# Optimisation of a Weightless Neural Network Using Particle Swarms

A thesis
presented to the University of Kent at Canterbury
in fulfilment of the
thesis requirement for the degree of
Doctor of Philosophy
in
Electronic Engineering

By

M.A. Hannan Bin Azhar
March 2008

*To my wife Tasmina*

I hereby declare that I am the sole author of this thesis.

I authorize the University of Kent to lend this thesis to other institutions or individuals for the purpose of scholarly research.

I further authorize the University of Kent to reproduce this thesis by photocopying or by other means, in total or in part, at the request of other institutions or individuals for the purpose of scholarly research.

(M.A. Hannan Bin Azhar)

# Abstract

Among numerous pattern recognition methods the neural network approach has been the subject of much research due to its ability to learn from a given collection of representative examples. This thesis is concerned with the design of weightless neural networks, which decompose a given pattern into several sets of $n$ points, termed n-tuples. Considerable research has shown that by optimising the input connection mapping of such n-tuple networks classification performance can be improved significantly. In this thesis the application of a population-based stochastic optimisation technique, known as Particle Swarm Optimisation (PSO), to the optimisation of the connectivity pattern of such "n-tuple" classifiers is explored.

The research was aimed at improving the discriminating power of the classifier in recognising handwritten characters by exploiting more efficient learning strategies. The proposed "learning" scheme searches for 'good' input connections of the n-tuples in the solution space and shrinks the search area step by step. It refines its search by attracting the particles to positions with good solutions in an iterative manner. Every iteration the performance or fitness of each input connection is evaluated, so a reward and punishment based fitness function was modelled for the task. The original PSO was refined by combining it with other bio-inspired approaches like Self-Organized Criticality and Nearest Neighbour Interactions. The hybrid algorithms were adapted for the n-tuple system and the performance was measured in selecting better connectivity patterns. The Genetic Algorithm (GA) has been shown to be accomplishing the same goals as the PSO, so the performances and convergence properties of the GA were compared against the PSO to optimise input connections.

Experiments were conducted to evaluate the proposed methods by applying the trained classifiers to recognise handprinted digits from a widely used database. Results revealed the superiority of the particle swarm optimised training for the n-tuples over other algorithms including the GA. Low particle velocity in PSO was favourable for exploring more areas in the solution space and resulted in better recognition rates. Use of hybridisation was helpful and one of the versions of the hybrid PSO was found to be the best performing algorithm in finding the optimum set of input maps for the n-tuple network.

# Acknowledgements

All praises are for the Almighty.

I would like to express my warm thanks to my supervisors Dr. Keith Dimond and Dr. Farzin Deravi for their continuous guidance, encouragement and patience throughtout this research and writing of this thesis. I enjoyed working with them and will never forget their unfailing support and inspiration and the hours spent discussing various ideas and methods.

I gratefully acknowledge the financial support from the Department of Electronics, University of Kent, UK for this research work.

I would also like to thank all the members of staff in the Department of Electronics. I am particularly grateful to Dr. Kostas Sirlantzis for sharing his knowledge.

I want to thank my friends for making my life more enjoyable. I would like to mention just a few in particular: Sam, Sanaul and Mike.

Finally my thanks go to my family for their never ending love and care, especially to my mother Halima, my father Azhar Ali and my wife Tasmina for their continuous support during the difficult days of this research and writing.

# Table of Contents

# List of Acronyms

ANN ..... Artificial Neural Network

AC ..... Absolute Confidence

BCN ..... Boolean Convergent Network

CI..... Confidence Interval

CL..... Criticality Limit

CV ..... Cross-validation

DB ..... Database

EA ..... Evolutionary Algorithm

EP ..... Evolutionary Programming

ES ..... Evolutionary Strategies

FDR..... Fitness-to-Distance Ratio

GA ..... Genetic Algorithm

GCN ..... Generalised Convergent Network

GSN..... Goal Seeking Neuron

HSF ..... Handwriting Sample Form

MDB..... Majority Decision Block

NIST..... National Institute of Standards and Technology

NN..... Neural Network

OCR ..... Optical Character Recognition

OL ..... Overlapping Limit

PCN..... Probabilistic Convergent Network

PE..... Processing Elements

PLN..... Probabilistic Logic Node

PS..... Particle Swarm

PSO..... Particle Swarm Optimisation

RAM..... Random Access Memory

RC..... Relative Confidence

RnP..... Reward and Punishment

SA..... Simulated Annealing

SD..... Standard Deviation

SD3..... Special Database 3

SD7..... Special Database 7

SOC..... Self-Organised Criticality

TS..... Tabu Search

WNN..... Weightless Neural Network

# List of Symbols

$i$ ..... particle index

$j$ ..... class index

$C_j$ ..... $j$-th class

$p'j$ ..... number of tuples matured for class $C_j$

$Tf$ ..... number of tuples whose scores are higher than a predefined threshold

$S$ ..... total number of vailable data samples

$S_l$ ..... available data samples for training

$S_t$ ..... available data samples for testing

$S_e$ ..... available data samples for evaluation

$S_{Cj}$ ..... number of $C_j$ samples correctly recognised

$S_{rj}$ ..... number of $C_j$ samples rejected

$S_{mj}$ ..... number of $C_j$ samples misclassified

$P_c$ ..... positive points for correct classification

$P_r$ ..... negative points for rejection

$P_m$ ..... negative points for misclassification

$O_{ji}$ ..... performance score of $i$-the individual for class $C_j$

$\tau$ ..... time constant

$TD$ ..... percentage drop of $\tau$

$d$ ..... dimension of the problem

$X_i$ ..... vector containing the current location in the solution space

$V_i$ ..... vector containing the velocity for each dimension of $X_i$

$V_{max}$ ..... maximum distance a particle can travel in an iteration

$X_{max}$ ..... sets bound for the search

$P_i$ ..... vector containing the location of particle's own best in history

$P_g$ ..... location of the best particle in the neighbourhood

$P_{fdr}$ ..... a third particle that maximizes fitness to distance ratio

$P_b$ ..... $P_{fdr}$'s best found position so far

$P_{bd}$ ..... location along dimension $d$ of $P_{fdr}$'s best found position

$l_{best}$ ..... location with highest value from a small group of particles

$V_{i,d}$ ..... velocity of particle $i$ along dimension $d$

$P_{i,d}$ ..... location along dimension $d$ at which the particle had the best fitness

$X_{i,d}$ ..... current position of particle $i$ along dimension $d$

$P_{gd}$ ..... the current location along dimension $d$ of the particle with best fitness

$S_{toch}$ ..... stochastic weight factor in the range $\{0,1\}$

$\psi 1$ ..... cognitive learning rate

$\psi 2$ ..... social learning rate

$\psi 3$ ..... learning rate for FDR component

$ran1$ ..... stochastic weight factor for cognitive component

$ran2$ ..... stochastic weight factor for social component

$ran3$ ..... stochastic weight factor for FDR component

$\omega$ ..... inertia weight

$CL_{min}$ ..... the lowest criticality limit

$Q$ ..... population size

$R$ ..... total available tuples to be optimised

$W$ ..... width of the image

$H$ ..... height of the image

$L$ ..... resolution of the image

$t_{exp}$ ..... experimental t value

$t_{th}$ ..... theoretical t value

$df$ ..... degrees of freedom

$\eta$ ..... connectivity pattern

$H_0$ ..... null hypothesis

$H_A$ ..... alternative hypothesis

$\alpha$ ..... probability that the null hypothesis will be rejected in error

$n_A$ ..... number of measurements in dataset A

$n_B$ ..... number of measurements in dataset B

$\bar{x}$ ..... mean

$t$ ..... time steps

$t_u$ ..... unproductive interval

# List of Figures

# List of Tables

# Chapter 1

# Introduction

## 1.1 General Introduction and Motivation

Pattern recognition as a field is extremely diversified and has been applied in many areas such as science, engineering, business, medicine etc. The aim of pattern recognition is to classify objects into identifiable categories or classes after extracting features from the data. This data may be numerical, pictorial, textural, linguistic or any combination of these categories. Numerous techniques for pattern recognition can be investigated in four general approaches of pattern recognition, as suggested in [Jain *et al.* 2000]: template matching, statistical techniques, structural techniques and neural networks (NNs). The template matching technique is based on matching the stored prototypes against the pattern to be recognized. Statistical technique is based on the assumption that there is an underlying and quantifiable statistical basis for generation of the patterns. In structural technique the underlying structure of the pattern provides the information fundamental for recognition. The neural classification emulates the computational paradigm of the behaviour of neurones and their interconnections in human brain. Instead of recognizing a pattern by following a set of human-designed rules, as in the structural approaches, neural nets learn the

underlying rules from a given collection of representative examples. Among neural network models, the weightless or n-tuple form of network [Bledsoe and Browning, 1959] stands out due to its own advantages over a variety of pattern recognition algorithms [Rohwer and Morciniec, 1998]. Considerable research activity has focused on the n-tuple method, both regarding theoretical issues [Rohwer and Morciniec, 1998; Jørgensen and Linneberg, 1999] as well as applications to real-world tasks [Rohwer and Cressy, 1989]. Several applications of n-tuple-based networks to handwritten character recognition tasks have been reported. Recognition of handwritten characters by a computer has been a topic of extensive research for many years [Govindan and Shivaprasad, 1990; Mori *et al.*, 1992; Nagy,1988]. It plays important role in many applications such as postal address interpretation, bank checks, tax forms and census forms reading.

The n-tuple method decomposes a given pattern into several sets of $n$ points, termed n-tuples. The classifier stores class-specific information about the training set in a number of look-up tables .The entries in each look-up table are addressed by sampling $n$ specific data locations of the input that constitutes a 'feature' of the pattern. A pattern is classified as belonging to the class for which it has the most features in common with at least one training pattern of that class. The input connection mapping of the n-tuple classifier determines the sampling and defines the locations of the pattern matrix. There will be a vast number of possible connections for a matrix with the dimension like 32 by 32. The classification and generalization performance are highly dependent on these input mappings [Bishop, 1990; Jørgensen *et al.*, 1995]. A random map is suitable for an un-optimised problem as it samples the point throughout the pattern matrix [Aleksander and Stonham, 1979]. Considerable research shows that by optimising the connections classification performance can be improved significantly [Bishop, 1990; Jørgensen *et al.*, 1995; Garcia, 2003]. Stochastic search algorithms like Particle Swarm [Kennedy and Eberhart, 1995] and

Genetic Algorithm [Holland, 1975] are used to find near-optimal solutions. 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 [Løvbjerg, 2002; Spall, 2003].

Particle swarm is a population based stochastic optimisation technique developed by Eberhart and Kennedy in 1995, motivated from the simulation of social behaviour of bird flocking or fish schooling. The particle swarm searches optima in the solution space and shrinks the search area step by step. It refines its search by attracting the particles to positions with good solutions. In a population based search each individual's performance is measured by a fitness function [Holland, 1975]. One advantage of Particle Swarm Optimisation (PSO) is that it can deal with a large number of problem parameters and no rigid assumption about the problem is necessary. PSO provides intermediate results at any time during the computation. So it can be stopped at any time depending on the precision wanted. Being successfully applied in many areas like function optimisation, artificial neural network training [Parsopoulos and Vrahatis, 2001b; Settles *et al.*, 2002] or fuzzy system control [Esmin *et al.*, 2002], the PSO seems to be a good candidate to find an optimal set of input maps for the n-tuple network.

Although, in general, PSO results good solutions, in high–dimensional spaces it might stumble on local minima [Kalyan *et al.*, 2003]. In order to be less susceptible to premature convergence, the maintenance of "diversity" in particle swarm is important [Kalyan *et al.*, 2003; Løvbjerg and Krink, 2002]. One way to add diversity in PSO is to use the Self-Organized Criticality (SOC) [Bak, 1996]. Self-organized criticality has been found in a variety of phenomena such as earthquakes, volcanic activity, the game of life, landscape formation and stock markets. SOC describes how small amounts of external influence can occasionally lead to the big changes observed in complex systems. Extending the PSO with SOC seems very promising reaching faster convergence and better solutions [Løvbjerg and Krink, 2002] and the

resulting algorithm can be named as SOC-PSO. An alternative way of improving the PSO is by hybridising it with a technique which considers the neighbourhood interactions that is naturally observed and expected in animal behaviour [Kalyan *et al.*, 2003]. A significant modification in particle dynamics is required to introduce the effects of multiple other particles in each particle. [Kalyan *et al.*, 2003] proposed a method where each particle is moved towards other nearby particles with a more successful search history, instead of just the best position discovered so far. The proposed algorithm is described as Fitness-Distance-Ratio (FDR) based PSO (FDR-PSO) and it selects an influential particle, which satisfies the fact that it must be near the particle being updated and it should have visited a position of higher fitness.

## 1.2 Thesis Scope

This thesis investigates the application of an efficient optimisation method, known as Particle Swarm Optimisation, to optimise the connectivity pattern of an n-tuple classifier. The research will aim to improve the discriminating power of the classifier in recognising patterns by exploiting more efficient learning. The "learning" scheme will select 'good' input connections of the n-tuples in an iterative manner. At each iteration the performance or fitness of each input connection will be evaluated, so a fitness function will be formulated. Development of PSO on n-tuple systems will be explained in detail. The original PSO can be refined by combining it with other bio-inspired approaches like Self-Organized Criticality and Nearest Neighbour Interactions [Kalyan *et al.*, 2003]. The hybrid PSO algorithms will be applied to optimise the n-tuple network. The performance of the hybrid system will be investigated. The Genetic Algorithm has proven to be accomplishing the same goal as the PSO [Kennedy and Spears, 1998], so the foundations, performances and convergence properties of the GA and PSO will be compared to select the optimum set of n-tuples. Different parameter settings of all bio-inspired approaches will be

examined. The performance of the optimised network will be measured in recognizing handwritten characters from the NIST [Wilkinson *et al.,* 1992] database. The character recognition research can be classified based upon two major criteria: 1) the data acquisition process (on-line or off-line) and 2) the text type (machine-printed or handwritten). The off-line handwritten character recognition has been selected as the application domain of this research as it is relatively more complex compared to on-line and machine-printed recognition [Arıca and Yarman-Vural, 2001]. Due to computational extensive nature of the simulations and also the stochastic nature of the proposed algorithms, all presented results in this thesis will be taken over several test runs. Statistical analysis like Student's t-test will be performed to explore the significance of the results.

## 1.3 Original Contribution

The main contributions of this thesis are:

- The adaptation of the Reward and Punishment (RnP) based performance measure to the evaluation of connectivity patterns of the n-tuple network.
- The development of a new stochastic search strategy so that more time is given to finding features for a difficult class than an easily recognisable class.
- The application of the particle swarm intelligence in finding an optimum set of input connections to an n-tuple classifier.
- The adaptation of the Self Organised Criticality algorithm for n-tuples and to extend the original PS algorithm with the SOC in exploring better connectivity patterns to n-tuples.
- The application of the hybrid PSO and Fitness-to-Distance-Ratio based algorithm in finding better n-tuples.

- A novel hybridisation of the SOC, PSO and FDR algorithm to form the SOC-FDR-PSO to optimise the n-tuple classifier.
- The evaluation of the efficiency of all proposed methods by applying the trained classifier to recognise handprinted digits from the well-known and widely used NIST database.
- and the comparison of the performance of Particle Swarm optimised training for the n-tuples with other algorithms including Genetic Algorithm based training.

## 1.4 Thesis Outline

In addition to this introductory chapter the thesis consists of seven more chapters and two appendices. Chapter 5 to Chapter 7 represents the core of the thesis whereas Chapter 2 to 4 describes the background knowledge, experimental framework and literature reviews of related works. The organization of the work is as follows:

**Chapter 2 Optimisation Algorithms**

This chapter briefly reviews the subject of optimisation. Traditional and stochastic optimisation methods will be discussed first. This will be followed by the discussion in Evolutionary Computation with more emphasis on Genetic Algorithms. Different controlling parameters of GA will be discussed. The area of swarm intelligence will be described next. An elaborated discussion of particle swarm optimisation and its various modification will be presented. In order to provide a complete coverage of swarm intelligence background, a brief overview of another swarm intelligence model, Ant colony Systems, will be given.

**Chapter 3 The n-tuple Classifier**

This chapter discusses the field of artificial neural networks with the emphasis on weightless approach. Learning in neural networks will be briefly introduced here.

This will be followed by the description of the memory based n-tuple network. Network's training and recognition techniques will be explained in detail. Different parameters controlling the performance of the n-tuple network will be addressed next. Finally the arguments behind optimising the connectivity pattern of the n-tuple network will be discussed.

**Chapter 4 Experimental Framework**

This chapter introduces the experimental infrastructure of the research. The experimental procedures have been explained here. The database for the experiments has been described. Rationale has been given for choosing the NIST database for experiments. Reasons have been given for choosing the specific n-tuple size and number of tuples in the experiments. The importance of using significance testing for the experiments has been explained and for this the Student's t-test has been introduced. The use of Box Plot has been described to facilitate statistical comparisons of results graphically. Finally this chapter mentions the list of software tools and programming language used in the experiments.

**Chapter 5 Reward and Punishment**

This chapter describes a stochastic tuple selection algorithm. The development of a measure of the 'goodness' of a solution based on a Reward and Punishment concept has been introduced in this chapter. The equations and different parameters of the RnP measure have been explained. Experimental results on optimising the learning of the n-tuple network by using the RnP based stochastic search have been presented in this chapter. The analysis of the statistical significance of the results has been also included.

**Chapter 6 Particle Swarm to Optimise n-tuples**

This chapter introduces the implementation of Particle Swarm Optimisation on the n-tuple network. The learning algorithm by PSO has been elaborately

explained. The affects of different controlling parameters on the performance of the PSO have been described. This is followed by the experimental results of optimisation by PSO and a comparison has been given with conventional approach and the RnP based approach. Later in the chapter the development of a GA algorithm has been described. This is followed by the analysis of experimental results comparing the performances and convergence properties of the GA and PSO based search in selecting a set of optimum n-tuples. Finally this chapter explains how the speed of the search can be increased without noticing any significant loss in performance.

**Chapter 7 Hybridising Particle Swarm**

This chapter first describes the importance of the diversity required in particle swarm based optimisation. This is followed by the implementation of the Self Organised Criticality algorithm on n-tuple networks. The hybrid SOC-PSO algorithm and its parameters have been explained. Next a second bio-inspired algorithm named Fitness-to-Distance-Ratio has been introduced in favour of adding diversity in PSO. The combined FDR-PSO algorithm has been described. This is followed by the hybridisation of PSO with both FDR and SOC algorithms and the resulting algorithm has been presented as SOC-FDR-PSO. Finally experimental results have been given comparing performances of n-tuple networks trained by various hybrid approaches. The affects of different parameters on the performance of the network have been investigated in this chapter.

**Chapter 8 Conclusion**

This chapter highlights the conclusions of this thesis and discusses direction for future research. A summary of what has been achieved in the thesis has been presented here.

A list of publications derived from the work has been presented in Appendix A. Appendix B shows an example image of Handwriting Sample Form (HSF) of the NIST database. A subset of this database has been used in the experiments of the research.

# Chapter 2

# Optimisation Algorithms

---

## 2.1 Optimisation

Optimisation problems are very much a part of pattern recognition and computer vision [Shang and Wah, 1996]. Optimisation algorithms seek values for a set of parameters that maximize or minimize objective functions subject to certain constraints [Rardin, 1998; Van den Bergh, 2002]. Any maximization problem can be converted into a minimization problem by taking the negative of the objective function, and vice versa. Three main ingredients for optimisation problems are: an objective function, a set of unknowns or variables and set of constraints that allow the unknowns to take on certain values but exclude others. A feasible solution is found when values for the set of parameters satisfy all constraints. 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 optimisation problem is the device sizing in electronic design, which is the task of choosing the width and length of each device in an electronic circuit. Here the variables represent the widths and lengths of the devices. The constraints represent a variety of engineering requirements, such as limits on the device sizes imposed by the manufacturing

process, timing requirements that ensure that the circuit can operate reliably at a specified speed, and a limit on the total area of the circuit. Optimisation techniques are used on a daily base for industrial planning, resource allocation, scheduling, decision making etc. Furthermore, optimisation techniques are widely used in many fields such as business, industry, engineering and computer science. Through active research in the field of optimisation new methods are regularly being developed [Chinneck, 2006].

Global optimisation is the task of finding the absolutely best set of parameters to optimise an objective function. Depending on problems the best set can generate either the highest (for maximization problem) or lowest (for minimization problem) function value. In a local optimisation problem the highest or lowest function value stays in a finite neighbourhood. There are many local optimisation algorithms in the literature. For more detail the reader is referred to [Aarts and Lenstra, 2003] and [Korte and Vygen, 2002]. Global optimisation problems are typically quite difficult to solve exactly and fall within the broader class of nonlinear programming (NLP) [Gray *et al.*, 1997]. More details about global optimisation can be found in [Pardalos *et al.*, 2002; Floudas and Pardalos, 1992; Horst *et al.*, 2000]. Remainder of this chapter has been organised as follows: next section will discuss about the traditional optimisation algorithms, Section 2.3 will introduce the stochastic approaches of optimisation. This will be followed by population based evolutionary algorithms for optimisations. Genetic algorithms and Particle Swarm Optimisation will be described in great detail.

## 2.2 Traditional Optimisation Algorithm

Traditional optimisation algorithms use exact methods to find the best solution. Exact methods involve more computational effort and usually require large amounts of computer memory. One exact method is exhaustive (or brute force) searching, where

the algorithm produces the entire solution space for the problem so that the global optimal solution is guaranteed to be found. A brute force algorithm is often the least desirable choice because its cost is proportional to the number of candidate solutions, which, in many practical problems, tends to grow very quickly as the size of the problem increases. Therefore, brute force search is typically used when the problem size is limited and is not appropriate for the class of problems known as NP-hard problems [Papadimitriou, 1994] that require an enormous (exponential) amount of computing power or time to be solved exactly. A problem is said to be NP-hard if it is solvable in polynomial time by a nondeterministic Turing machine [Turing, 1937]. The time to exhaustively search an NP-hard problem increases exponentially with problem size.

Another exact method is Branch-and-Bound [Hendy and Penny, 1982], which deals with optimisation problems over a search space that can be presented as the leaves of a search tree. It works when the search tree is monotonous - the score of each node in the search tree is at least as bad as that of any of its ancestors. Branch-and-Bound is guaranteed to find the optimal solution, but its complexity in the worst case is as high as that of exhaustive search. Other exact methods include linear programming and dynamic programming. More details about exact methods can be found in [Michalewicz and Fogel, 2000].

## 2.3 Stochastic Algorithms

Real world problems are normally NP-hard problems where real optimality condition is far too complex to be grasped by any particular method. In such cases it is desirable to find near optimal solutions with the assumption 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]. A stochastic algorithm is a method that proceeds by taking a random walk in the search space with the objective of finding a near optimal

solution, thus 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]. The expected run time for stochastic algorithms is usually shorter than for exact ones, but the worst case run time will often be the same or longer. Stochastic search algorithms are easy to implement and suitable for many combinatorial problems (problems with discrete variable parameters). They can benefit from parallelism and can be used in a multiprocessor environment.

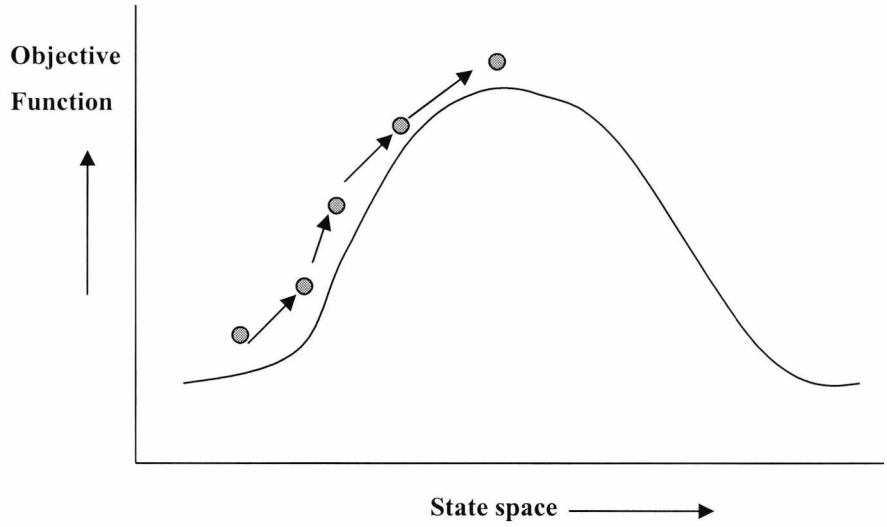



**Figure 2.1 Hill-Climbing**

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]. Hill-Climbing exploits the analogy of climbing hills (Figure 2.1) to find the optimum. It always looks for the next change which will improve the current state. In Hill-Climbing an initial candidate solution is generated randomly, the current solution. The technique then investigates its immediate

neighbourhood to find a better solution. This process is repeated until no more improvement can be made. Hill-Climbing techniques differ from each other in the way they explore their neighbourhood and in the way they replace their candidate solution. Binary encoded Hill-Climbers can be set to explore every neighbouring point by flipping every bit one by one. Deterministic Hill-Climbing techniques, such as steepest ascent Hill-Climbing as shown in Figure 2.2, are easy to use but can get stuck in local optima. Because it can only go uphill, it cannot climb down a local peak to find a higher one. In stochastic Hill-Climbing techniques a weaker neighbouring solution can replace the candidate solution with a given probability, which gives the ability to escape local optima [Michalewicz and Fogel, 2000].

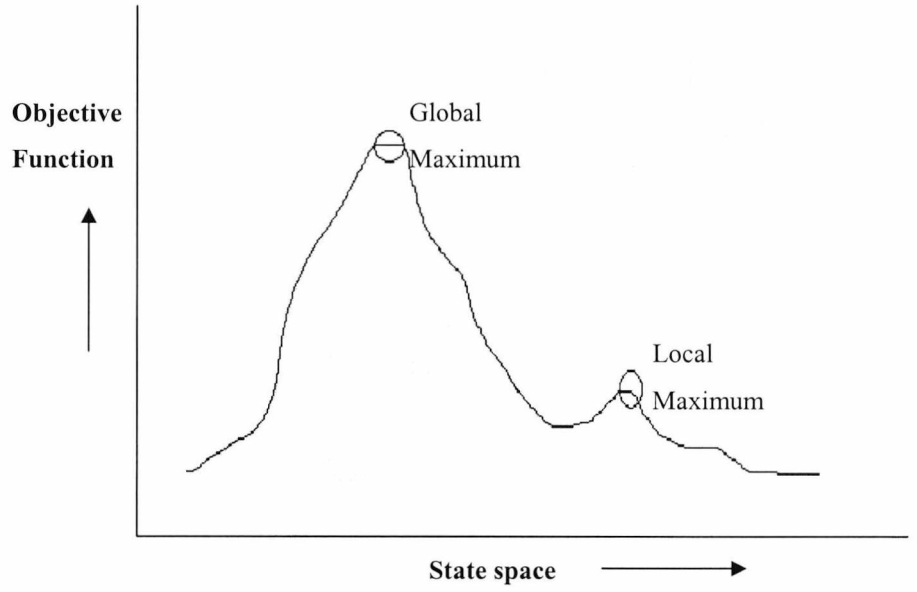



**Figure 2.2 Local and Global maximum in solution space**

Simulated Annealing (SA) [Van Laarhoven and Aarts, 1987] is a general purpose global optimisation technique for very large combinatorial problems. It is a stochastic search algorithm, which exploits an analogy between the way a metal cools and freezes into a minimal energy crystalline structure (the annealing process) and the

search for an optimum solution in a search process. In SA the probability of replacement of the current solution by a weaker solution depends on a temperature value. If temperature decreases over time the probability of changing to a lower energy state also goes down [Salman, 1999]. SA allows downhill movements (Figure 2.2) to be made and thus can escape from local optima and find the global optima.

Tabu search (TS) is a heuristic search algorithm [Glover, 1986], which makes use of some memory of the states that has already been investigated. The algorithm does not re-visit those states. Unlike hill climbing here inferior solutions are selected if better solutions are in the memory, thus, TS avoids being trapped in a local optimum. The moves that are not allowed to be re-visited are held in a list and these moves are called 'tabu'. The tabu list is used to avoid the search getting into a loop by continually searching the same area without actually making any progress [Gabarro, 2000]. Tabu search starts with a randomly chosen current solution. A set of test solutions is 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 [Salhi, 2002]. A common aspiration criterion could be to accept a move that results in better solution than the best solution so far. Another aspiration criteria could be a move favouring more drastically different solutions. Difference could be based on distance in search space or based on difference in the value of objective function compared to the current best solution.

## 2.4 Evolutionary Computation

Evolutionary Computation (EC) is a robust and powerful stochastic search mechanism inspired by biology [Back, 1992; 1994; Kim and Myung, 1997; Back *et al.*, 1996; 1997a; 1997b; Collins, 1998; Fogel, 1994; 1995; Spears *et al.* 1993]. EC differ from other optimisation methods, such as Hill-Climbing and Simulated Annealing, in the fact that in EC a population of potential solutions to a problem is

maintained, and not just one solution [Engelbrecht, 2002; Salman, 1999]. By repeatedly applying the evolutionary operators to the current population at each generation, a new population of individuals with better performance is created to search the space of potential solutions. The dominant methodologies of evolutionary computing are evolutionary programming (EP) evolutionary strategies (ES) and Genetic Algorithm (GA). 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 optimisation [Yao, 1997].

```
t = 0; /* Initial Generation*/
population_initialise(t);
evaluation(t);
repeat
        t=t+1; /* Next Generation */
        select_parents(t);
        crossover(t);
        mutate(t);
        evaluate(t);
        survivors_selection(t);
until best individual meets criterion;
```

**Figure 2.3 Pseudo-code of Genetic Algorithm**

## 2.5 Genetic Algorithm

Genetic Algorithms are search algorithms utilising the mechanics of Darwinian natural selection and genetics. GA performs well on many different types of problems and they are less susceptible to getting stuck at local optima than a gradient search methods. GAs introduced by John Holland [Holland, 1975] are adaptive search strategies based on a highly abstract model of biological evolution to find a possible solution in a given problem space. This space, referred to as the search space, comprises all possible solutions to the problem at hand. Figure 2.3 outlines a typical genetic algorithm. A population of individual structures is initialised and then evolved from generation $t$ to generation $t + 1$ by repeated applications of fitness evaluation, selection, recombination and mutation. Initial population of individuals is generated at random or heuristically. Every evolutionary step (generation), the individuals in the current population evaluated according to some predefined quality criterion, referred to as the fitness, which is equated with goodness of solution. Genetic algorithms are stochastic iterative processes that are not guaranteed to converge; the termination condition may be specified as some fixed, maximal number of generations or as the attainment of an acceptable fitness level. Gene representation, fitness or objective function and genetic operators are the three most important aspects of using GA.

### 2.5.1 Gene Representation

Individual structures in the population are encoded by chromosomes or genotypes, which may be represented by strings of bits (bit strings), where a single bit encodes a gene. This is known as binary encoding. The different values a gene can take are called alleles. The characteristics or features of each feasible solution are called as phenotype. Some of the non-binary representations of genes include floating point [Janikow and Michalewicz, 1991], integer [Bramlette, 1991], graycoded [Rana and

Whitley, 1998] and matrix [Michalewicz, 1996] type. For more detail about gene representations please see [Goldberg, 1989]. Uniform evolutionary operators can be used with binary representation for any problem [Van den Bergh, 2002], but for non-binary genes need different evolutionary operators for each representation.

### 2.5.2 Fitness function

The object or fitness function defines how good each solution or individual is. One of the key aspects for Genetic Algorithm's success is the right choice of a fitness function that accurately quantifies the quality of candidate solutions. Alongside good fitness, chromosome representation has to be correct to effectively solve a particular problem. These two parameters are problem dependent. A wrong selection of these two parameters will drastically affect the performance of GAs. While optimising combinatorial problems by GA a situation might exist in the search space where fitness function do not map to feasible solutions. A solution to this problem could be use of a penalty term with the original fitness function, which will result in chromosomes with infeasible solutions and eventually they will disappear from the population [Fletcher, 2000].

### 2.5.3 Selection

Selection is the competition among individuals of the population to become parents of the next generation. The fitter the member of the population the more likely it is to produce an offspring. In addition, the selection operator can be used to select elitist individuals. Given the fitness of each population member GA can select good members in the current population for the next population. A selection process is usually biased toward fitter chromosomes and it pushes the search on apparently more profitable regions in the search space [Angeline, 1998a]. Examples of well-known selection approaches are as follows:

- *Roulette wheel selection:* In this method the possibility of picking an individual is proportional to the individual's score or fitness. The fitter the chromosome, the more chance 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:* The previous selection will have problems when the fitness differs very much. When one or few chromosomes have very high fitness on the roulette wheel then the lower fit chromosomes will have very few chances to be selected. This will increase selection pressure, which will cause diversity to decrease rapidly resulting in premature convergence. This fact has been illustrated for 5 chromosomes in Figure 2.4. Rank selection first ranks the population and then every chromosome receives fitness from this ranking. As shown in Figure 2.5 the worst will have fitness 1, second worst 2 etc. and the best will have fitness 5 (number of chromosomes in population). After this all the chromosomes have a chance to be selected. Rank selection still prefers the best chromosomes; however, there is no domination as in the case of roulette wheel selection. But this method can lead to slower convergence, because the best chromosomes do not differ so much from other ones [Gray *et al.*, 1997].

- *Tournament selection:* Tournament selection [Goldberg, 1989] runs a "tournament" among a few individuals chosen at random from the population. The best individual is copied into the intermediate population. This process is repeated until the mating pool contains a sufficient number of chromosomes to start the mating process. Selection pressure can be easily adjusted by changing the

tournament size. If the tournament size is larger, weak individuals have a smaller chance to be selected.

- *Elitism:* In this approach, the best chromosome, or a user-specified number of best chromosomes, is copied to the population in the next generation. The remaining chromosomes are then chosen using any selection operator. Elitism can very rapidly increase performance of GA, because it prevents losing the best found solution to date. [Gray *et al.*, 1997].

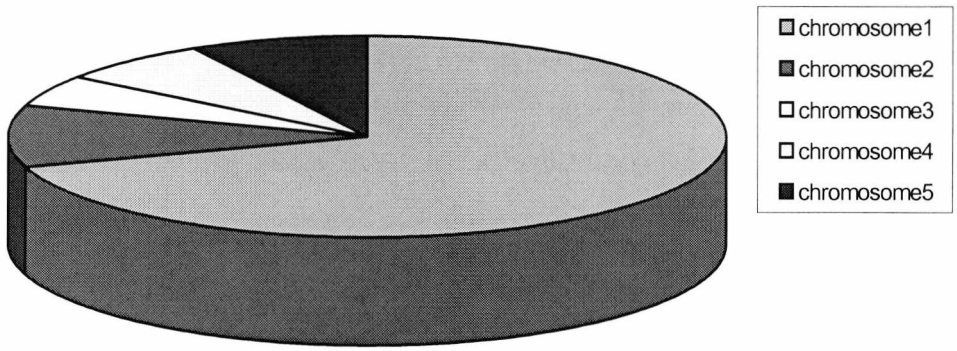



**Figure 2.4 Fitness graph before ranking**

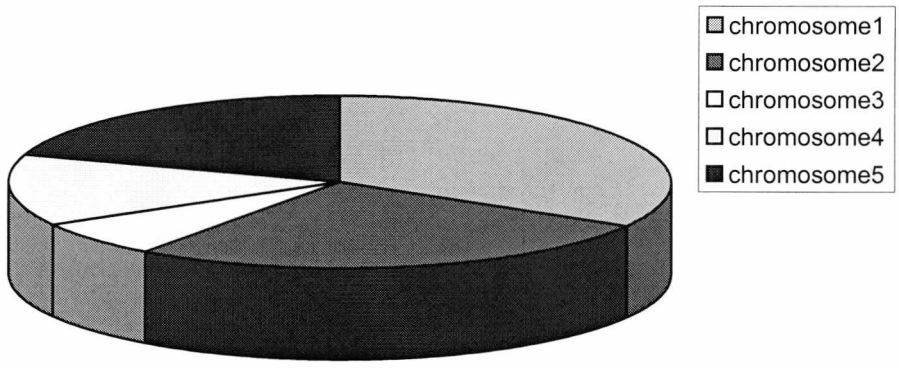



**Figure 2.5 Fitness graph after ranking**

### 2.5.4 Crossover

Crossover is the main operator in GA, which assists exploration to new locations in the search space [Salman, 1999]. Crossover uses the current diversity in the population to generate new solutions. Given two population members (or parents), crossover combines or mates parts of the two parents (or chromosomes) to yield two new chromosomes (offspring) with the hope that the new chromosome may be better than both of the parents if it takes the best characteristics from each of the parents. Crossover occurs during evolution according to a user-definable probability. The three main crossover operators are described below. Several other forms of crossover have been investigated in [Michalewicz, 1996; Booker *et al.*, 1997; Krink and Løvbjerg, 2002].

- *Single point crossover:* [Holland, 1975] provided one of the earliest analyses of "one-point" crossover. It can be implemented by randomly selecting a common crossover point in both parents, and swapping the right end of both chromosomes as illustrated in the following example:

  Parent A: 01001**010**

  Parent B: **11110**011

  Offspring A: 01001011

  Offspring B: **11110010**

- *n-point crossover:* [Jong, 1975] extended the above single point analysis to an "n-point" technique. It applies the same strategy, but divides the original strings in n cut-points and substrings are swapped among these points. For instance for a two-point crossover system, two positions are randomly selected. The middle parts of the two parents are then swapped to generate two new offspring. This is illustrated in the following example:

Parent A: 01**001**10

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

Offspring A: 01110010

Offspring B: **11001011**

- *Uniform crossover:* [Syswerda, 1989] introduced "uniform crossover" that does not use cut-points but instead creates offspring by deciding, for each allele of one parent, whether to swap that allele with the corresponding allele in the other parent. Uniform crossover is more flexible to achieve any combination of genes. In this approach, alleles are copied from either the first parent or the second parent with some probability, usually set to 0.5. An example has been given below.

Parent A: 0**1**001**0**

Parent B: **1**111**00**1

Offspring A: 01101011

Offspring B: **11010010**

### 2.5.5 Mutation

Mutation is the secondary operator to keep genetic diversity in the population [Back *et al.*, 1997a]. Mutation is applied to the offspring chromosomes after crossover is performed. Mutation implements a random change in the value of one or more genes for introducing new information into the system. Thus mutation introduces a certain amount of randomness to the search. It helps the search find solutions that crossover alone might not encounter. In a bit-string it can be realised by flipping a bit. However, mutation functions as a background operator with a very low probability of application. As it has been observed as an infrequent phenomenon in both nature and

GAs [Løvberg, 2002]. Alternately a mutation operator could be set as the inverse of the number of genes in a chromosome or chromosome size [Goldberg, 1989]. Mutation increases diversity and searches new areas of the state space. A destructive mutation keeps the mutated population member whatever its fitness, whether good or bad. On the other hand a constructive mutation only keeps the mutation if it is more fit than before mutation, but this requires an extra call to the fitness function.

### 2.5.6 Premature Convergence in GA

Genetic algorithms may lead to premature convergence if the population of a GA reaches such a sub optimal state that the genetic operators are no longer able to produce offspring that are able to outperform their parents [Fogel, 1994; Affenzeller and Wagner, 2004]. The intuitive reason for premature convergence is that the individuals in the gene pool are too 'alike'. If a chromosome is far fitter than its rivals early on, it can come to dominate a population, leading to loss of genes that may later lead to better solutions and this prevents further exploration of search space [Dorigo and Di Caro, 1999]. Thus one or few dominating chromosomes near local optima can attract individuals in a population before reaching the global optimum solution, resulting in premature convergence. So Premature convergence can be avoided by using some mechanisms like using subpopulation, employing high mutation rate, through fitness scaling etc.

- *Subpopulation*: When chromosomes are divided into subpopulations, each subpopulation is evolved independent of the other subpopulations. Subpopulations interact through exchange of a number of chromosomes. This scheme ensures diversity in the population to prevent premature convergence.
- *High Mutation rate*: Increase in mutation rate aids in exploring new areas in the search space and increases diversity too. As the modality of a search space increases the likeliness of a solution being trapped between ravines is more.

Crossover alone works as a local search operator in a multi-modal search space. To escape from this trap a high mutation rate could be used, as mutation re-initialises chromosomes to new locations. An overly high mutation rate should be avoided as then GA starts to resemble a random walk rather than a directed process.

- *Fitness Scaling*: Fitness scaling is a process that re-scales the fitness with respect to the average of the population, so that the fittest chromosome is only, say, twice as likely to be chosen for cross-breeding as the average chromosome.

## 2.6 Swarm Intelligence

A secondary area of research emerged in the field of evolutionary computation is known as the Swarm Intelligence where the searches are guided by social pressure rather than evolutionary pressure used in EAs. But like EAs, swarms also consist of populations of individuals representing candidate solutions to a problem. Swarm intelligence emulates the searching techniques of insects where the communications between the members of a swarm direct the search. Two significant algorithms have emerged in this field: Ant Systems and the Particle Swarm Optimiser.

### 2.6.1 Ant Systems

Ant systems simulate the search techniques of biological ants as they locate food and return it to the colony, reinforcing their paths on the return trip so that other ants can locate the same food source. The algorithm allows an initial population of 'ants' to walk randomly through the solution space. While walking each ant labels its path with a virtual pheromone marker proportional to the fitness of the path. When one ant finds a good path from the colony to the food source the other ants tend to follow that path, but may test alternate paths with some level of probability. If the alternative

path selected results in a stronger pheromone marker, there will be a greater probability of that path being selected by the next ant. The algorithm also allows the evaporation of the pheromone trail over time, which helps to avoid the convergence of the solution to a local optimum. Without evaporation the exploration of the solution space would be constrained. Ant systems have been applied successfully to path problems, network load balancing problems [Di Caro and Dorigo, 1998] and the quadratic assignment problem [Maniezzo and Colorni, 1999].

### 2.6.2 Particle Swarm Optimisation

A particle swarm optimiser is a population-based stochastic optimisation algorithm that emulates a flock [Kennedy and Eberhart, 1995; Kennedy and Eberhart 2001] searching over the solution landscape by sampling points and converging the swarm on the most promising regions. PSO is influenced by the simulation of social behaviour 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 GAs [Coello Coello and Lechuga, 2002]. PSO is easy to implement and has been successfully applied to solve a wide range of optimisation problems such as continuous non-linear and discrete optimisation problems [Kennedy and Eberhart, 1995; Kennedy and Eberhart, 2001; Eberhart and Shi, 1998a]. In particle swarm, a particle's movement is influenced by its velocity, an attraction to its previously found promising search area and an attraction towards the best area discovered by its neighbours. Thus the social pressure on a particle applied by other particles in the neighbour hood plays an important role behind the convergence in particle swarm.

The basic structure of a particle is significantly more complex than that of a member of a GA population. A particle is denoted by $i$. If there are $Q$ particles in total

then it is said that swarm size is $Q$ and $i$ can vary from 1 to $Q$. Each particle consists of the following main components:

- $X_i$ is a vector containing the current location in the solution space. The size of the $X_i$ is denoted by $d$, which is the dimension of the problem or the number of variables used by the problem being solved. For example, for some function $f$(u, v, w), $X_{i,1}$ corresponds to the $u$ value, $X_{i,2}$ corresponds to the $v$ value, and $X_{i,3}$ corresponds to the $w$ value.
- Fitness is the quality of the solution represented by the vector $X_i$. This is a problem specific evaluation function and refers too how well a particle performs. In a flock of birds this might be how close a bird is to a food source, in an optimisation algorithm this refers to the proximity of the particle to an optima.
- $V_i$ is a vector containing the velocity for each dimension of $X_i$. It defines the step size of movement along a dimension from the current position of a particle. It drives the direction a particle will move through the search space, that is, causing the particle to make a turn.
- 'pbest' is the fitness value of the best solution yet encountered by a particular particle, and $P_i$ is a copy of the $X_i$ for the location that generated the particle's pbest. Jointly pbest and $P_i$ comprise the particle's memory and influence particle's movement to pull the particle towards a promising search region.
- $P_g$ is an important parameter in particle swarm. $P_g$ is the location of the particle that currently produces the best score in the neighbourhood. When the swarm is divided in small groups of particles then the neighbourhoods overlap and every particle is in multiple neighbourhoods. On the other hand when entire swarm is considered as single neighbourhood, $P_g$ defines the location of the global best particle.

The update equations for velocity and the position of each dimension of a particle are shown in (2.1) and (2.2) respectively, given a problem of $D$-dimensions, for each particle $i$ and each dimension $d$, $d = [1..D]$.

$$V_{i,d}(t+1) = \omega \times V_{i,d}(t) + \psi 1 \times ran1 \times (P_{i,d} - X_{i,d}(t)) + \psi 2 \times ran2 \times (P_{gd} - X_{i,d}(t)) \quad \textbf{(2.1)}$$

$$X_{i,d}(t+1) = X_{i,d}(t) + V_{i,d}(t+1) \quad \textbf{(2.2)}$$

where $V_{i,d}$ is the velocity of particle $i$ along dimension $d$, *ran1* and *ran2* are random values on the range $\{0..1\}$, $X_{i,d}$ is the current position of particle $i$ along dimension $d$, $P_{i,d}$ is the location along dimension $d$ at which the particle previously had the best fitness measure, and $P_{gd}$ is the current location along dimension $d$ of the neighbourhood particle with the best fitness. The constant $\omega$ is the inertia weight described by [Shi and Eberhart, 1998a]. A high value of $\omega$ gives a global search and a low value gives local search. $\psi 1$ is the cognitive learning rate, $\psi 2$ is the social learning rate. The relative influence of the particle's memory (cognitive influence) and the neighbourhood best (the social influence) can be adjusted. Together, these influences make up the learning rate of the swarm. Dropping the social component $\psi 2$ results in the Cognition-Only Model of the velocity equation:

$$V_{i,d}(t+1) = \omega \times V_{i,d}(t) + \psi 1 \times ran1 \times (P_{i,d} - X_{i,d}(t)) \quad \textbf{(2.3)}$$

Now dropping the cognition component defines the Social-Only Model of the the velocity equation: :

$$V_{i,d}(t+1) = \omega \times V_{i,d}(t) + \psi 2 \times ran2 \times (P_{gd} - X_{i,d}(t)) \quad \textbf{(2.4)}$$

Other PSO parameters:

- $X_{max}$ and $X_{min}$ (optional) set bounds for the search area.
- $V_{max}$ (optional) sets bounds on the velocity of a particle.

- THRESHOLD (optional) sets the acceptable error level. A solution falling within THRESHOLD distance of a specified value would be considered an acceptable solution and the search would be terminated.

```
Initialisation of swarm:
for each particle i, i = [1..Q]
for each dimension d, d = [1..D]
      set X_{i,d} to a random value on the range [X_min ..X_max]
      set P_{i,d} to X_{i,d}
      set V_{i,d} to a random value on the range [V_min ..V_max]
      compute fitness_i
      set pbest_i to fitness_i
      set g to i if fitness_i > fitness_g
Perform search:
until a terminating condition is met
for each particle i, i = [1..Q]
for each dimension d, D = [1..D]
      compute V_{i,d} Equation (2.1)
      compute X_{i,d} Equation (2.2)
      compute fitness_i
      update g if fitness_i > fitness_g
      update P_{i,d} pbest if fitness_i > pbest
Report results
```

**Figure 2.6 Pseudo-code for PSO**

- MAXITERATIONS (optional) sets a limit on the number of iterations to be executed before terminating a search.

The complete algorithm for the Particle Swarm Optimiser is listed in Figure 2.6. Search starts with the random initialisation of particles' positions and velocities within the allowed range defined by $X_{max}$, $X_{min}$ and $V_{max}$. Usually $V_{min}$ is the negative of $V_{max}$. Equations (2.5) and (2.6) are used to limit the magnitude of velocity and position.

To limit velocity of a particle along dimension $d$:

$$V_{i,d}(t+1) = \min(V_{\max}, \max(-V_{\max}, V_{i,d}(t+1))) \quad \textbf{(2.5)}$$

To limit position of a particle along dimension $d$:

$$X_{i,d}(t+1) = \min(X_{\max}, \max(X_{\min}, X_{i,d}(t+1))) \quad \textbf{(2.6)}$$

Each particle keeps track of its own performance. At each iteration, the velocity of every dimension of a particle gets updated according to equation (2.1), where $V_{i,d}$, $P_{i,d}$ and $P_{gd}$ constitute the particle's momentum. As this momentum is different for different dimension of a particle, this has effect to force the particle to change the trajectory in the search space towards the most promising areas. This momentum is essential, as it is the feature of PSO that allows particles to escape the local optima. In addition the *ran1* and *ran2* in equation (2.1) adds some random adjustments in velocities, which is essential to avoid the situation where the particle endlessly follows the exact same path.

### 2.6.3 Drawbacks of PSO

PSO like any other stochastic algorithm may prematurely converge [Løvberg, 2002]. Fast rate of information flow between particles can create similar particles resulting

in less diversity in the system., which increases the possibility of being trapped in local optima [Riget and Vesterstrøm, 2002]. PSO is also very much problem dependent like any other stochastic search. No single parameter setting exists which can be applied to all problems [Løvberg, 2002]. For example choosing a value for the inertia weight, $\omega$ in (2.1), could be critical. A large inertia weight favours exploration (global search), while a small inertia weight favours local search [Shi and Eberhart, 1998a]. Thus finding the best value for $\omega$ is difficult and it may vary from problem to problem. PSO's problem-dependent performance can be avoided by using self-adaptive parameters. In self-adaptation, the search process uses a feed mechanism to inform the system in favour of adjusting parameters depending on the problem [Løvberg, 2002]. Successful application of self-adaptation has been seen on GAs before [Back, 1992]. It has been applied to PSO as well in several occasions [Clerc, 1999; Shi and Eberhart, 2001; Ratnaweera *et al.*, 2003; Tsou and MacNish, 2003; Yasuda *et al.*, 2003]. Hybridisation also helps to combat premature convergence in PSO. Hybridisation refers to combining different approaches to benefit from the advantages of each approach [Løvberg, 2002]. Hybridisation has been successfully applied to PSO by [Angeline, 1998b; Løvbjerg and Krink, 2002; Kalyan *et al.*, 2003; Reynolds *et al.*, 2003; Higashi and Iba, 2003; Esquivel and Coello Coello, 2003].

### 2.6.4 PSO vs. GA

Both GA and PSO start with a group of random generated population. Also both of them use fitness values to evaluate population. However unlike GA, PSO has no evolution operators such as crossover and mutation rather particles update themselves with the internal velocity. PSO uses memory, which is important to the algorithm. The cognitive operator of PSO is personal best history and velocity inertia, but for GA it is mutation. The social operator for GA is selection and crossover where as for PSO it is neighbourhood best position.

Experiments conducted by [Kalyan *et al.*, 2003] showed that a PSO performed better than GAs when applied on some continuous optimisation problems. [Robinson *et al.*, 2002] found superiority of PSO over GAs in designing a difficult engineering problem. [Mesot, 2004] was able to generate efficient locomotion pattern for modular robots by using PSO and he found PSO delivered constantly better results than GAs. [Kalyan *et al.*, 2003] successfully combined PSO and GA to develop GA-PSO and PSO-GA. In GA-PSO, the GA population is used to initialise the PSO population. For PSO-GA, the PSO population is used to initialise the GA population. Kalyan's results revealed that both PSO and PSO-GA showed better performance than both GA and GA-PSO. In training neural network PSO also shows better results than GA [Eberhart and Shi, 1998b; Van den Bergh and Engelbrecht, 2000; Ismail and Engelbrecht, 2000]. [Shi and Eberhart, 1998b] found that the PSO is not sensitive to the initial swarm size, which means that PSO with smaller population size can perform comparably to GAs with larger population.

## 2.7 Summary

This chapter provided a brief overview of optimisation. An introduction of global and local optimisation has been given. This was followed by a brief discussion of traditional and stochastic optimisation methods. Traditional algorithms are also known exact methods. Brute force and Branch-and-Bound algorithms were introduced as examples of exact methods. Then introduction of three major stochastic algorithms were given. These were Hill-Climbing, Simulated Annealing and Tabu Search. Evolutionary algorithms (with more emphasis on genetic algorithms) were then presented. Different control parameters for GA were explained. Mechanisms to tackle premature convergence in GA were also mentioned. This is followed by an elaborated discussion of particle swarm optimisation and its various modifications. PSO equations and parameters were explained. References have been given for the

use of hybridisation to combat premature convergence in PSO. Finally a brief comparison of the PSO and GA has been provided.

# Chapter 3

# The n-tuple Classifier

---

## 3.1 Introduction

An n-tuple classifier is a memory-based method. It is a type of a neural network with a structure that could be easily implemented using a RAM (Random Access Memory). It forms the basis of a commercial product [Aleksander *et al.*, 1984]. The n-tuple method is more specifically known as a type of Weightless Neural Networks (WNN) or RAM networks (RAM-net). The following sections will introduce both the weightless and weighted approach on neural networks and later the n-tuple system will be elaborately described. The motivation behind the optimisation of n-tuple method will be explained also.

## 3.2 Artificial Neural Network

An Artificial Neural Network (ANN) – an abstract model inspired by knowledge of the brain's function – is a collection of interconnected elements that can learn to recognise patterns [Boone *et al.* 1990a; 1990b; Rich and Khight, 1991]. ANNs contain a large number of very simple, neuron-like processing elements (PE) and a large number of weighted connections between these elements. A PE is essentially an equation which is often referred to as a transfer function.

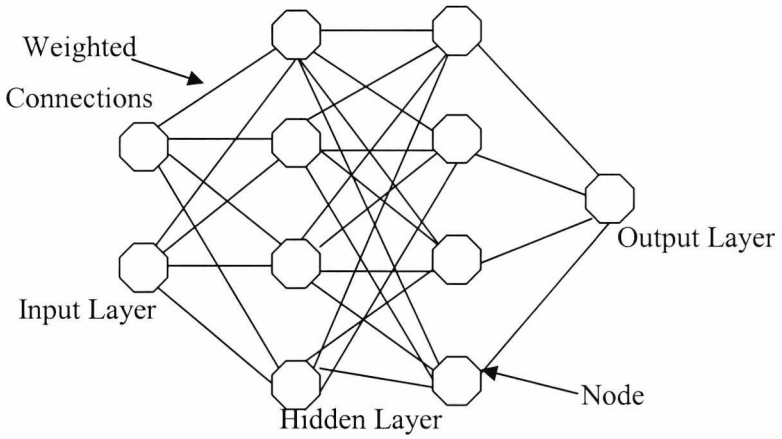



**Figure 3.1 Layers in ANN**

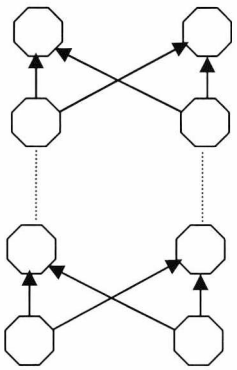


**Figure 3.2 Feedforward network**

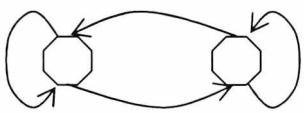


**Figure 3.3 Recurrent network**

A processing unit takes weighted signals from other neurons, possibly combines them, transforms them and outputs a numeric result. Many neural networks have their neurons structured in "layers". Layers are made up of a number of interconnected 'nodes' which contain an 'activation function'. Typically the PEs are arranged in layers (Figure 3.1); with the input layer receiving inputs from the real world and each succeeding layer receiving weighted outputs from the preceding layer as its input. Hence the creation of a feedforward ANN (Figure 3.2), where each input is fed forward to its succeeding layer. The first and last layers in this ANN configuration are typically referred to as input and output layers. Any layer between the input and output layers are called hidden layers because they do not have contact with any real world input or output data. Unlike a feed forward ANN, in a recurrent type network connections can go in either direction from all layers, Figure 3.3. Because of the feed back connection recurrent networks produce complex, time-varying outputs in response to simple static input which is important when generating complex behaviour. The architecture of an ANN is determined by the overall connectivity and transfer function of each node in the network.

### 3.2.1 Learning in an ANN

Most ANNs contain some form of 'learning rule' which modifies the weights of the connections according to the input patterns that it is presented with. In a sense, ANNs learn by example. Neural networks are "trained", meaning they use previous examples to establish (learn) the relationships between the input variables and the predicted variables by setting these weights. Once these relationships are established (the neural network is trained), the neural network can be presented with new input variables and it will generate predictions. The ability to identify the rules, to generalize, allows the system to make predictions. This property is known as the generalization of ANN. To simulate intelligent behaviour the abilities of

memorization and generalization are essential. Learning in ANNs can roughly be divided into supervised, unsupervised and reinforcement learning. Supervised learning is based on direct comparison between the actual output of an ANN and the desired correct output, also known as the target output. Back propagation [Hinton, 1989] is one of several possible learning rules to adjust the connection weights during learning by example. Learning occurs when the network weights are adjusted as a function of the error found in the output of the network. The error is the difference between the expected output and the actual output. The weights are adjusted backwards (back-propagated) through the ANN network until the error is minimized for a set of training data. In reinforcement learning the exact desired output is unknown, but it gets the information of whether the actual output is correct or not. Unsupervised learning even doesn't get the information on correct output. It is solely based on the correlations among input data. The algorithm of learning rules determines how the connection weights are changed. Among the popular learning rules there are delta rule, Hebbian rule, the anti-Hebbian rule and competitive learning rule [Hertz *et al.*, 1991].

### 3.2.2 Weightless Approach

The weightless neural networks use explicit storage elements to keep its state, rather than in its inter-element connections, as more conventional networks do. In weightless approach there is no variable weight between the nodes rather neuron functions are stored in look-up tables. The learning algorithm is very simple, the patterns are presented to the inputs of the network and then the patterns are stored in a certain way, which results in highly flexible and fast learning algorithms. In weighted models, training is much more complex since changing weights to train a given input-output mapping changes the nodes behaviour to other patterns learned

previously. The very first work in this field is described by [McCulloch and Pitts, 1943; Hebb, 1949 and Rosenblatt, 1958].

From the experimental studies presented in [Ludermir *et al.* 1999; Christensen *et al.*, 1996; Hepplewhite and Stonham, 1997; Jørgensen, 1997; McCauley *et al.*, 1994; Ramanan *et al.*, 1995; Rohwer and Morciniec, 1996; Wang *et al.*, 1996] the effectiveness of WNN models can be realised. Since the original work by [Bledsoe and Browning, 1959] many weightless models were proposed such as the Probabilistic Logic Node (PLN) by [Aleksander, 1989]. Aleksander also proposed the concept of multi-layer architecture [Kin and George, 2005]. Some of the other WNN models include the probabilistic RAM (*p*RAM) [Austin, 1994], Goal Seeking Neuron (GSN) [Austin, 1998], Boolean Convergent Network (BCN) [Howells *et al.*,1995], Generalised Convergent Network GCN [Howells *et al.*,1995], Probabilistic Convergent Network(PCN) [Howells *et al.*,1995], Moving Window Classifier [Hoque, 2001] and Deterministic Adaptive RAM Network (DARN) [Yee and Coghill, 2004].

## 3.3 An n-tuple Classifier

Although the n-tuple classifier is not famously popular compared to some other methods, such as multilayer perceptrons, the n-tuple classifier does have its own advantages over a variety of pattern recognition algorithms [Rohwer and Morciniec, 1998]. The networks based on the n-tuple method have two great strengths, they can be trained quickly and they can be implemented in conventional computers simply compared to other equation solving and minimising methods. The training of the basic classifier is a one-shot memorisation process. These advantages come at the cost of recognition robustness. It has been shown that the n-tuple method can result in quite reasonable recognition performance if used with care [Rohwer and Morciniec, 1998].

An n-tuple classifier is based on conventional random-access memory (RAM). The network is built out of RAM nodes and consists of a set of discriminators, each representing a class to be learned/recognised. Figure 3.4 shows a

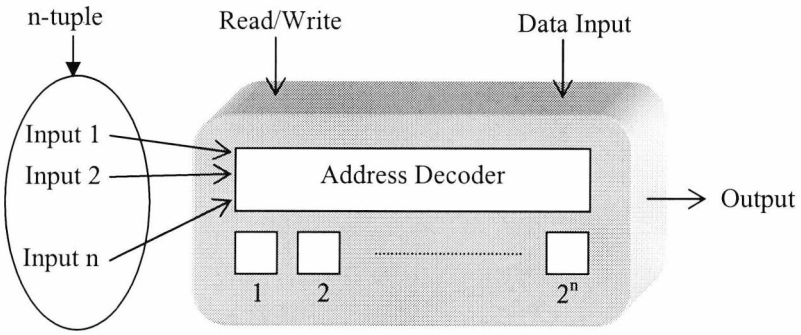



**Figure 3.4 1 bit Ram Node**

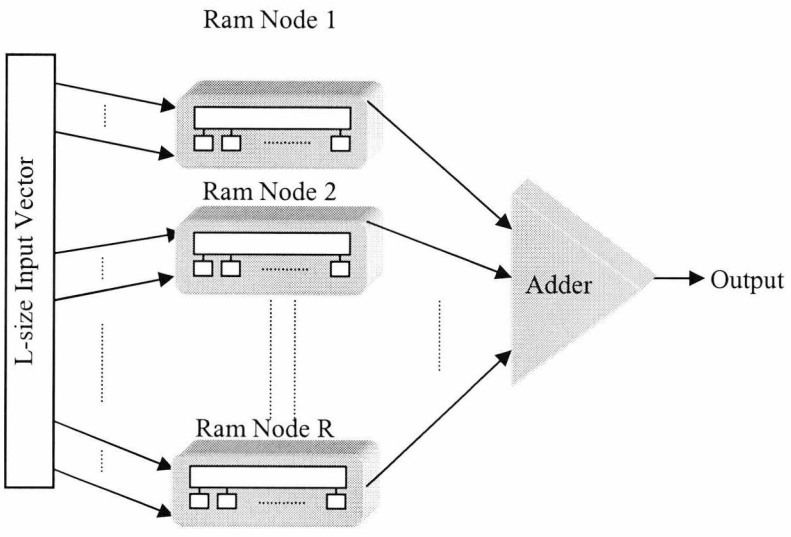



**Figure 3.5 A Discriminator**

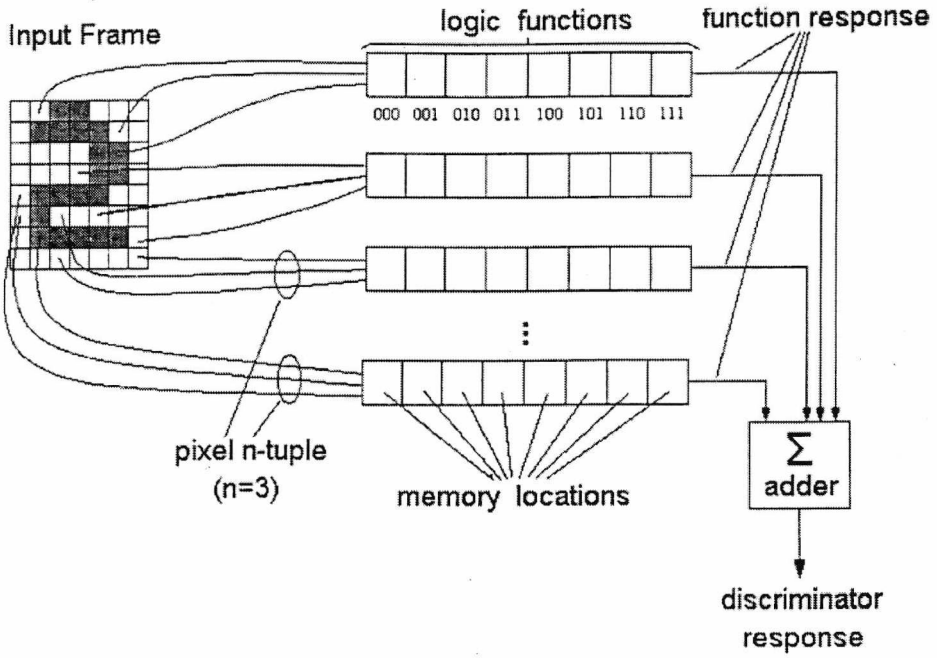



**Figure 3.6 A discriminator illustrated by [Tambouratzis, 2000]**

representation of a typical RAM node. The 1-Bit-RAM node is a device which can store one bit of information for each input address. The input address of the RAM unit is also known as "tuple". If the width of the address bus (also known as input connection map) is *n* bits (as shown in Figure 3.4) then the tuple is termed as "n-tuple". The width of the address bus is also known as "tuple-size" [Bledsoe and Browning, 1959]. A control input is available for switching the mode between `Write' and `Read' for learning and recall. Initially all memory units are set to `0'. During the learn (`Write') mode the memory is set to `1' for each supplied address; in the recall (`Read') mode the output is returned for each supplied address, either `1' (if the pattern was learned) or `0' (if the pattern was not learned). A RAM node is limited to learn binary patterns (which is the memory's address word), its output is also binary.

A group of RAM nodes in a tree-like structure is called a discriminator (Figure 3.5). The discriminator achieves its goal by presenting to each neuron only a subset of the input pattern, and adding up the outputs of its RAM nodes. This sum can be seen as a measure of the recognition confidence of the discriminator. Therefore, when the discriminator sees a previously learned pattern, its integer output reaches the discriminator's maximum. For an input vector, of size $L$, the number of necessary RAM nodes $R$ of connectivity $n$ that should be used to cover all inputs of the input vector should satisfy: $R \times n \geq L$. $L$ is known as the resolution of an image. If $W$ and $H$ denote the width and height of an input image, then image resolution, $L$, will be given by the following formula:

$$\textit{Image Resolution, } L = W \times H \qquad \textbf{(3.1)}$$

The $n$ bits of a tuple constitute a "feature" of the pattern. Collectively $R$ set of n-bit patterns are called "input mapping" or "connectivity pattern" denoted by $\eta$ [Rohwer and Morciniec, 1996]. $\eta$ comprises all the information from the $L$ bits input pattern available to the recogniser. Figure 3.6 illustrates how a group of tuples or a discriminator is connected to a binary picture of a digit. This particular example was used in [Tambouratzis, 2000] where the n-tuple size was 3. The resolution of the image is 56 bits according to equation ( 3.1). The figure shows just one discriminator There are eight memory locations correspond to $n$=3. An adder has been shown in Figure 3.6 to combine the responses of individual tuples in the group. The number of classes, which need to be distinguished by a network, determines the number of discriminators needed in a network. The network shown in Figure 3.7 can be used to distinguish a fixed number of classes. If it consists of $C$ discriminators, it can differentiate $C$ classes. The memory size required by the network will be given by equation ( 3.2), where $R$ is the total number of available tuples with tuple-size of $n$ bits and every memory location addressed in a tuple will have 1 bit reserved for each discriminator.

$$Memorysize = R \times 2^n \times C \quad (3.2)$$

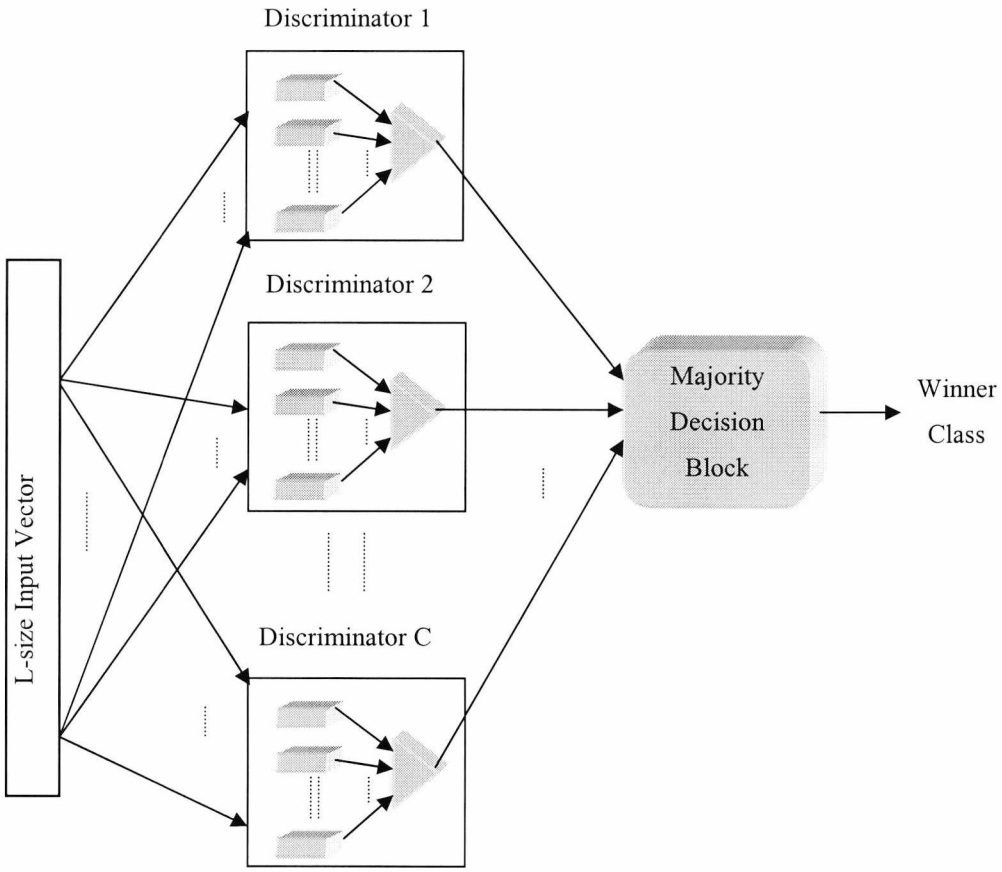



**Figure 3.7 An n-tuple network**

The Majority Decision Block (MDB) [Jørgensen *et al.*, 1995] at the outputs of the adders chooses the winner class using some criteria such as the greatest sum, a threshold of the greatest sum, difference between sums etc. In greatest sum approach, the discriminator containing the greatest number of active RAM nodes is selected. Thus a pattern is 'recognized' as the one whose discriminator 'fired' the most, that is, the discriminator with the highest count of memorized tuples. Two measures of

confidence can be used [Bishop, 1989; Mitchell and Minchinton, 1996], Absolute Confidence $AC$ and Relative Confidence $RC$:

$$AC = \frac{Most\ Number\ of\ 'fires' - Next\ highest}{Number\ of\ Tuples} \quad (3.3)$$

$$RC = \frac{Most\ Number\ of\ 'fires' - Next\ highest}{Most\ Number\ of\ 'fires'} \quad (3.4)$$

### 3.3.1 Architectural Parameters

The n-tuple network has the ability to memorise and generalise. Memorization is an obvious task in learning. This was implemented by storing the input samples explicitly. Generalization allows the system to make predictions on unknown data. Generalization can dramatically reduce the amount of memory needed, and produce a very efficient method of memorization. In equation ( 3.2) the exponential relationship makes the choice of $n$ very sensitive. The larger the n-tuples selected, the fewer the number of tuples required to cover the entire image. It can be noted that as $n$ increases the memory requirements by the network also increases. This imposes a physical limit to the value of $n$. For $n=L$ the system would have a single huge impractical memory and wouldn't be able to generalize i.e. be able to recognise patterns that were not exactly like those taught. On the other extreme if $n=1$, all the locations in the memory are filled with 1s quickly and the network saturates and loses its discriminative power [Ullmann,1969; Tarling and Rohwer, 1993]. Thus when larger value of $n$ is chosen saturation becomes less problem but with a very large value of $n$ the system loses it ability to generalize.

It has been found empirically that for a given size of training set, there is an optimum value $n$ which will give maximum performance [Aleksander and Stonham, 1979]. [Ullmann, 1969] showed how the percentage of correct classification is

increased to a peak with the increase in value of $n$. After the peak the percentage of correct classification falls with a further increase in $n$. For larger training sample a higher peak was found for a larger value of $n$ but the shape of the curve remained the same. [Hoque, 2001] demonstrated the performance of the n-tuple classifier for different training set sizes taken from the NIST numeric database [Wilkinson *et al.*, 1992]. Figure 3.8 shows the graphical plot of the reported results in [Hoque, 2001]. It depicts the relationship between the number of train images and the recognition rates of an n-tuple network with the tuple-size 12. The rise in accuracy with increase in training patterns was significant at 1000 training images per class.

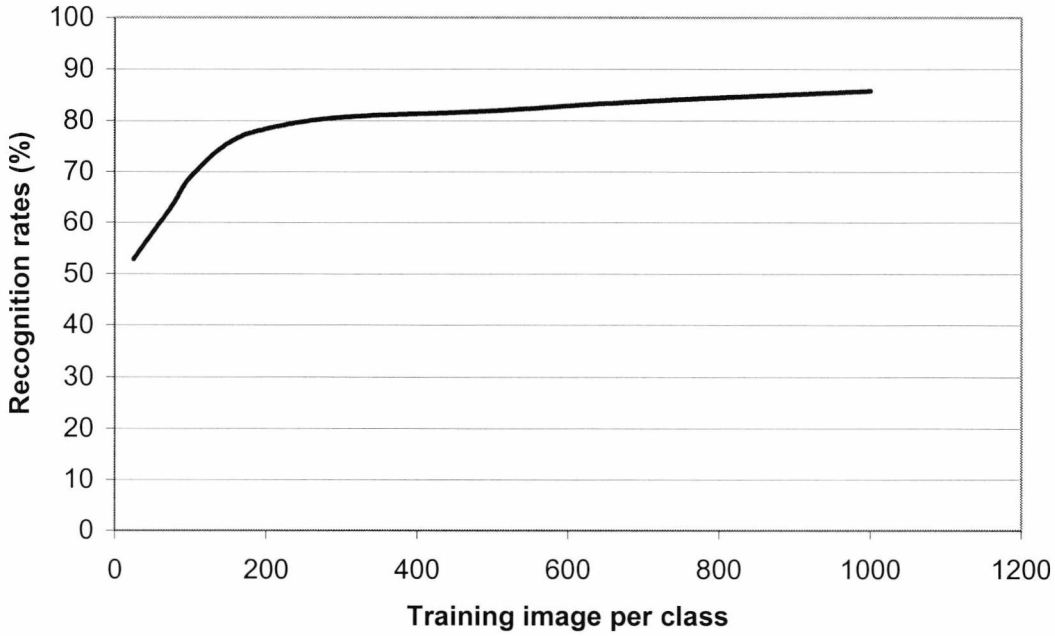



**Figure 3.8 Performance of n-tuple network as a function of training set size**

Input connection mapping ($\eta$) is another important parameter for n-tuples. Conventionally input mappings are randomly chosen [Bledsoe and Browning, 1959]. It has been demonstrated in [Picton, 2000] that a randomly connected system perform better than a network with an ordered map. Orderly fashioned input connection failed

because the patterns being discriminated were very similar to the way the system was organised. Random connection was favourable because randomness doesn't have a pattern with it. [Fairhurst and Stonham, 1976] have shown that the n-tuple scheme is relatively insensitive to the connection mapping. However [Aleksendar and Stonham, 1979] have argued that a random map is suitable for an un-optimised problem because sampling points distributed throughout the pattern matrix are more likely to detect global features than an ordered map. For an optimised case better selection of input mappings can give a relatively better performance [Aleksander and Stonham, 1979]. [Bishop *et al.*, 1990] demonstrated the importance of sampling sequence in discriminating similar classes. They illustrated the fact with two 16-bit data pattern as shown in Figure 3.9. If four 4-bits tuples are used and each tuple was formed from each column three of the tuples would be identical, so 75% pattern would be same. But if tuples are formed by taking bits from each row then none of the tuples are identical and it helps to discriminate the patterns.

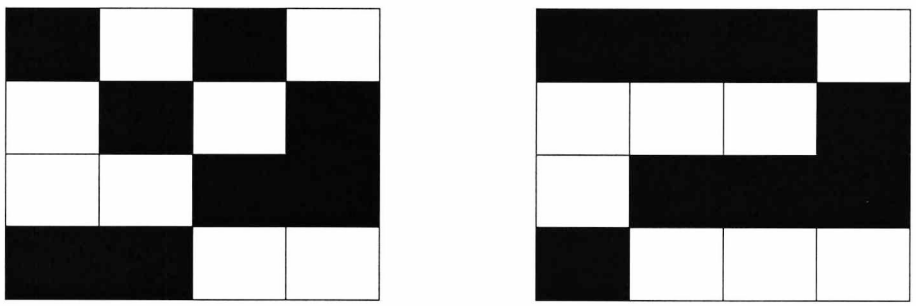


**Figure 3.9 Effect of input mapping shown in [Bishop *et al.*, 1990]**

## 3.4 Motivation for Optimisation

The classification performance of the n-tuple classifier is highly dependent on the input bits probed [Bishop *et al.*, 1990; Jørgensen *et al.*, 1995]. The number of possible connections for a 32 by 32 binary pattern matrix is enormous. Let us consider an n-tuple classifier of 140 tuples with the tuple-size 8 bits. For an input binary image with a resolution of 32 bits by 32 bits the total number of available pixels will be, from equation ( 3.1), 1024 bits. Now if the same pixel doesn't repeat in a tuple then the possible number of tuples that can be formed is found by the formulae of combination, $M = {}^{1024}C_8 \approx 2.91 \times 10^{19}$. $M$ tuples when divided into groups of 140 then total number of combinations will be $B = {}^{M}C_{140}$, which is a very large number. Therefore an exhaustive search for $B$ mappings is impossible.

The classification performance is a function of input mappings and it approximates to a normal distribution [Aleksander and Stonham, 1979], where the majority of the mappings give average performance, but a small number of connection mappings give a relatively better performance. Figure 3.10 demonstrates the classification histogram for a subset of the NIST [Wilkinson *et al.*, 1992] database where two thousands maps were generated randomly and the frequencies of maps for recognition rates were plotted. If all $B$ mappings as explained earlier were available, the tail on the right side of the histogram would go much further as shown by the imaginary dotted area in Figure 3.10. Finding mappings in this dotted optimisation zone is extremely challenging. The detection of mappings in this area by random search is governed by chance.

[Bishop *et al.*, 1990] demonstrated the importance of choosing right sequence in which input is sampled to discriminate similar classes. For example, consider an n-tuple system which is trying to recognise and classify 8 × 8 images of characters and for which the tuple size is 8. Figure 3.11 and Figure 3.12 show two similar characters, for *c* and for *e*, for which there are 8 pixels which are different and there are 8 tuples

to sample the image. Now consider a situation where the eight different pixels are sampled and formed into one tuple and the other tuples are formed by the pixels which are common to both the letters. This way of the eight tuples used to sample the image, seven will be identical. Therefore, if an *e* is presented to the two discriminators, all eight neurons in the 'e-discriminator' will fire, and seven neurons in the 'c-discriminator will fire. If, however, one sample of each tuple came from the area where the pixels differ, then all eight tuples will be different, so all eight neurons in the 'e-discriminator' will fire, and no neuron in the 'c-discriminator' will fire.

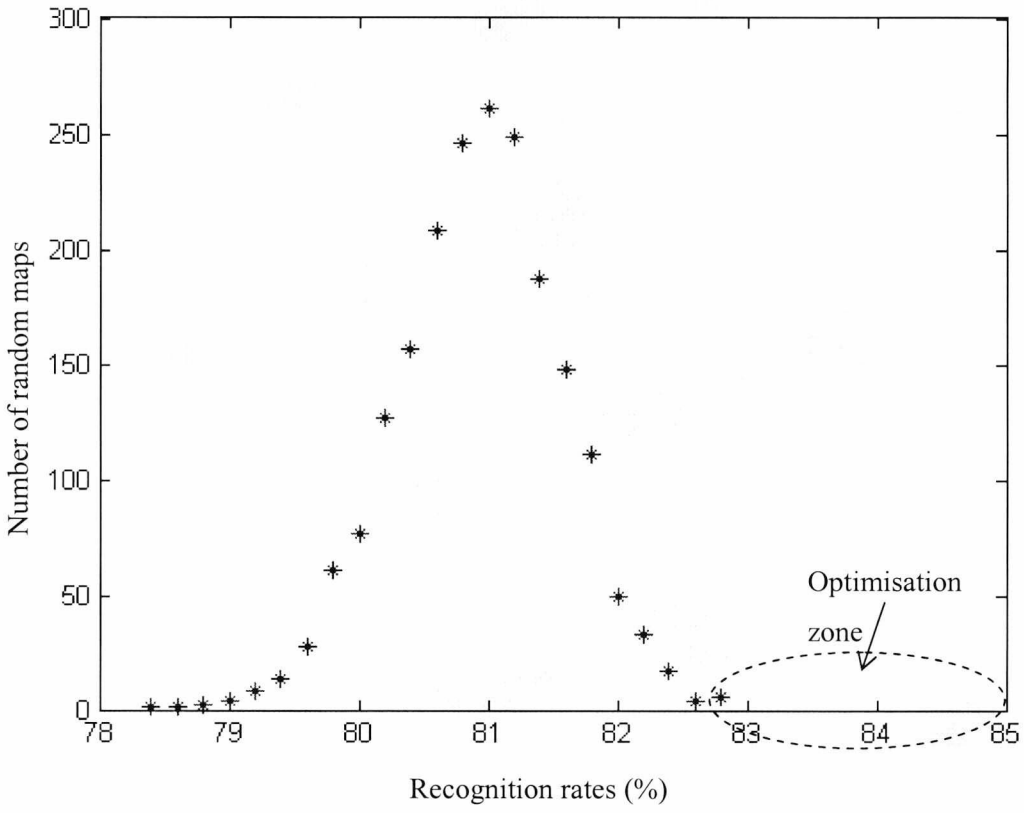



**Figure 3.10 Classification histogram using NIST database, 140 tuples of tuple-size 8**

Thus the sequence in which the input is sampled and the tuples formed can have a significant effect when discriminating between similar classes. When the letters being analysed are very different, for example, an *E* and an *i*, then the system will have little difficulty discriminating between the classes, irrespective of the sequence in which the input is sampled. Therefore, when attempting to discriminate between many classes, some of which are similar, the input should be sampled in one sequence for some class discriminators, but in other sequences for other class discriminators.

[Bishop *et al.*, 1990] applied a basic evolutionary technique to determine the sequence in which the input is sampled and the tuples formed. One sequence of input samples was used for the discriminators for most of the characters, but different sequences were used for the discriminators of characters which are too similar, such as *c* and *e* (Figure 3.11and Figure 3.12), *i* and *l*.

The solutions suggested by [Bishop *et al.*, 1990] were the sequence of samples to increase the orthogonality of response of each discriminator. A mutation technique was used to form tuples. The performance of the system with this mutation was evaluated by measuring the relative confidence ( 3.4) and a decision was made if the new mapping should be adopted. The mutation was realized by swapping addresses used to sample the data patterns so as to form the tuples. After Bishop's work genetic algorithm was revisited by others more recently [Garcia, 2003; Farhan-Khola and Howells, 2003] to optimise the input connections of n-tuple network. Other stochastic search algorithms like Tabu search and Simulated Annealing (described in Chapter 2) were applied by [Garcia and Souto, 2004] to choose the connectivity pattern of n-tuple classifier.

The backtracking method [Ellis and Sartaj, 1984; Thomas *et al.*, 1990] has been found to be more superior than any exhaustive search or random selection method as it strives to eliminate unviable solutions. The process of going back to the

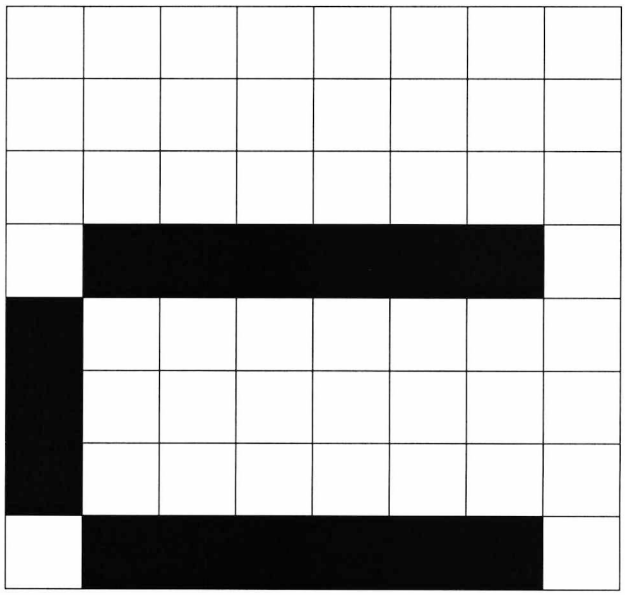


**Figure 3.11 Image of character c similar to e**

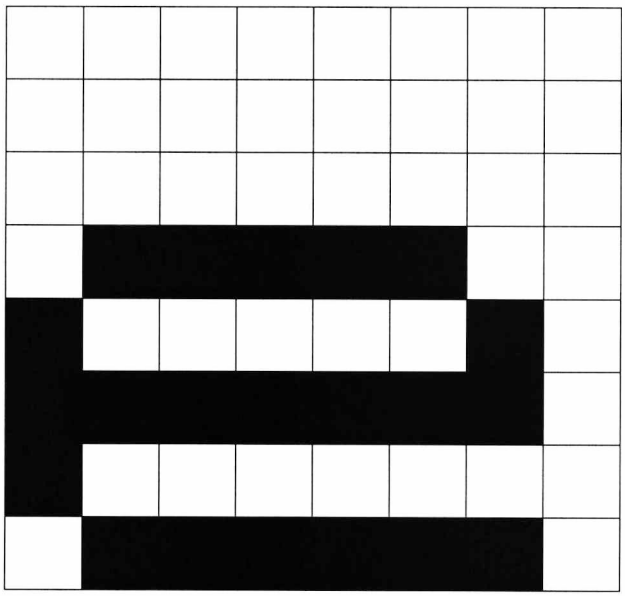


**Figure 3.12 image of character e similar to c**

nearest decision facilitates to choose new component in solution vector and this leads to the name backtracking. [Jung *et al.*, 1996] used backtracking algorithm to generate distinguishing tuples for selected character dichotomies. Initially they tried generating tuples at random. But random selection method found no tuples for more difficult dichotomies like *c* and *e*.

[Jørgensen *et al.*, 1995] described a simple input selection strategy that makes use of a leave-one-out cross-validation test [Hand, 1986]. It involves using a single observation from the original sample as the validation data, and the remaining observations as the training data. This is repeated such that each observation in the sample is used once as the validation data. [Jørgensen *et al.*, 1995] introduced an information measure (denoted the cogentropy) to evaluate the quality of a given combination of tuples. The concept of information measure was combined with the cross-validation and was found advantageous to obtain an n-tuple network with a better performance than a randomly trained network of the same size.

The explanation and examples mentioned above confirmed that it was favourable to optimise input connections of the n-tuple network. Improvement in performance due to optimisation was noticeable [Bishop *et al.*, 1990; Jørgensen *et al.*, 1995; Jung *et al.*, 1996]. It provided insight to this research to investigate other methodologies to improve the connectivity pattern of the n-tuple classifier. The proposed approach was based on particle swarm intelligence. In the selection process of the algorithm the performance of each tuple was measured with a reward and punishment based scheme [Azhar and Dimond, 2004a]. Each tuple associates a memory of its own performance. The strategy was to keep the best-performed configuration over a lifetime of a tuple. Once the target set of best-performed configurations are sought the optimal set of tuples are exploited for the final recognition task.

## 3.5 Summary

This chapter provided an introduction of Artificial Neural Networks with the focus on the weightless approach. A memory based n-tuple network has been chosen in this research for optimisation, so the architecture of this network has been described in detail. Both the training and learning algorithms of n-tuple networks have been explained. It has been shown that the connectivity pattern of the n-tuple system plays important role in selecting features of a pattern class. In the traditional method the connectivity pattern is defined at random and then fixed training. Choice of the right sampling sequence can improve the discrimination power of the network considerably. Considerable research presented in this chapter showed that it was promising to optimise the connectivity pattern of n-tuple networks for improved recognition performance. Being successfully applied in many other areas particle swarm intelligence has its merits in selecting important features of the input patterns. As in PSO the connectivity pattern is not defined at random, the RAM nodes will not be uniformly distributed along the input vector. Thus the PSO increases the probability of many RAM nodes to be connected to relevant features of the input vector. This chapter presented these arguments and motivations behind the proposed optimisation choices and also the theory of the chosen n-tuple network for this research.

# Chapter 4

# Experimental Framework

## 4.1 Problem Definition

The effectiveness of an algorithm was tested through experiments. Experiments were set-up to train an n-tuple network with a modelled algorithm and then to test if the training improves the recognition performance of the network. For the training and testing purpose a database were required. Selection of standard database for the experiment was very crucial. Section 4.2 will describe the arguments behind choosing a specific database for the experiments. Database contains different classes that will be recognized by the network. In a traditional n-tuple network there will be specific number of tuples and these will be connected to the input image through random connections. Number of tuples in the network will define the architecture of the n-tuple classifier. Images for the experiment will be handwritten digits in binary form. So a connection between a location of an image and the tuple will carry one bit of information.

Optimisation algorithms will be used to find better input connections to the tuples so that tuples get connected to the important areas of an image. The purpose of the experiments will be to find an optimal set of input connections to the tuples. It will be extremely rare to find a connection to a tuple such that the connection is

equally good for all different types of images or classes. This is because there will be similar classes in the system and the overlapping region between similar classes will cause the discrimination task much harder. There will be classes in the dataset which will have complicated pattern and it will be difficult for the classifier to recognize this pattern. This difficult class of images can be termed as a critical class. It is logical to say that the discriminating power of a critical class can be improved if enough tuples are available to connect to the important featured areas of a class. So for a better recognition of a critical class it is desirable to provide more tuples than the number of tuples required for a non-critical class. The strategy was to use the optimisation algorithm to tune more tuples for a critical class than a non-critical class.

A tuned tuple is known as a class-specific tuple, which best describes a specific class but also describes other classes to some extent. Thus there will be a number of class specific tuples for different classes. These groups of tuples will try to improve the recognition rates of their own specific classes as well as other classes and eventually by working together all these tuples will improve the overall recognition rate of the images. So clearly there will be different number of class-specific tuples for different classes. In the experiments these numbers were calculated by finding error rates of a randomly connected n-tuple classifier. At first a traditional randomly connected classifier was used to find the error rates of different classes. Then total available tuples were divided into different classes proportionately to the error rates. This assumes linear relationship and this assumption is the first approximation. So the class with the most error rate gets the most number of tuples and the class with the least error rate gets the least number of tuples. All experiments for optimisation task will find these optimum set of class specific tuples and once found the whole set will be used to recognise the test data set. Once recognition rates are found, the experiment will be repeated several times to facilitate statistical significance testing (Section 4.4) and to check if the results are consistent. Later a statistical plot (Section

4.5) will help to visually summarize the distribution of recognition rates. Reasons for choosing specific n-tuple size have been given in Section 4.3.

## 4.2 Database Selection

Standard databases have become very important to facilitate research in character recognition [Guyon *et al.*, 1997]. They are an essential requirement for the development, evaluation and comparison of different character recognition techniques. Databases can be collected in a laboratory environment in which subjects prepare samples on standard forms that are then digitised. But the awareness by the subjects of the use of their handwriting may introduce biases into the data. In light of this, an acceptable character image database should be produced from real world environments so that the writings are truly unconstrained and more representative. Some examples of such publicly available database can be found in [CEDAR CDROM-1, CEDAR CDROM-2, ERIM, NICI]. The database used in this research was a fairly large real life database compiled by U.S. National Institute of Standards and Technology, and is often popularly known as NIST database [Wilkinson *et al.*, 1992].

### 4.2.1 NIST Database

NIST released Special Database 3 (SD3) in February 1992 as the official training materials for the First Census Optical Character Recognition (OCR) Conference [Wilkinson *et al.*, 1992]. The conference discussed the performance of 45 OCR systems submitted by 26 academic and industrial organizations. SD3 was included of a CD-ROM distributed by NIST and the CD-ROM contains images of 3699 Handwriting Sample Forms (HSFs) and 814255 segmented handprinted digit and alphabetic characters from those forms. An example HSF form can be found in Appendix B. There are several partitions, denoted by *hsf_{0,1,2,3}*, in SD3 containing digits, upper and lower images. The writers of the SD3 partitions were Census Bureau

field personnel stationed throughout the United States. A separate partition of images, denoted by *hsf_{4}*, was released as the testing materials for the OCR conference and it was named as Special Database 7 (SD7) [Wilkinson, 1992]. Images of SD7 were obtained from 500 HSF forms completed by high school students in Bethesda, Maryland. Thus the training set and the test set used in the OCR conference were representative of different distributions: the training set consisted of characters written by paid US census workers, while the test set was collected from characters written by high school students. Examples from these training and test sets are shown in Figure 4.1 and 4.2. Notice that the test images contain some very ambiguous patterns. The general conclusion of the conference was that the testing images of *hsf_{4}* are more difficult, in a recognition sense, than the images of *hsf_{0,1,2,3}*. This was later demonstrated in [Grother, 1993].

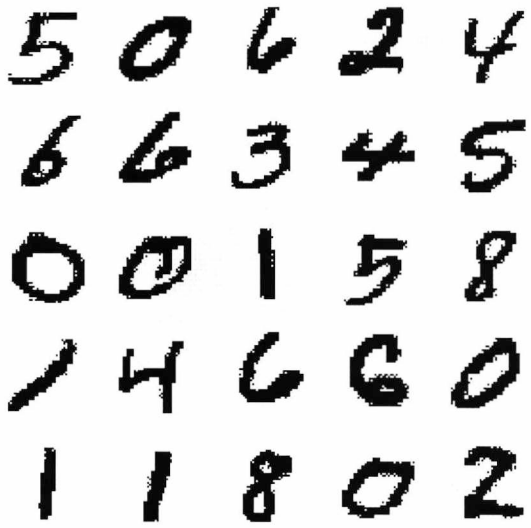


**Figure 4.1 Typical images from NIST training set**

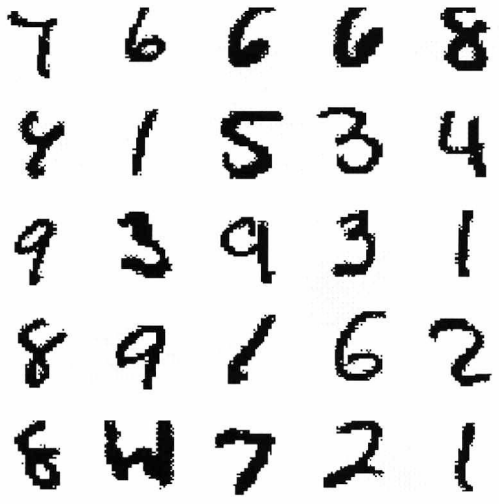


**Figure 4.2 Typical images from NIST test set**

In this research the partition *hsf_{0}* of the Special Database 3 was used for training and the partition *hsf_{4}* of the Special Database 7 was used for testing. NIST recommends using images of *hsf_{4}* for testing as they are more difficult from other partitions and this ensures the heterogeneity between the training and testing set (see analysis in [Wilkinson *et al.*, 1992]), a fact which is reflected in the results presented in this thesis. The numeric data set consisting only the digits (0,1...9) was used for the experiments. Each character is a binary image with the dimension 32 by 32. All digits are scaled into same dimension and centred. In the partition *hsf_{0}* there are 1000 images per class for training and in the partition *hsf_{4}* there are 1000 images per class for testing.

## 4.3 Tuple size and *R*

The performance of the n-tuple method depends mainly on the size of the n-tuple chosen [Aleksander and Stonham, 1979; Rohwer and Lamb, 1993; Ullman, 1969; Ullman and Kidd, 1969]. [Hoque, 2001] performed an experiment to compare the accuracy of n-tuple classifiers as the tuple size varied from 2 to 16. Numeric data

consisting of digits were used from three different databases. Results in the Figure 4.3 employed random connection mappings and correspond to the mean of several test runs. DB1 in the figure was extracted from machine printed postcodes supplied by British Post Office. The DB2 contains images extracted from envelops of British mail. Both the database contains 300 binary images of each character. DB3 was the NIST database with 1000 images per character as described in the previous section. DB1 and DB2 showed peak accuracies around $n=10$ where as DB3 showed peak accuracy around $n=12$. Hoque's work [Hoque, 2001] confirmed Ullmann's [Ullmann, 1969] explanation about the relationship between the recognition performance and the value of $n$. Being the largest database DB3 needed larger n-tuples because, when the number of training examples increases it generally becomes more difficult to find features of low dimension that can distinguish between examples of different classes.

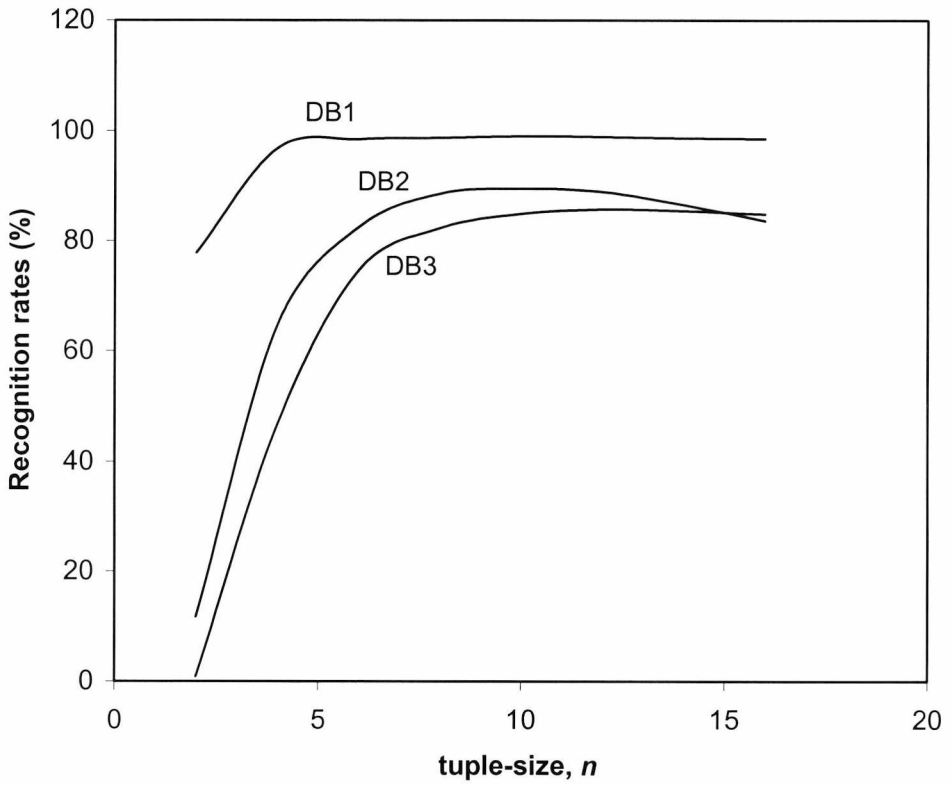



**Figure 4.3 Performance of n-tuple network as a function of n-tuple size**

The value of $n$ also plays important role determining the hardware memory space required ( 3.2) by the classifier. A balance has to be made between the memory size, the amount of training data and the value of $n$. Experiment results, for instance, showed that a bigger size $n$ is better, until it approaches an impractical large size, though a value of $n$ =8 turned out to be enough for many applications [Rohwer and Lamb, 1993]. [Jørgensen, 1997] made use of leave-one-out cross-validation (CV) [Hand, 1986] and found $n$=8 as the smallest tuplesize for the lowest CV error. [Rohwer and Morciniec, 1996] suggested 8 as a good choice for most data sets. For chosen database in this research $n$=8 gives reasonably good recognition performance of 80.93 [Azhar and Dimond, 2004a] with 150 tuples ($R$). [Rohwer and Morciniec, 1996] argued that the recognition rates should become increasingly consistent with increase of the total number available tuples, $R$. From practical experience they found that the values of 100 to 1000 for $R$ usually turned out to be adequate. [Jørgensen, 1997] reported error rates on the task of recognising handwritten digits for different values of $R$ and found better results for a higher value of $R$. Reported error rates with 935 and 807 tuples were 3.6% and 2.8% respectively. While choosing the value of $R$ the classification time has to be also looked at. [Jørgensen, 1997] found classification error rates with 807 tuples to be 30 ms and with 200 tuples to be 4 ms while running the code under Windows NT in a 90 MHz Pentium. The physical memory requirement of the network also limits the choice of $R$. With 150 tuples, tuple size of 8 and a 32 by 32 binary image the required memory will be 384000 bits (3.2). For a fixed $n$ the value of $R$ has to satisfy the relation $R \geq L/n$ (Section 3.3), so that enough tuples are available to cover the whole input matrix of size $L$.

## 4.4 Significance Testing

Once recognition performance have been gathered through experiments, statistical inference will allow us to assess evidence in favour or some claim about the

population from which the sample of recognition rates has been drawn. The methods of inference used to support or reject claims based on sample data are known as tests of significance. Significance testing is necessary where data is gathered from a sample and not from the entire population. Significance testing tells us how confident we can be that the survey sample accurately reflects the views of the entire population. A significance level is the probability that the result is true and not just a random variation. The t-test is a form of significance testing.

### 4.4.1 Student's T-test

A t-test decides if the two data sets come from the same population (Case I in Figure 4.4) or from different populations (Case II in Figure 4.4). The t-test measures the likelihood that two results being compared could have been found purely by chance. It does this by comparing the mean value of two sets of data. The difference between two means is normally distributed for large samples.

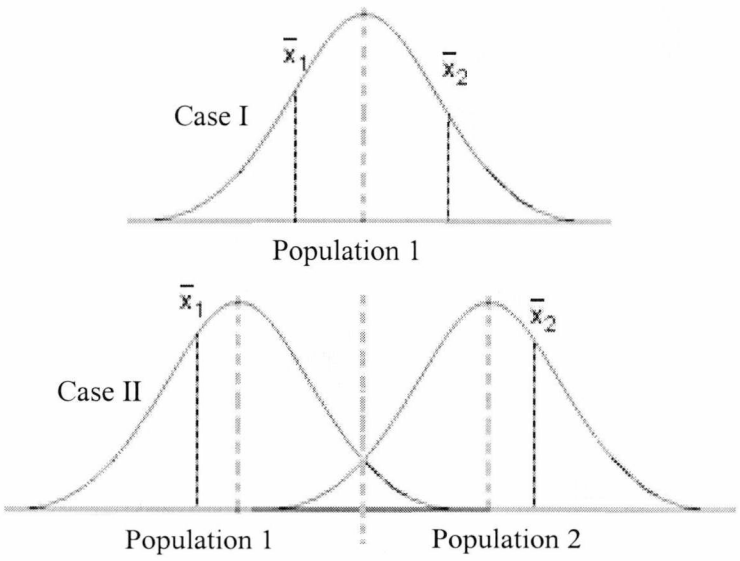



**Figure 4.4 Cases in t-test**

The t-distribution approximates this normal distribution in large samples. For small samples, the distribution of differences in the mean is not quite normal. This discrepancy was noted by a quality control statistician at Guinness Brewing (W.S. Gossett) [Porter, 1986]. Since quality control could involve only small samples, the statisticians required a test statistic that performed well for small samples. The t-distribution was widely used after this insight. Gossett and Pearson worked together for a short time and published their findings in 1896 (correlation/Pearson), 1900 (chi-square/Pearson) and 1908 (t-distribution/Gossett) [Porter, 1986]. However, because the brewery did not allow employees to publish their research, Gossett's work on the t-test appears under the name "Student".

Every test of significance begins with a null hypothesis $H_0$. A hypothesis is a statement designed to be proven or disproven. Null hypothesis says there is no difference between the means. An alternate hypothesis, $H_A$, is also set up, which is the hypothesis to be adopted if the null hypothesis is disproved. Case I in Figure 4.4 represents the null hypothesis $H_0$, indicating that the mean of group one equals the mean of group two; both samples come from the same population. Case II represents the alternate hypothesis $H_A$, indicating that the mean of group one does not equal the mean of group two; the two sample means are from different populations. A t-test decides which of these hypotheses to accept.

T-test assumes that the data are independently sampled from a normal distribution. The two means and the corresponding standard deviations are calculated by using the following equations ($n_A$ and $n_B$ are the number of measurements in data set A and data set B, respectively):

$$\overline{x_A} = \sum_{i=1}^{n_A} x_i \Big/ n_A \qquad \textbf{(4.1)}$$

$$\overline{x_B} = \sum_{i=1}^{n_B} x_i \Big/ n_B \qquad (4.2)$$

$$S_A = \sqrt{\frac{\sum_{i=1}^{n_A} \left(\overline{x_A} - x_i\right)^2}{n_A - 1}} \qquad (4.3)$$

$$S_B = \sqrt{\frac{\sum_{i=1}^{n_B} \left(\overline{x_B} - x_i\right)^2}{n_B - 1}} \qquad (4.4)$$

Then, the pooled estimate of standard deviation $S_{AB}$ is calculated:

$$S_{AB} = \sqrt{\frac{(n_A - 1){S_A}^2 + (n_B - 1){S_B}^2}{n_A + n_B - 2}} \qquad (4.5)$$

Finally, the statistic $t_{exp}$ (experimental t value) is calculated:

$$t_{\exp} = \frac{\left|\overline{x_A} - \overline{x_B}\right|}{S_{AB}\sqrt{\frac{1}{n_A} + \frac{1}{n_B}}} \qquad (4.6)$$

$t_{exp}$ value is compared with the critical or theoretical $t$ value, $t_{th,}$ corresponding to the given degree of freedom, '*df*' (in the present case $df = n_A + n_B - 2$) and the confidence level chosen. The confidence level is the percentage likelihood at which

the test was carried out. Tables of critical $t$ values can be found in any book of statistical analysis, as well as in many quantitative analysis textbooks. If $t_{exp}$ exceeds the tabled value, the means are significantly different at the confidence level that is listed. $t_{exp}>t_{th}$ means that the null hypothesis, $H_0$, is rejected and the alternate hypothesis is accepted. However, if the experimental $t$-value, $t_{exp}$, had been less than the theoretical $t$-value, $t_{th}$, the null hypothesis would have been retained. The higher the confidence level, the more certain one can be that there really is a difference in the two groups being tested. For example, 95% confidence means that there is only a 5% chance that such a difference in scores could have been found purely through the effects of sampling.

Instead of comparing the $t_{exp}$ and $t_{th}$ to determine significant difference, one may also compare the alpha level and $p$-values. An alpha level, $\alpha$, is the probability that the null hypothesis will be rejected in error when it is true (a decision known as a Type I error, or "false positive"). It is the number of times out of 100 someone will be incorrect if the null hypothesis is being rejected. If someone chooses an alpha level of 0.05, 5 times out of 100 he/she will be incorrect if the null hypothesis is rejected. 95 times out of 100, he/she will be correct because it is more likely that the means come from two different populations (Case II). A $p$-value, $p$, is the probability of observing the given result by chance given that the null hypothesis is true. Small values of $p$ cast doubt on the validity of the null hypothesis. If the p-value is less than the alpha level, the alternate hypothesis is accepted. However, if the $p$-value was greater than the alpha level, $p>\alpha$, the null hypothesis would be retained.

In Matlab [MathWorks, 2007] the following function performs a $t$-test to determine whether two samples from a normal distribution could have the same mean.

$$[H, P, CI, STATS] = TTEST2(X, Y, ALPHA, TAIL) \quad \textbf{(4.7)}$$

The null hypothesis for the above function is $H_0$="means are equal". For TAIL = 0 in equation (4.7) the alternative hypothesis, $H_A$, is: "means are not equal". For *TAIL = 1*, $H_A$ = "mean of *X* is greater than mean of *Y*". For *TAIL = -1*, $H_A$ = "mean of *X* is less than mean of *Y*". The default value of *TAIL* in (4.7) is zero. *ALPHA* is desired significance level (*ALPHA = 0.05* by default). *P* is the *p*-value. *CI* is a confidence interval for the true difference in means. *STATS* is a structure with two elements named '*tstat*' (the value of the *t* statistic) and '*df*' (its degrees of freedom). If equation (4.7) returns a zero value for *H* then it indicates that the null hypothesis cannot be rejected at significance level of alpha. If the value of *H* is returned as 1 then it means that the null hypothesis is rejected at significance level of alpha.

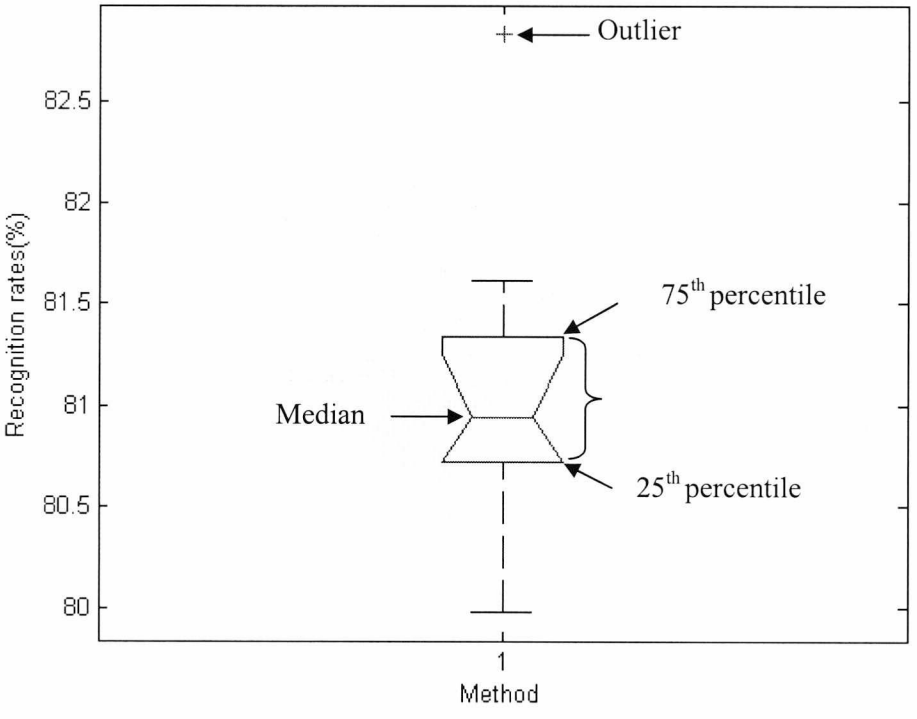



**Figure 4.5 Box plot**

## 4.5 Box Plot

A boxplot, or box and whisker diagram, [Chambers *et al.*, 1983] is a very useful tool for graphically portraying the empirical distribution of data. It gives a quick insight into the empirical distribution of data and its statistics. [Tukey, 1977] invented box plots as a powerful way of summarizing distributions of data. This graphical technique has been applied with success elsewhere [Yusta *et al.*, 1998; Bounessah and Atkin, 1994; O'Connor and Reimann, 1993; Kürzl, 1988]. Boxplots are especially useful when comparing two or more sets of data.

Figure 4.5 has several graphic elements: The lower and upper lines of the "box" are the 25th and 75th percentiles of the sample. The distance between the top and bottom of the box is the interquartile range. The 25th percentile is where, at most, 25% of the data fall below it. The 75th percentile is where, at most, 25% of the data is above it. The line in the middle of the box is the sample median. The median is the point where 50% of the data is above it and 50% below it. If the median is not cantered in the box, then it shows an indication of skewness. The "whiskers" are lines extending above and below the box. They show the extent of the rest of the sample (unless there are outliers). Assuming no outliers, the maximum of the sample is the top of the upper whisker. The minimum of the sample is the bottom of the lower whisker.

By default, an outlier is a value that is more than 1.5 times the interquartile range away from the top or bottom of the box. The plus sign at the top of the plot is an indication of an outlier in the data. This point may be the result of a data entry error, a poor measurement or a change in the system that generated the data. The point of the notch in Figure 4.5 falls at the mean, and the height of the notch corresponds to the 95% confidence interval [Streiner, 1997], which is defined as:

$$95\%CI = \bar{x} \pm 1.96 \times \frac{SD}{\sqrt{n}} \qquad \textbf{(4.8)}$$

Where $\bar{x}$ is the mean, *SD* the standard deviation, and $n$ is the sample size. Matlab's [MathWorks, 2007] statistical toolbox has a function to produce notched box and whisker plots for distributions of data. In Matlab notches represent a robust estimate of the uncertainty about the medians for box-to-box comparison. Boxes whose notches do not overlap indicate that the medians of the two groups differ at the 5% significance level.

## 4.6 Software used

Software was written for various optimisation algorithms and the n-tuple network. C++ language was used to code algorithms. The results were obtained with the code running under Windows XP on a 2 GHz Pentium 4 machine with 512 MB main memory. Matlab's statistic toolbox was used to perform Student's t-tests and produce box plots. Results were also plotted with Microsoft Excel's chart wizard.

## 4.7 Summary

This chapter explained the basic experimental procedure for experiments. Basic procedure involves in use of train-data to optimise the input connection of the n-tuple classifier and then to use the optimised classifier to recognize the test-data. Selection of NIST's Special database 3 and 7 as train and test data respectively has been argued. The term 'critical class' has been defined and use of 'class-specific' tuples to describe critical classes has been explained. Class-specific tuples are sought in the experiments. An optimum set of class-specific tuples eventually improves the recognition rates of the classifier. It has been mentioned that the experiments are repeated several times to confirm the consistency in results. Testing the statistical significance of the results are also important and it has been explained how student's t-test and box plot can help on this regard.

# Chapter 5

# Reward and Punishment Based Method

## 5.1 Introduction

Realizing the fact that the classification performance of the n-tuple classifier is highly dependent on the actual subset of the input bits probed [Bishop *et al.*, 1990; Jørgensen *et al.*, 1995], a novel approach was introduced based on a **R**eward a**n**d **P**unishment (**RnP**) scheme to select input mappings of the classifier. Classes with high error rates were termed as critical classes and different groups of tuples were formed for different classes. The strategy was to employ more number of tuples to a critical class-group than an easily distinguishable class. In order to illustrate the capabilities of the RnP based measure the task of recognizing hand-written digits from NIST [Wilkinson *et al.*, 1992] database was chosen. Next section will explain the importance of class-specific tuples. Section 5.3 will describe the tuple search algorithm. Section 5.4 will explain the objective function and its formulation. Experimental outcomes will be presented in Section 5.5.

## 5.2 Class-Specific Tuples

This research was aimed to find an optimal set of input connections for the n-tuple classifier to achieve higher recognition rates. The performance of each connectivity

pattern during search will be evaluated by a reward and punishment technique that will be explained in Section 5.4. Finding an input map that gives high scores for all classes will be extremely rare because there will be similar classes in the system and the overlapping region between similar classes will cause the discrimination task much harder. So there will be classes in the dataset which will be difficult to recognise and give low recognition rates. These classes can be termed as "critical" classes. One strategy to improve the discriminating power for a critical class would be to optimise or tune a sufficient number of tuples only for that class such that each tuple in that group can give high score for that class. Eventually all of these tuples when work together will try to improve the recognition rate for that critical class. This group of tuples tuned to a specific class is known as class-specific tuples.

## 5.3 Tuple Search Algorithm

A stochastic search algorithm was developed to find an optimum set of input connections to the n-tuple network. The unique strategy in the search algorithm was to reserve more tuples to a more critical class group (Section 5.2). The classes with high error rates were termed as critical classes. By using more class-specific tuples for a critical class the search algorithm would allow more time to be given to find features for a critical class. Before the search starts the total available tuples ($R$) was distributed among classes proportionately to the error rates. To calculate the number of class-specific tuples for a class at first the error rate of that class was divided by the total error rate and then the result of the division was multiplied with the total available tuples ($R$). The result of the multiplication was rounded to the nearest integer. No normalisation was used in the calculation of class-specific tuples. Providing more tuples to a class with a high error rate ensures that the extra care has been taken for a critical class group. Tuples engaged to a specific class best learn the features of that class and also learn some features for other classes to an extent. Thus

a class-specific tuple can give scores for both its specific class and other classes when the performance of the tuple is evaluated by an objective function (Section 5.4). But it should recognise its specific class better than other classes. The pseudo code of the search algorithm is presented in Figure 5.1.

```
LET  j = 0;//class index
     Tf = 0;//number of successful tuples in any
     iteration
     p'j = number of tuples responsible to best
     describe class Cj;
REPEAT
    GENERATE ([Q-Tf] set of tuples randomly);
    FIND SCORE (Q set of tuples based on class Cj);
    RANK (Q set of tuples based on their scores);
    CARRY (Tf set of successful tuples to next
          iteration);
    IF (Tf = p'j)
         THEN {
            SAVE(p'j set of tuples as mature tuples);
            SET (j = j + 1, Tf = 0);}
UNTIL (j < number of classes);
```

**Figure 5.1 Pseudo-code of RnP based search**

To understand the search algorithm consider a class index $j$ to identify a class '$C_j$'. Let us consider an objective function (Section 5.4) that gives scores for class-specific tuples. Let's say $Q$ is the total number of tuples in any iteration. The target is to find $R$ number of class-specific tuples in total. The distribution of $R$ tuples among the classes is proportionate to the error rates. The more the class is critical, the greater number of tuples it gets.

Let us assume $p'j$ is the number of tuples that will be matured for class $C_j$. Thus the summation of all $p'j$ ($\Sigma p'j$) will be equal to $R$. Every iteration scores for $Q$ sets of tuples are evaluated according to the objective function based on $C_j$. So the RnP based objective function will give the scores only for the class $C_j$. Then $Q$ sets of tuples are ranked based on their scores for the class $C_j$. The number of tuples, say $Tf$, whose scores are higher than a predefined threshold are treated as the successful tuples for a certain iteration and they are being carried to the next iteration by virtue of their good scores. So in the next iteration only $Q$-$Tf$ tuples will be created randomly. Before moving to the next iteration a check has to be made if the number of successful tuples ($Tf$) have met the number of class-specific tuples ($p'j$) in the class $C_j$. If $Tf = p'j$, then the mappings for these successful tuples will be saved and these tuples will be treated as the matured tuples those are specific for the class $C_j$. The whole process repeats until all matured class-specific tuples ($\Sigma p'j = R$) for all classes have been sought and later these matured tuples will be used for the final recognition task.

### 5.3.1 Flow Chart of Tuple Search Algorithm

Figure 5.2 and Figure 5.3 show the tuple search algorithm in a flow chart. The algorithm started by creating $Q$ number of tuples randomly. The randomness in selection ensures that the tuples are mapped all over the input matrix. Now performance of all tuples will be evaluated against an objective function. Based on

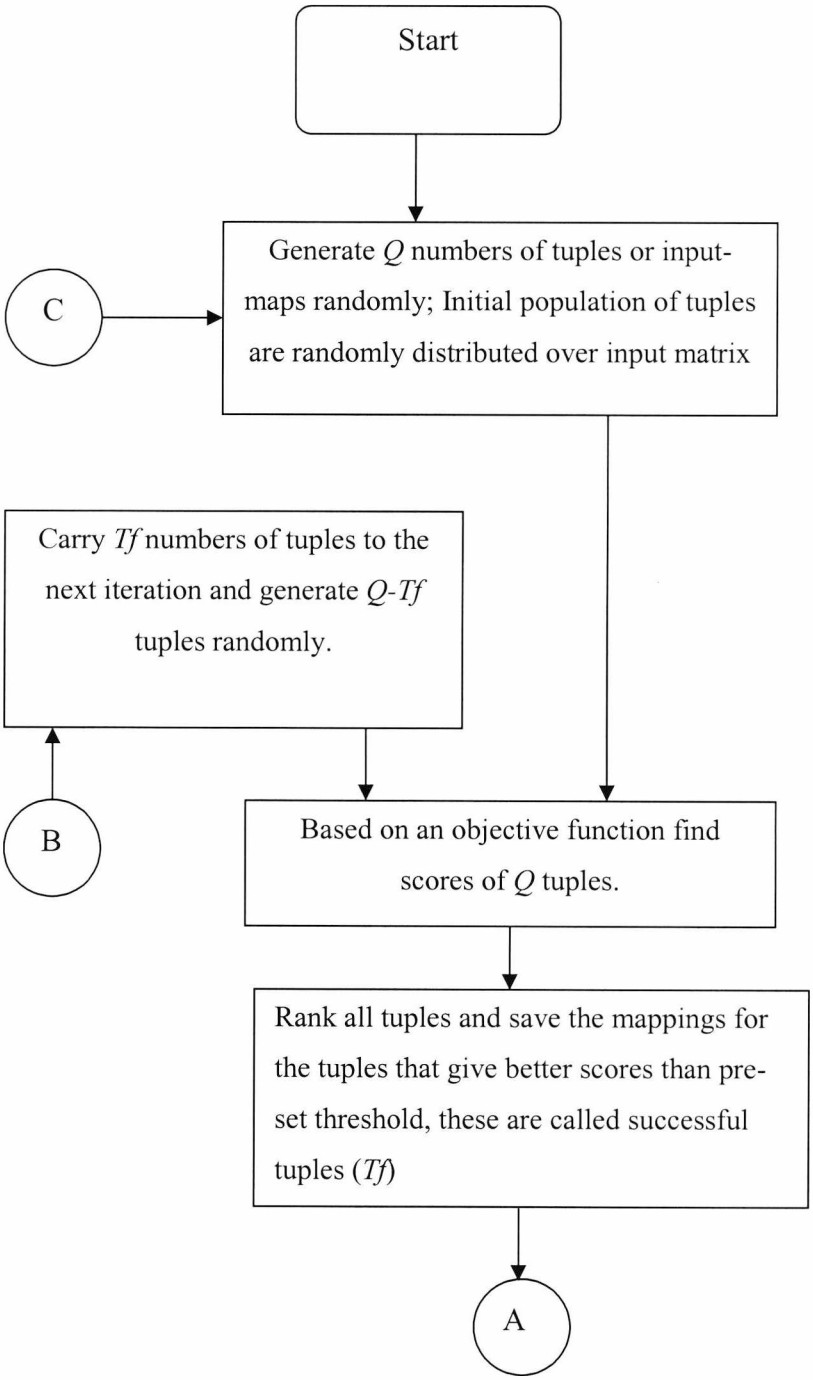



**Figure 5.2 Flow chart (Part1) of RnP based Tuple search algorithm**

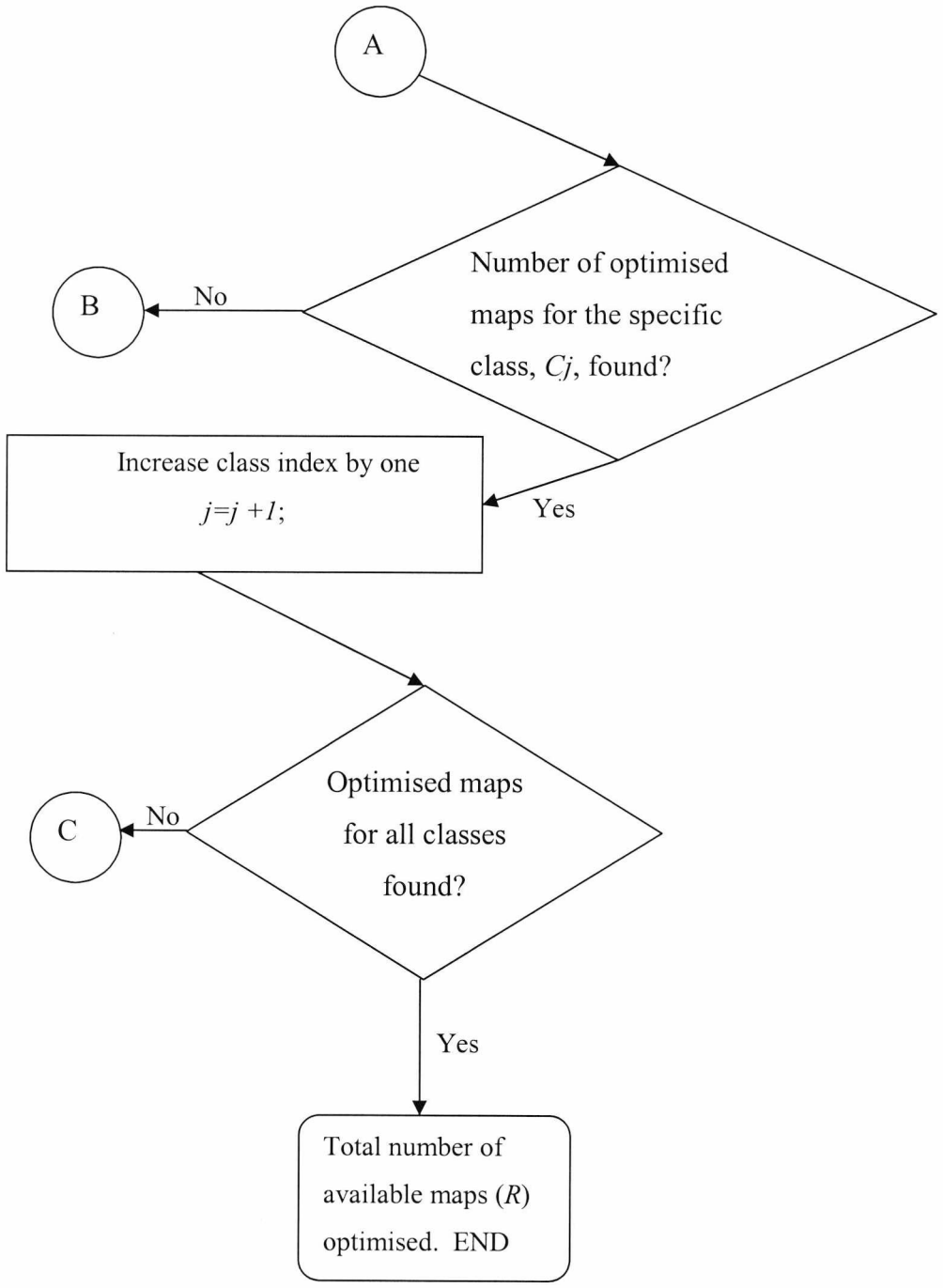



**Figure 5.3 Flow chart (Part2) of RnP based Tuple search algorithm**

the score given by each tuple, all the tuples will be ordered from high to low and then compared against a predefined threshold function. Tuples whose score are higher than the threshold will be successful (*Tf*) to go to the next iteration. At the end of each iteration the successful tuples will be counted and if the number is less than the required then the tuples those were not successful (score less than the threshold) will be remapped randomly to be evaluated in the next iteration. Performance of all tuples will be again measured against the objective function and all the steps will be repeated until required numbers of tuples are sought for a specific class. The steps will be repeated for other classes too until class specific tuples for all classes are found.

## 5.4 Reward and Punishment Based Performance Measure

A trained classifier can either recognize or misclassify or reject a test pattern. In the reward and punishment (RnP) scheme [Simões, 2000] a reward is associated with the correct recognition of the pattern and the penalties for misclassification and rejection. It takes account misclassification and rejection while measuring performance, where as standard error rates based cost function [e.g. CV error rate in Jorgensen *et al.*, 1995] ignores the difference between misclassification and rejection. Rejection will be less damaging than misclassification [Maltoni *et al.*, 2003]. The reward and punishment concept has been used before in both weighted and weightless neural networks. In reinforcement learning [Sutton and Barto, 1998] actions are associated with rewards and punishment for 'good' and 'bad' behaviours [Ackley and Littman, 1991]. [Aleksander, 1989] used a reward and punishment algorithm in an extended n-tuple model called 'probabilistic logic node' (PLN) [Penny and Stonham, 1990]. In addition to storing 0 or 1, a PLN memory location could be in a 'u' state, in which it was equally likely to output 0 or 1 when addressed. All locations are initially set to 'u' (in PLN 1 means 'Yes' pattern is for that class, 0 means 'No' pattern is not for

that class as that pattern is counter example of that class, 'u' means 'don't know'). After one of the training patterns is presented to the net if the output matches the desired output all the addressed locations are made to assume their current output (0/1) (Reward). In case the output the PLN network mismatches the desired output all the addressed locations are made to assume the "u" value (Penalty). This way the training continues until all training patterns are presented at the input. RnP algorithm described in this thesis doesn't alter the memory values; rather it measures the performance of a trained tuple on the validation dataset. For this the whole pattern data are divided into three parts: training set, evaluation set and test set. Let us consider $S_l$ is the total number of samples for training the classifier, $S_e$ is the number of samples available for evaluation purpose and $S_t$ denotes the number of samples in the test data set. If $S$ is the total number of available samples then $S = S_l + S_e + S_t$. Now for optimisation purpose the network is trained with $S_l$ and evaluated with $S_e$ dataset. For the final recognition task both the $S_l$ and $S_e$ are used for training the network and $S_t$ is used for testing. Finally, the dataset $S_e$ can be considered to have three parts: $S_{Cj}$, $S_{rj}$ and $S_{mj}$. $S_{Cj}$ is the number of $C_j$ samples correctly recognized, $S_{rj}$ is the number of $C_j$ samples rejected and $S_{mj}$ denotes the number of $C_j$ samples misclassified. Considering all these definitions the objective function for class $C_j$, denoted as $O_{ji}$, will be evaluated by the following equation:

$$O_{ji} = S_{cj} \times P_c + S_{rj} \times P_r + S_{mj} \times P_m \qquad \textbf{(5.1)}$$

Where $P_c$= Positive points associated with reward for recognition; $P_m$= Negative points for misclassification; $P_r$= Negative points for rejection; $i$= population index which varies from *1* to $Q$; $j$= class index.

The point scheme for the objective function will be explained in the next section. As explained in the previous section, in every iteration the search algorithm selects the number of tuples as the successful tuples according to a threshold.

Consequently a threshold function was developed for the system, which is shown in (5.2):

$$Threshold = \max_i(O_{ji}(t)) \times (1 - k(t)) \qquad \textbf{(5.2)}$$

$$k(t) = exp(-\tau / t) \qquad \textbf{(5.3)}$$

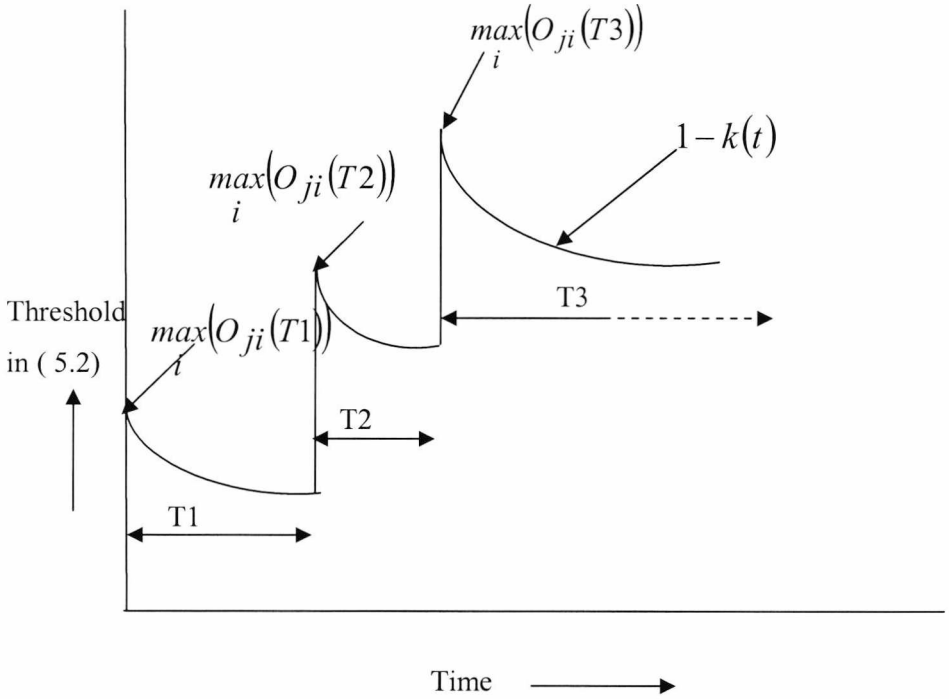



**Figure 5.4 Exponential decay of threshold function**

In ( 5.2) $\max_i(O_{ji}(t))$ is the score of the best-performed tuple among all the tuples in the current iteration. The value of $k$ sets a percentage of $\max_i(O_{ji}(t))$ which is the minimum score a tuple has to have to become successful in the current iteration. To accelerate searching, $k$ can be varied over time according to (5.3). Both $\max_i(O_{ji}(t))$ and $k$ are time dependent. The same value of $\max_i(O_{ji}(t))$ may sustain for several iterations but the $k$ will change in every iteration. Figure 5.4 illustrates a

fictitious search scenario where there are three time intervals: T1, T2 and T3. In the example $\max_i(O_{ji}(t))$ has different values increased in three stages. In every stage "Threshold" exponentially decreases according to the equation ( 5.2) and ( 5.3) where $\tau$ should be chosen suitably to set the exponential decay of the threshold over iterations. An equation similar to (5.2) was used in the simulated annealing algorithm [Bishop *et al.*, 1990], where initially large random jumps across the search were allowed and then the size of the possible jumps was reduced exponentially. Thus in SA the use of decay equation (5.2) allows the whole search space to be probed at the very initial stage, so as to find the general area of the optimum position. Like any other stochastic search, the RnP based search will produce better results if more time is spent on searching. So the value of $\tau$ should be carefully chosen and varied throughout the search as a trade-off between the performance and the speed.

### 5.4.1 Point Scheme for RnP

The point scheme determines what values should be set for $P_c$, $P_m$ and $P_r$ in ( 5.1). In general a rejection is thought to be more favourable than a misclassification. It is equivalent to the system getting confused rather than making the wrong decision. To find out what should be the point to set for the reward consider a point for misclassification as $P_m = -1$. If $J$ is the criterion deciding minimum number of samples of the class $C_j$ that must be recognized by one individual tuple to maintain a predefined minimum score, say $O_{min}$, then the value of $P_c$ can be found by the following formulae:

$$J \times Se_j \times P_c + P_m \times Se_j(1 - J) = O_{min} \qquad \textbf{(5.4)}$$

In the above formulae $Se_j$ is the number of samples available to evaluate a specific class $j$. $Se_j$ is the part of the evaluation data set $Se$, where Se = $\sum_j Se_j$ . $J$ in

(5.4) is a value in percentage and $O_{min}$ is a pre-set minimum score. If $O_{min}$ is set to 500, $J$=5%, $Se_j$ =500 and $P_m$=-1; then $P_c$ comes out as 39. So rewarding points should be 39 for a recognition of a pattern when $P_m$=-1. As rejection is more favourable than misclassification $P_r$ can be set empirically as –0.5. $J$=5% indicates that the pattern is so complex that even when the least 5% of the samples of the evaluation data set is recognized by a single tuple, the evaluation function will give a minimum score of 500. A tuple with larger tuple-size has the more ability [Section 3.3.1] to recognize patterns. So for a larger tuple-size higher percentage of $J$ can be chosen. It is more convenient to choose a lower value in $J$, as it works for both larger and smaller tuple-sizes.

## 5.5 Evaluation of RnP Optimisation

The proposed stochastic search method was applied to recognize handwritten characters from the NIST database. The partition *hsf_{0}* (Section 4.2.1) of the Special Database 3 was used for training and the partition *hsf_{4}* (Section 4.2.1) from the Special Database 7 [Wilkinson, 1992] was used for testing. NIST recommends using images of *hsf_{4}* for testing as they are more difficult from other partitions and this ensures the heterogeneity between the training and testing set (see analysis in [Wilkinson *et al.*, 1992]), a fact which is reflected in the results. The numeric data set consisting only the digits (0,1...9) was used for the experiments. Each character is a binary image with the dimension 32 by 32. All digits are scaled into same dimension and centred. In the partition *hsf_{0}* there are 1000 images per class for training and in the partition *hsf_{4}* there are 1000 images per class for testing. For the experiments the total train samples were again divided into two halves by the holdout method [Hand, 1986] and one part ($S_l$) was used to train the network in the evaluation phase and other part ($S_e$) to evaluate the RnP based objective function.

After finding the mature tuples, all training images ($S_l + S_e$) of *hsf_{0}* were used to train the classifier and the images from *hsf_{4}* were used to test.

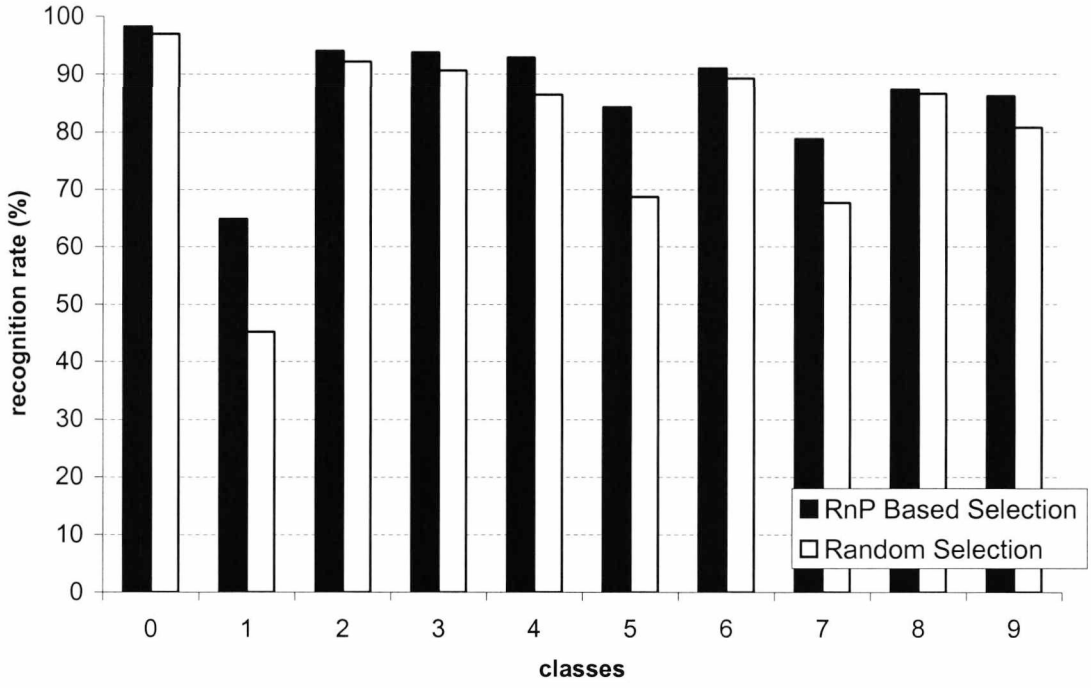



**Figure 5.5 Class-wise comparison of recognition rates**

**Table 5.1 Distribution of Class-specific tuples among various classes**

| **Class** | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 |
|---|---|---|---|---|---|---|---|---|---|---|
| **Error rates** | 2.9 | 54.8 | 7.6 | 9 | 13.25 | 31.5 | 10.1 | 31.9 | 13.7 | 19.25 |
| **Class-specific tuples** | *p`0=* 2 | *p`1=* 42 | *p`2=* 6 | *p`3=* 7 | *p`4=* 10 | *p`5=* 24 | *P`6=* 8 | *p`7=* 25 | *p`8=* 11 | *p`9=* 15 |

**Table 5.2. Experimental Settings for RnP Optimisation**

| Parameters | Values |
|---|---|
| Time constant (5.3), $\tau$ | 100 |
| Constant (5.4), $J$ | 5% |
| Point for misclassification, $P_m$ | -0.5 |
| Point for rejection, $P_r$ | -1 |
| Point for reward, $P_c$ | 39 |
| Population size, $Q$ | 200 |
| Total train images $(S_l + S_e)$ | 10,000 |
| Total test image , $S_t$ | 10,000 |
| Images per class to evaluate, $S_{ej}$ | 500 |

**Table 5.3 Improved overall recognition rate by RnP based optimisation**

| Methods | Average Recognition Rate (%) | Best Recognition Rate (%) |
|---|---|---|
| Conventional randomly selected n-tuple [Bledsoe and Browning, 1959] | 80.93 | 82.83 (in 2000 runs) |
| RnP Based Stochastic approach | 83.67 | 84.5 (in 10 runs) |

Two experiments were performed. The first one was to demonstrate how the recognition could be improved for a class when all the tuples in the network are tuned only for that particular class. In total 140 tuples were used for the network with the tuple-size of 8 bits. The number 140 was chosen empirically but it satisfies the relation $R \geq L/n$ (in Section 3.3) for a 32 by 32 binary image. The results are shown

in the Figure 5.5. It can be seen that character 1 is the most critical class to be recognized in the NIST database. For the random case the recognition of class 1 was 45.26, which was the mean of 2000 runs. The RnP based optimisation improves the recognition of class 1 by 19.66% (Figure 5.5). It improves the recognition rate for class 5 by 15.56%, class 7 by 11.02% and all other classes by several percentages. The recognition rates found by RnP based search correspond to the mean of ten runs.

The second experiment was required to demonstrate the improvement of the overall recognition rate by the stochastic search method. The overall rate was the average of all recognition rates of all classes. The total number of tuples for the experiment was 150 with the tuple-size 8 bits. Tuples were distributed among classes proportionately to the error rates of the classes. Table 5.1 reports the error rates of ten classes from character 0 to 9. It also presents the number of class specific tuples for each class. The method of calculating class-specific tuples was described in Section 5.3. Being the most critical class, character 1 gets 42 tuples out of 150. The randomly selected network was run for long enough (2000 iterations) to give it a chance to find better input maps that could be comparable with the maps found by the stochastic search. The best overall recognition rate by the random network was found to be 82.83% and the average rate was 80.93%. In case of stochastic optimisation (Table 5.2), the average overall recognition was 83.67%, which was 2.74% superior to the random case proposed by [Bledsoe and Browning, 1959]. The results were obtained with the code running under Windows XP on a 2 GHz Pentium 4 machine.

### 5.5.1 Students t-test results between RnP and basic n-tuple

The Student's t-test (Section 4.4.1) assesses whether the average recognition rates by any two algorithms, say $X$ and $Y$, are statistically different from each other. The null hypothesis for the test was ""average recognition rate by the RnP ($X$) is higher than conventional random selection ($Y$)". The degree of freedom (Section 4.4) for the test

was 18 as each algorithm was run for 10 trials. In the t-test, a t-value was calculated against the null hypothesis and compared with the tabulated values at different confidence levels with 18 degrees of freedom. By convention [Deacon, 2006], we say that a difference between means at the 95% confidence level is "significant", a difference at 99% level is "highly significant" and a difference at 99.9% level is "very highly significant". Tabulated t-values for 95%, 99% and 99.9% at 18 degrees of freedom were found to be respectively 2.10, 2.88, and 3.92. So if the calculated t-value exceeds the tabulated t-value of 3.92 it can be claimed that the difference in recognition rates between the RnP and random selection approaches was statistically "very highly significant". The calculated p-value indicates the probability of observing the result by chance, given that the null hypothesis is true. Small values of probabilities cast doubt on the validity of the null hypothesis. The p-value between the RnP and random selection approach was 1, which indicates that the null hypothesis "average recognition rate by the RnP ($X$) is higher than random selection approach ($Y$)" is extremely valid. The calculated t-value ($t_{exp}$ in Section 4.4.1) for the null hypothesis was 7.98 and it exceeded the tabulated value ($t_{th}$ in Section 4.4.1) of 3.92 at the 99.9% confidence. The result indicates that the increase in recognition rate by RnP based approach over traditional n-tuple method is statistically "very highly significant".

Figure 5.6 displays the side-by-side box plots of RnP and random selection process. Box plots (Section 4.5) are an excellent tool for conveying location and variation information in data sets, particularly for detecting and illustrating location and variation changes between different groups of data. In the experiment data samples of a particular algorithm were recognition rates observed from the ten trials of that algorithm. Thus each box in Figure 5.6 was constructed with the recognition rates of ten trials. The notches in the figure are drawn about the median so that, notches which don't overlap represent significant differences between medians (with

95% confidence). In Figure 5.6 the RnP clearly exhibits a significantly higher median than the conventional random selection. Box plots also show if there are unusual observations (outliers) in the dataset. One unusual observation was plotted for the random selection.

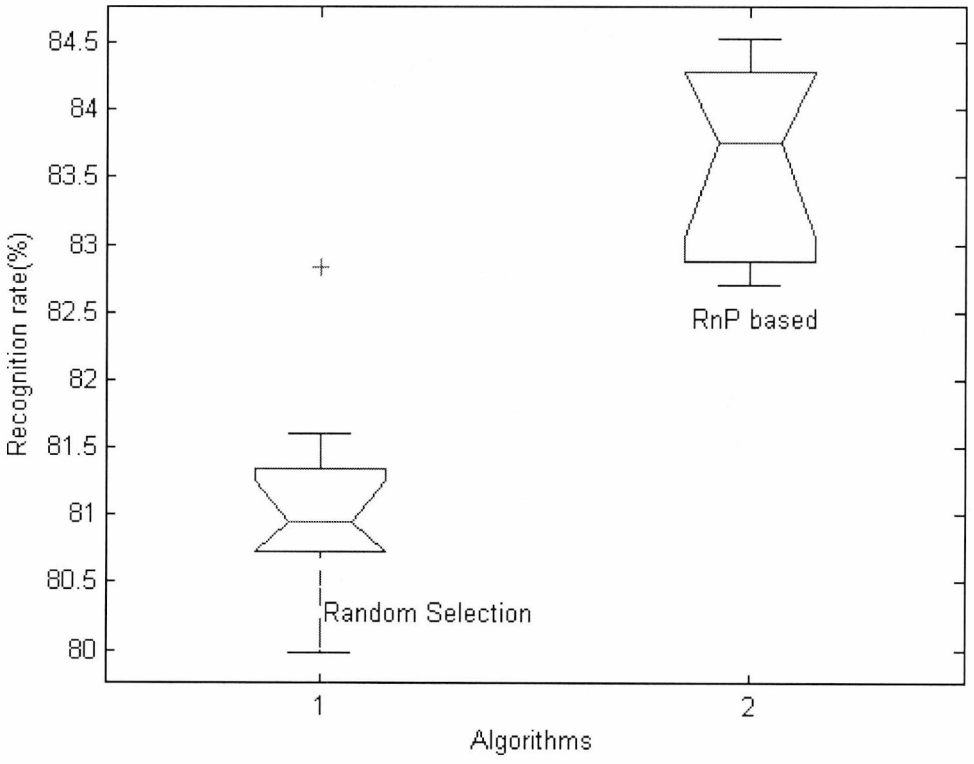



**Figure 5.6 Box Plot of RnP and Random selection**

## 5.6 Summary

This chapter presented the implementation of a new stochastic search strategy in finding an optimal set of n-tuples. The uniqueness in the algorithm was to distribute the total available tuples among the classes according to the error rates. Thus a

difficult class gets more attention by the algorithm. The RnP method spent more time in finding features of a difficult class than an easily recognisable class. The optimised network was tested on a handprinted database. Results showed an improvement of 2.74% in recognition rate by the RnP based approach over the conventional randomly selected method. The improvement was statistically very highly significant. This chapter is a good reference to realize the underlying methodology of the RnP based stochastic process for the n-tuple classifier. The proposed RnP method improved the recognition success at some small cost to the training speed. The high speed of the basic n-tuple network makes it entirely practical to carry out the pre-processing task of selecting input maps to find the suitable ones.

# Chapter 6

# Particle Swarm to Optimise n-tuples

## 6.1 Introduction

This chapter will investigate the implementation of the PSO on the n-tuple network to optimise the input connection mappings. Among different optimisation techniques the Particle Swarm Optimisation [Kennedy and Eberhart, 1995] exhibits good performance in finding solutions to static optimisation problems [Parsopoulos *et al.*, 2001a; Parsopoulos and Vrahatis, 2001b]. Being successfully applied in many areas like function optimisation, artificial neural network training [Parsopoulos and Vrahatis, 2001b; Settles and Rylander, 2002] or fuzzy system control [Esmin *et al.*, 2002], the PSO seems to be a good candidate to find optimal set of input maps for the n-tuple network. This chapter will describe how the particle swarm can be applied on the n-tuple network. Learning strategy of the n-tuple network by PSO will be explained. Different parameter settings of the PSO will be reported in Section 6.6. Section 6.7 will give the results of the experiments by the particle swarm optimised network. Genetic Algorithm will be applied on n-tuples to compare against the PSO. Performances, similarities and differences between the PSO and GA will be discussed in Section 6.8.

## 6.2 Particle Swarm on n-tuples

When the particle swarm is applied on the n-tuple, the "tuples" of the n-tuple can be termed as "particles". Thus each particle corresponds to an input connection map of the n-tuple network. The size of an n-tuple network is defined by the total number of tuples it is built with. Total number of tuples, denoted by $R$, is the number of tuples available to be optimised by particle swarm. $R$ depends on the network's structure. The particle swarm technique makes use of a population of particles or input-maps (for n-tuples), where each particle has a position, a velocity. The PSO formulae, as shown in equation (2.1) and (2.2) define each particle as a potential solution in a multi-dimensional search space.

The dimension of the PSO corresponds to the bits or the tuple-size (Section 3.3) of each tuple. As the tuples are "$n$" bits, so the PSO will be $n$ dimensional with the $i$-th particle represented as $X_i=(X_{i1},X_{i2},..X_{in})$. The PSO remembers the best position found by any particle which is known as global best, denoted by $P_g$. Additionally each particle remembers its own previously best found position designated as $P_i=(P_{i1},P_{i2},...P_{in})$ and its velocity $V_i=(V_{i1},V_{i2},...V_{in})$. Equation (2.1) and (2.2) will define the velocity and position of the $i$-th particle with $d$-th dimension. For example for the particle with index 1 and dimension 3 the equations will be:

$$V_{1,3}(t+1) = \omega \times V_{1,3}(t) + \psi 1 \times ran1 \times (P_{1,3} - X_{1,3}(t)) + \psi 2 \times ran2 \times (P_{g3} - X_{1,3}(t)) \quad \textbf{(6.1)}$$

$$X_{1,3}(t+1) = X_{1,3}(t) + V_{1,3}(t+1) \quad \textbf{(6.2)}$$

The PSO starts with a population of randomly generated particles (say $Q$) and detects the optimal solution through co-operation and competition among the individuals of the population. Every iteration a particle evaluates its position relative

to a goal or fitness. The velocity of each particle is updated by being pulled in the direction of its own previous best position and the best of all positions reached by all particles so far. Constants $\psi 1$ and $\psi 2$ in ( 6.1) determine the relative influence of the "individuality" and "sociality" traits of the particles and are usually both set the same to give each component equal weight as the individual and social learning rate. The constant $\omega$ is the inertia weight described by [Shi and Eberhart, 1998a]. The '*ran1*' and '*ran2*' are realizations of uniformly distributed random variables in {0, 1}.

## 6.3 Fitness Measure in PSO

The quality of particles is measured using a fitness function that reflects the optimality of a particular solution. The selection of fitness function depends on the problem types. For a classification problem, the rate of misclassified patterns can be viewed as the fitness value. The equation ( 5.1) described in the previous chapter was point-based function that incorporated a reward (positive points) for correct recognition of the pattern and the penalties (negative points) for misclassification and rejection. For consistency the fitness function is the same as for RnP optimisation described in Section 5.4. The position with the highest fitness value in the entire run is called the global best, $P_g$. Each particle also keeps track of its highest fitness value. The location of this value is called its personal best. If any fitness is greater than the global best, then the new position becomes $P_g$ and the velocities are accelerated toward that point. If a particle's fitness value is greater than its personal best, then 'personal best' is replaced by the current position and the particle is accelerated toward that position. Another point called the local best ($l_{best}$) is sometimes used. This is the position of the highest value from a small group of particles. The size of the group is usually about 15% of the population size. Particles are accelerated toward $l_{best}$ from their respective group. This technique, however, was not used in this research.

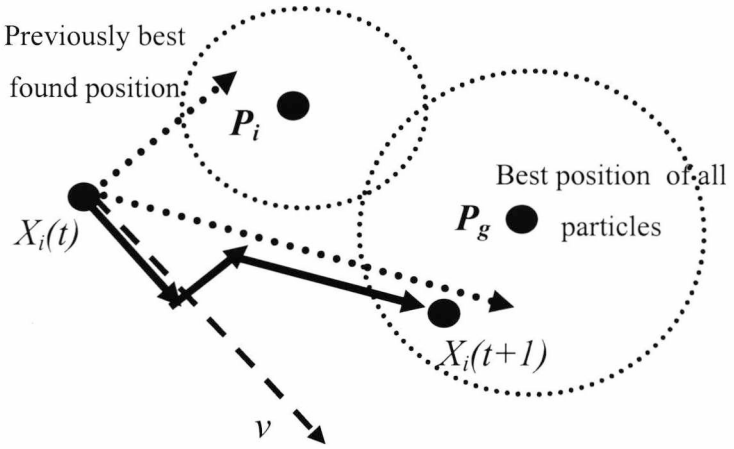



**Figure 6.1 Influence of global best and particle's own best**

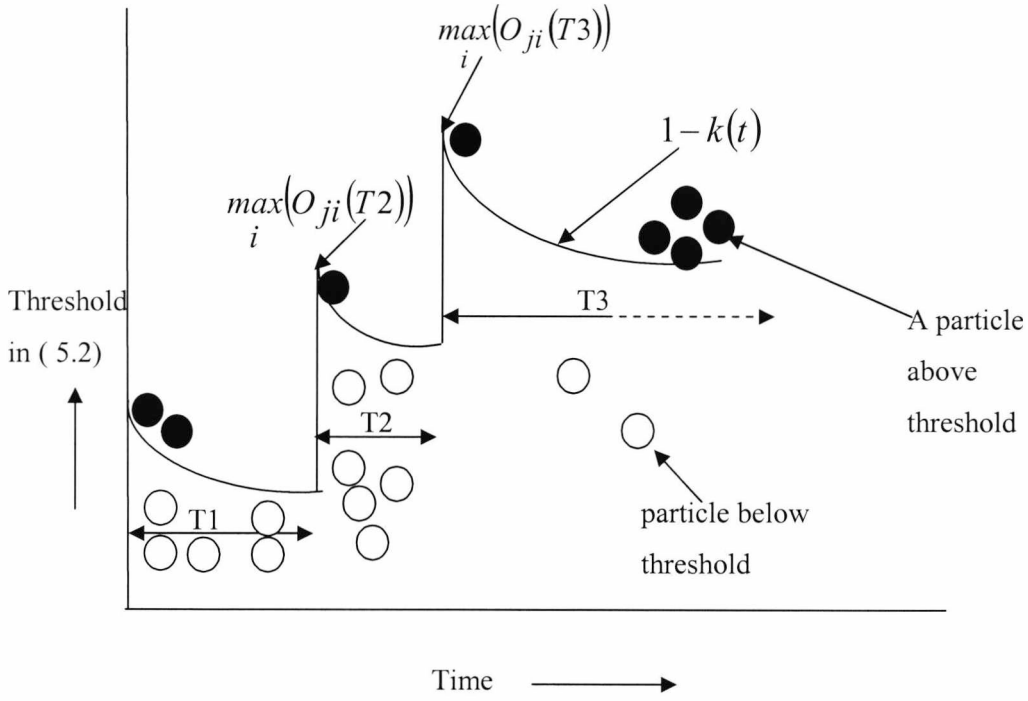



**Figure 6.2 More particles above threshold as the time goes**

Figure 6.1 shows how a particle's current position, $X_i(t)$, is influenced by that particle's previously found best position ,$P_i$, and best position of all particles ($P_g$). The dotted circles in the figure illustrate the influence of global best and particles own best. Particles current position is changed towards both $P_i$ and $P_g$ to account that influence. Fitness function that identifies $P_g$ and $P_i$ was described in Section 5.4. The same exponential threshold function ( 5.2) was used to set a decaying boundary in accepting solutions during search. A solution, which has a score above the threshold, will be considered as an acceptable solution.

Figure 6.2 shows fictitious case where white particles are the unsuccessful particles and the black ones are the particles whose fitness values are higher than the threshold value. In the figure there are three time intervals and it shows how fitness threshold exponentially decays in each time interval based on equation ( 5.2) and (5.3). As the time goes the threshold decays and this allow more black particles to come above threshold.

## 6.4 Learning Scheme by PSO

The learning scheme basically selects better n-tuples in an iterative manner. At each iteration particles with the fitness near to the maximum value, defined by the particle which is at the global best position, are kept. The definition of what is "near" is gradually tightened with iterations as the particles "flying" through the multi-dimensional search space. The particle swarm searches optima in the solution space and shrinks the search area step by step. It refines its search by attracting the particles to positions with good solutions. Like before (Section 5.3) the total available tuples ($R$) were distributed among classes proportionately to the error rates [Azhar and Dimond, 2004a]. Thus different classes had different number of class-specific particles. For this, each class has its own PSO to find the required number of class-specific particles. Pseudo code for the PSO based learning strategy is given below:

```
LET j= 0; //class index
    W= Width of a binary image;
    H=Height of a binary image;
    Q= Population size;
    i= particle index; // varies from 1 to Q
    R=Total number of available tuples to be optimised;
                        //depends on network  structure
    X_{i,d}=Particle's current position;
    P_{i,d}= Particle's previously best found position;
    P_{gd}= Global best; //best position visited so far by
                    any particle
    Pf = 0; //set of particles defined by particles
            //best positions so far and whose fitness
            //values are higher than a threshold
    p´j = number of tuples responsible to best describe
          class C_j;// Σp´j=R
REPEAT
    WHILE (Pf < p´j)
    {RUN PSO {
          For i=1 to the population size Q,
              For d=1 to the problem dimensionality n,
```

```
                Apply velocity update equation,(2.1)
                Limit magnitude of velocity (2.5)
                Update Position by applying equation (2.2)
                Limit position applying equation (2.6),
                        where Xmax= W×H and Xmin=1,
            End-for-d;
            Compute fitness of Xi,d(t+1) based on class Cj;
            If needed, update historical information
            regarding   Pi,d and Pgd;
        End-for-i;}
    RANK (All "Pi"s based on their fitness values);
    FIND (Pf);}
  SAVE (p´j set of particles as "mature" particles);
  SET (j = j + 1, Pf = 0);
UNTIL (j < number of classes);
```

To understand the learning scheme, consider a class index *j* to identify a class '$C_j$'. The fitness function is described by ( 5.1) which gives scores for class-specific particles. Let's say $Q$ is the total number of particles (population size) in any iteration. The target is to find $R$ number of class-specific tuples in total. The distribution of $R$ tuples among the classes is proportionate to the error rates. Let us assume $p'j$ is the number of tuples that will be matured or optimised only for the class

$C_j$. Thus the summation of all $p'j$ ($\Sigma p'j$) will be equal to $R$. $P_i$ is defined as the particle's previously best found position so far. At the end of each iteration, all particles defined by their best positions ($P_i$) are ranked based on their scores for the class $C_j$. Among these ranked particles $Pf$ numbers of particles have scores higher than a predefined threshold ( 5.2). Before moving to the next iteration a check has to be made if $Pf$ has met the number of class-specific tuples ($p'j$) in the class group $C_j$. If not, the PSO will be run to generate a new set of particles for the next iteration. If $Pf = p'j$, then the mappings for these successful particles will be saved and these particles will be treated as the matured tuples those are specific for the class $C_j$. The whole process repeats until all matured class-specific tuples ($\Sigma p'j = R$) for all classes have been sought and later these matured tuples will be used for the final recognition task.

### 6.4.1 Flow Chart of the PSO based Tuple Search

Tuple search algorithm that we explained in the previous section is being illustrated in the flow chart in Figure 6.3 and Figure 6.4. The algorithm starts with the $Q$ particles, initially randomly created. The $Q$ particles are distributed randomly over the whole pattern matrix. Next the fitness of each particle is measured according to the fitness equation ( 5.1).

Based on fitness results each particle's best positional values are updated. '$P_{i,d}$' defines the location along the dimension $d$ of the best positional value of each particle in the history. So '$P_i$'s represent best positions of all particles so far. Next in the flow chart fitness of all $P_i$ particles will be compared with a fitness threshold defined by equation (5.2).

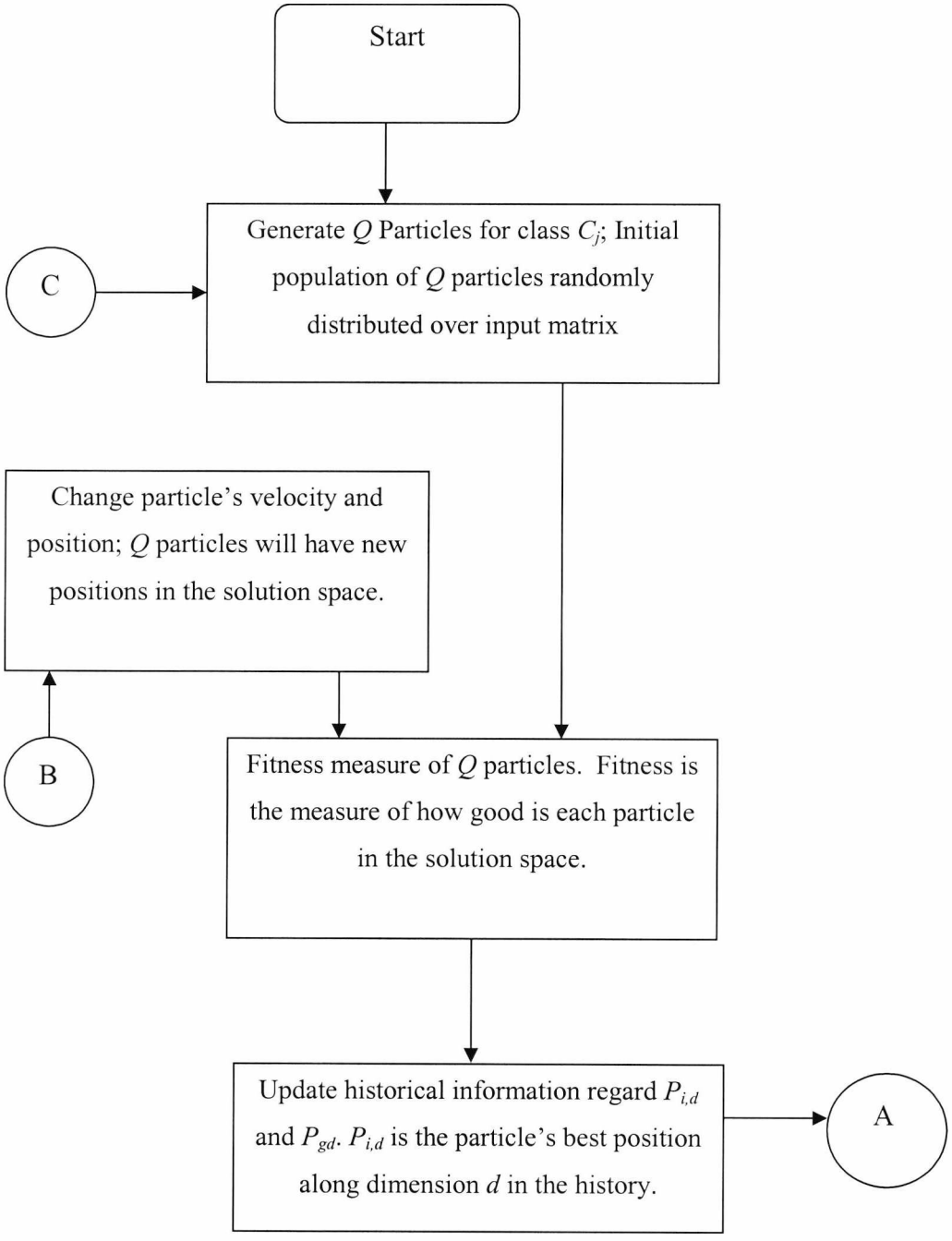



**Figure 6.3 Flow Chart (Part1) of the PSO based search algorithm**

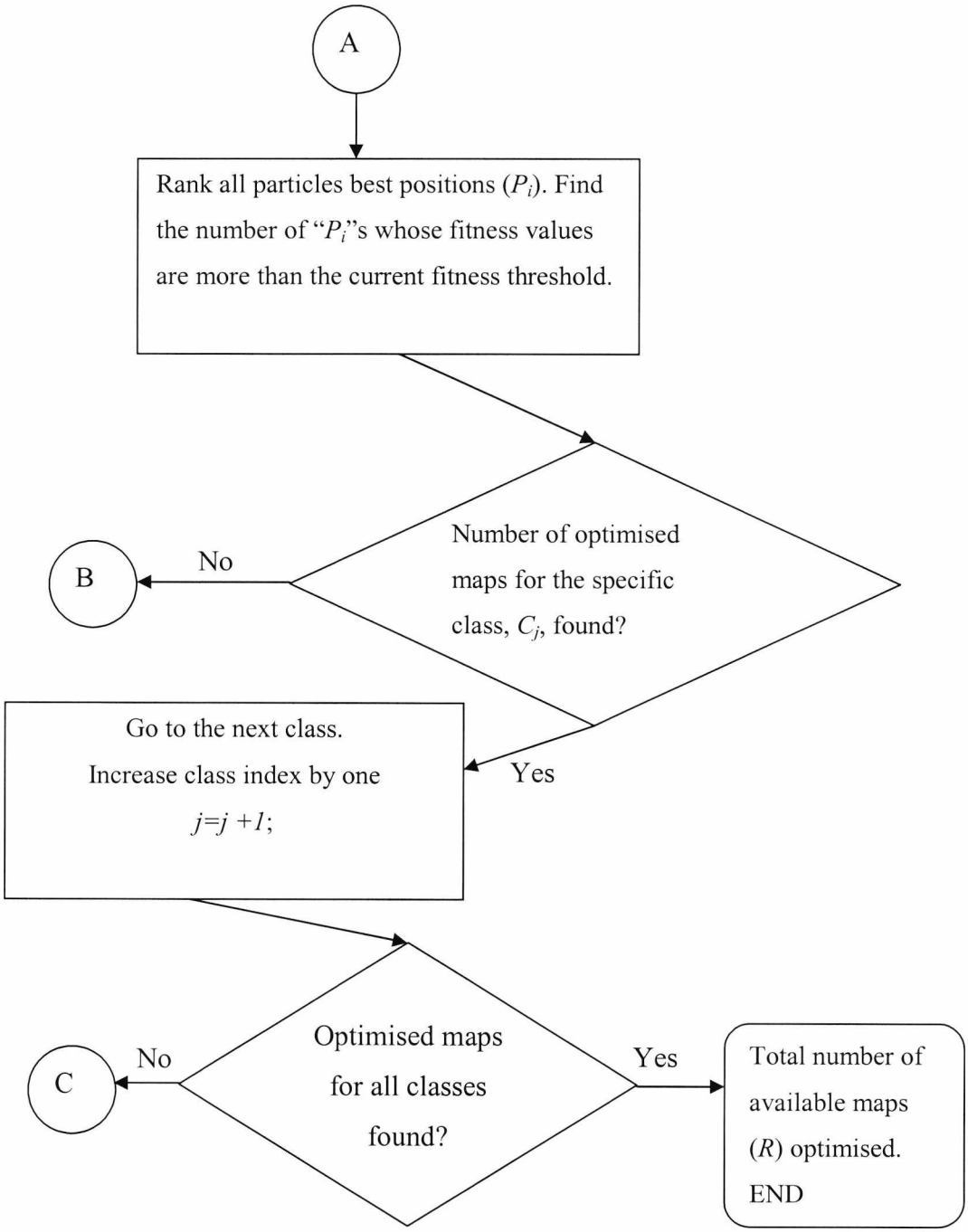



**Figure 6.4 Flow Chart (Part2) of PSO based search algorithm**

Particles whose fitness will be equal or higher than the threshold will be defined as the optimised particles for the current iteration. As the threshold varies from iteration to iteration, particles those are found as optimised in the current iteration may not be found as optimised in the next iteration. Next in the flow chart the algorithm checks if the total number of particles for a class has been found or not. If not found then the particles velocities and positions will be updated to explore new locations in the search space. After finding all optimised particles for a class the index of the class will be increased and tuples for the next class will be sought. At the beginning of searching for the next class a new population of $Q$ particles will be created randomly over the input pattern matrix. Once all the particles are sought for all classes the optimisation task will be completed and an optimal set of $R$ maps will be found. These $R$ maps will be used as input connection maps of the n-tuple network to recognize characters.

## 6.5 Parameter Settings

Particle Swarm Optimisation has certain parameters that require tuning to work well. This is also the case with other stochastic search algorithms e.g. for the tuning of GA mutation rates. No optimal parameter setting applies to all problems and tuning these parameter settings can result large performance variances. 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, $\omega$, 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 $\omega$ will decrease the speed of the particle resulting in more exploitation and less exploration. Thus finding the best value for $\omega$ is not an easy task and it may differ from one problem to another. Some parameters are crucial and some could be optional. 'MAXITERATIONS' is an optional parameter which sets a limit on the number of iterations to be executed before terminating a search. 'THRESHOLD' sets the

acceptable error level. A solution falling within THRESHOLD distance of a specified value would be considered an acceptable solution and the search would be terminated. The problem presented in this thesis was to optimise input connection maps of an n-tuple network by the PSO algorithm. The network was built out of 150 tuples with tuple-size 8. So the problem was 8 dimensional with $R$=150. For a 32 by 32 binary image (character) the image resolution was 1024 bits (3.1). When PSO was applied to search optimal set of connections for a specific class group, the values and choices of some PSO parameters were found to be very crucial. Following sections will discuss the chosen values of some of the important parameters.

### 6.5.1 Swarm size

Swarm size depends on the swarm termination criteria. If PSO stops after a fixed number of iterations, a choice has to be made, either choosing a larger swarm or having more iterations [Van den Bergh and Engelbrecht, 2001]. For the problem in this research the termination criteria was not based on iterations. The goal was to find a fixed number of particles whose fitness values were higher than a predefined threshold value. Thus the proposed PSO can stop at any iteration as soon as it finds the required number of matured class-specific particles ($p'j$ in Section 6.5). Imagine the case where there are only few class-specific tuples to be found. Because of fewer numbers of tuples the PSO might terminate in few iterations and due to this early termination there is a high possibility that the whole swarm might stuck in a local minimum. For a binary image of 32 by 32 dimension the size of the input vector, denoted by $L$, is 1024 (Section 3.3). Now for an 8 dimensional problem ($n$=8) to have more possibility that the swarm can pass over the entire input vector of 1024 bits even in a few iterations a suitable population size would be 200 ($Q$). In most of the experiments presented in this thesis the population size of 200 was used. A population size of 1000 was also tested but it showed a very slowly convergent system without any noticeable difference in performance.

### 6.5.2 Swarm Velocity

Due to the tendency for some particles to experience explosive growth in velocity, a $V_{\max}$ can be introduced which is an arbitrary cap placed on the magnitude of any particle's velocity. $V_{\max}$ is the step size of the swarm, the maximum distance a particle can travel in an iteration. Results in [Løvbjerg, 2002] show that in general performance improves as $V_{\max}$ shrinks. It was necessary to clamp the velocity of a particle to the range $\{-V_{max}, V_{max}\}$ to prevent the PSO from leaving the search space. The value of $V_{max}$ proved to be crucial, because large values could result in particles moving past good solutions and create excessive crowding or bumping around the best fit particle. Higher bumping could result in forming similar particles that would obstruct the exploration of new features in an image, causing premature convergence [Eberhart *et al.*, 1996]. In the experiments the $V_{max}$ of 2 was observed to be a good value to fine-tune the entire search space with 200 particles. Setting the value of $V_{max}$ to 40 returned poor recognition rates and this will be shown in an experiment in a later section.

### 6.5.3 Inertia Weight

The inertia weight is employed to control the impact of the previous history of velocities on the current velocity. In this way, the parameter $\omega$ regulates the trade–off between the global (wide–ranging) and local (nearby) exploration abilities of the swarm. A large inertia weight facilitates global exploration (searching new areas), while a small one tends to facilitate local exploration, i.e. fine–tuning the current search area. A suitable value for the inertia weight $\omega$ usually provides balance between global and local exploration abilities and consequently a reduction on the number of iterations required to locate the optimum solution. In 1998 Shi and Eberhart investigated modifications [Shi and Eberhart, 1998b] to PSO to improve the algorithm's local search characteristics. In a modified PSO they introduce $\omega$, their

inertia factor, to dampen the velocities of the particles. Although they discuss $\omega$ as being implemented in both constant and time dependent variations, the latter, as a linear decreasing function of time, showed the most promise. Thus a time decreasing inertia weight value can be a good choice as used by [Lovbjerg and Krink, 2002] where inertia parameter was decremented linearly with number of iterations from 0.7 to 0.4. As the PSO presented in this thesis could terminate at any iteration, the inertia parameter was chosen to be a constant value of 0.7.

### 6.5.4 Cognitive and Social Parameter

The cognitive ($\psi 1$) and social ($\psi 2$) parameters are not critical for PSO's convergence. Dropping the social component from equation ( 6.1) results in the Cognition-Only Model ( 2.3), whereas dropping the cognition component defines the Social-Only Model ( 2.4). In [Kennedy, 1998], Kennedy asserts that the sum of the values of the cognitive and social components of the PSO ($\psi 1$ and $\psi 2$) should be about 4.0, and common usage is to set them each 2.05 each. However, in an earlier work [Kennedy, 1997], Kennedy also looked at models where the two components had varying values, specifically, zero for the social component (cognition-only model), zero for the cognitive component (social-only model), and setting the two equal (full model). In that work, he found a performance advantage to the social-only model. Typically both the cognitive and social parameters are set to a value of 2 [Eberhart et al., 1996], although assigning different values sometimes leads to improved performance [Suganthan, 1999]. In most of the experiments presented in this thesis both the values were set to 1. Setting the values to 2 made no noticeable difference.

## 6.6 Overall recognition rates by PSO

The particle swarm was applied to optimise the input connection maps of the n-tuple classifier. The network was built out of 150 tuples with the tuple-size 8. The task was

to recognize handwritten characters from the NIST database. The numeric data set consists of digits was used for the experiments. Each character was a binary image with the dimension 32 by 32. The target of each optimisation method was to find 150 input maps or tuples with reasonably high fitness values for different classes. 150 maps were distributed among 10 classes according to the difficulty associated with each class to recognize it. So the most difficult class gets the highest number of tuples to be optimised. The strategy was to distribute the total available tuples ($R=150$) among classes proportionately to the error rates. The overall recognition rates, the average of all recognition rates of all classes, were sought. The overall recognition rate by PSO was compared with the RnP and randomly selected approaches described in the previous chapter. The rates found by all approaches were mean of ten runs. The best recognition rate by any algorithm in ten runs was also recorded. All training methods are listed in the Table 6.1. The first training approach in the table, which is the random selection process of tuples, is the conventional way of training an n-tuple network. The second approach shown in the table is the hill climbing type stochastic process as described in [Azhar and Dimond, 2004a]. Results corresponding to the pure particle swarm based training are listed in indexes 3 and 4 of Table 6.1. The PSO had no neighbourhood restriction, meaning that each particle can affect all other particles. Swarm velocity also plays important role. In the experiments the $V_{max}$ of 2 was observed to be a good value to fine-tune the entire search space with 200 particles. Setting the value of $V_{max}$ to 40 returned poor recognition rates as shown in indexes 4 of Table 6.1.

The best-performed PSO (index 3 of Table 6.1) was compared against a second algorithm from Table 6.1 and results were tabulated in Table 6.2. The null hypothesis for the test was "average recognition rate by the PSO ($X$) is higher than any second algorithm ($Y$)". Degrees of freedom (Section 4.4) for the test were 18 as 10 runs for each algorithm was used. In the t-test (Section 4.4.1), a t-value was

**Table 6.1 Recognition rates of n-tuple networks with Optimised Tuples**

| Index | Training algorithm for n-tuple network Total available tuples, R=150 and tuple-size, n=8 | Average Recognition Rate (%) | Best Recognition Rate (%), in 10 runs |
|---|---|---|---|
| 1 | Conventional random selection approach [Bledsoe and Browning, 1959] | 80.93 | 82.83 |
| 2 | RnP based stochastic approach [Azhar and Dimond, 2004a] | 83.67 | 84.50 |
| 3 | PSO ($\psi 1$=1, $\psi 2$=1,$V_{max}$=2, $\omega$=0.7,$Q$=200) | 84.82 | 85.35 |
| 4 | PSO ($\psi 1$=1, $\psi 2$=1, $V_{max}$=40, $\omega$=0.7,$Q$=200) | 82.78 | 83.76 |

**Table 6.2 Results of Student's t-test between PSO ($X$) and a second algorithm($Y$)**

| Index from Table 6.1 | Algorithm $Y$ | $t_{exp}$-value | p-value |
|---|---|---|---|
| 1 | Conventional random selection approach | 14.87 | 1.00 |
| 2 | RnP based stochastic approach | 5.06 | 1.00 |
| 4 | PSO ($\psi 1$=1, $\psi 2$=1, $V_{max}$=40, $\omega$=0.7,$Q$=200) | 9.08 | 1.00 |

calculated against the null hypothesis and compared with the tabulated values at difference confidence levels [Kreyszig, 1970] with 18 degrees of freedom. To understand the significance of the result the t-values in Table 6.2 have to be looked at. The values show that the increases in recognition rates by the PSO (index 3 in Table 6.1) over the conventional random selection, the RnP based stochastic approach and the version of the PSO with a high $V_{max}$ (index 4 in Table 6.1) are statistically "very highly significant" as the observed t-values for all of these cases were greater than 3.92 ($t_{th}$). The p-value in Table 6.2 indicates the probability of observing the result by chance, given that the null hypothesis is true. Small values of probabilities cast doubt on the validity of the null hypothesis. The p-values in Table 6.2 for all the cases were 1, which indicates that the null hypothesis "average recognition rate by the PSO ($X$) is higher than any second algorithm ($Y$)" is extremely valid.

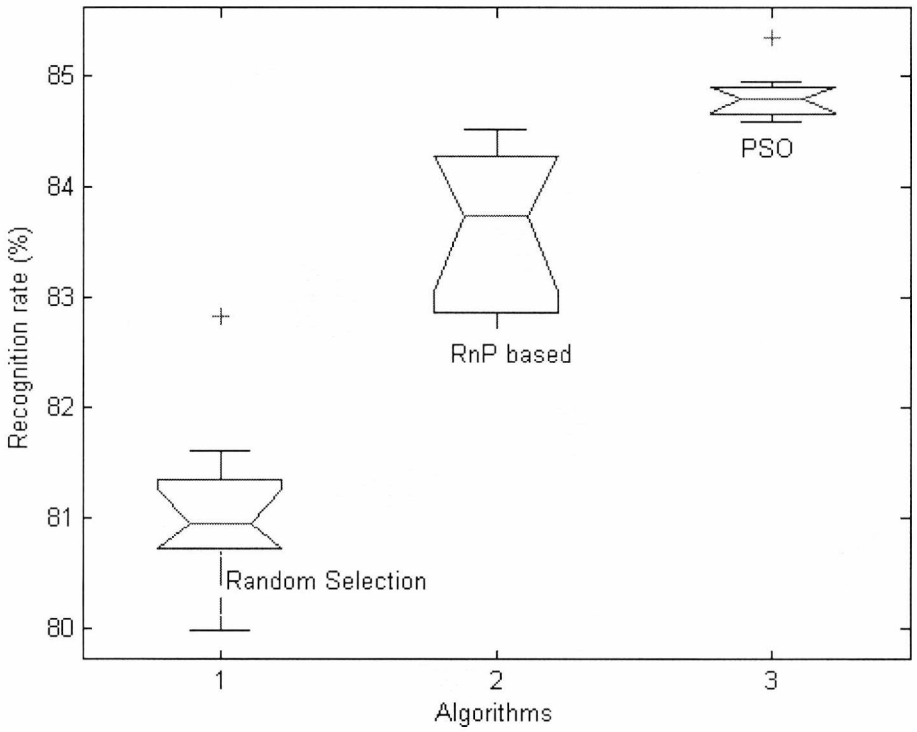



**Figure 6.5 Box Plots of PSO, RnP and Random selection**

The above figure visually compares the samples of recognition rates observed by applying three different approaches. The median of a distribution is shown as a line across the box in the figure. The median of recognition rates for PSO can be noted from the figure as near 85%, for RnP near 84% and for randomly selected approach near 81%. Clearly the distribution of results for PSO is clustered in the area with reasonably higher values of recognition rates and it shows better performance than the other two. The box plot of PSO also shows an unusual observation or outlier likewise the random selection approach.

## 6.7 Comparing PSO with GA

Results in the previous section showed how Particle Swarm (PS) could successfully find an optimal set of input maps. Like swarm intelligence the Genetic Algorithm is also a population based search technique and seems to be a good candidate to find input maps for n-tuples. GA has proven to be accomplishing the same goal as the PS optimisation [Kennedy and Spears, 1998]. It is also important to investigate the similarities and differences between the two algorithms for optimising n-tuple network. To do this task the GA will be applied to search for better maps of n-tuples for recognizing binary handwritten characters. Exactly same experimental set-up (Section 6.6) will be used for GA as it was used for the PSO, only difference will be the search algorithm. GA based tuple selection technique will be explained in the next section. Later this algorithm will be used to find the recognition rates for handwritten characters and it will be compared against the rates for the PSO. The foundations, differences and relative performance of the GA and the PS based optimisation for n-tuples will be investigated. Alongside the performances the convergence properties for both the algorithms will be compared too.

### 6.7.1 GA Based Tuple Selection

Genetic Algorithm introduced by [Holland, 1975] is an adaptive search strategy based on a highly abstract model of biological evolution to find a possible solution in a given problem space. It consists of a number of individuals refining their candidate solutions by interaction and adaptation. The individual is termed as 'chromosome'. For an n-tuple network each chromosome corresponds to each input map. If each map points to "*n*" locations of the input matrix then the chromosome will be formed with these *n* location-values called "genes" [Holland, 1975]. While GA is applied to the n-tuple network, a population of individual input maps is initialised and then evolved from generation *t* to generation *t + 1* by repeated applications of fitness evaluation, selection, recombination and mutation.

**Table 6.3 GA Operators**

| Number of fit tuples or maps or chromosomes | Number of parent chromosomes | Crossover probability | Mutation rates |
|---|---|---|---|
| 1 | 1 | 1 | 87.5% |
| 2 | 2 | 0.5 | 75% |
| 3 | 3 | 0.33 | 62.5% |
| 4 | 3 | 0.33 | 62.5% |
| 5 or more | 4 | 0.25 | 50% |

Initial population of maps is generated at random. Every evolutionary step (generation), the input maps in the current population will be evaluated according to

some predefined quality criterion, referred to as the fitness, which is equated with goodness of solution. A simple function ( 5.1) was used to measure the fitness. It is important to keep the fitness function simple as this does not eliminate the autonomy of evolution [Floreano and Mondada, 1996]. Any map that has a fitness value greater than a threshold ( 5.2) is considered as a fit map or tuple. All fit maps in the current generation are passed to the next generation without any modification. This helps passing the goodness of one generation to pass to the next generation. Maps that are not fit undergo crossover and mutation operation of GA. Before crossover partner selection is implemented. Selection is the competition among individuals of the population to become parents of the next generation. The fitter a member of a population the more likely it is to produce an offspring. Section 2.5.3 explained different selection techniques.

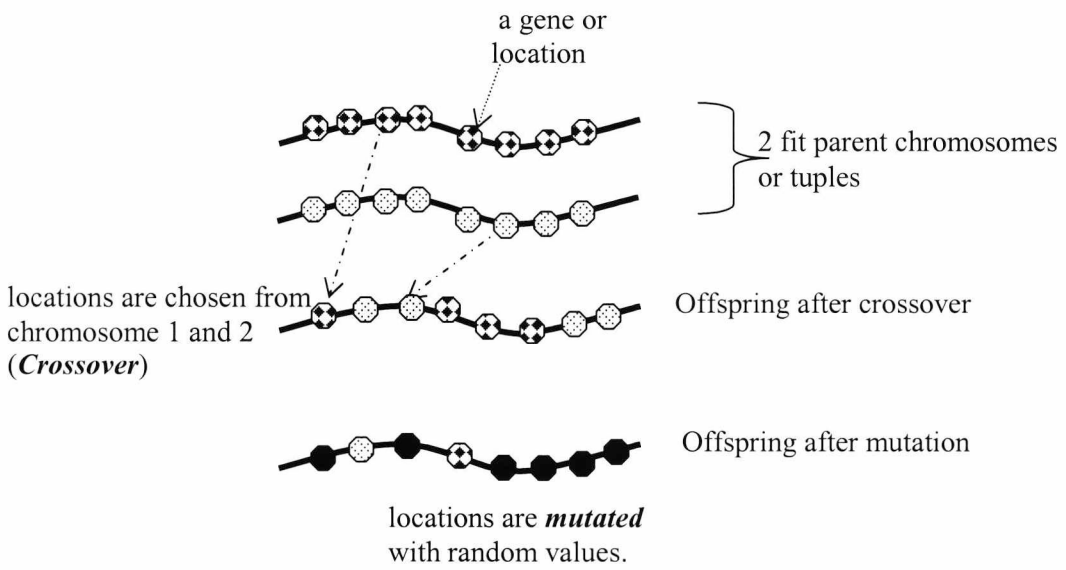



**Figure 6.6 Crossover and mutation in Tuples**

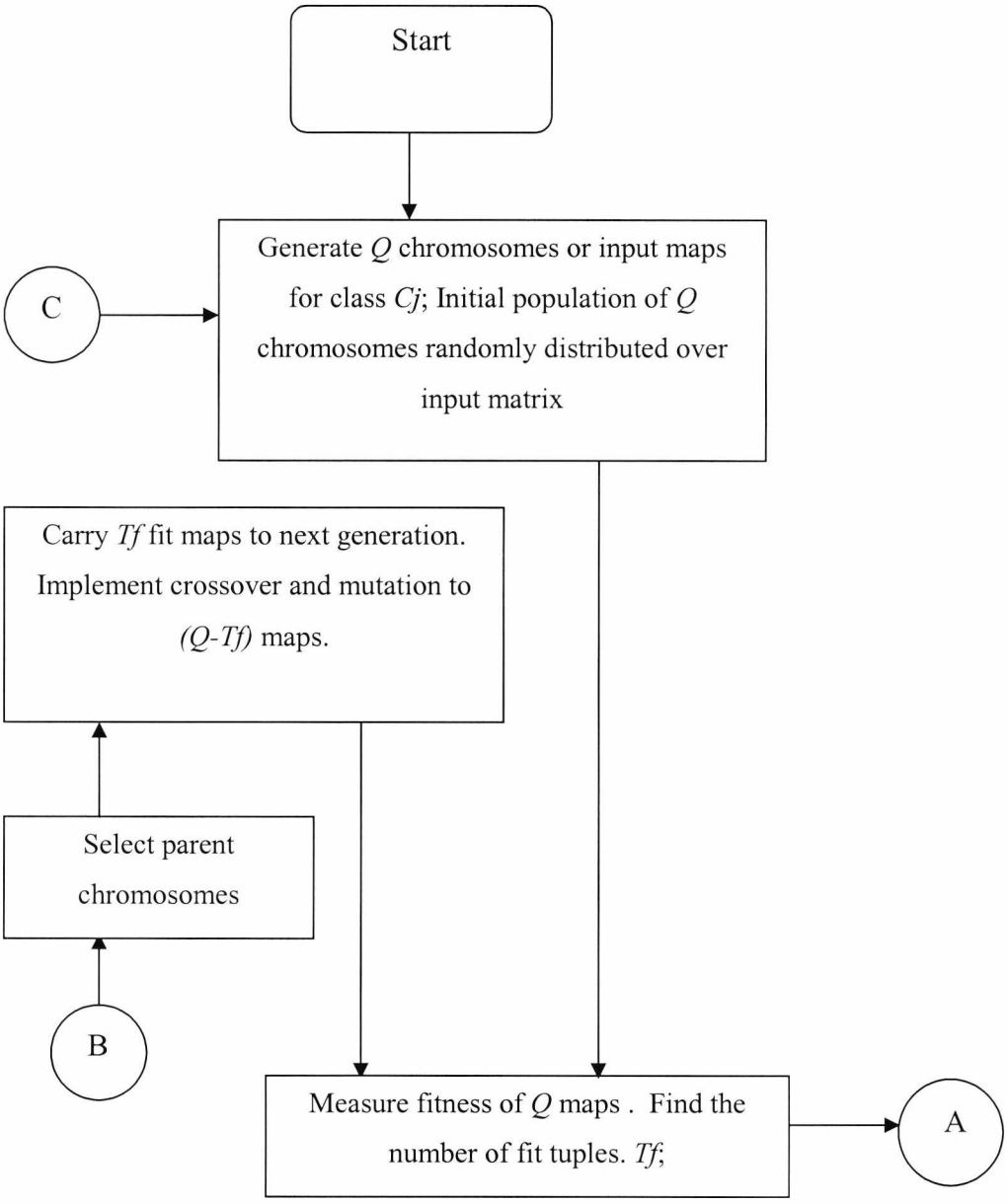



**Figure 6.7 Flow Chart (Part1) of the GA based search algorithm**

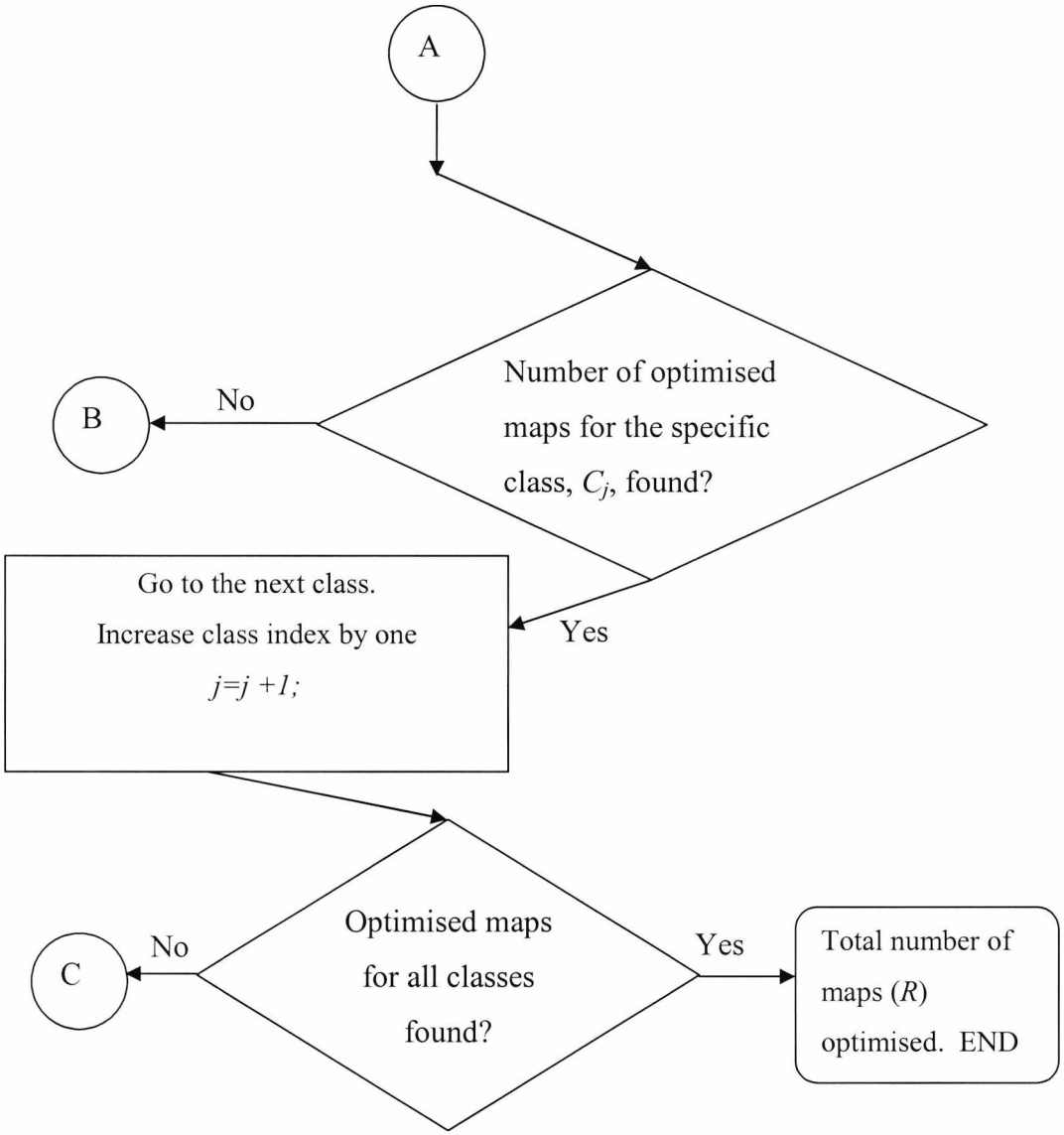



**Figure 6.8 Flow Chart (Part2) of GA based search algorithm**

The partner selection strategy followed for this research was as simple as choosing the top two, three or four input-maps with high fitness values as the parent chromosomes. This selection process is known as 'elitism' and was described in

Section 2.5.3. Elitism requires that the current fittest member (or members) of the population is not deleted and survives to the next generation [Tomassini, 1995]. How many chromosomes will be considered as parent chromosome will depend on the number of fit input maps at the current iteration. A conditional check was made at partner selection stage. If there were 5 or more fit tuples then the top 4 tuples were selected as parent tuples. For 3 or 4 fit tuples the number of parent chromosome was 3. For one and two fit tuples the number of parent chromosomes were one and two respectively. These numbers are also shown in Table 6.3. Choosing few input-maps as parent chromosomes instead of only one best chromosome allowed some diversity to be added in the system. [Eiben *et al.*,1994] found that increasing number of parents improves performance, but only for a certain number of parents and after this performance decreases. The optimal number of parents depends on the size of the search space [Eiben *et al.*,1994].

Genetically-inspired operators like crossover and mutation are used to introduce new individuals into the population [Holland, 1975]. Section 2.5 explains different mutation and crossover techniques. Mutation keeps genetic diversity in the population [Back *et al.*, 1997]. Mutation implements a random change in the value of one or more genes for introducing new information into the system. For the experiments presented in this thesis while implementing crossover genes to offspring was copied from parent's chromosomes. One of the $g$-th ($g = 1,..,n$, where $n$ is the chromosome length used for the n-tuple network) genes of the parents was selected randomly to be the $g$-th gene of the child. This is known as uniform scanning crossover. [Eiben *et al.*, 1995] introduced gene scanning as a reproduction mechanism that generalizes classical crossovers like uniform crossover and is applicable to an arbitrary number (two or more) of parents. Crossover probabilities for different number of parents are listed in Table 6.3. The process of crossover and mutation is illustrated in Figure 6.6 where two parent chromosomes are shown. Each

chromosome has 8 location-values or genes. An offspring was created by choosing genes from both parents with a crossover probability of 0.50. Mutation was realised by replacing a gene of the new offspring with a randomly selected location-value from the input matrix. In the figure small black circles show mutated genes. Large mutation rates with elitist selection turned out to be remarkably superior to that of traditional GA to obtain the global optimum solution effectively [Shimodaira, 1996]. Mutation rates used by [Gracia, 2004] in three different topologies of n-tuple networks were 95%, 60% and 75%. A variable mutation rate would theoretically make use of a high rate to speed up evolution until a certain "fitness level" is achieved, and then reduce mutation in order to increase the average fitness and produce a more balanced population [Harvey, 1993]. Mutation rates used for GA are listed in Table 6.3. For one parent chromosome only mutation (asexual crossover) was used with very high mutation rate of 87.5%. Mutation rates were reduced from 87.5% to 50% as more parent chromosomes were found. Reduction in values of mutation rates also enables to find the global optima by performing local search using good solutions obtained so far [Shimodaira, 1996].

### 6.7.2 Flow chart of GA on n-tuple system

Figure 6.7 and Figure 6.8 shows the GA based tuple selection algorithm in a flow chart. The algorithm starts by creating $Q$ number of input maps randomly for a class $C_j$, where $j$ is the class index and $Q$ is the population size. The $Q$ maps are evaluated according to the fitness equation (5.1) and threshold equation (5.2). A number of fit tuples, denoted by $Tf$, will emerge from this evaluation. If $Tf$ is less than the required number of class specific tuples for class $C_j$, then GA operators comes into play. First $Tf$ fit tuples of the current generation are passed to the next generation without any change and rest of the maps ($Q$-$Tf$) go through crossover and mutation operation. Before crossover parent chromosomes are chosen from the fit maps. After

mutation all $Q$ maps are evaluated again and previous steps are repeated or loop through until all optimised maps for all classes are being sought.

### 6.7.3 Comparing overall recognition rates of GA and PSO

An experiment was conducted to train an n-tuple network by GA and then use that trained network to recognize handwritten characters (0 to 9) from the NIST database. To compare the result with previously found results same experimental set-up was used as it was used in Section 6.7. Like before number of tuples in the network was 150 and tuple size was 8. Recognition rates for characters 0 to 9 were found separately by the GA trained n-tuple classifier and then an average of these rates were calculated to find the overall recognition rate. The experiment was repeated for ten times and results were averaged to calculate an average overall recognition rate and it was found to be 84.17 and the best recognition rate in ten runs was 84.96. Results revealed that the GA based approach performed better than conventional random selection and the RnP based method, but it couldn't outperform the PSO.

**Table 6.4 Results of Student's t-test between GA ($X$) and a second algorithm($Y$)**

| **Algorithm $Y$, Index from Table 6.1** | **$t_{exp}$-value** | **p-value** |
|---|---|---|
| 1 | 11.43 | 1.00 |
| 2 | 2.02 | 0.97 |
| 3 | -4.67 | 9.50e-005 |

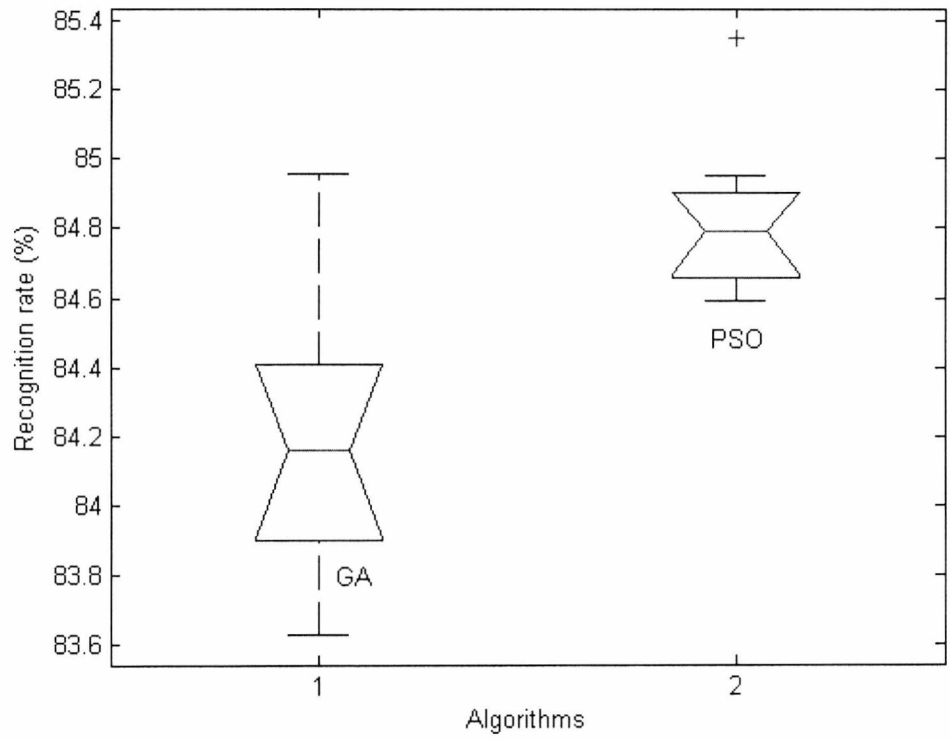



**Figure 6.9 Boxplot of GA and PSO**

Statistical significance of the results was analysed by a student's t-test. GA was compared with the other algorithms and the results were tabulated in Table 6.4. The null hypothesis for the test was "average recognition rate by GA ($X$) is higher than any second algorithm ($Y$) from Table 6.1". The t-value against the conventional random selection approach was much higher than the tabulated t-value of 3.92 (for 18 degrees of freedom) and it could be stated that the improved results of the GA over the random selection approach was "very highly significant". GA didn't perform well against the PSO. The t-value against the PSO is negative in the table, which means that the null hypothesis, $X>Y$, should be rejected and rather the alternative ($Y>X$) is true. Thus statistically the improved result of PSO over GA was "very

highly significant". The t-value against the RnP based approach is 2.02, which indicates that the mean of the GA is higher than the mean of the RnP at 90% confidence level. The p-values in the table indicate the probability of observing the result by chance given that the null hypothesis is true. A very small p-value against PSO agreed that the fact that the null hypothesis against PSO is invalid.

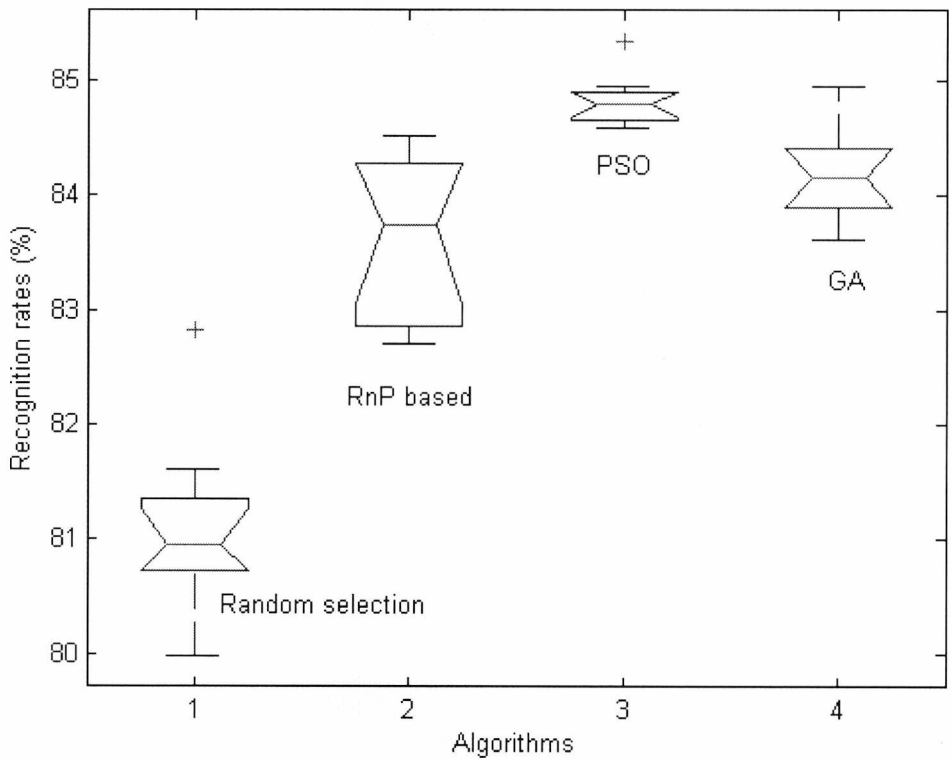



**Figure 6.10 Boxplot of several algorithms**

Figure 6.9 displays the side-by-side box plots of the GA and the PSO. Data samples of GA or PSO were obtained by running each algorithm for ten times. The notches in the figure are drawn about the median. Median of recognition rates for PSO was significantly higher (with 95% confidence) than the median for GA. Figure 6.10 portrays location and variation changes between different data groups of all

previously used algorithms to train n-tuple network. It can be noted that the recognition rates by conventional random approach were clustered at the bottom of the plot whereas data for PSO were clustered at the very top. GA was second in performance and RnP was third.

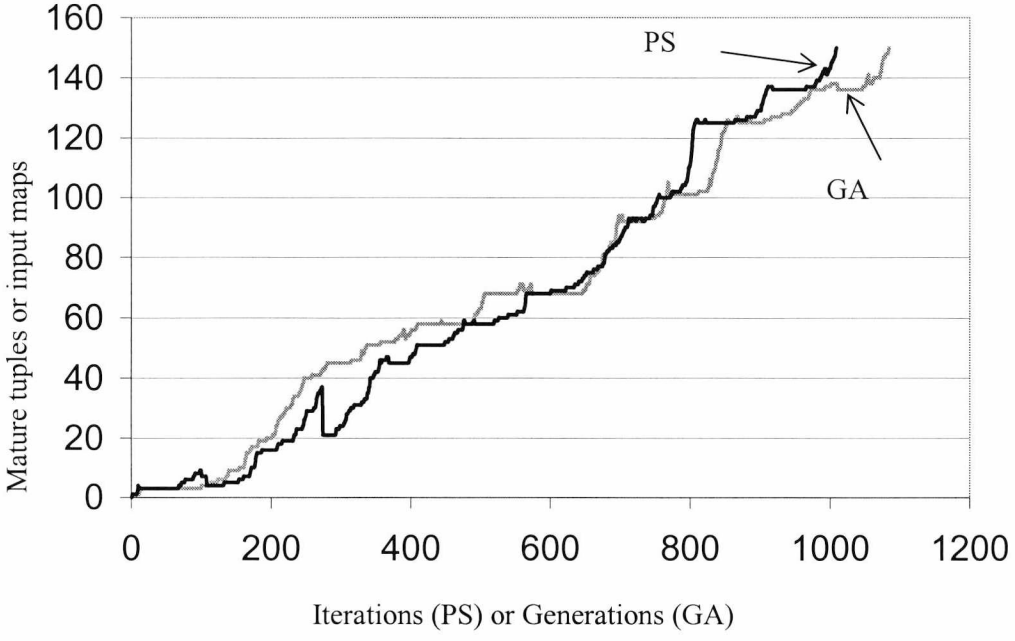



**Figure 6.11 Tuple maturity curve**

## 6.7.4 Convergence properties between PSO and GA

It has been found in the experiments that both the Particle Swarm and Genetic Algorithm can improve the recognition power of the n-tuple network. The superiority of the PS over GA was statistically very highly significant. The target of the experiment was to find 150 optimal maps for the n-tuple network. It took some time to find these maps. In GA this time was measured by the number of generations and

in PS it was measured by number of iterations. The more iterations or generations the system takes to find the 150 maps the more time is taken by the system to complete the search or converge. So the number of iterations or generations required to find the optimum number of maps measures the convergence time. Both GA and PS were executed for ten runs. Figure 6.11 compares the speed the convergence of GA and PS in finding 150 optimum maps. The curve in the figure is termed as the "tuple maturity curve". The best performed GA and PS in ten runs were selected for this comparison. PS took 1009 iterations to converge while the GA took 1084 iterations. So the PS converged slightly faster than the GA.

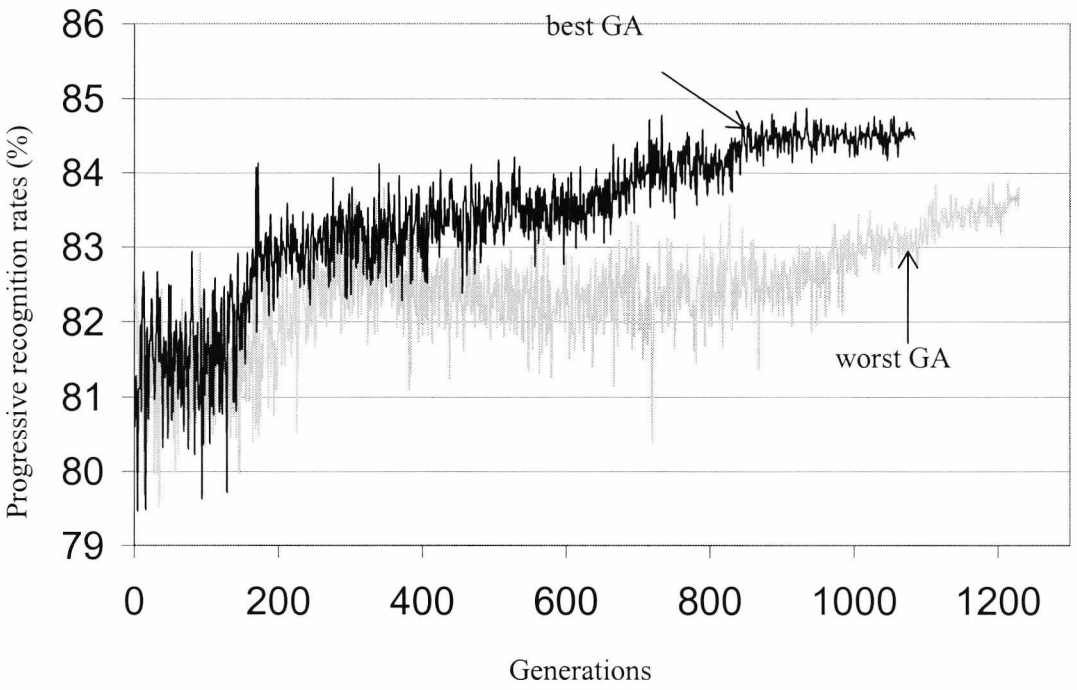



**Figure 6.12 Progressive recognition rates when optimised by GA**

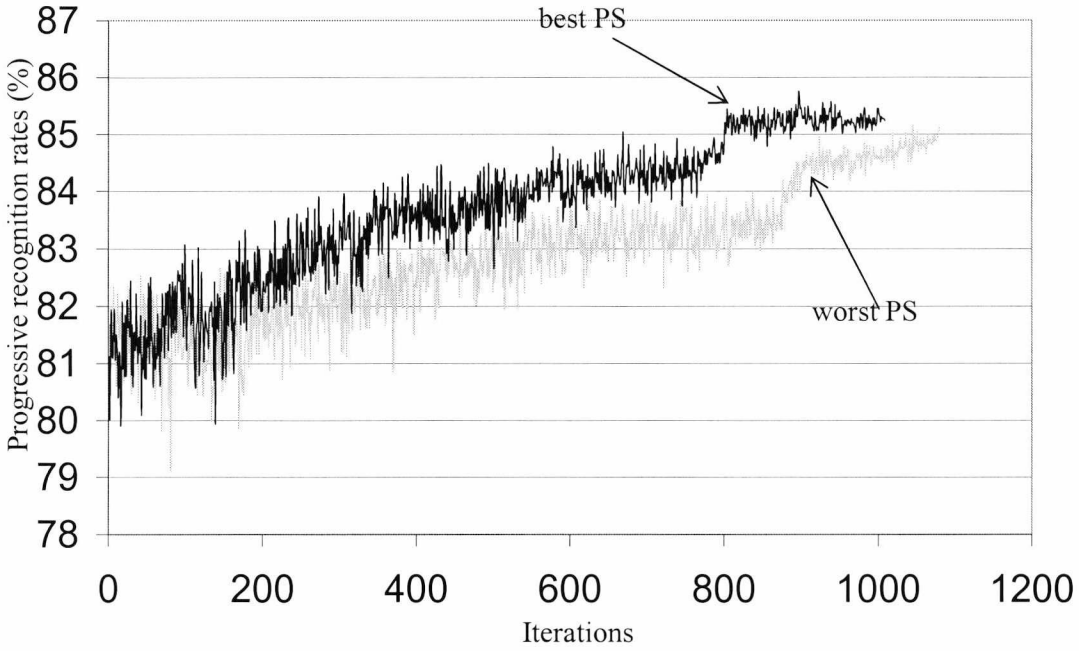



**Figure 6.13 Progressive recognition rates when optimised by PS**

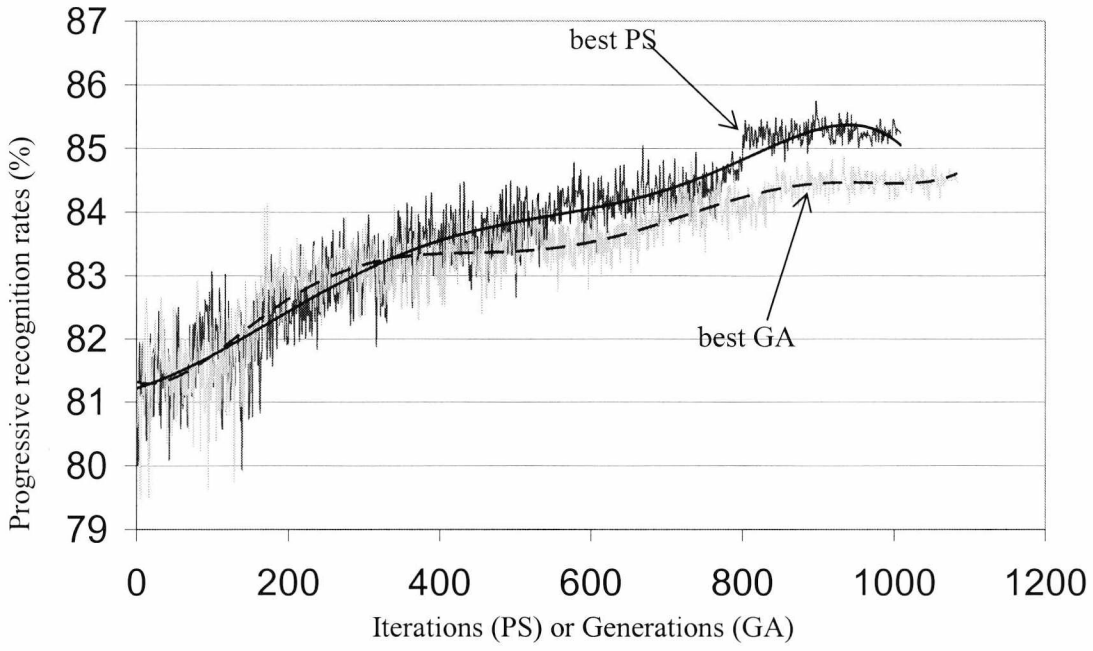



**Figure 6.14 Progressive recognition rates by the best PS and GA**

Figure 6.12 and Figure 6.13 present the progressive recognition rates by both algorithms as the search progresses. Progressive recognition rates were calculated by using 150 tuples at any iteration. Before the convergence out of this 150 tuples some will be matured tuples (found at any iteration) and the rest will be randomly selected. An overall recognition rate found from the mix of 150 matured and randomly selected tuples constitutes a progressive recognition rate. In Figure 6.14 the best run search by both the algorithms is compared. A polynomial trend line is drawn along each curve to clearly show the difference in values between the PS and GA. From the results shown in Figure 6.12, Figure 6.13 and Figure 6.14, it is clear that the PS takes slightly less number of iterations to converge and it achieves higher recognition rate compared to the GA.

### 6.7.5 Varying time constant for faster convergence

The value of time constant shown in equation ( 5.3) is crucial to compromise between the speed and performance of the search. Had a very small $\tau$ been selected, the system would convergence too quickly and it would not be given enough time to search for better solution. The main target was to improve the performance to the highest possible level and to do so an empirical value of 100 was chosen for $\tau$ in all previous experiments, which provided enough time to run the search for exploring solutions. A high value of 100 for $\tau$ resulted in a very slowly convergent system. A complete search of 150 maps took around four hours in a 2 GHz Pentium 4 machine. It was favourable to find a way to speed up the search process without loosing any noticeable difference in performance. For this a graphical plot was made with the number of mature tuples against the iterations and this plot was named as the "tuple maturity curve".

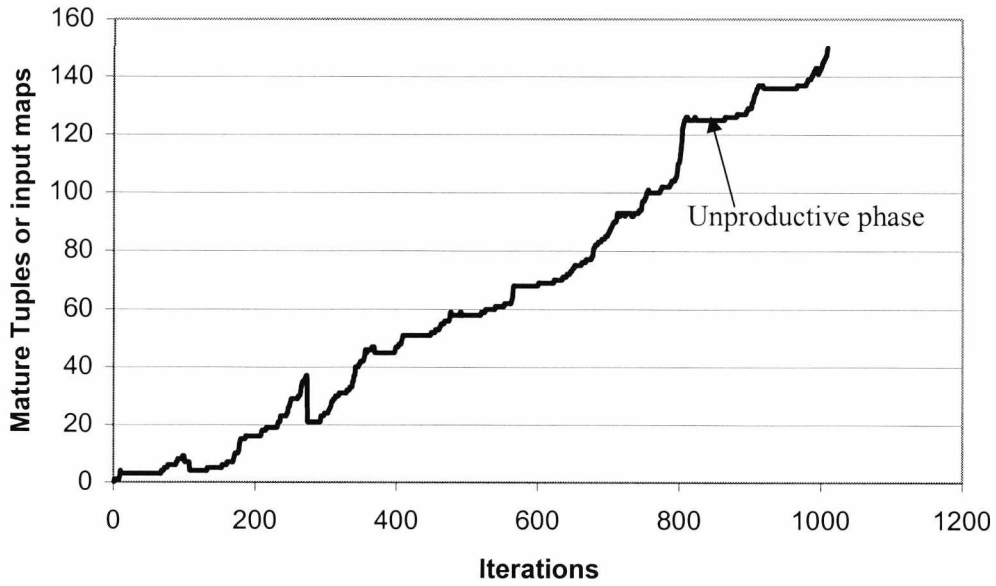



**Figure 6.15 Tuple maturity curve for a PSO run**

Figure 6.15 shows the tuple maturity curve in a typical complete search carried out with a fixed $\tau$ of 100. From the figure it can be noted that in many cases for a considerable number of iterations the system couldn't find any new map or tuple. The phase with no new map can be termed as unproductive phase (shown in Figure 6.15). One can argue that the phase was unproductive because of the threshold was too high during that time, hence unsuitable. The system had to go through this unproductive phase for quite a while until the threshold was dropped down to a suitably low value, defined by equation ( 5.2), to allow a new map to be included in the system. To reduce this time waste by unproductive phases an alternative technique was sought. The method is described below. On the tuple maturity curve (Figure 6.15) a tangent drawn at any point can tell the system if any new tuple has been discovered in the current iteration. A tangent can be measured by

finding the differences in number of matured tuples between the current iteration and the previous iteration. A zero tangent value will tell the system that the iteration that has been just finished was unproductive and the threshold equation should to be penalized for this. One way to penalize the threshold equation is to drop the value of $\tau$ by a percentage. A high percentage drop in the value of $\tau$ might damage the performance of the network. This is due to the stochastic nature of the search algorithm. The equation for penalizing an unproductive iteration was formed and it was as follows, where $t_u$ represents an unproductive interval:

$$\tau(t_u+1) = \tau(t_u) \times \left(1 - \frac{TD}{100}\right) \qquad \textbf{(6.3)}$$

**Table 6.5 Results for varying $\tau$**

| **% drop of $\tau$ from 100, *TD*** | **Convergence time for 10 runs (hours and mins)** | **Performance (Avg. recognition rate in 10 runs)** | **Relative performance, when compared with *TD=0*** |
|---|---|---|---|
| 0 | 37 hrs 51 mins | 84.89 | 0 |
| 5 | 9 hrs 34 mins | 84.96 | +0.07 |
| 20 | 3 hrs 49 mins | 84.40 | -0.49 |
| 50 | 1 hr 57 mins | 84.11 | -0.78 |
| 80 | 1 hr 10 mins | 83.96 | -0.93 |

To demonstrate the affect of *TD* (percentage drop of $\tau$), an experiment was conducted by incorporating equation (6.3) in the search algorithm. The task of the experiment was to find optimum input maps by using the PS method. Table 6.5

shows the outcomes of the experiment. For 5% drop in the value of $\tau$, ten complete searches took 9 hours and 34 minutes which was almost 4 times quicker than the

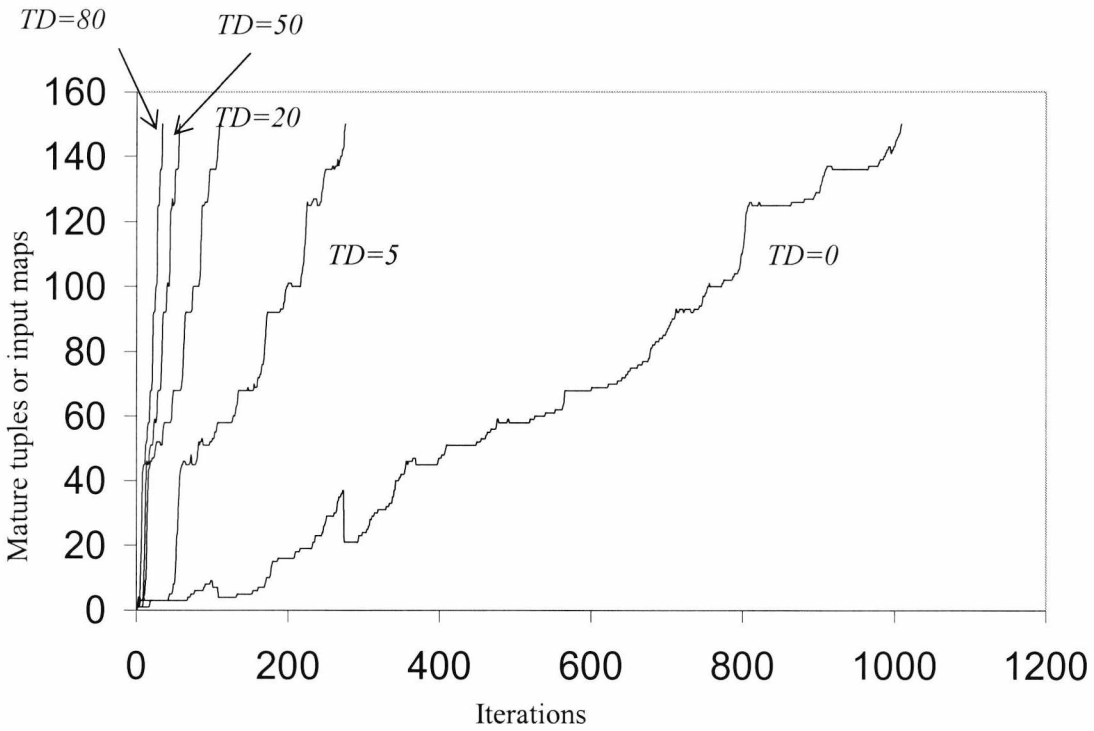



**Figure 6.16 Tuple maturity curves at different *TDs***

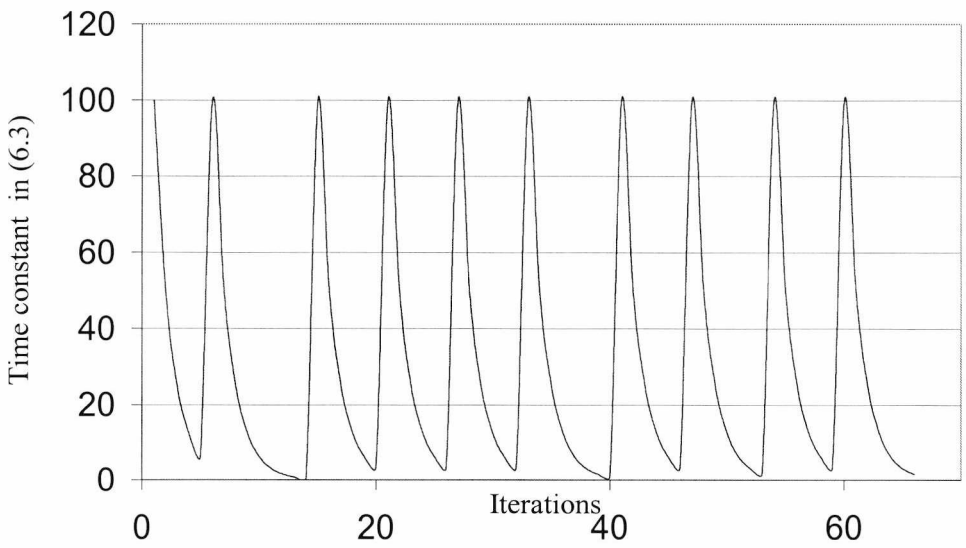



**Figure 6.17 Variation of $\tau$ for *TD=50***

previous PS search (index 3 in Table 6.1) where $\tau$ had always a fixed value of 100, in other words *TD* was zero. The performance of the n-tuple network for 5% drop in the value of $\tau$ turned out to be better than that for a fixed $\tau$ of 100. For 20% drop, the performance was reduced by 0.49 but the speed of the search was increased by more than 9 times when it was compared with a search for *TD=0*. So the general trend was: as the value of *TD* was increased, the speed of the convergence was substantially increased but the system performance was reduced to an extent. As long as the performance of the network does not get reduced to a noticeably low value, the drop in the value of $\tau$ can be acceptable for faster convergence.

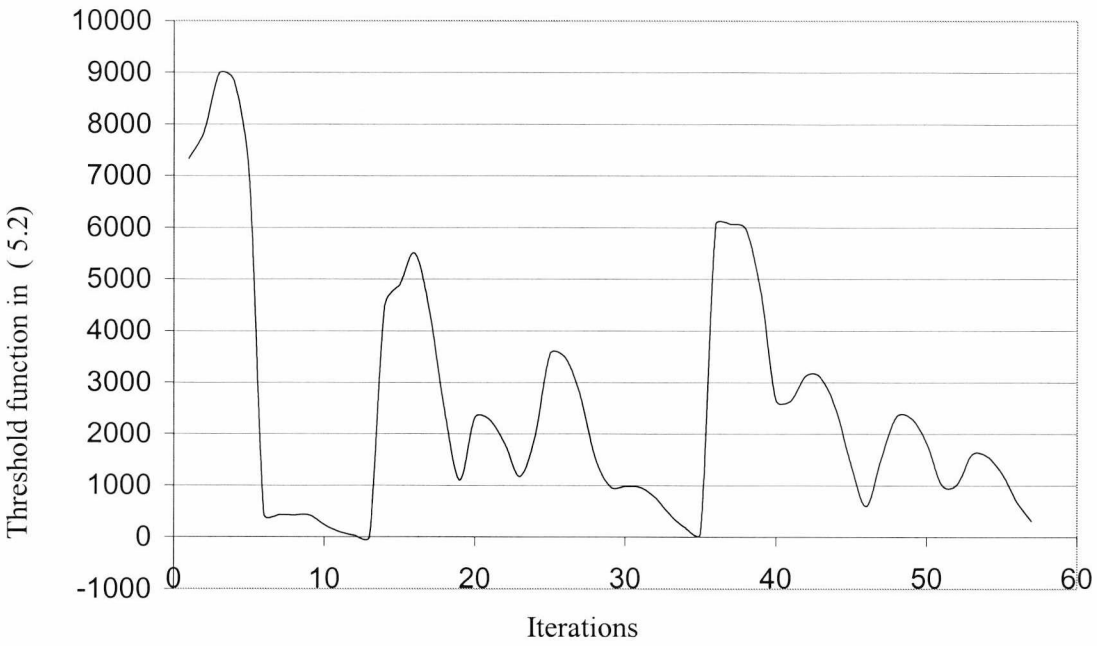



**Figure 6.18 Fitness threshold for *TD=50***

Figure 6.16 plots the number of optimum maps found at any iteration for different values of *TD*. From the figure it can be seen that the number of iterations required to find the 150 tuples is much less when the *TD* has a high value. Figure 6.17 shows a run from the experiment where the value of $\tau$ was varied throughout the

search with *TD=50*. As different input maps are sought for different class groups, in Figure 6.17 the value of $\tau$ is initialised to 100 every time the search algorithm switches from once class group to the next class group and after that the $\tau$ is reduced according to (6.3) with *TD=50*. Figure 6.18 shows how the fitness threshold described in ( 5.2) varies throughout a search when *TD* in (6.3) is fixed to 50. Data for Figure 6.15, Figure 6.16 and Figure 6.17 were taken from the same experiment.

## 6.8 Summary

This chapter described in detail the implementation of the particle swarm based tuple selection technique. Suitable values of the parameters for the algorithm were explained. PS optimised n-tuple network was used to recognise handprinted characters from the NIST database. Statistical analysis of the results showed that the improvement in recognition performance by the PS optimised network over the RnP based stochastic approach (described in Chapter 5) was very highly significant.

This chapter also provided a detailed comparison between the PSO and GA. From the procedure, it can be learnt that the PSO shares many common points with the GA. Both algorithms start with a group of a randomly generated population; both have fitness values to evaluate the population. Both update the population and search for the optimum. However, PSO does not have genetic operators like crossover and mutation. Particles update themselves with the internal velocity. They also have memory, which is important to the algorithm. There are control parameters involved in both GA and the PSO, and appropriate setting of these parameters is a key point for success. Section 6.7.3 demonstrated that the PSO performed better than GA and the improvement was statistically very highly significant. Later in this chapter the convergence characteristics of both the algorithms were presented for selecting input maps of the n-tuple network. Both PSO and GA were conducted for ten runs. The best run and worst run curves for GA were wider apart in Figure 6.12 compared to the

distance between the curves in Figure 6.13. This demonstrates that the results of the PS were more consistent than the results of the GA. It was also found that dropping the value of $\tau$ (6.3) during unproductive iterations can substantially increase the speed of the search. In the experiment a 5% drop in the value of $\tau$ from 100 was advantageous with slightly higher recognition rate in less number of iterations. Results reveal that it is not only the speed but the performance of the system has to be looked after also and to avoid a premature convergence with a low recognition rate the time constant has to be varied very carefully.

# Chapter 7

# Hybridising Particle Swarm

## 7.1 Swarm Diversity

Although, in general, PSO results in good solutions, in high–dimensional spaces it might stumble on local minima [Kalyan *et al.*, 2003]. It may be argued that many of the particles are wasting computational effort in seeking to move in the same direction (towards the local optimum already discovered), whereas better results may be obtained if various particles explore other possible search directions. Repetition of the same positional value of a pixel in forming a particle can obstruct the exploration of new features in an image. Again two particles with same pixel-location values will overlap. If any two particles have the same pixel-location values in all dimensions then they will completely overlap with each other and carry same information to the n-tuple network. Different particles with the same information will not eventually benefit the system. Thus some sort of diversity in the PSO could be helpful to achieve better recognition rate. One way to add diversity in PSO is to use the Self-Organized Criticality [Bak, 1996]. Extending the PSO with SOC seems very promising reaching faster convergence and better solutions [Løvbjerg and Krink, 2002]. Another way of improving the PSO is by hybridising it with a technique that considers the neighbourhood interactions, which is naturally observed and expected in animal

behaviour [Peram *et al.*, 2003; Kalyan *et al.*, 2003]. Hybridisation refers to combining different approaches to benefit from the advantages of each approach. Section 7.3 explains the algorithm described in [Peram *et al.*, 2003] which considers the nearest neighbour interactions in PSO. Implementation of the SOC algorithm [Løvbjerg and Krink, 2002] will be introduced in the next section.

## 7.2 Self-Organized Criticality

[Lovbjerg and Krink, 2002] have explored extending the PSO with the SOC to improve population diversity. To understand the concept of the SOC lets consider a pile of sand (Figure 7.1). At some point, as grains of sand are slowly and steadily added, the pile becomes "critical" or unstable, and an avalanche (Figure 7.2) occurs spontaneously. In the sand-pile model grains are dropped on a lattice, they can pile up until a specified height is reached, after which they fall on the neighbouring sites. In this way avalanches propagate trough the system until they fall out of the boundaries. Now, this visual and obviously simple system is, in fact, complex (there are truly many sand grains interacting), and, as the pile grows, it must attain the point of criticality, which initiates the dramatic reorganization caused by the avalanche. [Bak, 1996] developed a simple mathematical model to simulate a growing sand pile, and it also produced avalanches.

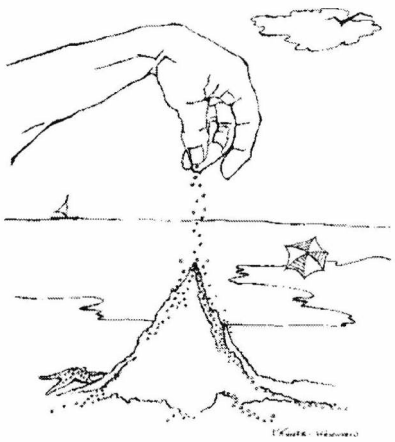


**Figure 7.1 Bak's Sand pile [Bak, 1996]**

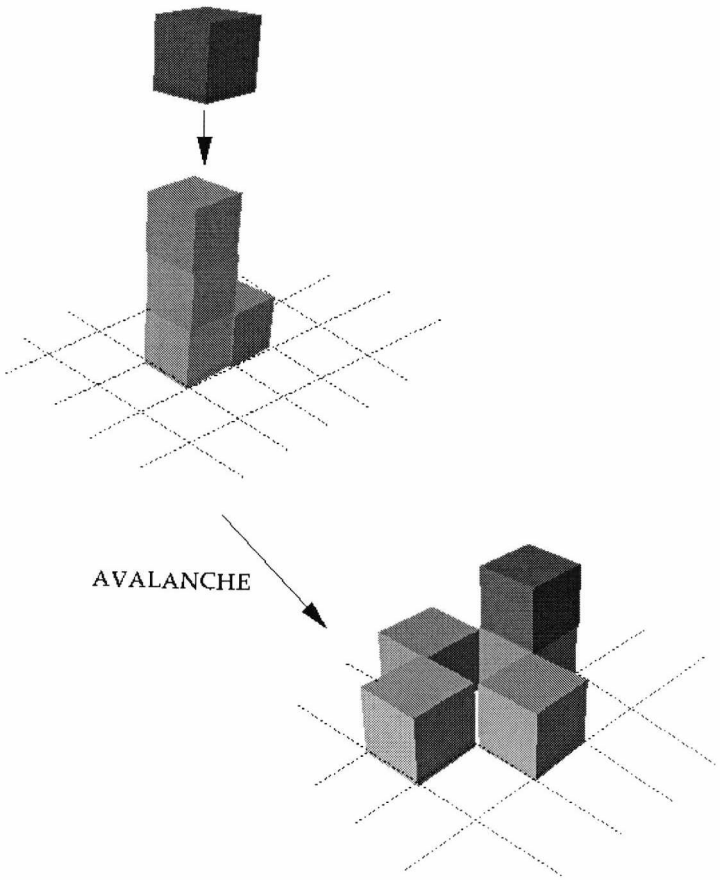



**Figure 7.2 Avalanche in sand pile model [Dickman *et al.*, 2000]**

The main idea in SOC is that most state transitions in a component of a complex system only affect its neighbourhood, but once in a while entire avalanches of propagating state transitions lead to a major reconfiguration of the system. Self-organized criticality has been found in a variety of phenomena such as earthquakes, volcanic activity, the game of life, landscape formation and stock markets. Chaotic systems can change dramatically without external influence and stable systems constantly change in very small steps. SOC describes how small amounts of external influence can occasionally lead to the big changes observed in complex systems. Evolutionary Algorithms and Particle Swarm Optimisation are models of complex

systems. In EAs for instance it can be a difficult task deciding when to apply certain operators. Self-Organized Criticality has been successfully applied to improve the performance of Evolutionary Algorithms. This was done in relation to mass extinction and mutation operator control by [Krink *et al.*, 2000; Krink and Thomsen, 2001] where extinction zones were formed (3×3 rectangles). Mutated copies of currently best individual then substituted individuals in these extinction zones. [Rickers *et al.*, 2000] used SOC in relation to spatial mating control, where most mates were immediate neighbours, but occasionally mates were selected from remote places. Occasional outbreeding improved the performance by counter balancing the effect of rigid neighbourhood inbreeding. Other aspects of SOC have been described and applied to search problems by [Boettcher and Paczuski, 1997].

The SOC-PSO algorithm used for the experiments in the research had a globally set "criticality limit", denoted by *CL*, which is the maximum number of times a position on the search space can be considered or taken in forming a particle. If the criticality value of a position on the search space exceeds this limit, the particle corresponding to that position responds by dispersing the criticality within its surrounding neighbourhood and then by relocating itself. Two types of relocation were investigated in [Lovbjerg and Krink, 2002]: the first re-initialises the particle, while the second pushes the particle with high criticality a little further in the search space. The second approach was followed in the SOC-PSO model used for this research. If the redistribution causes the criticality of the surrounding cell to be increased then process continues until criticalities of all the positions are below the maximum limit. The pseudo code of the SOC-PSO algorithm is given below:

```
begin
    initialise
    while(not terminate condition) do
      begin
```

```
run PSO{
  for i=1 to the population size Q,
      for d=1 to the problem dimensionality n,
        Apply the velocity update equation (2.1);
        Limit velocity magnitude, V_{i,d} ,(2.5);
        Update Position, X_{i,d},(2.2) and (2.6);
        criticality  [X_{i,d}]= criticality[X_{i,d}]+1;
        while (Criticality value at X_{i,d} >CL)
        {criticality[X_{i,d}] = criticality[X_{i,d}]-1;
         X_{i,d} =Disperse (X_{i,d});
         criticality[X_{i,d}] = criticality[X_{i,d}]+1;}
      End-for-d;
      Compute Fitness;
      If needed, update historical information
      regarding P_{i,d} and P_{gd};
  End-for-i;
 End
End
**********************************
Function Disperse (X_{i,d})
 {Xnew=f{ X_{i,d}, random(0 to 7)};
  return Xnew;}
```

From the pseudo code of SOC-PSO it can be seen that the SOC algorithm was implemented within the PSO loop. Once the velocity and a new positional value were found in PSO, the criticality value of the new position was being checked. If the value is more than the criticality limit then the dispersion phenomena was realized and it was implemented by choosing a new location next to the previously found position. New position's criticality value was checked again and if the value was found to be more than the criticality limit than again the dispersion will occur. Thus the dispersion continues until the system finds a location where the criticality value is less than the limit. The flow chart of the SOC is shown in the following figure.

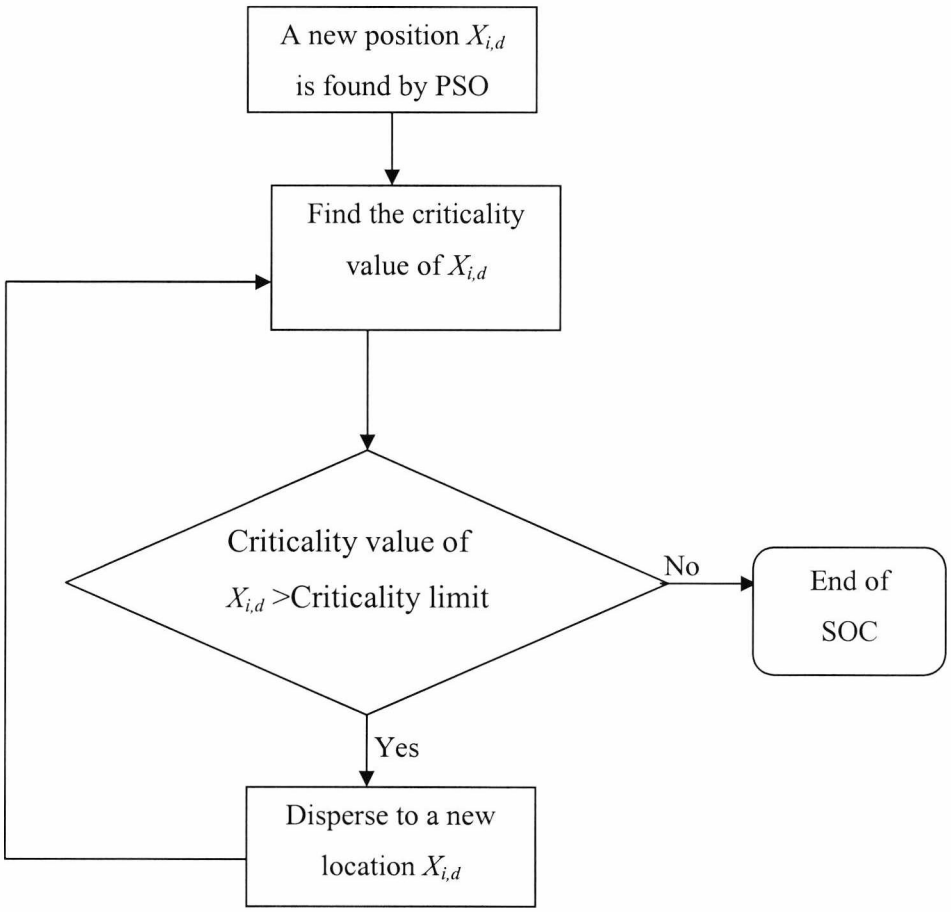



**Figure 7.3 Flow chart of Self-Organizing Criticality in PSO**

The positions on the search space can be considered as a grid as shown in Figure 7.4. $X_{i,d}$ in the figure represents a position which was found to have a critical value more than the limit. The dispersion was realized by a random jump from $X_{i,d}$ to one of its surrounding positions. There are eight possible positions to jump around $X_{i,d}$ numbered from 0 to 7. In dispersion one of the values from 0 to 7 was chosen randomly and this value will define the new position. The arrow in the figure shows the direction of jump.

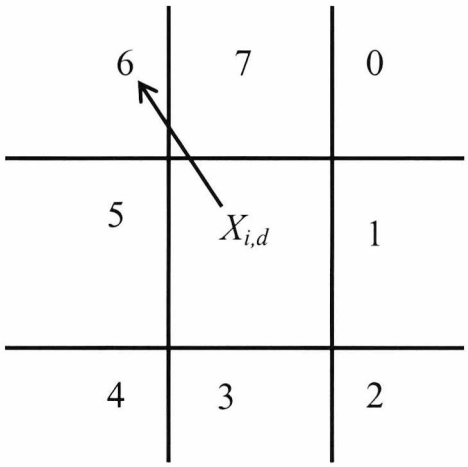



**Figure 7.4 Dispersion by Random Jump**

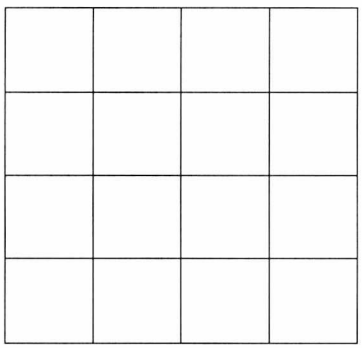


**Figure 7.5 A 4 by 4 input matrix**

Criticality limit ($CL$) can control the diversity of the PSO. A particle will disperse when criticality of any of its dimension will be more than the limit. Criticality limit has to be carefully chosen. An increased criticality limit will allow more particles to be crowded in the same location, thus will make the system less diverse. With a small value of $CL$ only fewer particles might bring the system to a critical point and results in more dispersion. Later an experiment will be carried out to count the number of times particles disperse in a search for different values of $CL$.

The total number of particles and its dimensionality also play important role in setting up a value of $CL$. To understand this lets consider a swarm system of 4 particles with dimensionality 8. Each particle sits on a 4 by 4 input matrix as shown in Figure 7.5. So there are 16 locations in the matrix. With the dimensionality of 8 each particle takes 8 places in that matrix. There are 4 particles, so the total places required by all particles are 32. Because there are only 16 positions available in the matrix so to accommodate 32 positions for all 4 particles each position needs to be repeated at least twice. So the criticality limit for this system has to be at least 2. If the limit is 1 then each position will be allowed to repeat only once, so there will be only 16 positions available and this will not be enough to accommodate 32 positions required by 4 particles. So a criticality limit 1 for this system will be invalid. The criticality limit has to be chosen carefully such that input matrix will have enough potential locations to accommodate all particles. Now if the input matrix is extended to an image area of 32 by 32 and number of particles to 200 with dimensionality 8 then the lowest criticality limit will come out as 2. This is because each location in 32 by 32 image area needs to be repeated at least twice to accommodate 200 particles. An equation can be formulated to find the lowest criticality limit. The smallest criticality limit, denoted by $CL_{min}$, can be found by the (7.1) where $W$ and $H$ are the width and height of a binary image, $Q$ is the population size and $D$ is the number of dimensions (Section 2.6.2) of particles in PSO.

$$CL_{\min} = \left\lceil \frac{Q \times D}{W \times H} \right\rceil \qquad (7.1)$$

## 7.3 Nearest Neighbour Interactions in PSO

From natural observations and expectations of animal behaviour it can be stated that the others in its neighbourhood can influence the particle's behaviour or solution. Thus conventional particle dynamics in PSO can be improved by using neighbour interactions for searching better solutions. To battle premature convergence in PSO neighbourhood interactions in PSO dynamics can be considered. A significant modification in particle dynamics is required to introduce the effects of multiple other particles in each particle. [Peram *et al.*, 2003] proposed a method where each particle is moved towards other nearby particles with a more successful search history, instead of just the best position discovered so far. This is in addition to the terms in the original PSO update equation, ( 6.1). The proposed algorithm is described as Fitness-Distance-Ratio [Peram *et al.*, 2003] based PSO (FDR-PSO) which selects only one other particle when updating each velocity dimension and which is chosen to satisfy two following criteria:

1. It must be near the particle being updated.

2. It should have visited a position of higher fitness.

In FDR-PSO each velocity dimension is updated by selecting a particle that maximizes the ratio of the fitness difference to the one-dimensional distance. In other words, the $d$-th dimension of the $i$-th particle's velocity is updated using a particle called the $P_{fdr}$, with prior best position $P_b$, chosen to maximize the following ratio:

$$FDR\ (b,i,d) = \frac{Fitness(P_b) - Fitness(X_i)}{|P_{bd} - X_{i,d}|} \qquad (7.2)$$

where $b \neq i$ and |...| denotes the absolute value. A new term was introduced into the velocity update equation with a new coefficient '$\psi 3$' and a new stochastic weight factor *'ran3'*. Like in original PSO (Section 6.2) '*ran3*' can be uniformly distributed in {0,1} or can have a constant value of 1. Note that the FDR-PSO with $\psi 3 = 0$ is the same as the usual PSO algorithm described by [Kennedy and Eberhart, 1995]. The modified velocity equation for FDR-PSO is presented below:

$$\begin{aligned} V_{i,d}(t+1) = \omega \times V_{i,d}(t) + \psi 1 \times ran1 \times (P_{i,d} - X_{i,d}(t)) \\ + \psi 2 \times ran2 \times (P_{gd} - X_{i,d}(t)) \\ + \psi 3 \times ran3 \times (P_{fdr} - X_{i,d}(t)) \end{aligned} \quad \textbf{(7.3)}$$

The pseudo-code for the FDR-PSO algorithm is given below:

```
begin
    initialise
    while(not terminate condition) do
      begin
        run PSO{
           for i=1 to the population size Q,
               for d=1 to the problem dimensionality n,
                 Apply the velocity update equation,
                     (7.3); In (7.3) Pi,d is the best
                     position visited so far by Xi,d, Pgd is
                     the best position visited so far by
                     any particle and Pfdr is chosen by
```

maximizing $\frac{Fitness(P_b) - Fitness(X_i)}{|P_{bd} - X_{i,d}|}$, where

$P_b$ is $P_{fdr}$'s previously best found position

Limit magnitude, $V_{i,d}$;

Update Position, $X_{i,d}$;

End-for-d;

Compute Fitness;

If needed, update historical information regarding $P_{i,d}$ and $P_{gd}$;

End-for-i;

End

End

From the above pseudo code it can be observed that the only difference between the PSO and FDR-PSO is the use of equation (7.3) where the influence of a third particle $P_{fdr}$ is applied. As equation (7.3) is used for every dimensionality of a particle, so each dimension of a particle will have the influence of a $P_{fdr}$. In the previous chapter it was explained that for an n-tuple classifier the dimensionality is equivalent to the tuple-size. If the tuple-size is 8 then the dimensionality of a tuple or a particle will be 8. So for a particle with dimension 8 the number of $P_{fdr}$ will be 8. The FDR-PSO algorithm was applied to optimise a set of input maps. Tuple-size used in the experiment was 8. Table 7.1 shows some of the results from an experiment where FDR-PSO was used to find better n-tuples.

**Table 7.1 Examples of third particle, $P_{fdr}$, for each dimension of a particle $P_i$**

| Particle index, $I$ | Dimension, $d$ | Index of third particle, $P_{fdr}$ |
|---|---|---|
| 1 | 1 | 95 |
| | 2 | 175 |
| | 3 | 45 |
| | 4 | 45 |
| | 5 | 153 |
| | 6 | 36 |
| | 7 | 58 |
| | 8 | 29 |
| 4 | 1 | 126 |
| | 2 | 61 |
| | 3 | 101 |
| | 4 | 45 |
| | 5 | 197 |
| | 6 | 165 |
| | 7 | 165 |
| | 8 | 95 |

Table 7.1 shows the results of only two particles (with index 1 and 4) to illustrate the fact that each particle had 8 dimensions and each dimension had a $P_{fdr}$. For example the velocity of the 5$^{th}$ dimension of the particle with index 1 was influenced by a particle with index 153, velocity of the 2$^{nd}$ dimension of the particle with index 4 was influenced by a particle with index 61 and so on. $P_{fdr}$ satisfies the equation (7.2) and this eventually influence the velocity of the dimension defined by the equation (7.3). As FDR was an extension to the original PSO algorithm, so along with the $P_{fdr}$ two other particles (global best and its own best) also influenced a particle's velocity. According to [Kalyan *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 ($\psi 1 = \psi 2 = 1$, $\psi 3 = 2$,) outperformed PSO and several other variations of PSO in different tested benchmark problems [Kalyan *et al.*, 2003].

## 7.4 Combining SOC and FDR with PSO

When the SOC algorithm is combined with the FDR and PSO a new level of hybridisation is achieved. FDR affects on the velocity of a particle while SOC pushes a particle to relocate to its neighbourhood's position due to its criticality value in the current position. A SOC-FDR-PSO algorithm is a hybrid technique where criticality values of positions are taken into account once the velocities and positions of particles are being updated by the FDR-PSO. Thus SOC-FDR-PSO can add more diversity in searching. The pseudo-code of SOC-FDR-PSO is given below. From the code it can be noted that for every dimension of a particle criticality of a position is checked and if the value is found to be more than the limit than the dispersion starts and it continues until a new position with a criticality value less than the limit is sought.

```
begin
    initialise
    while(not terminate condition) do
      begin
        run PSO{
           for i=1 to the population size Q,
               for d=1 to the problem dimensionality n,
                  Apply velocity update equation (7.3)
                  Limit magnitude, V_{i,d};
                  Update Position, X_{i,d};
                  criticality  [X_{i,d}]= criticality[X_{i,d}]+1;
                  while (Criticality value at X_{i,d} >CL)
                   {criticality[X_{i,d}] = criticality[X_{i,d}]-1;
                    X_{i,d}=Disperse (X_{i,d});
                  criticality[X_{i,d}] = criticality[X_{i,d}]+1;}
                  End-for-d;
                  Compute Fitness;
                  If needed, update historical information
                  regarding P_{i,d} and P_{gd};
           End-for-i;
      End
  End
```

## 7.5 Experimental Results

Experiments were conducted with several variations of PSO, FDR-PSO, SOC-PSO and SOC-FDR-PSO obtained by changing different parameter values like particle velocity, inertia constant ($\omega$), criticality limit ($CL$), swarm size ($Q$), stochastic weight factor etc. The network was built out of 150 tuples with tuple-size 8. So total tuples available to be optimised was 150 and this was denoted as $R$ in Section 6.2. Because the tuple-size was 8, so the dimensionality of the hybrid PSO algorithm was 8. The task was to use hybrid PSO algorithms to selectively choose tuples that describes the classes better and later use these tuples to recognise a test data set. The NIST (Section 4.2.1) database consists of handwritten digits (0,1...9) was used in the experiments. Like the experiments described in last two chapters each character was a binary image with the dimension 32 by 32. All digits were scaled into same dimension and centred. Like experiments described in the last two chapters the available tuples were distributed among classes according to the difficulty associated in recognizing the patterns. It was reported in Section 5.5 that character 1 was most difficult class to recognize, so it gets most number of tuples to describe the class. The same numbers of class specific tuples presented in Table 5.2 were used for the experiments with hybrid PSO. To compare results, the n-tuple network was trained with various methods as listed in Table 7.2. The overall recognition rates, the average of all recognition rates of all classes, were found in the experiments. The recognition rates found by different approaches were mean of ten runs. The best recognition rate by any algorithm in ten runs was also recorded. The average recognition rates by different methods are also compared in a scattered column chart as shown in Figure 7.6. Numbers on the $X$-axis of the chart correspond to the indexes of Table 7.2.

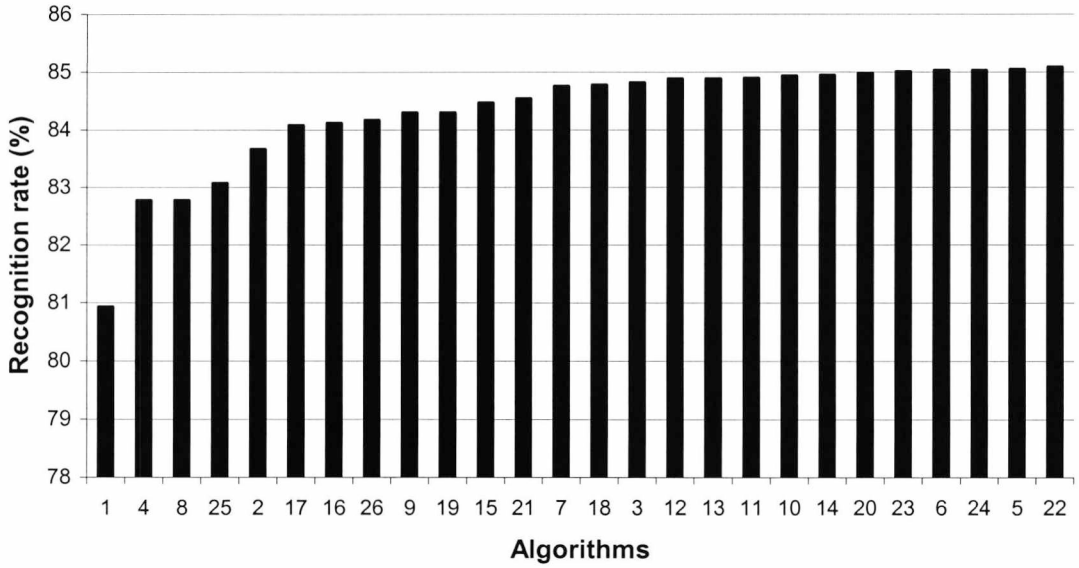



**Figure 7.6 Different approaches from Table 7.2**

Moving from the left to right of the *X*-axis of the chart gives better recognition rates. Thus the algorithm corresponding to the rightmost column gives the best recognition rate. The left most column in the chart corresponds to the traditional approach of n-tuple training, which is basically random selection process of input maps. The second approach ("RnP" based in Chapter 5) shown in the table is a pure hill climbing type method where the input maps are created randomly, so the candidate solutions neither compete nor co-operate. SOC-PSO [Lovbjerg and Krink, 2002] and FDR-PSO [Peram *et al.*, 2003] are hybrid PSO methods [Azhar and Dimond, 2004c] with added diversity. Results corresponding to the pure particle swarm based training are listed in indexes 3 and 4 of Table 7.2. Experimental outcomes clearly reveal that hybrid PSO algorithms with proper parameter settings can outperform other approaches. The best-performed algorithm, which is presented by the rightmost column on Figure 7.6, was a version of a SOC-FDR-PSO (index 22

in Table 7.2). In that algorithm $\psi 2$ was set to zero and it means one of the main components (social parameter) of the old PSO algorithm was completely deleted.

**Table 7.2 Recognition rates of the n-tuple network by different optimisation**

| Index | Training algorithm for n-tuple network Total available tuples, R=150 and Tuple-size, n=8 Population size for PSO or GA, Q=200 | Average Recognition Rate (%) | Best Recognition Rate in 10 runs (%) |
|---|---|---|---|
| 1 | Conventional random selection approach [Bledsoe and Browning, 1959] | 80.93 | 82.83 |
| 2 | RnP based stochastic approach introduced [Azhar and Dimond, 2004a] | 83.67 | 84.50 |
| 3 | PSO ($\psi 1$=1, $\psi 2$ =1,$V_{max}$=2, $\omega$=0.7, $S_{toch}$={0,1}) | 84.82 | 85.35 |
| 4 | PSO ($\psi 1$=1, $\psi 2$ =1, $V_{max}$ =40, $\omega$=0.7, $S_{toch}$={0,1}) | 82.78 | 83.76 |
| 5 | SOC-PSO($\psi 1$=1,$\psi 2$ =1, $V_{max}$ =2, $\omega$=0.7, $CL$=2,$OL$=1, $S_{toch}$={0,1}) | 85.05 | 85.71 |
| 6 | SOC-PSO($\psi 1$=2,$\psi 2$ =2,$V_{max}$ =2, $\omega$=0.7,$CL$=2,$OL$=1, $S_{toch}$ ={0,1}) | 85.03 | 85.45 |
| 7 | SOC-PSO($\psi 1$=1,$\psi 2$ =1,$V_{max}$ =2, $\omega$=0.7,$CL$=10,$OL$=1, $S_{toch}$ ={0,1}) | 84.76 | 85.07 |
| 8 | SOC-PSO($\psi 1$=1,$\psi 2$ =1,$V_{max}$ =40, $\omega$=0.7, $CL$=40,$OL$=4, $S_{toch}$ ={0,1}) | 82.78 | 83.24 |
| 9 | SOC-PSO($\psi 1$=1,$\psi 2$ =1,$V_{max}$ =10, $\omega$=0.7, $CL$=2,$OL$=1, $S_{toch}$ ={0,1}) | 84.30 | 84.80 |

| Index | Training algorithm for n-tuple network<br>Total available tuples, R=150 and Tuple-size, n=8<br>Population size for PSO or GA, Q=200 | Average Recognition Rate (%) | Best Recognition Rate in 10 runs (%) |
|---|---|---|---|
| 10 | SOC-PSO($\psi 1$=1,$\psi 2$=1,$V_{max}$=2,$\omega$=0.7, $CL$=2,$OL$=4, $S_{toch}$={0,1}) | 84.94 | 85.32 |
| 11 | FDR-PSO ($\psi 1$=1, $\psi 2$=1, $\psi 3$=2, $V_{max}$=2, $\omega$=0.7, $S_{toch}$={0,1}) | 84.90 | 85.49 |
| 12 | FDR-PSO ($\psi 1$=2, $\psi 2$=2, $\psi 3$=2, $V_{max}$=2, $\omega$=0.7, $S_{toch}$={0,1}) | 84.89 | 85.42 |
| 13 | FDR-PSO ($\psi 1$=1, $\psi 2$=0, $\psi 3$=1, $V_{max}$=2, $\omega$=0.7, $S_{toch}$={0,1}) | 84.89 | 85.34 |
| 14 | FDR-PSO ($\psi 1$=1, $\psi 2$=0, $\psi 3$=2, $V_{max}$=2, $\omega$=0.7, $S_{toch}$={0,1}) | 84.95 | 85.45 |
| 15 | FDR-PSO ($\psi 1$=1, $\psi 2$=1, $\psi 3$=1, $V_{max}$=2, $\omega$=0.9, $S_{toch}$=1) | 84.47 | 84.73 |
| 16 | FDR-PSO ($\psi 1$=1, $\psi 2$=1, $\psi 3$=2, $V_{max}$=2, $\omega$=0.9, $S_{toch}$=1) | 84.12 | 84.68 |
| 17 | FDR-PSO ($\psi 1$=0, $\psi 2$=1, $\psi 3$=2, $V_{max}$=2, $\omega$=0.9, $S_{toch}$=1) | 84.08 | 84.68 |
| 18 | FDR-PSO ($\psi 1$=0, $\psi 2$=0, $\psi 3$=2, $V_{max}$=2, $\omega$=0.9, $S_{toch}$=1) | 84.78 | 85.49 |
| 19 | FDR-PSO ($\psi 1$=1, $\psi 2$=1, $\psi 3$=1, $V_{max}$=2, $\omega$=0.7, $S_{toch}$=1) | 84.30 | 84.78 |
| 20 | FDR-PSO ($\psi 1$=1, $\psi 2$=0, $\psi 3$=2, $V_{max}$=2, $\omega$=0.7, $S_{toch}$=1) | 84.98 | 85.32 |

| Index | Training algorithm for n-tuple network<br>Total available tuples, R=150 and Tuple-size, n=8<br>Population size for PSO or GA, Q=200 | Average Recognition Rate (%) | Best Recognition Rate in 10 runs (%) |
|---|---|---|---|
| 21 | FDR-PSO ($\psi 1$=2, $\psi 2$=2, $\psi 3$=0, $V_{max}$=2, $\omega$=0.7, $S_{toch}$=1) | 84.54 | 85.48 |
| 22 | SOC-FDR-PSO($\psi 1$=1,$\psi 2$=0,$\psi 3$=1,$V_{max}$=2,$\omega$=0.7, $CL$=2, $OL$=1, $S_{toch}$={0,1}) | 85.09 | 85.62 |
| 23 | SOC-FDR-PSO($\psi 1$=1,$\psi 2$=0,$\psi 3$=2,$V_{max}$=2,$\omega$=0.7, $CL$=2, $OL$=1, $S_{toch}$={0,1}) | 85.01 | 85.48 |
| 24 | SOC-FDR-PSO($\psi 1$=1,$\psi 2$=1,$\psi 3$=2,$V_{max}$=2,$\omega$=0.7, $CL$=2, $OL$=1, $S_{toch}$={0,1}) | 85.03 | 85.75 |
| 25 | SOC-FDR-PSO($\psi 1$=1,$\psi 2$=1,$\psi 3$=2,$V_{max}$=40,$\omega$=0.7, $CL$=40, $OL$=4, $S_{toch}$={0,1}) | 83.08 | 83.97 |
| 26 | Genetic algorithm based approach | 84.17 | 84.96 |

It can be observed from the results that the random variable within the range {0,1} for the stochastic weight factor, in equation (6.1) and (7.3), is more favourable than a constant value of 1. '$S_{toch}$' denotes the stochastic factor in the table. Swarm velocity also plays important role in the experiments. The $V_{max}$ of 2 was observed to be a good value to fine-tune the entire search space with 200 particles and this agreed with the result presented in the previous chapter (Table 6.1). Setting the value of $V_{max}$ to 40 returned poor recognition rates as shown in indexes 4, 8 and 25 of Table 7.2. Several variations of the SOC-PSO and SOC-FDR-PSO were conducted with different values of the criticality limit $CL$. A criticality limit of 2 exhibited high diversity and better performance than a value of 40 or 10. In the experiments in

addition to SOC more diversity was added to the system by prohibiting any duplication of a pixel location in forming a particle.

**Table 7.3 Student's t-test between SOC-FDR-PSO ($X$), index 22 in Table 7.2, and a second algorithm ($Y$)**

| Index from Table 7.2 | Competing Algorithm ($Y$) | t-value | p-value |
|---|---|---|---|
| 1 | Conventional random selection approach | 15.26 | 1.00 |
| 2 | RnP based stochastic approach | 5.92 | 1.00 |
| 3 | PSO ($\psi 1$=1, $\psi 2$ =1,$V_{max}$=2, $\omega$=0.7, $S_{toch}$={0,1}) | 2.18 | 0.97 |
| 5 | SOC-PSO($\psi 1$=1,$\psi 2$ =1, $V_{max}$ =2, $\omega$=0.7, $CL$=2,$OL$=1, $S_{toch}$={0,1}) | 0.32 | 0.62 |
| 26 | Genetic algorithm based approach | 5.79 | 1.00 |

Further diversity was added by reducing the overlapping level, denoted by '*OL*' in Table 7.2, between any two particles. In the table $OL$=1 means that only one dimensional value of a particle is allowed to match with any one dimensional value of any other particle. PSO based approach was also compared with Genetic algorithm based training (index 26 in Table 7.2) and it was found that both GA and PSO based approaches improved recognition rates of the classifier from the conventional counterpart. But PSO showed superior results compared to GA.

The experimental results of the hybrid PSO algorithm were assessed statistically by using the student's t-test (Section 4.4.1). The best-performed algorithm (SOC-FDR-PSO) was compared with a second algorithm from Table 7.2 and results were tabulated in Table 7.3. The null hypothesis for the test was "average recognition rate by the SOC-FDR-PSO ($X$) is higher than any second algorithm ($Y$)". For 10 trials of each algorithm the degrees of freedom was 18. Tabulated t-values for the confidence level 95%, 99% and 99.9% and 18 degrees of freedom were 2.10, 2.88, and 3.92. The t-values were calculated from the experimental results and these are presented in Table 7.3. Results show that the increases in recognition rates by SOC-FDR-PSO (index 22 in Table 7.3) over conventional random selection (index 1), RnP based stochastic approach (index 2) and the GA based method (index 26) are statistically "very highly significant" because the observed t-values for all of these cases were greater than 3.92. A very low t-value of 0.32 between the SOC-FDR-PSO and SOC-PSO (index 5) demonstrates that statistically there was no noticeable difference between the recognition rates by these two algorithms. The t-value between the SOC-FDR-PSO and PSO (index 3) is greater than 2.10 and it implies that due the added diversity the SOC-FDR-PSO performs better than the original PSO and the superior results of the hybrid method were "significant" at 95% confidence level. The p-value in Table 7.3 indicates the probability of observing the result by chance given that the null hypothesis is true. Small values of probabilities cast doubt on the validity of the null hypothesis.

Figure 7.7 displays the side-by-side box plots of the results found in the experiments. Each method in Table 7.2 was run for 10 times and then the average was taken. Recognition rates for all ten runs of an algorithm when grouped together visually, it creates a box as shown in Figure 7.7. Thus each box in the figure was constructed with the recognition rates of ten trials. The box plot conveys location and variation information in data sets, particularly for detecting and illustrating location

and variation changes between different data groups of algorithms. The notches in the Figure 7.7 are drawn about the median so that notches that don't overlap represent significant differences between medians (with 95% confidence). The median of recognition rates for SOC-FDR-PSO was above 85%, for PSO was just below 85%, for RnP was just below 84%, for GA was just above 84% and for randomly selected approach was near 81%. Clearly the SOC-FDR-PSO exhibited a significantly higher median than any other algorithm. Box plots also show if there are unusual observations (outliers) in the dataset. Outliers are individually identified with a plus symbol in Figure 7.7. Two unusual observations were plotted: one for the random selection and the other one for the PSO.

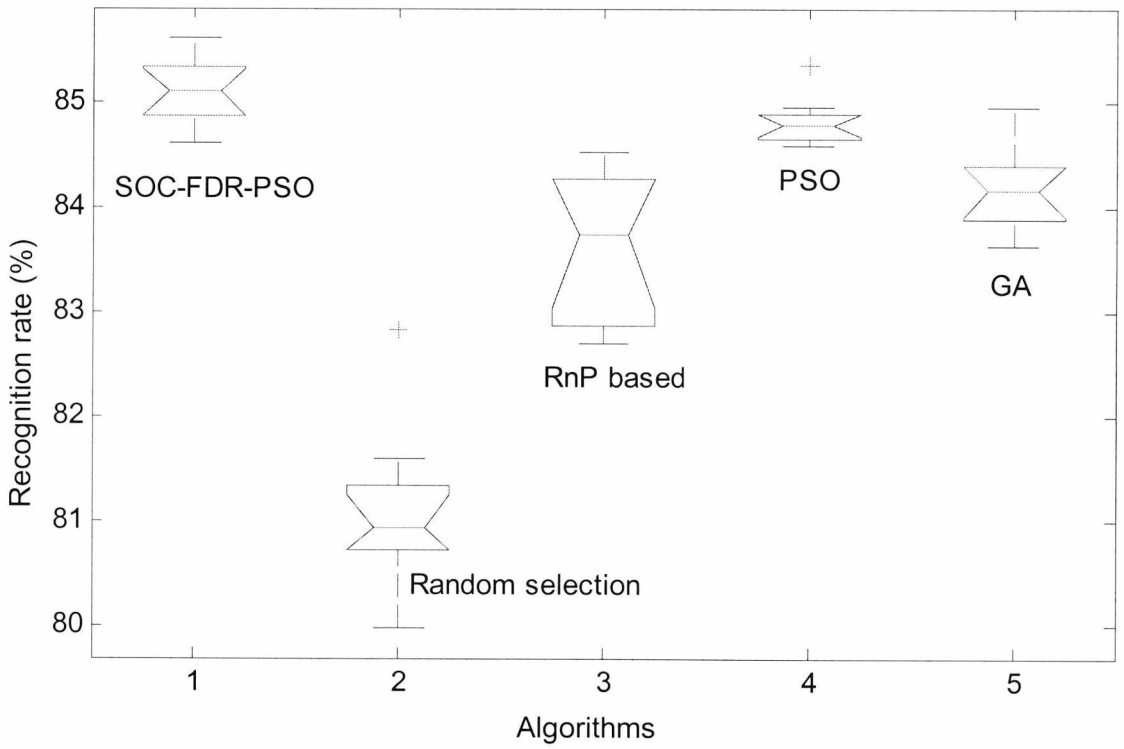



**Figure 7.7 Box plot of several algorithms**

**Table 7.4 Dispersion for different Criticality set-up**

| Criticality limit, *CL* | Dispersion count (Avg. of 30 cycles) |
|---|---|
| 1 | ∞ |
| 2 | 748 |
| 3 | 111 |
| 4 | 28 |
| 5 | 8 |
| 6 | 2 |
| 7 | 1 |
| >8 | 0 |

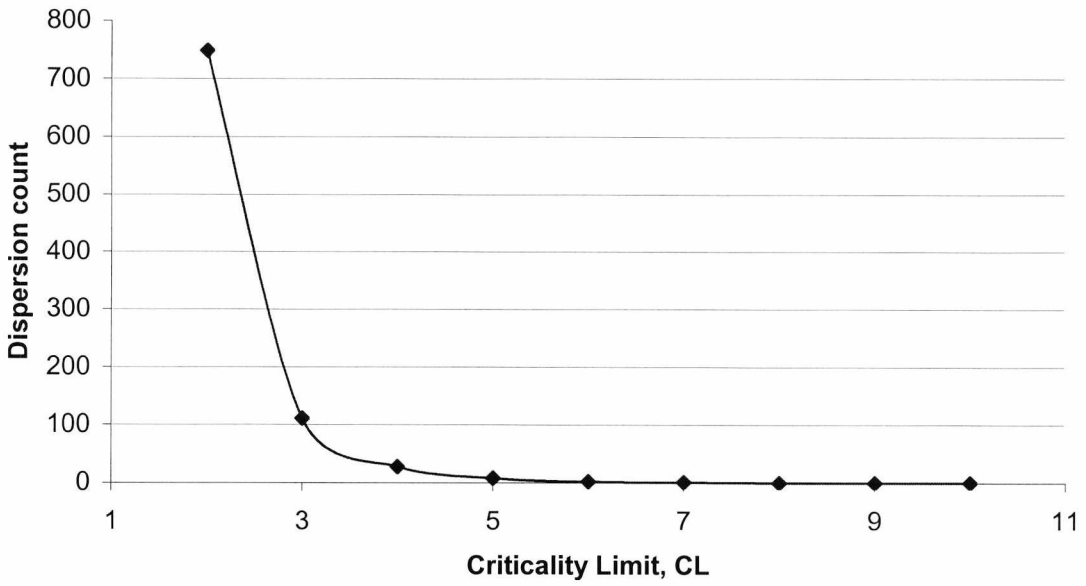



**Figure 7.8 Dispersion of particles decreases with criticality**

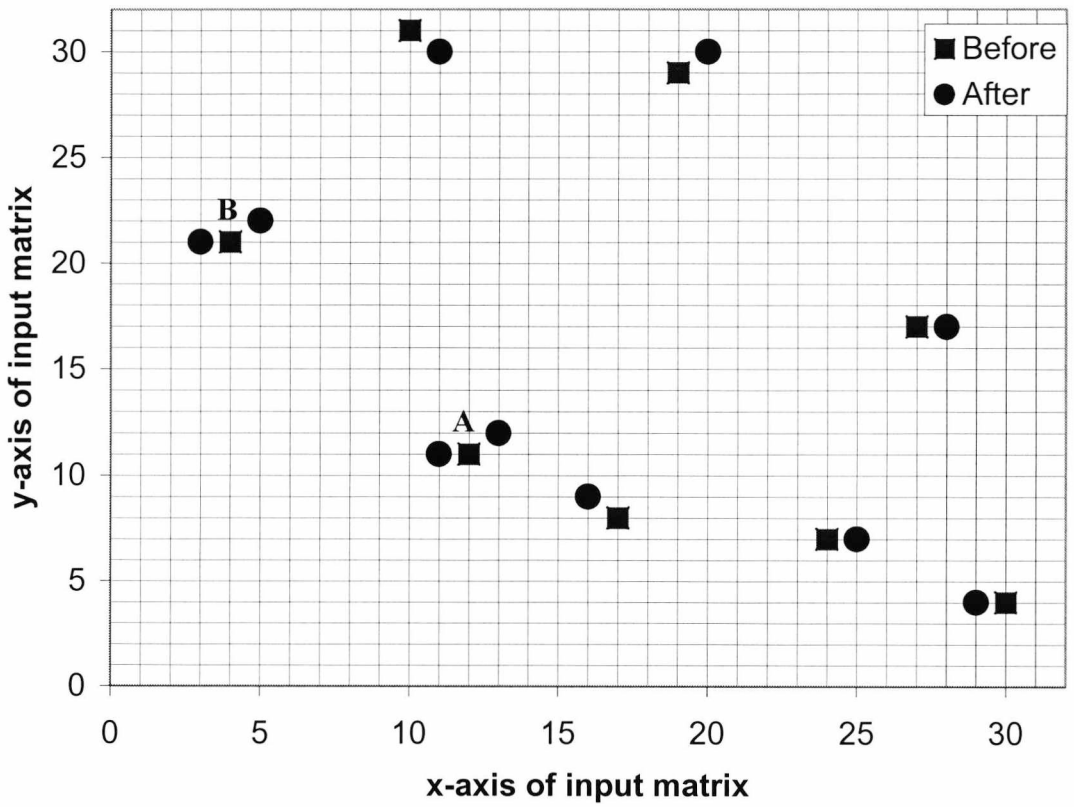



**Figure 7.9 Co-ordinates of $X_{i,d}$ (Table 7.5) before and after dispersion**

Table 7.4 shows dispersion count or number of times particles dispersed for different values of criticality limit. Dispersion count in the table was calculated by finding the average numbers of dispersion in 30 cycles or iterations. Results show that when the criticality limit, *CL*, was equal to or greater than 8 there was no dispersion by any particle. This is because there was no situation where a particle's criticality could cross the limit. Dispersion count was found to be high for a small value of a criticality limit. It showed highest value for a criticality limit of 2 and then the value was gradually dropped to 1 when the criticality limit was 7. Relationship between the *CL* and dispersions count is presented in Figure 7.8. It was found (Table 7.2) that a small value in *CL* (2) was favourable for higher recognition rates. It was

due to the fact that more dispersion adds diversity in the system, in the other words helps exploring solutions to new locations. But this benefit was achieved by the system with the expense of spending more time in searching due to dispersion.

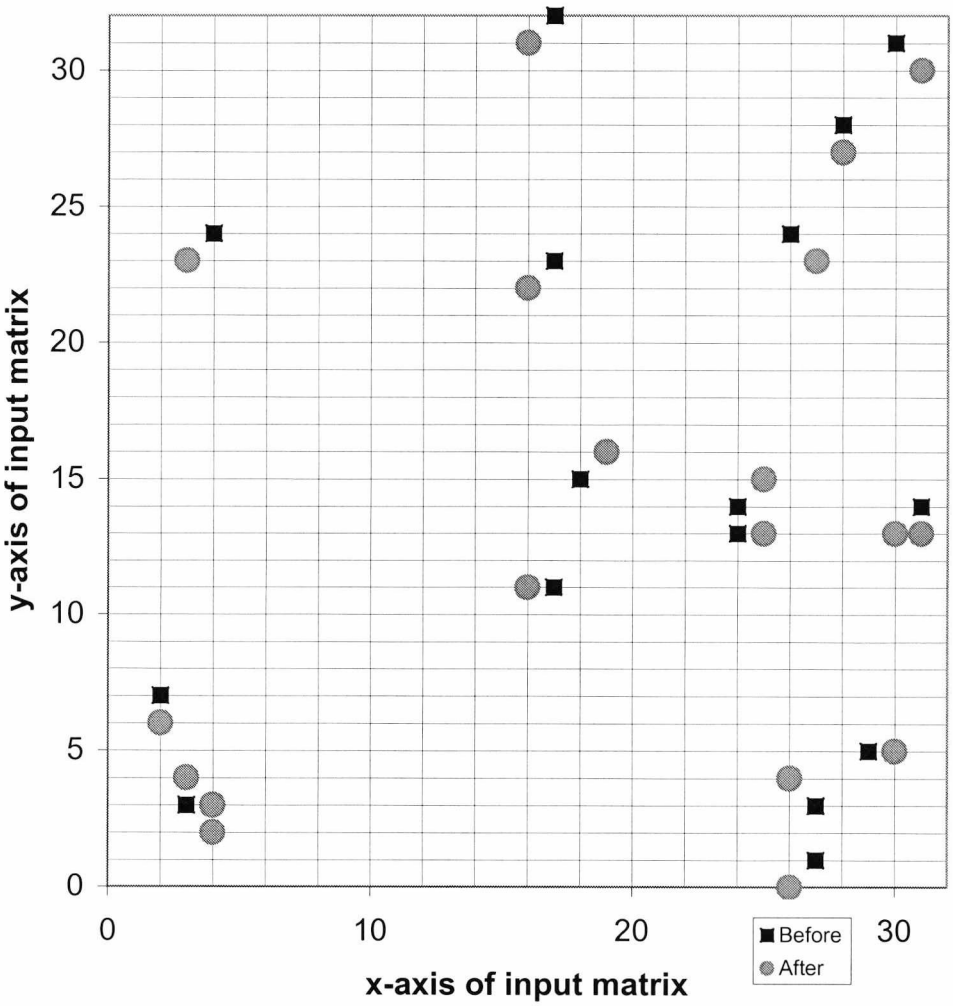



**Figure 7.10 Dispersion in a typical SOC-PSO cycle for *CL*=4**

From Table 7.4 it can be noted that when *CL* was 1, particles dispersed for infinite times or forever. An infinite loop made the system non-convergent and hence

was not acceptable. To avoid a situation where the system might fall into an infinite loop, equation (7.1) was formulated. If equation (7.1) is applied to the system the minimum value of $CL$ can be calculated. In the experiment there were 200 particles ($Q$), each with a dimensionality 8 ($n$). Area of each image was 32 ($W$) by 32 ($H$). Once these values are put in equation (7.1), $CL_{min}$ has come out to be 2. Thus once the equation (7.1) is applied 1 would be automatically rejected as a valid value for $CL$ and this would clearly prevent the system to fall into an infinite loop.

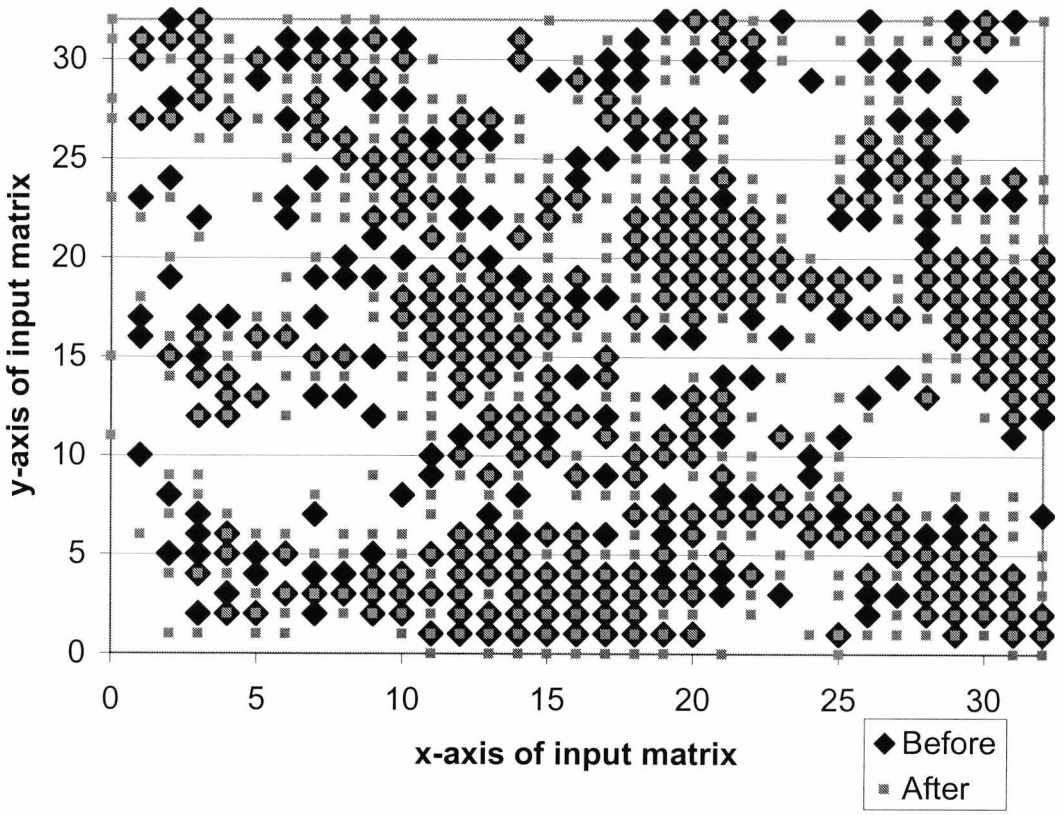



**Figure 7.11 Dispersion in a typical SOC-PSO cycle for $CL$=2**

**Table 7.5 Dispersion in a typical SOC-PSO cycle for *CL*=5**

| Particle position, $X_{i,d}$ | Direction of dispersion |
|---|---|
| 107 | 0 |
| 536 | 2 |
| 373 | 0 |
| 107 | 3 |
| 847 | 7 |
| 289 | 6 |
| 579 | 0 |
| 373 | 3 |
| 956 | 3 |
| 761 | 7 |
| 289 | 6 |

Table 7.5 holds the data of dispersion of particles in a SOC extended PS optimised system. Results in the table were taken from a typical search cycle of a SOC-PSO simulation in training n-tuple classifier for *CL*=5. The first column in the table shows positions of the particles where dispersion occurred and the second column shows the direction of dispersion or direction of jump around $X_{i,d}$ as defined by Figure 7.4. Figure 7.9 shows the x-y co-ordinates of $X_{i,d}$ from Table 7.5 in a 32 by 32 image area. Small squares and circles in the figure depict the positions of the particles before and after dispersion respectively. As dispersion is realized in the nearest neighbourhood area, so a circle in the close proximity of a square would most likely represent the position after dispersion. Position "*A*" in Figure 7.9 corresponds to a value of 373 of $X_{i,d}$ in Table 7.5. It can be noted from the table that there are two

occasions where the value of $X_{i,d}$ was 373, but for both the cases the direction of jump were different and this fact is portrayed by the two circles next to the position *A* in Figure 7.9. A similar situation was observed next to the position B in the figure. Figure 7.10 and Figure 7.11 illustrate the dispersion phenomena for lower values of *CL*. Dispersion was most observed when the value of *CL* was 2. A low value of *CL* forces the system to reach to the criticality point too often and therefore causes more dispersion. Dispersion for *CL*=4 was not as high as for *CL*=2, but it was more than the dispersion for *CL*=5 presented in Figure 7.9 and this acknowledges the fact presented in Figure 7.8 and Table 7.4.

## 7.6 Summary

This chapter presented different variations of hybrid PSO algorithms in training n-tuple network. Original PSO was extended by the hybridisation of the PSO separately with the Self Organised Criticality and the FDR. The FDR algorithm was implemented by incorporating a third particle in the neighbourhood of the current particle and it changes the velocity equation of the original PSO. A novel hybridisation was described in this chapter by combining the SOC and FDR with the PSO to create an algorithm called SOC-FDR-PSO. Results revealed that the original PSO was refined and performed better after hybridisation. This chapter elaborately described different hybridisation techniques and how these approaches were applied to optimise n-tuple networks for recognising binary handwritten characters from the NIST database. A version of the SOC-FDR-PSO performed better than any other approach. The differences in recognition rates by different approaches were assessed by statistical tests. Results for hybrid PSO was found to be statistically significant when compared with the original PSO. It was important to note that the performance of any hybrid approach was very much dependent on different parameter values of the algorithm. Values of the criticality limit played important role in exploration of

solutions for a SOC optimised network. A low value in $CL$ was preferred to assist exploration for new solutions. But the value of $CL$ shouldn't be less than the $CL_{min}$ (7.1), otherwise the training algorithm would fall into an infinite loop making it a non –convergent system. It was clear that the hybrid PSO added some diversity in the system and it helped to explore new locations to find better maps for the n-tuples.

# Chapter 8

# Conclusion

## 8.1 Summary of the thesis

This thesis investigated the application of an efficient optimisation method known as Particle Swarm Optimiser to the field of pattern recognition. Optimisation was realised for the connectivity pattern of the n-tuple network which was proposed by [Bledsoe and Browning, 1959] and described in Chapter 3 of this thesis. Motivation for optimisation was explained in this chapter too. Conventional n-tuple system has been implemented in hardware before by [Aleksander *et al.*, 1984; Azhar and Dimond, 2003]. Optimised n-tuple networks presented in this thesis require less memory than a conventional network for a given performance demand. This is because an optimised network requires less number of tuples than a conventional network requires for a fixed performance level. So the optimised network will facilitate the hardware implementation of the classifier due to its less memory requirements. The goal of the optimisation was aimed at improving the recognition performance of the n-tuple network to classify binary handwritten digits from the NIST database. Important findings of this research are given below:

A novel implementation of the reward and punishment (RnP) based objective function or fitness function was presented in Chapter 5. The equation (5.1) and parameters for the function were modelled which helped to measure the goodness of a solution by a stochastic search. Implementation of the point scheme (5.4) was unique and was explained in Chapter 5. Understanding of the threshold function used to select solutions near the best-performed solution was depicted in this chapter. The exponential decay of the threshold was controlled by careful consideration of the value of the time constant in the threshold equation. The objective function described in Chapter 5 was used to evaluate performance of solutions with all versions of PSO and Genetic Algorithm described in Chapter 6 and Chapter 7.

A new search strategy was developed and was described in Chapter 5. The uniqueness of the strategy was to search for different target number of tuples for different classes. The number was proportionate to the error rates. The search algorithm named as RnP based search spent more time in finding tuples for a difficult pattern class than an easily recognisable pattern. Experiments were conducted to find an optimum set of n-tuples using the RnP based search to recognise handwritten characters from the NIST database. The controlling parameters for the experiments were listed in this chapter. The rationale behind choosing the specific database was given in Chapter 4. A brief overview of the experimental set-up was provided in this chapter. RnP based stochastic algorithm achieved 2.74% improved recognition rates over the random case proposed by [Bledsoe and Browning, 1959]. Results by the RnP method were statistically very significant as well. The statistical test and plot were explained in Chapter 4.

Chapter 6 reported the novel implementation of the particle swarm optimisation on the n-tuple network. Learning scheme by PSO was explained and the pseudo-code was given. Equations for PSO were included and the parameters controlling the performance of PSO were explained. Implementation of GA has been

explained in the same chapter to facilitate a comparison with PSO. Literature reviews for both PSO and GA were reported in Chapter 2. Both algorithms were applied to optimise connectivity pattern of n-tuple networks. PSO trained network showed better results than the GA trained network. Both GA and PS optimisation performed better than the stochastic approach described in Chapter 5. Statistical analysis revealed that the higher value of overall recognition rates by PSO over GA was very highly significant. Appropriate settings of the control parameters in both PSO and GA were the key point for the success. Low particle velocity in PSO was favourable to explore more areas in search space and resulted better recognition rates. The progressive recognition rates of PS and GA trained network were compared to investigate the convergence characteristics of the search. An equation to penalize the unproductive iterations was developed to speed up the search. It was found that dropping the value of the time constant of the threshold equation (described in Chapter 5) during unproductive iterations substantially increased the speed of the search. Results revealed that it was not only the speed but the performance of the system had to be looked after also and to avoid a premature convergence with a low recognition rate the time constant had to be varied very carefully.

Chapter 7 presents the novel application of the hybrid particle swarm techniques in optimising the n-tuple network. Hybridisation was required due the risk of premature convergence in PSO [Kalyan *et al.*, 2003]. To hybridise PSO with the Self-Organised Criticality approach the later algorithm was modelled and adopted for the n-tuple system. Hybridisation of PSO with another bio-inspired approach called the Fitness-to- Distance Ratio or FDR was realized in [Peram *et al.*, 2003]. Chapter 7 presents the application of this hybrid algorithm for the first time to optimise the n-tuple classifier. This chapter also describes the novel hybridisation of the PSO with both the SOC and FDR and the resulting algorithm was named SOC-FDR-PSO. While using the SOC based hybrid approach explanation was given for choosing the

right value of the criticality limit in experiments. Pseudo-code for all hybrid approaches was provided in this chapter. Experiments were performed to compare the performance of all hybrid approaches with varying parameters. A version of the SOC-FDR-PSO outperformed all algorithms in optimising n-tuples. Superior results of the SOC-FDR-PSO over PSO, GA and RnP based stochastic search were statistically significant. The dispersion phenomena in a SOC extended PSO approach was illustrated in experiments and more dispersion was found for lower values of criticality limit. It was found in the research that performance of any hybrid approach was very much dependent on the appropriate settings of the parameters in that algorithm.

## 8.2 Future research

The points described below will indicate the lines of research that can be pursued.

- This thesis has been presented the optimisation of the n-tuple network. Various bio-inspired approaches have been used to optimise the connectivity pattern of the n-tuple system. The same methodologies could be applied for other memory-based network as those networks are connected to the input image in a similar fashion. One example is the optimisation of connectivity pattern for the GCN [Howells *et al.*, 1995] network by using Genetic Algorithm presented in [Farhan-Khola and Howells, 2003]. Particle swarm based optimisation could be equally applied to that network. So applications of all the proposed algorithms in this thesis for optimising connectivity patterns of other memory based networks could be a fruitful research in future.

- It has been shown that the performance of the n-tuple network depends on the size of the n-tuple (Section 4.3). Hoque's [Hoque, 2001] work confirmed Ullmann's [Ullmann, 1969] explanation about the relationship

between the recognition performance and the value of $n$. For all the experiments in this thesis tuple size had a fixed value of 8. But in future effects of different tuple sizes could be investigated. For this, the optimisation method could be designed to seek a set of optimum tuples with different tuple-sizes. In this case PSO and its hybrid versions would search for not only the connectivity pattern but also the size ($n$) of the pattern for different tuples.

- In the experiments presented in this thesis the total available tuples were divided proportionately among classes according to the error rates. This was an initial working hypothesis, which needs further investigation in future. Also the calculated error rates were not normalised. So the effect of normalised error rates will be explored in future as well.

- The proposed optimisation methods were successfully applied for the off-line optical character recognition (OCR) task. The performances of the optimised network for non-OCR applications will be investigated in future.

- Experimental results suggested that the hybridisation of PSO with SOC and FDR algorithms was favourable and resulted in better recognition rates. This inspires in near future to investigate other hybrid techniques like the LifeCyle model described in [Løvbjerg, 2002]. LifeCycle model combines Genetic Algorithms, Particle Swarm and Stochastic Hill-climbing and gives better solution over the individual algorithms themselves. The pseudo-code of the algorithm can be found in [Løvbjerg, 2002], where the individuals (for n-tuple it will be connection maps) start as PSO particles, then switch to GA individuals, then to hill-climbers, then back to PSO particles. The switching happens if an individual makes no fitness improvement.

- Equation (6.3) was used to decrease the threshold value (5.2) of fitness allowing new maps to be selected yielding faster convergence. A zero tangent on the tuple maturity curve (Figure 6.15) indicated an unproductive interval. During search an opposite of the zero tangent, an infinite tangent, could also exist. An infinite tangent would tell the system that a large number of tuples were generated without diversity leading to premature convergence. In this situation fitness threshold has to be increased so that the new tuples require a higher fitness to be accepted as successful. One way to increase the value of the threshold (5.2) would be to increase the value of $\tau$ in (5.3). A negative value of *TD* (percentage drop of $\tau$) in (6.3) could increase the value of $\tau$. A negative of *TD* would mean a percentage rise of $\tau$ instead of a percentage drop. In future, experiments will be conducted to investigate if better results can be found by incorporating the increase of threshold for iterations when large numbers of tuples are generated without diversity.

- Tuple search algorithm takes considerable amount of time to be executed. This is because in software the mappings for different classes can be searched only sequentially. Use of dedicated hardware engine would make the search much faster, because with hardware search for different classes could be run in parallel. So in future the optimisation algorithms will be realised in hardware. Due to the advantage of the re-configurable feature of an FPGA, it can be a good choice for hardware implementation [Azhar and Dimond, 2003].

- Different neighbourhood topologies could be realized in PSO as suggested by [Kennedy, 1999; Kennedy and Mendes, 2002]. In a ring topology particles could be arranged in ring, with same number of particles to the right and left of a particle's neighbourhood. In a Von Neumann topology

particles are connected using a grid where each particle is connected to its four neighbour particles. Topologies have affect on propagation of the best particle in the swarm. Slower propagation enables the particles to explore more areas in the search space and thus decreases the chance of premature convergence. Investigation of effects of different topologies in swarm population for optimum selection of n-tuples could be a good research in near future.

# Bibliography

Aarts, E. and Lenstra, J.K. (2003). Local Search in Combinatorial Optimization, Princeton University Press.

Ackley, D. and Littman, M. (1991). Interactions Between Learning and Evolution, in Artificial Life II, SFI Studies in the Sciences of Complexity, vol. X, eds. By C.G. Langton, C. Taylor, J. D. Farmer & S. Rasmussen, Addison Wesley Publishers, pp. 487-509

Affenzeller, M. and Wagner, S. (2004). SASEGASA: A new generic parallel evolutionary algorithm for achieving highest quality results. Journal of Heuristics, Special Issue on New Advances on Parallel Meta-Heuristics for Complex Problems, 10(3), pp. 239-263.

Aleksander, I., and Stonham, T.J. (1979). Guide to Pattern Recognition using random –access Memories, Computers and Digital Techniques, 2, pp. 29-40.

Aleksander, I., Thomas, W. V., and Bowden, P. A. (1984). WISARD: a radical step forward in image recognition, Sensor Review, 4(3), pp. 120-124.

Aleksander, I. (1989). The logic of connectionist systems, in Neural Computing Architectures, I Aleksander (ed), MIT Press, pp. 133-155.

Angeline, P. (1998a). Using Selection to Improve Particle Swarm Optimization. In. International Conference on Evolutionary Computation, Piscataway, New Jersey, USA, IEEE Service Center, pp. 84-89.

Angeline, P. (1998b). Evolutionary Optimization versus Particle Swarm Optimization:. Philosophy and Performance Difference. In Proceedings of the Seventh Annual Conference on Evolutionary Programming, pp. 601-610.

Arıca, N. and Yarman-Vural, F.T. (2001). An Overview of Character Recognition Focused on Off-line Handwriting. IEEE Transactions on Systems, Man, and Cybernetics, 31, pp.216-233.

Austin, J. (1994). A Review of RAM based Neural Networks, Proceedings of the Fourth International Conference on Microelectronics for, IEEE Computer Society Press, pp. 58-66.

Austin, J. (1998). RAM-Based Neural Networks, a Short History. In RAM-Based Neural Networks, Austin, J. (Ed.), York, UK, pp. 3-17.

Azhar, M.A.H.B. and Dimond, K.R. (2003). An FPGA Based Evolutionary Controller for Mobile Robots, The 2003 International Conference on Machine Learning; Models, Technologies and Applications (MLMTA'03), IEEE Computer Press, June 23-26, Monte Carlo Resort, Las Vegas, Nevada, USA.

Azhar, M. A. H. B. and Dimond, K.R. (2004a). A Stochastic Search Algorithm to Optimize an N-tuple Classifier by Selecting Its Inputs, International Conference on Image Analysis and Recognition, Porto, Portugal, Springer-Verlag, September 29 - October 1.

Azhar, M.A.H.B. and Dimond, K.R. (2004b). Using Particle Swarm with Self-Organized Criticality to Optimize Mapping in N-tuple Networks, Best paper award winner in IEEE SMC UK-RI 3rd Workshop on Intelligent Cybernetic Systems (ICS'04), 7-8 September, University of Ulster at Magee, Londonderry, NI, UK.

Azhar, M.A.H.B. and Dimond, K. R. (2004c). Hybridizing Particle Swarm with Nearest Neighbour Interactions and Self-Organized Criticality to Optimize

Training of N-tuples, The 5th International Conference on Recent Advances in Soft Computing (RASC 2004), Nottingham, United Kingdom, 16-18 December.

Back, T. (1992). Evolutionary Algorithms. ACM SIGBIO Newsletter, pp. 26-31.

Back, T. (1994). Evolutionary Algorithms: Comparisons of Approaches. In R. Paton, editor, Computing with biological Metaphors, Chapman and Hall, pp. 227-243.

Back, T., and Schwefel, H. (1996). Evolutionary Computation: an overview. In Proceedings of third IEEE Conference on the Evolutionary Computation, IEEE press, pp. 20-29.

Back, T., Fogel, D. B., Whitley, D. & Angeline, P. J. (1997a). Mutation.In Back, T., Fogel, D. B. and Michalewicz, Z., editors, Handbook of Evolutionary Computation, Oxford University Press, pp. C3.2:1-C3.2:14.

Back, T., Hammel, U. and Schwefel, H. (1997b). Evolutionary computation: comments on the history and current state. IEEE Transaction on the Evolutionary Computation,1(1),pp.3-17.

Back, T. (1992). Self-Adaptation in Genetic Algorithms. In Proceedings of the First European Conference on Artificial Life, MIT Press, pp. 227-235.

Bak, P. (1996). How nature works: the science of self-organized criticality (Copernicus, New York).

Bishop, J.M. (1989). Anarchic techniques for pattern classification, Ph.D. Thesis, Reading University.

Bishop, J.M., Crowe, A.A., Minchinton, P.R. and Mitchell R.J. (1990). Evolutionary Learning to Optimise Mapping in n-Tuple Networks, IEE Colloquium on Machine Learning, 28 June, Digest 1990/117.

Bledsoe, W. and Browning, I. (1959). Pattern recognition and reading by machine, Proceedings of Eastern Joint Computer Conference, Birmingham, pp.225-232.

Booker, L. B., Fogel, D. B., Whitley, D. & Angeline, P. J. 1997 Recombination.In Back, T., Fogel, D. B. and Michalewicz, Z., editors, Handbook of Evolutionary Computation. 97/1, C3.3, IOP Publishing Ltd. and Oxford University Press.

Boone, JM., Gross, GW, Greco-Hunt, V. (1990a). Neural networks in radiologic diagnosis. I. Introduction and illustration. Invest Radiol, vol. 25, pp. 1012-1016.

Boone, JM, Sigillito, VG and Shaber, GS. (1990b). Neural networks in radiology: An introduction and evaluation in a signal detection task, Medical Physics, vol. 17, pp. 234-241.

Boettcher, S. and Paczuski, M. (1997). Aging in a Model of Self-Organized Criticality, Physical Review Letters, The American Physical Society, vol. 79, no. 5, pp. 889-892.

Bounessah, M. and Atkin, B.P. (1994). Comparison of the relative efficiency of total and cold-extractable stream sediment chemistry in exploration geochemical surveys in a semi-arid climate, Collo area, north-eastern Algeria. Journal of African Earth Sciences, 19(1-2), pp. 51-60.

Bownamker, R.G. & Coghill G.G., (2002). Improved Recognition Capabilities for Goal Seeking Neuron, Electronics Letters, vol. 28, no.3, pp.220-221.

Bramlette, M. (1991). Initialisation, Mutation and Selection Method in Genetic Algorithms. for Function Optimization, In Proceedings of the Fourth International Conference in Genetic Algorithms, pp. 100-107.

CEDAR CDROM-1, State University of New York at Buffalo, UB Commons, 520 Lee Entrance, Suite 202, Amherst,. NY 14228-2567, USA.

CEDAR CDROM-2, State University. of. New York at Buffalo, UB. Commons, 520 Lee Entrance, Suite 202, Amherst, NY 14228-2567, USA.

Chambers, J., William, C., Beat, K., and Paul, T. (1983), Graphical Methods for Data Analysis, Wadsworth.

Charles, E. L. and Ronald, L.R. 1990. Introduction to Algorithms, MIT Press.

Chinneck, J. (2006). Practical Optimization: A Gentle Introduction, http://www.sce.carleton.ca/faculty/chinneck/po.html

Christensen, S.S., Andersen, A.W., Jørgensen, T.M. and Liisberg, C.(1996). Visual guidance of a pig evisceration robot using neural networks, Pattern recognition letters, vol 17(4): pp. 345- 355.

Clerc, M. (1999). The Swarm and the Queen: Towards a Deterministic and Adaptive Particle Swarm Optimization. In Proceedings of the IEEE Congress on Evolutionary Computation, July, vol. 3, pp. 1951-1957.

Coello Coello, C.A. and Lechuga, M. (2002). MOPSO: A Proposal for Multiple Objective Particle Swarm Optimization. In Congress on Evolutionary Computation, Piscataway, New Jersey, USA, IEEE Service Center, vol. 2, pp. 1051-1056.

Collins, T. 1998. Understanding Evolutionary Computing: A hands on approach. In The Proceedings of the International Conference on Evolutionary Computation (ICEC'98). Part of the IEEE World Congress on Computational Intelligence, Anchorage, Alaska. Morgan Kaufmann, CA

Deacon, J. (2006). The Really Easy Statistics Site, Biology Teaching Organisation, University of Edinburgh, http://www.biology.ed.ac.uk/research/groups/jdeacon/ statistics /tress1.html

Dickman, R., Muñoz, M.A., Vespignani, A., and Zapperi, S., (2000). Paths to Self organized criticality, Braz. J. Phys. 30, 27.

Dorigo, M. and Di Caro, G. (1999). The Ant Colony Optimization Meta-Heuristic. New Methods in Optimization, D. Corne, M. Dorigo and F. Glover, Eds., McGraw-Hill.

Di Caro, G., Dorigo, M.(1998). Ant Colonies for Adaptive Routing in Packet-Switched Communications Networks, Proceedings of PPSN V - Fifth International Conference on Parallel Problem Solving from Nature, Amsterdam, Holland, September 27-30, Springer-Verlag, Lecture Notes in Computer Science, vol. 1498

Eberhart, R., Simpson, P. K., and Dobbins, R. W. (1996). Computational Intelligence PC tools, 1st ed., Academic press professional, Boston, MA.

Eberhart, R. and Shi, Y. (1998a). Comparison between Genetic Algorithms and Particle Swarm Optimization, In Proceedings of the Seventh Annual Conference on Evolutionary. Programming, Springer-Verlag, pp. 611-619.

Eberhart, R. and Shi, Y. (1998b). Evolving Artificial Neural Networks. In Proceedings of the the 1998 International Conference on Neural Networks and Brain, pp. PL5- PL13.

Eiben, A.E., Raué, P-E., and Ruttkay, Zs. (1994). Genetic algorithms with multi-parent recombination, In Y. Davidor, H.-P. Schwefel, and R. Männer, editors, Proceedings of the 3rd Conference on Parallel Problem Solving from Nature, number 866 in LNCS, Springer-Verlag, pp. 78-87.

Eiben, A.E., Kemenade, C.H.M.V. and Kok, J.N. (1995). Orgy in the computer: Multi-parent reproduction in genetic algorithms. In F. Moran, A. Moreno, J.J. Merelo, and P. Chacon, editors, Proceedings of the 3rd European Conference on Artificial Life, number 929 in LNAI, Springer-Verlag, pp. 934-945.

Ellis, H. and Sartaj, S. (1984). Fundamental of Computer Algorithms, Computer Science Press.

Engelbrecht, A. (2002). Computational Intelligence: An Introduction. John Wiley and Sons.

ERIM: Environmental Research Institute of Michigan, Document Processing Research Program, P.O. Box 134001, Ann Arbor, Michigan 48113-4001, USA.

Esmin, A. A. A., Aoki, A. R., and Lambert-Torres, G. (2002). Particle swarm optimization for fuzzy membership functions optimization. Proc. of the IEEE Int. Conf. on Systems, Man and Cybernetics, pp. 108-113.

Esquivel, S.C. and Coello Coello, C.A.(2003). On the Use of Particle Swarm Optimization with Multimodal Functions. In Proceedings of the IEEE Transactions on Evolutionary Computation, vol. 2, pp. 1130-1136.

Farhan-Khola, S. and Howells, W.G.J.(2003). Design of a Genetic Feature Selection Algorithm for Neuron Input Mapping in N-Tuple Based Classifiers International Conference on Machine Learning: Models, Technologies and Applications (MLMTA), Las Vegas, Nevada, USA, pp.23-26.

Fairhurst, M.C. and Stonham, T.J. (1976). A Class. System for Alpha-Numeric Characters Based on Learning Network Techniques, Digital Processes, 2, pp. 321-339

Fletcher, R.(2000). Practical Methods of Optimization, second edition. John Wiely & Sons.

Floreano, D. and Mondada, F. (1996). Evolution of Homing Navigation in a Real Mobile Robot. In IEEE Transactions on Systems, Man, and Cybernetics - Part B: Cybernetics, vol. 26, no.3, pp. 396-407.

Floudas, C. and P. Pardalos, P.(1992). Recent Advances in Global Optimization. Princeton. Universality Press.

Fogel, D.B. (1995). Evolutionary Computation: Toward a New Philosophy of Machine Learning Intelligence, IEEE Press, NJ.

Fogel, D.B. (1994). An introduction to simulated evolutionary optimization. IEEE Trans. On Neural Network, 5(1), pp.3-14.

Gabarro, K. (2000). Tabu Search Algorithm, http://www.lsi.upc.es/~mallba/public/library/firstProposal-BA/node11.html

Garcia, L.A.C. (2003). Methods of Global Otimização for Choice of the Standard of Conectividade de Neural Redes without Weight. Mestrado in Computer science Federal university of Pernambuco, UFPE, Brazil.

Garcia, L.A.C. and Souto, M.C.P. (2004). Global Optimisation Methods for Choosing the Connectivity Pattern of N-tuple Classifiers. In: IEEE International Joint Conference on Neural Networks, Budapeste. Proc. of the IEEE International Joint Conference on Neural Networks. pp. 2263-2266.

Glover, F.(1986). Future paths for Integer Programming and Links to Artificial Intelligence, Computers and Operations Research, 5, pp.533-549.

Glover, F. (1989). Tabu Search – Part I. ORSA Journal on Computing, vol.1, no.3, pp. 190-206.

Glover, F. (1990). Tabu Search – Part II. ORSA Journal on Computing, vol.2, no.1, pp. 4-32.

Goldberg, D.E. (1989). Genetic algorithms in search, optimization, and machine learning. Reading, Mass.: Addison-Wesley.

Grother, P.J. (1993). Cross Validation Comparison of NIST OCR Databases, D. P. D'Amato, Editor, SPIE, San Jose, USA, vol. 1906.

Govindan,V.K. and Shivaprasad, A.P. (1990). Character recognition - A review. Pattern Recognition 23(7), pp. 671-683.

Gray, P., Hart, W., Painton, L., Phillips, C., Trahan, M. and Wagner, J. (1997). A Survey of Global Optimization Methods, Sandia National Laboratories, http://www.cs.sandia.gov/opt/survey/

Guyon, I., Haralick, R, Hull, J. and Phillips, I. (1997). Database and benchmarking, In H. Bunke and P. Wand, editors, Handbook of Character Recognition and Document Image Analysis, World Scientic, chapter 30, pp. 779-799.

Hand, D.J. (1986). Recent Advances in Error Rate Estimation, Pattern Recognition Letters, vol.4, pp.335 -346.

Harvey, I. (1993). Evolutionary Robotics and SAGA: the Case for Hill Crawling and Tournament Selection. In Artificial Life III, Langton, C. (Ed.), Publisher: Addison-Wesley, pp. 299-326.

Hebb, D.O. (1949).The Organization of Behavior. John Wiley & Sons. New York.

Hendy, M.D. and Penny, D. (1982). Branch and bound algorithms to determine minimal evolutionary trees. Mathematical Biosciences, 60, pp.133-142.

Hepplewhite, L. and Stonham,T.J. (1997). N-tuple texture recognition and the zero crossing sketch, Electronics Letters, vol. 33, no. 1, pp. 45-46.

Hertz, J., Krogh, A., and Palmer, R.(1991). Introduction to Neural Computation. MA: Addison-Wesley.

Higashi, H. and Iba, H. (2003). Particle Swarm Optimization with Gaussian Mutation. In Proceedings of the IEEE Swarm Intelligence Symposium, pp. 72-79.

Hinton, G. E. (1989). Connectionist Learning Procedures, Artificial Intelligence 40(1-3), pp.185-234.

Holland, J. H. (1975). Adaptation in Natural and Artificial Systems. Ann Arbor, Mich., University of Michigan Press.

Hoque, S. (2001). An approach to high performance image classifier design using a moving window principle. PhD Thesis, Department of Electronics, University of Kent, Cantebrury, Kent, UK.

Horst, R., Pardalos, P. and Thoai, N. (2000). Introduction to Global Optimization, second. edition. Kluwer Academic Publishers.

Howells, W.G., Bisset, D.L. and Fairhurst, M.C. (1995). A New Paradigm for RAM-based Neural Networks Proceedings of Second Weightless Neural Network Workshop, University of Kent, pp.11-16

Ismail, A. and Engelbrecht, A.P.(2000). Global Optimization Algorithms for Training Product Unit Neural Networks. In Proceedings of the IEEE International Joint Conference on Neural Networks, vol.1, pp.132-137.

Jain, A.K., Duin, R.P.W. and Mao, J. (2000). Statistical pattern recognition: A review, IEEE Trans. Pattern Anal. Machine Intell., vol. 22, pp. 4-38.

Janikow, C. and Michalewicz, Z. (1991). An Experimental Comparison of Binary and Floating. Point Representations in Genetic Algorithm. In Proceedings of the Fourth International Conference in Genetic Algorithms, pp. 31-36.

Jørgensen, T.M., Linneberg, C. (1999). Theoretical analysis and improved decision criteria for the n-tuple classifier. IEEE Transactions on Pattern Analysis and Machine Intelligence, vol. 21 , pp.336-347.

Jørgensen, T.M. (1997). Classification of Handwritten Digits Using a RAM Neural Net Architecture, International Journal of Neural Systems, vol. 8, no.1, pp. 17-25.

Jørgensen, T. M., Christensen, S. S. and Liisberg, C. (1995). Crossvalidation and information measures for RAM based neural networks. Proc. of the Weightless Neural Networks Workshop, University of Kent at Canterbury, UK, pp. 87-92.

Jong, K.A.D. (1975). An analysis of the behavior of a class of genetic adaptive systems. PhD thesis, University of Michigan, Ann Arbor.

Jung, D., Krishnamoorthy, M.S., Nagy, G., Shapira, A. (1996). N-Tuple Features for OCR Revisited. IEEE Transactions on Pattern Analysis and Machine Intelligence 18(7), pp. 734-745.

Kalyan, V., Thanmaya, P., Chilukuri, K.M. and Lisa, A. O. (2003). Optimization Using Particle Swarms with Near Neighbor Interactions, GECCO, July 11-16, Chicago, Illnois.

Kennedy, J. (1998). The behavior of particles, In 7th Annual Conference on Evolutionary Programming, San Diego, CA, USA.

Kennedy, J. (1997). The particle swarm: Social adaptation of knowledge. In IEEE International Congress on Evolutionary Computation, IEEE Press.

Kennedy, J., and Eberhart, R.C. (1995). Particle swarm optimization. Proc. of the 1995 IEEE Int. Conf. on Neural Networks (Perth, Australia).

Kennedy, J. and Spears, W.M. (1998). Matching Algorithms to Problems: An Experimental Test of the Particle Swarm and Some Genetic Algorithms on the Multimodal Problem Generator. Proceedings of the IEEE Int'l Conference on Evolutionary Computation.

Kennedy, J. and Eberhart, R. (2001). Swarm Intelligence. Morgan Kaufmann.

Kim, J. and Myung, H. (1997). Evolutionary Programming Techniques for Constrained Optimization Problems IEEE Transaction on Evolutionary Computation, 1(2), pp. 129-140.

Kin, L.W. and George, C. (2005). Data Classification with an improved weightless neural network, Journal of IT in Asia, vol. 1, no. 1, pp.17-34.

Korte, B. and Vygen, J. (2002). Combinatorial Optimization: Theory and Algorithms, Second Edition, SpringerVerlag, Berlin.

Kreyszig, E. (1970). Introductory Mathematical Statistics, John Wiley, section 13.4.

Krink, T. and Løvbjerg, M. (2002). The Life Cycle Model: Combining Particle Swarm Optimisation, Genetic Algorithms and Hill Climbers. In Proceedings of the Parallel Problem Solving from Nature Conference, Lecture Notes in Computer Science, Springer-Verlag, vol. 2439, pp. 621-630.

Krink, T., Thomsen, R. and Rickers, P. (2000) Applying Self-Organised Criticality to Evolutionary Algorithms, Parallel Problem Solving from Nature - PPSN VI (2000), vol. 1, pp. 375-384.

Krink, T. and R. Thomsen, R.(2001). Self-Organized Criticality and Mass Extinction in Evolutionary Algorithms, Proceedings of the Third Congress on Evolutionary Computation (CEC-2001), vol. 2, pp. 1155-1161.

Kürzl, H., 1988. Exploratory data analysis: recent advances for the interpretation of geochemical data. Journal of Geochemical Exploration, 30, pp.309-322.

Løvbjerg, M. (2002). Improving particle swarm optimization by hybridization of stochastic search heuristics and self-organized criticality, Master's thesis Department of Computer Science, University of Aarhus.

Løvbjerg, M. and Krink, T. (2002). Extending particle swarm optimisers with self-organized criticality. Proc. of the IEEE Congress on Evolutionary Computation, Honolulu, Hawaii USA.

Ludermir, T.B., Carvalho, A., Braga, A.P. and Souto, M.C.P.(1999). Weightless Neural Models: a review of current and past work. Published in the Journal Neural Computing Surveys, vol. 2, pp. 41-61.

Maltoni, D., Maio, D., Jain, A.K. and Prabhakar S. (2003). Handbook of Fingerprint Recognition, Springer-Verlag, New York.

Maniezzo, V. and Colorni, A. (1999). The ant system applied to the quadratic assignment problem. Knowledge and Data Engineering, vol.11, no.5, pp. 769-778.

MathWorks (2007). MathWorks Support web pages, The MathWorks, Inc., http://www.mathworks.com/support/

McCauley, J., Thane, B. and Whittaker, A. (1994). Fat Estimation in Beef Ultrasound Images Using Texture and Adaptive Logic Networks. Transactions of ASAE, vol. 37, pp. 997-1002.

Mcculloch, W. S. and Pitts, W. (1943). A Logical Calculus of the Ideas Immanent in Nervous Activity. In: Bulletin of Mathematical Biophysics, vol. 5. pp. 115-133.

Mesot, B. (2004). Self-Organization of Locomotion in Modular Robots: A Case Study, Unpublished Diploma Thesis, http://birg.epfl.ch/page42735.html

Michalewicz, Z. (1996). Genetic Algorithms + Data Structures = Evolution Programs, third. edition. Springer-Verlag, Berlin.

Michalewicz, Z. and Fogel, D. (2000). How to Solve It: Modern Heuristics. Springer-Verlag,. Berlin.

Mitchell, R.J., Minchinton, P.R. (1996). Optimising memory usuage in n-tuple networks, Mathematics & Computers in Simulation, 40, pp: 549-563.

Mori, S., Suen, C.Y. and Yamamoto, K.(1992). Historical review of OCR research and development. Proceedings of the IEEE, July, Special Issue on Optical Character Recognition, 80(7), pp.1029-1058.

Nagy, G.(1988).Chinese character recognition: A twenty-five year retrospective, in Proc. 9th International Conference Pattern Recognition, pp. 163-167.

NICI: Nijmegen Institute for Cognition and Information, Handwriting Recognition Group, Nijmegen University, The Netherlands.

O'Connor, P.J. and Reimann, C. (1993). Multi-element regional geochemical reconnaissance as an aid to target selection in Irish Caledonian terrains. Journal of Geochemical Exploration, 47, pp.63-87.

Papadimitriou, C.H. (1994). Computational complexity, Addison-Wesley Publishing Company.

Pardalos, P., Migdalas, A. and Burkard, R. (2002). Combinatorial and Global Optimization. World Scientific Publishing Company.

Parsopoulos, K.E., Plagianakos, V.P., Magoulas, G.D., Vrahatis, M.N. (2001a). Stretching technique for obtaining global minimizers through Particle Swarm Optimization, Proc. of the PSO Workshop, Indianapolis, USA, pp.22-29.

Parsopoulos, K.E. and Vrahatis, M.N. (2001b). Modification of the Particle Swarm Optimizer for locating all the global minima, Artificial Neural Networks and Genetic Algorithms, V. Kurkova et al. (Eds.), Springer, pp.324-327.

Penny, W.D. and Stonham, T.J. (1990). Learning algorithms for logical neural networks, in IEEE International conference on Systems Engineering, Pittsburgh, PA, USA , pp. 625-628.

Peram, T., Veeramachaneni, K. and Mohan, C.K. (2003). Fitness-Distance-Ratio based Particle Swarm Optimization. In Proceedings of the IEEE Swarm Intelligence Symposium, IEEE Press, pp. 174-181.

Picton, P. (2000). Neural Networks (Grassroots Series), 2nd Edition , Palgrave Publishers Ltd.

Porter, T. M. (1986), The Rise of Statistical Thinking. 1820-1900. Princeton, New Jersey. Princeton University Press.

Ramanan, S., Petersen, R.S., Clarkson, T.G. and Taylor, J.G.(1995). pRAM nets for detection of small targets in sequences of infra-red images, Neural Networks 8 (7-8), pp.1227-1237.

Rana, S. and Whitley, D. (1998). Search, Binary Representations, and Counting Optima. In. Proceeding of a Workshop on Evolutionary Algorithms, Sponsored by the Institute for Mathematics and its Applications.

Ratnaweera, A., Halgamuge, S. and Watson, H. (2003). Particle Swarm Optimization with Self-Adaptive Acceleration Coefficients. In Proceedings of the First International Conference on Fuzzy Systems and Knowledge Discovery, pp.264-268.

Rardin, R.(1998). Optimization in Operations Research, Prentice Hall, New Jersey, USA.

Reynolds, R., Peng, B. and Brewster, J. (2003). Cultural swarms II: Virtual algorithm emergence, In Proceedings of IEEE Congress on Evolutionary Computation 2003 (CEC 2003), Canbella, Australia. pp. 1972-1979.

Rich, E. and Khight, K.(1991). Artificial Intelligence. (2 ed.) New York: McGraw-Hill, Inc.

Rickers, P., Thomsen, R. and Krink, T. (2000). Applying Self-Organized Criticality to the Diffusion Model, Late Breaking Papers at the 2000 Genetic and Evolutionary Computation Conference, Morgan Kaufmann Publishers, vol. 1, pp. 325-330.

Riget, J. and Vesterstrøm, J.S. (2002). A Diversity-Guided Particle Swarm Optimizer -The ARPSO. Technical report, Department of Computer Science, University of Aarhus.

Robinson, J., Sinton, S. and Rahmat-Samii, Y. (2002). Particle Swarm, Genetic Algorithm, and Their Hybrids: Optimization of a Profiled Corrugated Horn Antenna. In Proceedings of the IEEE Antennas and Propagation Society International Symposium and URSI National Radio Science Meeting, vol. 1, pp. 314-317.

Rohwer, R. and Morciniec, M. (1998). The Theoretical and Experimental Status of the n-tuple Classifier. Neural Networks 11(1), pp.1-14.

Rohwer, R. and Morciniec, M.(1996). A Theoretical and Experimental Account of n-tuple Classifier Performance. Neural Computation, vol. 8, pp. 629-642.

Rohwer, R. and Lamb, A.(1993). An exploration of the effect of super large n-tuple on single-layer RAMnets. In N. M. Allison, editor, Proceedings of the Weightless Neural Network Workshop (WNNW93), University of York, UK, pp.33-37.

Rohwer, R. and Cressy, D. (1989). Phoneme classification by boolean networks, Proceedings of the European Conference on Speech Communication and Technology, pp.557-560.

Rosenblatt, F. (1958). The Perceptron. a Probabilistic Model for Information Storage and Organization in the Brain. In: Psychological Review, vol. 65, pp.386-408.

Salhi, S. (2002). Defining tabu list size and aspiration criterion within tabu search methods. Computers and Operations Research, 29(1), pp.67–86.

Salman, A. (1999). Linkage Crossover Operator for Genetic Algorithms, PhD Dissertation. School of Syracuse University, USA.

Shang, Y. and Wah., B.W. (1996). Global optimization for neural network training. IEEE Computer, 29(3), pp. 45-54.

Shi, Y, and Eberhart, R. (1998a). Parameter selection in particle swarm optimization, in Evolutionary Programming VII: Proceedings of 7th Annual Conference on Evolutionary Programming, Springer-Verlag, Lecture Notes in Computer Science, pp. 591-600.

Shi, Y, and Eberhart, R. (1998b). A Modified Particle Swarm Optimizer. In Proceedings of the IEEE Congress on Evolutionary Computation, pp. 69-73.

Shi, Y. and Eberhart, R. (2001). Fuzzy Adaptive Particle Swarm Optimization. In Proceedings Congress on Evolutionary Computation, Seoul, S. Korea.

Shimodaira,H. (1996). A new genetic algorithm using large mutation rates and population-elitist selection (GALME), Proceedings of Eighth IEEE International Conference on Tools with Artificial Intelligence, 16-19 Nov. pp. 25-32.

Simões, E.D.V. (2000). Development of an embedded evolutionary controller to enable collision-free navigation of a population of autonomous mobile robots, PhD Thesis, University of Kent, Canterbury, UK, November.

Settles, M. and Rylander, B. (2002).Neural network learning using particle swarm optimizers. Advances in Information Science and Soft Computing, pp. 224-226. WSEAS Press.

Spall, J. (2003). Introduction to stochastic search and optimization: estimation, simulation, and control, Wiley-Interscience.

Spears, W.M., DeJong, K.A., Back, T., Fogel, D.B., de Garis, H.(1993). An Overview on Evolutionary Computation, Proceedings of European Conference on Machine Learning, Vienna, Austria.

Streiner, D.L.(1997). Speaking Graphically: An Introduction to Some Newer Graphing Techniques. Canadian Journal of Psychiatry, vol. 42, pp.388-394.

Suganthan, P. N. (1999). Particle swarm optimiser with neighbourhood operator. Proc. of the IEEE Congress on Evolutionary Computation, IEEE Service Center, Piscataway, NJ, pp. 1958-1962.

Sutton, R. and Barto, A.(1998). Reinforcement learning: an introduction. Adaptive computation and machine learning. MIT Press, Cambridge, MA.

Syswerda, G. (1989). Uniform crossover in genetic algorithms. In Schaffer, D. (ed.), Proc. of the Third Int. Conf. on Genetic Algorithms. Morgan Kaufmann Pub.

Tambouratzis, G.(2000). Variable Sensitivity in Unsupervised Clustering Tasks with an n-tuple-based Self-Organising Neural Network. International Journal of Neural Systems, vol. 10, no. 2, pp. 107-121.

Tarling, R. and Rohwer, R.(1993). Efficient use of training data in the n-tuple recognition method, IEE Electronics Letters, 29(24), pp. 2093-2094.

Tsou, D. and MacNish, C. (2003). Adaptive Particle Swarm Optimisation for High-Dimensional Highly Convex Search Spaces. In Proceedings of the IEEE Congress on Evolutionary Computation, vol. 2, pp. 783-789.

Tukey, J.W. (1977). Exploratory Data Analysis, Reading, MA: Addison-Wesley.

Turing, A. M. (1937). On Computable Numbers, with an Application to the Entscheidungsproblem. Proc. London Math. Soc. Ser. 2 42, pp. 230-265, Reprinted in The Undecidable (Ed. M. David). Hewlett, NY: Raven Press, 1965.

Tomassini, M. (1995). A Survey of Genetic Algorithms, Annual Reviews of Computational Physics, World Scientific,Vol.III, pp.87-118.

Thomas, C., Leiserson, C. and Rivest, R.(1990). Introduction to Algorithms. McGraw-Hill.

Ullman, J.R. (1969). Experiments with the n-tuple method of pattern recognition, IEEE Transactions on computers, pp. 1135-1137.

Ullman, J.R. and Kidd, P.A. (1969). Recognition experiments with typed numeral from envelopes in the mail. Pattern Recognition, (1), pp.273 -289.

Van den Bergh, F. and Engelbrecht, A.P. (2000). Cooperative Learning in Neural Networks using Particle Swarm Optimizers. South African Computer Journal, vol. 26, pp.84-90.

Van den Bergh, F. (2002) An Analysis of Particle Swarm Optimizers. PhD thesis, Department of Computer Science, University of Pretoria, Pretoria, South Africa.

Van den Bergh, F. and Engelbrecht, A. P. (2001). Effects of swarm size on cooperative particle swarm optimisers. Proc. of the Genetic and Evolutionary Computation Conference, San Francisco, USA

Van Laarhoven, P. and Aarts, E.(1987). Simulated Annealing: Theory and Applications. Kluwer Academic Publishers.

Wang, Y.S., Griffiths, B.J., Wilkie, B.A. (1996). A novel system for coloured object recognition, Computers in Industry, 32(1), pp. 69-77.

Wilkinson, R., Geist, J., Janet, S., Grother, P., Burges, C., Creecy, R., Hammond, B., Hull, J., Larsen, N., Vogl, T., and Wilson, C. (1992). The first census optical character recognition systems conference. Technical Report NISTIR 4912, National Institute of Standards and Technology (NIST), Gaithersburg, USA.

Wilkinson, R. (1992). Handprinted segmented characters database. Technical Report Test Database 1, TST1, National Institute of Standards and Technology, April.

Yao, X. (1997). Global optimisation by evolutionary algorithms, Proc. of the Second Aizu International Symposium on Parallel Algorithm/Architecture Synthesis (pAs-97), Aizu-Wakamatsu, Japan, 17-21 March, IEEE Computer Society Press, pp.282-291.

Yasuda, K., Ide, A. and Iwasaki, N.(2003). Adaptive Particle Swarm Optimization. In Proceedings of the IEEE International Conference on Systems, Man, and Cybernetics, vol. 2, pp. 1554-1559.

Yee, P. and Coghill, G. (2004). Weightless Neural Networks: A Comparison Between the Discriminator and the Deterministic Adaptive RAM Network. Knowledge-Based Intelligent Information and Engineering Systems, 8th International Conference, KES 2004, Wellington, New Zealand, September 20-25, Proceedings, part II, pp. 319-328.

Yusta, I., Velasco, F. and Herrero, J.M., (1998). Anomaly threshold estimation and application to lithogeochemical exploration in Lower Cretaceous Zn-Pb carbonate-hosted deposits, Northern Spain. Applied Geochemistry, 13 (4), pp.421-439.

# Appendix A

# Publications arising from this work

**Journal Paper (Refereed)**

- M. A. H. B. Azhar, F. Deravi and K. R. Dimond. (2008). "Particle Swarm Intelligence to Optimise the Learning of N-tuples", to be appeared in the Journal of Intelligent System with a theme on "Cybernetic systems: Fuzzy, neural and evolutionary computing approaches", ISSN: 0334-1860.

**Award Winning Paper (Refereed)**

- M. A. H. B. Azhar and K. R. Dimond. (2004). "Using Particle Swarm with Self-Organized Criticality to Optimize Mapping in N-tuple Networks" IEEE SMC UK-RI 3rd Workshop on Intelligent Cybernetic Systems (ICS'04), 7-8 September, 2004, University of Ulster at Magee, Londonderry, UK. (Best Paper Award with Financial Prize).

**Conference Papers (Refereed)**

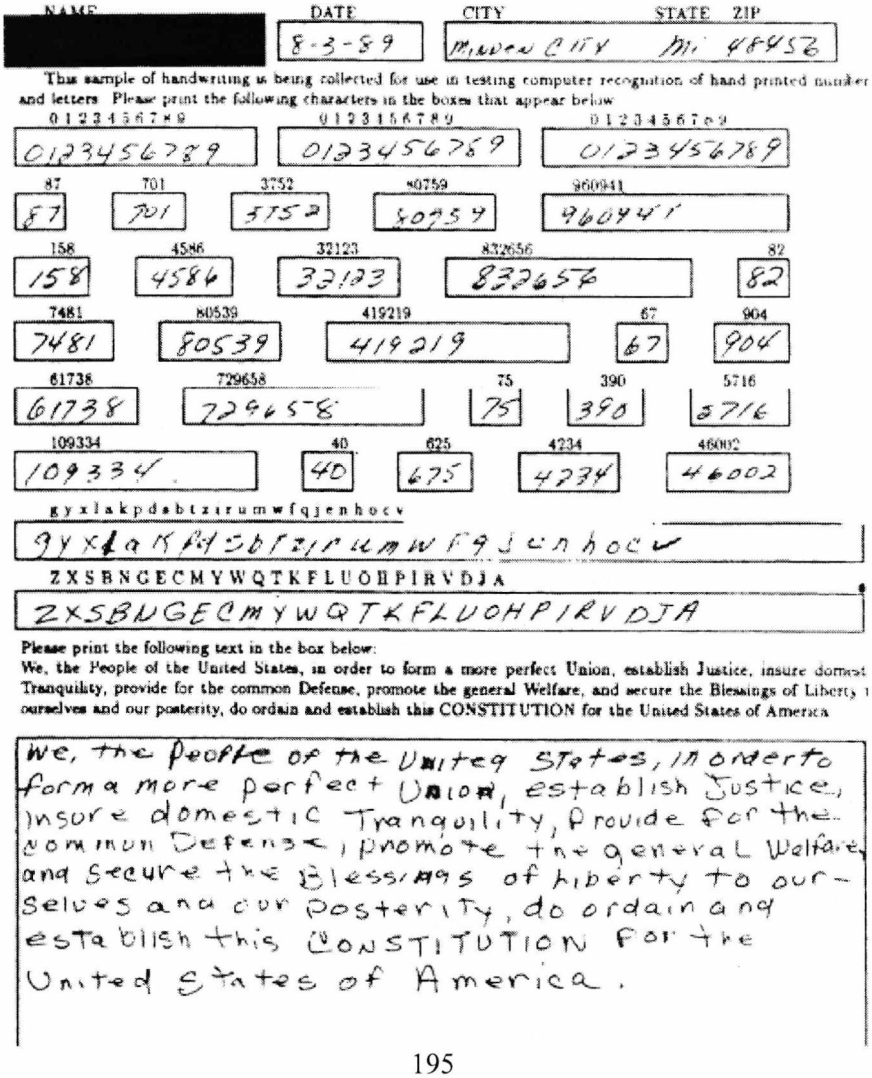
- M. A. H. B. Azhar, F. Deravi and K. R. Dimond. (2008). Criticality Dispersion in Swarms to Optimize N-tuples, Genetic and Evolutionary Computation Conference (GECCO 2008), July 12-16, Atlanta, Georgia, USA.
- M. A. H. B. Azhar, F. Deravi and K. R. Dimond. (2006). Convergence of Particle Swarm and GA to Optimize N-tuples, July 10-12 Proceedings of 6th International Conference on Recent Advances in Soft Computing (RASC2006), pp. 103-108, University of Kent, Canterbury, UK.

- M. A. H. B. Azhar, F.Deravi and K. R. Dimond. (2005). Relative Performances of Swarm Intelligence and Genetic Algorithm to Select Better N-tuples, IEEE SMC UK-RI 4th Conference on Applied Cybernetics, (AC2005). September 7-8, pp.111-116, Birley Lecture Theatre, City University London, United Kingdom.
- M. A. H. B. Azhar and K. R. Dimond. (2004). Hybridizing Particle Swarm with Nearest Neighbour Interactions and Self-Organised Criticality to Optimize Training of N-tuples, The 5th International Conference on Recent Advances in Soft Computing (RASC 2004), Nottingham, 16-18 December, United Kingdom.
- M. A. H. B. Azhar and K. R. Dimond. (2004). A Stochastic Search Algorithm to Optimize an N-tuple Classifier by Selecting Its Inputs, The International Conference on Image Analysis and Recognition (ICIAR 2004), Springer-Verlag, September 29 - October 1, Porto, Portugal.

# Appendix B

# Handwriting Sample Form in NIST

Following is an example HSF image file located in the CD-ROM distributed by NIST. All fields except the first line in such HSF forms were segmented by NIST.


NAME [redacted] DATE 8-3-89 CITY Mineola City STATE MI ZIP 48455

This sample of handwriting is being collected for use in testing computer recognition of hand printed number and letters. Please print the following characters in the boxes that appear below

0 1 2 3 4 5 6 7 8 9

0123456789 | 0123456789 | 0123456789

87 | 701 | 3752 | 80759 | 960941

158 | 4586 | 32123 | 832656 | 82

7481 | 80539 | 419219 | 67 | 904

61738 | 729658 | 75 | 390 | 5716

109334 | 40 | 625 | 4234 | 46002

g y x l a k p d s b t z i r u m w f q j e n h o c v

9y x l a k p d s b t z i r u m w f q j e n h o c v

Z X S B N G E C M Y W Q T K F L U O H P I R V D J A

ZXSBNGECMYWQTKFLUOHPIRVDJA

Please print the following text in the box below:
We, the People of the United States, in order to form a more perfect Union, establish Justice, insure domest Tranquility, provide for the common Defense, promote the general Welfare, and secure the Blessings of Liberty t ourselves and our posterity, do ordain and establish this CONSTITUTION for the United States of America

We, the People of the United States, in order to form a more perfect Union, establish Justice, insure domestic Tranquility, Provide for the common Defense, promote the general Welfare, and secure the Blessings of liberty to ourselves and our Posterity, do ordain and establish this CONSTITUTION for the United States of America.