, \boldsymbol{C}_{kk}) = \min_{\boldsymbol{u} \in \boldsymbol{C}_k, \boldsymbol{w} \in \boldsymbol{C}_{kk}} d(\boldsymbol{u}, \boldsymbol{w}),$$

where $d(\boldsymbol{u}, \boldsymbol{w})$ is the Euclidean distance between $\boldsymbol{u}$ and $\boldsymbol{v}$; $diam(\boldsymbol{C})$ is the diameter of a cluster, defined as

$$diam(\boldsymbol{C}) = \max_{\boldsymbol{u}, \boldsymbol{w} \in \boldsymbol{C}} d(\boldsymbol{u}, \boldsymbol{w})$$

An "optimal" value of K is the one that maximizes the Dunn's index. Dunn's index suffers from the following problems [Halkidi *et al*. 2001]:

- it is computationally expensive, and
- it is sensitive to the presence of noise.

Several Dunn-like indices were proposed in Pal and Biswas [1997] to reduce the sensitivity to the presence of noise.

Another well known index, proposed by Davies and Bouldin [1979], minimizes the average similarity between each cluster and the one most similar to it. The Davies and Bouldin index is defined as

$$DB = \frac{1}{K}\sum_{k=1}^{K} \max_{\substack{kk=1,\ldots,K \\ k\neq kk}} \left(\frac{diam(\boldsymbol{C}_k) + diam(\boldsymbol{C}_{kk})}{dist(\boldsymbol{C}_k, \boldsymbol{C}_{kk})} \right) \tag{3.19}$$

An "optimal" value of *K* is the one that minimizes the *DB* index.

Recently, Turi [2001] proposed an index incorporating a multiplier function (to penalize the selection of a small number of clusters) to the ratio between intra-cluster and inter-cluster distances, with some promising results. The index is defined as

$$V = (c \times \mathrm{N}(2,1) + 1) \times \frac{\text{intra}}{\text{inter}} \tag{3.20}$$

where *c* is a user specified parameter and N(2,1) is a Gaussian distribution with mean 2 and standard deviation of 1. The "intra" term is the average of all the distances between each data point and its cluster centroid, defined as

$$\text{intra} = \frac{1}{N_p}\sum_{k=1}^{K}\sum_{\forall \boldsymbol{u}\in \boldsymbol{C}_k} \left\| \boldsymbol{u} - \boldsymbol{m}_k \right\|^2$$

This term is used to measure the compactness of the clusters. The "inter" term is the minimum distance between the cluster centroids, defined as

$$\text{inter} = min\{\left\| \boldsymbol{m}_k - \boldsymbol{m}_{kk} \right\|^2\}, \forall\, k = 1,\ldots,K-1 \text{ and } kk = k+1,\ldots,K.$$

This term is used to measure the separation of the clusters. An "optimal" value of *K* is the one that minimizes the *V* index.

According to Turi [2001], this index performed better than both Dunn's index and the index of Davies and Bouldin on the tested cases.

Two recent validity indices are *S_Dbw* [Halkidi and Vazirgiannis 2001] and *CDbw* [Halkidi and Vazirgiannis 2002]. *S_Dbw* measures the compactness of a data

set by the cluster variance, whereas separation is measured by the density between clusters. The *S_Dbw* index is defined as

$$S\_Dbw = scat(K) + Dens\_bw(K) \tag{3.21}$$

The first term is the average scattering of the clusters which is a measure of compactness of the clusters, defined as

$$scat(K) = \frac{1}{K}\sum_{k=1}^{K} \|\sigma(\boldsymbol{C}_k)\| / \|\sigma(\boldsymbol{Z})\|$$

where $\sigma(\boldsymbol{C}_k)$ is the variance of cluster $\boldsymbol{C}_k$ and $\sigma(\boldsymbol{Z})$ is the variance of data set $\boldsymbol{Z}$; $\|\boldsymbol{z}\|$ is defined as $\|\boldsymbol{z}\| = (\boldsymbol{z}^{\mathrm{T}}\boldsymbol{z})^{1/2}$, where $\boldsymbol{z}$ is a vector.

The second term in equation (3.21) evaluates the density of the area between the two clusters in relation to the density of the two clusters. Thus, the second term is a measure of the separation of the clusters, defined as

$$Dens\_bw(K) = \frac{1}{K(K-1)}\sum_{k=1}^{K}\left[\sum_{\substack{k\,k=1 \\ k\neq k\,k}}^{K} \frac{density(\boldsymbol{b}_{k,kk})}{max\{density(\boldsymbol{C}_k), density(\boldsymbol{C}_{kk})\}}\right]$$

where $\boldsymbol{b}_{k,kk}$ is the middle point of the line segment defined by $\boldsymbol{m}_k$ and $\boldsymbol{m}_{kk}$. The term *density*($\boldsymbol{b}$) is defined as

$$density(\boldsymbol{b}) = \sum_{ll=1}^{n_{k,kk}} f(\boldsymbol{z}_{ll}, \boldsymbol{b})$$

where $n_{k,kk}$ is the total number of patterns in clusters $\boldsymbol{C}_k$ and $\boldsymbol{C}_{kk}$ (i.e. $n_{k,kk} = n_k + n_{kk}$). The function $f(\boldsymbol{z},\boldsymbol{b})$ is defined as

$$f(\boldsymbol{z},\boldsymbol{b}) = \begin{cases} 0 & \text{if } d(\boldsymbol{z},\boldsymbol{b}) > \sigma \\ 1 & \text{otherwise} \end{cases}$$

where

$$\sigma = \frac{1}{K}\sqrt{\sum_{k=1}^{K}\left\|\sigma(\boldsymbol{C}_k)\right\|}$$

An "optimal" value of *K* is the one that minimizes the *S_Dbw* index. Halkidi and Vazirgiannis [2001] showed that, in tested cases, *S_Dbw* successfully found the "optimal" number of clusters whereas other well-known indices often failed to do so. However, *S_Dbw* does not work properly for arbitrary shaped clusters.

To address this problem, Halkidi and Vazirgiannis [2002] proposed a multi-representative validity index, *CDbw*, in which each cluster is represented by a user-specified number of points, instead of one representative as is done in *S_Dbw*. Furthermore, *CDbw* uses intra-cluster density to measure the compactness of a data set, and uses the density between clusters to measure their separation.

More recently, Veenman *et al*. [2002; 2003] proposed a validity index that minimizes the intra-cluster variability while constraining the intra-cluster variability of the union of the two clusters. The sum of squared error is used to minimize the intra-cluster variability while a minimum variance for the union of two clusters is used to implement the joint intra-cluster variability. The index is defined as

$$IV = min\sum_{k=1}^{K} n_k Var(\boldsymbol{C}_k) \tag{3.22}$$

where n_k is the number of patterns in cluster $\boldsymbol{C}_k$ and

$$Var(\boldsymbol{C}_k) = \frac{1}{n_k}\sum_{z_p \in \boldsymbol{C}_k}\left\|\boldsymbol{z}_p - \boldsymbol{m}_k\right\|^2$$

such that

$$Var(\boldsymbol{C}_k \cup \boldsymbol{C}_{kk}) \geq \sigma^2_{\max}, \quad \forall \boldsymbol{C}_k, \boldsymbol{C}_{kk}, k \neq kk$$

where $\sigma^2_{\max}$ is a user-specified parameter. This parameter has a profound effect on the final result.

The above validity indices are suitable for hard clustering. Validity indices have been developed for fuzzy clustering. The interested reader is referred to Halkidi *et al*. [2001] for more information.

3.1.5 Determining the Number of Clusters

Most clustering algorithms require the number of clusters to be specified in advance [Lee and Antonsson 2000; Hamerly and Elkan 2003]. Finding the "optimum" number of clusters in a data set is usually a challenge since it requires *a priori* knowledge, and/or ground truth about the data, which is not always available. The problem of finding the optimum number of clusters in a data set has been the subject of several research efforts [Halkidi *et al*. 2001; Theodoridis and Koutroubas 1999], however, despite the amount of research in this area, the outcome is still unsatisfactory [Rosenberger and Chehdi 2000]. In the literature, many approaches to dynamically find the number of clusters in a data set were proposed. In this section, several dynamic clustering approaches are presented and discussed.

ISODATA (Iterative Self-Organizing Data Analysis Technique), proposed by Ball and Hall [1967], is an enhancement of the K-means algorithm (K-means is sometimes referred to as *basic* ISODATA [Turi 2001]). ISODATA is an iterative procedure that assigns each pattern to its closest centroids (as in K-means). However, ISODATA has the ability to merge two clusters if the distance between their centroids is below a user-specified threshold. Furthermore, ISODATA can split elongated clusters into two clusters based on another user-specified threshold. Hence, a major advantage of ISODATA compared to K-means is the ability to determine the number of clusters in a data set. However, ISODATA requires the user to specify the values of

several parameters (e.g. the merging and splitting thresholds). These parameters have a profound effect on the performance of ISODATA making the result subjective [Turi 2001].

Dynamic Optimal Cluster-seek (DYNOC) [Tou 1979] is a dynamic clustering algorithm which is similar to ISODATA. DYNOC maximizes the ratio of the minimum inter-cluster distance to the maximum intra-cluster distance. This is done by an iterative procedure with the added capability of splitting and merging. However, as in ISODATA, DYNOC requires the user to specify a value for a parameter that determines whether splitting is needed [Turi 2001].

Snob [Wallace 1984; Wallace and Dowe 1994] uses various methods to assign objects to clusters in an intelligent manner [Turi 2001]. After each assignment, a means of model selection called the Wallace Information Measure (also known as the Minimum Message Length) [Wallace and Boulton 1968; Oliver and Hand 1994] is calculated and based on this calculation the assignment is accepted or rejected. Snob can split/merge and move points between clusters, thereby allowing it to determine the number of clusters in a data set.

Bischof *et al*. [1999] proposed an algorithm based on K-means which uses a similar concept to the Wallace Information Measure called the Minimum Description Length [Rissanen 1978] framework. The algorithm starts with a large value for K and proceeds to remove centroids when this removal results in a reduction of the description length. K-means is used between the steps that reduce K.

Modified Linde-Buzo-Gray (MLBG), proposed by Rosenberger and Chehdi [2000], improves K-means by automatically finding the number of clusters in data set by using intermediate results. MLBG is an iterative procedure that starts with K clusters. In each iteration, a cluster, $\boldsymbol{C}_k$, maximizing an intra-cluster distance measure

is chosen for splitting. Two centroids are generated from the splitting process. The first centroid, $\boldsymbol{m}_1$, is initialized to the centroid of the original cluster, $\boldsymbol{C}_k$. The second cluster centroid, $\boldsymbol{m}_2$, is chosen to be the pattern in $\boldsymbol{C}_k$ which is the most distant from $\boldsymbol{m}_1$. K-means is then applied on the new K+1 centroids. The new set of centroids is accepted if it satisfies an evaluation criterion based on a dispersion measure. This process is repeated until no valid partition of the data can be obtained. One of the main problems with MLBG is that it requires the user to specify the values of four parameters, which have a profound effect on the resultant number of clusters.

Pelleg and Moore [2000] proposed another K-means based algorithm, called X-means that uses model selection. X-means starts by setting the number of clusters, K, to be the minimum number of clusters in the data set (e.g. K = 1). Then, K-means is applied on the K clusters. This is followed by a splitting process based on the Bayesian Information Criterion (BIC) [Kass and Wasserman 1995] defined as

$$BIC(\boldsymbol{C} \mid \boldsymbol{Z}) = \hat{l}(\boldsymbol{Z} \mid \boldsymbol{C}) - \frac{K(N_d + 1)}{2} \log N_p \qquad (3.23)$$

where $\hat{l}(\boldsymbol{Z} \mid \boldsymbol{C})$ is the log-likelihood of the data set $\boldsymbol{Z}$ according to model $\boldsymbol{C}$. If the splitting process improves the BIC score the resulting split is accepted, otherwise it is rejected. Other scoring functions can also be used.

These two steps are repeated until a user-specified upper bound of K is reached. X-means searches over the range of values of K and reports the value with the best BIC score.

Recently, Huang [2002] proposed SYNERACT as an alternative approach to ISODATA. SYNERACT combines K-means with hierarchical descending approaches

to overcome the drawbacks of K-means mentioned previously. Three concepts used by SYNERACT are:

- a hyperplane to split up a cluster into two smaller clusters and compute their centroids,
- iterative clustering to assign pixels into available clusters, and
- a binary tree to store clusters generated from the splitting process.

According to Huang [2002], SYNERACT is faster than and almost as accurate as ISODATA. Furthermore, it does not require the number of clusters and initial location of centroids to be specified in advance. However, SYNERACT requires the user to specify the values of two parameters that affect the splitting process.

Veenman *et al*. [2002] proposed a partitional clustering algorithm that finds the number of clusters in a data set by minimizing the clustering validity index defined in equation (3.22). This algorithm starts by initializing the number of clusters equal to the number of patterns in the data set. Then, iteratively, the clusters are split or merged according to a series of tests based on the validity index. According to Veenman *et al*. [2002], the proposed approach performed better than both K-means and EM algorithms. However, the approach suffers from the following drawbacks, namely

- it is computationally expensive, and
- it requires the user to specify a parameter for the validity index (already discussed in Section 3.1.4) which has a significant effect on the final results (although the authors provide a method to help the user in finding a good value for this parameter).

More recently, Hamerly and Elkan [2003] proposed another approach based on K-means, called G-means. G-means starts with a small value for *K*, and with each iteration splits up the clusters whose data do not fit a Gaussian distribution. Between each round of splitting, K-means is applied to the entire data set in order to refine the current solution. According to Hamerly and Elkan [2003], G-means works better than X-means, however, it works only for data having spherical and/or elliptical clusters. G-means is not designed to work for arbitrary-shaped clusters [Hamerly 2003].

Gath and Geva [1989] proposed an unsupervised fuzzy clustering algorithm based on a combination of FCM and fuzzy maximum likelihood estimation. The algorithm starts by initializing *K* to a user-specified lower bound of the number of clusters in the data set (e.g. $K = 1$). A modified FCM (that uses an unsupervised learning process to initialize the *K* centroids) is first applied to cluster the data. Using the resulting centroids, a fuzzy maximum likelihood estimation algorithm is then applied. The fuzzy maximum likelihood estimation algorithm uses an "exponential" distance measure based on maximum likelihood estimation [Bezdek 1981] instead of the Euclidean distance measure, because the exponential distance measure is more suitable for hyper-ellipsoidal clusters. The quality of the resulting clusters is then evaluated using a clustering validity index that is mainly based on a hyper-volume criterion which measures the compactness of a cluster. *K* is then incremented and the algorithm is repeated until a user-specified upper bound of *K* is reached. The value of *K* resulting in the best value of the validity index is considered to be the "optimal" number of clusters in the data set. Gath and Geva [1989] stated that their algorithm works well in cases of large variability of cluster shapes. However, the algorithm becomes more sensitive to local optima as the complexity increases. Furthermore, because of the exponential function, floating point overflows may occur [Su 2002].

Lorette *et al.* [2000] proposed an algorithm based on fuzzy clustering to dynamically determine the number of clusters in a data set. In this thesis, the proposed algorithm is referred as the Unsupervised Fuzzy Clustering (UFC) algorithm. A new objective function was proposed for this purpose, defined as

$$J_{\text{UFC}} = \sum_{k=1}^{K}\sum_{p=1}^{N_p} u_{k,p}^{q} d^2(\boldsymbol{z}_p, \boldsymbol{m}_k) - \beta \sum_{k=1}^{K} p_k \log(p_k) \tag{3.24}$$

where q is the fuzziness exponent, $u_{k,p}$ is the membership value for the p^{th} pattern in the k^{th} cluster, β is a parameter that decreases as the run progresses, and p_k is the *a priori* probability of cluster $\boldsymbol{C}_k$ defined as

$$p_k = \frac{1}{N_p}\sum_{p=1}^{N_p} u_{k,p} \tag{3.25}$$

The first term of equation (3.24) is the objective function of FCM which is minimized when each cluster consists of one pattern. The second term is an entropy term that is minimized when all the patterns are assigned to one cluster. Lorette *et al.* [2000] use this objective function to derive new update equations for the membership and centroid parameters.

The algorithm starts with a large number of clusters. Then, the membership values and centroids are updated using the new update equations. This is followed by applying equation (3.25) to update the *a priori* probabilities. If $p_k < \varepsilon$ then cluster k is discarded; ε is a user-specified parameter. This procedure is repeated until

convergence. The drawback of this approach is that it requires the parameter ε to be specified in advance. The performance of the algorithm is sensitive to the value of ε.

Similar to UFC, Boujemaa [2000] proposed an algorithm, based on a generalization of the competitive agglomeration clustering algorithm introduced by Frigui and Krishnapuram [1997].

The fuzzy algorithms discussed above modify the objective function of FCM. In general, these approaches are sensitive to initialization and other parameters [Frigui and Krishnapuram 1999]. Frigui and Krishnapuram [1999] proposed a robust competitive clustering algorithm based on the process of competitive agglomeration. The algorithm starts with a large number of small clusters. Then, during the execution of the algorithm, adjacent clusters compete for patterns. Clusters losing the competition will eventually disappear [Frigui and Krishnapuram 1999]. However, this algorithm also requires the user to specify a parameter that has a significant effect on the generated result.

3.1.6 Clustering using Self-Organizing Maps

Kohonen's Self Organizing Maps (SOM) [Kohonen 1995] can be used to automatically find the number of clusters in a data set. The objective of SOM is to find regularities in a data set without any external supervision [Pandya and Macy 1996]. SOM is a single-layered unsupervised artificial neural network where input patterns are associated with output nodes via weights that are iteratively modified until a stopping criterion is met [Jain *et al*. 1999]. SOM combines competitive learning (in which different nodes in the Kohonen network compete to be the winner when an input pattern is presented) with a topological structuring of nodes, such that adjacent nodes tend to have similar weight vectors (this is done via lateral feedback)

[Mehrotra *et al*. 1997; Pandya and Macy 1996]. A general pseudo-code of SOM [Pandya and Macy 1996] is shown in Figure 3.1.

Let $\eta(t)$ be the learning rate parameter and $\Delta_w(t)$ be the neighborhood function

Randomly initialize the weight vectors, $\boldsymbol{w}_k(0)$

Initialize the learning rate $\eta(0)$ and the neighborhood function $\Delta_w(0)$

Repeat

For each input pattern $\boldsymbol{z}_p$ **do**

Select the node whose weight vector is closest (in terms of Euclidean distance) to $\boldsymbol{z}_p$ as the winning node

Use competitive learning to train the weight vectors such that all the nodes within the neighborhood of the winning node are moved toward $\boldsymbol{z}_p$:

$$\boldsymbol{w}_k(t+1) = \begin{cases} \boldsymbol{w}_k(t) + \eta(t)[\boldsymbol{z}_p - \boldsymbol{w}_k(t)] & k \in \Delta_w(t) \\ \boldsymbol{w}_k(t) & \text{otherwise} \end{cases}$$

Endloop

Linearly decrease $\eta(t)$ and reduce $\Delta_w(t)$

Until some convergence criteria are satisfied

Figure 3.1: General pseudo-code for SOM

In Figure 3.1, $\eta(t)$ starts relatively large (e.g. close to 1) then linearly decreases until it reaches a small user-specified value. The neighborhood function $\Delta_w(t)$ defines the neighborhood size surrounding the winning node. A large value of $\Delta_w(t)$ is used at the beginning of the training. This value is then reduced as the training progresses in

order to get sharper clusters [Pandya and Macy 1996]. A typical neighborhood arrangement is the rectangular lattice shown in Figure 3.2 [Pandya and Macy 1996].

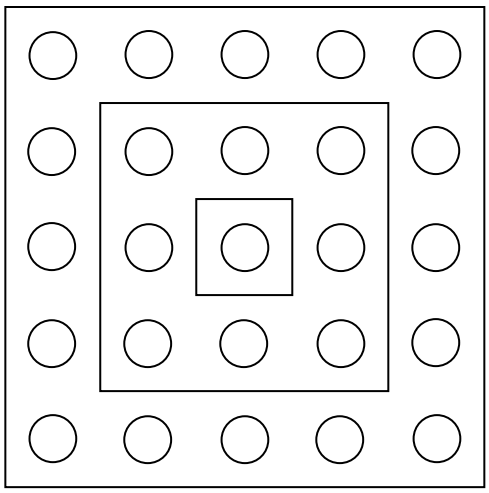


Figure 3.2: Rectangular Lattice arrangement of neighborhoods

SOM suffers from the following drawbacks [Jain *et al*. 1999]:

- It depends on the initial conditions.
- Its performance is affected by the learning rate parameter and the neighborhood function.
- It works well with hyper-spherical clusters only.
- It uses a fixed number of output nodes.
- It depends on the order in which the data points are presented. To overcome this problem, the choice of data points can be randomized during each iteration [Pandya and Macy 1996].

3.1.7 Clustering using Stochastic Algorithms

Simulated annealing (discussed in Section 2.3) has been used for clustering [Klein and Dubes 1989]. In general, a simulated annealing based clustering algorithm works as shown in Figure 3.3 [Jain *et al*. 1999].

An initial partition P_0 of the data set is randomly chosen

Repeat

A neighbor of P_0 is chosen

If the new partition is better than P_0 **then**

move to the new partition

Else

move to the new partition with a probability that decreases as the algorithm progresses.

Until a stopping criterion is satisfied

Figure 3.3: General simulated annealing based clustering algorithm

One problem with simulated annealing is that it is very slow in finding an optimal solution [Jain *et al*. 1999].

Tabu search (discussed in Section 2.3) has also been used for hard clustering [Al-Sultan 1995] and fuzzy clustering [Delgado *et al*. 1997] with encouraging results. A hybrid approach combining both K-means and tabu search that performs better than both K-means and tabu search was proposed by Frnti *et al*. [1998]. Recently, Chu and Roddick [2003] proposed a hybrid approach combining both tabu search and simulated annealing that outperforms the hybrid proposed by Frnti *et al*. [1998].

However, the performance of simulated annealing and tabu search depends on the selection of several control parameters [Jain *et al*. 1999].

Most clustering approaches discussed so far perform local search to find a solution to a clustering problem. Evolutionary algorithms (discussed in Section 2.4) which perform global search have also been used for clustering [Jain *et al*. 1999]. Raghavan and Birchand [1979] used GAs to minimize the squared error of a clustering solution. In this approach, each chromosome represents a partition of N_p patterns into K clusters. Hence, the size of each chromosome is N_p. This representation has a major drawback in that it increases the search space by a factor of $K!$. The crossover operator may also result in inferior offspring [Jain *et al*. 1999].

Babu and Murty [1993] proposed a hybrid approach combining K-means and GAs that performed better than the GA. In this approach, a GA is only used to feed K-means with good initial centroids [Jain *et al*. 1999].

Recently, Maulik and Bandyopadhyay [2000] proposed a GA-based clustering where each chromosome represents K centroids. Hence, a floating point representation is used. The fitness function is defined as the inverse of the objective function of K-means (refer to equation (3.6)). The GA-based clustering algorithm is summarized in Figure 3.4.

According to Maulik and Bandyopadhyay [2000], this approach outperformed K-means on the tested cases. One drawback of this approach is that it requires the user to specify the number of clusters in advance.

1. Initialize each chromosome to contain K randomly chosen centroids from the data set
2. For $t = 1$ to t_{max}

(a) For each chromosome i

(i) Assign each pattern to the cluster with the closest centroid

(ii) Recalculate the K cluster centroids of chromosome i as the means of their patterns

(iii) Calculate the fitness of chromosome i

(b) Apply roulette wheel selection

(c) Apply single point crossover with probability p_c

(d) Apply mutation with probability p_m. The mutation operator is defined as

$$\boldsymbol{x} = \boldsymbol{x} \pm (r + \gamma)\boldsymbol{x}$$

where $r \sim U(0,1)$ and γ is a user-specified parameter such that $\gamma \in (0,1)$

Figure 3.4: General pseudo-code for GA-based clustering algorithm

Lee and Antonsson [2000] used an evolution strategy (ES) to dynamically cluster a data set. The proposed ES implemented variable length individuals to search for both the centroids and the number of clusters. Each individual represents a set of centroids. The length of each individual is randomly chosen from a user-specified range of cluster numbers. The centroids of each individual are then randomly initialized. Mutation is applied to the individuals by adding/subtracting a Gaussian random variable with zero mean and unit standard deviation. Two point crossover is also used as a "length changing operator". A (10+60) ES selection is used where 10 is the

number of parents and 60 is the number of offspring generated in each generation. The best ten individuals from the set of parents and offspring are used for the next generation. A modification of the mean square error is used as the fitness function, defined as

$$J_{\text{ES}} = \sqrt{K+1}\sum_{k=1}^{K}\sum_{\forall \boldsymbol{z}_p \in C_k} d(\boldsymbol{z}_p, \boldsymbol{m}_k) \tag{3.26}$$

The modification occurs by multiplying the mean square error by a constant corresponding to the square root of the number of clusters. This constant is used to penalize a large value of *K*. According to Lee and Antonsson [2000], the results are promising. However, the proposed algorithm needs to be compared with other dynamic clustering approaches and its performance needs to be investigated as the dimension increases.

In general, evolutionary approaches have several advantages, namely [Jain *et al*. 1999]:

- they are global search approaches,
- they are suitable for parallel processing, and
- they can work with a discontinuous criterion function.

However, evolutionary approaches generally suffer from the following drawbacks [Jain *et al*. 1999]:

- they require the user to specify the values of a set of parameters (e.g. population size, p_c, p_m, etc.) for each specific problem, and

- the execution time of EAs is significantly higher than the execution time of other traditional clustering algorithms (e.g. K-means and FCM), especially when applied to large data sets.

3.1.8 Unsupervised Image Classification

Image classification is the process of identifying groups of similar image primitives [Puzicha *et al*. 2000]. These image primitives can be pixels, regions, line elements and so on, depending on the problem encountered.

There are two main approaches to image classification: supervised and unsupervised. In the supervised approach, the number and the numerical characteristics (e.g. mean and variance) of the classes in the image are known in advance (by the analyst) and used in the training step, which is followed by the classification step. There are several popular supervised algorithms such as the minimum-distance-to-mean, parallelepiped and the Gaussian maximum likelihood classifiers [Lillesand and Kiefer 1994]. In the unsupervised approach the classes are unknown and the approach starts by partitioning the image data into groups (or clusters), according to a similarity measure, which can be compared with reference to data by an analyst and used to segment the image.

Therefore, unsupervised classification is a special case of the general clustering problem where the data set is an image (or a set of images) and the patterns are the pixels of the image(s).

In general, the unsupervised approach has several advantages over the supervised approach, namely [Davies 1997]

- For unsupervised approaches, there is no need for an analyst to specify in advance all the classes in the image data set. The clustering algorithm automatically finds distinct clusters, which dramatically reduces the work of the analyst.
- The characteristics of the objects being classified can vary with time; the unsupervised approach is an excellent way to monitor these changes.
- Some characteristics of objects may not be known in advance. The unsupervised approach automatically flags these characteristics.

3.2 Image Segmentation using Clustering

Image segmentation is a fundamental process in several image processing and computer vision applications. It can be considered as the first low-level processing step in image processing and pattern recognition [Cheng *et al*. 2001]. Image segmentation is defined as the process of dividing an image into disjoint homogenous regions. These homogenous regions should represent objects or parts of them [Lucchese and Mitra 2001]. The homogeneity of the regions is measured using some image property (e.g. pixel intensity) [Jain *et al*. 1999]. Image segmentation can be formally defined as follows:

Given an image I and a homogeneity predicate P. The segmentation of image I is the partitioning of I into K regions, $\{R_1, R_2,\ldots,R_K\}$, satisfying the following conditions:

- Each pixel in the image should be assigned to a region, i.e.

  $\cup_{k=1}^{K} R_k = I$

- Each pixel is assigned to one and only one region, i.e.

$R_k \cap R_{kk} = \phi$ where $k \neq kk$

- Each region satisfies homogeneity predicate *P*, i.e.

  $P(R_k) = \text{True}, \quad \forall k = 1,\ldots,K$

- Two different regions can not satisfy *P*, i.e.

  $P(R_k \cup R_{kk}) = \text{False}$ where $k \neq kk$

There are many techniques for image segmentation in the literature; details can be found in Fu and Mui [1981], Pal and Pal [1993], Cheng *et al*. [2001], Lucchese and Mitra [2001] and Turi [2001]. In general, these techniques can be categorized into thresholding, edge-based, region growing and clustering techniques [Turi 2001]. Each of these categories are discussed in the following sections.

3.2.1 Thresholding Techniques

Thresholding [Gonzalez and Woods 1992; Jain *et al*. 1995] is the simplest image segmentation technique. In its simplest version an image is divided into two segments: object and background by specifying a threshold. A pixel above the threshold is assigned to one segment and a pixel below the threshold is assigned to the other segment. For more sophisticated images multiple thresholds can be used.

3.2.2 Edge-based Techniques

In edge-based techniques [Gonzalez and Woods 1992; Jain *et al*. 1995; Kwok and Constantinides 1997], segmentation is achieved by finding the edges of the regions.

This is usually accomplished by moving a mask (e.g. a 3×3 window) over the image to detect local changes in the image intensity.

3.2.3 Region growing Techniques

In region growing [Gonzalez and Woods 1992; Jain *et al*. 1995; Fuh *et al*. 2000], a set of seed pixels are chosen. Neighboring pixels of a seed are agglomerated if they satisfy a homogeneity criterion. This is repeated until no more pixels can be added to the region. This approach has some problems [Turi 2001]:

- The selection of the seed pixels which is not a straightforward task.
- The selection of the homogeneity criterion.

Region splitting and merging divide the image into regions. A region is then split if it does not satisfy a homogeneity condition. Regions can also be merged if their merging results in a region that satisfies some condition. This is repeated until no more splitting and merging can occur [Gonzalez and Woods 1992].

3.2.4 Clustering Techniques

Image segmentation can be treated as a clustering problem where features describing each pixel correspond to a pattern and an image region (i.e. segment) corresponds to a cluster [Jain *et al*. 1999]. This similarity is obvious by comparing the clustering problem definition (refer to section 3.1.1) and the image segmentation problem definition (refer to section 3.2). Therefore, clustering algorithms have been widely used to solve the problem of image segmentation (e.g. K-means [Tou and Gonzalez

1974], FCM [Trivedi and Bezdek 1986], ISODATA [Tou and Gonzalez 1974] and snob [Wallace and Dowe 1994]). However, it should be noted that the number of clusters is usually not known *a priori* in image segmentation. Therefore, clustering algorithms that do not require the user to specify the number of clusters are usually preferred.

In this thesis, the clustering problem and the image segmentation problem are considered to be similar. Thus, algorithms are proposed for both problems interchangeably. In the following, several representative clustering-based techniques are presented.

A hybrid approach combining agglomerative hierarchical clustering and region-based segmentation was proposed by Amadasun and King [1988]. The image is first divided into regions. Homogenous regions are specified and mean feature vectors are then determined for each homogenous region. The most similar mean feature vectors are merged. This process is repeated until the specified number of clusters is reached. One advantage of this approach is that it is computationally efficient, because hierarchical clustering is applied on the mean feature vectors instead of the image pixels. However, this approach has several drawbacks, namely [Turi 2001],

- it requires the user to specify the number of clusters in advance,
- it depends on the region size, and
- it depends on the used homogeneity criterion.

Clustering algorithms are usually applied to feature space, and as such they do not use any spatial information (e.g. the relative location of the patterns in the feature space). However, for image segmentation spatial information is important because pixels with

similar features are usually found near each other in the spatial domain [Liew *et al*. 2000]. To address this issue, a generalization of K-means that is adaptive and includes spatial information was proposed by Pappas [1992]. In this approach, *a posteriori* probability function is defined which constrains the region intensity and imposes spatial continuity [Turi 2001]. The iterative algorithm alternates between maximizing the *a posteriori* probability function and calculating the cluster centroids. The cluster centroids are initially equal to the K-means cluster centroids. The centroids are updated by averaging them over a sliding window. The size of the sliding window is progressively decreases [Lucchese and Mitra 2001]. Chang *et al*. [1994] extends this algorithm to color image segmentation. Saber *et al*. [1996] extends the approach of Chang *et al*. by proposing a hybrid approach combining color image segmentation and edge linking. Chen *et al*. [1998] applied an approach similar to Pappas [1992] to biomedical images. A drawback of the generalization of K-means approaches is that they require the user to specify the number of clusters in advance [Turi 2001].

A color map image segmentation algorithm combining FCM and a supervised neural network was proposed by Wu *et al*. [1994]. FCM is first applied giving a set of prototypes satisfying some validation criteria. A neural network with supervised learning is then used to optimize these prototypes. The optimized prototypes are used to segment the image using the nearest neighbor rule [Turi 2001].

A fuzzy image clustering algorithm which incorporates spatial contextual information was proposed by Liew *et al*. [2000]. A dissimilarity measure which considers the eight neighboring pixels of each pixel was proposed. The dissimilarity measure is adaptive in the sense that the effect of the neighboring pixels is suppressed in nonhomogenous image regions. In addition, a merging process that merges clusters based on their closeness and their degree of overlap is also used to determine the

"optimal" number of clusters. According to Liew *et al*. [2000], due to the incorporation of spatial information, this approach is faster, less sensitive to noise and more suitable for arbitrary shaped clusters than FCM.

Lim and Lee [1990] proposed a two-stage process called *thresholding and FCM*. In the first stage, a *coarse* segmentation is obtained by smoothing the histogram of each color component by a Gaussian convolution. Thresholds are set as the valleys of the smoothed histograms (the valleys are obtained using the first and second derivative of the smoothed histograms). A safe area around each threshold is determined. Each pixel outside these safe areas is assigned to a cluster according to its red, green and blue values. Cluster centroids are then calculated. In the second stage, a *fine* segmentation is obtained by assigning pixels in safe areas to their closest clusters as determined from the fuzzy membership functions. One advantage of this approach is that it dynamically determines the number of clusters. However, the number of clusters obtained is significantly affected by the smoothing function parameter and the size of the safe area [Turi 2001].

Color image segmentation using competitive learning based on the least-squares criterion was proposed by Uchiyama and Arbib [1994]. An image segmentation approach based on the mean shift algorithm was proposed by Comaniciu and Meer [1997]. Shi and Malik [1997] addressed image segmentation using clustering as a graph partitioning problem.

Zhang *et al*. [2001] proposed a hybrid approach combining hidden Markov random field (HMRF) and the EM algorithm to segment brain magnetic resonance (MR) images. A HMRF model is a stochastic process generated by a MRF. The HMRF state sequence can be observed through a field of observations [Zhang *et al*. 2001]. An advantage of HMRF is that it encodes spatial information, which is very

Book Chapter:

6. M. Omran, A. Engelbrecht and A. Salman. Image Classification using Particle Swarm Optimization. *Recent Advances in Simulated Evolution and Learning*, K. Tan, M. Lim, X. Yao and L. Wang (Editors), World Scientific, Series on Advances in Natural Computation, 2004.

Conference Publications:

7. M. Omran, A. Salman and A. Engelbrecht. Image Classification using Particle Swarm Optimization. In *Conference on Simulated Evolution and Learning*, Singapore, pp. 370-374, November 2002.