Stochastic Methods and Genetic Algorithms for Neural Network Learning

Antoniya Georgieva

The thesis is submitted to the **University of Portsmouth** for the degree of
Doctor of Philosophy

Director of Studies: **Dr. Ivan Jordanov**

February 2008

School of Computing
University of Portsmouth
United Kingdom

THESIS CONTAINS

VIDEO CD DVD TAPE CASSETTE

DECLARATION

Whilst registered as a candidate for the above degree, I have not been registered for any other research award. The results and conclusions embodied in this thesis are the work of the named candidate and have not been submitted for any other academic award.

Antoniya Georgieva

ABSTRACT

This thesis presents results from the development, investigation, testing and evaluation of novel meta-heuristic techniques aiming to further improve the *state-of-the-art* of algorithms for local minima free Neural Network supervised learning.

Several approaches for solving Global Optimisation problems that make use of novel meta-heuristic techniques, so-called Low-discrepancy Sequences, and hybrid Evolutionary Algorithms are proposed here, investigated, and critically discussed. Furthermore, the novel methods are tested on a number of multimodal mathematical function optimisation problems, as well as on a variety of Neural Network learning tasks, including real-world benchmark datasets. Comparison of the results from the investigated methods with such from standard Backpropagation, Evolutionary Algorithms, and other stochastic approaches (Simulated Annealing, Tabu Search, etc.) is conducted in order to demonstrate their competitiveness in terms of number of function evaluations, learning speed, and Neural Network generalisation abilities.

Finally, the investigated techniques are applied and tested on real-world problems for the intelligent recognition and classification of cork tiles. An Intelligent Computer Vision system is built. The system includes the following stages: image acquisition; image processing (feature extraction and statistical data processing); Neural Network architecture design; supervised learning utilising the proposed Global Optimisation techniques; and finally, extensive system evaluation.

The presented examples and case studies demonstrate that the proposed techniques can be effectively applied for the optimisation of mathematical, multimodal functions. The investigated methods are successful in local minima free Neural Network learning, and they can be used for solving real-world industrial problems.

DEDICATION

To my mother

ACKNOWLEDGEMENTS

I would like to express my sincerest gratitude to my supervisor and friend Dr. Ivan Jordanov, firstly, for dragging me into this adventure called PhD; secondly, for the constant personal support and encouragement; and finally, for his competence, guidance, motivation, and endless corrections of my work. I would also like to thank his wonderful wife Rossi for the support and pleasant time during our trips to different conferences.

Thank you to my second supervisor Dr. Alexander Gegov from the School of Computing, for being there for me with advice, motivation, and understanding. To his wife, Juliet, for all the 'women talks', parties, dinners, and ice-creams.

I would like to thank the School of Computing, University of Portsmouth, for the sponsorship during three wonderful years. Thank you to all my colleagues in Room 1.04, especially, to Muohammd Al-Kasassbeh for all the moments we have shared – good and bad.

My love and gratitude go to my boyfriend Christos for supporting me, taking care of me, and keeping the noise in the house down to zero while I have been working on this research.

I would like to thank my great friend and colleague Nikola Chalashkanov from the University of Leicester for the numerous research discussions and thoughtful advices.

Finally, my gratitude goes to my parents for their care and encouragement. I would like to dedicate this thesis to my mother for her support during my long studies back in Bulgaria, listening to my long telephone speeches from UK, and being a fighter and wonderful example in live.

SUMMARY OF NOTATIONS

Abbreviations:	*Common notations:*
AI – Artificial Intelligence	l – number of features
BP – Back-propagation	n – NN dimensionality
EA – Evolutionary Algorithms	W – the n dimensional vector of NN weights
EP – Evolutionary Programming	T – target dimension
ES – Evolutionary Strategies	H – number of hidden layers
GA – Genetic Algorithms	P – number of training samples
GO – Global Optimisation	$\boldsymbol{x}$ – sample vector
LDS – Low-Discrepancy Sequences	$\boldsymbol{o}$ – output vector
LMP – Local Minima Problem	$\boldsymbol{t}$ – target vector
NN – Neural Networks	

TABLE OF CONTENTS

1 Introduction

This chapter considers the background and history of the current state of the Artificial Intelligence field. The motivation and the objectives of this research are outlined. Finally, an overview of the rest of the thesis is presented.

1.1 Background and History

It is believed that the human brain consists of 10^{11} neurons (a specialised cell, transmitting nerve impulses) and each of them is believed to have about 10^4 connections with other neurons (Hagan *et al.*, 1996). The total number is about 10^{15} connections. It is accepted that the function of the brain depends mainly on the strength of these connections (synapses). It could be weaker or stronger – changing with time (process of remembering or forgetting). The parallelism of the biological neural system provides an extremely high speed of execution of different tasks and processes. Inspired by one of the greatest human riddles (although the study of the brain is thousands of years old), a numerical method with wide range of applications has been introduces and developed during the last sixty years – artificial Neural Network (NN). It is considered (Engelbrecht, 2002) to be a branch from the field of Artificial Intelligence (AI).

It is important to be noted, that only the idea of *how the human brain works* is adopted and primitively simulated. Both NN and the brain are built up of simple computational particles (blocks), which are highly interconnected. In both cases the connections between the blocks determine the network function. However, NNs are far from competing with the human brain as some people may still believe (Bishop, 1995; Burke and Ignizio, 1997).

The first step towards NN development was made in 1943 by Warren McCulloch and Walter Pitts (McCulloch and Pitts, 1943) who proposed the artificial neuron. In their famous paper, they united the studies of neurophysiology and mathematical logic. The authors showed that a network with sufficient number of artificial neurons and synaptic connections set properly could calculate any computable function. It is generally agreed (Haykin, 1999) that with this significant result the fields of Neural Networks and of Artificial Intelligence were born. The concept was further developed in 1949 by Hebb (Hebb, 1949) in his book *The organization of behavior*, where he

proposed the idea of adaptive learning of the NNs. In the 1950s Alan Turing established the first definition of AI and the term AI was coined in 1956 at the Dartmouth conference organised by John MacCarthy (Engelbrecht, 2002). One significant result of these years was the Kalmogorov's *superposition theorem* which stated that every continuous function of several variables (for a closed and bounded domain) can be represented as a superposition of a small number of functions of one variable. This was used later on to prove that every continuous mapping can be represented exactly by a three layered NN with a finite number of neurons in the hidden layers (Bishop, 1995). Back in 1960s, following the Rosenblatt's work on the *perceptron* (Rosenblatt, 1962) as well as Widrow and co-workers' (Widrow and Hoff, 1960) investigation of ADAptive LINear Element (*ADALINE*), researchers began exaggerating and generalising the abilities of NNs (Bishop, 1995). This was easily picked up by the media and other culture issues (literature, cinema, etc.). Serious researchers and engineers are usually discouraged by such hyperbolas (Burke and Ignizio, 1997). On the other hand, the promises and expectations were not fulfilled. Thus, combined with the lack of computing efficiency in those early years of the NNs' *life*, caused its slow development up to 1980s. In 1986, the popularisation of the *back-propagation* training method proposed by Rumelhart, Hinton and Williams has changed matters considerably (Rumelhart *et al.*, 1986). They proposed a very effective training method for NNs and during the last twenty years, the field has been growing quickly. NNs have proven to be very effective for the solution of a number of benchmark and real-world problems. They are very successful in function approximations. Moreover, a network with two layers with monotonically increasing transfer functions has been proved to be able to approximate any continuous functional mapping with arbitrary accuracy, provided that the hidden layer has enough neurons (Hornik *et al.*, 1989; Bishop, 1995; Engelbrecht, 2002). NNs have been applied to *classification* (the task is to acknowledge the type of an input vector); *pattern matching* (the task is to produce a pattern best associated with given input); *diagnosis of diseases, image processing, speech recognition*; *robot control* (for a given input, appropriate action is suggested); *data mining (knowledge discovery)*; *optimisation*, etc. (Engelbrecht, 2002). What these applications have in common is that they could be combined in the concept of *Pattern Recognition.*

1.2 Motivation

Supervised NN learning is a process, in which *training* input patterns and known *target* values are presented to the NN and it *learns* to respond as desired (Bishop, 1995).

The *learning* is mathematically defined as an optimisation problem, i.e., an error function representing differences between desired and actual output, is being minimised (Bishop, 1995).

The most popular *supervised learning* techniques are gradient based (Backpropagation) and they suffer from the so-called Local Minima Problem (LMP). That is, the optimisation method is trapped in a sub-optimal solution which leads to poor NN performance (Smagt, 1994; Bishop, 1995; Burke and Ignizio, 1997; Yao, 1999; Plaginakos *et al.*, 2001; Engelbercht, 2002; etc.). This motivated the employment of Global Optimisation (GO) methods for the supervised NN learning. GO techniques are stochastic, heuristic, and evolutionary approaches and have demonstrated promising performance (Battiti and Tecchiolli, 1995; Smagt, 1994; Yao, 1999; Sexton *et al.*, 1998; Plaginakos *et al.*, 2001; Rocha *et al.*, 2003; Alba and Chicano, 2004; Ludemir *et al.*, 2006; etc.). Unlike gradient based methods, GO techniques can escape from the region of attraction of a local minimum. This is usually accomplished by using non-gradient information, i.e. stochastic or heuristic knowledge.

The research presented here is motivated by the LMP and is concentrated on the development and further improvement of GO methods for supervised NN learning.

1.3 Objectives

The purpose of this research is the investigation, development, testing and evaluation of novel meta-heuristic techniques aiming to further improve the *state-of-the-art* of the algorithms for local minima free Neural Network supervised learning.

In this research we proposed, investigated, and critically discussed several approaches for solving Global Optimisation problems that make use of so-called Low-discrepancy Sequences, novel meta-heuristic techniques, and hybrid Evolutionary Algorithms. The investigated methods are tested on a number of function optimisation

problems, as well as a variety of NN learning tasks (including real-world benchmark datasets). The results from the investigated methods were compared with such from standard Backpropagation, Evolutionary Algorithms, and other stochastic approaches (Simulated Annealing, Tabu Search, etc.) in order to demonstrate their competitiveness in terms of number of function evaluations, time for learning, and NN generalisation abilities.

Finally, the investigated techniques were applied and tested on real-world problems for automation of an industrial assembly line. An Intelligent Computer Vision System is built that is capable of:

- acquiring images of products;
- processing them by feature extraction and data processing techniques;
- feeding the data to the NN classifier that is trained with the proposed and researched here methods;
- evaluating the system performance.

1.4 Layout of the Thesis

This thesis is organised in seven chapters. The first chapter gives a brief history and introduction to the field of Artificial Intelligence, presenting the motivation for this study and the research objectives. The second chapter contains a deeper overview of some of the principles and techniques that will be used into the investigation of novel NN supervised learning algorithms. The problem of Pattern Recognition is stated and an overview of available AI techniques and classifiers is presented. Chapter 2 also identifies the major characteristics and properties of Feed-forward NN; reviews several data processing techniques; provides broad critical review and categorisation of current Global Optimisation (GO) methods; and finally, introduces the Low-discrepancy Sequences of points.

In Chapter 3, a novel GO technique, based on Low-discrepancy Sequences is proposed, investigated, and discussed. Its properties are researched in detail. It is initially tested on a moderate set of functions and NN learning problems. Based on the demonstrated promising performance, the technique is subsequently combined with a local search, and its performance is investigated in detail via numerous tests.

In order to handle higher dimensional problems, the proposed in Chapter 3 technique is combined with Genetic Algorithms in Chapter 4 to form a novel evolutionary approach. It is investigated thoroughly, numerous tests are performed and the results are compared with *state-of-the-art* GO methods to show its competitiveness.

An introduction to the case study of this research – recognition and automated classification of cork tiles by an Intelligent Computer Vision System – is presented in Chapter 5. The basic concepts of texture feature description and extraction are introduced. In addition, detailed discussion of the properties of cork is given as well as motivation of the case study. Chapter 5 also presents results from studies conducted by other authors for the recognition of cork products (cork stoppers and cork planks).

Chapter 6 describes the Intelligent Computer Vision System that is considered in the case study of this research. It includes acquisition, processing, and classification of commercially available cork tiles. The experiment is conducted in several stages and at each one the results are presented and critically discussed.

Chapter 7 presents a brief conclusion, outlines the author's contributions and gives directions to future work.

In addition, Appendix A presents details and illustrations of the multimodal mathematical functions used for testing in Chapter 3 and Chapter 4. Summary and screen-shot illustrations of the source code, developed for this research, are shown in Appendix B. The full source code, as well as the cork tiles images from the case study are given in the attached CD.

2 Review of Related Work

This chapter gives a deeper overview of some of the principles and techniques that will be covered and used throughout the thesis. In particular, the Pattern Recognition problem is introduced and the issues associated with it. One demonstrative example is discussed and a brief overview of the general types of classifiers available today is given. This chapter also identifies the major characteristics and properties of feed-forward NNs, gives a review of several data processing techniques, provides broad critical review and categorisation of current Global Optimisation methods, and finally, introduces the Low-discrepancy Sequences.

2.1 The Pattern Recognition Problem

2.1.1 Statement of the Problem

Pattern Recognition deals with the classification of different *objects* into a number of *classes* (categories). In the following work only the *supervised* Pattern Recognition problem would be considered, i.e. when input patterns and their corresponding K classes are known *a priori* while the task of the classifier is to learn and extract the rules that assign each pattern to a specific class. In the *unsupervised* problem, only a set of pattern characteristics is given, but no target values, and the aim is to unravel the underlying *similarities*, and *cluster* (group) the similar vectors together. We will refer to the supervised Pattern Recognition problem as *classification*.

Classification could be considered as a *class prediction* task, where from the available data measures (*features*), conclusion may be drawn about the probability of an object belonging to a certain class. These objects can be images, industrial products, persons, etc., or in other words, they can be members of any logical categorisation. Pattern Recognition provides a rule for assigning each point of feature space to one of K discrete classes or categories. However, there are other Pattern Recognition tasks, which are referred to as *regression* problems, in which the outputs (targets) represent the values of continuous variables. On the other hand, both regression and classification problems can be seen as particular cases of *function approximation*. In the case of regression problems, it is the regression function which

we wish to approximate. For classification problems, the functions which we seek to approximate, are the probabilities of membership of the different classes, expressed as functions of the input variables (Bishop, 1995).

Let us denote by $\boldsymbol{x}_i = [x_1, \ldots, x_l]^T$ each pattern and by $\boldsymbol{t}_i = [t_1, \ldots, t_T]^T$ its corresponding *target* ($i = 1, \ldots, P$), where P is the number of training samples available. Here l is the number of *features* for each sample and, if we consider the samples as rows of a data matrix $D = (P \times l)$, the l columns would be called *feature vectors* (each with P dimensions). Features and feature vectors are treated as *random* variables and vectors, as the measurements for different samples have random variation. This is due to the measurement noise and the distinct characteristics of each pattern (Theodoridis and Koutroumbas, 2006). Mathematically, the supervised classification problem could be stated as follows: Given P patterns with their class assignments, find a functional mapping $F : \mathbb{R}^l \rightarrow \mathbb{R}^T$, such that for as many as possible from the P training inputs holds $F(\boldsymbol{x}_i) = \boldsymbol{t}_i$, $i = 1, \ldots, P$. After this process of *learning*, the mapping F is applied to new unseen patterns. If it is capable of classifying them correctly with high probability, it is said that F *generalises* well.

In general, Pattern Recognition could be described by the stages shown in Fig. 2.1. The first stage involves identification of properties (or *features*) that make classes of objects *distinct* from each other. Then, we need to obtain these features for each sample of the set of interest (data acquisition stage). The features could be, for example: blood results of patients from two categories (healthy or non-healthy); data for industrial products from several categories (e.g. different size and colour); persons' salary and number of dependants in order to approve the persons for bank credit, etc. This is the process of assigning a *pattern* of values to each object of interest. Subsequently, on the basis of this data, a prediction could be made about the class of the samples. Usually the second stage involves statistical processing of the measurements obtained and transformation of the data into *suitable* form. The third stage is the *classifier design*, where the information from the available data is used for building a class prediction model. The main contribution of this research concerns the *classifier design* stage, in particular – the *learning* process. Finally, there is a system evaluation stage, where the performance of the classifier is tested and conclusions about its effectiveness are made. In practice, very often these stages are interrelated

and, depending on the results, one may go back to redesign earlier stages in order to improve the system performance (Theodoridis and Koutroumbas, 2006).

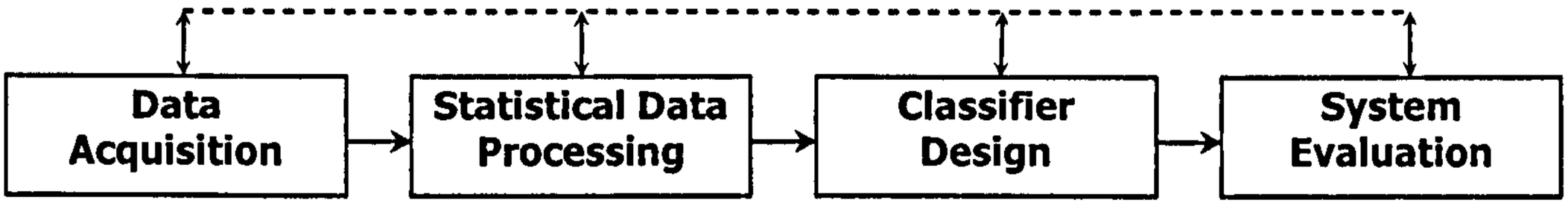
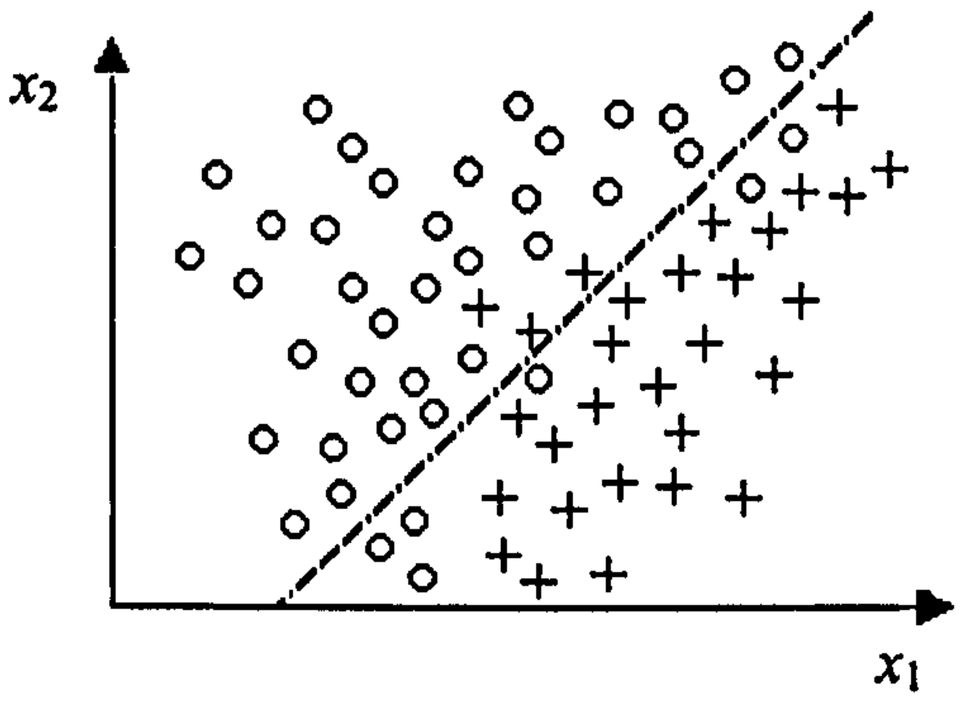
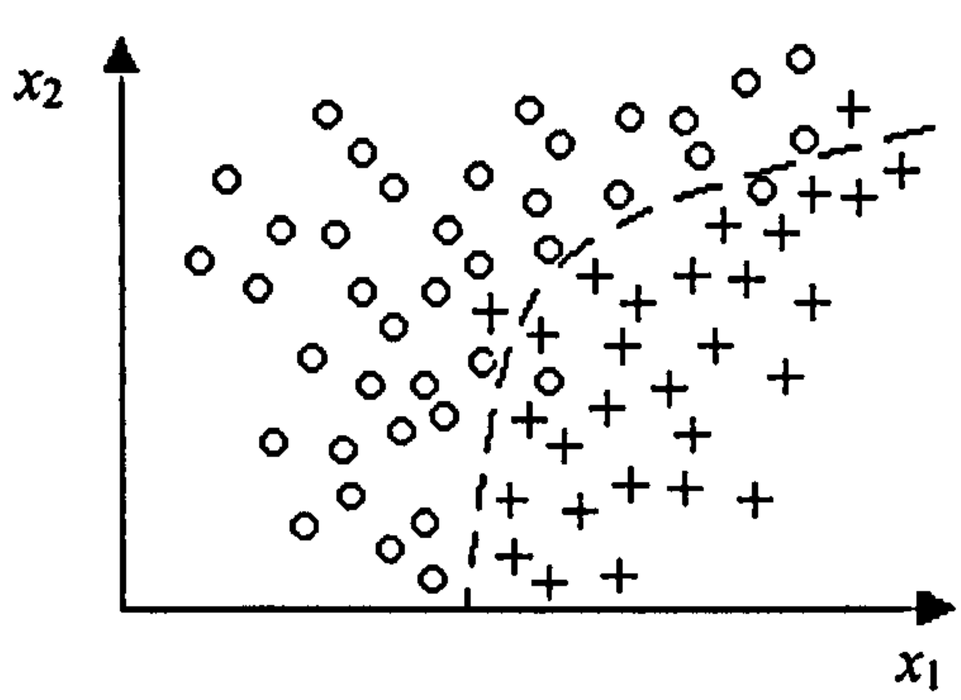


Figure 2.1. The basic stages involved in the design of Pattern Recognition systems.

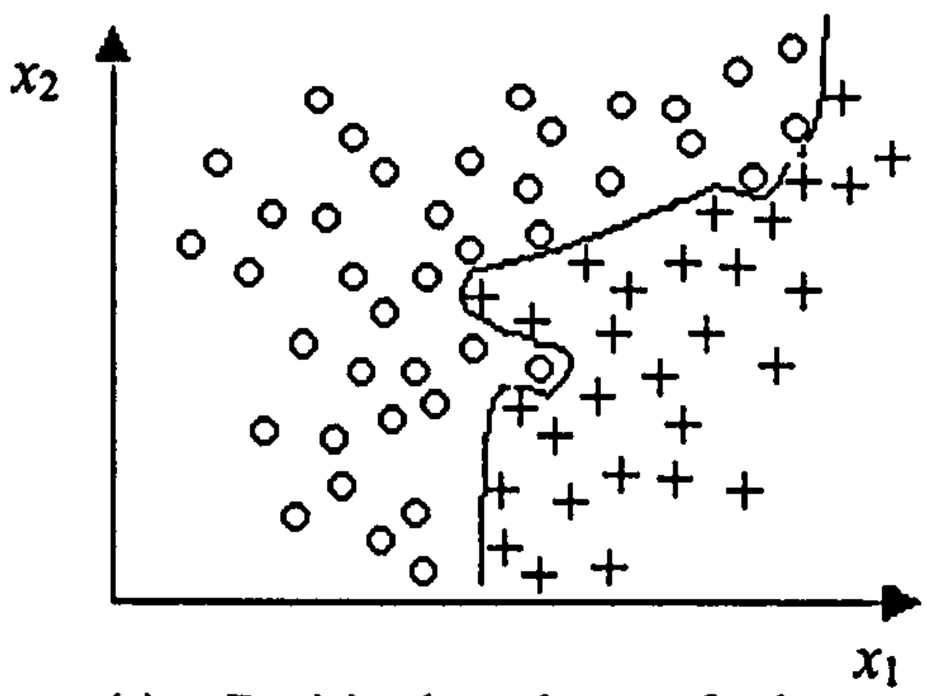
The classification task could be illustrated by Fig. 2.2 where two feature variables x_1 and x_2 represent the training patterns from two classes. The lines are called *decision boundaries* (or *discriminant* functions) and three different ones are considered here. New patterns that lie above the decision boundary are classified as of Class 1, while patterns falling below the decision boundary are classified as belonging to Class 2.



(a) Linear decision boundary.



(b) More flexible decision boundary.



(c) Decision boundary perfectly separating the data.

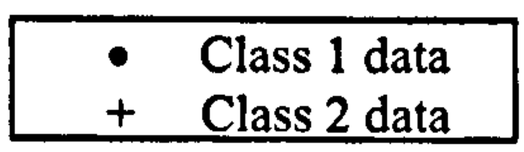


Figure 2.2. Different decision boundaries separating the data of two classes (adapted from similar diagrams in Bishop (1995)).

The linear boundary (Fig. 2.2(a)) is able to provide separation of two classes but there are still some patterns which would be incorrectly classified by this boundary. Fig. 2.2(b) shows a more flexible (and complex) decision boundary (given by the dotted line), which is capable to better separate the training data. The curve on the last figure (Fig. 2.2(c)) shows a decision boundary of a very flexible classification model that is able to achieve perfect separation of the training data. However, in many applications the distributions of the data from different classes overlap, and, while a linear classifier might give poor generalisation, the best performance is achieved by a model with intermediate complexity and not by the one that perfectly learns the data (Bishop, 1995). In this case, the classifier tends to learn the noise in the training data loosing the ability to generalise. This issue is discussed in Section 2.2.2.

In the next subsection, a further demonstration is given using the *Exclusive-OR* (XOR) task, which is a classical toy problem.

2.1.2 Demonstrative Example – the XOR Problem

One of the most famous artificial benchmark classification tasks is the *Exclusive-OR* (XOR) problem (Prechelt, 1994; Smagt, 1994; Magoulas *et al.*, 1997; Plaginakos *et al.*, 2001; Bortoletti *et al.*, 2003, Wang *et al.*, 2004). It provides a simple example of a task that is not linearly separable and, thus, can not be solved with one linear decision boundary. Since during the first era of NN research in the 1960s, no learning algorithm was known to be able to solve a not linearly separable task, such as XOR, according to Prechelt (1994), the popularity of this problem originates from the fact that being able to solve it was a major breakthrough. Depending on the values of the input patterns $\boldsymbol{x} = [x_1, x_2]^T$, the output is either 0 or 1, and $\boldsymbol{x}$ is classified into one of the two classes A(1) or B(0). The corresponding truth table for the XOR operation is shown in Table 2.1.

Table 2.1. Truth table of the XOR problem.

x_1	x_2	XOR	Class
0	0	0	B
0	1	1	A
1	0	1	A
1	1	0	B

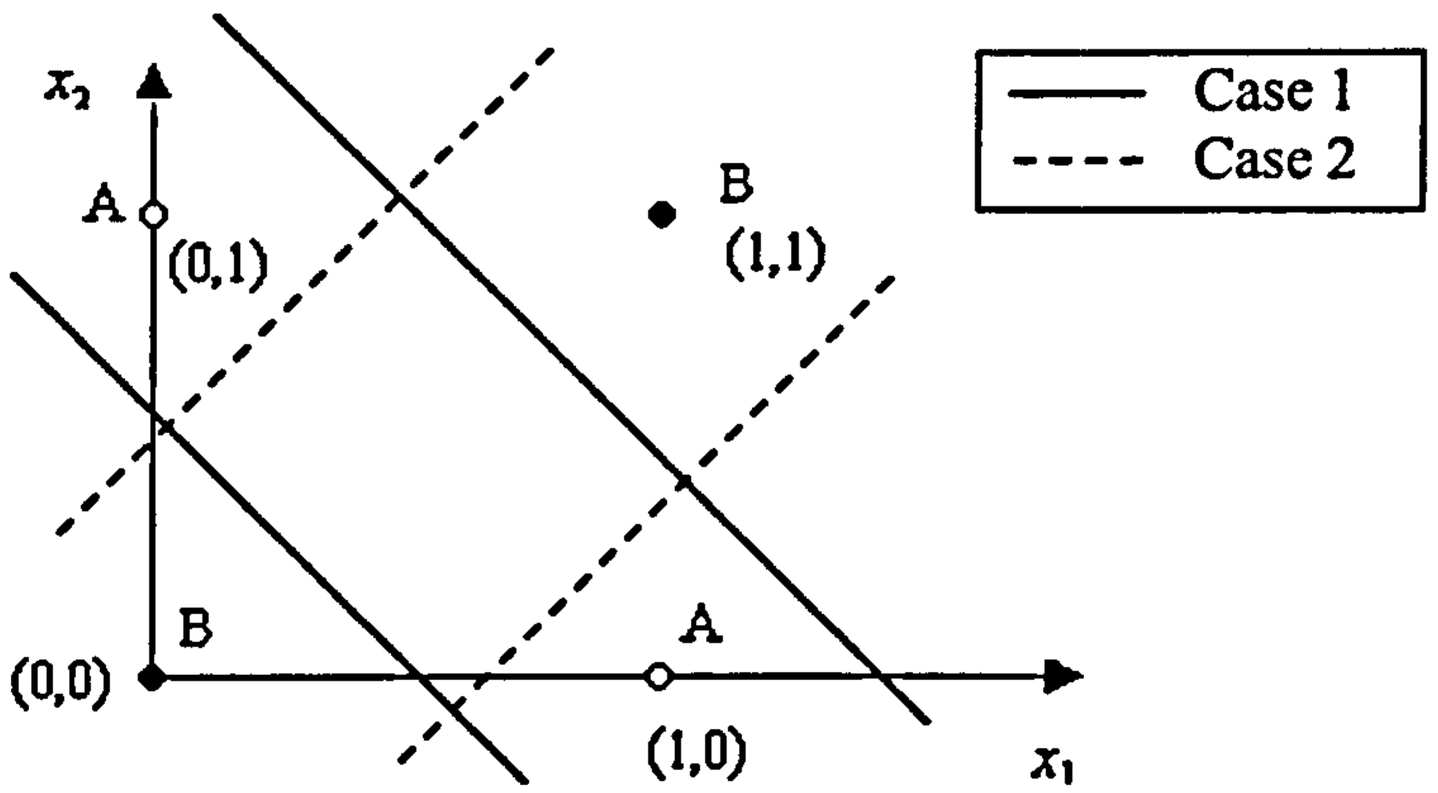


Figure 2.3. Decision boundaries for the XOR problem. A solution is provided by two linear boundaries and two possible cases are shown.

There are four training patterns in the two-dimensional space, corresponding to all four possible combinations of the binary inputs x_1 and x_2. The positions of the four patterns in the feature space are shown in Fig. 2.3. One can see rather intuitively that only one linear decision boundary is not capable of separating the two classes. However, two linear boundaries perfectly solve the classification task. The XOR is very popular problem and will be referred throughout the next chapters. It is known that the minimal NN architecture capable of solving it, is 2-2-1 (two hidden units and one output unit) with connections only between the successive layers, or 2-1-1 when there is a connection directly between the input and output layers (Hohil *et al.*, 1999).

The XOR problem can be generalised to N dimensions in which case it is known as the *N-Parity* problem and will also be referred to in the future chapters. In this case, the data set consists of all possible binary input vectors of length N, which are classified as class A if there is an even number of 1's in the input vector and as class B otherwise. These problems have the property that the smallest possible change in the input patterns produces the largest possible change in the output.

2.1.3 A Brief Review of Various Classifiers

Various types of classifiers have been investigated in the literature. They could be broadly categorised into three groups: *Bayesian*, *Linear*, and *Non-linear* classifiers.

The Bayesian approach is based on the statistical nature of the features pointed out earlier. These classifiers are based on the principle of assigning *the most probable*

class for each pattern. This approach was considered in detail in Theodoridis and Koutroumbas (2006). One other method that could be considered as belonging to this category is the *k*–Nearest Neighbour classifier (*k*NN) which simply assigns to the pattern at hand the same class label that have most of the *k* neighbouring training samples. If $k = 1$, each sample is assigned to the same class to which belongs its nearest neighbour from the training set. There are several interesting theoretical results for *k*NN which give an estimate of the upper bound of the classification error probability. They show that providing the training set is large enough, *k*NN can have good performance (Bishop, 1995; Theodoridis and Koutroumbas, 2006). One of the drawbacks of *k*NN, as pointed out in Theodoridis and Koutroumbas (2006), is the complexity in the search of nearest neighbours, especially when *k* is large.

On the other hand, linear classifiers are simple and easy computable techniques. Even when the training data can not be linearly separated, one could still easily construct optimal linear decision boundaries based on an appropriate optimality criterion (Theodoridis and Koutroumbas, 2006). One very famous representative is the *perceptron* (or a single layered Neural Network), which is discussed in more details in Section 2.2.1. One other approach – the *linear discriminant* functions and decision hyper-planes is considered only as a data processing technique later in this chapter, and more details can be found in Dillon and Goldstein (1984), Theodoridis and Koutroumbas (2006). In the latter, two other linear classifiers were also discussed – *Least Squares* and *Linear Support Vector Machines*.

Non-linear classifiers deal with problems that are not linearly separable. One representative branch is the so-called *Decision Trees* that are multistage decision systems, in which classes are sequentially rejected until a class label is finally accepted. Detailed survey of decision trees can be found in Quinlan (1993a). Non-linear *Support Vector Machines* were discussed in Theodoridis and Koutroumbas (2006). However, maybe the most popular approach is the Multi-layered Neural Network which is the subject of this study and is considered in the following sections.

Brief classification of the available NN learning models can be found in Yao (1999). Supervised learning is based on direct comparison of the NN output with the desired target value. It is often formulated as the minimisation of an error function and this type of NN is the major focus of this research. On the other hand, *reinforcement*

learning is a special case of supervised learning where the exact desired output is unknown. It is based only on the information whether the actual output is correct or not. More details can be found in Sutton and Barto (1998). *Unsupervised* learning is based only on the correlation among the input data. No target values are available. Such methods are often called *clustering* or *self-organising maps* and more details can be found in Theodoridis and Koutroumbas (2006). There is also the so-called *semi-supervised* learning approach, which makes use of both patterns with known and unknown class labels – typically a small amount of labeled data with a large amount of unlabeled data. Detailed investigation of them was given in Chapelle *et al.* (2005).

2.2 Feed-forward Neural Networks

2.2.1 Definition and Characteristics

- The *Artificial Neuron* (*McCulloch-Pitts neuron*)

As mentioned earlier, all NNs are built of simple computational units (blocks), which are highly interconnected with each other and called *neurons*, in association with their real-world inspiration – the brain neural cells. The basic architecture of an artificial neuron is shown in Fig. 2.4, where its major components are defined as inputs (representing the brain *dendrites*), weights, *summing* function (representing the *synapse* of the biological cell) and *activation* (or *transfer*) function. The input values x_i are *weighted* with coefficients w_i, ($i = 1,\ldots, I$) and the *bias* weight w_0 and the *bias* x_0=1 are added. Thus, their scalar product is fed into a function $\bar{f}$ (usually nonlinear or *treshold* function). The output $\tilde{o}$ is given by expression (2.1):

$$\tilde{o} = \bar{f}\left(\sum_{i=1}^{I} w_i x_i + w_0 x_0\right) \tag{2.1}$$

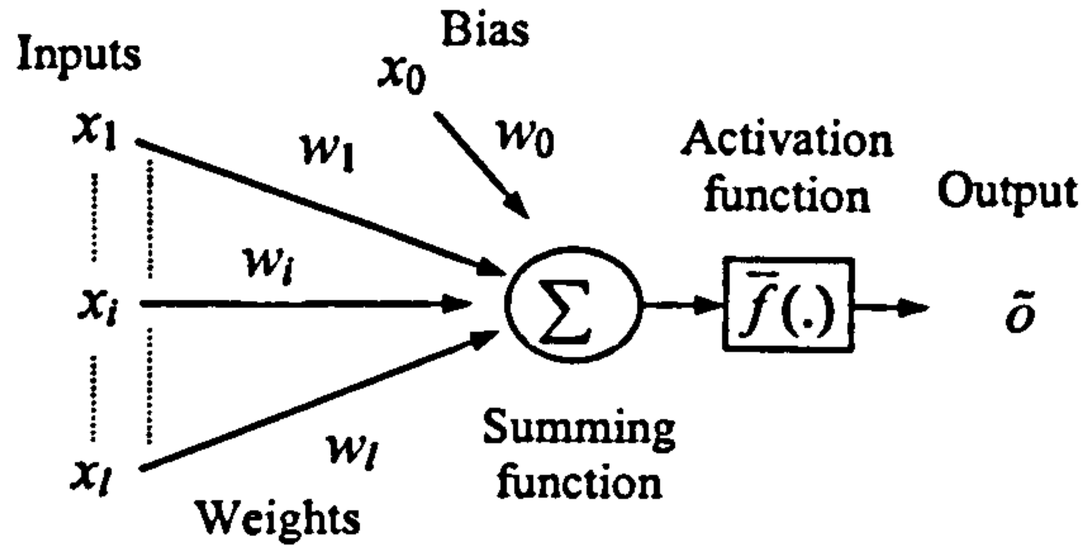


Figure 2.4. General architecture of an artificial neuron.

- The *Perceptron* and *ADALINE*

The simplest kind of feed-forward NNs are considered to be the perceptron and *ADALINE* (ADAptive LINear Elements, Widrow and Hoff, 1960). This simple NNs consist of only one neuron with a threshold activation function, given by (2.2) (See also Fig. 2.6(a)).

$$\bar{f}(a)=\begin{cases}0, & a<0\\ 1, & a\geq 0.\end{cases} \tag{2.2}$$

Often the anti-symmetric version of the threshold function (*Signum* function) is used:

$$\bar{f}(a)=\begin{cases}-1, & a<0\\ 1, & a\geq 0.\end{cases} \tag{2.3}$$

The innovative work on the perceptron and *ADALINE* is indeed the *learning rule* – that is the mechanism of adapting the weights in order to obtain optimal outputs. The learning rule is different for the perceptron (proposed in Rosenblatt (1962)) and for the *ADALINE* neural networks (proposed by Widrow and Hoff (1960)). However, both NNs are capable of solving only linearly separable problems, because they can result only in a linear decision boundary. Thus, the XOR problem considered earlier can not be solved by a single perceptron or *ADALINE*. Only if two neurons are used in combination, they could produce two linear decision boundaries and solve the XOR problem (Fig. 2.3).

- Feed-Forward Multi-layer NN

This is the reason why assemblies of neurons are being grouped together, resulting in the *single-layer* Neural Networks and later in *Multi-layer* Neural Networks. Single-layer networks correspond to a very narrow class of possible solutions and in many practical situations may not represent the optimal choice. To allow more general mappings, we consider networks with H layers, with different number of neurons in each, as illustrated in Fig. 2.5. The layers apart from the input and output ones, are considered to be *hidden* and have *hidden* neurons. Networks with just two layers with monotonically increasing transfer functions have been proved to approximate any continuous functional mapping with arbitrary accuracy, provided that the hidden layer has enough neurons (Hornik *et al.*, 1989; Bishop, 1995; Engelbrecht, 2002). Here we consider only NNs with no feedback loops, i.e. the signals are processed only in *feed-*

forward direction as shown in Fig. 2.5. A network is said to be feed-forward if it is possible to attach successive numbers to the neurons such that each unit only receives input signals from units having a smaller number. Such a network has the properties that the outputs can be expressed as deterministic functions (Bishop, 1995; Yao, 1999). If such numbering does not exist, the NN is called *recurrent*.

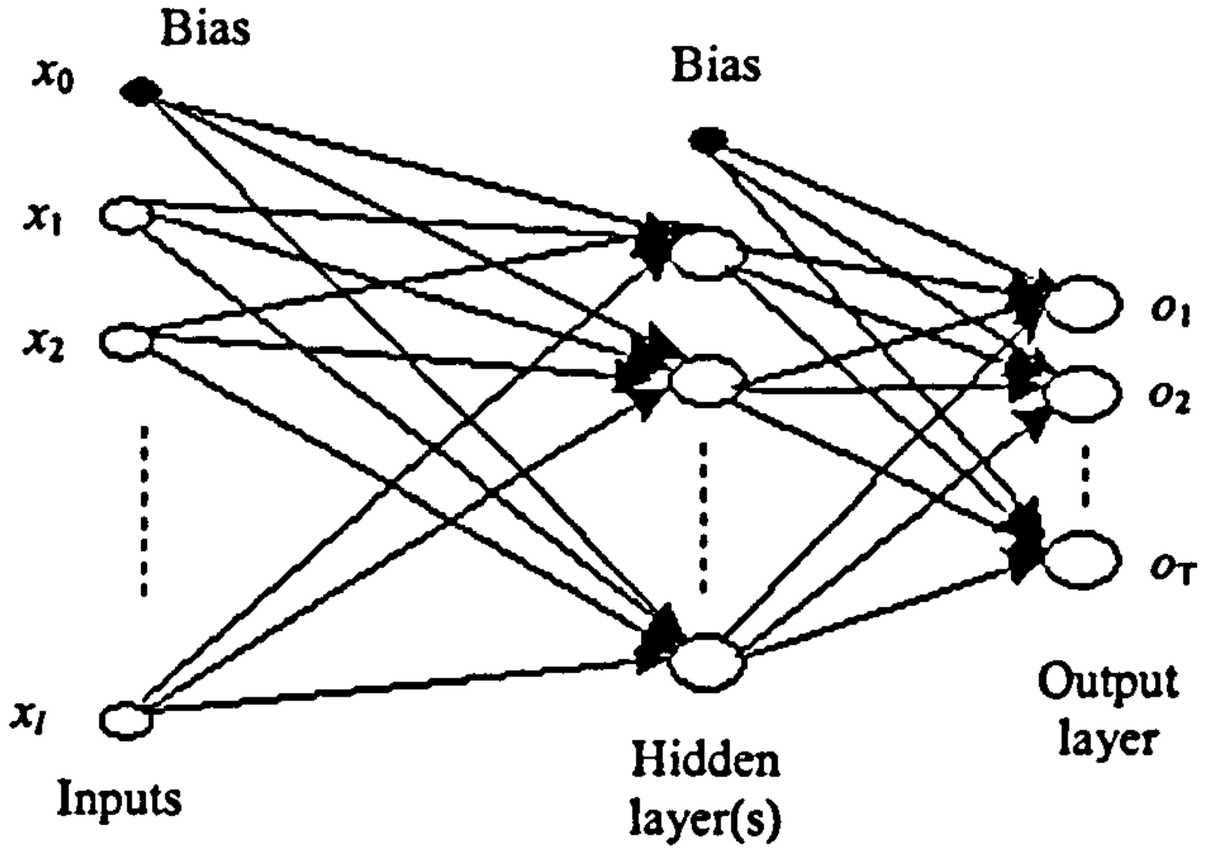


Figure 2.5. General architecture of a multi-layer Neural Network.

Often a constant number called *bias* is considered as additional unit to the layers, as demonstrated in Fig 2.2. The bias neuron lies in one layer and is connected to all the neurons in the next layer, but none in the previous one and it always has a value 1. Its purpose is to linearly shift (or translate) the separation boundaries into the search space, allowing more flexible learning.

The appropriate number of layers and neurons is problem-specific and there are few heuristic methods of choosing them: network growing, network pruning (Bishop, 1995; Haykin, 1999), and *rules of thumb* (Heaton, 2005). There are some theoretical results (discussed in Section 2.2.2) which connect the number of hidden neurons with the number of training samples.

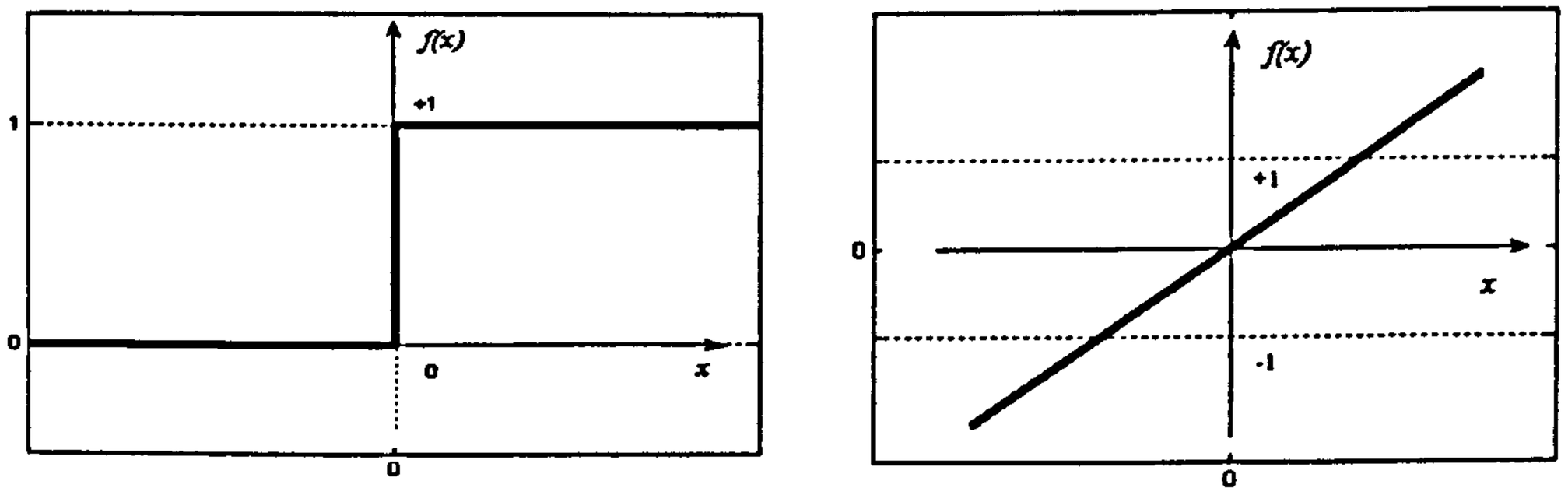
Further on, we will denote a specific architecture with the following notation: $I - h - o$, where I is the number of inputs, h is the number of neurons in the hidden layer and o is the number of outputs. If there is a bias in the input and the hidden layer, the total number of weights to be optimised is $n = (I+1)*h + (h+1)*o$, which defines n-dimensional optimisation problem.

- Types of activation functions

The activation function can be linear or nonlinear and typically is chosen to satisfy some specification of the problem that the NN is attempting to solve. Four of the most commonly used functions are described below:

A. The *Heaviside step* (*Treshold*) activation function is given by expression (2.2) and is graphically illustrated in Fig. 2.6(a). Typically, it is used in the output neurons when the NN aims to classify the input patterns into distinct categories. Sometimes the *Signum* function given with (2.3) might be used instead.

B. The *Linear* activation function $\bar{f}(x) = x.$ is a simple linear mapping and is illustrated in Fig. 2.6(b). Typically, it is used in the output layer when unbounded continuous output is required.



(a) *Treshold* activation function. (b) *Linear* activation function.

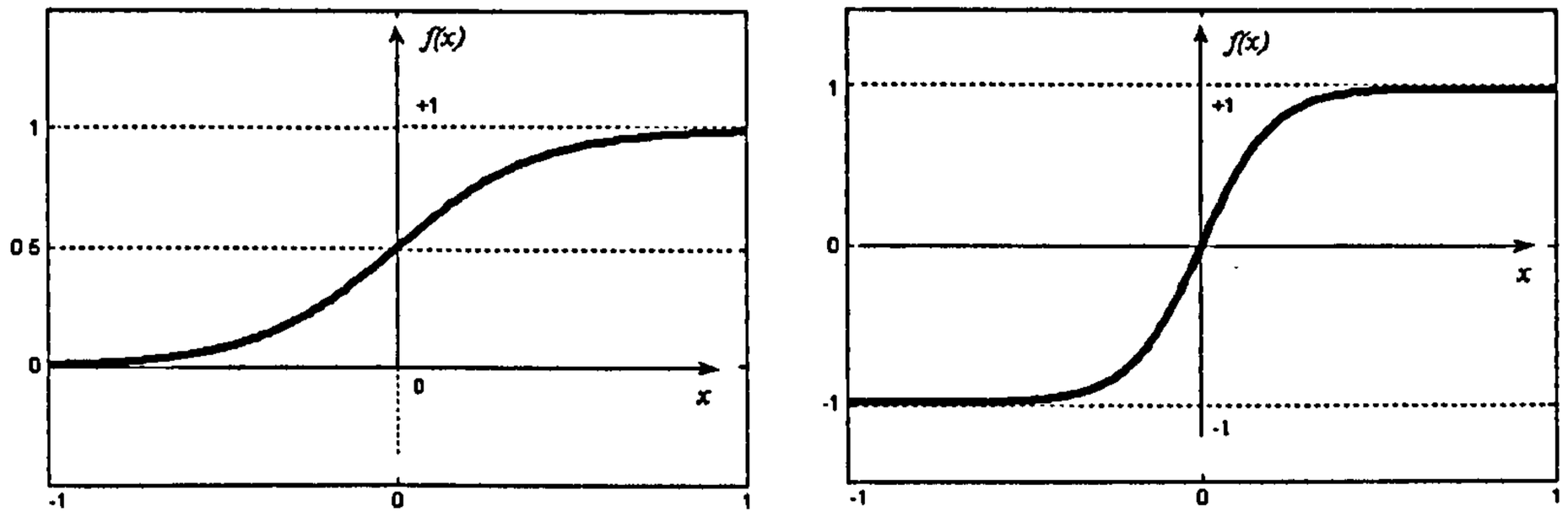
Figure 2.6. *Treshold* and *Linear* activation functions.

C. The *Standard Logistic* activation function (Fig. 2.7(a)) is a typical nonlinear *Sigmoid* transfer functions (because of its *S*-shaped graph), obtaining continuous values between 0 and 1.

$$\bar{f}(x) = \frac{1}{1+e^{-x}} \tag{2.4}$$

D. The *Tanh* activation function (Fig. 2.7(b)) obtains values in [-1, 1] which is also a type of *Sigmoid* transfer function.

$$\bar{f}(x) = \tanh(x) = \frac{e^{x} - e^{-x}}{e^{x} + e^{x}} \tag{2.5}$$



(a) *Standard Logistic* activation function. (b) *Tanhip* activation function.

Figure 2.7. Non-linear *Sigmoid* activation functions.

Bishop (1995) and Engelbrecht (2002) state that usually the use of *Tanh* activation function will result in a faster training than if *Logistic* one is used.

- Neural Network learning

The process in which the available P patterns are used to obtain decision boundaries (by adapting the NN weights) is called *learning* or *training*. It is based on the definition of a suitable *error function*, which is then minimised with respect to the weights and biases of the network.

Depending on the way training patterns are presented to the NN, *online* (*incremental*) and *batch-mode* training modes can be distinguished. In the online training mode, the weights of the network are updated each time an input is presented to the network. In batch training the weights are only updated after all the inputs are presented. Online mode is more commonly used with dynamic networks like adaptive filters (Hagan *et al.*, 1996). However, one might consider the paper of Leung *et al.* (2001), where online training is used for problems like the regression task and generalised XOR problem.

Recently, it has become popular for both the NN architecture and weights to be optimised simultaneously and detailed review on this topic can be found in Yao (1999).

There might be various types of error functions, but the most popular one is the Mean-squared error function (MSE), given by (2.7) for batch-mode training. It is just the mean of the squared errors for each training pattern given by (2.6) (which would be the error function in online mode).

$$E_p = E_p(\bar{W}) = \frac{1}{2}\sum_{i=1}^{T}\left(o_i^p - t_i^p\right)^2 \tag{2.6}$$

$$F = \frac{1}{P}\sum_{p=1}^{P} E_p(W). \tag{2.7}$$

Here $o^p = (o_1, \ldots, o_T)$ denotes the output of the NN for the pattern p, t^p – the desired target vector for it ($p = 1, \ldots, P$), where P is the number of training patterns. As it can be seen, F is a function of the NN weights and biases. Some other error functions include the Sum-squared error, or *Root MSE*. In this research, we adopted the MSE because it is the most common and widely-accepted error function (Bishop 1995, Engelbrecht, 2002).

The learning process is simply adjusting the weights (representing the *strength* of connections between the neurons) in order to produce a minimal difference between the actual outputs of the NN (o) and the corresponding targets (t). This difference is expressed in terms of error function. The weights are considered successively to form a vector W and the error function is minimised with respect to it. The learning task for a feed-forward NN is therefore considered as a *Global Optimisation* (GO) problem, where a minimum of the error function needs to be determined. The weights are usually randomly initialised at the beginning of the NN learning process. The initial interval should be adjusted with the NN activation functions. For example, if the NN has *Sigmoid* activation functions, Bishop (1995) and Engelbrecht (2002), recommend smaller initial weights (according to Engelbrecht (2002), the *active domain* of the *Sigmoid* function is $[-\sqrt{3}, \sqrt{3}]$), or weights that are from the same order of magnitude as the input values. The overall complexity of the learning problem depends on the NN architecture, training set size and the optimisation technique at hand. The architecture and training set are going to be discussed further in the following section.

Various opitmisation techniques applied for NN training will be considered further in this section.

If the error function is differentiable (e.g., the MSE given by (2.7)), we can evaluate the derivatives and one of the ways to approach this problem is to use *gradient optimisation methods*, i.e. methods that use gradient information in order to adapt the function weights in a suitable way. In the context of NN learning, they are often called *back-propagation* (BP) methods (Rumelhart *et al.*, 1986), deriving the name from the algorithm for the evaluation of the derivatives that corresponds to a propagation of errors backwards through the network. The term back-propagation is currently used with different meanings and a detailed discussion of this matter as well as derivation of the algorithm could be found in Bishop (1995). Here we refer to BP as any method applied for NN training, that uses gradients, e.g., gradient descend (Rumelhart *et al.*, 1986), conjugate gradients (Smagt, 1994), *Newton* methods, *Quasi-Newtonian* (Smagt, 1994; Bortoletti *et al.*, 2003) *Levenberg-Marquardt* method (Hagan and Menhaj, 1994), etc. Three different BP implementations were compared in Hagan and Menhaj (1994): conjugate gradient BP, BP with variable learning and the Levenberg-Marquardt BP. They were applied for NN approximation of five functions and Leveneberg-Marquardt BP was reported to be the most powerful of the three BP implementations. In general, all BP methods have several drawbacks that were discussed in detail in Burke and Ignizio (1997), but the most important one is the so called Local Minima Problem (LMP) that is the inability of BP methods to get out of a local minimum region of attraction (Smagt, 1994; Bishop, 1995; Yao, 1999; Plaginakos *et al.*, 2001; Engelbercht, 2002). Therefore, BP methods are considered to be *local searches* and are highly sensitive to the initial conditions as the most deterministic methods are. The LMP and the approach to its solution will be discussed in detail in Section 2.2. Nowadays, BP can be powerful and fast after some improvements (Smagt, 1994; Man *et al.*, 2006), but it is still very likely to fail for most practical problems, because the Error Function surface has multiple local minima as well as wide plateaus that can slow down the BP methods (Hush *et al.*, 1992; Bishop, 1995; Magoulas *et al.*, 1997; Plaginakos *et al.*, 2001). The existence of local minima is due to the fact that the error function in the general case is indeed a superposition of nonlinear activation functions that may have minima at different points, which results in non-convex surfaces (Smagt, 1994; Magoulas *et al.*, 1997).

There is research specifically on the conditions that imply *local minima free* function surface (Hush *et al.*, 1992; Gori and Tesi, 1992; Binachini and Gori, 1996; Man *et al.*, 2006). Conditions such as having linearly separable training data and the pyramidal structure of the NN, as well, as the need for as many hidden neurons as training patterns, make these results difficult for practical implementation. Also, there are several modifications of BP that aim to handle the LMP. Some authors proposed conditions on the error function that would guarantee LM free surface (Gori and Tesi, 1992; Bianchini and Gori, 1996; Magoulas *et al.*, 1997).

In order to handle the LMP even more effectively, techniques are adopted from the field of Global Optimisation (GO). Here, only a review of the application of such methods to NN training and some major results are discussed, while the GO techniques themselves are considered in more detail in Section 2.3. The variety of GO methods applied for NN learning includes stochastic techniques that use heuristic knowledge to generate points in the domain of interest – a number of random and quasi-random methods, such as Tabu Search (TS), Simulated Annealing (SA) and others. Other stochastic methodologies are the Evolutionary Algorithms (EA) which include Evolutionary Programming (EP), Evolutionary Strategies (ES), Genetic Algorithms (GA) and Differential Evolution (DE). They can be considered as *population based* algorithms (because at each *generation* (iteration) they improve not only a single solution, but a population of candidates). On the other hand, GO techniques can be combined with each other, or with gradient based approaches to form *hybrid* NN training methods. Currently, the hybrid methods seem to be quite promising for the NN training tasks (Alba and Chicano, 2004; Ludemir *et al.*, 2006).

Several authors used SA and TS to avoid LMP in NN training. Smagt (1994) used an artificially generated toy problem in order to assess the training ability of TS for a NN with six hidden units and compared the test performance with that of a NN trained with BP. TS derived solutions were significantly better than the ones of BP. Reactive Tabu Search (RTS) is a modified version of TS, proposed by Battiti and Tecchiolli (1995). In their approach, each connection weight was represented by a binary string in *Gray* code. This is coding that allowed successive integer numbers to be obtained by changing a single bit in the string at hand. The results obtained for RTS show that the method performed successfully for classification tasks.

In Chalup and Maire (1999), NNs with two hidden units were trained with various *hill-climbing* techniques and tested on the 5-bit parity problem. A hill-climbing method which uses in-line search was proposed and shown to perform better than SA and other methods.

When Evolutionary Algorithms (that include EP, ES, GA and DE) are used for NN training problems, the authors sometimes speak of *evolution* of connection weights (Yao, 1999). Unlike the case with BP methods, when EAs are used, the error function does not need to be differentiable or even continuous. EAs threat large, complex, non-differentiable and multimodal functions, which are a typical case in the real world. The EA approach to NN training consists of two major phases – first, to decide the representation of the weights – binary or continuous. Second, the minimisation process performed by EA need to be decided – use and types of cross-over and mutation. Different representations and search operators can lead to quite different training performance (Yao, 1999). Yao (1999) provided an extensive survey of the literature devoted to EAs and the way they can be applied to improve the NN performance – not only evolution of the weights, but also optimisation of the architecture, learning rules and input features. He listed almost 100 publications, devoted only to the application of EA to NN weights optimisation, pointing out various advantages of this approach. The author discussed the effects of the possible coding mechanisms, use of different cross-over and mutation operators, etc.

In Sexton *et al.* (1998), 12 variations of SA (varying parameter values) were compared with a GA, for six analytical toy problems, solved with NN architecture 3-7-1, (n = 25). The authors reported results in favour of GA in terms of both quality of the solution (assessed by training and testing errors) and computational effort. Subsequently, GA and SA were compared with BP for the NN training of two real-world problems (cancer diagnosis and financial time series prediction). After detailed discussion of the obtained results, the authors concluded that both BP and SA depended heavily on the starting points and the algorithm parameters, which may significantly impact the solution. On the other hand, GA outperformed both SA and BP for all testing examples.

EA appear more powerful than BP and its modification, but indeed hybrid methods that combine the advantages of EA or other GO techniques with those of local

searches are proved to be even better (Yao, 1999; Rocha *et al.*, 2003; Alba and Chicano, 2004; Ludemir *et al.*, 2006).

Hybrid methods were promoted over local searches and simple population based techniques in Alba and Chicano (2004). In their paper, the authors compared five methods: two BP implementations (gradient descent and Levenberg-Marquardt), GA, and two hybrid methods, combining GA with different local methods. They are used for NN learning applied to problems arising in medicine.

Ludemir *et al.* (2006) optimised simultaneously NN weights and topology with optimisation technique that is a hybrid method combining SA, TS and BP for the training of NNs. During each iteration, a set of new solutions was generated by rules of TS, but the best solution was not always accepted since this decision was guided by a probability distribution, which is the same as the one used by conventional SA. Meanwhile, the topology of the NN was also optimised and the best solution kept. Finally BP was used to train the best NN topology found in the previous stages. The new methodology was shown to compare favourably with SA and TS alone on four classification and one prediction problems.

Plaginakos *et al.* (2001) performed several experiments to evaluate various training methods – six DE implementations (with different mutation operators), BP, BPD (BP with deflection proposed in Magoulas *et al.*, 1997), SA, hybridisation of BP and SA (BPSA), and GA. They reported poor performance for the SA method, but still promoted the use of GO methods instead of standard BP. The reported results indicated that the population based methods (GA and DE) are promising and effective, although the *winner* in their study was their BPD method.

Several methods were employed in Rocha *et al.* (2003), for the NNs training of ten classification and regression problems and their performances were compared. One is a simple EA, two others are combining EA with local searches (in *Lamarckian* approach, that will be discussed in Section 2.3.3), differing in the adopted mutation operator, and their performance was compared with BP and modified BP. A hybrid of local search and EA with random mutation (*macro-mutation*) was found to be the most successful technique in this study.

Lee *et al.* (2004) used a deterministic hybrid technique that combines a local search method with a mechanism for escaping the local minima. The authors compared its

performance with five other methods, including GA and SA, for the solution of four classification problems. The authors reported worst training and testing results for GA and SA, and concluded that their method (proposed in the paper) is substantially faster than the other methods.

Yao (1999) discussed hybrid methods combining EA with BP (or local search), giving reference to a number of papers that report encouraging results, as well as pointing out some controversial results. He stated that the best optimiser is problem dependant and there was no overall winner.

2.2.2 Generalisation and System Evaluation

The process of NN learning is assessed by the value of the error function but what qualifies the learning as successful is the ability of the trained NN to recognise new unseen during the training patterns. This is called *generalisation* ability of the NN and is usually assessed by additional *testing* set of samples. The testing process is the *system evaluation* stage where, depending on how well the NN recognises (classifies) the new patterns, the system can be assessed. Then, if the results are not acceptable, changing and improving the previous stages could enhance the performance (the different stages of the Pattern Recognition process are interrelated, as was shown in Fig. 2.1).

In order to obtain good generalisation abilities, the NN has to learn the underlying similarities and differences of samples from different classes, while avoiding learning the inevitable *noise* and exactly *memorising* the data. Typical evaluation measures are the percentage of correctly classified test samples (Engelbrecht, 2002) – *success rate*, or the *testing error* that is a mean of the squared differences between the actual and desired output. If the output neurons have treshold activation functions, the success rate is the natural measure to be used. If the output data is continuous, either the testing error should be used, or the success rate could be measured by assigning to the output value the class that is closest to it, i.e. has the shortest *distance*. One additional measure of the system performance is the *correlation* between the output and target values for all test patterns (Engelbrecht, 2002).

Let us consider the conditions when a NN of certain topology is actually able to *learn* the data. It is preferable that the training patterns are *representative* for the

classes, i.e., be near to the class mean. Classification problems like XOR and *N*-parity provide all possible training samples, which is not the case for practical problems when only representative data is available. Therefore *N-Parity* might not be very appropriate for assessing a NN performance. The number of training patterns from different classes should also not vary too much from class to class, in order to avoid bias towards a certain category. For example, if the samples from class one are twice as many as those from class two, it is natural to conclude that the NN will learn better the first class.

Finally, the overall number of training samples is crucial. Too small number might not provide enough information about the classes, but too large number might provide too much knowledge and increase the information entropy. An estimate of the number of training patterns that are needed by a specific network with W number of weights, can be given in terms of the so-called *Vapnik-Chervonenkis* dimension – d^{VC} (Bishop, 1995; Hole, 1996; Engelbrecht, 2002). The parameter d^{VC} is used to derive bounds on the generalisation error as function of network and training size. The research of Baum and Haussler (1989) states, in summary, that for a two-layered NN with treshold activation functions with total number of weights W, in order to obtain testing error of above 90%, approximately $10*W$ number of training samples are necessary. Bishop (1995) and Engelbrecht (2002), stated that in practice this is an *upper bound* and a smaller number of training patterns might also achieve good generalisation abilities. This estimate is limited to discrete input values and generalisation limits for real valued inputs were derived by Hole (1996). Umbaugh (2005) suggested empirical assessment of the NN performance for increasing number of training samples.

In practice, if the NN learns the training set exactly, it does not perform well on the testing set, as if the training had stopped in another *near-optimal* solution (Bishop, 1995; Engelbrecht, 2002). This problem is called *over-fitting*, because the NN *fits* the training data too well. According to some authors (Burke and Ignizio, 1997; Engelbrecht, 2002) over-fitting occurs when the NN has too many hidden neurons and/or irrelevant input units. Then, if the NN is trained *too long* it starts memorising even the noise contained in the training patterns and loses the ability to generalise. Therefore, if a NN of particular size will over-fit the data, depends on *quality* of the training data used (Burke and Ignizio, 1997). The question about disposing of irrelevant input units and general improvement of the training data will be discussed

in detail in the following section discussing *data processing*. Here, we will consider some possibilities of handling the over-fitting problem without changing the architecture or the inputs. This is done by introducing a *validation* set. This is a set of patterns that are not involved in the teaching, but are used at each epoch to assess the NN performance. Initially, as the training error decreases, the validation error decreases as well. However, after a certain point, if over-fitting occurs, the validation error will start increasing since the NN starts memorising and loosing its generalisation abilities. We can stop training as soon as over-fitting is encountered, and hence, this method is also called *training with early stopping*. Usually, even if a validation set is used, the evaluation stage still involves assessing the NN performance on a third independent test set.

If the number of available data is limited, it might not be possible to have three independent sets for training, validation and testing. In this cases, Bishop (1995) recommends using *cross-validation.* The training set is divided to S random equal subsets. Then, the NN is trained with the $S-1$ sets and its performance is evaluated with the remaining set. This process is repeated for each combination of $S-1$ sets, and finally, the test error is averaged over all the S cases. In this way, a high proportion of the available data is used for training, while all patterns are employed for the estimate of the evaluation the cross-validation errors. However, this approach involves S independent training procedures and if they are time consuming, the learning might be impractical.

Bishop (1995) also discusses in detail the concept of using *Committees* of NNs. That is the training of a number of NNs with different parameters (different topology, learning mechanism, etc.), with the same data set and considering the average output obtained by this committee.

2.2.3 Data Processing

As stated earlier, one of the crucial moments in NN learning is the use of the available training data. There are many useful improvements of the quality of the training data that can be performed before *feeding* it to the NN. Some of them deal with the compatibility of the features, noise removal, de-correlation, dimensions reduction, etc.

- Compatibility of features, scaling and coding

Dillon and Goldstein (1984), Bishop (1995), and Davies (2005) discussed the cases when a set of features has no special comparability and claimed that placing them in the same features space, assuming that the scales on the various axes have the same weight factors, should be invalid. To solve this problem, Davies (2005) suggested *scaling* of the values. *Scaling* is indeed very important in the cases when a MSE error function is used, since the features that have greater values will dominate the MSE and the ones with small values will not effect the training much, i.e., the training will be biased towards learning the features with greater values. The simplest way of performing scaling using a linear mapping (over each feature of each pattern), where the points from interval $[a, b]$ are *linearly translated* into $[c, d]$ by the following rule: if $x \in [a, b]$, then $y = c + (d - c)^*(x - a)/(b - a)$. If the new interval $[c, d] \equiv [0, 1]$, then $y = (x - a)/(b - a)$.

Since it is just a simple linear translation, the correlations in each class are preserved, while the values are simply translated. Indeed, a more useful scaling is the *standardisation* of the input data. It is given by the following formula:

Let $x_1, x_2, \ldots, x_p$ be a sequence of values, $\bar{x}$ is their mean and σ is the standard deviation, then $y_i = (x_i - \bar{x})/\sigma$, for $i = 1, \ldots, p$. The new sequence produced by this linear operator will have zero mean and unit standard deviation. Therefore, all features transformed in this way will have the same mean and standard deviation. Compared to the simple scaling, standardisation is more useful because it spreads the features uniformly in the space. However, if further feature analysis that uses the vector means and deviations (such as *t-test* or *Linear Discriminant Analysis*) is to be performed, one needs to be careful, because in such cases, simple scaling would be a better option.

One of the data processing stages is the *coding* of nominal values. For example, if the data contains categories such as *colour*, 'yes' and 'no', output classes *A*, *B*, and *C*, etc., one of the possible approaches is to assign to each a nominal value, a number (continuous or integer). However, this might deteriorate the classification since it will make the values linearly dependant. A better approach would be to use 1-of-*c* coding (Bishop, 1995). In this way, values are coded as binary strings, that are linearly independent, i.e. (1; 0), (0; 1), (0; 0), etc. The drawback is that this coding will increase the dimensionality of the problem, since each nominal value would be

exchanged for several binary ones (the length of the binary string depends on the number of nominal categories). Often, in classification problems, the NN output layers have treshold activation functions and the target classes are coded as 1-of-*c*.

- De-correlation and dimension reduction

After using the scaling techniques, we might be interested in more serious data processing, that usually has the following major objectives:

De-correlation of the feature vectors – highly correlated features do not convey additional important information and, in general, would only add redundant information and increase the information entropy (Bishop, 1995; Davies, 2005). Simply said, if the features are strongly correlated, this means they contain the same information repeated many times;

Dimensionality reduction – having too many inputs for the neural network increases the dimensionality and complexity of the problem (so-called *curse of the dimensionality*). Using reasonably easy computable dimension reduction techniques, or selecting a subset of the feature vectors could reduce the problem complexity and save a lot of computational effort. Appropriate features could be either selected or combined into a fewer number. Feature selection could be done in several ways and based on several selection criteria (Liu and Wang, 2003; Peng *et al.*, 2005; Puig and Garcia, 2006). It is also discussed in detail in Bishop (1995) and Theodoridis and Koutroumbas (2006). Here we mention only one selection criteria based on *Multivariate analysis* – t-test, ANOVA (ANalysis Of VAriance), or MANOVA (Multivariate ANOVA), considered in detail in Hogg and Ledolter (1992). By the use of these classical statistical methods we are able to distinguish which of the available features have significantly different mean values for all classes and, thus, *separate* the classes.

A better approach to the dimensionality reduction problem is the combination of features to form new ones that are uncorrelated, independent, with better ability of separating the classes, or other useful properties (Hsieh *et al.*, 2006). As stated in Sierra and Echeverría (2006), there are two ways of combining the features: *unsupervised* and *supervised*. When no supervision is available, Principal Component Analysis (PCA) can be used and in supervised situations, Fisher's Linear Discriminant Analysis (LDA) is widely used.

PCA referred to as the Hotelling, Karhunen-Loeve, or eigenvector transform (Umbaugh, 2005). This is a linear transformation which searches new basis in the feature space that optimally represents this specific data. The new features, called Principal Components (PCs), are linear combinations of the original ones and their purpose is to explain as much of the total variation as possible, with as few of the PCs as possible. Mathematically, it involves finding the eigenvectors of the covariance matrix and producing a new feature set with the following properties: the new features (PCs) are optimally uncorrelated and disposed of redundant information (Davies, 2005; Theodoridis and Koutroumbas, 2006); the PCs are sorted in decreasing order of importance that represents decreasing variance. This property is very useful for the dimensionality reduction objective. The PCA is considered to be a unsupervised method, because it does not take into account the class labels of the data and does not use any information in order to maximise the *class separability*.

Class separability is defined as the ratio of the between-class variance and the within-class variance (Dillon and Goldstein, 1984; Theodoridis and Koutroumbas, 2005). This objective is also related to the previous two, and it addresses the problem that we are actually interested in – the features that distinguish well between classes and not the features that represent similarities. If the investigated features depict the class differences well, it would be easier for the NN to learn and generalise this classification problem. To address this issue, Fisher's LDA could be employed. Similarly to PCA, this is a transformation that seeks new features that are linear combinations of the original ones. Mathematically, it involves solving the generalised eigenvector problem for the between-class scatter matrix and the within-class matrix. The generalised eigenvectors form a transformation matrix, in which they are ordered accordingly to their corresponding eigenvalues: the greater the eigenvalue, the more discriminating power the eigenvector has. The new feature set has the following properties: the new features are uncorrelated; they are sorted in decreasing order of importance that is, decreasing ratio of the corresponding eigenvalue to the sum of all eigenvalues; they have optimal class separability (minimal within-class variance and maximal between-class variance).

Evolutionary Discriminant Analysis (EDA), proposed in Sierra and Echeverría (2006) is a novel and interesting approach that provides both data processing and class learning simultaneously. EDA looks for combinations of the original features so as to

have patterns projected as closer to their class mean as. The cost function of the algorithm is the classification error, calculated by assigning patterns to the class of the closest projected mean. However, EDA minimises this cost directly instead of maximising the class separability criterion defined above. It can be applied to situations where the tradiotional LDA cannot be used due to singularity of the involved covariance matrices. EDA is discussed in detail and results from testing on a number of classification problems are presented in Sierra and Echeverría (2006).

2.2.4 Benchmark Datasets and Real-world Applications

A collection of real-world *benchmark* datasets is given in Prechelt (1994), including problems for medical diagnosis like cancer, thyroid, heart, etc. These datasets are taken from the collection of the University of California, Irvine (UCI), repository (http://mlearn.ics.uci.edu/MLRepository.html). The datasets in this collection are meant for use in general machine learning problems.

Rabunal and Dorrado (2006) report a collection of real-world applications of NNs, some of them given in the following sections: time series prediction; data mining; civil engineering (hydrology and building areas); financial analysis (bond- and credit-rating prediction); music creation; support system for fisheries; cost minimisation in production schedule setting; intruder detection; astronomy application for stellar images, and others.

A review of applications of NN is given in Egmont-Petersen *et al.* (2002), where examples of NNs being successfully applied in document processing; identification – fingerprint analysis, face detection; defense – navigation and guidance systems, target recognition, etc.

Evolutionary Programming is used to train NN for the diagnosis of breast cancer employing radiographic features and patient age in Fogel *et al.* (1997). Small error rates are reported showing that NNs can be successfully applied for this problem.

Neural Networks with BP learning are successively applied for modelling of the telecommunication traffic in Austria (Fischer and Gopal, 1994). The NN prediction of the interregional teletraffic flows is compared with that of classical regression approach to show that NNs have better generalisation and prediction abilities.

Computer Vision is one of the fields in which NNs are applied to various tasks for the automation in industry – quality and process control of various products (Shapiro and Stockman, 2001; Graves and Batchelor, 2003, Sonka *et al.*, 2007). In addition, the case study of this research involves building and investigating an Intelligent Computer Vision System for the recognition and classification of cork tiles, and details can be found in Chapter 5 and Chapter 6, where the results from the experiments are presented.

2.3 Global Optimisation

The bound-constrained Global Optimisation problem is defined as to find a point $P^* \in \Pi$ from a nonempty, compact feasible set $\Pi \subset R^n$ that minimises (maximises) an objective function F, i.e. find P^*, such that:

$$F^* = F(P^*) \leq F(P), \quad \forall P \in \Pi, \tag{2.8}$$

where $F : R^n \rightarrow R$ is a continuous, real valued objective function.

Without loss of generality, we consider only the minimisation problem, since the maximisation problem could be easily transformed into a minimisation one and vice-versa. Local minimum (LM) is defined as a point P^*, such that $F(P^*) < F(P)$ for $\forall P$ in the neighbourhood of P^*. A global minimum (GM) is defined as one with the smallest value of all local minima. The compactness of Π and the continuity of F guarantee that a global minimum exists and is attained in the searched region Π. However, there are no mathematical conditions that can guarantee a certain point to be a global optimum, unlike a local optimum, for which gradient information (*Jacobian* and *Hessian*) can provide sufficient conditions. Therefore, as stated in Arora *et al.* (1995), an exact global solution of the problem can be obtained only by an *exhaustive search* that will require infinite number of calculations. GO methods operate by finding *approximations* of a global optimum for non-convex functions.

The methods for function optimisation could be broadly classified into two major groups – local and global. The deterministic classical gradient-based methods are local search techniques that depend on the initial conditions and could be trapped in the nearest local minimum (LMP). As mentioned in Section 2.2, these include gradient descent, conjugate gradients, *Newton* based methods, etc. More details of these methods can be found in Fletcher (2000). Clearly, the gradient methods cannot

be applied for optimisation problems of non-differentiable functions. There are several local searches that do not use gradients, e.g., *Simplex Search* (Nelder and Mead, 1965), which is discussed in detail in Chapter 3. Local searches are usually fast and powerful techniques in cases when the objective function has limited number of variables and most importantly is unimodal. However, in most real-world problems, the objective functions are multimodal, complex, multidimensional, and even discontinuous (Fletcher, 2000; Pardalos *et al.*, 2000).

This has led to the study and development of methods that search for global minima – GO methods. Broadly, the GO techniques could be separated into four groups – deterministic, heuristic and meta-heuristic (or stochastic), Evolutionary Algorithms, and hybrid methods. Deterministic methods are usually based on certain assumptions on the objective function that are not easy to check (Arora *et al.*, 1995). They also involve calculation of derivatives or Lipschits constants, which provide analytical rules and mechanisms and usually guarantee reaching a global optimum in a finite number of steps. The methods from the second group use heuristic and stochastic rules to guide the search, with meta-heuristic ones having even *adaptive* rules that tailor the search to the particular landscape of the objective function, without specific parameter tuning. Evolutionary Algorithms also use stochastic rules and are very powerful search techniques that are capable of handling high dimensional and complex problems, mainly because at each *generation* a whole *population* of points is improved rather than a single solution. The algorithms from all these three groups have specific advantages and drawbacks and in order to improve their performance, authors often combine them into hybrid approaches, which usually result in faster and more robust techniques (Yao, 2002). That is why we will consider the hybrid methods as a separate, fourth group.

Some methods are derived for discrete combinatorial problems, while others for continuous ones. Usually the ones that deal with combinatorial problems are enhanced in order to be able to handle continuous problems and further on, we will not distinguish between them explicitly. The same is valid for methods that handle constrained or unconstrained problems. Without loss of generality we are going to consider only the bound-constrained task (Arora *et al.*, 1995). What is common for all techniques is that the search results are a trade-off between the computational effort and the quality of the solution, i.e., the search is for the best possible solution found

with minimal effort. It could also be said that most of the techniques look for appropriate balance between *exploration* and *exploitation*. That is the computational effort spent in the search for promising regions (search on a global scale), and the one spent for a localised search, trying to exploit locally a region of interest found in the exploration phase.

The number of function evaluations is widely accepted measure for the efficiency and convergence speed of the GO methods (Yao *et al.*, 1999; Bessaou and Siaary, 2001; Chellouah and Siarry, 2000a, 2000b, 2003, 2005; Hedar and Fukushima, 2003; Lee and Yao, 2004; etc.). Deterministic GO methods usually need additional auxiliary computations, such as calculating derivatives or adjusting parameters of the algorithm for each particular function or a class of functions. In some cases the whole problem is predefined and a new optimisation problem needs to be solved during each iteration. This leads to numerous auxiliary computations, while the heuristic approaches and EAs usually solve directly the problem at hand. Conversely, population based methods need more objective function evaluations than the deterministic ones, and in many cases they require additional amount of computer memory (Litnetski and Abramzon, 1998). However, the growth of computer power and reduction of computational cost promoted recently the investigation and use of stochastic methods (Ali *et al.*, 1997).

Considering the above statements, one can conclude that the choice of a GO technique is a problem dependant one. If the objective function is very expensive to compute, maybe deterministic methods should be used, since they need smaller number of function evaluations, at the price of other auxiliary *expenses*. Otherwise, for easy computable objective functions of moderate dimensionality, heuristic and meta-heuristic techniques will do quite well. If the function is of higher dimensionality (above 50), population based techniques, may be the most suitable choice for handling such problems. But, for these problems, if in addition a high accuracy of the solution is needed, hybrid methods could be a better alternative (Joines and Kay, 2002; Hart *et al.*, 2005).

However, according to the *No Free Lunch Theorem* (Wolpert and Macready, 1997), on average, all methods behave similarly on the total set of search and optimisation problems. In other words, if technique *A* performs better than method *B* on a certain

set of problems Ω, it always can be found another set of problems $\Theta \cap \Omega = \varnothing$, for which the method *B* will perform better than *A*. Therefore, algorithms that adapt its parameters to the landscape of the objective function at hand (i.e., methods using meta-heuristics), exploiting the information within the problem search space, might be a naturally better choice when no information of the objective function is available *a priori* (Munteanu and Rosa, 2004).

In general, GO methods have applications in almost every problem that involves mathematical modeling - such applications include finance, allocation and location problems, operations research, statistics, structural optimisation, engineering design, network and transportation problems, chip design and database problems, nuclear and mechanical design, chemical engineering design and control, molecular biology, etc., (Ali *et al.*, 1997; Glover and Laguna, 1997; Fletcher, 2000; Pardalos *et al.*, 2000, Ali *et al.*, 2005). The above stated four groups of deterministic, heuristic, evolutionary algorithms and hybrid methods will be discussed separately in the following subsections.

2.3.1 Deterministic Global Searches

Deterministic approaches could be defined as methods that exploit analytical properties of the problem in order to generate a deterministic sequence of points converging to a global optimal solution. Convergence of the methods in a finite number of iterations is guaranteed in most of the cases. Deterministic GO methods usually involve transformation of the original problem into a new one, imposing auxiliary computations (Cetin *et al.*, 1993; Barhen *et al.*, 1997; Björkman and Holmström, 2000). Additional computations are encountered also for calculation of the derivatives or for parameters adjustment for each particular function. As an example we may consider the Terminal Repeller Unconstrained Subenergy Tunneling (TRUST) proposed by Cetin *et al.* (1993) and further investigated in Barhen *et al.* (1997). This method uses *subenergy tunneling transformation* of the objective function in order to define a new, more suitable for optimisation function. The transformed function is further investigated by means of gradient and Jacobian estimates, Lipschitz conditions, etc. It involves numerous complicated and computationally expensive conditions and indeed guarantees finding a global

minimum only in the case of one-dimensional problems, while for multi-dimensional cases there is no analytical guarantee of the convergence.

Acknowledging the need of improvement of TRUST for higher dimensional problems (results from TRUST are reported only for 1, 2 and 3 dimensional functions), Oblow (2001) proposed Stochastic Pijavskij Tunneling (SPT) that combines TRUST with a stochastic approach. SPT can give only a probabilistic guarantee that the best local minimum found is a global one. SPT is tested on benchmark functions from one to six dimensions and one real-world problem.

In other deterministic approaches, e.g., Radial Basis Functions (RBF) proposed by Björkman and Holmström (2000), no gradient information is used but the whole problem is predefined and a new optimisation problem needs to be solved during each iteration. This leads to numerous auxiliary computations, while stochastic methods usually solve directly the problem at hand.

2.3.2 Heuristic and Meta-heuristic Methods

In most of the cases, stochastic techniques that do not use any information in order to bound the solution (e.g. Lipschits estimates), guarantee only *asymptotic* convergence to a global minimum. This means that as the number of search points increases, the probability of finding a GM also increases (Törn and Žilinskas, 1989; Arora *et al.*, 1995; Oblow, 2001, Ali *et al.*, 2005).

Stochastic heuristic methods use rules that incorporate random values. Meta-heuristic ones have even *adaptive* rules that tailor the search to the particular landscape of the objective function without specific parameter tuning (Ribeiro and Hansen, 2002).

One example of heuristic method is the Simulated Annealing (SA) proposed in Kirkpatrick *et al.* (1983). This technique is based on random evaluations of the objective function, in such a way that transitions out of local minima are possible. A point with worse value of the objective function could be accepted for further interest with a certain probability decreasing during the search (called *cooling temperature*). Obviously higher temperature allows more such points to be investigated and thus increases the *exploration* abilities of the algorithm. With time, as the temperature parameter decreases, the *exploitation* phase of the method takes over. SA does not

guarantee that a GM will be found, but if the function has many good near-optimal solutions, SA has a good chance of finding one (Plaginakos *et al.*, 2001). Bilbro and Wesley (1991) proposed *Tree SA*, that is a method based on the ideas of SA, but naturally handling continuous variables. There are several successful improvements of SA, e.g. Fast SA (Szu and Hartley, 1987), Very Fast SA (Ingber, 1989), New SA (Yao, 1995), etc., that aim to increase its very slow speed of convergence. Enhanced SA is proposed in Siarry *et al.* (1997), designed to handle higher dimensional problems (up to 100 dimensions). The method is tested and compared with six other algorithms, including three modifications of SA, and proved to be more efficient. Hybridisations of SA with other techniques will be discussed in Section 2.3.4.

Another popular heuristic technique is the Tabu Search (TS), proposed initially in Glover (1989) and Glover (1990). The first comprehensive book for TS is Glover and Laguna (1997). TS is maybe the first GO technique that tries to utilise adaptive memory. This is done by keeping a record (so called *tabu list*) of the points in the search space that have been already investigated and are not considered as promising. Points that are *close* to these in the tabu list are always rejected. TS guarantees finding a GM only asymptotically. History of the work on TS applied for the optimisation of continuous functions is given in Teh and Rangaiah (2003).

Zheng *et al.* (2005) proposed a Staged Continuous Tabu Search (SCTS) that comprises three stages, with each devoted to one task, that are based on Continuous TS with different neighbour-search strategies. The method was tested on benchmark functions from 1 to 20 dimensions and the performance compared with Improved Genetic Algorithm to show that SCTS is more efficient on this testing set. Further testing was performed on the optimisation of fiber grating design for optical communication system where SCTS outperformed GA and SA.

One interesting method that is claimed to be "intermediate between purely heuristic and methods that allow assessment of the quality of the minimum obtained" is the Multilevel Coordinate Search (MCS), proposed in Huyer and Neumaier (1999) and based on the DIRECT method, proposed in Jones *et al.* (1993). Therefore, there could be derived deterministic guarantee for reaching a global optimum in a finite number of steps. This algorithm could be considered as a *branch and bound* one, since the searched space is divided into smaller and smaller hyper-cubes of interest. In MCS the

search starts from a single initial point by computing several other testing points in connection with the first one. The algorithm stops as soon as a value close to the pre-known minimum is reached and, thus, uses previous knowledge of the function which might be the main reason for MCS performing so efficiently on functions with up to 10 dimensions.

In Ali *et al.*, 1997, a number of stochastic techniques are tested on some complex practical GO problems and their performance compared. The considered methods include four different controlled random searches, two SA implementations and two clustering algorithms. The authors conclude that there is not a overall winner and the optimal choice of GO method is problem dependant. The same conclusion is promoted in Arora *et al.* (1995) where a major review of GO techniques is presented.

Some other stochastic techniques include: DTA – Dynamic Tunneling Algorithm (Yao, 1989); Topographical Global Optimisation using pre-sampled points (Törn and Viitanen, 1994); MARS – Multistart Adaptive Random Search (Litnetski and Abramzon, 1998); GRASP – Greedy Randomised Adaptive Search Procedures (Feo and Resende, 1995; Festa and Resende, 2000); EGO – Efficient Global Optimisation (Jones *et al.*, 1998); Particle Swarm Optimisation (Clerc, 2005); Discrete gradient method, where line search is partly replaced by a global search (Bagirov *et al.*, 2005), and others. Stuckman and Easom (1992) compare and discuss the performance and properties of clustering method, SA and the Monte Carlo method. Glover and Kochenberger (2002) as well as Ribeiro and Hansen (2002) provide useful surveys on meta-heuristic techniques, where SA, TS, GRASP, Ant Colony Optimisation, and others (including several EAs) are considered.

2.3.3 Evolutionary Algorithms

Here, we adopt the view of Yao (2002) and use the term Evolutionary Algorithms (EA) to include all three major branches: Evolutionary Strategies (ES), Evolutionary Programming (EP), and Genetic Algorithms (GA). However, here we will also consider Differential Evolution (DE) as a fourth major subsection of EA. All four branches have similar ideas, structure and properties. All EAs are population-based, and there is information exchange among individuals in the population by the selection and/or recombination operators. The search operators (or also called *genetic*

operators) are used to generate offspring (new population) from *parents* (current population). This process is summarised in Fig. 2.8.

EAs differ mainly in individuals' representation and the way the algorithms operators are used for modifying the population. The differences between ES, EP and GA are considered in detail in Yao (2002). ES use real-coding of the individuals, and although *recombination* operators have been introduced into ES, their primary search operator is still *mutation*. EP is used successfully in combinatorial problems, and when applied to numerical optimisation, it differs from ES only because it has different *recombination* operator and selection. Originally, GAs use binary string representations and their main search operator is the cross-over, while mutation is used with low probability rate. The major difference of DE and ES or GA is that in order to produce the new population, DE perturbs points with the scaled difference of two randomly selected population vectors. Different representations, search operators, and stopping conditions, typical for GAs are considered in detail in Goldberg (1989), Reeves and Rowe (2002), Yao (2002), Price *et al.* (2005). Here, we give a brief summary of some of them:

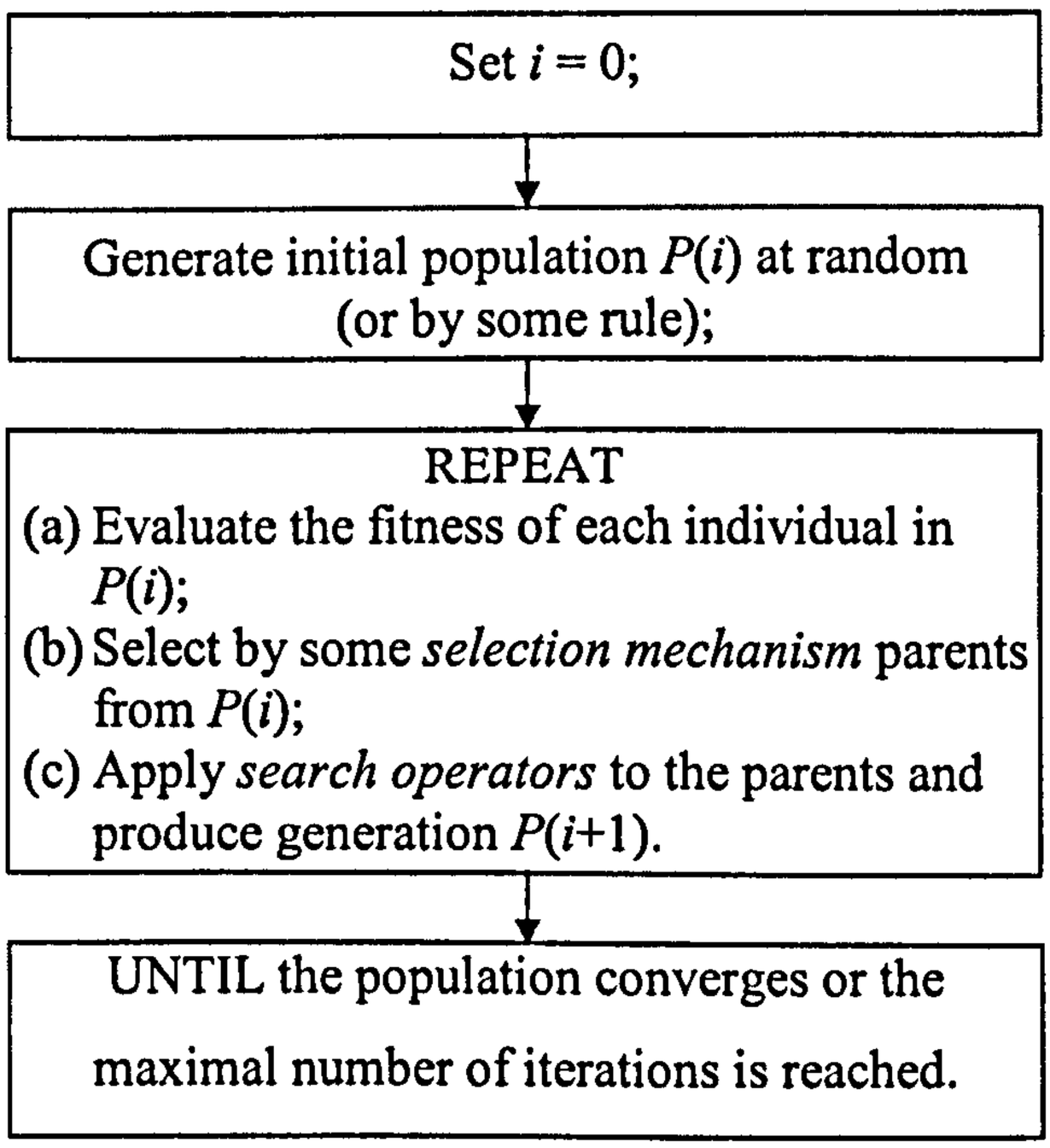


Figure 2.8. A general framework of Evolutionary Algorithms (adopted from Yao, 2002).

Representation of individuals

The representation of individuals depends on the problem at hand and for combinatorial problems, *binary* representation is more appropriate, i.e. each possible solution is represented by a binary string. Different encoding mechanisms are proposed, e.g. *Gray* coding (Goldberg, 1989) and *delta* coding (Mathias and Whitley, 1994). Initially, Holland (1975) and Goldberg (1989) proposed GA as a method that uses binary representations even when searching for a optimal point for an objective function in the real space. However, nowadays, it is considered unnecessary to transform from real space to binary and back, so most authors use *real-coding*. This means that all population points are represented as real-valued vectors in the n dimensional space. Often authors that adopt real coding still call the method Genetic Algorithm (although it differs from the originally proposed one and indeed is transformed into ES method), (Yao, 2002; Price *et al.*, 2005).

Search operators

- *Recombination* or *cross-over*

Different methods for recombination are considered in Goldberg (1989), Mitchel (2001), Yao (2002), Engelbrecht (2002), etc. Some of them are appropriate for GA that use binary coding (*k-point* and *uniform* cross-over) and others for GA with continuous coding (*discrete* and *intermediate* cross-over).

- *Mutation*

Mutation operators are considered in Goldberg (1989), Mitchel (2001), Yao (2002), Engelbrecht (2002), etc. as well as for continuous and binary coding. They include *Gaussian*, *Cauchy*, *Bit-fliping*, *Random bit*, etc.

- *Selection*

The typical types of selection operators include *Random*, *Roulette Wheel*, *Rank*, and *Tournament* selection, discussed in Yao (2002), Engelbrecht (2002), and others. If the selection mechanism includes *Elitist* strategy, i.e. the best individual in each iteration is copied to the next one without any modification, it is proved by Rudolph (1994) that the convergence of GA is guaranteed.

A detailed survey of Genetic Algorithms is provided by Mitchell (2001) and a more advanced coherent overview of Evolutionary Computation by De Jong (2006).

In Reeves and Rowe (2002), there is a summary of the literature that investigates why GAs work from different perspectives: schemata, statistical mechanics, Markov chain theory, connections of GA with neighbourhood search methods, etc. Rudolph (1994) shows that GAs that adopt elitist strategy are obviously consistent and always converge to a global optimum.

The performance difference if *Cauchy* or *Gaussian* mutations is used for ES and EP respectively is investigated in Yao and Liu (1997) and Yao *et al.* (1999). A mutation operator using the *Lévy* probability distribution is investigated for EP in Lee and Yao (2004). Both *adaptive* and *non-adaptive* mutations are proposed and the methods are tested on a number of test functions (up to 30 dimensions) to show that EP with adaptive *Lévy* mutations outperforms classical EP using *Gaussian* mutations.

GA using real-coding are shown to have superior performance to classic GA (binary representations) and SA in Houck *et al.* (1995), where also a Matlab implementation is considered. Chelouah and Siarry (2000a) investigate Continuous GA (CGA) with real-coding and compare its performance with SA and TS. In Bessaou and Siarry (2001) CGA is further improved by splitting into sequence of three processes in order to refine the initial population by sub-sampling and the CGA performance is showed to be improved on a test set of benchmark functions from up to six dimensions. The initial population is improved by introducing novel techniques in the work of Leung and Wang (2001). Leung and Wang make use of *orthogonal design* to generate initial population and to define a new recombination operator and the technique is called Orthogonal GA with Quantisation (OGA/Q). Since the *orthogonal design* is applicable only to discrete variables, the authors use *quantisation* technique to adapt the method to continuous problems as well. The technique is tested on a number of multidimensional mathematical functions and compared with conventional GA in order to demonstrate the superiority of the proposed method.

Ho *et al.* (2004) proposed two Intelligent EAs, designed to handle problems of high dimensions by utilising heuristic rules in addition to classical search operators. The methods are tested on twelve functions of up to 100 dimensions. The stopping condition used is the number of function evaluations, and it is kept quite low (10,000 in the case of 100 dimensional problems) and therefore, the solutions obtained by the methods are often far from the optimal one (if more function evaluations were

allowed, better values would be obtained). It is possible that this comparison does not give a realistic measure of performance, since some of the methods usually outperform others after a good *exploration* of the search space. One of the new techniques is compared exclusively with OGA/Q (Leung and Wang, 2001). The Intelligent EA (IEA) outperforms OGA/Q in terms of number of function evaluations (for some functions, IEA performs six times faster). The authors of both papers often report cases for which the minimal value is zero, that the mean value achieved from their methods over 50 runs is exactly 0.0 with standard deviation 0.0. One might argue that, in general, such accuracy is achievable with a numerical method only by an *accidental hit* of the optimal point. This might be due to the way initial points are calculated. For example, in the cases of functions g_2, g_3 and g_{13} (*Rastrigin*, *Ackley*, and *Schwefel* functions), IEA obtains a solution 0.0 (0.0) for only 8420 function evaluations. These 30 dimensional functions are well known in literature to have multiple local minima, usually EAs need above 100000 function evaluations for solving them (Yao and Liu, 1997; Yao *et al.*, 1999, Leung and Wang, 2001).

Similar problems (*incidental hits*) are encountered in the paper of Tu and Lu (2004), where the authors investigate and propose a GA that employs a novel *stochastic coding* strategy, called Stochastic GA (StGA). The method is described in detail and testing results are shown for more than twenty mathematical functions of different dimensionalities of 2 to 100. For all testing examples with no exception, StGA outperforms all other eight Evolutionary Algorithms (EA) that is compared with. The difference in terms of efficiency (number of function evaluations) is dramatically in favour of StGA, reducing the computational effort sometimes with more than 100 times when compared with other methods. The novelty in StGA is the *Stochastic Coding Mechanism* – individuals are not coded as points in the space with n coordinates, but as a region with mean and variance in each direction of space. However, this idea is very similar to the one in Memetic Algorithms (considered in the next section), and also the simplex coded GA proposed in Hedar and Fukushima (2003). Several problems, mistakes and misunderstandings that could be found in the paper are critically discussed in Jordanov and Georgieva (2007b), as well as in Section 4.2.2.

Differential Evolution (DE) is proposed by Price and Storn (1995) and is considered as part of the Evolutionary Algorithms. In this method, initial population of N_p

individuals is randomly generated and on each iteration new population of points is derived by the DE operators. To produce a new trial vector, the scaled difference of two randomly selected population vectors is added to a third randomly selected population vector. In the selection stage, the trial vector competes against the population vector of the same index (trial vector u_i competes with the i-th individual for each $1 \leq i \leq N_p$). The procedure repeats until all N_p population vectors have competed against a randomly generated trial vector. The survivors of the N_p point-wise competition become parents for the next generation. DE terminates when a predefined *Value to reach* (VTR) is reached.

Extensive study of the properties and the performance of DE was provided in Price *et al.* (2005), where a survey of recent work on DE could be found as well. The authors consider four different implementations of the method and apply them for optimisation of a number of test functions of 2 to 30 dimensions.

Problems of DE like premature convergence and dependence on the parameter algorithms are considered in Zielinski *et al.* (2006), and recommendations concerning settings of DE control parameters are given.

The work of Ali *et al.* (2005) compares the performance of SA, two similar heuristic techniques (abbreviated IHR and HNS), DE, real-coded GA, and Controlled Random Search (CRS). A general conclusion is reached that "if the practitioner is willing to use $100n^2$ function evaluations, GA or CRS would be preferred, but if only $10n$ function evaluations were allowed then IHR and HNS should be implemented". In general DE did not perform encouraging in comparison with the other techniques in this study. However, DE is quite sensitive to its parameter values (Price *et al.*, 2005; Zielinski *et al.*, 2006) and is possible that the authors would achieve much better performance if other parameter values are chosen. This argument is somehow relevant to earlier results in Ali and Törn (2004), where DE modifications (also incorporating local searches) are compared with GA and CRS reporting superior performance for the improved DE versions.

Finally, we mention here a GO technique called *Scatter Search*, that is proposed and investigated in Glover (1998), Laguna and Martí (2005) and references therein. It is considered to be an Evolutionary Algorithm, which operates on small set of solutions and makes only limited use of randomisation (Laguna and Martí, 2005).

2.3.4 Hybrid Methods

One broad branch of hybrid GO methods are the Hybrid Evolutionary Algorithms, that combine GAs (or any other type of EA) with local searches (Joins and Kay, 2002; Hart *et al.*, 2005). They are often called *Memetic Algorithms* (MA). These methods are inspired by models of adaptation in natural systems that combine evolutionary adaptation of populations of individuals with individual learning within a lifetime. MAs are also known as Hybrid Genetic Algorithms, Genetic Local Search, Lamarckian Genetic Algorithms, Baldwinian Genetic Algorithms (Hart *et al.*, 2005). There are different kinds of MAs depending on where the Local Search is applied – before or after cross-over, before or after mutation, etc. In all cases, MAs aim to combine the good exploration abilities of EAs with the good exploitation ones of local searches (Joins and Kay, 2002). Joins and Kay (2002) provide an extensive survey on the subject, as well as some results from testing different MAs on a set of optimisation functions as well as a couple of non-linear integer programming algorithms. Hart *et al.* (2005) presents a *state-of-the-art* survey of the research on MAs. Memetic version of DE is presented in Noman and Iba (2005), where two different implementations of DE are considered. One method that could be considered as MA combining GA and Nelder-Mead Simplex Search (Nelder and Mead, 1965) is proposed in Hedar and Fukushima (2003). The authors considered each individual in the population of the GA to be a *simplex* instead a simple point. Each individual is represented by the best point of the simplex, obtained by few iterations of the Simplex Search.

Sometimes, local search (LS) is used only after the EA has found a minimal solution. In this case, the LS is utilised only to refine the solution and these types of algorithms (although also called Hybrid EAs) fall outside the MA frame. Such a method is proposed in Chelouah and Siarry (2003), which initially uses real-coded GA and after its population has converged, employs Nelder-Mead Simplex Search (Nelder and Mead, 1965). Later on, the same authors (Chelouah and Siarry, 2005) proposed a similar hybrid method, this time combining TS with the Simplex Search (CTSS) and compare it with the above one. It is shown that the CTSS method is more efficient for functions of lower dimensionalities (up to four).

One very interesting and promising EA based technique is proposed in Munteanu and Rosa (2004) that includes a number of novel adaptive heuristic rules and makes

use of local searches. The heuristic rules aim to define the "ruggedness" of a certain search region and according to this measure to switch between the exploration and exploitation phases. In order to define the measure of *ruggedness* the method makes use of local searches, and therefore, it is considered as a type of MA. The method is tested on eight benchmark functions (three of them specially designed for this algorithm) and the performance is compared with other EAs. However, the computational cost is not clearly stated, since the number of function evaluations for the local searches are not reported, and it might be expected to be very high.

Heuristic methods are often hybridised with LS or EA. For example, Adler (1993) made "a marriage proposal" for SA and GA, where SA based mutation and recombination are proposed. Yao (1991) proposed optimisation by Genetic Annealing (another hybrid of GA and SA). Yiu *et al.* (2004) proposed hybridisation of SA with gradient based LS. Hybridisation of *Artificial Ants* and LS is proposed in Solnon (2001). Siarry *et al.* (1997) investigated a version of their Enhanced SA that is hybridised with local searches. We already mentioned the hybrid combining SA, TS and local search used by Ludemir *et al.* (2006) for NN training.

2.4 Low-discrepancy Sequences of Points

2.4.1 Motivation

Most stochastic, heuristic, and evolutionary techniques use random points to start the search (Goldberg, 1989; Mitchell, 2001; Price *et al.*, 2005; others). It is important to have uniformly distributed starting points in the searched area in order to explore every promising region. There are studies showing that quasi-random points, which are generated deterministically could be used to produce better results in a variety of tasks, e.g., numerical integration (Niederreiter, 1992; Bratley and Fox, 1992); stochastic optimisation (Sobol', 1979; Kucherenko and Sytsko, 2005; Liberti and Kucherenko, 2005), etc.. The low-discrepancy sequences (LDS) of points (Sobol', 1979, 1985; Niederreiter, 1981; Faure, 1982; Bratley and Fox, 1992; Kucherenko and Sytsko, 2005) are uniformly distributed sequences which have some useful properties that could be easily implemented in GO techniques. For example, the *LPτ* sequences (Sobol', 1979, 1985), which were introduced in the late 1970s, have very good properties in terms of uniformity and discrepancy and produce very good results when

applied for global optimization (Kucherenko and Sytsko, 2005; Liberti and Kucherenko, 2005). The application of LDS in GO methods is investigated in Kucherenko and Sytsko (2005) where the authors conclude that the Sobol's $LP\tau$ sequences are superior to pseudorandom sampling. Some of the basic characteristics and properties of LDS are considered here.

2.4.2 Characteristics and Properties

Uniformly distributed sequences

Let infinite arbitrary sequence of points $\{P_i\}$: $P_0, \ldots, P_i, \ldots, P_N, \ldots$ be given in one-dimensional interval [0, 1], and let $\Delta \subseteq [0, 1]$ be an arbitrary subinterval. If for N points from this sequence $P_0, \ldots, P_{N-1}$, we denote with $S_N(\Delta)$ the number of points belonging to this subinterval, then the sequence $\{P_i\}$ is uniformly distributed in [0, 1] if the following equation holds:

$$\lim_{N\to\infty} S_N(\Delta)/N = |\Delta|, \ \forall \Delta \subseteq [0, 1]. \tag{2.9}$$

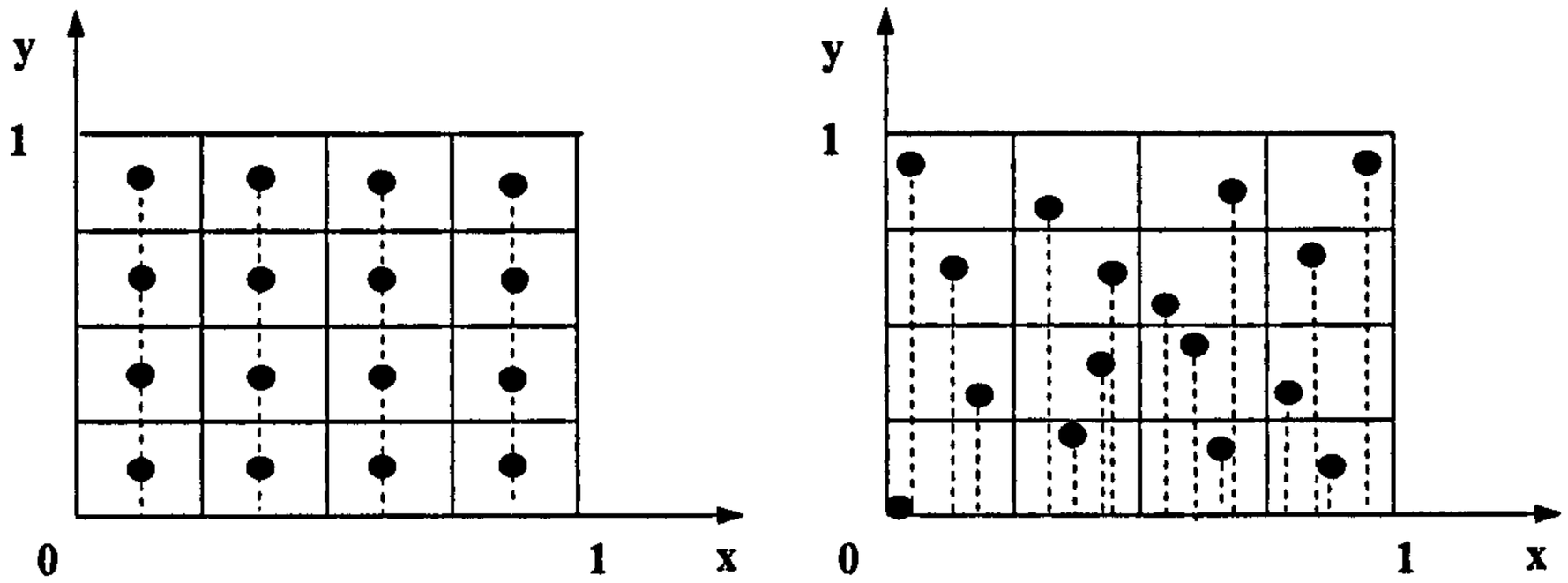
The geometrical interpretation of (2.9) is that when N is large enough, the number of points belonging to Δ, which we denote with $S_N(\Delta)$ is proportional to the size $|\Delta|$ of the subinterval, i.e., $S_N(\Delta) \approx N|\Delta|$ (Niederreiter, 1992; Sobol', 1985). In a multidimensional case with n dimensions, the uniformity of a sequence of points $\{P_i\}$, $P_i = (p_{i,1}, \ldots, p_{i,n})$, $i = 0, \ldots, N\text{-}1, \ldots$, in a unit hyper-cube C^n, is defined by the equality (Sobol', 1985):

$$\lim_{N\to\infty} S_N(D)/N = V_D, \ \forall D \subseteq C^n. \tag{2.10}$$

In (2.10), $D \subseteq C^n$ is an arbitrary hyper-cube with volume $V_D > 0$, and $S_N(D)$ is the number of points belonging to D. The geometrical interpretation of (2.10) is that when N is large enough, the number of points $S_N(D)$, belonging to an arbitrary hyper-cube $D \subseteq C^n$, is proportional to its volume V_D. Subsequently, when using such sequence in GO, if such hyper-cube that contains a GM point can be found, then the number of points of the sequence in it will be proportional to its volume.

Equation (2.10) defines the uniformity of a sequence, but when comparing two uniformly distributed sequences, it does not specify which is the better one (Fig. 2.9). In Fig. 2.9, the point sequence shown on the right hand-side is considered to be the

better one, because it explores the function behaviour with respect to the different variables. To quantify the uniformity, characteristics such as *discrepancy* and *dispersion* are used.



(a) Cubic sequence. (b) $LP\tau$ sequence.

Figure 2.9. Two different uniform sequences in two dimensions.

Discrepancy

Let us consider the one-dimensional case for a sequence with N points $\{P_i\}$, i=0,..., N-1, in the interval [0, 1], and let Δ be again an arbitrary subinterval, $\Delta \subseteq [0, 1]$. If we denote with $S_N(P)$ the number of points belonging to a half-open subinterval $\Delta = [0, P)$, then $S_N(P) = S_N(\Delta)$, and the discrepancy ρ of the sequence $\{P_i\}$ is given by the number

$$\rho(P_0, \ldots, P_{N-1}) = \sup_{P \in [0,\,1]} |S_N(P)/N - P|, \qquad (2.11)$$

where the supremum is extended over all half-open subintervals $|\Delta|$ (Niederreiter, 1992; Sobol', 1985).

Three arbitrary sequences with $N = 4$ points are shown on the P-axis of Fig. 2.10. The figure also shows the sequences ideal distribution (NP), and the actual number of points ($S_N(P)$), distributed in half-open subintervals $\Delta = [0, P)$, $0 \leq P \leq 1$.

Fig. 2.11 illustrates the geometrical interpretation of the discrepancy (given with (2.11)) for these three sequences. It is a quantitative measure for the deviation of the actual distribution $S_N(P)$ from the ideal distribution of points NP. The first of the three sequences from Fig. 2.10 is a non-uniform (equation (2.10) does not hold for it – one subinterval with no points and one subinterval with two points, Fig. 2.10(a)). The two other sequences (Fig. 2.10(b) and Fig. 2.10(c)) are uniformly distributed in terms of

(2.10). It can be seen from Fig. 2.11(a) that the first one has the worst discrepancy, and the third one (Fig. 2.10(c)) is with the best (lowest) discrepancy.

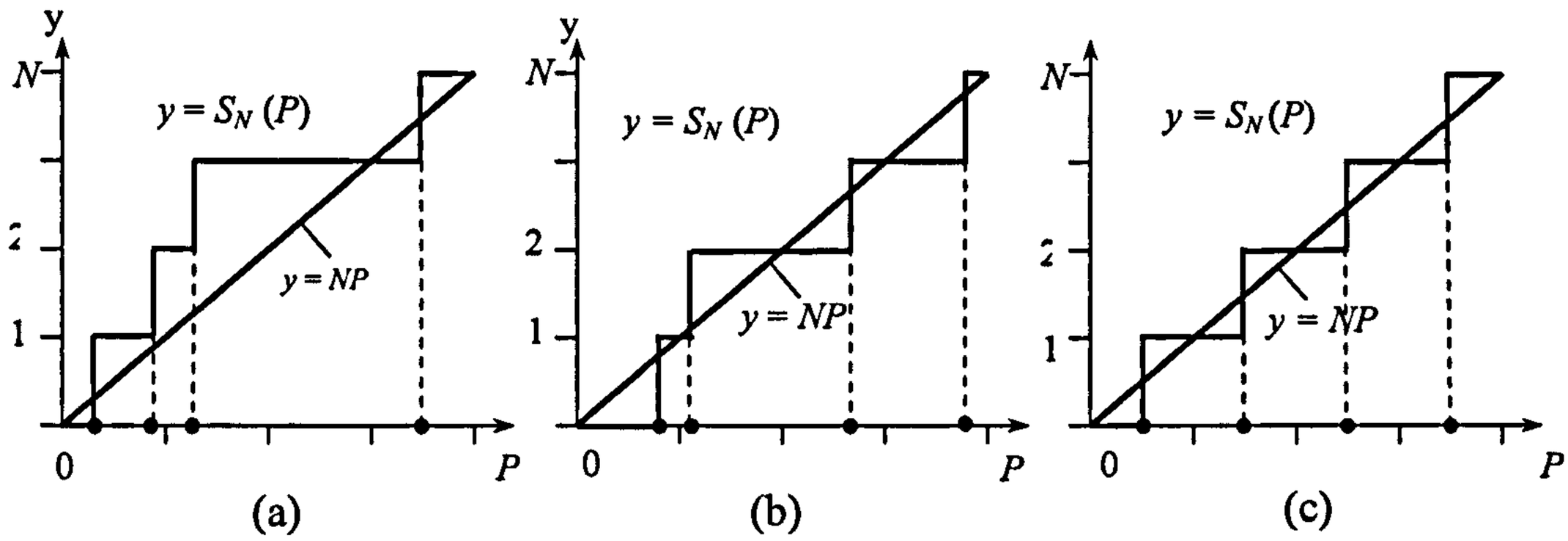


Figure 2.10. Ideal (NP) and actual ($S_N(P)$) number of points in [0, P) interval for three arbitrary sequences (one-dimensional case, n = 1, N= 4).

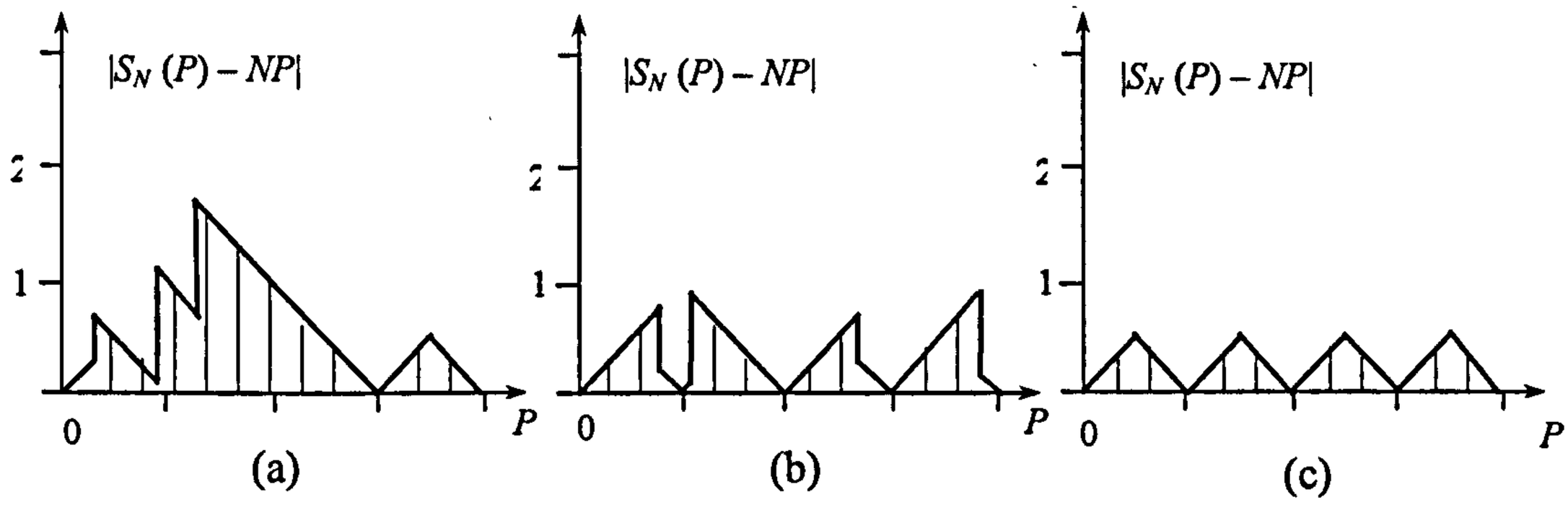


Figure 2.11. Discrepancy of the three sequences from Fig. 2.10.

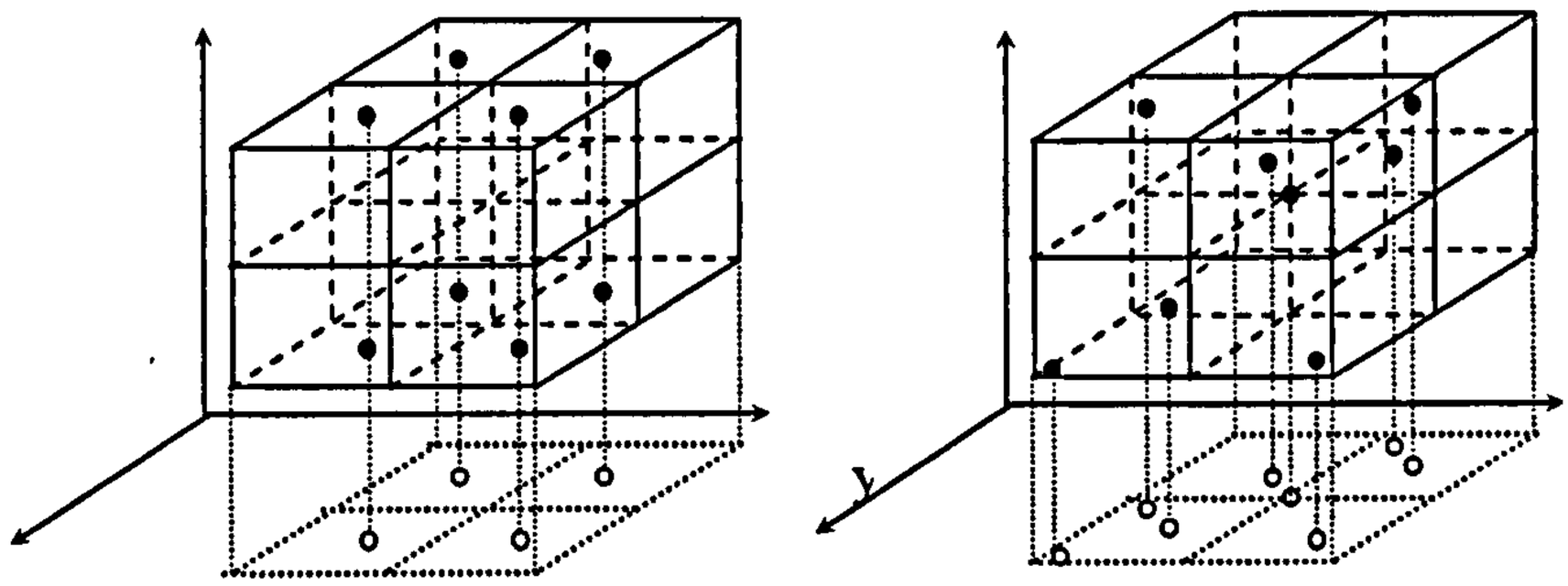
In the multidimensional case, the discrepancy of a sequence $\{P_i\}$ of N points in a unit hyper-cube C^n, where each point $P_i = (p_{i,1}, \ldots, p_{i,n}) \in C^n$, $i = 0, \ldots N\text{-}1$ is given with:

$$\rho(P_0, \ldots, P_{N-1}) = \sup_D |S_N(D)/N - V_D|, \qquad (2.12)$$

where $S_N(D)$ is the number of points belonging to an arbitrary n-dimensional sub-cube D, such that, $P_i \in D$, $D \subseteq C^n$ and V_D is its volume (Sobol', 1979, 1985).

The main aim of constructing LDS point sets is to minimise ρ from (2.12), because the smaller (lower) the discrepancy, the better the uniformity of the sequence. Constructions with successively smaller discrepancy values are obtained by Halton -

sequence with N variable (Bratley and Fox, 1992), Sobol' - *Pτ and LPτ sequences* (Sobol', 1985), Faure - *r-nary LP_0 sequence* (Faure, 1982), and Niederreiter – (τ, s) (τ, s) *sequence in base b* (Niederreiter, 1992).



(a) Cubic sequence. (b) *LPτ* sequence.

Figure 2.12. Two different uniform sequences in three dimensions.

Fig. 2.9 shows two different two-dimensional uniformly distributed sequences. The equality (2.10) holds for both of the shown sequences. Despite the fact that both sequences contain one point in every small square, the uniformity of the *LPτ* sequence is better than that of the cubic one when the dimensionality $n > 1$. The advantage of LDS is that they avoid the so called *shadow effect*, i.e., when projections of several points on the projective axes (hyper-planes) are coincident. As it can be seen from Fig. 2.9, the projections from the cubic sequence give four different points on the projection axis, each of them repeated four times; whereas the *LPτ* sequence gives sixteen different projection points. Therefore, the LDS would describe the function behaviour much better than the cubic one, and this advantage increases for greater dimensionalities. This feature is especially important when the optimised function is weakly dependent on some of the variables and strongly dependent on the rest of them (Sobol', 1985; Bratley and Fox, 1988; Kucherenko and Sytsko, 2005). The same shadowing effect is demonstrated in the three dimensional case in Fig. 2.12.

The application of LDS in GO methods is investigated in Kucherenko and Sytsko (2005) where the authors conclude that the Sobol's *LPτ* sequences are superior to the other LDS. Many useful properties of *LPτ* points are shown in Sobol', (1985) and tested in Niederreiter (1992), Bratley and Fox (1992), and Kucherenko and Sytsko (2005). LDS properties could be summarised as follows:

1. They retain their properties when transferred from a unit hyper-cube to a hyper-parallelepiped, or when projected on any of the sides of the hyper-cube;

2. They explore the space better, in a sense that avoid the *shadowing effect* discussed earlier and this property is very useful when optimising functions that depend weakly on some on the variables, and strongly the others.

3. Unlike the conventional random points, successive LDS *know* about the position of the previous ones and try to fill the gaps that are left (this property is true for all LDS). This is further demonstrated in Chapter 3 by Fig. 3.1;

4. It is widely accepted (Sobol', 1979; Niederreiter, 1992) that no infinite sequence can have a discrepancy ρ that converges to zero with smaller order of magnitude than $O(N^{-1}\log^n(N))$. The *LPτ* sequence satisfies this estimate. Moreover, due to the way *LPτ* are defiend, for values of $N = 2^k$, $k = 1, 2, \ldots, 31$, ρ converges with rate $O(N^{-1}\log^{n-1}(N))$ as the number of points increases (Sobol', 1979).

Relation between discrepancy and dispersion

The discrepancy is closely related with another characteristic of sequences – the *dispersion*. The dispersion of a sequence $\{P_i\}$ is given with:

$$d(P_0, \ldots, P_{N-1}) = \sup_{P \in C^n} \min_{0 \le k \le N-1} |P - P_i|,$$

and $\{P_i\}$ is considered to be *dense* in C^n if and only if $\lim_{N\to\infty} d(P_0, \ldots, P_{N-1}) = 0$.

Therefore, the dispersion of a point sequence is considered as a "suitable measure of denseness for sequences in order to estimate the rate of convergence" of quasi-random search methods (Niederreiter, 1981). If d is the dispersion (defined in means of Euclidean distance) and ρ is the discrepancy of any uniform sequence, then the following estimate holds: $d \le \sqrt{n}\,\rho^{1/n}$ (Niederreiter, 1981). Therefore, as $\rho \to 0$ with the increase of N, the better convergence rate of the discrepancy would ensure better convergence rate of the dispersion and better (*dense*) exploration of the region.

2.4.3 Generation

We can calculate a sequence of points $\{P_i\}$, $i = 0,\ldots, N-1$, with Cartesian co-ordinates $(p_{i,1}, \ldots, p_{i,n})$, in a hyper-parallelepiped Π^n using:

$$p_{i,j} = p_j' + (p_j'' - p_j')q_{i,j},\ j = 1, \ldots, n, \text{ where } \quad p_j' \le p_{i,j} \le p_j''. \qquad (2.13)$$

In (2.13), the superscripts $'$ and $''$ denote respectively lower and upper bounds of each dimension of the parameter space (and define Π^n). For the calculation of $q_{i,j}$, we use Gray code representation in binary system. If we denote with G(k) the Gray code representation of the deciamal number k and with B(k) its classical binary representation, the following connection holds: $G(k) = B(k) \wedge B(k/2)$, where "$\wedge$" denotes *exclusive-or* (XOR) operation and slash "/" denotes integer division. When programming, this expression can be easily coded using bitwise operators as $B(k) \wedge (B(k) >> 1)$, ">>" being the *shift left* operator. Starting recurrence with $q_{0,j} = 0,\ j = 1, \ldots, n$, subsequent $q_i = q_{i-1} \wedge r_{j,l}\, 2^{-l},\ i = 1, \ldots, N$ can be calculated. Direction constants $r_{j,l}$ can be obtained from Sobol' (1985). In the above equation, $1 \le l \le 30$, $0 \le N \le 2^{30}$, and $n < 370$. This method of generating the *LPτ* sequence we found to be very fast (there is also a slower method, using arithmetic formula), as also reported from Bratley and Fox (1988).

2.5 Summary

In this chapter the Pattern Recognition has been introduced and demonstrated by the famous XOR problem discussed in Section 2.1.2. The current *state-of-the-art* of various classifiers has been briefly presented providing an overview of the position of the research conducted in this study. In addition, the basic concepts of the feed-forward Neural Network learning problem have been introduced. Description and characteristics of the general form of a NN have been considered, as well as issues with the generalisation ability and NN evaluation. Further on, the Global Optimisation problem has been introduced and a review of the *state-of-the-art* GO literature has been given, pointing out the major differences between Local and Global Optimisation techniques as well as deterministic, heuristic, meta-heuristic, and Evolutionary GO methods. Finally, the Low-discrepancy Sequences have been discussed in detail.

3 Novel Global Optimisation Techniques Based on Low-discrepancy Sequences

This chapter presents a novel GO technique, based on meta-heuristic rules that utilise low-discrepancy sequences of points and their properties. The first section gives motivation and overview of the novel technique. The method is proposed (in Section 3.1.2), developed and its properties investigated (in Section 3.1.3). The novel technique is tested initially on a set of benchmark functions of moderate dimensions and subsequently employed for the NN training for several benchmark classification and prediction problems (Section 3.1.4). In order to improve the accuracy of the solution obtained by the novel technique, it is combined with the Nelder-Mead Simplex Local Search (in Section 3.2.1 and Section 3.2.2). The hybrid method is tested on a number of function optimisation (Section 3.2.3) and NN training problems (Section 3.2.4), and results are discussed in detail.

3.1 The *LPτO* Global Optimisation Technique

3.1.1 Motivation and Introduction

The first objective of this research is to propose, develop, and investigate novel meta-heuristic techniques that utilise the properties of the *LPτ* point sequences.

The advantages of meta-heuristic techniques were already mentioned in the previous chapter (Section 2.2.2). In the view of the *No Free Lunch Theorem* (Wolpert and Macready, 1997), it is clear that there always exists a set of functions on which a certain GO technique will perform poorly. The chances of improving a method's performance on a broader number of functions are increased by constructing meta-heuristic methods that aim to tailor the search and adapt the algorithm parameters to the landscape of the function at hand.

The advantages of the *LPτ* sequences were already discussed in Section 2.3. The *uniformity* of the *LPτ* points is optimal, they explore the objective function behaviour better (avoiding the *shadow effect*), and, when additional *LPτ* points are generated, they *know* the position of the others and try to *fill in the gaps*. The novel technique

proposed here utilises these properties for the construction of efficient meta-heuristic rules that guide the search through the different iterations. This is the reason why the method is called *LPτ Optimisation* (*LPτO*) underlying the key role of the *LPτ* sequences, which are employed in all iterations of the method.

The general structure of the *LPτO* technique is simple and conventional: *seed* the search space with *trial* points and compute the objective function for them; based on some rules (Section 3.1.2), select the points of future interest; based on this choice, generate new trial points; continue the above process until a halting condition is satisfied. There should be a condition (or usually a parameter), that allows switching between exploration and exploitation phases and ensures the convergence of the method. The same general structure lies behind SA, TS, EA, etc.

Stochastic techniques depend on a number of parameters that play decisive role for the algorithm performance assessed by the speed of convergence, the computational load, and the quality of the solution. Some of these parameters include the number of initial and subsequent trial points, and a parameter (or more than one) that defines the speed of convergence (cooling temperature in SA, probability of mutation in GA, etc.). Assigning values (*tuning*) to these parameters is one of the most important and difficult part from the construction of a GO technique. The larger the number of such decisive parameters, the more difficult (or sometimes even impossible) is to find a set of parameter values that will ensure an algorithm's good performance for as many as possible functions. Normally, authors try to reduce the number of such *user defined* parameters, but one might argue that in this way the technique becomes less flexible and the search depends more on random variables.

The advantage of the *LPτO* technique is that the values of these parameters are selected in a meta-heuristic manner – depending on the function at hand, while guided by the user. For example, instead of the user choosing a specific number of initial points N, in *LPτO*, a range of allowed values (N_{min} and N_{max}) is defined by the user and the technique adaptively selects (using the *filling in the gaps* property of *LPτ* sequences) the smallest allowed value that gives enough information about the landscape of the objective function, so that the algorithm can continue the search effectively. Therefore, the parameter N is exchanged with two other user-defined parameters (N_{min} and N_{max}) which allows flexibility when automatically N is selected,

depending on the function at hand. Another example is the parameter that allows switching between exploration and exploitation and, thus, controls the convergence of the algorithm. In SA, this is done by the *cooling temperature* (decreased by *annealing shedule*); in GA, by the *probability of mutation*, etc. These parameters are user-defined at the beginning of the search. In the *LPτO* method, the convergence speed is controlled by the size of the future interest regions, given by a radius R, and, in particular, the *speed* with which R decreases, defined by C_1 (Section 3.1.2). If R decreases slowly (large C_1), than the whole search converges slowly, allowing more time for exploration. If R decreases quickly (small C_1), the convergence is faster, but the risk of missing a GM is higher. In the *LPτO*, R and C_1 are not simple user-defined values, but are determined adaptively on each iteration and depend on the current state of the search, the *importance* of the region of interest, as well as the complexity of the problem (dimensionality and size of search domain). The convergence speed depends also on the parameter M, which is the maximal allowed number of future regions of interest. M is a user defined upper bound of the number of future regions of interest M_{new}, while the actual number is adaptively selected on each iteration within the bounds $[1, M]$.

The stability of the *LPτO* method with respect to these parameters (in particular M and N_{max}) is investigated in Section 3.2.3, as well as the stability with respect to the initial points and the search domain. The analytical properties of the technique are considered in Section 3.1.3 and results from testing the method on a number of benchmark functions are presented and discussed in Section 3.1.4 and Section 3.2.3.

3.1.2 The *LPτO* Method – a Detailed Proposal

Briefly, the *LPτO* algorithm could be described as:

Let n be the dimensionality of the problem.

Step 1. Generate N initial points in the area of interest and compute the objective function for them. Arrange the function values in non-descending order;

Step 2. Select several points that define several *promising* regions, that will be of future interest (see subsection '*Selection of the hyper-cubes of interest and their size*');

Step 3. In hyper-cubes, each containing as a centre one of the selected in step 2 points, generate new points and calculate the objective function for them;

Step 4. If no better point is detected among the new points, or the predefined maximum number of iterations is reached, stop. Otherwise, repeat steps 2-4 for the new points.

In the above algorithm, the adopted parameters are:

- number of initial points $N \in [N_{min}, N_{max}]$, where $N_{min} = 2^{n+1}$ and $N_{max} = 2^{n+4}$;
- *points of interest* to be selected $M_{new} \in [1, M]$, $M = 2^{n-1}$ (or another appropriate value);
- side a_i, $i = 1, \ldots, M_{new}$, of each hyper-cube surrounding a selected point;
- number of new points to be generated in each sub-cube $\overline{N}_i$, $i = 1, \ldots, M_{new}$, ($\overline{N}_i \in [\bar{N}^l_{min}, \bar{N}^l_{max}]$, where $\bar{N}^l_{min} = 2^n$ and $\bar{N}^l_{max} = 2^{n+3}$ (or another appropriate value)).

All of these parameters are interdependent and therefore, an *inappropriate*, choice of one of them could be compensated to a high degree by the others. The parameters N_{min}, N_{max}, $\bar{N}^l_{min}$, $\bar{N}^l_{max}$, and M are in general user-defined values that give lower and upper bounds of the number of *trial* points to be generated. Their recommended values are given as functions of the problem dimensionality n and were chosen after tedious testing procedure (demonstrated in Section 3.2.2, Table 3.6 and Table 3.7). The numbers N and $\overline{N}_i$ are always equal to the powers of 2, since we keep inline with the property of $LP\tau$ points to have optimal uniformity if their number is exponent of the number 2 (Section 2.3.2).

Discussion 3.1

As new points are generated in the neighbourhood of each selected point, the point itself is not lost and is still competing. Thus, the choice of the next best point (after the current pass) would be not worse than the previous one. This *descent property* guarantees the *consistency* of the algorithm.

The algorithm would terminate if the best point has not been improved during the last pass, or if the predefined number of passes is reached. It does not guarantee that a global minimum is obtained, but guarantees that a region of attraction is found, in which a local search could be started. In general, we have a probabilistic guarantee for

a GM to be found, which means that with the increase of the number of trial points, the probability of reaching a GM tends to one. This property is further discussed in Section 3.1.3.

Sometimes the search area is rather large in comparison with the area of attraction containing a global minimum and if a lesser number of points are generated in its neighbourhood, no improvement could be achieved during the first pass and the stopping condition could be misused to exit the algorithm prematurely. This is the reason why the algorithm is forced initially to run for at least two passes, i.e., we enforce the stopping condition after the second run.

Generating the initial points of the method

Since the method does not assume a priori knowledge of the GM, all parts of the parameter space must be equally treated, therefore, the points should be uniformly distributed in the whole region of initial search. The *LPτ* low-discrepancy sequences and their properties fulfill this issue satisfactorily. We also use the *LPτ* sequences property that additional *LPτ* points *fill the gaps* between the other *LPτ* points (Section 2.3.3). For example, if we have an *LPτ* sequence with four points and we would like to double their number, the resulting sequence will include the initial four points plus the new four ones in-between them. This is demonstrated in Fig. 3.1.

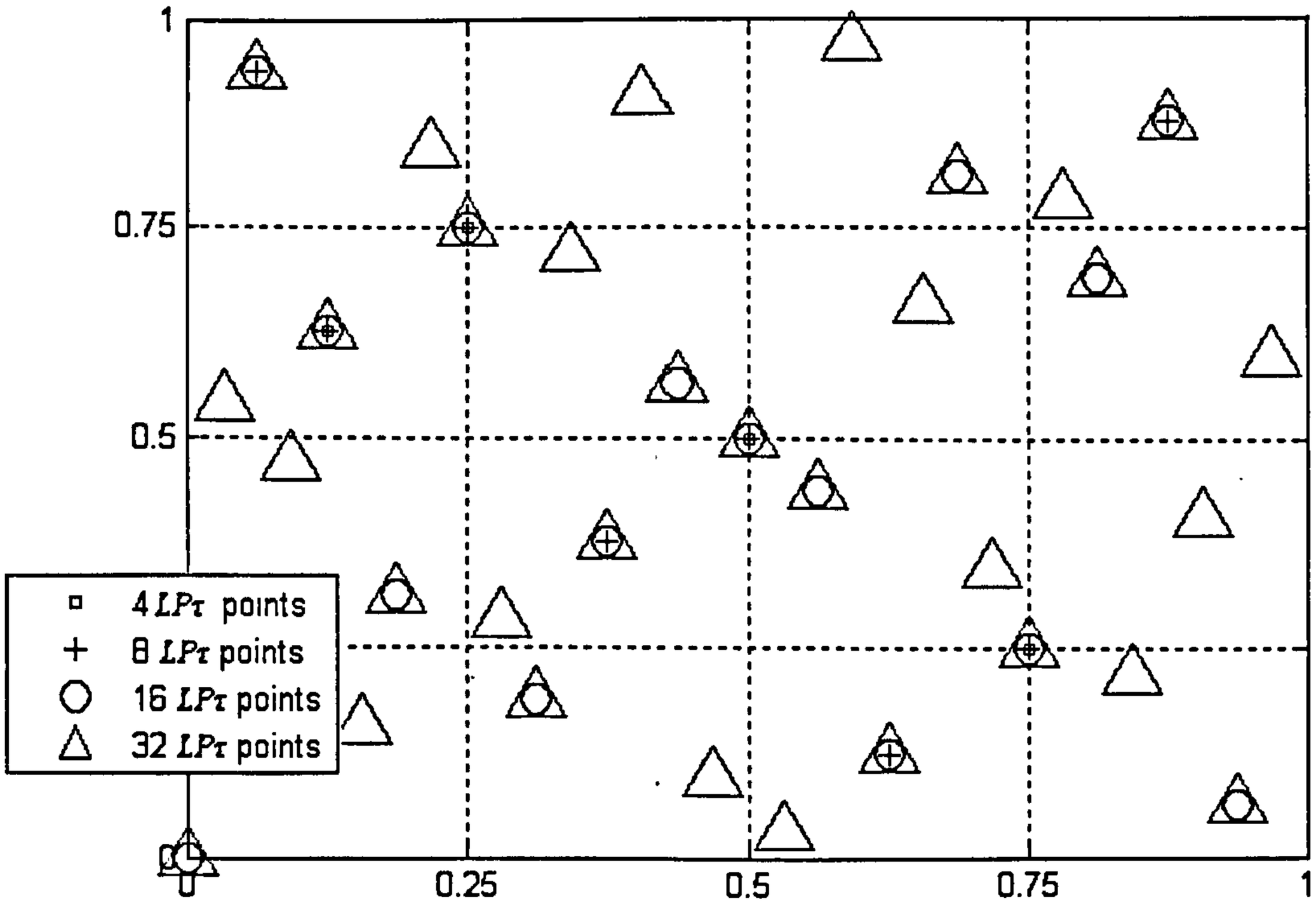


Figure 3.1. *Fill in the gaps* property of the *LPτ* sequences.

A range of allowed values (N_{min} and N_{max}) is defined and the technique adaptively selects the smallest allowed value that gives enough information about the landscape of the objective function, so the algorithm can continue the search effectively. Simply said, after the minimal possible number is selected, the function at hand is investigated with those points, and if there are not *enough promising points*, additional ones are generated and the process is repeated until an *appropriate* number of points is selected or the maximal of the allowed allowed values is reached. The algorithm is illustrated graphically in Fig. 3.2, where the following steps are considered:

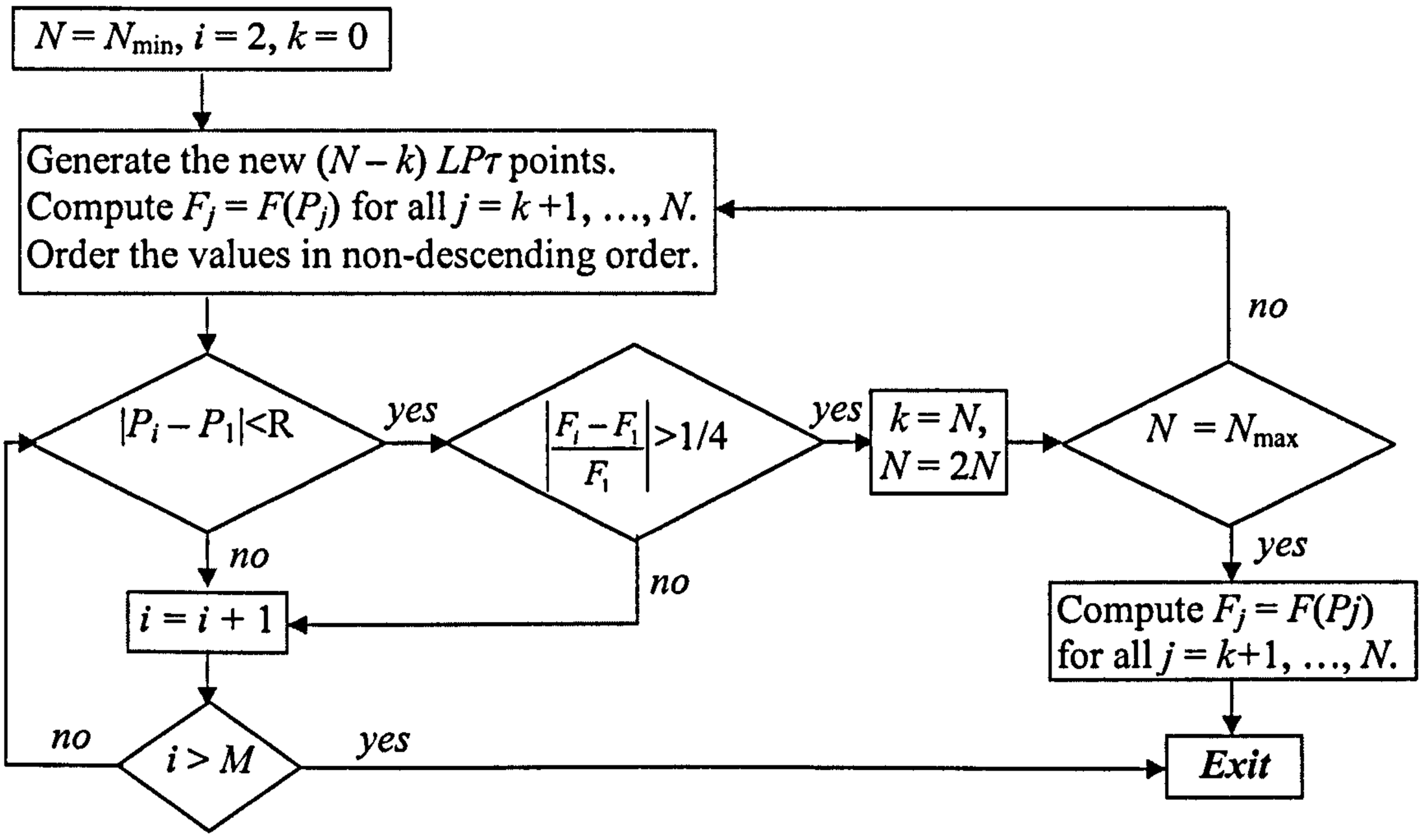


Figure 3.2. The algorithm for adaptive selection of the number of initial points N within the user-defined bounds $[N_{min}, N_{max}]$.

Let $M = 2^{n-1}$ (or another user-defined value), where n is the dimensionality of the problem.

Step 1. Generate $N = N_{min}$ $LP\tau$ points and evaluate the objective function for each of them;

Step 2. Keep the function values in non-descending order and keep a record of their corresponding points. Consider the first point – P_1 (which provides the *best* function value);

Step 3. Compute a number R, which is a function of the volume V of the whole searched space, the number of points N, and the dimensionality of the problem n. It is assumed R to be the average Euclidean distance between any two *neighbouring* points from the initial population (will be further discussed);

Step 4. Consider the next point P_2, compute the Euclidean distance to P_1 and denote it by r. If $r > R$ and

$$\left|\frac{F(P_2)-F(P_1)}{F(P_1)}\right| > \frac{1}{4} \tag{3.1}$$

double the number of testing points ($N = 2N$), and if the maximum number of points allowed (N_{max}) is not reached, repeat from Step 2. If $N = N_{max}$, then exit. If no change of the number of points is evoked, repeat step 4 for the next points P_3 , …, P_M. If the first M points are already considered, then exit. Here the heuristic threshold value ¼ was selected as optimal after several trials with similar values.

Discussion 3.2

When sorted in non-descending order, the first M points correspond to the *best* function values. The first one of them, P_1, corresponds to the current minimal value found by the algorithm. If for all these M points the condition (3.1) is satisfied, then there is no need to generate more initial testing points and the algorithm can proceed to the next step (all M points are *good* and there is enough information to proceed). If for any of those points – say $\overline{P}$, the inequality (3.1) is not satisfied, this would mean that the algorithm is unable to find M points corresponding to function values that are close to the current minimum. Then two cases are considered:

If $\overline{P}$ is *distant* from P_1 (see also Note 3.2.1), the algorithm starts from the beginning and the number of initial points is doubled.

If $\overline{P}$ is *close* to P_1, the algorithm proceeds to the next point without taking any action, since $\overline{P}$ is considered to be in the same region of interest as P_1 and, thus, does not bring any *new* information.

After executing the algorithm, we either have the best M points in one region of attraction, with no better point, generated outside, or the best M points have similar values in different regions and the search should be continued in neighbourhoods of each of them.

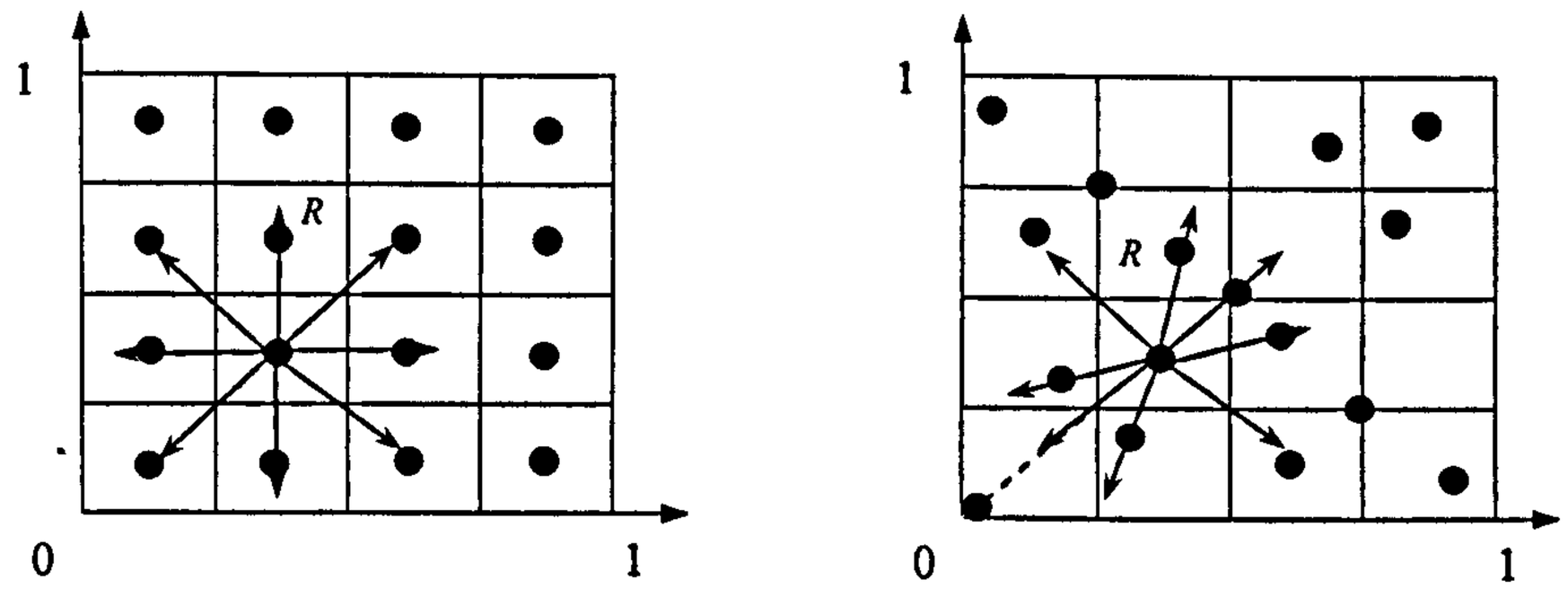
It may also be useful to note, that although the proposed method has shown satisfactory results for the tested functions (see Section 3), for general problems we recommend to start the algorithm again with as many points as possible. Choosing the number of initial test points is a matter of trade-off between the computational cost and the probability of reaching a GM.

The total number N is doubled every time an "inappropriate" point from the first M is found. The procedure is repeated again for the next generation of points. This means all M points are treated equally and are able to cause an increase of N.

The parameter R is chosen to be

$$R = \sqrt{n}\sqrt[n]{V/N}. \tag{3.2}$$

Virtually, the volume V of the space of interest is subdivided into N equi-volumed cubes (V/N) and the diagonal of such a cube is taken to be R. We consider R as the average distance between any two *neighbouring* points, where *neighbour* means the points are situated in two neighbouring virtual hyper-cubes. We use the parameter R to measure whether two points are close or distant – they are considered to be close if the distance between them is smaller than R.



(a) Cubic sequence. (b) $LP\tau$ sequence.

Figure 3.3. R is approximately the average distance between any two neighbouring points.

Obviously, other metrics could be introduced and used as well, but in our case, the proposed metric produced satisfactory results for the $LP\tau$ sequences used. For example, if n=2, V=1, and N=16, then it follows that R = 0.353553, which is also appropriate for a cubic sequence as it can be seen from Fig. 3.3(a).

Selection of the hyper-cubes of interest and their size

During the search, we need to choose a number of promising points and continue the search in regions around them. The best point is guaranteed to be chosen and not more than M points could be chosen to become "core" points of regions of interest, thus, $M_{new} \in [1, M]$. We consider the points P_i for each $i = 2, \ldots, M$: a point would be of further interest (new points to be generated in its neighbourhood) only if its function value does not increase significantly (condition (3.1) is not satisfied), i.e., if it is *good enough*. The side a of a surrounding hyper-cube is established depending on the distance and difference of function values of P_i and P_1. Figure 3.4 illustrates the following algorithm:

step 1. For each point P_i, ($i = 2, \ldots, M$) we establish a relation with P_1, namely if it is close (or distant) and if the function value does (or does not) increase significantly accordingly to (3.1);

step 2. If the two points are close:

if $F(P_i)$ grows significantly, discard P_i, and take the side of the hyper-cube, corresponding to P_1, to be $a_1 = C_1R$, where $C_1 = C_1(n)<1$;

if $F(P_i)$ does not grow significantly, search in a neighbourhood of both points with a small size of $a_i = C_2R$ and $a_1 = C_1R$, where $C_2 = C_1/2$;

step 3. If the two points are distant:

if $F(P_i)$ grows significantly, discard P_i and take the side of the hyper-cube, corresponding to P_1 to be $a_1 = C_3R$, where $C_3 = 2C_1$, and is considered to be relatively large;

if $F(P_i)$ does not grow significantly, search in a neighbourhood of both points in hyper-cubes with greater sides $a_i = C_4R$, where $C_4 = 1.5\ C_1$;

Discussion 3.3

The metric defining two points to be close (or distant) is getting more complicated after the first iteration. The number R is no longer relevant, since all considered points could be situated in different hyper-cubes (regions of attraction). That is why for each hyper-cube of interest H_i, we introduce a local metric R_i. If $\overline{N}_i$, $i = 1, \ldots, M_{new}$ is the number of $LP\tau$ points, generated in each hyper-cube H_i with a core point P_i and V_i is its volume, then in correspondence with the formula 3.2

$$R_i = \sqrt{n}\sqrt[n]{V_i / \overline{N}_i}. \qquad (3.3)$$

Each point that has a greater than R_1 distance to P_1, is considered as distant (it still could be situated in the same region of interest as P_1). Afterwards, the size of each H_i is evaluated with the use of the local R_i (for instance, $a_1 = C_1 R_i$ in step 2).

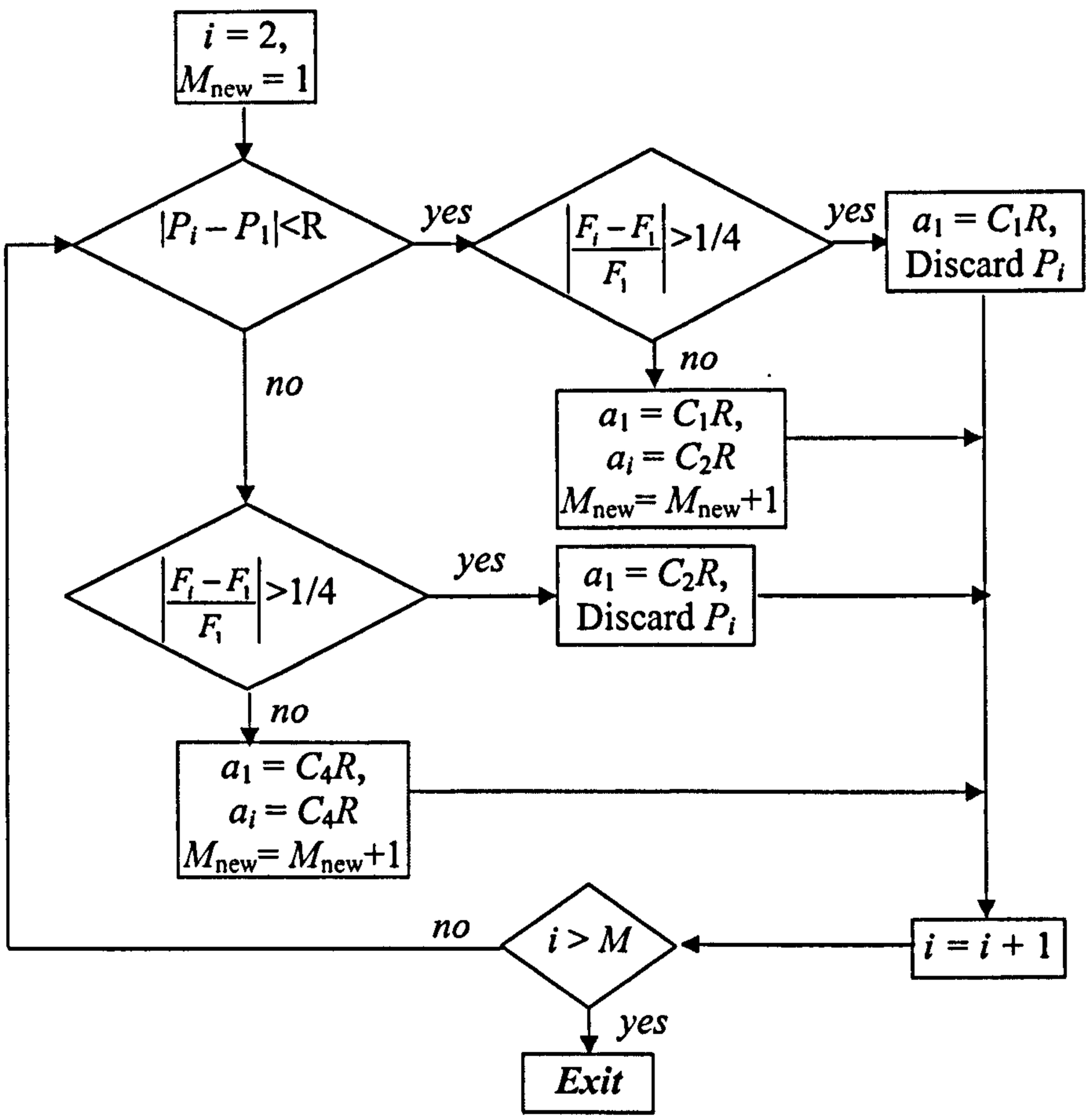


Figure 3.4. Algorithm for adaptive selection of points of future interest (in total $M_{new} \leq M$) and the sides of their surrounding hyper-cubes (a_i, $i = 1, \ldots, M_{new}$). The first (best) point is always selected.

As it could be seen, the size a_i of each region of interest depends on several objectives:

1. the stage of the search (a_i decreases proportionally on each iteration, because the metric R_i does);
2. the metric R_i is given by formulas (3.2) and (3.3) and depends on the number of points generated in the hyper-cube of interest ($\bar{N}_i$), the size of it (V_i) and the dimensionality n;
3. The constant C_1 is indeed the controlling variable of the convergence speed. It is always smaller than one, in order to guarantee decreasing sizes of a_i and the algorithm convergence. Its choice would determine the trade off between exploration and exploitation;
4. We recommend and have used in our tests $C_1=1/2\sqrt{n}$ for a very clear reason: in the general case R_i is given by formula (3.3) and represents the average difference between any two trial points in the region of interest H_i. Then H_i is divided to $\bar{N}_i$ smaller hyper-cubes and, we assume that each of them contains one trial point in its center (this assumptions is not true in general, but gives us an *approximation* of the real situation and is provided by the *good* uniformity of the *LPτ* sequence) as demonstrated in Fig. 3.3. The side of each small hyper-cube is given by $\sqrt[n]{V_i / \bar{N}_i}$. Therefore, when searching around one of the points, it is not necessary to go out of its own surrounding hyper-cube, because there is another trial point in the next one. If the neighbouring hyper-cube is promising, it would be investigated separately. Therefore, the side a_i of the future hyper-cube of interest should not be bigger than $\sqrt[n]{V_i / \bar{N}_i}$. Taking in mind everything said, if $C_1=1/2\sqrt{n}$, this would ensure that the largest side possible is in step 3 and it is $a_1 = C_3R = 2C_1R = \sqrt[n]{V_i / \bar{N}_i}$. All other sides are smaller and are adjusted according to the rules in step 2 and step 3.

We refer to Fig. 3.5 where the four basic cases are shown. We consider the relation between the current best point P_1 and the other points in terms of distance between them and difference in their function values.

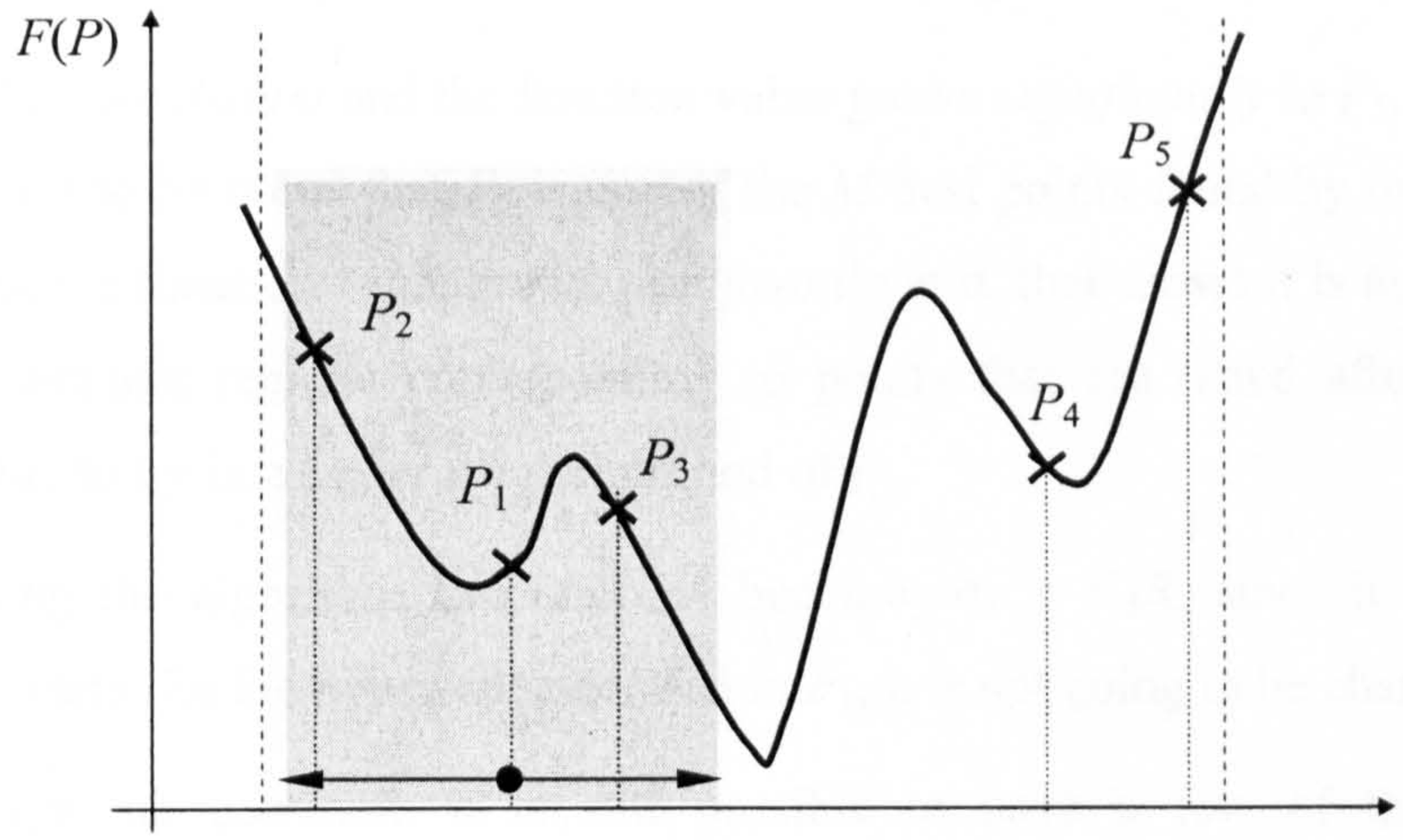


Figure 3.5. Extracting information about the function behaviour in the interval $[a, b]$.

Two points are said to be *distant* (*close*) if the Euclidean distance between them is greater (*less*) than R. A function value is considered to be *similar* (*differ significantly*) with the current best point's one if condition (3.1) holds (does not hold).

Let us consider Fig. 3.5, where instance points are shown.

P_1 and P_2 are *close* but the function value *grows significantly* in P_2. The search is not continued in the neighbourhood of P_2, but the relation between them shows that P_1 is situated in a steep region and it is reasonable to take only a *small* neighbourhood of P_1 for further search (it could be considered as process of extracting *gradient* information, that shows that the function value increases as small perturbation of the point is made).

P_1 and P_3 are *close* and the function values are *similar*. Since P_3 is also a promising point, the search is going to continue the search in both neighbourhoods. It is reasonable to use *small* neighbourhoods for the future search in order to examine closely the behaviour of the function around the core points and detect if there are two minima corresponding to them (this is the case in Fig. 3.5), or if they are in the same basin of attraction.

P_1 and P_4 are *distant* and the function values are *similar*. The search is going to continue the search in the neighbourhoods of both points, but in this case it is

reasonable to investigate a larger domain around each of them, since we want to find in which of the two regions the function is decreasing more.

P_1 and P_5 are *distant* and the function value grows *significantly* in P_5. It is important in this case to be noted that P_5 is one of the M best points found by the algorithm so far. Since the function value grows *significantly* in it, that shows it is not likely to find other promising regions corresponding to points that are listed after P_5 and it is reasonable to try in a larger neighbourhood of P_1.

If during the algorithm execution a_1 becomes $a_1 = C_3R$, since it is the greatest possible value for the hyper-cube centred in P_1, it is not going to be changed anymore.

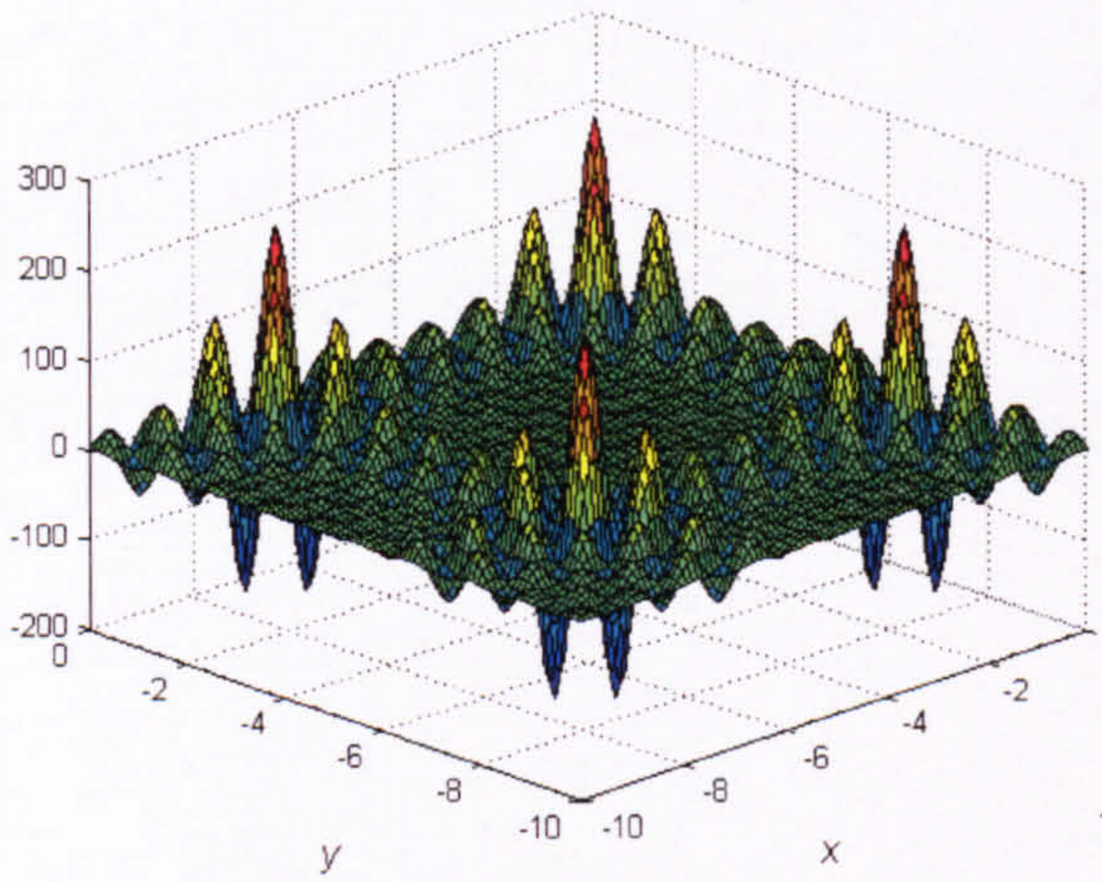
Although not common, it is still possible to have a few of the hyper-cubes intersected in some area and the same point could be generated more than once. This would detriment the algorithm and for this purpose we have a function that disposes the duplicated points (for the test functions considered, this never actually happened).

Choice of the number of points to be generated in each hyper-cube

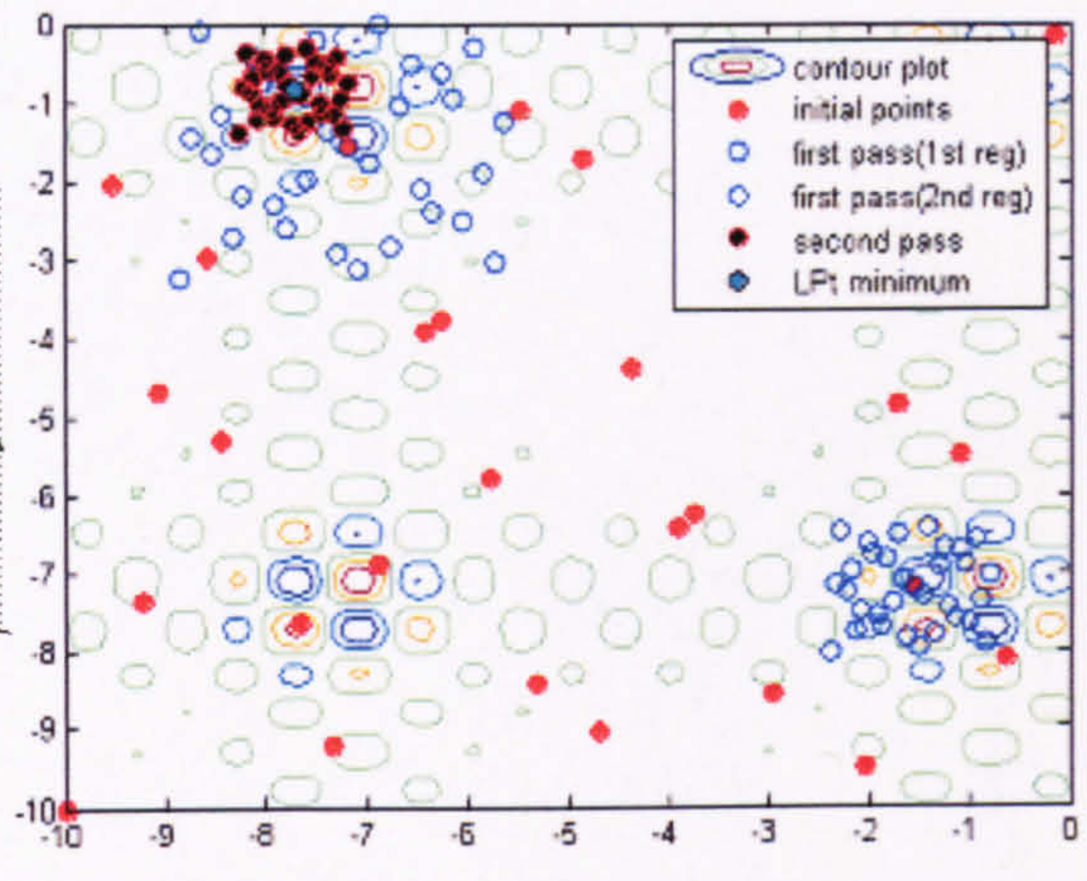
It is reasonable to have more search points in a hyper-cube in which the function is changing rapidly (a greater function change corresponding to a small space deviation), i.e., if it has steep slopes, and vice versa - less points if the function is smooth. Therefore, a slightly modified version of the algorithm for choosing the number of initial points is used here. It is described in subsection "*Generating the initial points*", whilst it is applied only in the hyper-cube of interest instead in the whole search space. The algorithm for choosing an appropriate number of points uses information about the *ruggedness* of the function at hand in order to select the appropriate number of points. The tests on benchmark problems (see Appendix A), considered in Section 3.1.4, show that the algorithm handles this issue very well. For all tested functions (see Section 3.1.4), this algorithm appears to be efficient in choosing the number of trial points in each small hyper-cube.

For instance, in the case of *Shubert* function (Fig. 3.6(a)), once a region of attraction for the global minimum is detected during the first pass, 32 points are generated in it, since the minimum is in a relatively steep region (Fig. 3.6(b)). On the contrary, for the *Branin function* (Fig. 3.7(a)), once the region of attraction is localised during the first pass, only 16 points are generated in it, since the function is relatively flat there (Fig. 3.7(b)). The number of these points is automatically chosen by the *LPτO* method for

each test function. Fig. 3.6(b) and Fig. 3.7(b) do not show the whole search domain, but only the area where a GM was ultimately found.

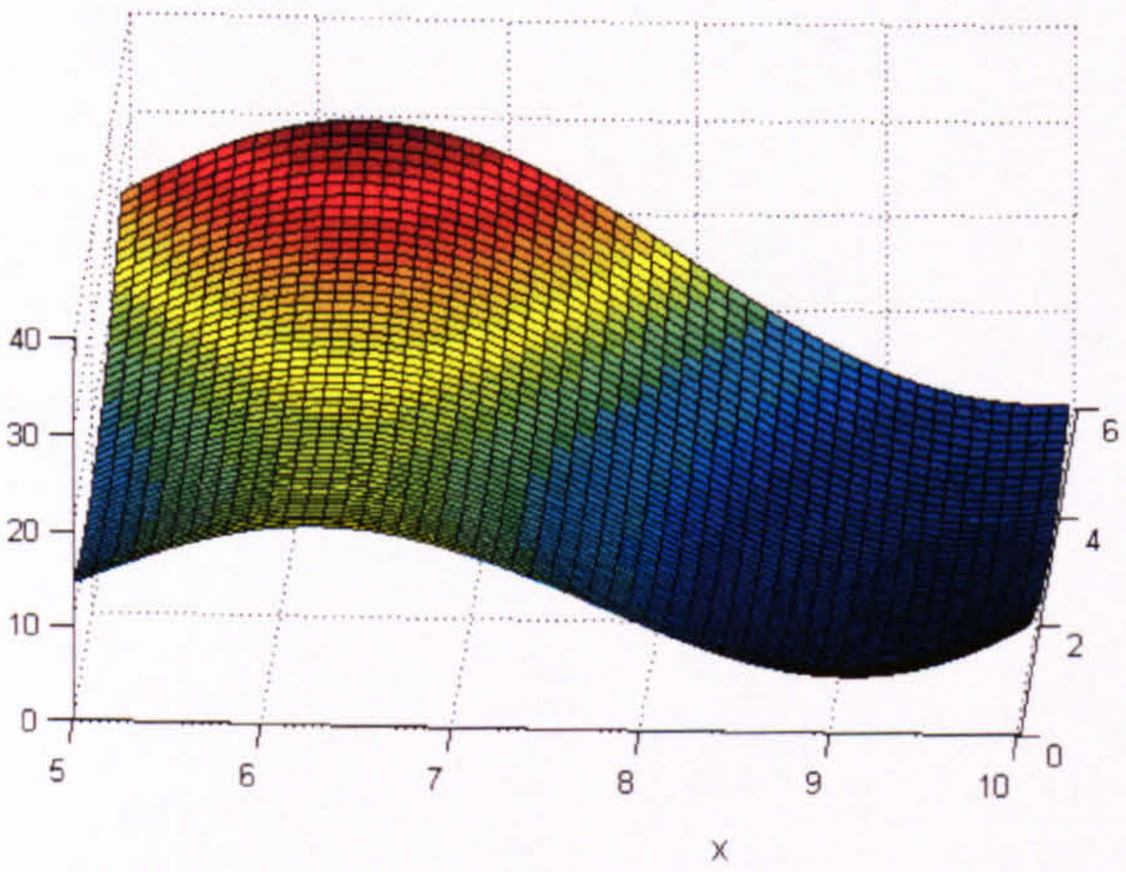


(a) *Shubert* function.

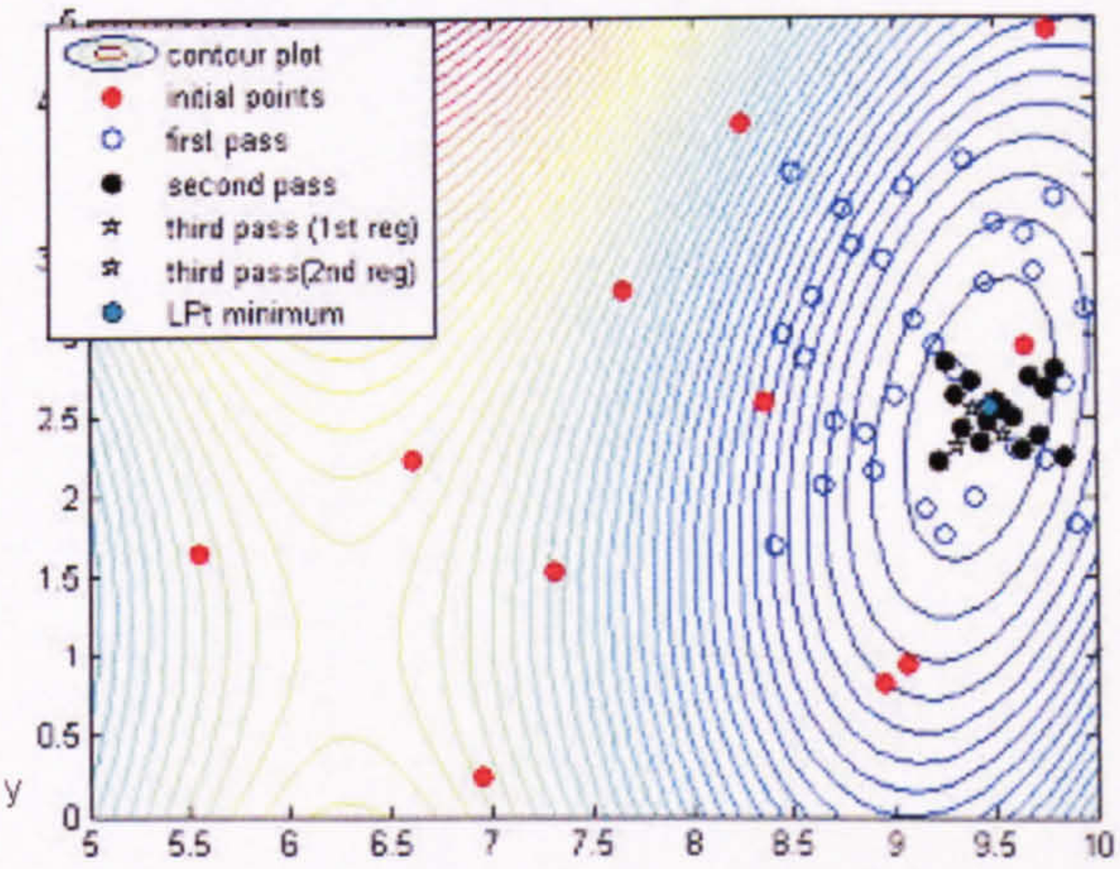


(b) Minimization process.

Figure 3.6. *Shubert* function optimised by *LPτO*.



(a) *Branin* function.



(b) Minimization process.

Figure 3.7. *Branin* function optimised by *LPτO*.

3.1.3 Properties

The investigated *LPτO* technique is *consistent*, since at each iteration not worse solution is accepted. The algorithm does not use any gradient information and could be applied to the minimisation even of functions which are dis-continuous.

LPτO converges to a GM only with a probabilistic guarantee. This means that as the number of testing points increases, the probability of finding a GM also increases. This property is often found in many stochastic methods (Törn and Žilinskas, 1989; Arora *et al.*, 1995; Oblow, 2001, Ali *et al.*, 2005).

Theoretically, the convergence of *LPτO* is established as follows: the initial N testing points are uniformly distributed, and therefore, they form a dense set over the search region Π (Niederreiter, 1981). From Theorem 1.3 in Törn and Žilinskas (1989), stating that a GO method is convergent to a GM of any continuous function if and only if the sequence of trial points of the algorithm is everywhere dense in the compact search region Π, directly follows that *LPτO* converges to a GM as $N \to \infty$. In practice, we would have probability of reaching a GM that is less than 1, but tends to 1, as the number of initial points increases. The parameters adopted in the algorithm aim to find optimal balance between accuracy and computational cost of the solution.

We consider the property of *LPτ* sequences discussed in Section 2.3.2, that for values of $N = 2^k$, $k = 1, 2, \ldots, 31$, the discrepancy ρ (defined by (2.12)) converges to zero with rate $O(N^{-1}\log^{n-1}(N))$, as the number of points increases. From the discussion in Section 2.3.2 it is clear that as $\rho \to 0$, with the increase of N, the better convergence rate of the discrepancy would insure better convergence rate of the dispersion and better convergence of the *LPτO* algorithm.

3.1.4 Initial results from testing *LPτO*

Some initial testing of *LPτO* was presented at the 2005 *IADAT* conference in Spain (Jordanov and Georgieva, 2005). It includes testing on a few benchmark GO functions, as well as on several NN training problems for classification of well known test sets.

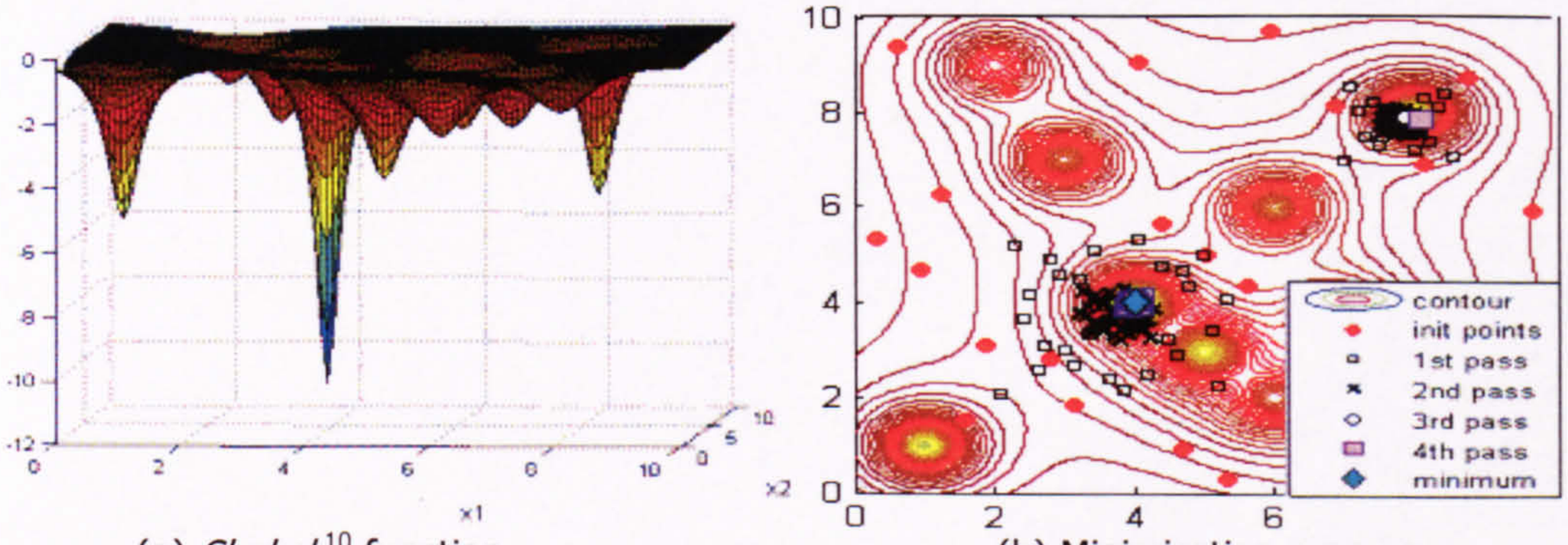
Tests on multi-modal mathematical functions

LPτO was tested on several multi-modal mathematical benchmark functions of two to four dimensions. Results, compared with other algorithms, in terms of function evaluations, are given in Table 3.1. For all test functions, the *LPτO* algorithm found the GMs with better than 0.5% accuracy. The first four optimisation methods, listed in Table 3.1, are deterministic and are discussed in Section 2.2.1. As stated there, they need smaller number of function evaluations, for the expense of other auxiliary

computations. It can be seen from the table that *LPτO* outperforms these methods only for 2 of the test functions and is not so efficient for the rest of them. The other four methods (ESA, TS-QN, SCTS, and ECTS) are heuristic and use stochastic rules, therefore, they could be considered as being from the same group as *LPτO*. TS-QN is a hybrid method combining heuristic Tabu Search with deterministic quasi-Newton local search. *LPτO* clearly outperforms the other heuristic techniques, including SA and TS modifications. However, all methods differ widely in their stopping criteria and fair comparison of their performances is hardly possible. Nevertheless, the obtained results show that the investigated method is competitive for the tests performed.

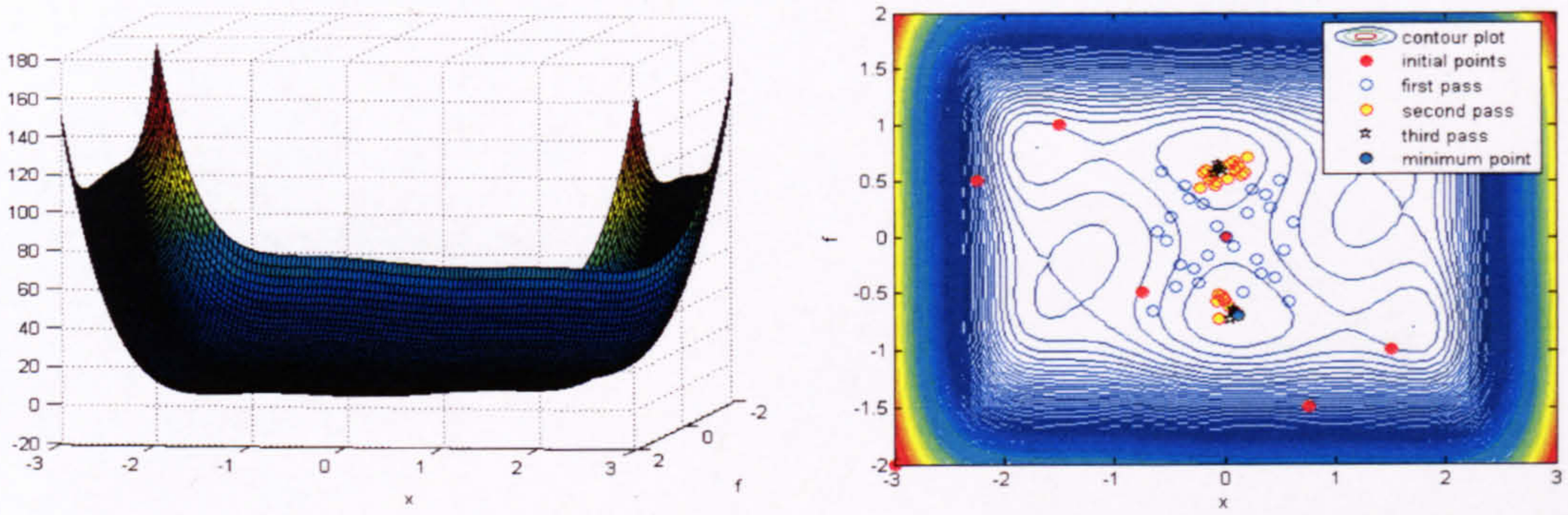
For demonstrative purposes, Fig. 3.8, Fig. 3.9, and Fig. 3.10 show successful *LPτO* minimisation processes for three of the testing multi-modal functions (*Shekel*, *Six-Hump*, and *Rastrigin*).

Fig. 3.10 illustrates the way $LP_{\tau}O$ works in the case of two-dimensional *Rastrigin function*. For better clarity, a smaller search interval than the actual one used for testing is considered. There are four local minima in the investigated interval shown in Fig. 3.10(a), one of which is also a global. One can see from the figure that after the initial *seeding*, four regions of attraction are detected and more testing points are seeded in all of them during the first pass. Subsequently, one of the regions is left out as less promising, and after the second pass, another two of the regions are left out, before the search is concentrated into the last region, where finally a GM is found. A zoomed picture of the search in the last region (where a GM is found) is shown in Fig. 3.10(b).



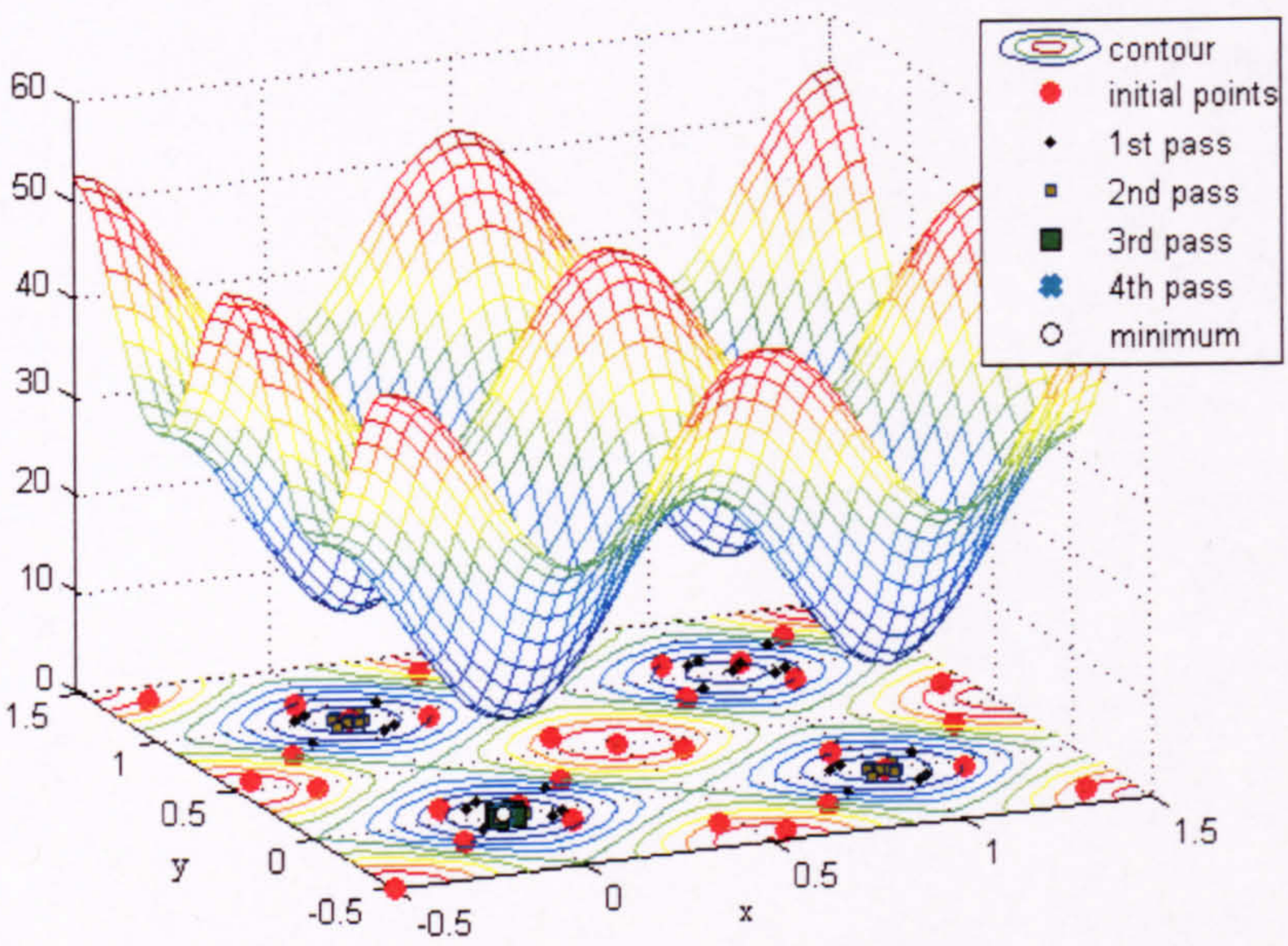
(a) *Shekel* ^{10} function. (b) Minimization process.

Figure 3.8. *Shekel* ^{10} function optimised by *LPτO*.

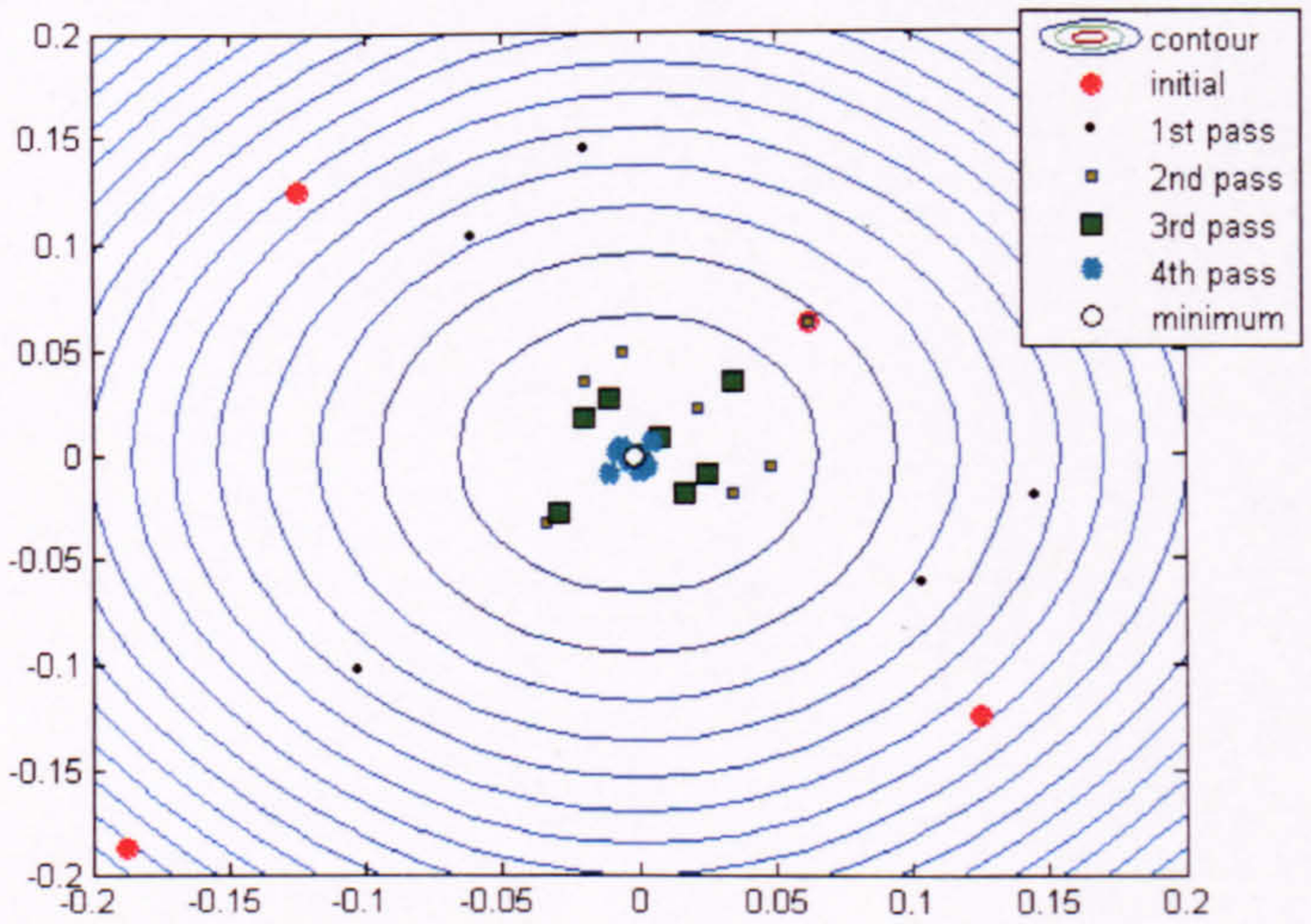


(a) *Six-Hump Camelback* function. (b) Minimization process.

Figure 3.9. *Six-Hump Camelback* function optimised by *LPτO*.



(a) 2-dimensional *Rastrigin* function optimised by *LPτO*.



(b) Zoomed region of attraction.

Fig. 3.10. *Rastrigin* function optimisation by *LPτO*.

Table 3.1. Comparison of the performance of *LPτO* with other techniques in terms of the number of function evaluations for reaching a GM.

F / Method	Sh	SxH	Ra	GP	Br	Hr3	2Dim	$Shkl^{10}$
DIRECT	2967	285	-	191	195	199	-	145
MCS	48	44	-	194	57	128	-	330
TRUST	72	31	59	103	55	58	38	-
SPT	150	26	140	123	67	75	-	-
ESA	-	-	-	783	-	698	-	1363
TS-QN	355	-	-	301	-	386	-	-
SCTS	521	-	-	696	491	691	-	-
ECTS	370	-	-	231	245	548	-	-
LPτO	241	65	155	79	192	163	101	76

Methods: DIRECT – Jones *et al.* (1993); MCS – Multilevel Coordinate Search of Huyer and Neumaier (1999); TRUST - Terminal Repeller Unconstrained Subenergy Tunneling by Barhen *et al.* (1997); SPT – Stochastic Pijavskij Tunneling by Oblow (2001); ESA – Enchansed Simmulated Annealing by Siarry *et al.* (1997); TS-QN – Tabu Search combined with Quasi-Newton local search by Teh and Rangaiah (2003); SCTS – Staged Continuous Tabu Search by Zheng *et al.* (2005); ECTS – Enhanced Continuous Tabu Search by Chelouah and Siarry (2000).

Test Functions (*n*): Sh – *Shubert* (2); SxH – *Six-hump Camelback* (2); Ra – *Rastrigin* (2); GP – *Goldstein-Price* (2); Br – *Branin* (2); Hr3 – *Hartman* (3); 2Dim – *N*-Dimensional Test Function (2); $Shkl^{10}$ – *Shekel*^{10} (4). Formulas and illustrations could be found in Appendix A.

Results from Neural Network training with LPτO

The investigated NN architectures are with one hidden layer, static topology in which only the adjacent layers are fully connected and have standard *Sigmoid* transfer functions. We considered batch-mode training, where, for a given experiment with P learning samples the *Mean Squared Error* (MSE) function is given by formula (2.7).

Each unit i in layer l, $0 \le l \le L$, is indexed as i_l, $i_l = 1,2, \ldots, u_l$, where u_l is the number of units in that layer and L is the number of layers. The input and target patterns are denoted with $\bar{x}^p$ and $\bar{t}^p$, $p = 1,2,\ldots, P$, respectively. For a given pattern p, the activation of a unit i from the l-th layer, is $a_i^p = g(\bar{w}_i, \bar{x}_{l-1}^p) = \sum_{j=0}^{u_{l-1}} w_{i,j} x_j^p$, where $w_{i,j}$ is the weight associated with the connection between unit j from the previous layer and unit i from the l-th layer. The bias is $x_0^p = 1$ with corresponding bias weight $w_{i,0}$. The transfer function is the standard *Sigmoid* one given by (2.4), and the error

function for each sample is given by (2.6). The network weight vector is n-dimensional real Euclidean vector $\overline{W}$, whose components are the weights of the network. The *LPτO* Global Optimisation algorithm is applied to minimise function (2.7) and to perform optimal training. The proposed algorithm is tested on three simple benchmark problems with different dimensionalities. The first two are the XOR and 4-Parity problems discussed in Section 1.2.2. The third one is a real-world problem – a dataset from the UCI Machine Learning repository (http://mlearn.ics.uci.edu/MLRepository.html). For comparison, a BP (Levenberg-Marquardt) is also performed using Matlab7.0.1 Neural Network Toolbox. Since the standard BP is strongly dependent on the initial weights, each test is performed 50 times and the average results are reported.

- Classification of XOR problem

The XOR problem is considered in detail in Section 1.2.2. Here, NN architecture 2-2-1 (two inputs, two hidden units, one output, and biases) is adopted with a bias in the input and hidden layers, which defines $n = 9$ dimensional optimisation problem. The obtained optimal results are given in Table 3.2. The NN is tested with 40 examples of noisy data, where the noise is up to 15% (e.g., (0.15,0.85) and (0.3, 1.0) are both samples with 15% noise of the training sample (0.0, 1.0)).

Table 3.2. Optimal errors for the *LPτO* and the BP (XOR problem).

Criterion / Method	Worst Train Error	Worst Test Error	Average Test Error
LPτO	0.0007	0.0281	0.0016
BP	0.2533	0.3429	0.2313

Criterion: Worst Train Error – from all independent runs; Worst Test Error – from all independent runs.

Methods: BP – Backpropagation (Levenberg-Marquart); *LPτO* – proposed here.

- Classification of *N-Parity* problem

This problem is also considered in Section 1.2.2. For the case with $N = 4$, we adopted network architecture of 4-4-1 (four inputs, four hidden units, one output, and bias). It contains 25 connection weights and thus defines $n = 25$ dimensional learning problem. There are $P = 16$ input-target patterns for the training set.

The obtained training and testing (with noisy input) errors are given in Table 3.3. The trained NN is tested with 50 examples of noisy data where the noise is up to 10%.

Table 3.3. Optimal errors for the *LPτO* and the BP (4-*Parity* problem).

Criterion / Method	Worst Train Error	Worst Test Error	Average Test Error
LPτO	0.0012	0.0736	0.0052
BP	0.0441	0.1471	0.0762

Criterion: Worst Train Error – from all independent runs; Worst Test Error – from all independent runs.
Methods: BP – Backpropagation (Levenberg-Marquart); *LPτO* – proposed here.

- Classification of Lenses problem

This problem (from the UCI repository, discussed in Section 2.1.4) has three categories of patients to be classified: whether to fit a patient with hard, soft or no contact lenses at all. A network with 4-3-1 architecture is designed in order to produce continuous output with the three classes coded as 0.9 (hard), 0.5 (soft), and 0.1 (neither). This network defines 19-dimensional optimisation problem (n = 19). The dataset is split into two sets – one for training and one for testing, with 12 instances in each, having in both nearly the same class distribution, as in the original set. The obtained optimal solutions with the training and testing errors are given in Table 3.4.

Table 3.4. Optimal errors for the *LPτO* and the BP (*Lenses* problem).

Criterion / Method	Worst Train Error	Average Train Error	Worst Test Error	Average Test Error
LPτO	0.0075	0.0028	0.8097	0.1714
BP	0.0207	0.0082	0.6802	0.1755

Criterion: Worst Train Error – from all independent runs; Worst Test Error – from all independent runs.
Methods: BP – Backpropagation (Levenberg-Marquart); *LPτO* – proposed here.

Discussion

Result from the NN training show that *LPτO* outperforms BP, obtaining better training and testing results for the XOR and 4-*Parity* problems. For the *Lenses* task, the *LPτO* training error is much lower than the BP one, however, the testing results are very similar. This might be due to the small set of training data (only 12 samples),

and the nature of the data itself. The training set is obviously not capable of providing enough information for the NN to learn *generalising* well. In such cases, the cross-validation technique, as described in Section 2.1, might be of use.

3.2 *LPτO* Hybridised with a Local Search to form *LPτNM*

In order to improve the accuracy of the solution found by *LPτO*, we employed the Nelder-Mead (*NM*) Simplex Local Search (Nelder and Mead, 1965). *NM* is applied successively, after *LPτO* is executed, to refine the obtained solution and provide better accuracy. Therefore, we called the hybrid method *LPτNM*. In the next sections, the novel method is proposed briefly and is tested extensively on a number of continuous functions and NN training tasks.

3.2.1 The *NM* and *LPτNM* Method

The Nelder-Mead (*NM*) simplex method for function optimisation is a fast local search technique (Nelder and Mead, 1965) that makes use only of function values and requires continuity of the function, but no smoothness, Lipschitz continuity, etc. It has been used in numerous hybrid methods to refine the obtained solution (Chelouah and Siarry, 2003; 2005), and for coding of GA individuals (Hedar and Fukushima, 2003).

Let us consider Euclidean space $\mathbb{R}^n$. A *simplex* is defined as a convex set $S \in \mathbb{R}^n$, defined as the convex hull (the convex polytope) of the vertices P_0, P_1,...,P_n, ($P_i \in \mathbb{R}^n$, $i = 0,\ldots, n$) which are affinely independent. This means that none of them can be presented as affine combination of the others, i.e., a combination of the type:

$$P_i = a_0P_0 + \ldots + a_{i-1}P_{i-1} + a_{i+1}P_{i+1} + \ldots + a_nP_n,$$

where $a_i \in \mathbb{R}$, $j = 1, \ldots, n$, and $a_0 + \ldots + a_{i-1} + a_{i+1} + \ldots + a_n = 1$. In the simplex minimisation technique the function values are calculated only in the vertices of the simplex. During each iteration, the worst vertex (let us say P_w, $w \in [0, n]$) is replaced by a better point. Changes of the simplex are made by three basic rules: *reflection, expansion* and *contraction*. These operations are defined as:

Let $\overline{P}_w = \sum_{i=0}^{n} P_i$, for $i \neq w$ is the centroid of the other vertices, different than P_w.

Then *reflection* is defined as: $P_r = (1+\alpha)\overline{P}_w - \alpha P_w, \alpha > 0,$

expansion as: $P_e = \overline{P}_w + \gamma(P_r - \overline{P}_w),\ \gamma > 1,$

and *contraction* as: $P_c = \beta P_w + (1 - \beta)\ \overline{P}_w, \beta \in (0, 1).$

The algorithm itself is given in detail in Nelder and Mead (1965). For the parameters (α, β, γ) we adopt the values (1, 0.5, 2), which has been recommended by the authors as one of the best combinations. The speed of convergence (measured by the number of function evaluations) depends on them, but mostly, it depends on the choice of the initial simplex - its coordinates, form and size. We chose the initial simplex to have one vertex in the best point found by the $LP_\tau O$ search and another n vertices distanced from it in positive direction along each of its n coordinates with coefficient λ

We consider the optimisation procedure to be successful if the following inequality is satisfied:

$$\frac{1}{n}\sum_{i=0}^{n}(\|P_i\| - \|\overline{P}_\omega\|) < \epsilon, \qquad (3.4)$$

where $\|.\|$ stands for the Euclidean norm and ϵ is a small number that implies the accuracy of the solution - in our tests (see Section 3.2.2) $\epsilon = 10^{-4}$ is used. This stopping condition shows how small has the simplex become in a sense how far the vertices are from the centroid $\overline{P}_w$.

As for the choice of parameter λ, we connected it with the value of R_1, which corresponds to the average distance between the testing points in the region of attraction, where the best solution is found by *LPτO*. After performing a number of tests on all functions, we chose $\lambda = 1.5R_1$. Different choices of λ could decrease the number of function evaluations for a particular function. In our case, this choice has shown good results for all testing functions.

The *consistency* and *convergence* properties of the novel *LPτO* method (Section 3.1.3) and the *NM* Simplex Search define the consistency and convergence of the proposed hybrid method *LPτNM*.

3.2.2 Results from Testing *LPτNM* on GO Benchmark Functions

The preliminary proposal and results of *LPτO* were reported in an international conference (Georgieva and Jordanov, 2005a). The detailed proposal of *LPτNM*, as well as the numerical results from testing on a number of benchmark functions, were submitted for publication in the *Journal of Global Optimisation* (Georgieva and Jordanov, 2005b).

The proposed *LPτNM* method is tested on 18 benchmark multimodal mathematical functions of 2 to 20 variables. Table 3.5 shows the user-defined parameter values adopted for the tests without individual tuning for each test function. Naturally, as the dimensionality of the problems increases, the parameter values increase as well, allowing more trial points to be generated. These values are heuristically chosen after performing tests with several appropriate values.

Table 3.5. User-defined parameter values adopted in the numerical tests reported in Table 3.8 and Table 3.9.

Parameter / Dimension	M	$[N_{\min}, N_{\max}]$	$[\bar{N}^l_{\min}, \bar{N}^l_{\max}]$
$n = 2,3,5,6$	4	$[2^{n+1}, 2^{n+4}]$	$[2^n, 2^{n+3}]$
$n = 10$	15	$[2^9, 2^{10}]$	$[2^5, 2^9]$
$n = 20$	20	$[2^{11}, 2^{13}]$	$[2^6, 2^{10}]$

Parameters and cases: M – maximal future interest regions allowed on each iteration; $[N_{\min}, N_{\max}]$ – lower and upper bounds for the number of initial points N; $[\bar{N}^l_{\min}, \bar{N}^l_{\max}]$ – lower and upper bounds for the number of points to be generated in each region of interest through the iterations; n – dimensionality of the problem at hand.

Table 3.6. *Goldstein-Price* function (n = 2) optimised by *LPτNM* with different values for M *and* $N_{\max}$. All other parameter values are given in Table 3.5.

M / $N_{\max}$	2	3	4	6	8
2^{n+3}	156 (72)	156 (84)	159 (84)	161 (84)	168 (85)
$\mathbf{2^{n+4}}$	184 (95)	184 (99)	**187 (100)**	191 (100)	193 (100)
2^{n+6}	354 (84)	365 (96)	366 (98)	368 (100)	375 (100)
2^{n+8}	1089 (100)	1096 (100)	1098 (100)	1106 (99)	1111 (100)

The numbers in this stable present the number of function evaluations needed for the optimisation and in brackets gives the success rate in % (out of 100 runs for each test). The numbers in bold show the parameters combination selected in Table 3.5 and used in the rest of the tests (Table 3.8 and Table 3.9).

M – maximal future interest regions allowed on each iteration; $N_{\max}$ – upper bound for the number of initial points N.

In order to demonstrate this process and to investigate the stability of the proposed technique with respect to some of these parameters, Table 3.6 and Table 3.7 show results for two of the test functions with different dimensionalities ($n = 2$ and $n = 20$), where several values of the parameters M and N_{max} are considered. These results illustrate how the change of M and N_{max} affects the performance of the method (published in the *IEEE Transactions on Neural Networks* (Jordanov and Georgieva, 2007a)). In general, as they increase (and allow greater number of trial points to be generated), the success rate of the method improves. However, the goal is to achieve *good* performance for as small number of points as possible. The results in Table 3.6 and Table 3.7 show quite stable performance of the *LPτNM* technique, i.e. stable increase of the success rate with the increase of the trial points. It is interesting to note that in the case of 20-dimensional *Brown* function (Table 3.7) after a certain point, the increase of M does not improve the success rate and even slightly deteriorates it. This might be due to the fact that, having to investigate too many regions of future interest, the algorithm sometimes gets distracted from the real GM regions in favour of LM ones, that, at certain point of the optimisation, have looked more attractive.

Table 3.7. *Brown* function ($n = 20$) optimised by *LPτNM* with different values for M *and* N_{max}. All other parameter values are given in Table 3.5.

M / N_{max}	10	15	**20**	25	30
2^{11}	5151 (98)	5719 (99)	5901 (98)	6552 (97)	6813 (97)
2^{12}	7128 (98)	7519 (98)	7997 (100)	8674 (100)	8680 (99)
2^{13}	10755 (98)	11284 (99)	**11425 (100)**	11582 (99)	11666 (98)
2^{14}	19275 (98)	19602 (100)	19731 (100)	19930 (98)	20351 (97)

The numbers in this stable present the number of function evaluations needed for the optimisation and in brackets gives the success rate in % (out of 100 runs for each test). The numbers in bold show the parameters combination selected in Table 3.5 and used in the rest of the tests (Table 3.8 and Table 3.9).

M – maximal future interest regions allowed on each iteration; N_{max} – upper bound for the number of initial points N.

The obtained results from *LPτNM* testing are compared with other stochastic methods – implementations of Tabu Search and Genetic Algorithms, namely, Enhanced Continuous Tabu Search (ECTS, Chelouah and Siarry, 2000b), Staged Continuous Tabu Search (SCTS, Zheng *et al.*, 2005), Continuous Hybrid Algorithm (CHA, Chelouah and Siarry, 2003), and Differential Evolution (for the DE experiments we used the Matlab source code provided by Price *et al.* (2005), and

default values of the parameters (DE/rand/1/exp, $CR = 0.9$, $NP = 10n$), while for F we used $F = 0.6$ (instead of $F = 0.8$), which gave much more efficient results for all the test functions.

As for the most optimisation algorithms, the choice of initial population of points is very important for the *LPτNM*. In order to investigate the impact of the starting points and evaluate the effectiveness (or stability with respect to the initial testing points) of the algorithm, we performed tests with *perturbed* original domain of interest, which gave us 101 different cases for each test function.

Table 3.8. Comparison of *LPτNM* with other GO methods in terms of mean number of function evaluations. If the obtained success rate is less than 100%, it is shown separately in Table 3.9.

Method Function (n)	*LPτNM*	ECTS	SCTS	CHA	DE
Shubert (2)	**303**	370	521	345	4498
Goldst.-Price (2)	**182**	231	696	259	595
Branin (2)	247	**245**	492	295	807
Rosenbrock (2)	**226**	480	-	459	-
Zakharov (2)	**180**	195	-	215	-
Easom (2)	**248**	1284	-	952	-
Sphere (3)	**266**	338	-	371	331
Hartman (3)	**292**	548	560	492	679
*Shekel*10 (4)	1079	898	-	**635**	-
*Shekel*7 (4)	837	910	-	**620**	3064
*Shekel*5 (4)	839	825	-	**698**	3920
Rosenbrock (5)	2353	**1142**	-	3290	9211
Zakarov (5)	1163	2254	-	**950**	4123
Hartman (6)	1552	1520	-	**930**	-
Rosenbrock (10)	**9188**	15720	-	14532	54134
Zakharov (10)	6826	4630	-	**4291**	34532
Levy (20)	**10987**	-	17443	-	29268
Brown (20)	**11425**	-	15142	-	28032

Methods: ECTS – Enhanced Continuous Tabu Search by Chelouah and Siarry (2000); SCTS – Staged Continuous Tabu Search by Zheng *et al.* (2005); CHA – Continuous Hybrid Algorithm by Chelouah and Siarry (2003); DE – Differential Evolution by Price *et al.* (2005).

Let $\Pi = \{P = (p_1, \ldots, p_n): a_i \leq p_i \leq b_i, i = 1, \ldots, n\}$ is the original domain of interest for a given test function. We performed 101 independent minimisation procedures for

each function with perturbation of the original values of $a = (a_1, \ldots, a_n)$ and $b = (b_1, \ldots, b_n)$ with up to 5% of the intervals' lengths. In other words, we performed minimisation for each region

$\Pi^j = \{P=(p_1, \ldots, p_n)$: $a_i + q|b_i - a_i| \leq p_i \leq b_i + q|b_i - a_i|$, where $q = (j0.0001 - 0.05)$ and $j = 0, \ldots, 100$.

Table 3.8 shows the mean value of the number of function evaluations (illustrated in Fig. 3.11) and in Table 3.9 the rate of success for some of the functions is given (only the cases in which a 100% success rate was not achieved for one ore more of the methods).

When estimating the success rate, the solution is considered to be successful if the following condition holds:

$$|\widehat{F}^* - F_{\min}| \quad < 10^{-4}|F_{\min}|, \tag{3.5}$$

where $\widehat{F}^*$ is the solution found by our method and $F_{\min}$ is a pre-known solution. This condition guarantees that the solution found by our method has a relative error less than 0.1%. For the reported results in the SCTS case, this parameter is relaxed to 1%. In the case of $F_{\min} = 0$, the condition (3.5) reads as $|\widehat{F}^*| < 10^{-4}$. Of course, higher accuracy could be easily achieved if needed (by reducing ϵ from (3.4)), but it will cost more function evaluations.

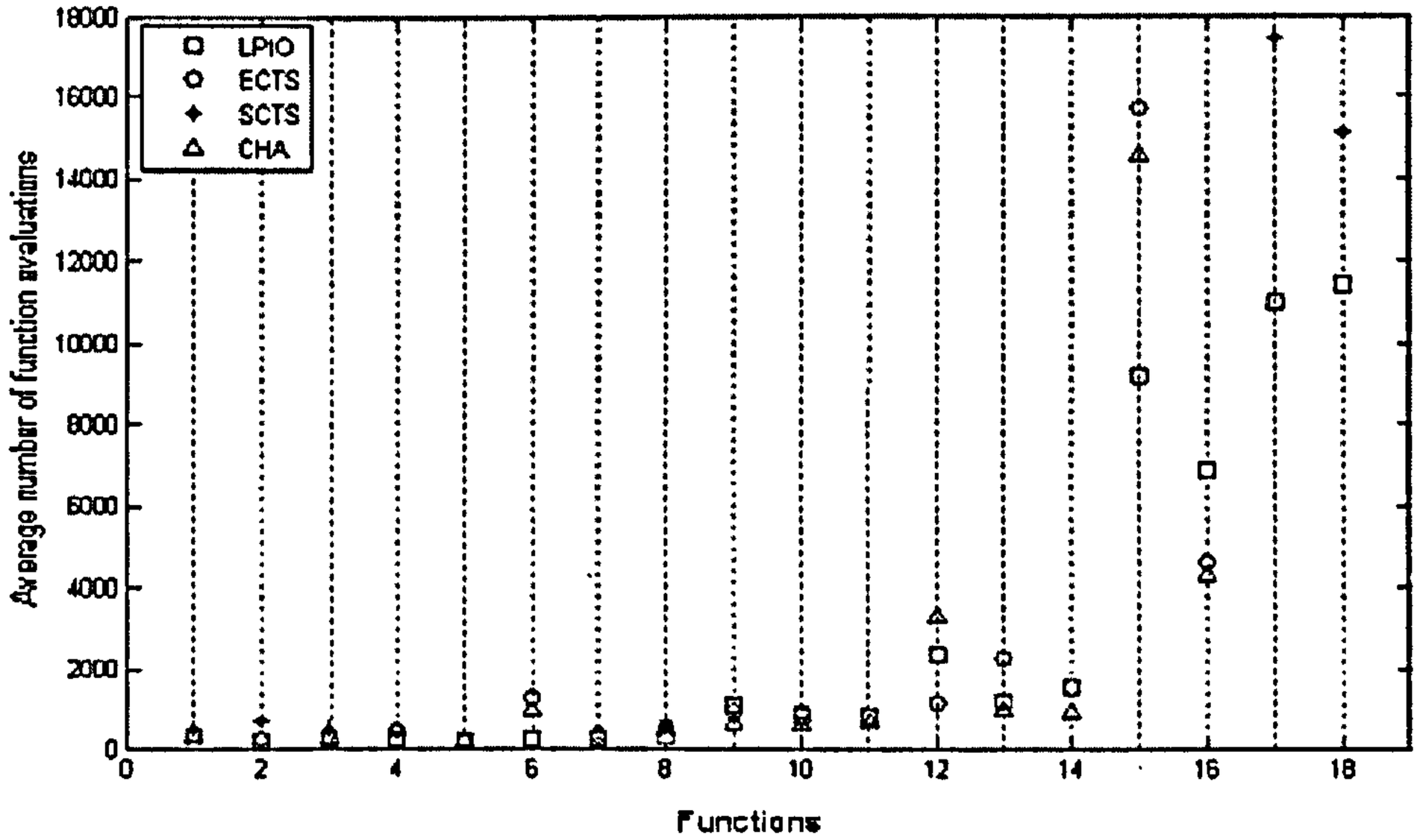


Figure 3.11. Illustration of the results presented in Table 3.8.

Table 3.9. Comparison *of LPτNM* with other GO methods in terms of mean success rate (%).

Method Function (n)	*LPτNM*	ECTS	CHA	DE
Shubert (2)	85	100	100	95
*Shekel*10 (4)	96	75	85	100
*Shekel*7 (4)	100	80	85	100
*Shekel*5 (4)	100	75	85	100
Rosenbrock (5)	91	100	100	100
Rosenbrock (10)	88	85	83	100

Methods: ECTS – Enhanced Continuous Tabu Search by Chelouah and Siarry (2000b); CHA – Continuous Hybrid Algorithm by Chelouah and Siarry (2003); DE – Differential Evolution by Price *et al.* (2005).

The results reported in Table 3.8 (Fig. 3.11) and Table 3.9 could be improved for each particular function by adjusting the parameters of the algorithm, as well as their bounds of variation (maximal and minimal possible values). We selected these particular parameter values since our tests showed them as leading to the best overall results if no tuning for each function is performed.

The results show that the proposed *LPτNM* technique can be successfully applied for optimisation of continuous functions. No derivatives, Lipschitz estimates, or any initial information for the objective function are needed. This makes *LPτNM* applicable in cases where classical gradient and recent deterministic GO methods could not be used (Georgieva and Jordanov, 2005b).

The multiple execution of the algorithm for each function with slightly changed boundaries of the searched space, gives us the opportunity to evaluate the *effectiveness* of *LPτNM* and to investigate the stability of the algorithm with respect to the initial population of points. Only in the case of the *Shubert* function, we might consider the performance as not very stable, indicated by the achieved unreasonably low success rate. However, for all other functions, the high percentage of successful runs shows that *LPτNM* is robust and the position of initial *LPτ* points does not affect it significantly.

In terms of number of function evaluations, in general, *LPτNM* showed to be strongly competitive when compared with the other methods given in Table 3.8. For

example, for the *Shekel* functions and the ten dimensional *Zakharov* function – CHA and ECTS methods performed better, but for the other functions *LPτNM* outperformed them. The results in terms of success rate also showed that our method is strongly competitive (Table 3.9). For instance, *LPτNM* improved significantly the success rate for all *Shekel* family functions, where the other algorithms performed poorly. Although *LPτNM* needed greater number of function evaluations for this family, the achieved success rates are much higher. On the other hand, DE achieved 100% success rate for the *Shekel* family, but with the price of many more function evaluations than *LPτNM*. Generally, three of the methods used for comparison appeared more effective in the case of *Shubert* function, where *LPτNM* and DE were not able to reach 100% success rate. The *LPτNM* has also performed less effectively for the five-dimensional case of the *Rosenbrock* function (but not for the ten-dimensional case of this function (Table 3.8)). For both 20-dimensional test functions *Levy* and *Brown,* the *LPτNM* and DE performed with 100% success rate whilst *LPτNM* needed fewer function evaluations. For example, in the case of the Brown function we needed 11425 function evaluations, whereas the SCTS method needed 15142 and DE – 28032.

Nevertheless, with the increase of problem dimensionality, the computational load for *LPτNM* becomes too heavy. For problems of higher than 20 dimensions, our method did not show promising results (often getting trapped in a local minimum). This motivated us to combine it with EAs, since they are currently the best explorers of high dimensional spaces as discussed in Section 2.2. The hybrid method that includes EA is proposed, investigated and evaluated in Chapter 4.

3.2.3 Results from *LPτNM* Applied for NN Training

A review of *LPτNM* and the function optimisation results as well as the presented in this Section results from the application of *LPτNM* to several NN training problems, are published in the *IEEE Transactions on Neural Networks* (Jordanov and Georgieva, 2007a).

For all the experiments reported here, we used the same general NN architecture, as stated in Section 3.1.4, with varying number of neurons and different output transfer functions according to each task at hand. BP (Levenberg-Marquardt) is again

performed in parallel for comparison purposes. Here, we also use DE for NN training and report the results (the DE Matlab source code is obtained from Price *et al.*, 2005).

- Classification of XOR Problem

For the classification of the XOR problem, the same architecture as the one used in Section 3.1.4 is adopted here. Testing and training is performed in the same way with the same data. Nevertheless, more sophisticated measures of performance are reported here in Table 3.10.

Table 3.10. Optimal errors for the *LPτNM*, BP, and DE (XOR problem).

Criterion Method	Error Function (Std. Dev.)	Mean Test Error (Std. Dev.)	Mean CPU time in seconds
BP	0.085 (0.07)	0.2521 (0.048)	0.4
DE	0.01335 (0.044)	0.2021 (0.049)	6
LPτNM	1.37e-06	0.001 (0.007)	0.7

Method: BP – Backpropagation with Levenberg-Marquardt optimisation (the source of Matlab 7 is used); DE – Differential Evolution, the source provided by Price *et al.* (2005) is used.

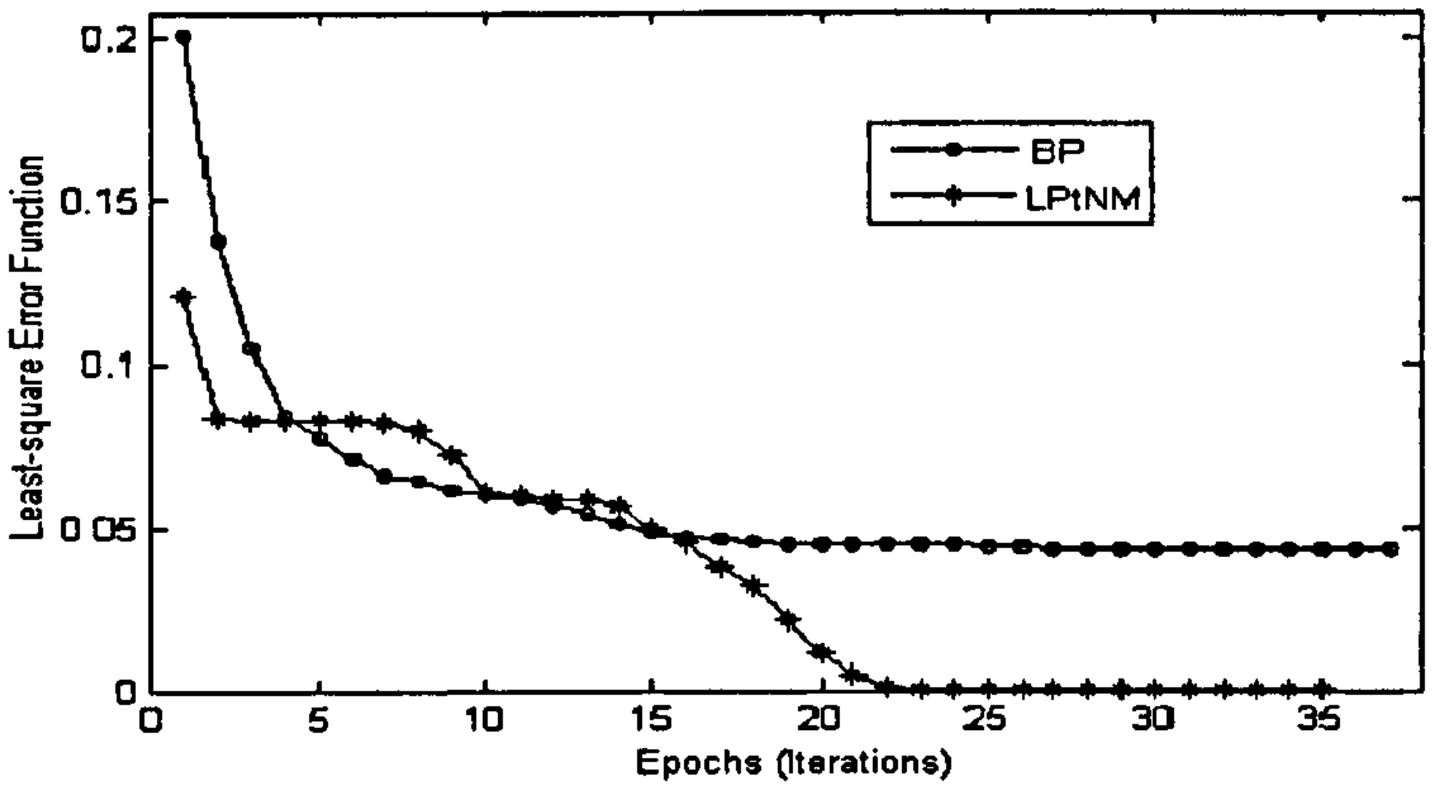


Figure 3.12. Error function for the XOR problem when BP or *LPτNM* is used.

The process of error function minimisation (NN training) is illustrated in Fig.3.12, where *LPτNM* is compared with BP (average of 100 runs). It can be seen that in the early stages of the minimisation, BP outperforms our method. These are the stages where *LPτNM* still localises the regions of attraction, but subsequently, once they are determined, the errors produced by our method become smaller than those of BP.

- Classification of *N*-Parity Problem

Again, this problem is treated as in Section 3.1.4, adopting the same architecture and testing set. The optimal results obtained (along with BP and DE ones) from the training and testing with 100 testing samples (with a 15% noise) are shown in Table 3.11.

Table 3.11. Optimal errors for the *LPτNM*, BP, and DE (4-*Parity* problem).

Criterion / Method	Error Function (Std. Dev.)	Mean Test Error (Std. Dev.)	Mean CPU time in seconds
BP	0.0168 (0.042)	0.1539 (0.096)	1
DE	0.1099 (0.003)	0.2526 (0.049)	107
LPτNM	6.98e-06	0.1488 (0.332)	38

Method: BP – Backpropagation with Levenberg-Marquardt optimisation (the source of Matlab 7 is used); DE – Differential Evolution, the source provided by Price *et al.* (2005) is used.

- Classification of *Iris* Problem

The Iris classification problem is maybe the most popular benchmark NN training task from the UCI repository database (Bortoletti *et al.*, 2003; Rocha *et al.*, 2003; Lee *et al.*, 2004; Ludemir *et al.*, 2006). This problem has three classes to be classified: whether an iris sample is from the type Setosa, Versicolour, or Virginica (sample pictures are shown in Fig. 3.13). The 150 samples in the database are equally distributed for the three classes (50 samples for each type). The first class is linearly separable from the other two and the latter ones are not linearly separable from each other (Fig. 3.14). Each sample has four attributes characterising the length/width (in cm) of the flower's sepals and petals. The four attributes have continuous values in the range: sepal length in [4.3, 7.9], sepal width in [2.0, 4.4], petal length in [1.0, 6.9], and petal width in [0.1, 2.5].

Figure 3.13. Samples of different *Iris* flowers.

A network architecture 4-4-1 is employed to produce continuous output with the three classes coded as 0.1 (Setosa), 0.5 (Versicolour), and 0.9 (Virginica). The network defines $n = 25$ dimensional learning optimisation problem. For the training set we randomly chose 25 samples of each type, forming a training set of 75 examples (keeping the same class distribution as in the whole dataset). The rest of the samples are used for testing. Table 3.12 shows the obtained optimal results from the training and testing.

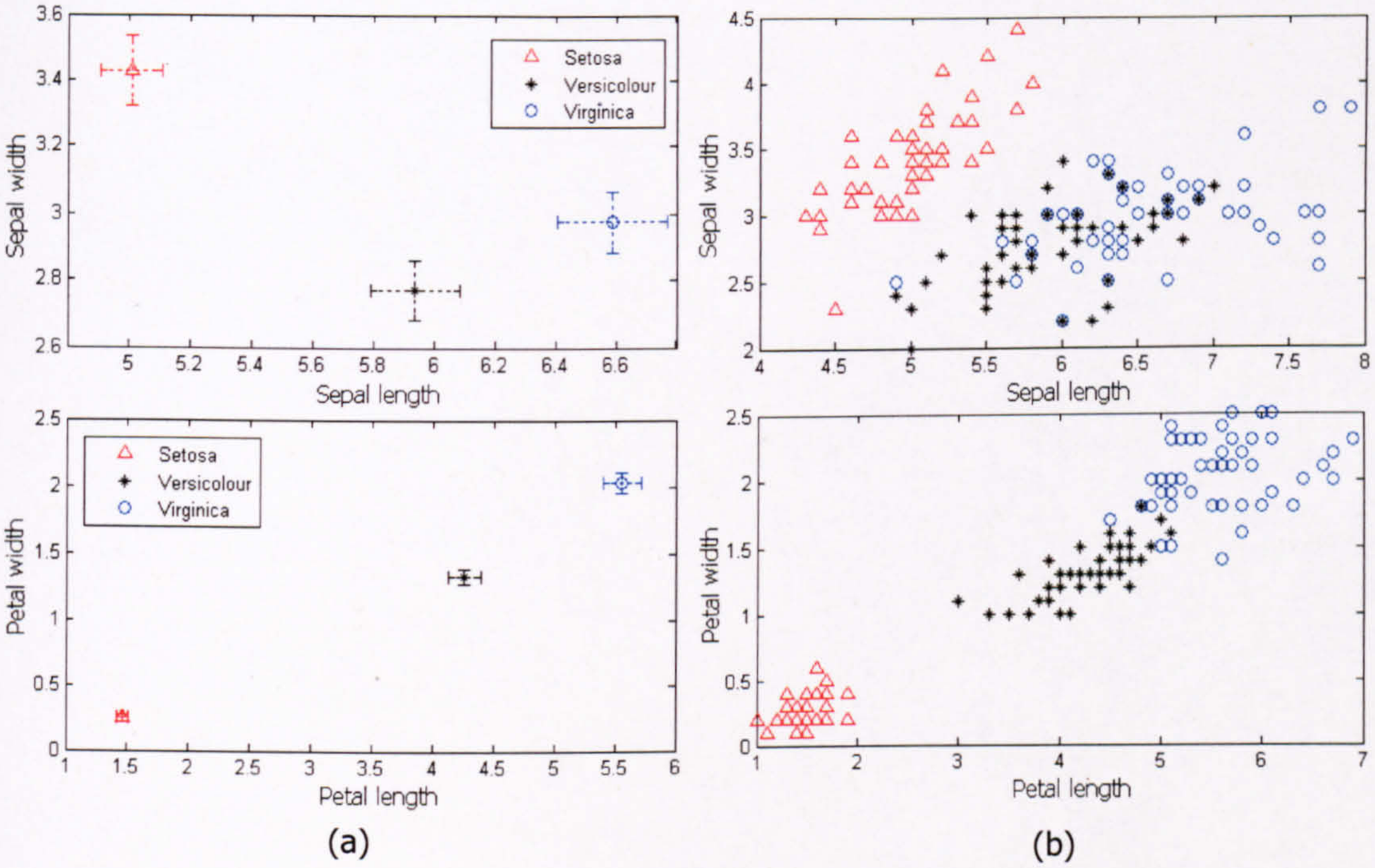


Figure 3.14. The *Iris* classes *Setosa*, *Versicolour*, and *Virginica*: (a) Classes means with 95% confidence intervals; (b) Scatter plot of the *Iris* classes.

Table 3.12. Optimal errors for the *LPτNM*, BP, and DE (*Iris* problem).

Criterion Method	Error Function (Std. Dev.)	Mean Test Error (Std. Dev.)	Mean CPU time in seconds
BP	0.0131 (0.0619)	0.049 (0.0736)	1.36
DE	0.0085 (5e-05)	0.0607 (0.0553)	24
LPτNM	0.0008	0.022 (0.069)	6.9

Method: BP – Backpropagation with Levenberg-Marquardt optimisation (the source of Matlab 7 is used); DE – Differential Evolution, the source provided by Price *et al.* (2005) is used.

- Predicting the rise time of a servomechanism

The Servo data collection is another database from the UCI repository. It represents an extremely non-linear phenomenon Quinlan (1993b) – predicting the rise time of a servomechanism, depending on four attributes: two continuous gain settings and two discrete choices of mechanical linkages. It has been considered in several relevant works – Quinlan (1993b), Rocha *et al.* (2003), Huang *et al.* (2006), and others. The database consists of 167 samples and the output is a continuous value (the time in seconds). In order to avoid computational inaccuracies, we have normalised the set of outputs to have a zero mean and unit standard deviation.

A network with 4-4-1 topology (25-dimensional problem) was again adopted to produce continuous output. The dataset was divided into two parts – 84 training examples and 83 testing ones. In this case, the output layer had a linear transfer function (instead of a *Sigmoid* one) in order to be able to produce output outside [0, 1] interval. Optimal results obtained from the learning and testing are given in Table 3.13.

Table 3.13. Optimal errors for the *LPτNM*, BP, and DE (*Servo* problem).

Criterion Method	Error Function (Std. Dev.)	Mean Test Error (Std. Dev.)	Mean CPU time in seconds
BP	0.0474 (0.06)	0.04171 (0.5515)	3.76
DE	0.0085 (5e-05)	0.2872 (0.443)	45
LPτNM	0.0271	0.2121 (0.2529)	57

Method: BP – Backpropagation with Levenberg-Marquardt optimisation (the source of Matlab 7 is used); DE – Differential Evolution, the source provided by Price *et al.* (2005) is used.

Fig. 3.15 shows the error values for each testing sample as well as the average errors for both BP and *LPτNM*. It can be seen that our method produced lower mean test values and most of the outliers (considerably larger test errors) belong to BP.

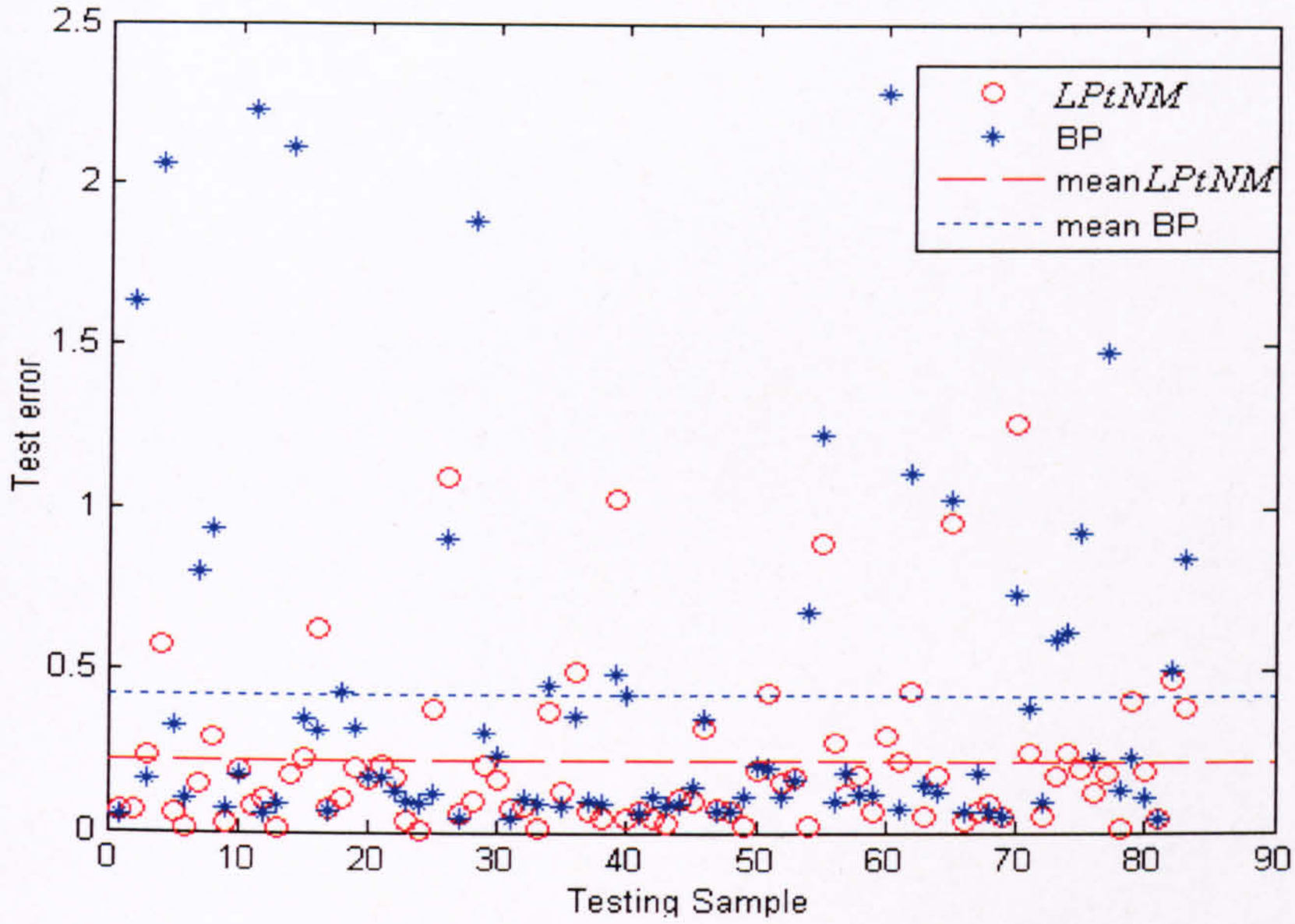


Figure 3.15. Test errors and mean test errors for BP and *LPτNM*.

Discussion

For all simulations, our method outperformed BP and DE, in terms of both training and testing errors, and the NN trained with *LPτNM* demonstrated better generalisation abilities. It can be seen from Table 3.10 to Table 3.13 that the difference in mean test errors between BP, DE and our technique in the case of XOR problem is considerably greater than the difference in the other three simulations, and in all cases our method produced favourable results. For example, the mean test errors for the XOR case are 0.25 for BP; 0.30 for DE; and 0.001 for our method (Table 3.10). For this problem, many researchers report low success rates for BP, with frequent entrapment in local minima (Bortoletti *et al.*, 2003; Wang *et al.*, 2004). In the case of 4-*Parity* problem (Table 3.11), the results for the three methods are comparable, whereas for the last two problems, our average test errors are better than those of BP and DE. For example, in the *Iris* case, the average test error is 0.048 for BP, 0.061 for DE and 0.022 for our method (Table 3.12). Our result is also better than the best (0.03) of the results reported in Rocha *et al.* (2003) for several other methods. Multiple entrapment in local minima for the *Iris* problem was also reported by Bortoletti *et al.* (2003). For the *Servo* problem we obtained a 0.212 mean test error, against 0.417 for BP (Table 3.13), and 0.287 for DE. Our result is also better than the best results reported in Rocha *et al.* (2003) and Quinlan (1993b). Superior than our results are given in Huang *et al.* (2006), but they are achieved with much more complex NN architectures and

greater computational expense. In terms of convergence speed, BP demonstrated fastest performance, whereas DE overtook our method only in the Servo case. This is not surprising, since BP is a gradient based method, DE is an evolutionary algorithm, and *LPτNM* is a population based heuristic technique.

In conclusion, the investigated *LPτNM* Global Optimisation method has been applied to NN learning and the results from multiple (100) independent test runs have shown consistent and stable performance (although slower than BP). For all of the reported problems, the proposed method has produced NN with better generalisation abilities (compared to BP and DE), demonstrating very competitive results that qualify it as an efficient and reliable technique for training neural networks with moderate degrees of dimensionality.

3.3 Summary

In this chapter has been proposed a novel GO technique based on *LPτ* Low-discrepancy sequences and novel heuristic rules. The method has been discussed in detail and after the initial promising tests on function optimisation and NN training, the technique has been hybridised with a Nelder-Mead local search. The hybrid method called *LPτNM* has been investigated through a number of tests, including stability with respect to the initial points, search domain, user-defined algorithm parameters, etc. The results from testing on 18 mathematical function and 4 NN training benchmark problems have shown *LPτNM* as reliable and very promising GO technique.

With the increase of problems dimensionality, the computational load increases considerably, which is a problem to be tackled in the next Section. Nevertheless, when large size NN are trained, *LPτNM* could be used as part of a hybrid method, for initial search and localisation of starting points for local searches.

4 A novel hybrid method based on Genetic Algorithms and Low-discrepancy Sequences

This chapter presents a novel hybrid Global Optimisation technique, based on Genetic Algorithms and the LPτNM method described in Chapter 3. A novel hybrid technique is proposed, discussed, and tested on a number of function optimisation problems. Its performance is compared with other state-of-the-art Evolutionary Algorithms. Subsequently, it is applied for several Neural Network learning benchmark problems and the network's generalisation abilities are compared with those of Backpropagation.

4.1 The Evolutionary Hybrid *GLPτS* Technique

4.1.1 Motivation and Introduction

The research and results reported in this chapter were submitted for publication in the *European Journal of Operational Research* (Georgieva and Jordanov, 2007a).

Genetic Algorithms are known for their very good exploration abilities. When optimal balance with their exploitation abilities is found, they can be powerful and efficient global optimisers (Leung and Wang, 2001; Mitchell, 2001; Yao, 2002). Exploration dominated search could lead to excessive computational expense. On the other hand, if the exploitation is favourable, the search is in danger of premature convergence or simply turning into a local optimiser. Keeping the balance between the two and preserving the selection pressure relatively constant through the whole run is an important characteristic of any GA technique (Mitchell, 2001; Ali *et al.*, 2005). Other problems associated with GA are their relatively slow convergence and low accuracy of the solutions found (Yao *et al.*, 1999; Ali *et al.*, 2005). This is the reason why GA are often combined with local search techniques (Joins and Kay, 2002). Our hybrid method aims to tackle these problems effectively by complementing GA and *LPτO* search (proposed and investigated in Chapter 3).

The *LPτO* technique can be summarised as follows: initially we *seed* the whole search region with *LPτ* points, from which we choose the most promising ones to be centers of regions in which we *seed* new *LPτ* points. Then we choose few promising from the new ones and again *seed* in the neighbourhood of each one and so on, until a halting condition is satisfied. We combine the *LPτO* technique with a GA of moderate population size. The GA aims to explore the searched space and improve the initial *seeding* with *LPτ* points by applying genetic operators in a few generations. Subsequently, a heuristic-stochastic rule is applied in order to select some of the individuals and to start *LPτO* search in the neighbourhood of each selected one. Finally, we use the Nelder-Mead Simplex Search, discussed in Chapter 3, to refine the solution and achieve better accuracy.

4.1.2 The *GLPτS* Technique – a Detailed Proposal

In this section we introduce a hybrid method called *Genetic LPτ and Simplex Search* (*GLPτS*), which combines the effectiveness of GA during the early stages of the search with the advantages of the proposed in Chapter 3 *LPτO* method, and the local improvement abilities of Simplex Search. In Chapter 3 were reported promising results for the *LPτO* Global Optimisation technique over a set of multi-modal test functions of moderate dimensionsionalities. However, in order to be able to handle efficiently problems of higher dimensions, we propose here an improved hybrid method that utilises genetic operators for finding promising regions. Based on the complexity of the searched landscapes, most authors intuitively choose population size that could vary from 100s to 1000s (De Jong, 2006). We employ smaller number of points that leads to a final population with promising candidates from regions of interest, but not necessarily to a GM. Also, our initial population points are not random (as in a conventional GA), but uniformly distributed *LPτ* points.

Generally, the technique could be described as follows:

Step 1. Generate a number I of initial *LPτ* points;

Step 2. Select G points, ($G < I$), that correspond to the best function values. Let this be the initial population $p(G)$ of the GA;

Step 3. Perform GA until a halting condition is satisfied;

Step 4. From the population *p(G)* of the last GA generation, select *g* points of future interest ($1 \leq g \leq G/2$);

Step 5. Initialise the *LPτO* search in the neighbourhood of each selected point;

Step 6. After the stopping conditions of the *LPτO* searches are satisfied, initialise a local Simplex Search in the best point found by all *LPτO* searches.

Here, the selection, recombination and mutation GA operators are briefly described and the way GA is combined with *LPτO* is discussed in detail. The *GLPτS* properties - consistency, convergence, and the improvements resulting from the hybridisation of GA with *LPτO* are also considered in this chapter.

Genetic operators

The basic conventional genetic operators are introduced in Chapter 2 (Section 2.2.3). The main goal of the selection mechanism is to find an efficient balance between the exploration and exploitation abilities of the search during the run. Too much selection pressure increases the exploitation and the probability of turning the population homogeneous sooner, rather than later. This could diminish the ability of the reproductive operators to produce variation in the population and could decrease the likelihood of converging to a global optimum. The reproductive operators support in some degree both – exploration and exploitation, but, in general, depending on the homogeneity of the population, the recombination (cross-over) supports more the exploitation and mutation enforces more exploration in the search. We use conventional one-point recombination and our mutation operator is the same as in (Leung and Wang, 2001). We keep constant population size starting with *G* individuals. The general form of the performed GA is:

Step 1. From the current population *p(G)*, each individual is selected to undergo recombination with probability P_r. If the number of selected individuals is odd, we dispose of the last one selected. All selected individuals are randomly paired for *mating*. Each pair produces two new individuals by recombination;

Step 2. Each individual from the current population *p(G)* is also selected to undergo mutation with probability P_m;

Step 3. From the *parent* population and the *offspring* generated by recombination and mutation, the best *G* individuals are selected to form the new generation *p(G)*.

Step 4. If the halting condition is not satisfied, the algorithm is repeated from step 1.

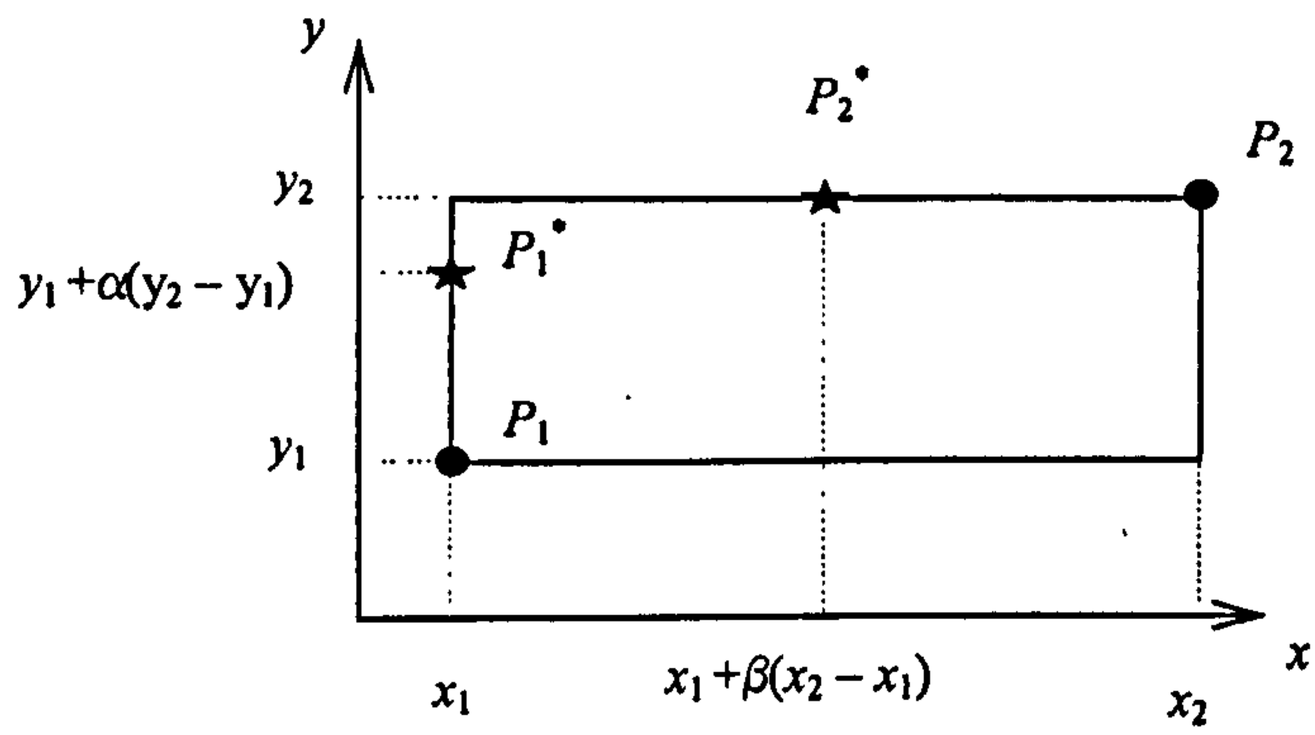


Figure 4.1. Recombination in the two dimensional case: P_1, P_2 (dots) are the *parents*, and P_1^*, P_2^* (stars) are the *children*.

- Recombination

Let $x = (x_1, \ldots, x_n)$ and $y = (y_1, \ldots, y_n)$ be two individuals from the *parent* population and let n is the dimensionality of the problem. Then, a random integer $k \in [1, n-1]$ is drawn, and two *children* $\bar{x}$ and $\bar{y}$ are obtained as follows:

$$\begin{aligned} \bar{x} &= (x_1, \ldots, x_k, \bar{x}_{k+1}, \ldots, \bar{x}_n), \\ \bar{y} &= (y_1, \ldots, y_k, \bar{y}_{k+1}, \ldots, \bar{y}_n), \end{aligned} \tag{4.1}$$

where $\bar{x}_i = x_i + \alpha_i(y_i - x_i)$ and $\bar{y}_i = y_i + \beta_i(x_i - y_i)$, for $i = k+1, \ldots, n$, and α_i, β_i are randomly drawn numbers in the interval [0, 1]. Fig. 4.1 shows a two dimensional illustrative example of how children are produced by shifting one of the coordinates of each *parent*.

- Mutation

If an individual is selected for mutation (with probability P_m), we draw a random integer m, $1 \leq m \leq n$. Then, the k-th coordinate of this individual is replaced by a randomly drawn value in the search interval for this coordinate.

- Stopping condition

We use two stopping criteria that represent two common approaches, first - to stop when a homogeneous population is reached, and second - to stop when given computational resource is exhausted. We compute the mean $\bar{f}$ of all function values

in *p(G)*. If f^* is the current best function value, evolved in the previous generations, the stopping condition is considered satisfied if $|\bar{f} - f^*| < 0.001|f^*|$. This condition guarantees that the GA population has converged to a function value that can not be improved significantly any further. If this condition is not satisfied, the algorithm stops when a predefined number of generations is reached. The adopted population size and maximal number of iterations are smaller than the usually used in conventional GA algorithms. For example, we adopted a population size of 60 for the 30 dimensional problems, compared with the size of 200 adopted by (Leung and Wang, 2001), size of 100 used by (Yao, 1999), and size of up to 1000 proposed by (De Jong, 2006). The choice of smaller population size is driven by the fact that, at this stage, we do not aim to find a GM, but only to locate promising regions, in which *LPτO* will continue the search.

Combining GA with LPτO to form *GLPτS*

To determine the number g of subsequent *GLPτS* searches (**Step 4**), we proceed as follows:

Let *p(G)* be the population of the last generation, found by the GA run. Firstly, we sort in non-descending order all G individuals using their fitness values and then associate rank r_i to the first half of them by using formula (4.2):

$$r_i = \frac{f_{\max} - f_i}{f_{\max} - f_{\min}},\ i = 2, \ldots, G/2. \tag{4.2}$$

In (4.1), $f_{\max}$ and $f_{\min}$ are the maximal and minimal fitness values of the population and the rank r_i is given with a linear function which decreases with the growth of f_i and takes values within the range [0, 1].

The best individual of the population *p(G)* from the last generation has rank $r_1 = 1$ and always competes. It is used as a centre for a hyper-cube (with side $2R$), in which the *LPτO* search will start. The parameter R is heuristically chosen with the formula

$$R = 50/G + \text{int}_{\max} * 0.001, \tag{4.3}$$

where $\text{int}_{\max}$ is the largest of all initial search intervals. This parameter estimates the trade-off between the computational expense and the probability of finding a GM. The greater the population size G in the GA, the smaller the intervals of interest that are going to be explored by the *LPτO* search. This formula is heuristically determined for

a test set of problems with 10-150 dimensions (Section 4.2). The next individual P_i, $i = 2, \ldots, G/2$ is then considered, and if all of the Euclidean distances between this individual and previously selected ones are greater than $2R$ (so that there is no overlapping in the *LPτO* search regions), another *LPτO* search will be initiated with a probability $r_i P_{LP}$. Here P_{LP} is a user-defined probability constant in the interval [0, 1]. Therefore, individuals with higher rank (that corresponds to lower fitness) will have greater chance to initiate *LPτO* searches. This heuristic algorithm is illustrated in Fig. 4.2.

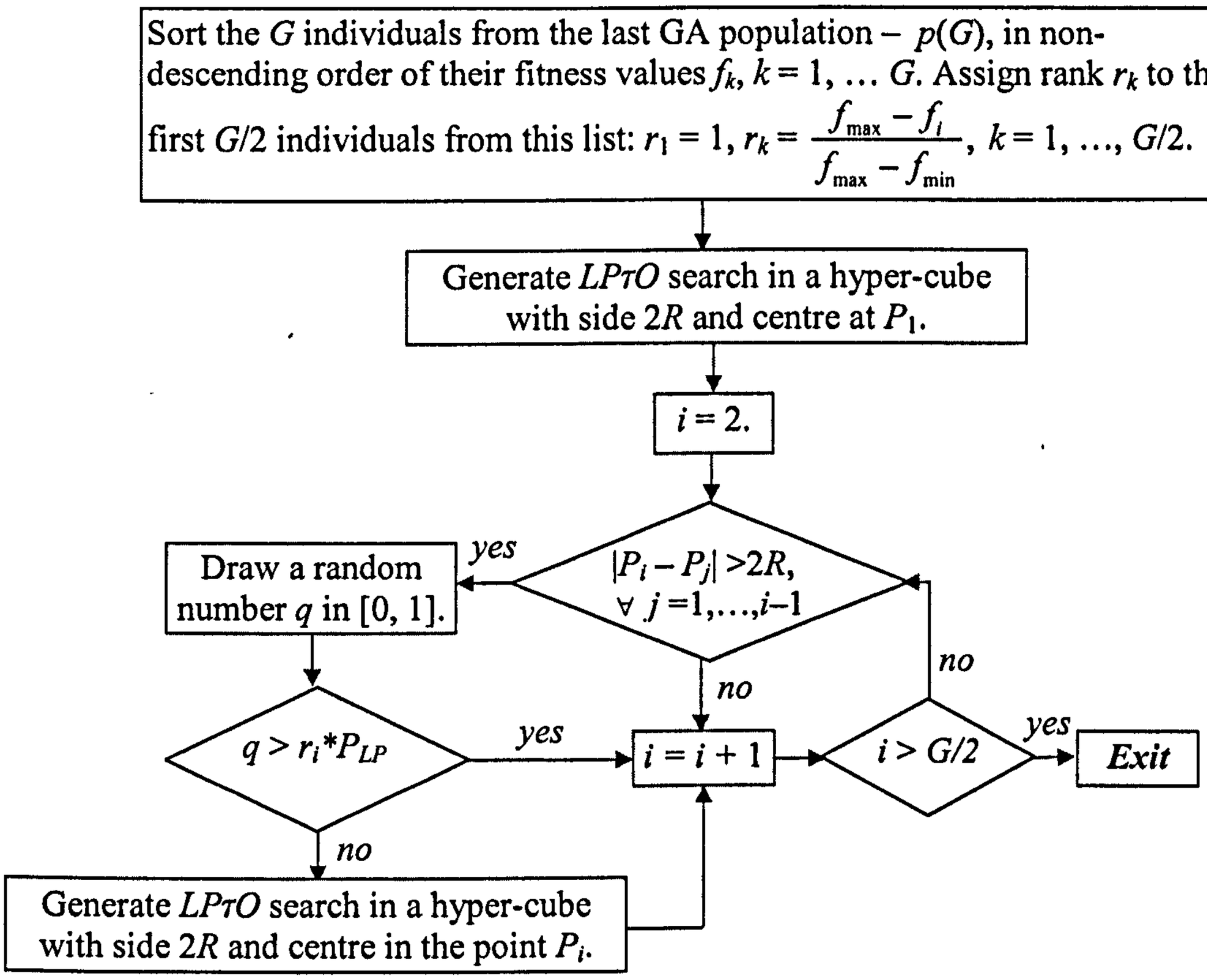


Figure 4.2. Algorithm for adaptive selection of points of future interest from the last population of the GA run.

After the execution of *LPτO* searches is complete, Nelder-Mead Local Simplex Search is applied to the best function value found in previous stages of *GLPτS*. The *NM* is initialised in the same manner as described in Chapter 3 (Section 3.2.1). The only difference is that for the tests here $\epsilon = 10^{-9}$, which increases the number of *NM* iterations and provides better overall results. The parameter β is also slightly changed here, instead of $\beta = 0.5$, as in the tests in Chapter 3, here we adopt $\beta = 0.8$, which

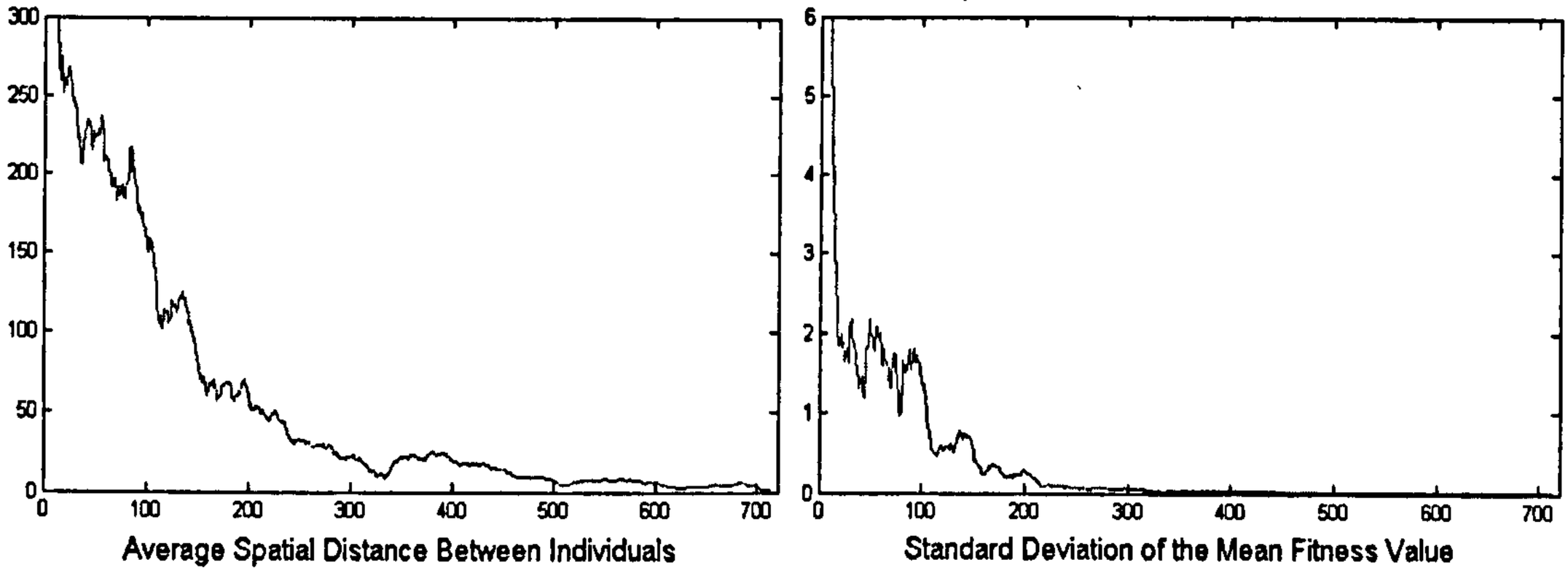
provided overall faster convergence for the test set of high dimensional functions used in Section 4.2.

4.1.3 Properties

The *consistency* of the *GLPτS* method follows directly from the way the algorithm is performed. During all stages of the implementation, the *best* point from the previous iteration continues to compete in the following ones and the result at each iteration of the algorithm cannot be worsened.

The *convergence* of the *GLPτS* method is determined by the properties of the GA, *LPτO*, and the way these methods are combined. The GA speed of convergence depends on a number of factors: initial conditions (size of initial population); selection mechanism; choice of reproductive operators; stopping conditions; etc. Strong selection pressure in the early stages of the run could result in quick but possibly premature convergence. Reducing the pressure would slow down the search process, but will increase the probability of converging to a global solution. In order to assess the performance of GA, (De Jong, 2006), recommends population homogeneity and global objective fitness improvements as fairly robust and problem independent measures of convergence. For the homogeneity – spatial dispersion and entropy could be used as metrics. An illustrative example of the way the GA converges, in the case of thirty dimensional *Griewank* function, is given in Fig. 4.3. The first graph shows the mean spatial distance of all population points at each generation. Because in the early generations the variance in the population is still considerable, the new individuals generated by the reproductive operators, will have greater spatial dispersion. This reflects in the peaks being greater and more frequent in the left-hand side of the graph in Fig. 4.3(a), than those at the right-hand side, where the population gets more homogenous. The same tendency can be seen in Fig. 4.3(b), where the standard deviation (*std*) of the fitness values is shown. This is another feature of the population homogeneity and again, the *std* variance illustrates the same tendency from the left to the right hand side of the graph.

The *GLPτS* speed of convergence is also controlled by the user-defined parameter P_{LP} that influences the probability of initiating *LPτO* search in a suitable region of interest. If more computational resources are available, $P_{LP} = 1$ would provide better exploration for the price of higher number of objective function evaluations.



(a) Mean distance between individuals.

(b) Standard deviation of the mean fitness value of the population.

Figure 4.3. GA homogeneity during successive generations (30-dim. *Griewank* function).

After the GA run, the *LPτO* technique takes over and continues the search. The radius of *LPτO* searches, given by (3.2), has a strong influence on the convergence of the *GLPτS* (as discussed in detail in Section 3.1.2). It depends on the population size, the dimensionality of the problem, and the volume of the initial search space. The greater the value of this parameter, the greater is the probability of finding a GM. The theoretical convergence properties of the *LPτO* method are based on its deterministic characteristics, which have been discussed in Section 3.1.3.

In order to investigate how the hybridisation with *LPτO* improves the performance of the GA algorithm, some numerical tests addressing specifically this issue are performed. Results and discussion can be found in Section 4.2.1.

4.2 Results from Testing *GLPτS* on GO Benchmark Functions

4.2.1 Improving the method performance by hybridising GA and *LPτO*

In order to investigate how the hybridisation of *LPτO* and GA improves the overall method performance, we conducted some tests that specifically address this issue. We consider six test functions that are used in Chelouah and Siarry, (2003) (could be found in Appendix A) to assess the performance of their method which hybridise GA with *NM* Simplex Search. We use their technique called *Continuous Hybrid Algorithm* (CHA) to compare the performance of our *LPτNM* method that hybridise *LPτO* and *NM* Simplex Search. The reported results in Chapter 3 show that *LPτNM* can be very

effective (high percentage success rate), although in some cases with the expense of additional number of function evaluations (e.g., the *Shekel* family and ten-dimensional *Rosenbrock* function in Table 4.1), where CHA performed efficiently (lesser number of function evaluations), but not very effectively (lower success rate). On the other hand, *LPτNM* demonstrated poorer success rate for two of the functions in Table 4.1 (*Shubert* and five-dimensional *Rosenbrock* functions). Table 4.1 also shows additional results for *GLPτS*, which are discussed further on. More details about the functions given in Table 4.1 can be found in Appendix A.

Table 4.1. Comparison *of GLPτS*, *LPτNM*, and CHA in terms of mean number of function evaluations and mean success rate given in parenthesis (%).

Method Function (*n*)	*GLPτS*	*LPτNM*	CHA
*Shekel*10 (4)	1 117(100%)	1079 (96%)	635 (85%)
*Shekel*7 (4)	894 (100%)	837 (100%)	620 (85%)
*Shekel*5 (4)	1125 (86%)	839 (100%)	698 (85%)
Shubert (2)	267 (100%)	303 (85%)	345 (100%)
Rosenbrock (5)	2 722 (100%)	2 353 (91%)	3 290 (100%)
Rosenbrock (10)	10 674 (100%)	9 188 (88%)	14 563 (83%)

Methods: *GLPτS* – Genetic *LPτ* Search, proposed here; *LPτNM* – *LPτ* Nelder-Mead search proposed in Chapter 4; CHA – Continuous Hybrid Algorithm by Chelouah and Siarry (2003).

Table 4.2. Improvement of the performance due to inclusion of the *LPτO* stage (with parameters tuned individually for each function)

Function (*n*)	Method	*LPτO* stage	Success rate	Number of function evaluations	Function value for the successful runs: mean (std)	Function value for all runs: mean (std)
Ackley (30)		Yes	50/50	46157	8.9e-6 (1.6e-6)	8.9e-6 (1.6e-6)
		No	42/50	36744	9e-6 (1.5e-6)	0.43 (1.85)
PLMI (30)		Yes	49/50	37146	5e-8 (7e-9)	0.002 (0.015)
		No	44/50	33594	2e-7 (5e-7)	0.223 (1.16)
NDTF (30)		Yes	47/50	32685	-78.3323 (5e-9)	-78.2758 (4.7)
		No	41/50	29644	-78.3 (0.06)	-78.11 (14)

Functions: *PLMI* – Penalized Levy and Montalvo I Function; *NDTF* – *N*–dimensional Test Function.

In addition, we investigated here the performance of *GLPτS* for three of the thirty dimensional functions that are used for testing in Section 4.2.2. A hybridisation of GA with *NM* Simplex Search (omitting the *LPτO* stage) is compared with *GLPτS* in terms of success rate, number of function evaluations and mean function values. The results are given in Table 4.2 (identical parameter values are used in both cases).

The third column in Table 4.2 shows the number of successful runs out of 50 (a run is considered successful if the relative error between the found solution and the analytic minimum is less than 0.001). The fourth column shows average number of function evaluations, including those for the unsuccessful runs. The fourth column presents averages value of the minimum found by the corresponding method (the standard deviation is given in parentheses) for the successful runs only, while the last column of Table 4.2 shows the same metrics when the unsuccessful runs are also considered.

The obtained results for several low-dimensional problems (n = 2, 4, 5 and 10), given in Table 4.1, show that in five out of the six cases, the two techniques complement each other very well and the hybridised *GLPτS* method shows efficient and successful characteristics. In the cases where CHA (GA and Simplex Search) is not very effective in terms of success rate (ten-dimensional *Rosenbrock* and the *Shekel* functions), *LPτNM* (*LPτO* and Simplex Search) performs well and *GLPτS* achieves 100% success rate, for the expense of a small additional number of function evaluations (compared to *LPτNM*). Nevertheless, there is one exception for the case of *Shekel*5 function, where *GLPτS* performed poorly and further research is necessary to investigate this issue. In the cases of relatively higher dimensionality (5 and 10 dimensional *Rosenbrock* functions), *GLPτS* outperforms CHA, achieving 100% success rate. In the cases where *LPτNM* shows worse than CHA results (*Shubert* and five-dimensional *Rosenbrock*), *GLPτS* (combining the useful properties of GA and *LPτO*) demonstrates superior to the other two methods results.

4.2.2 Performance of *GLPτS* for Problems with Higher Dimensionalities

The *GLPτS* method proposed in Section 4.1 is tested on 16 benchmark mathematical functions from 10 to 150 dimensions (listed in Appendix A). From all functions, only *Schwefel II*, *Sphere*, *Hyper-Ellipsoid*, and *Rosenbrock* are unimodal, and the rest are multimodal, having many local minima. For functions *Ackley*, *Griewank*, *Schwefel I*,

PLMI, *PLMII*, *Rastrigin*, the number of local minima increases exponentially with the increase of their dimensionality (Yao *et al.*, 1999). The way *LPτ* low-discrepancy sequence points are generated guarantees that there will be always one point in the middle of the initial search region. For all of the above functions that have a GM in the origin and are investigated in symmetrical (to the origin) intervals, we intentionally translated the intervals (keeping their size the same) to asymmetric ones, in order to avoid 'accidental hit' of a GM point.

Results from testing our method on functions *Schwefel II*, *Sphere*, *Ackley*, *Griewank*, *Schwefel I*, *PLMI*, *PLMII*, *Rastrigin*, *Michalevicz*, *and NDTF*, without specific tuning of the method parameters are shown in sub-section 4.2.1. This is in correspondence to the testing performed in (Leung and Wang, 2001; and Yao *at al.*, 1999) of their methods OGA/Q and FEP respectively. Results from testing $GLP_{\tau}S$ on few of the above mentioned functions and, in addition, on functions *HE*, *Rosenbrock*, *Langerman*, *Shekel*[30], *Whitley*, and *LJ*, are given in sub-section 4.2.2. Similar testing was performed for DE in (Price *at al.*, 2005), where tuning of the parameters is carried out for each of the test functions. In order to choose the best settings, runs with different values of the method parameters are performed.

Results from further testing of *GLPτS* (with and without *LPτO* stage and parameters tuned individually for each function) on a few 30-dimensional cases are reported in Table 4.2. One can see from the table that when *LPτO* is included, the *GLPτS* method needs more function evaluations, but both the success rate and the accuracy are improved. It can also be seen that the success rate is increased by 12% on average (when *LPτO* is included), at the expense of about 15% additional function evaluations and the overall accuracy achieved from all runs (last column of Table 4.2) is improved, especially in the case of *Ackley* function.

- Comparison of the $GLP_{\tau}S$ Performance with GA and EP

The proposed in Section 3 $GLP_{\tau}S$ method is tested on a number of benchmark multimodal mathematical functions of 10 to 150 variables. The obtained results are compared in this section with GA and EP implementations. All tests are performed 50 independent times and the average results are given in Table 4.4 and Table 4.5. There is no tuning of the parameters to the specificity of a particular function in the tests reported in this sub-section. Some of the parameters of the $LP_{\tau}O$ algorithm, in regards

to problems' dimensionalities, are given in Table 4.3. For the rest of the parameters, the following values are adopted.

Table 4.3. Parameters of the *LPτO* method.

n	K	$[N_{\min}, N_{\max}]$	$\bar{N}^l_{\min}$, $\bar{N}^l_{\max}$
30	20	$[2^{11}, 2^{13}]$	$[2^6, 2^8]$
100	40	$[2^{14}, 2^{16}]$	$[2^{10}, 2^{12}]$
150	60	$[2^{16}, 2^{18}]$	$[2^{12}, 2^{14}]$

Parameters: n – dimensionality of the problem; K – maximal regions of interest on each iteration; $[N_{\min}, N_{\max}]$ – lower and upper bounds of the initial number of points; $\bar{N}^l_{\min}$, $\bar{N}^l_{\max}$ – lower and upper bounds of the number of points in each region of interest.

Table 4.4. Comparison of *GLPτS* with other GO methods in terms of mean number of function evaluations, mean function value and standard deviation in parentheses. There is no tuning of the method parameters for each function.

Method	Mean number of function evaluations / mean function value (std)			Analitic GM
Function (n)	*GPτS*	OGA/Q	FEP	
Schwefel II (30)	105673 / 3e-6 (1.2e-5)	112612 / 0 (0)	200000 / 8.1e-3 (7.7e-4)	0
Sphere (30)	117355 / 5e-8 (7.2e-9)	112559 / 0 (0)	150000 / 5.7e-4 (1.3e-4)	0
Ackley (30)	103229 / 9.2e-8 (1.1e-8)	112421 / 4e-16 (4e-17)	150000 / 1.2e-2 (2.1e-3)	0
Griewank (30)	101495 / 1.2e-2 (8.7e-3)	134000 / 0 (0)	200000 / 1.6e-2 (2.2e-2)	0
Schwefel I(30)	96600 / -12569.5 (3.8e-9)	302166 / -12569.5 (7e-4)	900000 / -12554.5 (52.6)	-12569.5
PLMI (30)	111884 / 5.2e-8 (7e-9)	134556 / 6e-6 (1.2e-6)	150000 / 9.2e-6 (3.6e-6)	0
PLMII (30)	122835 / 1.2e-3 (3e-3)	134143 / 1.9e-4 (2.6e-5)	150000 / 1.6e-4 (7.3e-5)	0
Rastrigin (30)	132183 / 5.7e-8 (5.4e-9)	224710 / 0 (0)	500000 / 4.6e-2 (1.2e-2)	0
Michalevicz (100)	233995 / -95.82 (1.33)	302773 / -92.83 (2.6e-2)	-	-99.278
NDTF (100)	205530 / -78.33 (9e-7)	245930 / -78.3 (6e-3)	-	-78.332

Methods: *GLPτS* – Genetic *LPτ* Search, proposed here; OGA/Q – Orthogonal Genetic Algorithm with Quantization by Leung and Wang (2001); FEP – Fast Evolutionary Programming by Yao *et al.* (1999).

Problems of 30-dimensions: number of initial points $I = 2^{14}$ for functions *Griewank* and *Schwefel I* (since the search interval is larger) and $I = 2^{11}$ for all of the rest. The GA population size is taken to be $G = 60$. Maximum number of GA generations is $100n = 3000$. Probability of recombination is $P_r = 0.1$ (on average, 10% of the current population undergo recombination) and the mutation probability is $P_m = 0.9$ (on

average, 90% of the current population undergo mutation). Usually, greater P_r and smaller P_m values are adopted in the literature, but here we employ the GA only for initial guidance into the search space and do not use it for finding a GM at this stage. Therefore, fast and broad space exploration is more valuable in *GLPτS* than slow and exploitative convergence to an optimal value. The parameter P_{LP} is taken to be 0.1, and depending on the function at hand, this determines 2 to 4 independent $LP_\tau O$ searches on average.

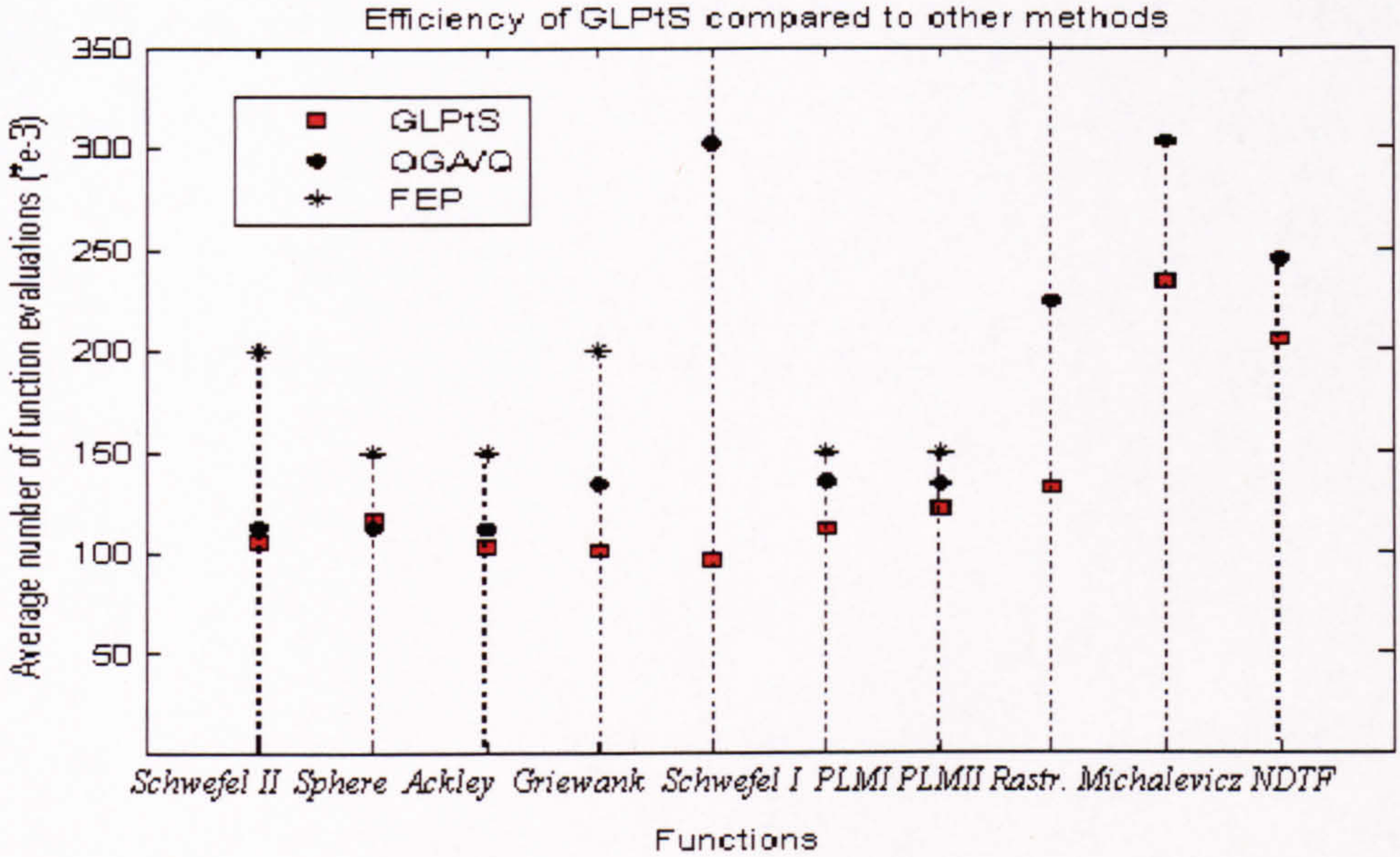


Figure 4.4. Average number of function evaluations for ten test functions needed by OGA/Q, FEP and *GLPτS*.

Problems of 100-dimensions: number of initial points $I = 2^{17}$ for *Griewank* and *Schwefel I*, and $I = 2^{15}$ for the rest. The GA population size is $G = 100$ and maximum number of GA iterations is assumed $100n = 10000$. All other parameters are the same as in the 30-dimensional cases.

Problems of 150-dimensions: number of initial points $I = 2^{19}$ for *Griewank* and *Schwefel I*, and $I = 2^{17}$ for the rest. The GA population size is $G = 120$ and maximum number of GA iterations is assumed $100n = 15000$. All other parameters are the same as in the previous cases.

Table 4.4 and Table 4.5 show test results for functions *Schwefel II*, *Sphere*, *Ackley*, *Griewank*, *Schwefel I*, *PLMI*, *PLMII*, *Rastrigin*, *Michalevicz*, *and NDTF* (with 30 and 100 dimensions, given in Appendix A). In Table 4.4, comparison with reported results

from other GA and EP implementations is given, in particular, with OGA/Q (Leung and Wang, 2001) and FEP (Yao *et al.*, 1999). Each of columns 2 - 4 shows average number of function evaluations separated by a forward slash from the mean value of the obtained solution and its standard deviation given in parentheses. Furthermore, Fig. 4.4 illustrates the average number of function evaluations needed for each of the three techniques to reach a GM. Table 4.5 shows the *GLPtS* performance for even higher dimensionalities of functions *Schwefel II*, *Sphere*, *Ackley*, *Griewank*, *Schwefel I*, *PLMI*, *PLMII*, *Rastrigin*, *Michalevicz*, *and NDTF*. We were not able to find testing results for such dimensions in the literature, therefore, comparison with other methods is not given for these cases.

Table 4.5. Results of *GLPtS* for functions of 100 and 150 dimensions. There is no tuning of the method parameters for each function.

	Analytic GM	Mean number of function evaluations / mean function value (std)		Analytic GM
Function	(n = 100)	n = 100	n = 150	(n = 150)
Schwefel II	0	529895 / 5e-3 (1.5e-2)	1021507 / 0.198 (0.15)	0
Sphere	0	638789 / 3.8e-3 (5.4e-3)	1218385 / 2e-2 (2.3e-2)	0
Ackley	0	561539 / 4.1e-3 (3e-3)	1038907 / 0.038 (2.41)	0
Griewank	0	563895 / 3.7e-2 (6.2e-2)	1313503 / 0.1232 (0.117)	0
Schwefel I	-41898.3	418545 / -41812 (102.14)	1107670 / -61998(372)	-62847.4
PLMI	0	644384 / 3e-6 (5e-6)	1251530 / 2e-5 (1e-5)	0
PLMII	0	684705 / 4.4e-4 (2.2e-3)	1308566 / 2.1e-3 (2.8e-3)	0
Rastrigin	0	715013 / 3.8e-4 (4e-4)	1374747 / 1e-2 (1e-2)	0
Michalevicz	-99.278	233995 / -95.82 (1.33)	539721 / -141.5 (3.33)	NA
NDTF	-78.332	205530 / -78.33 (9e-7)	986416 / -78.29(0.1689)	-78.332

The above experimental results show that the proposed *GLPtS* technique can be successfully applied for optimisation of continuous functions (no derivatives, Lipschitz estimates, or any additional information for the objective function are needed). This makes *GLPtS* applicable in cases where classical gradient and recent deterministic GO methods could not be efficiently used.

Empirically achieved, without tuning of the method parameters for each function, good results are reported in Table 4.4, Fig. 4.4, and Table 4.5. It can be seen from the mean and standard deviation values that, in most of the cases, a 100% success rate is achieved. Even in the cases of functions with many local minima, the technique successfully finds a global minimum. It can be also seen from Table 4.4 that for all testing multi-modal functions, the obtained mean function values are very close to the optimal ones. The standard deviation (given in parentheses in the tables) also shows low variance of the solution values, indicating a good stability of the results. Table 4.4 and Fig. 4.4 compare our results in terms of number of function evaluations with solutions obtained by (Yao *et al.*, 1999), using their Fast Evolutionary Programming (FEP) method, and by Leung and Wang's *orthogonal algorithm* OGA/Q (Leung and Wang, 2001). Fair comparison, based only on the number of function evaluations, is not possible because of the variance in the stopping conditions adopted by the different methods, and also sometimes there are hidden and auxiliary function calculations used for calibration of the methods' parameters. Nevertheless, the overall comparison with the other two methods demonstrates very competitive, stable and efficient results for our method in terms of both number of function evaluations and mean function values. For example, if we consider *PLMII* function (Table 4.4), the computational expense of our method is 122835 mean number of function evaluations, whereas for OGH/Q and FEP it is 134143 and 150000 respectively (although, it should be noticed that the accuracy of the other two methods is slightly better). Often, a method that performs better for particular functions (or group of functions) performs worse for others. This is the case with *Sphere* function, which is the only test function for which *GLPtS* is slightly more *expensive* than the OGH/Q method. In the cases of the 100 dimensional functions (*Michalevicz* and *NDTF*) we outperformed OGH/Q in terms of both accuracy and number of function evaluations, whilst Yao *et al.* (1999) did not report FEP results for these dimensionalities.

Table 4.5 presents results of tests with even higher dimensions – 100 and 150. The results show acceptable accuracy for an acceptable number of function evaluations, when compared with results from the 30 dimensional problems. However, further research of GO methods in general, and *GLPtS* in particular, is needed in order to reduce the computational load and make possible its use for optimisation of functions of even higher dimensionalities. As it can be seen from Table 4.5, the mean number of

function evaluations increases to over one million already for 150 dimensions, and for problems with higher dimensionalities the computational load is becoming very excessive.

- Comparison of the *GLPtS* Performance with DE

The proposed *GLPtS* method is further tested with a set of 10 to 30 dimensional functions used by Price *et al.* (2005), to investigate four different implementations of DE. In Table 4.6 and Table 4.7, average number of function evaluations per successful run (AES) are shown for variety of DE implementations and *GLPtS* method. For the cases with successful rate less than 100%, the number of successful runs (out of 10) is given in parenthesis. The best values achieved for each particular function are given in bold.

Particularly, for the 10 dimensional cases, Price *et al.* (2005) used rotated versions of the functions. Rotation is a technique used for transforming separable functions to parameter-dependent ones. The authors stated that GA usually have good performance on this set of functions, if not rotated, due to the use of low mutation rates, and poor performance on a set of rotated ones. In our experiment, the original non-rotated problems are used, but with very high mutation rate (usually, P_m= 0.9) and low recombination rate (usually, P_r = 0.1). Another major difference in DE implementations and *GLPtS* is the stopping condition. The one used by DE involves previous knowledge of the function minimum – the method stops when so-called VTR (Value To Reach) is obtained. On the contrary, *GLPtS* does not need any previous knowledge of the function minimum. Finally, the DE results for the 10 dimensional functions are obtained after exhaustive tuning of the parameters for each particular test problem. Table 4.6 shows the best possible performance of each of the four DE versions for some of the test functions. We performed only a moderate tuning of the algorithm parameters, trying about 10 different combinations for each test problem (compared with the total of 400 for all DE versions).

Because of the differences stated above, the results shown in Table 4.6 could be only used as indicative comparison of *GLPtS* and DE performances.

Table 4.6. Tests with 10 dimensional functions compared with DE.

Method	DE				*GLPtS*
Function	Rand	Best	Target-to-best	Either-or	
Ackley	11581	5395	6075	20051	11511
Griewank	22279	*31678 (1)*	*1139120 (4)*	19198	15375
Rastrigin	*421317 (2)*	*1295790 (1)*	*1658980 (2)*	490648	17541
Michalevicz	19531	12225	10756	13612	10221
Rosenbrock	44292	20897	21262	46614	6900
Langerman	56157	128728	122968	293159	111823
Shekel[30]	*1205800 (9)*	*176545 (2)*	*87404 (2)*	710499	150207
Whitley	177305	*369544 (4)*	*453481 (5)*	70381	*NA*
LJ	34592	10166	21550	59117	6775

Methods: Rand, Best, Target-to-best, and Either-or – four different implementations of DE proposed by Price *et al.* (2005); *GLPtS* – Genetic *LPτ* Search, proposed here.

Table 4.7. Tests with 30 dimensional functions compared with DE.

Method *Function*	Rand DE	*GLPtS*
Ackley	18741	46157
Griewank	14446	34210
Schwefel I	20691	14333
Rastrigin	118936	66729
Rosenbrock	115137	43186 (8)

Methods: Rand DE proposed by Price *et al.* (2005); *GLPtS* – Genetic *LPτ* Search, proposed here.

Table 4.7 shows the performance of *GLPtS* for a few 30-dimensional functions compared with DE. Unfortunately, only few tests were reported in Price *et al.* (2005), for 30 dimensional functions and only one version of DE was considered (*Rand*). Here, the original non-rotated functions are used.

As it can be seen in Table 4.6, for six of the nine tasks, our method converges faster than all four DE implementations. DE-rand outperforms *GLPtS* only for *Langerman* function. In the case of *Ackley* function our method is third, outperforming two of the

DE implementations. For the *Whitley* function, *GLPtS* was not capable to find a GO for acceptable number of function evaluations. For this function, *DE-best* and *DE-target-to-best* also experienced convergence difficulty (4 and 5 out of 10 successful runs respectively). *DE-either-or* and *DE-rand*, however, achieve very good speed and a 100% success rate. Our testing shows that for the *Whitley* function, the first stage of *GLPtS* (the GA stage) fails to obtain any acceptable function values. This is not surprising, since even the 2-dimensional version of the *Whitley* function is extremely steep in the interval [-100,-1]x[1,100], having values that vary between 10^{16} and 10^{1}. In the region [-1, 1]x[-1, 1], however, the function has a number of local minima where its values are between 0 and 10. Obviously, much greater population size and computational effort are needed for the GA, in order to start converging towards the area of the optimum. Comparison between *GLPtS* and *DE-rand* for five 30-dimensional functions is given in Table 4.7. The comparison shows that *DE-rand* performs better on the first two functions, whereas for the last three cases our method outperforms DE (in the case of *Rosenbrock* function, *GLPtS* has better speed, but achieved with lower success rate).

- Comparison of the *GLPtS* Performance with StGA

Stochastic Genetic Algorithm (StGA) that employs a novel *stochastic coding* strategy was proposed by Tu and Lu (2004) and already was mentioned in the review of Evolutionary Algorithms in Section 2.2.2. The results shown below were presented in a critical review, submitted for publication to *IEEE Transactions on Evolutionary Computations* (Jordanov and Georgieva, 2007b).

In Tu and Lu (2004), the method is discussed in detail and testing results are reported for more than twenty mathematical functions with different dimensionalities (from 2 to 100). Details about the functions can be can be found in Tu and Lu (2004) and here the same notations are adopted. Although StGA is similar to known stochastic techniques with *memetic* approaches (Joines and Kay, 2002; Hedar and Fukushima, 2003; Hart *et al.*, 2005), for all testing examples (with no exceptions), StGA outperforms all the other eight Evolutionary Algorithms (EA) used for a comparison. The difference in terms of efficiency (number of function evaluations) is dramatically in favour of the StGA method, reducing the computational effort sometimes more than a 100 times when compared with other techniques (Table 4.8). It

is also very dominant in the cases when parameter tuning of the other methods is performed individually for each testing function (Table 4.9). These results provoke our curiosity and raise some concerns about its outstanding superiority, so that the method is further investigated here and parallel simulations of StGA as proposed in Tu and Lu (2004) are performed (for this purpose, Matlab is employed).

Table 4.8. Comparison of FEP, *GLPtS* and StGA in terms of number of function evaluations. Function tuning is performed only in the case of StGA.

Method / *Function*	StGA	*GLPtS*	FEP
f_1	30 000	117 355	150 000
f_2	17 600	105 673	200 000
f_8	1 500	96 600	900 000
f_9	28 500	132 183	500 000
f_{10}	10 000	103 229	150 000
f_{11}	52 500	101 495	200 000
f_{12}	8 000	111 884	150 000
f_{13}	16 000	122 835	150 000

Methods: StGA – Stochastic Genetic Algorithm (Tu and Lu, 2004); FEP – Fast Evolutionary Programming (Yao *et al.*, 1999); *GLPtS* – Genetic *LPτ* Search, proposed here.
The functions f are given in Tu and Lu (2002).

Table 4.9. Comparison of DE, GLPtS and StGA in terms of number of function evaluations with tuning performed individually for each function.

Method / *Function*	StGA	*GLPtS*	DE
f_5	45 000	43 186	115 137
f_8	1 500	14 333	20 691
f_9	28 500	66 729	118 936
f_{10}	10 000	18 741	46 157
f_{11}	52 500	14 446	34 210

Methods: StGA – Stochastic Genetic Algorithm (Tu and Lu, 2004); DE – Differential Evolution (Price *et al.*, 2005); *GLPtS* – Genetic *LPτ* Search, proposed here.
The functions f are given in Tu and Lu (2002).

StGA Method

StGA could be described briefly as follows (Tu and Lu, 2004):

1. Initialise population with size *NP*.
2. Perform *local selection* for the current population.
3. Perform *global selection* for the current population.
4. Perform *recombination* with the selected individuals.
5. Perform *mutation* with probability P_m and compute the fitness function for the new individuals.
6. If the predefined number of iterations is not reached, repeat steps 2 to 6 with the new population.

The novelty in StGA is the *Stochastic Coding Mechanism* – individuals are not coded as points in the space with n coordinates, but as a region with mean and variance in each space direction. For example, one chromosome is given as

$$C_i = [(M_{i1}, V_{i1}), (M_{i2}, V_{i2}), \ldots, (M_{in}, V_{jn})],$$

where n is the dimensionality of the problem and $i = 1 \ldots NP$. M_{ij} is the mean value in each direction and V_{ij} – the corresponding variance. This coding plays a role only during the *local selection* stage. All other stages of StGA are quite trivial.

One peculiarity of StGA is that the authors used a flexible number of bits for the binary representation of the chromosomes for each test function.

Binary coding Implemented In StGA

For demonstrative purposes, Fig. 4.5 is considered, where two chromosomes from a parent generation are denoted with C_1 and C_2.

During the *local selection*, five points ($N = 5$, *asexual children*) are generated randomly in *stochastic* regions around points M_1 and M_2. The size of the regions corresponds individually to each chromosome C_i and is different for all n dimensions ($n = 2$ in Fig. 4.5). If any of the *asexually* produced children has better fitness value than the corresponding mean M_i, it takes the place of M_i (and the stochastic region shrinks *a bit*), otherwise, the stochastic region expands *a bit*. After the local selection phase, there is a global tournament selection, where the chromosomes are selected for the *mating pool*.

Let C_1 and C_2 be such two points, which are expected to undergo crossover to produce a new chromosome $\bar{C}_3$. As described by Tu and Lu (2004), and further clarified in our correspondence with the authors, the crossover in the mating pool has the following steps:

The mean points M_1 and M_2 (that correspond to the chromosomes C_1 and C_2) are coded as *integer* binary strings.

After one-point crossover of M_1 and M_2 and taking the average of the corresponding radii of the stochastic regions, a new chromosome $\bar{C}_3$ is produced with a mean $\bar{M}_3$ and stochastic region with $R_3^1 = (R_1^1 + R_2^1)/2$ and $R_3^2 = (R_1^2 + R_2^2)/2$.

Although M_1 and M_2 are points with real coordinates, when coded, their coordinates are transformed into integers. The authors claimed that their coding mechanism allows "a somewhat coarse division of the variable space without compromising the accuracy of the final solution", because "those points that are not covered by the binary string could also be approached by *StGA* through numerical sampling of the stochastic regions" (provided by the local selection phase). However might be argued that such an *integer* binary coding can lead to misinterpreting the actual search and can produce misleading results mainly due to the following two propositions:

Let the chromosomes C_1 and C_2, that are chosen for the *mating pool* during the tournament selection, have fitness values $F_1=F(M_1)$ and $F_2=F(M_2)$ respectively. After coding M_1 and M_2 as *integer* binary strings, two new points M_1^* and M_2^* (with integer coordinates) that most likely have two different fitness values – F^*_1 and F^*_2 will be obtained. These values will not necessarily be as *good* as the values of F_1 and F_2. Or, at least, can be said that the fitness values used in the selection procedure, do not correspond to the actual coded individuals. Therefore, the new solution $\bar{M}_3$ (obtained by one-point crossover of M_1^* and M_2^*) is most likely not to correspond and not to lead to an improved fitness value, which in turn puts a question mark about the method's convergence to a global minimum.

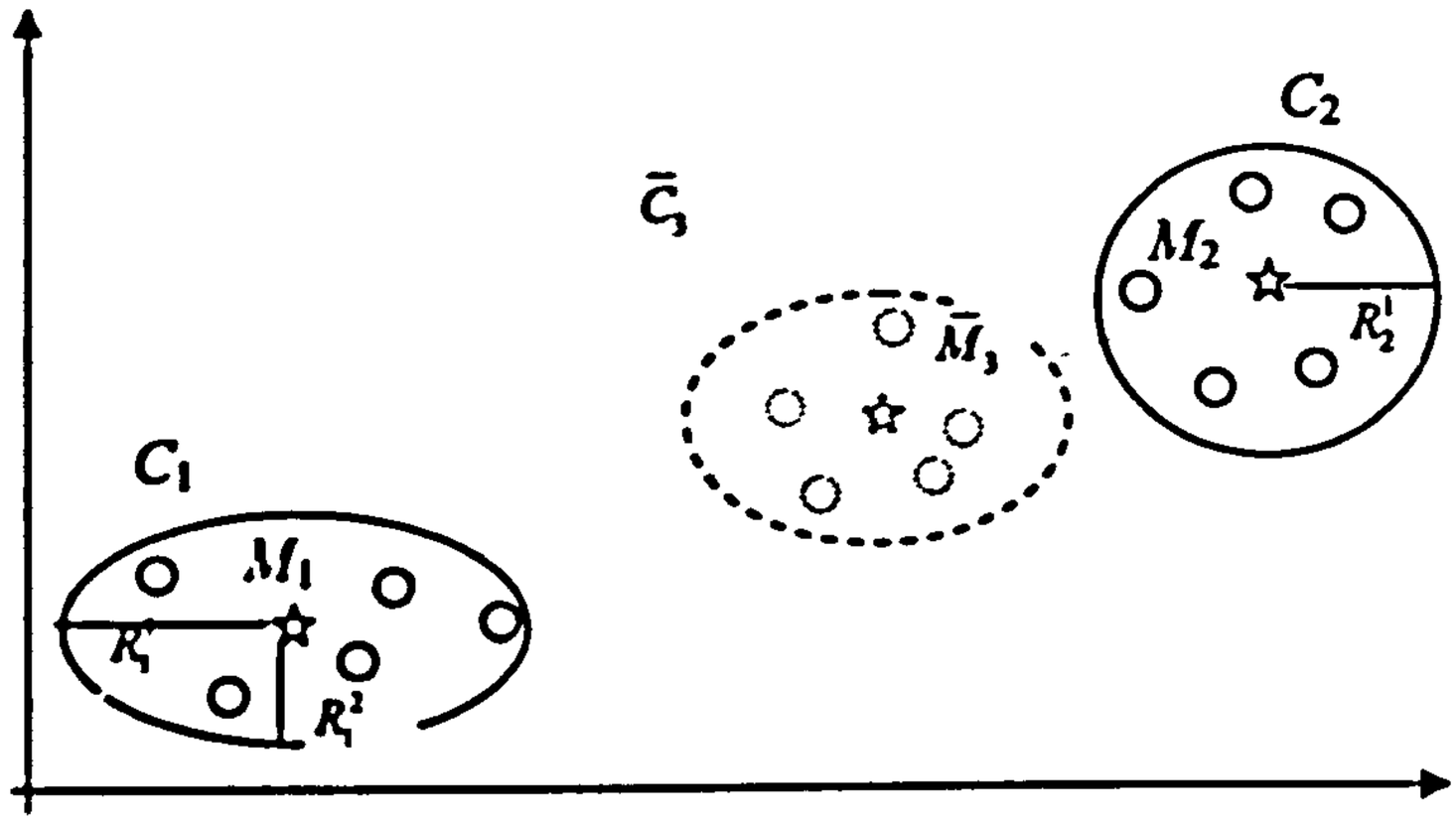


Figure 4.5. Illustration of StGA cross-over.

It is obvious that $\bar{M}_3$ will have coordinates with integer values (as it is generated as an integer binary string). In the next generation, during the local selection stage, $N = 5$ points will be generated in the stochastic region of point $\bar{M}_3$. However, even if a better point $\hat{M}_3$ ($F(\hat{M}_3) < F(\bar{M}_3)$) is encountered during this stage, and is selected to be a *representative* of the chromosome $\bar{C}_3$, when chosen for the *mating pool*, its coordinates will be transformed into integers as well. Therefore, it is likely (especially if the stochastic region is *small*) that the new point $\hat{M}_3$ will coincide with $\bar{M}_3$ after the coding. As stated by the authors in our correspondence: "the integer determined by each binary string corresponds to a real number in the physical space of the variable being represented". If we consider the cases of functions f_7, f_9 and f_{20}, where 8-bit binary presentation is used, the number of values (chromosomes) presented with this integer binary coding will be $2(2^7\text{-}1) + 1$ (assuming one bit is used for the sign). It is seen that not too many numbers (chromosomes) can be presented in this way and that the numerical grid is rough enough to miss vital optimisation points in the searched space.

Considering the last proposition, one can conclude that the size of a stochastic region plays very important role during the local selection (the size is given by the radii R_i^j, $j = 1, \ldots, n$ and $i = 1, \ldots, K$, where K is the population size). According to Tu and Lu (2004), radii R_i^j take random values in the region

$$R_i^V = (1/120 \sim 1/80)\,(B_{ui} - B_{li}) \qquad (4.4)$$

and can decrease or increase during the optimisation with an adaptation step

$$\delta_i = (2/100\text{~}5/100)\ V_i^R, \tag{4.5}$$

where V_i^R takes random value within R_i^r.

For demonstrative purposes, let us consider the *Rastrigin* function used as a testing function f_9 in Tu and Lu (2004). The global minimum is in the origin, $f_{min}(0, \ldots, 0) = 0$ and the function is investigated in the interval [-5.12, 5.12], $n = 30$. From (4.4) follows that $R_i^r = [0.853, 0.128]$ and the maximal possible value for $V_i^R = 0.128$. Therefore, the maximal possible interval for $\delta_i = (0.00256, 0.0064)$. After k iterations, providing the stochastic region is always expanding, the maximal possible value for the radii R_i^l will be $R_i^l = 0.128+0.128*k*0.0064$. Let us assume $k = 200$ (the authors used 190 iterations in total), then the maximal possible value for any radius of any stochastic region will be $R_i^l = 0.292$. Following our previous discussion, the coordinates of all asexually produced children when coded in a region with this radius will coincide with the closest integer. In the case of *Rastrigin* function, we argue that these values are 'very small' and the authors' coding implementation makes the local selection practically useless, since all *asexual children* will be too close to M_i

Counting the number of function evaluations

We believe that the number of function evaluations needed for StGA to optimise each test function were not correctly calculated in Tu and Lu (2004).

Let us consider the results, shown in Table II in Tu and Lu (2004). For example, the following information is given for function f_1 (Table 4.10). It is clear that the Mean Number of Function Evaluations (*MNFE*) is calculated as $(NP*NS)*NG$ (Table 4.10), which would give the number of function evaluations only for the local selection stage, but not for the computation of each new generation and also excludes the number of initial points. The actual process count, in our view, should be:

$$NP + NP*NS + NP + NP*NS + NP + \ldots + NP*NS + NP.$$

Therefore, the total number of function evaluations should be:

$$NP + NG*(NP*NS+NP).$$

For function f_1, this would give 39030 number of function evaluations in total, instead of 30000 (as reported in Tu and Lu (2004)). However, even if counting the number of function evaluations was correctly done, there is still significant difference between the performance of StGA and the other Evolutionary Algorithms used for a comparison.

Table 4.10. Counting the number of function evaluations in StGA.

Function	*NP*	*NS*	*NG*	*MNFE*
f_1	30	5	200	30 000

NP - population size; *NS* - number of children produced during local selection in each region; *NG* - number of generations; *MNFE* - mean number of function evaluation.

Simulation of StGA and comparison of results

We simulated in Matlab the proposed by Tu and Lu StGA method and evaluated our implementation on the same testing functions and used the same parameter values as given by the authors (Table II and Table VII in Tu and Lu (2004)).

Here, in the second column of Table 4.11 we show the accuracy (mean value and standard deviation given in parentheses) for each function reported in Tu and Lu (2004); in the third column - the analytic minimum that we want to achieve; in the fourth column - the accuracy results (mean and standard deviation) achieved by our simulation of StGA; in the fifth column - the number of successful runs out of 20; and in the last two columns we give the best and worst optimisation values achieved in those 20 runs. Results for *Schwefel II* since are not reported since boundary constraints in our simulation were not implemented. The stopping criterion is the number of function evaluations given in Table II in Tu and Lu (2004), as it is expected, that the method will converge to a global minimum for this expense (as reported by the authors). As it can be seen from Table 4.11, the StGA method failed to converge in most of the cases (with the exception for functions f_{14} to f_{17}).

Table 4.11. Comparison of the accuracy results reported in Tu and Lu (2004) and our implementation of StGA. The same parameter values are used in both cases.

Method Function	StGA in Tu and Lu (2004)	Analytic Min	StGA – our simulation	Successful Runs	Min	Max
f_1	2.45_x10^{-15} (5.25_x10^{-16})	0	2.54_x10^{+3} (741.58)	0/20	505.31	3.9_x10^{+3}
f_2	2.03_x10^{-7} (2.95_x10^{-8})	0	17.6 (18.67)	0/20	6	93.875
f_3	9.98_x10^{-29} (6.9_x10^{-29})	0	6.8_x10^{+5} (1.86_x10^{+5})	0/20	4.5_x10^{+5}	1.2_x10^{+6}
f_4	2.01_x10^{-8} (3.42_x10^{-9})	0	25.47 (3.76)	0/20	17.97	32.84
f_5	0.04435 (0)	0	563.37 (686.37)	0/20	54.05	2.3_x10^{+3}
f_6	0.0	0	3.58_x10^{+3} (465.8)	0/20	2.65_x10^{+3}	4.6_x10^{+3}
f_7	8.4_x10^{-4} (1.0_x10^{-3})	0	2.31_x10^{-4} (2.1_x10^{-4})	20/20	6.4_x10^{-5}	9.5_x10^{-4}
f_9	4.42_x10^{-13} (1.14_x10^{-13})	0	16.155 (23.95)	9/20	0	101.7
f_{10}	3.52_x10^{-8} (3.51_x10^{-9})	0	16.33 (2.8)	0/20	8.593	18.85
f_{11}	2.44_x10^{-17} (4.54_x10^{-17})	0	145.29 (28.99)	0/20	98.65	214.25
f_{12}	8.03_x10^{-7} (1.96_x10^{-14})	0	173.34 (34.97)	0/20	121.28	275.38
f_{13}	2.01_x10^{-8} (3.42_x10^{-9})	0	3.8_x10^{+3} (1.34_x10^{+4})	0/20	266.45	6_x10^{+4}
f_{14}	1.0 (0.0)	1	0.998 (7_x10^{-8})	20/20	0.998	0.998
f_{15}	3.1798_x10^{-4} (4.726_x10^{-6})	3.08_x10^{-4}	5.49_x10^{-4} (2.8_x10^{-4})	20/20	3.2_x10^{-4}	0.0012
f_{16}	-1.03034 (1.00_x10^{-3})	-1.0316	-1.0316 (4.53_x10^{-6})	20/20	-1.0316	-1.0315
f_{17}	0.3986 (6.00_x10^{-4})	0.398	0.3979 (2.54_x10^{-5})	20/20	0.3979	0.3980
f_{18}	-9.828 (0.287)	-10.1532	-4.28 (2.71)	3/20	-10.1532	-2.63
f_{19}	-10.40 (0.0)	-10.4029	-6.01 (3.71)	8/20	-10.403	-1.8371
f_{20}	-10.450 (0.037)	-10.5363	-5.21 (3.21)	5/20	-10.5363	-1.8589
f_5^{100}	1.01 (1.1959)	0	6.8_x10^{+6} (3_x10^{+6})	0/20	2.65_x10^{+6}	1.22_x10^{+7}
$g(x)$	-78.29368 (0.03)	-78.3322	-56.6687 (1.57)	0/20	-60.23	-53.74

The functions f_i and g are given in Tu and Lu (2002).

The results from our StGA implementation can be summarised as follows:

1. For the 100 dimensional cases (f_5^{100} and $g(x)$) the StGA simulation (using the proposed by the authors parameter values) produced much worse than the reported in Tu and Lu (2004) results, and we were not able to achieve even a single successful run.

2. For the 30 dimensional test functions (f_1 - f_{13}), we achieved successful runs for two functions only (about 50% success rate for function f_9 and 100% success rate for f_7). For all other functions the method simply failed to converge for the specified number of function evaluations. The obtained results were not even close to the ones reported in Tu and Lu (2004). Our hypothesis is that the accidental success for functions f_7 and f_9 is due to the small number of binary bits used for them – 8, as well as the position of the minima – in the origin (0, ..., 0). Also the small search intervals for those functions and the way binary coding works helped StGA to converge to a global minimum in both of those cases.

3. For the cases of 4 dimensional test functions ($f_{18} - f_{20}$), the results of StGA simulation looked a bit better, with an average of 25% percent successful runs, but were still far from the ones reported in Tu and Lu (2004). For one of the four-dimensional cases – function f_{15}, our results were comparable with those given in Tu and Lu (2004), (but with worse accuracy).

4. For all two-dimensional cases (functions f_{14}, f_{16} and f_{17}) we received very close to the published in Tu and Lu (2004) results (in the cases of functions f_{16} and f_{17}, they were even slightly better).

The results of our simulation of StGA method lead to a conclusion that the accuracy and convergence characteristics (such as number of function evaluations for reaching a global minimum) significantly differ from those reported in Tu and Lu (2004). For most of the test cases (with the only exception for the two-dimensional test functions) the method simply failed to reach a global solution with the specified accuracy and for the specified number of function evaluations. The authors rightly claim their stochastic coding strategy as a novelty, but in our view, their implementation of integer binary coding of chromosomes limits significantly the power of their search

approach. We belief the method either accidentally hits a global solution (mainly, because most of the test functions are with global minimum in the origin), or their software implementation contains a programming error that produces these exceptionally superior results.

4.3 Results from *GLPτS* Applied for NN Training

The results reported in this section have been published in Georgieva and Jordanov (2006).

Here, the proposed technique is tested on five known benchmark problems with different dimensionalities (last three problems are from the UCI repository database). For comparison, a standard BP (Levenberg-Marquardt) is also performed using Matlab NN Toolbox. Both methods were ran 50 times and their average values are reported below. In Tables 4.8 – 4.12, the *Error Function* column shows the obtained least-squared error value and the standard deviation (in parentheses), and the *Mean Test Error* column demonstrates the NN generalisation abilities. Finally, one function approximation example is conducted, and the obtained results are compared with results from other methods reported in the literature (Table 4.13).

- Classification of XOR Problem

For the classification of the XOR problem discussed in details in Chapter 2 and Chapter 3, the same NN topology and testing strategy as in Section 3.1.4 and Section 3.2.3 are adopted.

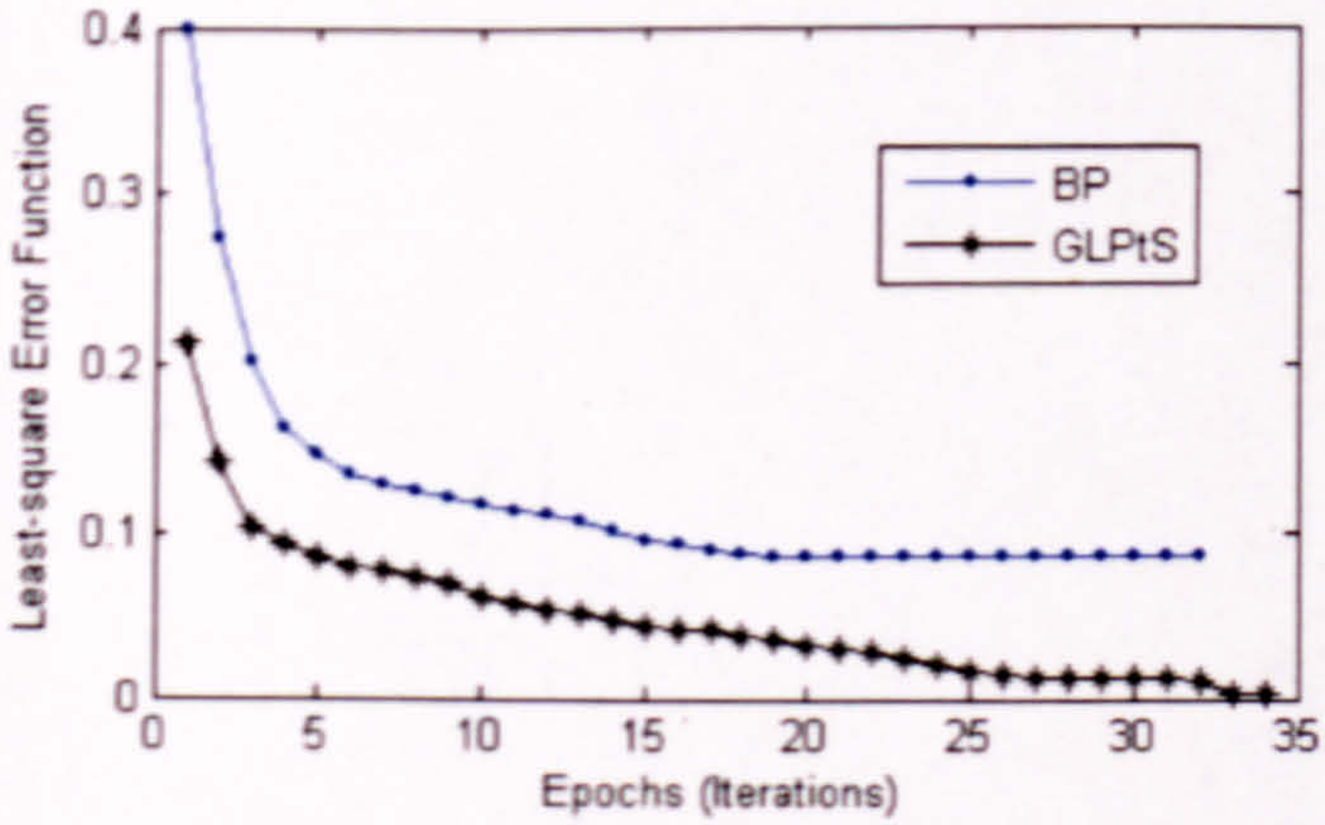


Figure 4.5. Error function for the XOR problem when BP and *GLPτS* are used.

Table 4.12. Optimal errors for the *GLPtS* and BP (XOR problem).

Criterion / Method	Error Function (Std. Dev.)	Mean Test Error (Std. Dev.)
BP	0.08 (0.09)	0.1987 (0.0290)
GLPtS	7.6e-08 (7e-08)	8.3e-07 (3.3e-7)

Method: BP – Backpropagation with Levenberg-Marquardt optimisation (the source of Matlab 7 is used).

The process of error function minimisation (NN training) is illustrated in Fig. 4.5, where *GLPtS* is compared with BP. It can be seen from the figure that after the 20^{th} epoch, BP does not improve the error function, while our method continues to minimise it. To assess the ability of the trained NN to generalise, tests with 100 random samples of noisy data are performed, where the noise is up to 15% (e.g., (0.15, 0.85) and (0.3, 1.0) are both samples with 15% noise with respect to the training sample (0.0, 1.0)). Obtained optimal results from the training and testing are given in Table 4.12.

- Classification of *N-Parity* Problem

For the classification of the *N-Parity* problem the same NN topology and testing strategy is as in Section 3.1.4 and Section 3.2.3 adopted.

Obtained optimal results from the training and testing with 100 testing samples (again with a 15% noise) along with the BP ones are shown in Table 4.13.

Table 4.13. Optimal errors for the *GLPtS* and the BP (4-*Parity* problem).

Criterion / Method	Error Function (Std. Dev.)	Mean Test Error (Std. Dev.)
BP	0.0219 (0.05)	0.1229 (0.0902)
GPtS	0.0069 (0.009)	0.0614 (0.156)

Method: BP – Backpropagation with Levenberg-Marquardt optimisation (the source of Matlab 7 is used).

- Classification of *Iris* Problem

Iris classification problem was considered in detail in Section 3.2.3 of Chapter 3. Here, the same topology and parameters are used. The obtained results, in terms of training and testing errors, are given alongside of the BP ones in Table 4.14.

Table 4.14. Optimal errors for the *GLPτS* and BP (*Iris* problem).

Method	Criterion Error Function (Std. Dev.)	Mean Test Error (Std. Dev.)
BP	0.0091 (0.05)	0.042 (0.078)
GLPτS	0.00097 (0.00056)	0.029 (0.073)

Method: BP – Backpropagation with Levenberg-Marquardt optimisation (the source of Matlab 7 is used).

- Predicting the rise time of a servo mechanism

Servo predicting problem was considered in detail in Section 3.2.3 of Chapter 3. Here the same topology and parameters are used. The obtained results, in terms of training and testing errors, are given alongside of the BP ones in Table 4.15.

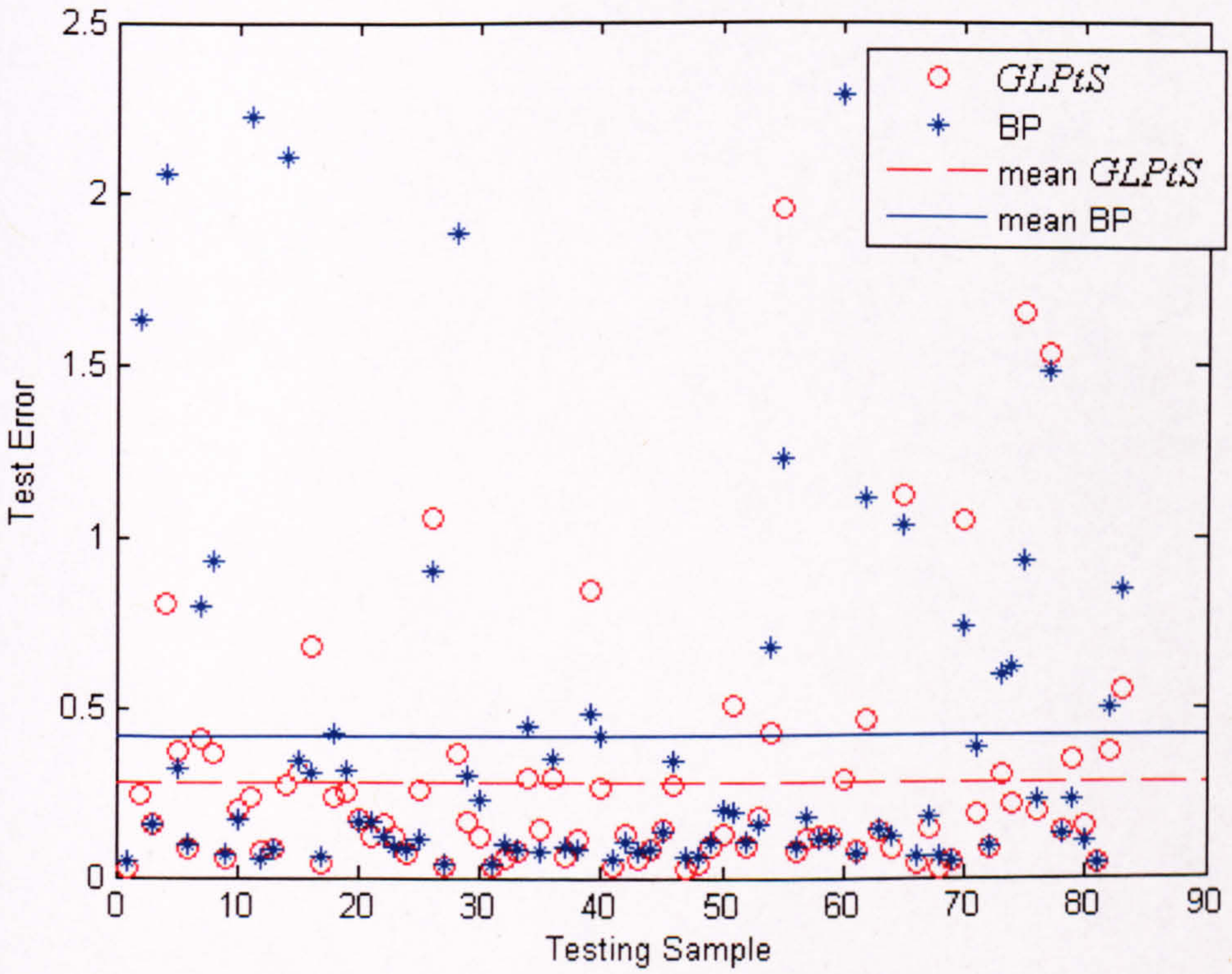


Fig. 4.6. Test errors and mean test errors for BP and *GLPτS*.

Table 4.15. Optimal errors for the *GLPτS* and BP (*Servo* problem).

Criterion / Method	Error Function (Std. Dev.)	Mean Test Error (Std. Dev.)
BP	0.0474 (0.06)	0.4171 (0.5515)
GLPτS	0.0245 (0.005)	0.2841 (0.4448)

Method: BP – Backpropagation with Levenberg-Marquardt optimisation (the source of Matlab 7 is used); DE – Differential Evolution, the source provided by Price *et al.* (2005), is used.

Average values of the errors for each testing sample for both BP and *GLPτS* are given in Fig. 4.6. One can see that there are more outliners (greater test errors) in the case of BP and that overall smaller mean test value is achieved by the *GLPτS* method.

- Classification of *Pima Indians Diabetes* Database

In the *Diabetes* data collection, the investigated, binary-valued variable is used to diagnose whether a patient shows signs of diabetes or not (Rocha *et al.*, 2003). All patients are females of at least 21 years old and of Pima Indian heritage. The data set comprises 500 instances that produce an output 0 (non-positive for diabetes), and 268 with output 1 (positive for diabetes). Each sample has 8 attributes: number of times pregnant, age, blood test results, etc. In order to avoid computational inaccuracies, in our experiment all attributes were normalised to have a zero mean and a unit standard deviation.

A network with 8-8-1 architecture (81-dimensional problem) was adopted to produce continuous output in the range [0, 1]. The dataset was divided into two parts – training one of 384 samples, from which 145 correspond to output 1, and the other subset of 384 patterns was used for testing. Table 4.16 shows the obtained optimal solutions for the training and testing errors.

Table 4.16. Optimal errors for the *GLPτS* and BP (*Diabetes* problem).

Criterion / Method	Error Function (Std. Dev.)	Mean Test Error (Std. Dev.)
BP	0.0764 (0.07)	0.2831 (0.2541)
GLPτS	0.001 (0.005)	0.2619 (0.3861)

Method: BP – Backpropagation with Levenberg-Marquardt optimisation (the source of Matlab 7 is used); DE – Differential Evolution, the source provided by Price *et al.* (2005) is used.

- Function Fitting Regression Example

We performed a function fitting example, where the network is trained with noisy data. The function to be approximated is the Hermit polynomial:

$$G(x) = 1.1(1 - x + 2x^2)\exp\left(-\frac{x^2}{2}\right). \tag{4.6}$$

We conducted the same experiment as in Leung *et al.* (2001), with the only difference that we used batch-mode instead of on-line training.

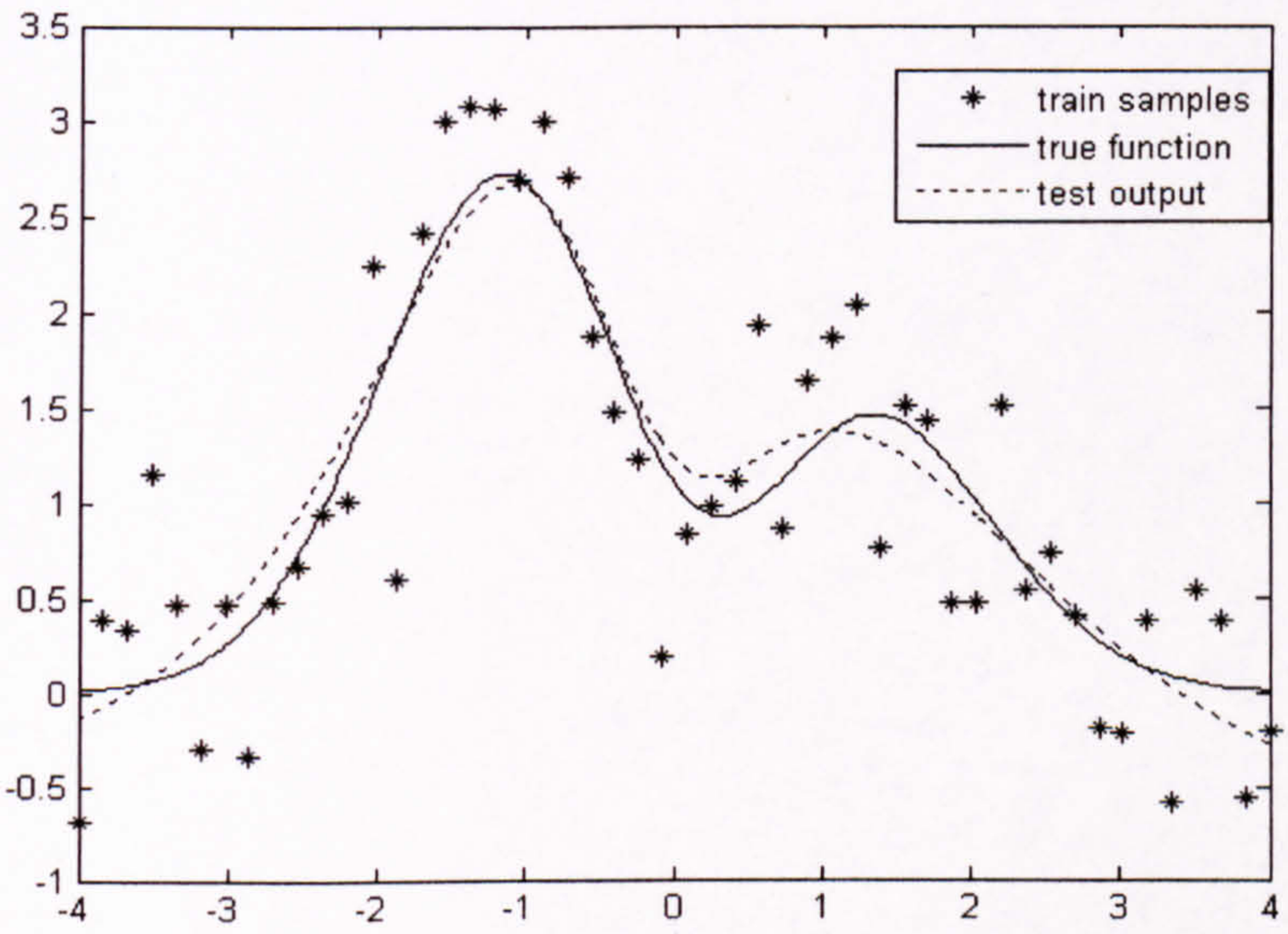


Figure 4.7. Output of the network trained with *GLPτS* for the function fitting example.

We also have to notice that the function equation given in Leung *et al.* (2001), does not correspond to their graphics and is slightly different from (4.6). The test results from 2000 testing samples and 20 independent runs of the experiment are shown in Table 4.17. It can be seen from the table that our results improve slightly the best ones given in Leung *et al.* (2001). Fig. 4.7 graphically illustrates the results and shows the Gaussian noise that we used for training, the function to be approximated, and the NN output.

Table 4.17. Test results for the *GLPτS* and the methods in Leung *et al.* (2001).

Criterion / Method	Average	Max	Min	Std. Dev.
RLS	0.1901	0.2567	0.1553	0.0259
IPRLS	0.1453	0.1674	0.1207	0.0076
TWDRLS	0.1472	0.1711	0.1288	0.0108
GLPτS	0.1349	0.1602	0.1184	0.01

Method: By Leung *et al.* (2001): RLS – Recursive Least Squares; IPRLS – Input Perturbation RLS; TWDRLS – True Weight Desay RLS.

Discussion

In this section *GLPτS* is applied for training NN to solve correctly four classification problems (9 to 81 dimensions), one 25-dimensional problem for predicting continuous output, and one 46-dimensional function fitting and approximation task. The obtained training and testing results (Tables 4.8-4.12) show that our method converged successfully to a solution in every experiment. For comparison, BP is also performed in parallel, producing inferior results.

For the classification experiments, the *GLPτS* least-square error is at least two times smaller than the one achieved by BP. Multiple independent runs of our method show that the obtained solutions are also stable. In the case of XOR, the *GLPτS* method outperformed BP considerably. BP produced mean error of 0.08 in comparison with 7.6e-8 for the proposed here method (Table 4.8). For this task Wang *et al.* (2004), also reported low success rate for BP with frequent entrapment in local minima. In the case of *Servo* problem the superiority of our method is not so dominant as in the case of XOR, but still shows better standard deviation of both measures – 0.005 against 0.06 for the error function, and 0.44 against 0.55 for the test error (Table 4.11). This indicates a better and more stable solution for our method. In Rocha *et al.* (2003), the reported results from five different methods for the same task and architecture are also with worse error function values compared to ours. Those observations indicate that further improvement of the solution could not be found for the investigated 4-4-1 NN architecture. Experiments with different architectures could lead to better results, as stated in Binachini and Gori (1996). The comparison of the training results for *Iris* and

Diabetes data sets, given in Rocha *et al.* (2003), also confirm the advantages of the *GLPτS* method.

The obtained testing results show that the generalisation abilities of the NN trained with the *GLPτS* are better, although, in two of the experiments our test results are less stable than those of BP. For example, in the case of XOR a very small mean test error is observed for *GLPτS* and its deviation is of order 1.0e-7, compared to order 1.0e-2 for BP (Table 4.8). However, in the cases of *4-Parity* and *Diabetes*, where again better mean test errors are reported for *GLPτS*, the standard deviation shows less stable solutions. This might be due to over-fitting, since the training errors are very small, but better training error not always guarantees better generalisation ability. To test this, in future experiments, we intend to use additional validation subset during the training. In the case of function fitting, the generalisation abilities of the NN, trained with our method, slightly improved the best results reported in Leung *et al.* (2001).

4.4 Summary

A hybrid Global Optimisation *GLPτS* method that combines Genetic Algorithms, *LPτO* search and Nelder-Mead simplex search, has been proposed in this chapter. By utilising GA, the investigated technique aims to handle the problems that *LPτO* experiences when used for the optimisation of higher dimensional functions (30 to 150). The simplex search has been employed at the final phase of the optimisation, in order to refine the solution found at the previous stage. The proposed *GLPτS* hybrid heuristic has been tested on a number of benchmark mathematical functions and has shown very reliable performance. When compared with Genetic Algorithms, Evolutionary Programming, and Differential Evolution, the investigated and proposed method has demonstrated strongly competitive results in terms of both number of function evaluations and success rate. In addition, detailed critical review of Stochastic Genetic Algorithm proposed by Tu and Lu (2004) is presented. Subsequently, *GLPτS* has been applied for supervised NN training and tested on a number of benchmark problems. Based on the reported and discussed findings, it can be concluded that the investigated and proposed *GLPτS* technique is very competitive and demonstrates reliable performance.

5 Image Texture Analysis and Automated Inspection of Cork Products

This chapter is introductory to the case study (automated inspection of cork tiles with Intelligent Computer Vision System, presented in Chapter 6) of this thesis. Here, some basic concepts of Machine Vision applied for the image texture analysis are introduced in Section 5.1. A couple of classical texture extraction methods are presented in detail – Co-occurrence matrices (Section 5.1.2) and Laws' Masks (Section 5.1.3). Subsequently, in Section 5.2, motivation of the case study given Chapter 6 is presented. The way cork is harvested and processed into cork tiles is discussed and useful properties of cork tiles are outlined in the next section. Section 5.3 provides a review of work that has been conducted to date, concerning the automated inspection of cork products (planks and stoppers).

Computer Vision for product quality control automation of assembly line has been of interest for researches in the last twenty years (Graves and Batchelor, 2003). One very early survey on automated visual inspection in industry is presented by Chin (1988) and includes more than 600 references. Davies (2005) presents a detailed and broad comprehensive study of *state-of-the-art* Intelligent Computer Vision methods ranging from low-level vision to real-time Pattern Recognition systems. One branch of the Machine Vision research is the texture feature extraction which provides methods of mathematical characterisation of texture images in a suitable way (considered in the following Section 5.1).

5.1 Texture Feature Extraction

Texture is defined as the variation of intensity (or variation of colour) in the image. There are five major categories of features for texture classification: statistical; geometrical; structural; model-based; and signal processing features (Randen and Husøy, 1999). A general overview of the feature extraction approaches is given in the following Section 5.1.1. For the automated inspection of cork tiles, presented in Chapter 6, we use two classical techniques (one statistical and one signal processing method) that have been successfully applied for numerous tasks. They are discussed in the following Section 5.1.2 and Section 5.1.3.

Before applying any feature extraction technique to the images, it might be useful to apply a image processing technique that aims to remove artifacts caused by uneven lighting by subtracting a local average from every pixel, for example, a 15x15 size window (Shapiro and Stockman, 2001; Umbaugh, 2005). The window is moved across the image, the average gray level value is calculated and subtracted from the current pixel in the center of the window. The output is stored in another new image, so, that the current image is not overridden. The new image created in this way has average local gray levels close to zero (Umbaugh, 2005).

5.1.1 Literature Review of Texture Feature Extraction Techniques

Randen and Husøy (1999) provide a very broad and popular study of feature extraction methods. The study deals mostly with filtering approaches that are generally structured as follows:

The images are submitted to a linear transform, filter, or a filter bank, followed by some *energy measure* to form so-called *energy feature images*. The pixel values of them are then combined to form the elements of the feature vectors. Randen and Husøy (1999) consider Laws' masks (Section 5.1.3), ring and wedge filters, Wavelet transform, discrete cosine transform, quadrature mirror filters, eigenfilters, Gabor filters, and several others. They also employ two classical non-filtering approaches – the Co-occurrence matrices (Section 5.1.2) and model based feature extractors. All techniques are tested on a number of image sets. The authors conclude that there is not an overall *winner* and try to provide a general guidance for what approach might be suitable for a certain task.

A detailed overview of most types of discrete transforms suitable for filtering is provided in Umbaugh (2005). In this book, Fourier, cosine, Walsh-Hadamard, Haar, and Wavelet transforms are considered in detail and comprehensive examples are provided.

On the other hand, Liu and Wang (2003) propose integration of filter responses instead of energy mapping. The authors claim that indeed the specific form of the filters is not crucial.

Ojala and Pietikäinen (2002) propose and investigate a texture feature extraction method that is statistically based and is claimed to have several useful properties –

rotation invariance, computational efficiency, and robustness in terms of gray-scale variations. The method is called *Local Binary Patterns* and its performance is demonstrated with two problems showing promising performance.

In Theodoridis and Koutroumbas (2006) many interesting feature extraction approaches (first and second order statistics, parametric models, etc.) are considered besides the linear filtering transforms like the Fourier, cosine and sine, Haar, Wavelet, etc.

For our experiments, reported in Chapter 6, we selected two classical feature extraction methods – one filtering approach (Laws' masks) and one statistical approach (Co-occurrence matrices). These methods have been considered and tested on various problems and have proven their abilities (Umbaugh, 2005; Davies, 2005). Since the aim of this research is to further improve and investigate NN learning mechanisms, the feature extraction stage is kept to the basics and other feature extraction techniques are not used in the experiment. The Laws' masks and Co-occurrence matrices are presented in more detail in the following sections. Some other related books, that consider feature extraction techniques, and in particular, Laws' masks and Co-occurrence matrices, include Shapiro and Stockman (2001); Davies (2005); Sonka *et al.* (2007).

5.1.2 Co-occurrence Matrices (Gray-Tone Spatial Dependence Matrices)

One introductory and basic example of the idea of extracting *texture features* is the work of Haralick *et al.* (1973). In it the authors proposed the use of Gray-Tone Spatial Dependence Matrices (or also called *Co-occurrence matrices*) and 14 different measures (features) based on these matrices. According to Davies (2005) this approach became to a large degree a standard one for texture analysis during the 1970s. Haralick's measures were successfully used for classification of different materials, i.e. wood, corn, grass, etc. (Davies, 2005).

Easily computable texture measures are used to utilise the gray-tone spatial dependencies in this method. These features are based on second-order statistics and encounter for the relative distances between pixels with the same grey level and their relative orientation (Theodoridis and Koutroumbas, 2006). All of them are based on

the assumption that the texture information of an image is contained in the overall or *average* spatial relationship, which the gray tones in the image have to one another.

A digital image consists of so called pixels (picture cells). To each pixel corresponds a gray-intensity (that usually ranges between 0 and 255) and the picture is represented in the computer as a two-dimensional matrix. Each entry in the matrix represents the gray value of the pixel positioned at the corresponding place in the picture. Let us consider one easy example introduced by Haralick *et al.* (1973). Given is the matrix corresponding to a 4x4 pixels in a picture. The gray value is an integer that takes values between 0 and 3 (Fig. 5.1).

0	0	1	1
0	0	1	1
0	2	2	2
2	2	3	3

Figure 5.1. 4x4 image with four gray-tone values (ranging 0-3).

Table 5.1. General form of any co-occurrence matrix for images with gray-tone values in 0-3.

	0	1	2	3
0	#(0,0)	#(0,1)	#(0,2)	#(0,3)
1	#(1,0)	#(1,1)	#(1,2)	#(1,3)
2	#(2,0)	#(2,1)	#(2,2)	#(2,3)
3	#(3,0)	#(3,1)	#(3,2)	#(3,3)

For an image with 0-3 gray-tone values, the general form of the co-occurrence matrix is given in Table 5.1, where $\#(i, j)$ stands for the number of times that gray-tones i and j are *neighbours*. Four different Co-occurrence matrices can be generated – horizontal, vertical, left-diagonal, and right-diagonal. They depend on the definition of a *neighbour*. When neighbours are considered to be the horizontal neighbours (for example (i, j) and $(i, j + 1)$), the matrix is called *horizontal* and when the neighbours are vertical (as (i, j) and $(i + 1, j)$), it is called *vertical* matrix. Neighbours of the type

(i, j) and $(i + 1, j + 1)$, and (i, j) and $(i + 1, j - 1)$ correspond to a left- and right-diagonal matrices respectively. For the example from Fig. 5.1, the four different matrices are:

$$P_H = \begin{pmatrix} 4 & 2 & 1 & 0 \\ 2 & 4 & 0 & 0 \\ 1 & 0 & 6 & 1 \\ 0 & 0 & 1 & 2 \end{pmatrix}, \qquad P_V = \begin{pmatrix} 6 & 0 & 2 & 0 \\ 0 & 4 & 2 & 0 \\ 2 & 2 & 2 & 2 \\ 0 & 0 & 2 & 0 \end{pmatrix},$$

$$P_{LD} = \begin{pmatrix} 2 & 1 & 3 & 0 \\ 1 & 2 & 1 & 0 \\ 3 & 1 & 0 & 2 \\ 0 & 0 & 2 & 0 \end{pmatrix}, \quad \text{and} \quad P_{RD} = \begin{pmatrix} 4 & 1 & 0 & 0 \\ 1 & 2 & 2 & 0 \\ 0 & 2 & 4 & 1 \\ 0 & 0 & 1 & 0 \end{pmatrix},$$

where H stands for horizontal, V for vertical, LD and RD stand for left- and right-diagonals. Note that the dimensionality of the Co-occurrence matrices depends on the gray scale range, which is usually 0-255 which results in the need of large storage space. However, the symmetry of the matrices could be used to reduce this necessity. Based on these gray-tone scale dependence matrices, 14 different texture measures are proposed by Haralick *et al.* (1973). Five of them are most commonly used and considered as *standard* by Shapiro and Stockman (2001) after subsequent research had shown that the others are highly correlated (Randen and Husøy, 1999; Sonka *et al.*, 2007).

- *Angular Second Moment* (or also called *Energy*)

$$f_1 = \sum_{i=1}^{N_g} \sum_{j=1}^{N_g} \left(\frac{P(i,j)}{R} \right)^2, \tag{5.1}$$

where N_g is the dimensionality of the matrix, or the number of possible gray-values. R is a normalisation constant. The value of f_1 is a measure of the homogeneity, or the *smoothness* of the image.

- *Contrast*

$$f_2 = \sum_{n=0}^{N_g - 1} n^2 \left\{ \sum_{|i-j|=n} \left(\frac{P(i,j)}{R} \right) \right\}. \tag{5.2}$$

This is a measure of the amount of local variations present in the image.

- *Correlation*

$$f_3 = \frac{\sum_{i=1}^{N_g}\sum_{j=1}^{N_g}[ijP(i,j)/R] - \mu_x\mu_y}{\sigma_x\sigma_y}, \qquad (5.3)$$

where μ_x, μ_y, σ_x, and σ_y are the means and the standard deviations corresponding to the distributions $p_i^{(x)} = \sum_{j=1}^{N_g} P(i,j)$ and $p_j^{(y)} = \sum_{i=1}^{N_g} P(i,j)$. This feature is a measure of gray-tone linear-dependencies in the image.

- *Homogeneity*

$$f_4 = \sum_{i=1}^{N_g}\sum_{j=1}^{N_g}\frac{P(i,j)}{1+|i-j|}. \qquad (5.4)$$

This feature measures the closeness of the distribution of elements in the matrix to the diagonal and is adopted in and Stockman (2001) and the Matlab implementation. This feature is also called *Inverse Difference Moment*, where instead of the absolute difference, the squared difference is taken, as originally proposed by Haralick *et al.* (1973). In our work we use the four features given above as the most important features (also adopted in Matlab).

One other feature that is often given by authors (Haralick *et al.*, 1973; Randen and Husøy, 1999; Umbaugh, 2005; Theodoridis and Koutroumbas, 2006; Sonka *et al.*, 2007, and others) is:

- *Entropy*

$$f_5 = -\sum_{i=1}^{N_g}\sum_{j=1}^{N_g} P(i,j)\ln(P(i,j)). \qquad (5.5)$$

This feature is a measure of randomness and takes low values for smooth images.

5.1.3 Laws' Masks

The use of filters for the extraction of texture information became popular during the 80s, after their use was proposed by Laws (Laws, 1980). Since the publication of his work, many modifications and different filtering banks are introduced and tested but the general structure of the method remains the same, as summarised at the beginning of Section 5.1.1.

Filters are designed to highlight points of high *texture energy* in the image. In our experiments (Chapter 6) we use the 1x5 masks proposed by Laws (1980), that are designed to pick up the average gray *Level*, *Edges*, *Spots*, *Ripples*, and *Waves*. The initial letters of the masks indicate the ***Local averaging***, ***Edge*** detection, ***Spot*** detection, ***Ripples***, and *Waves* (Davies, 2005):

$$\mathbf{L5} = [\,1\ \ 4\ \ 6\ \ 4\ \ 1\,] = \mathrm{L3}*\mathrm{L3};$$

$$\mathbf{E5} = [-1\ -2\ \ 0\ \ 2\ \ 1\,] = \mathrm{L3}*\mathrm{S3};$$

$$\mathbf{S5} = [-1\ \ 0\ \ 2\ \ 0\ -1\,] = \mathrm{S3}*\mathrm{S3};$$

$$\mathbf{R5} = [\,1\ -4\ \ 6\ -4\ \ 1\,] = \mathrm{L3}*\mathrm{E3};$$

$$\mathbf{W5} = [-1\ \ 2\ \ 0\ -2\ \ 1\,] = -\mathrm{E3}*\mathrm{S3}.$$

They are derived from three simple vectors of length 3, L3 = (1, 2, 1), E3 = (–1, 0, 1), and S3 = (–1, 2, –1) using the convolution operator '*'. L3, E3, and S3 represent the one-dimensional operations of center-weighted local averaging, symmetric first differencing (edge detection), and second differencing (spot detection) They are multiplied with each other to produce 25 ifferent *masks* that form a *complete* set (Davies, 2005). The five-dimensional masks are convolved with the collection of images and a set of filtered images is obtained. Subsequently, each filtered image is converted to a *texture energy map*. This process is also called *smoothing* and the aim is to deduce the local magnitudes of the quantities of interest (edges, spots, etc.). An extensive overview of various smoothing techniques is presented in Randen and Husøy (İ999). In our work we adopt the *energy function* as proposed by Laws. A smoothing window of size 15x15 (Laws, 1980; Umbaugh, 2005) is applied to each filtered image F_k(r, c) for the k-th mask (k = 1, …, 25) and new *energy images* are obtained where each pixel is given by:

$$E_k(\mathrm{r,c}) = \sum_{j=c-7}^{c+7} \sum_{i=c-7}^{c+7} |F_k(i,j)|, (k = 1, \ldots, 25), \qquad (5.6)$$

where (r, c) denotes rows and columns indexes. After obtaining 25 energy maps for each image, we use the *power* metric, which is a sum of the squared absolute quantities for each value in the map (Umbaugh, 2005), to finally obtain 25 different values for each sample.

5.2 Cork: Motivation and Impact

5.2.1 Cork Harvesting

Cork is the bark of the cork oak (*Quercus suber Linnaeus*), a tree that grows in Mediterranean regions such as Portugal, Spain, Italy, France, and others. Portugal with 60% of the cork trees in the world is the largest producer of cork today (WWF Report, 2006). Cork oaks are broadleaved evergreens adapted to surviving the extreme summers and once-frequent fires. The cork product is one hundred percent natural, recyclable, and biodegradable (Portuguese Cork Association). Cork harvesting is an environmentally friendly process during which not a single tree is cut down. The first cork oak is not harvested until the tree is about 20 years old, and it is done carefully by hand (Fig. 5.2). The tree bark re-grows completely, with a smoother texture after each *stripping*. Each tree is harvested only once in 8-10 years and the cork oak itself can live up to 150-200 years, well after multiple *strippings*.

Cork is a product with unique natural characteristics according to the Portuguese Cork Association:

- very light;
- impermeable to liquids and gases;
- elastic and compressible;
- an excellent thermal and acoustic insulator;
- incombustible;
- highly abrasion resistant;
- resistance to rotting.

The bark is fire-resistant: the outer layers will char, but not burn, during a wildfire. The cork cellular structure is very similar to that of a honeycomb: each square centimetre is composed of millions of cells. These cells are filled with a gaseous mixture similar to air. Naturally occurring fatty substances (suberin and cerin) prevent the cultivation of mould and keep away bugs and mites. Cork is resistant to more than 38 species of insects, including the termite. It prevents the cork products from water damage and deterioration. A plank of cork contains nearly 60% gaseous elements, which explains its extraordinary lightness. These small cushions of air are what make cork so remarkably compressible. At the same time, suberin makes the walls of the cork cells impermeable and therefore airtight. The gas they contain cannot escape, which is the reason for the elasticity of the tissue and also its low thermal conductivity (Portuguese Cork Association).

(a) The cork oak. (b) Cork harvest.

Figure. 5.2. The cork oak (images are adopted from the Portuguese Cork Association).

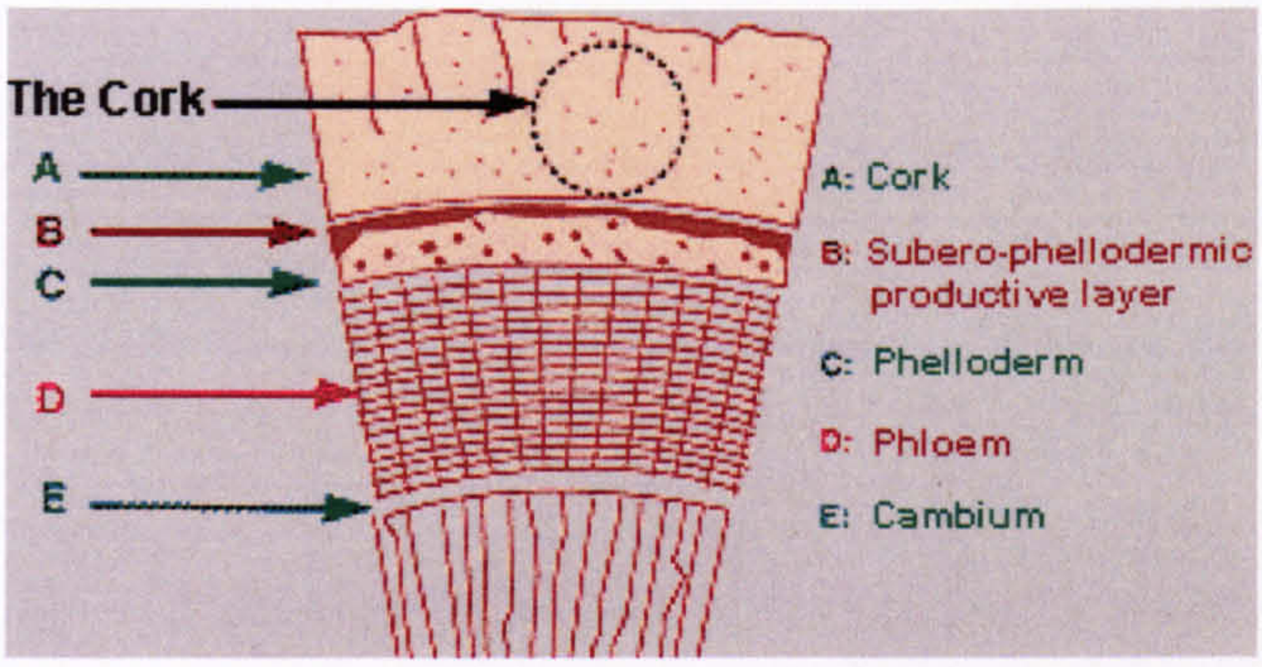


Figure 5.3. Cross-section of cork oak bark, showing the different layers (image is adopted from WWF Report (2006)).

The cork oak stem is illustrated in Fig. 5.3, where the different layers forming the bark are shown. Although the primary use of cork is in the wine stoppers production (70% of the total cork market), cork floor and wall covering give about 20% of the total cork business (WWF Report, 2006).

5.2.2 History of Cork Tiles

Detailed history of cork products use in time can be found in the Portuguese Cork Association website (www.realcork.org). In brief, it could be summarised as:

- In 3000 BC, cork was already being used in fishing tackle in China, Egypt, Babylon and Persia;
- Around 1680 a French Benedictine monk Dom Pierre Pérignon began to use cork to seal bottles of his famous Dom Pérignon champagne;
- 1729-1743 French champagne makers began to seal their bottles with cork stoppers and in 1750 the first cork factory was opened in Spain;
- Insulation corkboard was actually discovered in the 18th century and widely used in military vessels, railcars, etc.;
- During the early 1900s most of cork's unique properties were discovered and it became one of the most popular and widely used resilient floor materials;

- 1900-1945 characterised by extensive use of cork in flooring (www.expanko.com). While Portugal and Spain produced the raw material, Great Britain, United States and Switzerland produced most of the cork tiles;
- In the second half of the 20th century, due to interest in other products, cork tiles lost much of the market share;
- Recent increase of alternative wine stoppers arises serious attention and concerns, since this is reducing the economical value of cork lands and might lead to abandonment, degradation and loss of irreplaceable biodiversity (WWF Report, 2006);
- In the last decades cork begins to regain popularity, as it is used for a number of products (Fig. 5.4);

Figure 5.4. Cork products of the last decade (images are adopted from the Portuguese Cork Association).

5.2.3 Cork Floor and Wall Covering: Advantages and Manufacturing

In the past several years of technological advancement, cork has become one of the most effective and reliable natural materials for floor and wall covering (www.builddirect.com). Some of the advantages of the cork tiles are their durability, ability to reduce noise, thermal insulation, and reduction of allergens. Many of the cork floors installed during the *golden age* of cork flooring (Frank Lloyd Wright's *Fallingwater* (Fig. 5.5 (a)); *St. Mary of the Lake* Chapel in Mundelein (Fig. 5.5(c)); US Department of Commerce Building, etc.), are actually still in use, which is the best proof of their durability and ever-young appearance.

(a) Frank Lloyd Wright chose cork for many of his residential designs including *Fallingwater* in Western Pennsylvania (built 1935).

(b) Lafayette College, Easton, PA (built around 1930).

(c) St. Mary of the Lake Chapel, Mundelein (built around 1920).

Figure 5.5. From 1900 to 1945 cork was frequently used in government buildings, banks, universities and homes (www.expanko.com).

The properties of cork listed in Section 5.2.1 are indeed very useful for cork floor and wall covering. Cork flooring is silent, warm, comfortable, easy to clean and exceptionally resistant. For public buildings such as schools, hospitals, etc., it is a highly effective and economic choice because it also does not retain dirt and reduces footstep or impact noise. The same properties can also be extended to wall and ceiling coverings, since cork boards are versatile and easy to install (Portuguese Cork Association). In addition, current technological advances allow cork tiles to be produced in variety of textures and user friendly appearance (Fig. 6.11).

Virgin cork bark and cork waste from manufacturing of other cork products are recycled and grounded into small granules. Any rejected material during production is recycled back into future products. Dyes and colourings are not used often.

During manufacturing, all raw materials are consumed, either for the finished tiles or as energy source. The granules are baked in moulds at varying temperatures, allowing shade variations in the finished tile product from light to dark. After being baked, cut into slabs, sanded and finished with wax or polyurethane, the tiles are hand sorted to assure a high quality final product (www.expanko.com).

5.3 Automated Inspection of Cork Products

Selection is an operation designed to separate finished stoppers into different grades, which are determined by visual inspection of their surface (Fig. 5.6). Image analysis techniques are applied for the automated visual inspection of cork stoppers and planks (this is the raw tree material) in Pereira *et al.* (1996), Chang *et al.* (1997), Radeva *et al.* (2002), Costa and Pereira (2006), and according to the authors, the image-based inspection systems have high production rates. Such systems are based on a line-scan camera and a computer, embedded in an industrial sorting machine and capable of acquiring and real-time processing of the product surface image (Fig. 5.6(b)). During this stage, besides the definition of grades, defective stoppers are also eliminated.

Pereira *et al.* (1996) promote the automated inspection with Computer Vision techniques of the porosity of cork planks. The main feature considered is the *porosity coefficient* that is the percentage of pore area in relation to total area. The cork planks are classified in six quality grades. The authors performed numerical experiment with a number of cork boards and planks.

(a) Manual selection. (b) Machine Vision selection.

Figure 5.6. Selection and quality assessment of cork stoppers (Portuguese Cork Association).

Chang e*t al.* (1997) investigate a classification task for eight different types of cork stoppers with varying qualities. The authors use morphological filtering and *Contour Extraction and Following* in the feature generation stage, obtaining eight features in total. Subsequently, they employed fuzzy-neural networks and reported only 6.7% misclassification rate.

Radeva *et al.* (2002) consider visual inspection of five types of cork stoppers. They obtain 43 different features in the feature generation stage and further investigate them in the feature analysis stage. They perform PCA and LDA in combination with Nearest Neighbor classifier and *Independent Component Analysis* (ICA) in combination with Bayesian classifier. They report up to 58% success rate for different implementations of LDA; 46% for the PCA set, and up to 98% for the ICA set. However, ICA is not suitable for all types of data, because it imposes independence conditions on the features and also involves additional computational cost (Radeva *et al.*, 2002; Theodoridis and Koutroumbas, 2006).

Costa and Pereira (2006) investigate a system for intelligent inspection of seven types of cork stoppers. They report overall success rate of 46%-58%. The obtained low rate is mostly due to inappropriate feature generation and lack of other than LDA classifiers, which obviously is not effective enough for problems with similar complexity.

5.4 Summary

This chapter serves as an introduction to the case study presented in Chapter 6 of this thesis that is the automated inspection of cork tiles by texture analysis and Neural Network classification. Here, texture feature extraction techniques have been briefly reviewed and related literature introduced. Subsequently, the methods adopted in the experiments of Chapter 6, have been discussed in detail. In Section 5.2 has been given a motivation for interest in cork products and, in particular, cork tiles. Finally, in Section 5.3 have been considered results from relevant research in automated inspection of cork planks and stoppers.

6 Case Study: Automated Inspection of Cork Tiles

Here is described a Computer Vision system that is the case study of this research. It involves acquisition, processing, and classification of commercially available cork tiles. An experimental setup is proposed and results from the simulation are presented. The study is conducted in three separate stages. Firstly, only the initial testing of the system is provided, where a limited number of cork products were available for classification (Experiment I). Later on, four classes are considered and data processing is included (Experiment II). Finally, a larger data set with samples of seven different classes is considered, processed and learnt by the NN (Experiment III). The final system is thoroughly evaluated by numerous tests and discussion.

6.1 Experiment I

This experiment is just a preliminary step to the following ones from Section 6.2 and Section 6.3. Here, we consider three types of cork tiles (Fig. 6.1), that need to be learned by a NN, and later on the samples to be correctly classified. Only basic and simple feature extraction is provided, resulting in four-dimensional feature vectors. The results of this experiment were published in Georgieva *et al.* (2007).

(a) Type 1 (b) Type 2 (c) Type 3

Figure 6.1. Samples from each of the three cork tile types considered (the crude tile images are courtesy of Prof. B.Batchelor and Mr. S. Caton from Cardiff University, UK).

6.1.1 Texture Features Generation and Experimental Setup

- Co-occurrence Matrices

We use the Matlab Image Processing Toolbox in order to compute the co-occurrence matrices with the default eight gray levels. As usually accepted by other authors (Randen and Husøy, 1999), we use the nearest neighbour pairs at orientations 0°, 45°, 90°, and 135°. The Matlab Image Toolbox is utilised for the computation of the Energy, Homogeneity, Correlation, and Contrast characteristics in each direction. We consider their mean values in all four directions to be the parameters fed into the NN. Therefore, each image is characterised by a four dimensional *feature vector*. Typical examples of the feature vectors for the three classes are given in Table 6.1.

Table 6.1. Example of four-dimensional feature vector for each class.

Class	Contrast	Correlation	Homogeneity	Energy
1st	0.2381	0.4211	0.8829	0.4547
2nd	0.4795	0.5712	0.7971	0.1814
3rd	0.3034	0.4679	0.8539	0.3467

Fig. 6.2 and Fig. 6.3 show how the feature values are distributed for each of the classes and illustrate the mean values and the standard deviations of the four characteristics (16 samples for each class). It is seen from the figures that there is some major overlapping of all features for the first and the third class and they are not linearly separable. For the *Contrast, Homogeneity,* and *Energy* characteristics, the first class contains most of the values of the third class. There is greater difference for the *Correlation* feature, but still the classes are not separable. This makes the task of correctly classifying them very challenging. The second class overlaps with the first one only for the *Correlation* feature (Fig. 6.3), while it is completely separated from the third class, which makes the correct classification of instances of class two easier.

For the Neural Networks learning problem, we use non-overlapping images of three different tile types (samples are shown in Fig. 6.1), adopting a size of 150x150 pixels for each sample. The aim is to classify correctly each testing image. From each of the three classes we use 8 samples for training and 8 different ones for testing purposes.

In total, the NN is trained with 24 samples and tested with another 24. We adopt the architecture described in the benchmark testing in Section 3.1.4. Here, the input layer consists of four neurons. For the hidden layer, three different cases are considered with number of neurons being: 7, 8 and 10 (this implies three architectures: 4-7-2 (n = 51), 4-8-2 (n = 58), 4-10-2 (n = 78). In all cases, the third layer has two neurons and the outputs for the three classes are coded as (0, 0) – for the first, (1, 0) – for the second, and (0, 1) – for the third class.

The investigated NNs are not only with different topology but also two types of transfer functions are employed for the output layer – *Heaviside* and *Sigmoid*. In the second case, output values greater than 0.5 are assumed to be 1, and otherwise – 0.

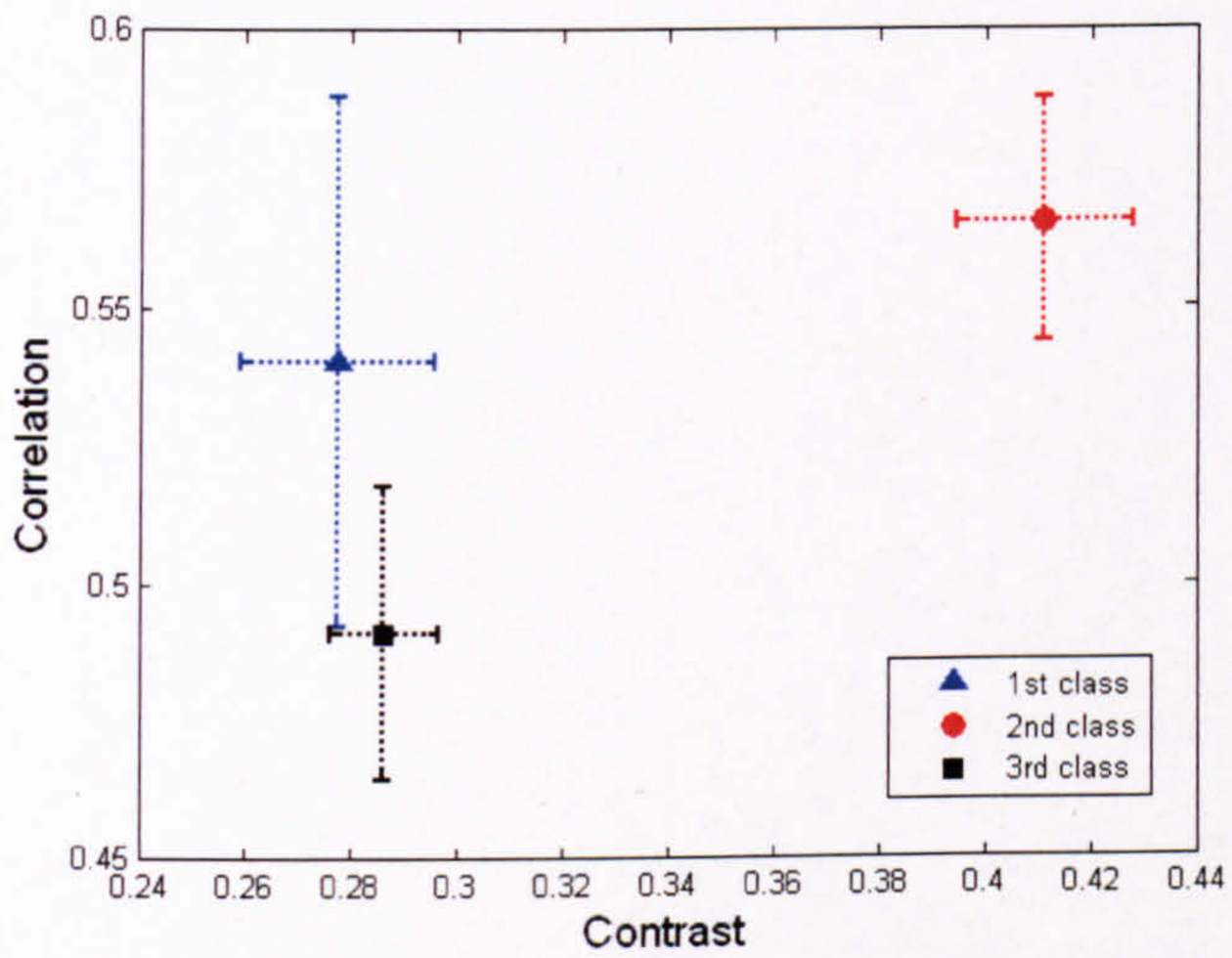


Figure 6.2. Distribution of the Contrast and Correlation features for the three classes – mean and standard deviation values.

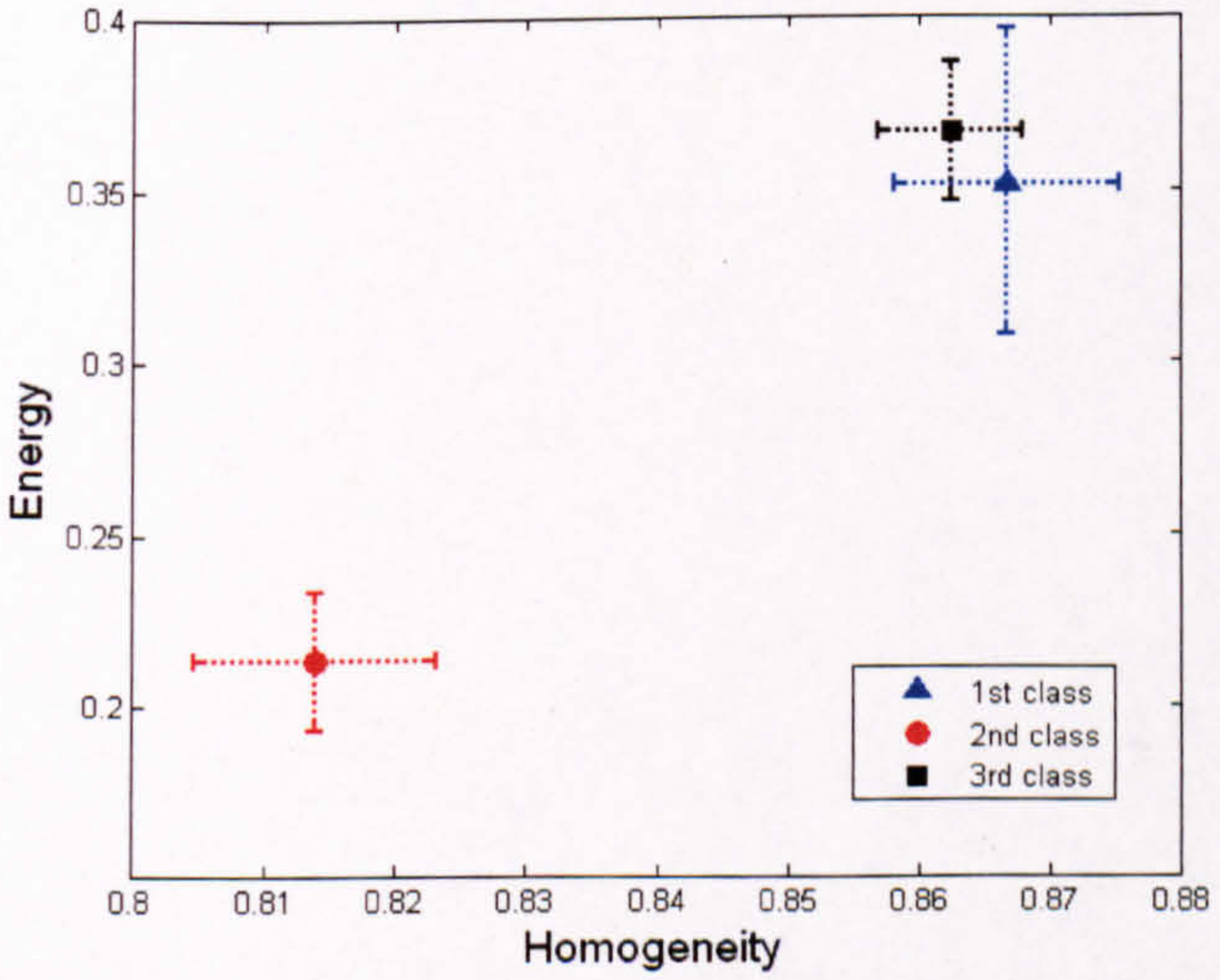


Figure 6.3. Distribution of the Homogeneity and Energy features for the three classes – mean and standard deviation values.

6.1.2 Results and Discussion

Results from the training and testing are presented in Table 6.2, where the three different NN architectures are investigated. For each configuration, a NN is trained and tested 50 independent times and the average results are shown in Table 6.2. The last column illustrates the classification rate and evaluates the generalisation ability of the corresponding Neural Network. The third column shows the mean training error from equation (2.7) and the corresponding standard deviation (given in parentheses). It demonstrates and assesses the minimisation abilities of the *GLPτS* Global Optimisation technique.

For all configurations, the test success rate is more than 70%. These are promising results, since this experiment is just in an initial stage of building methodology for automated visual inspection of cork tiles. In all cases the training error function is minimised to values less than 0.072 and the small standard deviation values show that these are stable results for all 50 runs. Best minimisation results are achieved in the case of 4-7-2 architecture with a *Heaviside* transfer function in the output layer. This is also the case with the best success rate – 74.9%, which gives us an optimal NN configuration with acceptable generalisation abilities for the problem at hand.

Table 6.2. Cork tiles classification with two different transfer functions in the output layer. Training and testing results.

Criterion Architecture (*n*)	Output transfer function	Mean Training Error (std)	Testing Success Rate
4-7-2 (51)	*Heaviside*	0.059 (0.021)	74.9%
	Sigmoid	0.072 (0.0015)	72.17%
4-8-2 (58)	*Heaviside*	0.0633 (0.0117)	72.9%
	Sigmoid	0.069 (0.0015)	71.33%
4-10-2 (72)	*Heaviside*	0.0746 (0.015)	72.25%
	Sigmoid	0.067 (0.0014)	71.5%

Success rate over 70% is a very good achievement, considering that only few texture values are used and the data set is very limited. However, we might expect that the results will improve significantly when the following steps are implemented:

Examining higher number of texture features and employing different techniques for feature extraction (e.g., filtering);

Investigating the effect of statistical processing of the input data;

Increasing the number of training and testing samples, as well as including more types of cork tiles;

Investigating different cross-validation approaches when larger data sets are available.

All of these objectives are addressed later on in the following experiments (Section 6.2 and Section 6.3).

6.2 Experiment II

For the second experiment, we built a Computer Vision system (shown in Fig. 6.4) capable of acquiring and processing images with several feature generation and analysis techniques. Results of this experiment are published in Georgieva and Jordanov (2007b). In this experiment, the classifier design steps are conducted in accordance with the discussion in Section 2.2.1 and the algorithm given in Fig. 2.1.

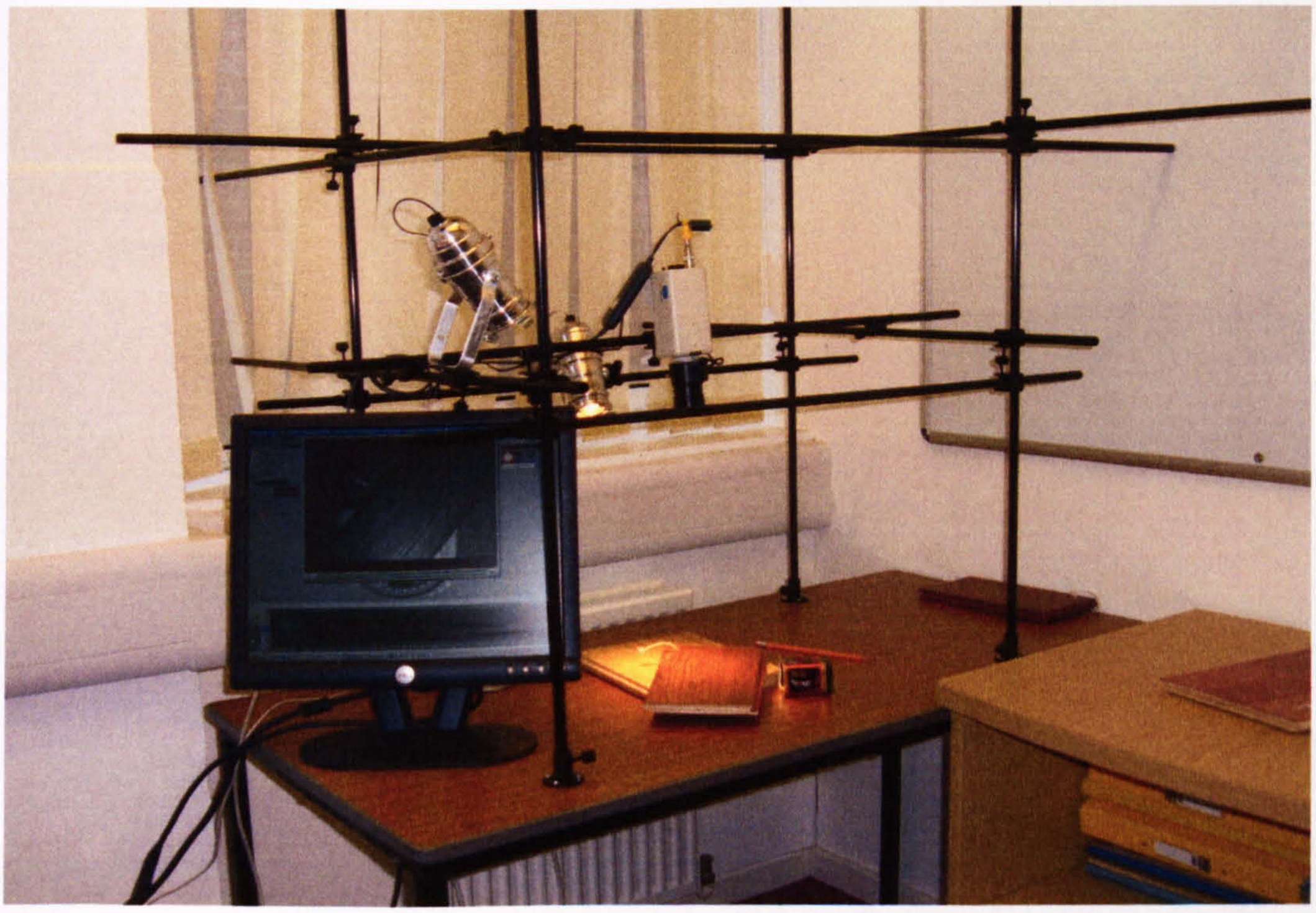


Figure 6.4. The Computer Vision system built for Experiments II and III. It consists of scaffolding, CCD camera, frame grabber, lighting, and a PC.

6.2.1 Equipment and Image Acquisition

The equipment we use consists of the following major elements:

- CCD Camera and frame grabber

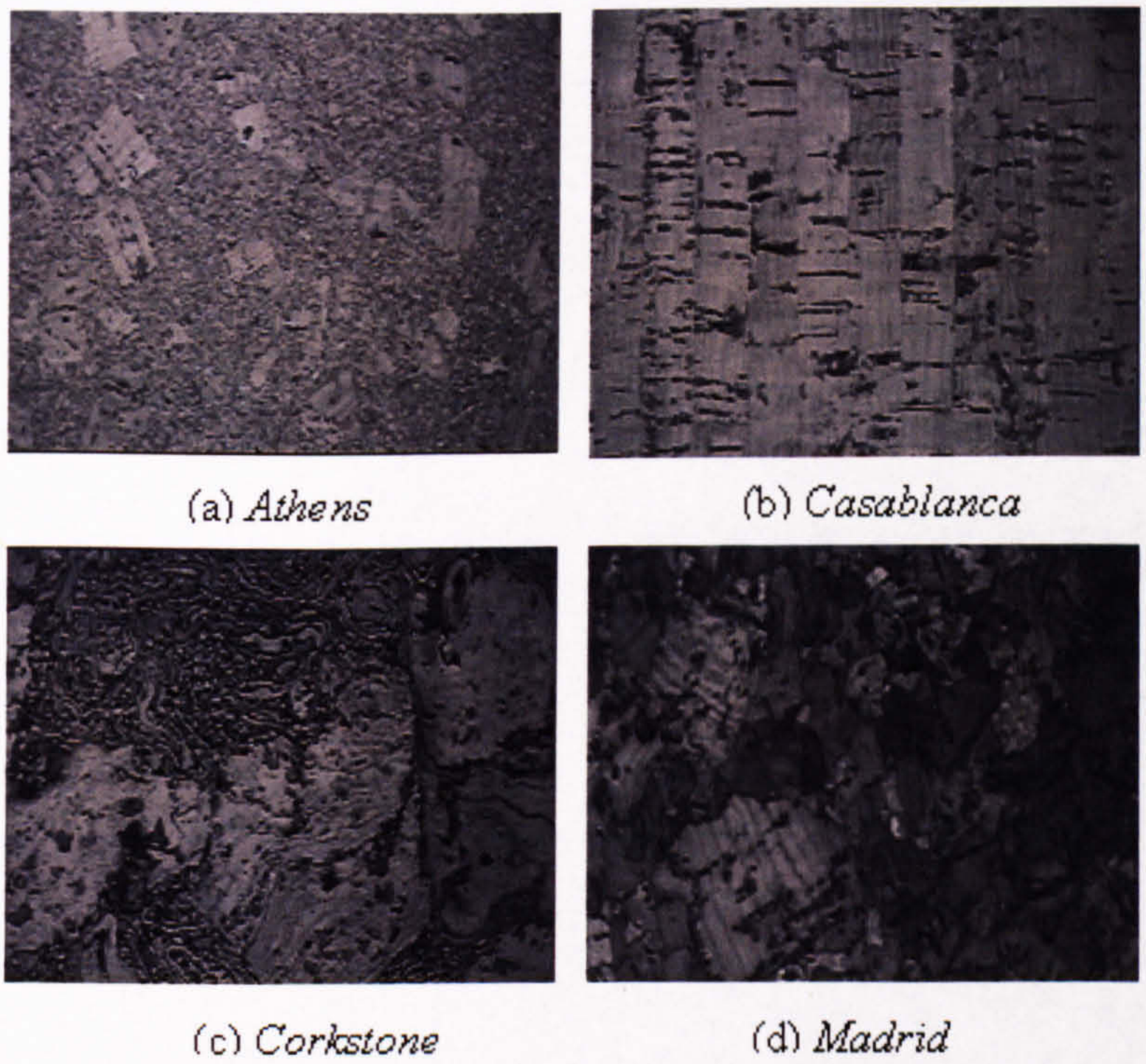
For the image acquisition we use a CCD camera equipped with a *Genie Varifocal Lens* (focal length within 5-50mm) capable of capturing images from small distance with fine detail. The camera is connected with the PC through a *Hi Speed USB DVD Creator* provided with *Ulead VideoStuio* 6.0 software.

- Lighting

Except the usual lighting in the room, two additional 12V dichroic lamps are used especially for lighting the objects that are being pictured.

- Scaffolding

The camera and the lights are attached in a flexible, specially built *Climpex* scaffolding. It is made of clamps and tubes that can be easily rearranged if necessary (Fig. 6.4), providing flexible and yet stable scaffolding.



(a) *Athens* (b) *Casablanca*

(c) *Corkstone* (d) *Madrid*

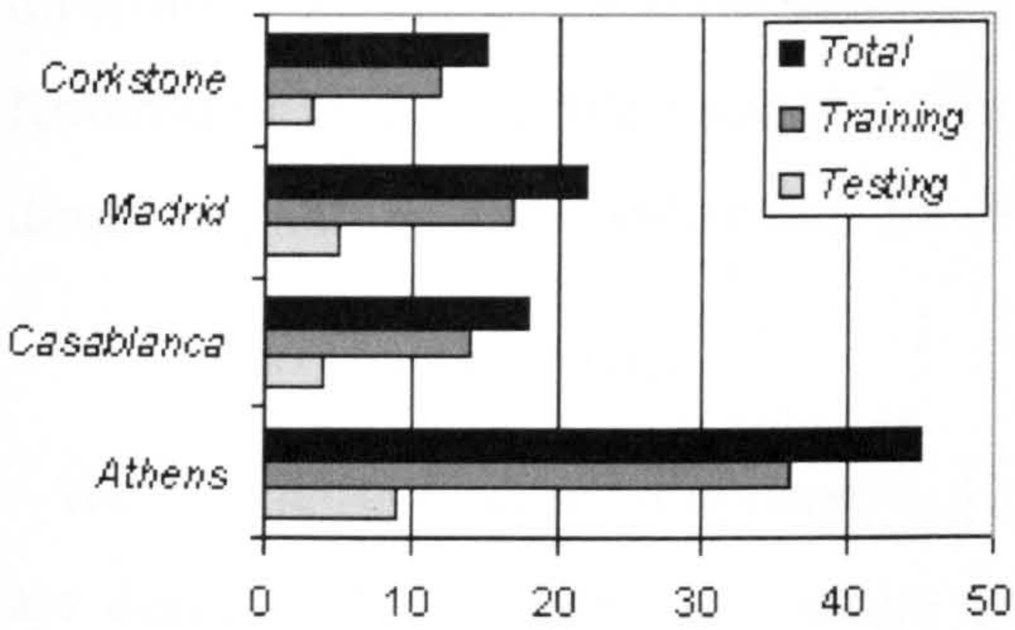
Figure 6.5. Four types of cork-tiles, commercially available from *Jelinek Cork Group* (www.CorkStore.com).

In this experiment four types of cork tiles are investigated (Fig. 6.5). The images are considered in a grayscale format, since at this stage, the texture is of prime interest (colour images might be investigated in future work).

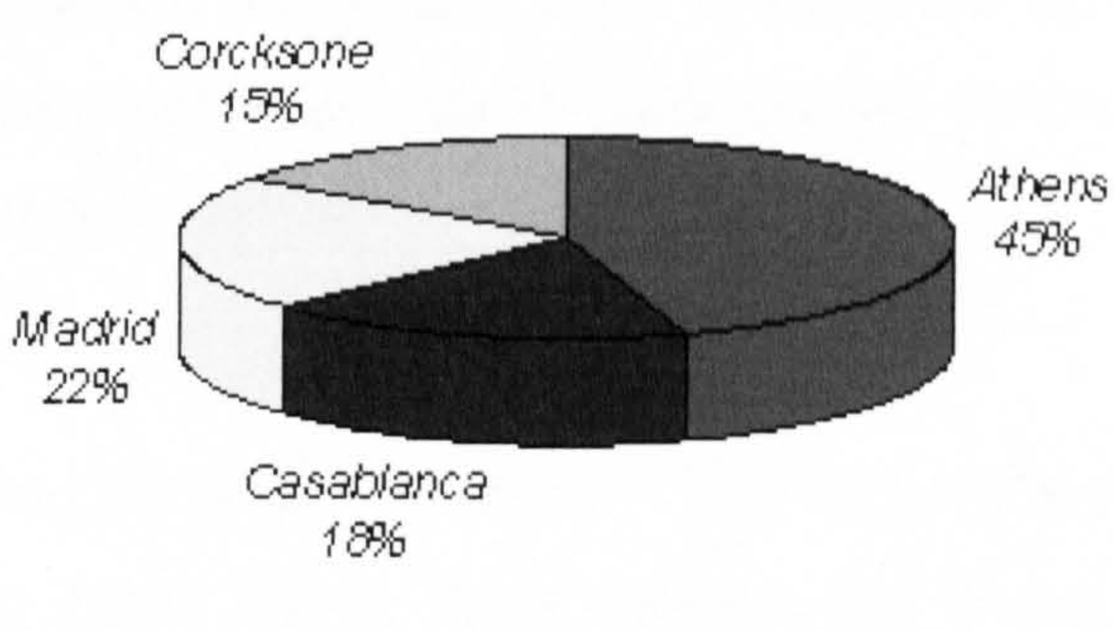
6.2.2 Texture Features Generation and Analysis

The Dataset

A total of 176 different images are collected with the Visual System, from which approximately 45% are of type *Athens*, 18% of *Casablanca*, 22% of *Madrid*, and 15% of type *Corkstone*, as shown in Fig. 6.6. The images are taken under equal lighting and camera conditions. From the dataset, 80% is used for training and 20% for system evaluation (Fig. 6.6).



(a) Number of samples from each cork type (80% training, 20% testing).



(b) The percentage distribution of each cork type.

Figure 6.6. Dataset sample distribution for Experiment III.

Texture Analysis

The images are to be classified into four different classes – *Athens*, *Casablanca*, *Corkstone*, and *Madrid* (sample images are shown in Fig. 6.5). The first step of the feature generation stage is to reduce the effects of illumination (discussed in Section 4.1). Initially, texture features are *generated* with the use of conventional techniques for *feature extraction* – co-occurrence matrices (Haralick *et al.*, 1973; Umbaugh, 2005; Theodoridis and Koutroumbas, 2006) and Laws' masks (Laws, 1980; Umbaugh, 2005; Theodoridis and Koutroumbas, 2006). However, instead of choosing one of the techniques, as usually done by other authors (Randen and Husøy, 1999; Radeva *et al.*, 2002), we combine them and use different statistically based approaches in order to select and further improve the features at hand. This is done with the aim of reducing the dimensionality of the problem and disposing of

information redundancy. The obtained image characteristics are fed into a NN classifier for which several NN architectures are designed, trained, tested and evaluated.

- Co-occurrence Matrices

We considered two spatial relationships – the direct neighbours and pixels with difference of five. We use the Matlab Image Processing Toolbox in order to compute the co-occurrence matrices with fifteen gray levels. As usually accepted by other authors (Randen and Husøy, 1999), we use orientations of horizontal (0°), right diagonal (45°), vertical (90°), and left diagonal (135°). The Matlab Image Toolbox is used for the computation of *Energy*, *Homogeneity*, *Correlation*, and *Contrast* characteristics in each direction. The values of the features are averaged over the four directions, giving us rotation invariant features (Umbaugh, 2005; Theodoridis and Koutroumbas, 2006). We compute eight co-occurrence features – four features for the direct neighbours and another four for the pixels with difference of five.

- Laws' Masks

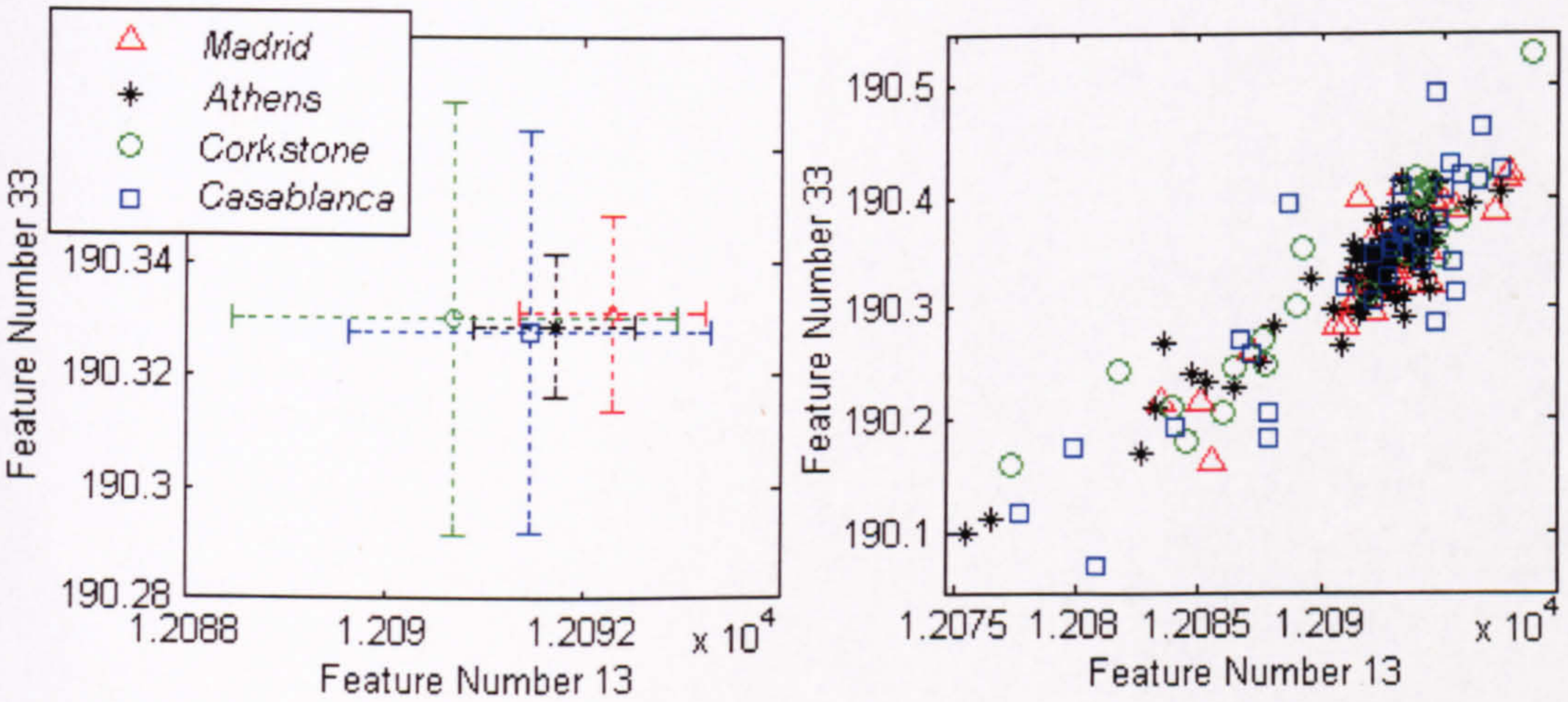
We use the 1x5 masks proposed by Laws (1980) and discussed in Section 5.1. They are designed to pick up the average gray *Level*, *Edges*, *Spots*, *Ripples*, and *Waves*. They are multiplied with each other and in this way 25 different *masks* are produced. The masks are convolved with the set of images and a set of filtered images is obtained. Subsequently, each filtered image is converted to a *texture energy map*. For the energy map we adopt 15x15 window, as proposed by Laws (1980). After obtaining 25 energy maps for each image, we use the *power* metric, which is the sum of the squared absolute values for each pixel in the map (Umbaugh, 2005), to finally obtain 25 different values for each sample.

Feature analysis and selection

In order to reduce the dimensionality of the problem and dispose information redundancy, we would like to be able somehow to choose the *best* features (or their transforms) from the original 33 available. We use statistical approaches to select two different sets of features and the NN performance for both of them is discussed in Section 6.2.3.

- Analysis of Variance

Initially, we perform *Analysis of Variance* (ANOVA) of the data. By the use of this classical statistical method we are able to distinguish which of those 33 features have significantly different mean values for all cork classes (and thus *separate* the classes). We perform ANOVA tests to find out that only 15 feature vectors have significantly different means for *all* four cork classes. Especially the 13th and 33rd features from our dataset have on average the same mean values for all the classes and appear to be insignificant for our classification task. Fig. 6.7(a) illustrates the idea, showing the class means with 95% confidence intervals for the 13th and 33rd features and in Fig. 6.7(b) the actual samples are shown by a scatter plot. At this stage we remove these two features from our dataset for all further analysis. Further on, we processed the data set in two different ways and obtain two different feature sets.



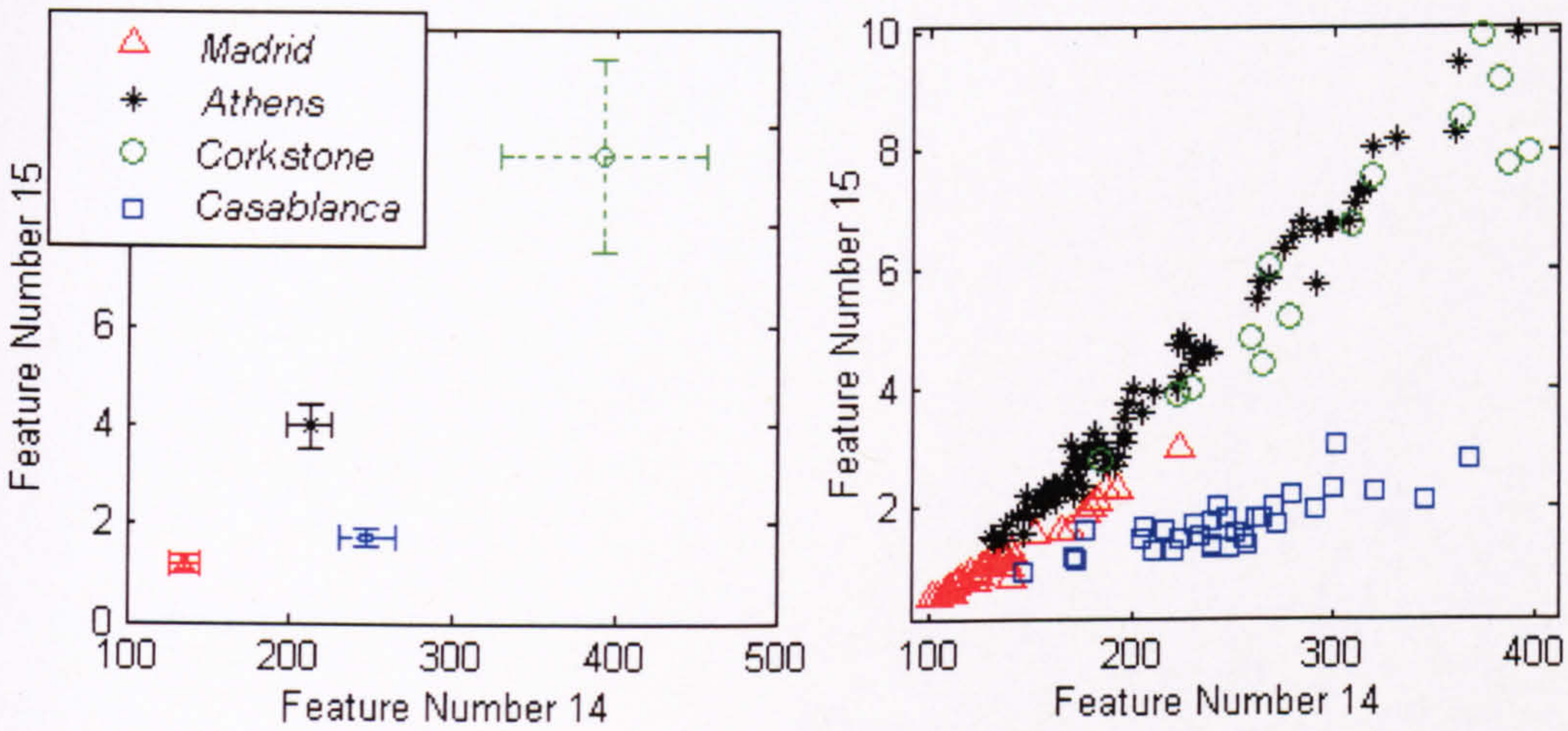
(a) Cork classes means with 95% confidence intervals.

(b) The cork samples for the four investigated classes.

Figure 6.7. Features 13 and 33 from the original data.

Feature set I: We keep only 15 features that have statistically significant means for *all* the four classes. In general, the more variance one feature vector has, the more information it provides for the cork classes (Egmont-Petersen *et al.*, 2002; Umbaugh, 2005). Therefore, we compute the variance for all 15 features and rate them in decreasing order accordingly to their variance. Thus, we obtain a set of 15 ordered in

decreasing importance features. It is interesting to note that only one of the co-occurrence features is removed with the ANOVA test, but after the ordering, the first seven features appear to be obtained by Laws' masks. Fig. 6.8(a) shows the means with 95% confidence intervals for the first two selected features (numbers 14 and 15 from the original data set), and Fig. 6.8(b) shows the actual samples. Obviously, all four classes have significantly different means for these features and are well *separated*.



(a) Cork classes means with 95% confidence intervals.

(b) The cork samples for the four investigated classes.

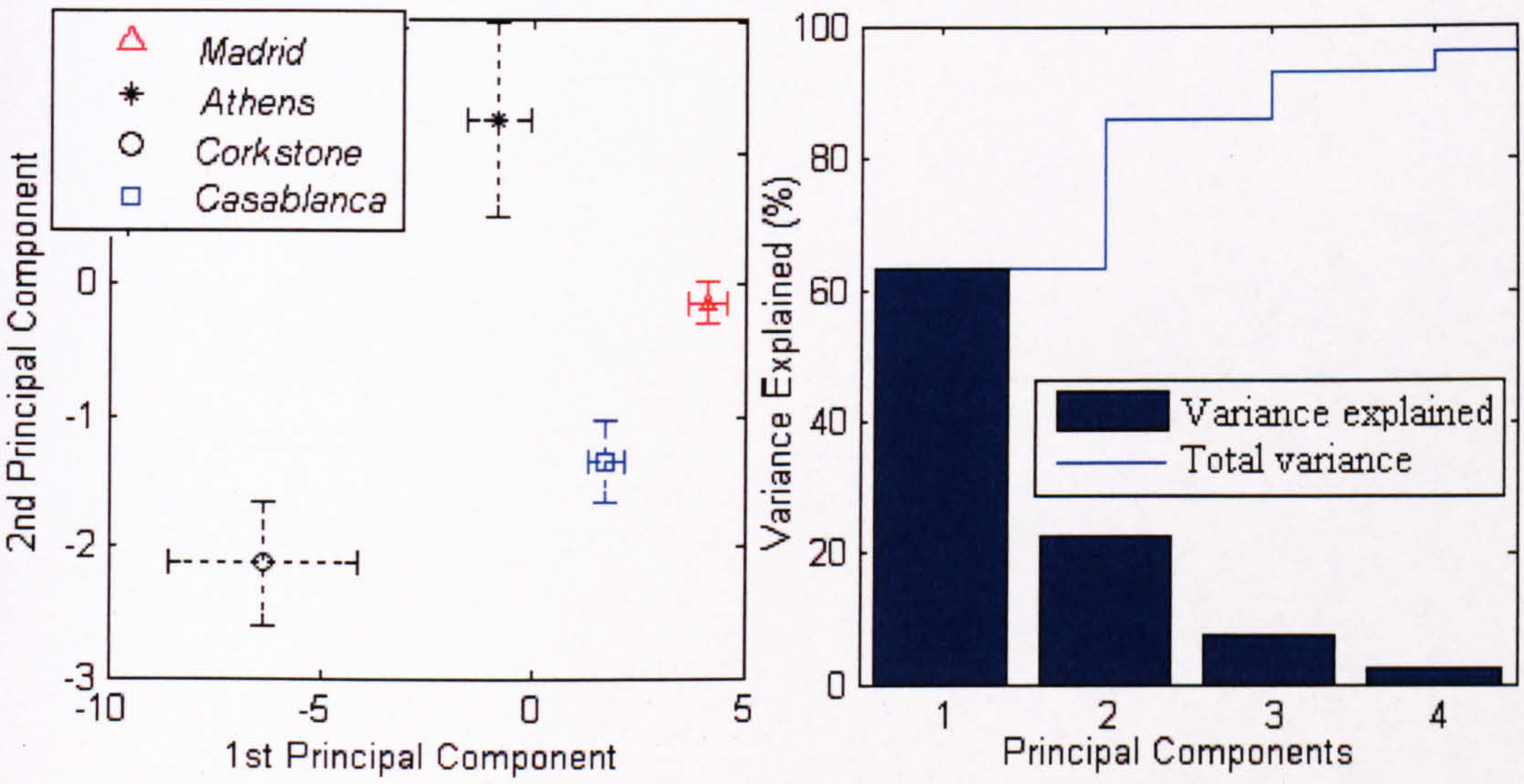
Figure 6.8. Features 14 and 15 from the original data.

- Principal Component Analysis

Feature set II: After removing features 13 and 33 from the data set, in order to obtain the second feature set we perform *Principal Component Analysis* (PCA) on the 31 features and thus obtain 31 *Principal Components* (PCs). The main goal of PCA is to *de-correlate* the data and transform it linearly to a new data set, where the information is presented in a *better* way – the feature vectors are orthogonal and are ordered with decreasing variance.

The analysis in our case shows that the first 8 PCs contain in total 99.9% of the whole variation (information) of the original data. One can see from Fig. 6.9(b) that the first 3 PCs already contain 97% of the variation in the data. However, PCA transforms the data considering all examples (in our case cork image features)

independently, without taking into account what cork class they belong to. In other words, the transform itself aims to de-correlate the data and obtain as much variance as possible in the first components, but does not try necessarily to separate the cork classes. Therefore, we perform additional ANOVA on the PCs to see how they separate the different cork classes and choose which PCs to include in the feature set. For example, the 4th PC (or 4th new feature) does not separate any of the cork classes and we do not use it in the feature set. On the other hand, the 10th PC, although with a lower variance, appears to separate the classes well and we include it in the set. The ANOVA shows also that the first 2 principal components, or new features, separate the classes well, which can be seen from Fig. 6.9(a).



(a) Cork classes means with 95% confidence intervals with respect to PC1 and PC2.

(b) 97% variation from the original data is represented by the first three PCs.

Figure 6.9. PCA results.

6.2.3 Results and Discussion

We conduct multiple NN training and testing experiments in order to evaluate the NN generalisation abilities. Depending on the NN topology and coding of the four classes – two different types of NN are considered:

- *Heaviside*

Two output neurons and a *Heaviside* transfer functions in the output layer. In this case, each class is coded as a pair of binary outputs – (0, 0), (1, 0), (0, 1) and (1, 1)

correspondingly. The mean results over 20 runs for this NN topology are reported in Table 6.3, where the first column shows the dimensionality of the optimisation problem when K-6-2 NN topology is adopted (K being the number of features). The second and fourth columns show the success rate from the test set that represents 20% of the original data (Fig. 6.6).

Table 6.3. NN training and testing results for the case a *Heaviside* function in the output layer.

Criterion	*Feature set I*		*Feature set II*	
K (n)	Success	MSE (σ)	Success	MSE (σ)
3 (38)	77%	0.164 (0.02)	83%	0.126 (0.016)
5 (50)	71%	0.17 (0.012)	89%	0.026 (0.009)
9 (74)	72%	0.17 (0.027)	91%	0.0061 (0.009)
10 (80)	70%	0.16 (0.013)	89%	0.0229 (0.025)

- *Tanh*

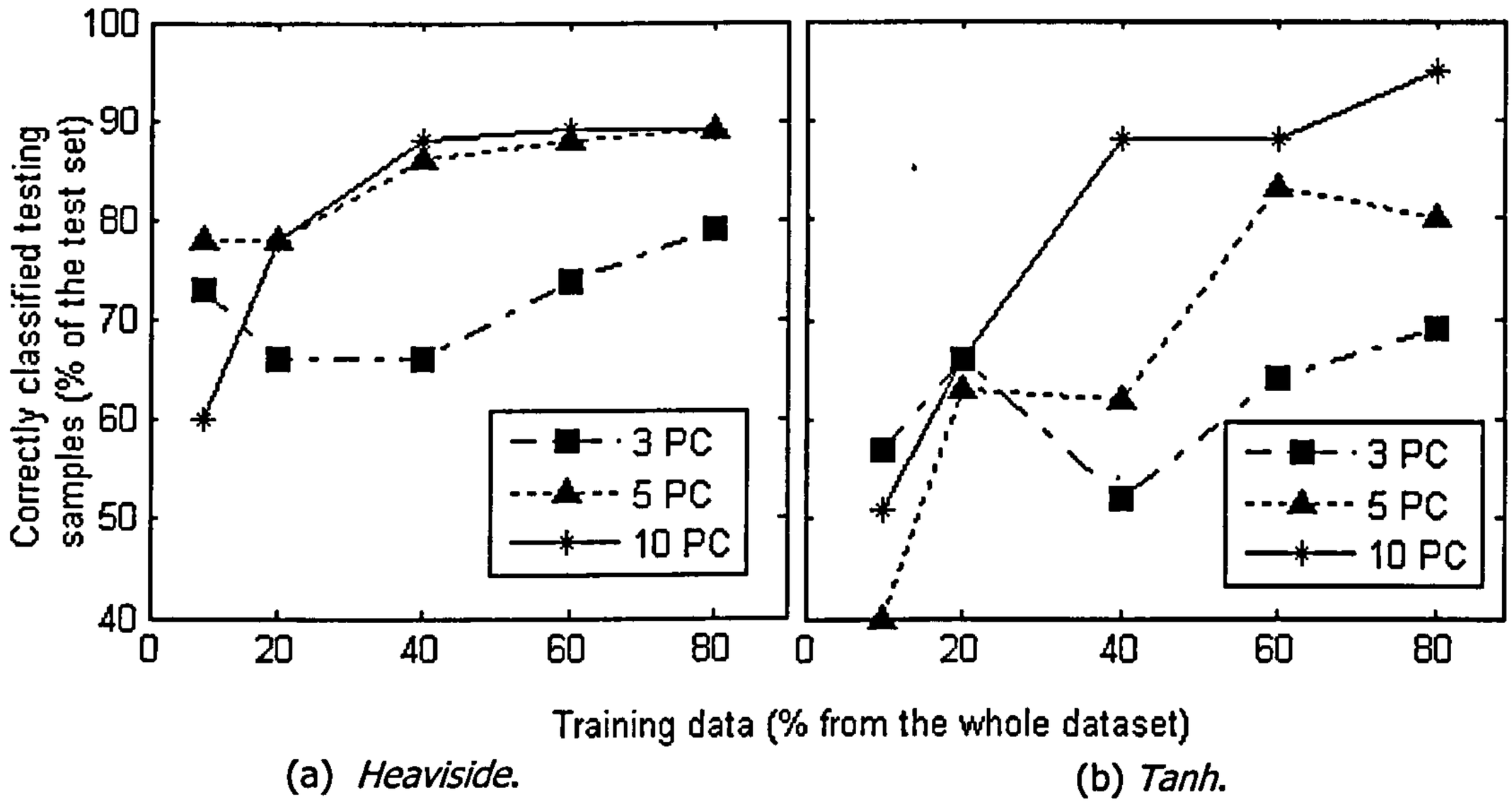
Here, a *Tanh* transfer function is employed in the output layer. In this case each class is coded with an output value of –0.25, –0.75, 0.25, and 0.75 correspondingly. The mean results over 20 runs for this topology (K-6-1) are shown in Table 6.4.

Table 6.4. NN training and testing results for the case a *Tanh* function in the output layer.

Criterion	*Feature set I*		*Feature set II*	
K (n)	Success	MSE (σ)	Success	MSE (σ)
3 (31)	69%	0.03 (0.0039)	69%	0.025 (0.0068)
5 (43)	74%	0.024 (0.005)	80%	0.013 (0.003)
9 (67)	80%	0.018 (0.005)	92%	0.0061 (0.009)
10 (73)	84%	0.017 (0.003)	95%	0.0013 (7e-4)

Table 6.3 and Table 6.4 rows show the change of NN performance with respect to the number of features utilised to describe the data. The characteristics are used accordingly to their importance (variance), which reflects the ordering discussed in Section 6.2.2 and the more of them we use, the better the overall performance is as a whole. Fig. 6.10 shows the obtained testing success rate for the two NN topologies

with different number of features (*Feature set II*), when the number of training samples is increased. The idea is to assess whether the used features do provide complete and reliable information for the different cork classes. We use 20% of the whole data as testing set and the remaining 80% as training set (increasing the percentage of training samples employed). If the success rate increases proportionally to the increase of the training set size, then the features are considered suitable (Umbaugh, 2005).



(a) *Heaviside*. (b) *Tanh*.

Figure 6.10. Testing success rate (%) for different number of features and increasing percentage of training set samples.

It can be seen from Fig. 6.10 that more reliable results are obtained when 10 features from *Feature set II* are used, in comparison with the case when only the first 3 are employed. Although the first 3 features contain 97% of the total variance, it seems their use alone cannot give reliable NN training.

The first important observation that can be made from Table 6.3 and Table 6.4 is, that in all cases the *Feature set II* produces better training and testing results. The maximal testing success rate for the first feature set is 84% when the *Tanh* transfer function with 10 features is used. This ia the case when the maximal success of 95% is achieved for the second feature set as well. These results validate the PCA as a very reliable dimension reduction technique suitable for pre-processing the training data. It is interesting to note that in the case of NN with *Heaviside* transfer function with 9 and 10 features, in some of the runs MSE is equal to zero. This means that with these

features an actual *learning* of the training set is possible, which is not a case for many real-world problems. We believe this is due to the careful preprocessing and features selection.

The overall best testing result achieved is 95%, which is can be considered as a very good result for a real world application problem. A 79% overall success rate for the classification of three types of cork stoppers was reported by Costa and Pereira (2006), whereas for the classification of five types of cork stoppers, classification results of 40-98% were reported in Radeva *et al.*, 2002. The authors of the latter publication also used cross-validation that might have improved further their results. Our future work will also include validation, more cork type classes, and larger data sets.

6.3 Experiment III

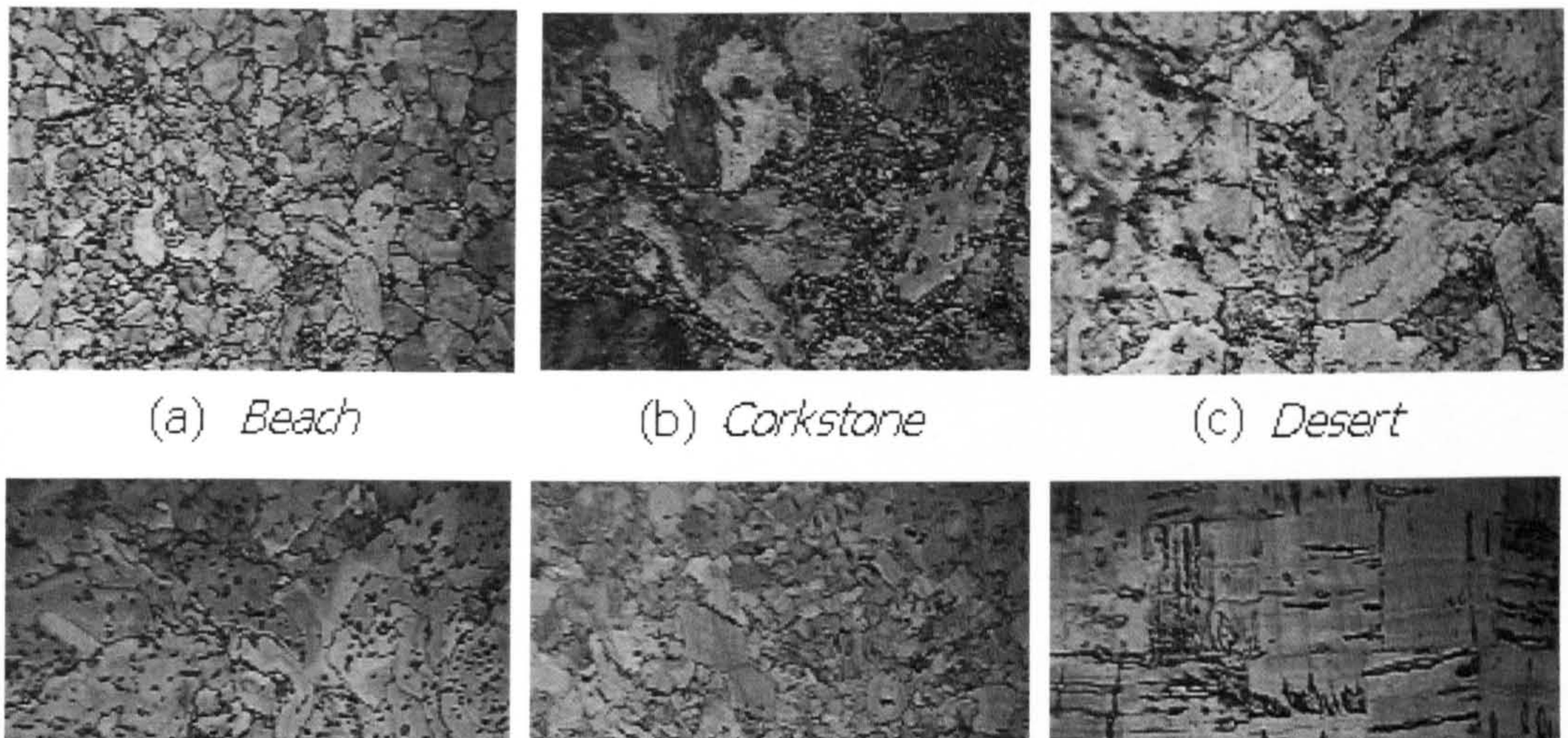
The results of this experiment were submitted for publication in Georgieva and Jordanov (2007c).

6.3.1 Image Acquisition

The aim of this experiment is to design, develop, and further investigate the intelligent system for visual inspection proposed in Section 6.2, which is able to automatically classify different types of cork tiles for floor and wall covering. Currently, the cork tiles are sorted "by hand" (e.g., see www.expanko.com), and the use of such system could automate this process and increase its efficiency. In this case study we experiment with seven types of cork wall tiles with different texture. The tiles we consider are available on the market by *Jelinek Cork Group* (www.CorkStore.com) and samples of each type are shown in Fig. 6.11.

For all cork types we use grayscale images of size 230x340 pixels and, in total, we collected 770 different images for all classes. Fig. 6.12 shows the percentage distribution of each type of cork tiles. We use 25% of all images for testing (not shown to the NN during training) and assessing the generalisation abilities of the networks. In general, when the classifier is well designed, 570 training samples would imply testing rate higher than 90%, if the NN dimensionality is about $n = 57$,

according to the *Vapnik-Chervonenkis* bound discussed in Section 2.1.2. The dimensionalities of the NNs considered here vary between 57 and 80.

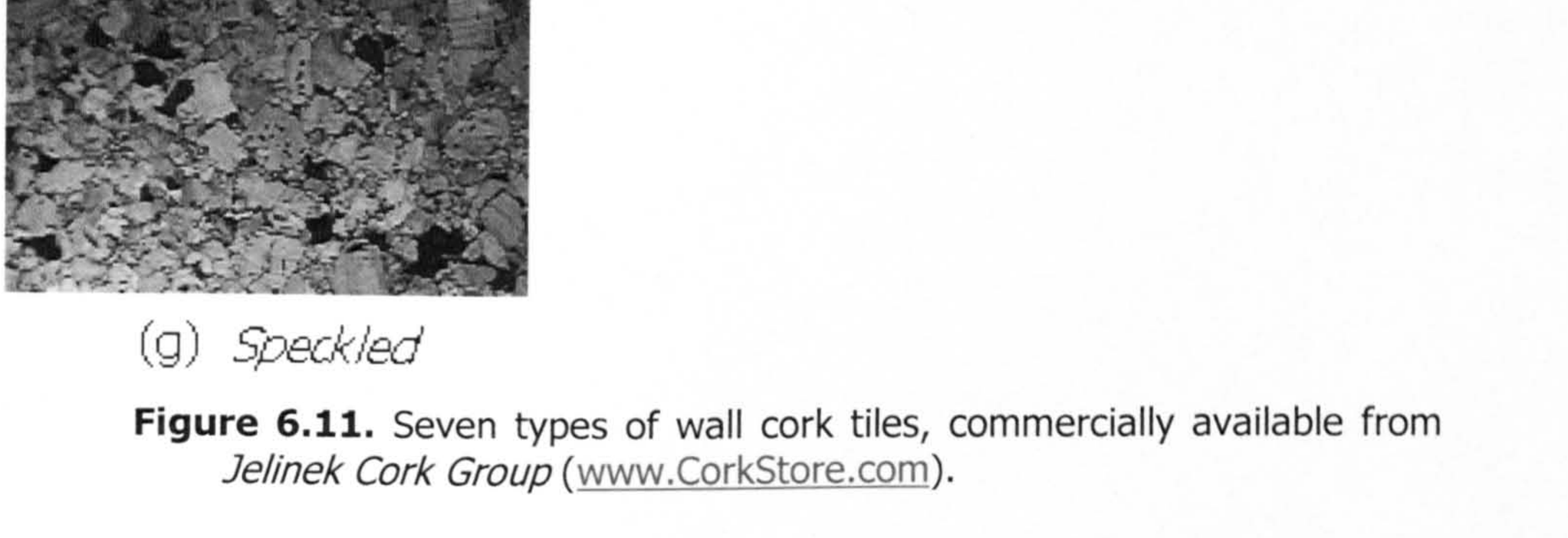


(a) *Beach* (b) *Corkstone* (c) *Desert*

(d) *Lisbon* (e) *Pebble* (f) *Precision*

(g) *Speckled*

Figure 6.11. Seven types of wall cork tiles, commercially available from *Jelinek Cork Group* (www.CorkStore.com).



(a) Dataset sample distribution (50% training, 25% testing, 25% validation). (b) The percentage distribution of each cork type.

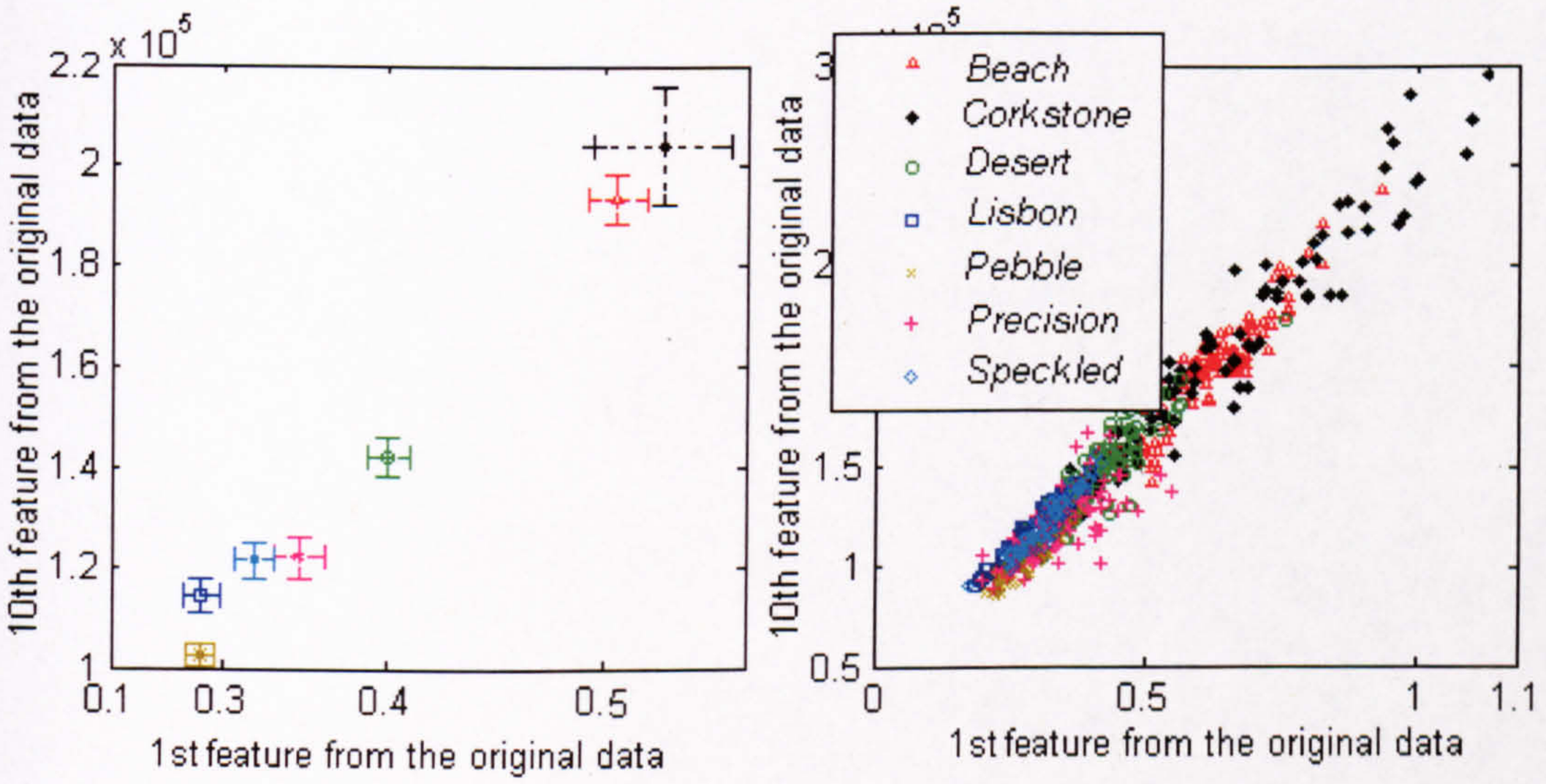
Figure 6.12. Dataset sample distribution for Experiment III.

6.3.2 Texture Features Generation and Analysis

The first step of the feature generation stage is to reduce the effects of illumination (as discussed in Section 5.1). Subsequently, we use two classical approaches to generate image texture characteristics: the Haralik's co-occurrence method and the Laws' filter masks, discussed in Chapter 5, and also utilised in Experiment II. We employ both methods and use the obtained features to generate one dataset, without taking into account the feature generation technique. We employ 8 co-occurrence characteristics and 25 Laws' masks to obtain 33 features for each image (in the same way as in Section 6.2). These features are further processed statistically in order to extract the most valuable information and to present it in a compact form, suitable for NN training.

Before processing the data, we took out 120 samples to be used later as a testing subset, therefore this data was not involved in the feature analysis stage.

Fig. 6.13 shows the distribution of the seven cork classes represented by two randomly selected features from the 33 available. The classes means with 95% confidence interval are shown in Fig 6.13(a) and one can see from Fig 6.13(b) that there is great overlapping between the classes (as well as strong linear correlation).



(a) Cork classes means with 95% confidence intervals

(b) Scatter plot of the cork samples from the seven classes

Figure 6.13. Two randomly selected features from the original data.

Initially, we performed correlation analysis of the features and found out that they are highly correlated. One *mulativariate analysis* technique that could solve this problem (as discussed in Section 2.1.3) is the *Principal Component Analysis* (PCA). In our preliminary investigation of the intelligent visual system in Experiment II (Section 6.2), we considered Analysis of Variance (ANOVA) for the feature selection. We also used PCA and our research showed that it outperforms significantly the simple ANOVA (at least for the investigated cork-tiles classification task). Following these results, in Experiment III presented here, we decided not to use ANOVA, and instead, to apply Linear Discriminant Analysis (LDA).

- Principal Component Analysis

Davies (2005) and Dillon and Goldstein (1984) investigated the case when a set of features has no special comparability and claimed that placing them in the same features space, and assuming that the scales on the various axes have the same weight factors, should be invalid. To solve this problem, Davies (2005) suggested rescaling of the values. Similarly, in our case, the first step before the PCA is to standardise the data, so that each column has a zero mean and a unit variation.

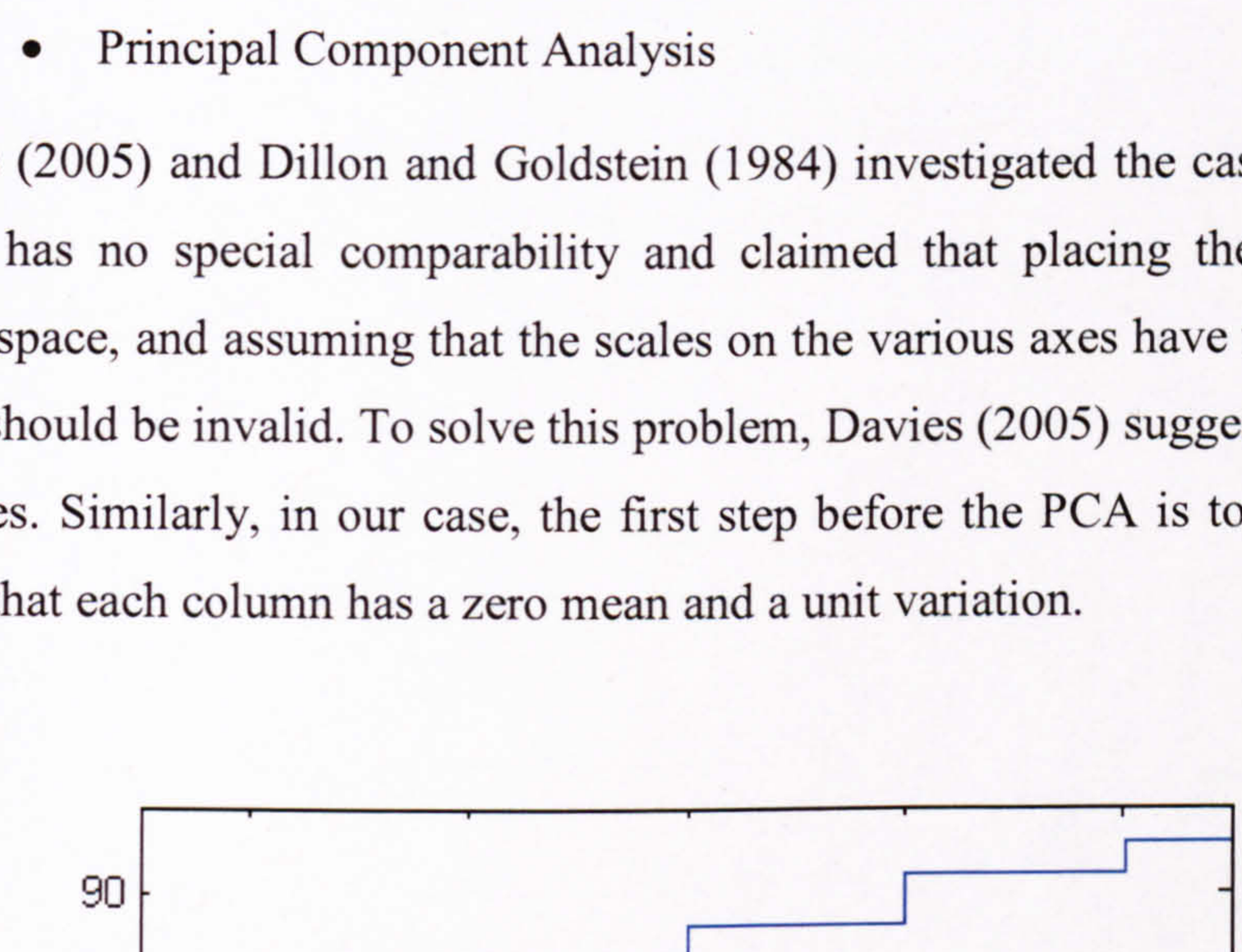
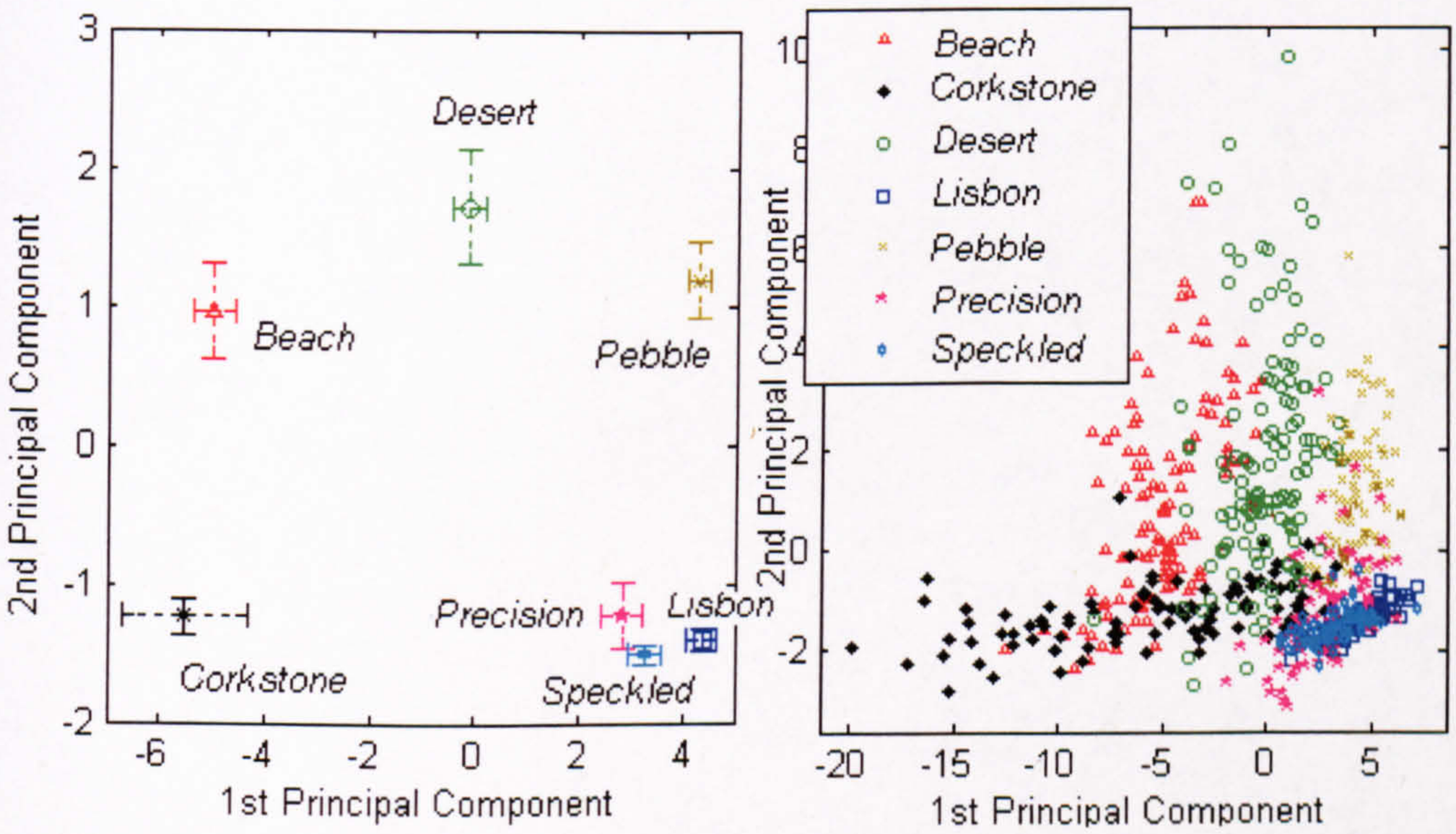


Figure 6.14. Percentage of the information from the original data contained in the first five principal components.

The application of PCA to the original data produces a new data set, in which the first five features (PCs) contain about 96% of the total variation (information) in the original data (Fig. 6.14). One can see from Fig. 6.14 that the first PC dominates significantly the others in terms of the variance explained by it. Later, when training our NN, we use the first seven PCs, since they contain in total 98% of the overall information.

Fig. 6.15 shows the distribution of the seven cork classes represented by the first and second PCs. It can be seen from Fig.6.15(a) that four of the seven classes (*Corkstone, Beach, Desert*, and *Pebble*) are well separated from the others and one can expect optimal NN learning and good generalisation for them. However, the other three classes (*Precision, Speckled*, and *Lisbon*) are close to each other and their overlapping could reflect in deteriorating the NN performance. This problem is connected with the drawback of PCA, when applied for classification of the data discussed earlier in Section 2.1.3 – PCA is unsupervised transform, that does not consider the classes labels of the data.



(a) Cork classes means with 95% confidence intervals

(b) Scatter plot of the cork samples from the seven classes

Figure 6.15. Representation of the data by the first and second Principal Components.

- Linear Discriminant Analysis (LDA)

Similarly to PCA, this is a transformation that seeks new features that are linear combinations of the original ones. LDA was discussed in Section 2.1.3.

In general, the number of the new features is always one less than the number of classes. In our case, the dimensionality of the feature space is in fact reduced from 33 to 6 without considerable loss of information about the class separability (Dillon and Goldstein, 1984; Theodoridis and Koutroumbas, 2006).

Here, we do not standardise the data before the LDA (as we do for PCA), because in this case, this would affect the transform, and instead, we standardise the data after the LDA transform. This is needed, since features with higher values would affect more the Mean Squared Error (MSE) given with (2.7), and features with low ones will be neglected during training.

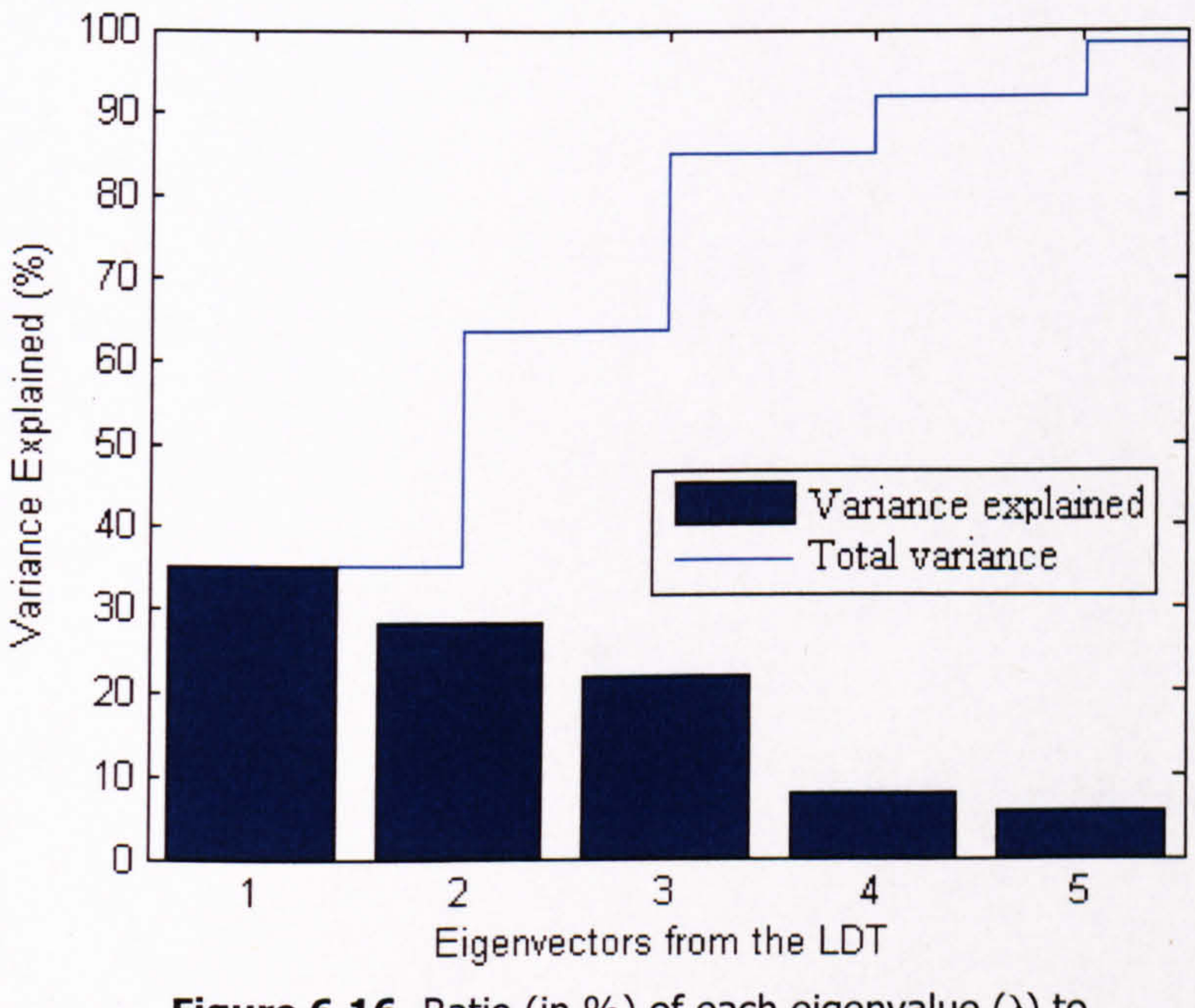
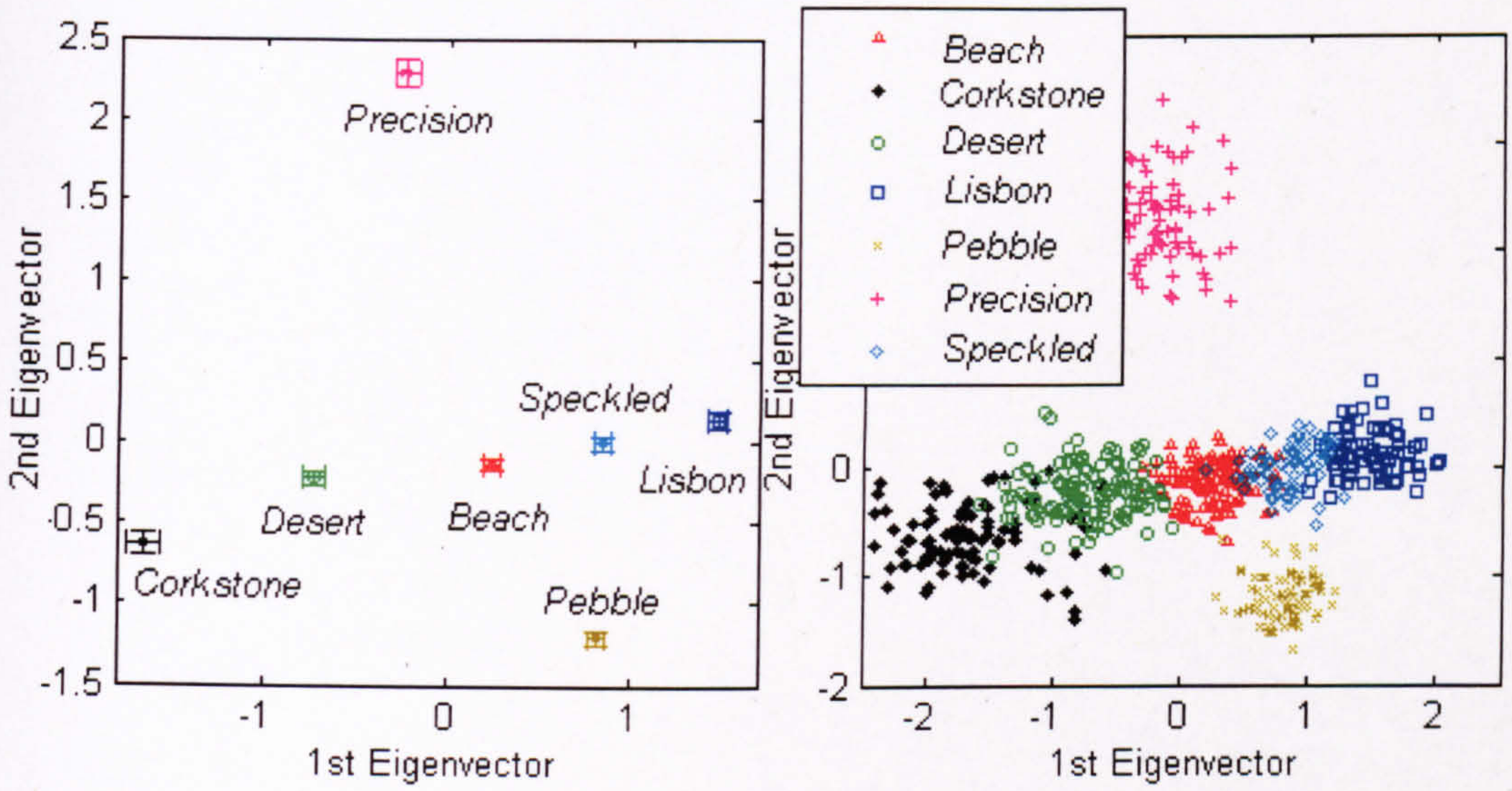


Figure 6.16. Ratio (in %) of each eigenvalue (λ) to the eigenvalues sum: $\lambda_j / \sum_{i=1}^{6} \lambda_i, j = 1,...,6$.

Fig. 6.16 shows the contribution to the sum (in percentage) of each one of the six eigenvalues. One can see that the first five eigenvalues encounter for 98.5% of the total eigenvalue sum, leaving only 1.5% for the last sixth one. In other words, the first

five eigenvectors contain 98.5% of the information in the original data. The scatter plot and the class means with 95% confidence intervals of the data transformed with the LDA are shown in Fig. 6.17. One can see from Fig. 6.17(a) that the within-class variance is minimised, resulting in small confidence intervals for the class means. On the other hand, the between-class variance is maximised and all seven classes are well separated, especially if compared with the data shown in Fig. 6.13.



(a) Cork classes means with 95% confidence intervals

(b) Scatter plot of the cork samples from the seven classes

Figure 6.17. The features, projected onto the first two eigenvectors of the LDA (the best two features of the transformed data).

6.3.3 Results and Discussion

We employed three different topologies (with biases) for the NNs and used different coding of the seven classes of interest. In the first case we employed three neurons in the output layer (with a *Heaviside* transfer function). The seven classes were coded as binary combinations of the three neurons ('*1-of-c*' coding, as proposed in Bishop, 1995), e.g., *Beach* was coded as (0, 0, 0), *Corkstone* as (1, 0, 0), etc. The last, (8th) combination (1, 1, 1) was simply not used. In the second designed topology the output layer contained only one neuron (with *Tanh* transfer function and continuous output). Since the *Tanh* function has values in [-1, 1], we coded the seven classes as (-0.8571, -

0.5714, -0.2857, 0, 0.2857, 0.5741, 0.8571) respectively. When assessing the system generalization abilities, we considered each testing sample as correctly classified if |output – target| < 0.1429. For the last topology, we used an output layer with seven neurons *Heaviside* transfer function. Each class was coded as a vector of binary values where only one output is 1, and all others are 0. For example, *Beach* was coded as (1, 0, 0, 0, 0, 0, 0), *Corkstone* as (0, 1, 0, 0, 0, 0, 0), etc.

The number of neurons in the input layer depends on the number of features (K) that characterize the problem samples (the choice of K is discussed in the *Feature Analysis* section). This number will be considered again in the next subsection but in a different aspect. After experimenting, we chose the number of neurons in the hidden layer to be N=7 utilizing the *rules of thumb* given by Heaton (2005). We employed the three different architectures for both datasets obtained by the PCA and LDA processing: K-7-3 (3-binary coding of the targets), K-7-1 (continuous coding of the targets), and K-7-7 (7-binary coding), where K is the number of features.

Results from NN training without validation

At the system evaluation stage we use 25% of the total data as a testing set, only 1/3 of which is present at the feature analysis phase (used in the preprocessing with PCA and LDA) and remaining 2/3 part of the test set is kept untouched. Further on, we consider the testing results as average test errors for both testing subsets.

Table 6.5 shows training and testing results for both topologies with $K = 7$ for the PCA dataset and $K = 6$ for the LDA dataset (these values are in accordance with the feature analysis in Section 6.3.2). In Table 6.5, MSE (mean value of the error function (2.7)) and its standard deviation (given in parentheses) average results from 50 runs are independently reported for each dataset. The minimal and maximal values obtained for the different runs are also shown in this table. The system is evaluated with the test rate, given by the percentage of correctly classified samples from the test set. Similarly, Table 6.6 shows results for the same topologies and datasets, with the only difference being the NN training technique. For the training of the NNs in Table 6.5, *GLPτS* is used, and for Table 6.6 – the Matlab implementation of the gradient-based *Levenberg-Marquardt* minimisation, denoted here as *Backpropagation* (BP). All test results are jointly illustrated in Fig. 6.18.

The analysis of the results given in Table 6.5, Table 6.6, and Fig. 6.18 led us to the following conclusions:

1. The generalisation abilities of the networks trained with *GLPτS* are strongly competitive with those of NN trained with BP algorithm. Indeed, the best mean testing results of 95% are obtained for NN trained with *GLPτS*, LDA dataset, and three binary outputs;

2. In general, the BP results are not as stable as the *GLPτS* ones, having significantly larger difference between attained minimal and maximal testing rate values. This might be due to entrapment of BP in local minima resulting in occasional very bad solutions;

3. The LDA dataset results have better test rate and smaller MSE than those corresponding to the PCA dataset. In our view this advantage is due to the LDA property to look for optimal class separability;

4. The three output binary coding targets leads to NN architecture with higher dimensionality, but gives better results than the continuous one. This is not surprising, since the binary coding of the targets provides linearly independent outputs for the different classes, which is more suitable for classification tasks (compared to the continuous coding), (Bishop, 1995). However, in the case of seven binary outputs, the NN performance deteriorates since the dimensionality is increased unnecessarily.

Table 6.5. Neural Network Training with *GLPτS* and Performance Evaluation: two different datasets with binary and continuous output.

Feature Set	Measure	Three outputs (*binary coding*)	One output (*continuous coding*)
PCA	MSE (std), [min, max]	0.052 (0.0094) [0.03, 0.074]	0.014 (0.0044) [0.011, 0.036]
	Test rate, [min, max]	86% [79%, 94%]	66% [41%, 77%]
LDA	MSE (std), [min, max]	0.0038 (0.0029) [0, 0.014]	0.0037 (0.0022) [0.0005, 0.0113]
	Test rate, [min, max]	95% [88%, 99%]	88% [74%, 98%]

Feature set: Principal Component Analysis (PCA) and Linear Discriminant Analysis – discussed in Section 2.1.3.

Table 6.6. Neural Network Training with BP and Performance Evaluation: two different datasets with binary and continuous output.

Feature Set	Measure	Three outputs (*binary coding*)	One output (*continuous coding*)
PCA	MSE (std), [min, max]	0.025 (0.053) [0.001, 0.245]	0.0489 (0.1473) [0.0113, 0.9116]
	Test rate, [min, max]	85% [39%, 94%]	71% [0%, 85%]
LDA	MSE (std), [min, max]	0.022 (0.06) [0, 0.244]	0.0049 (0.027) [0, 0.1939]
	Test rate, [min, max]	89% [40%, 98%]	90% [45%, 98%]

Feature set: Principal Component Analysis (PCA) and Linear Discriminant Analysis – discussed in Section 2.1.3.

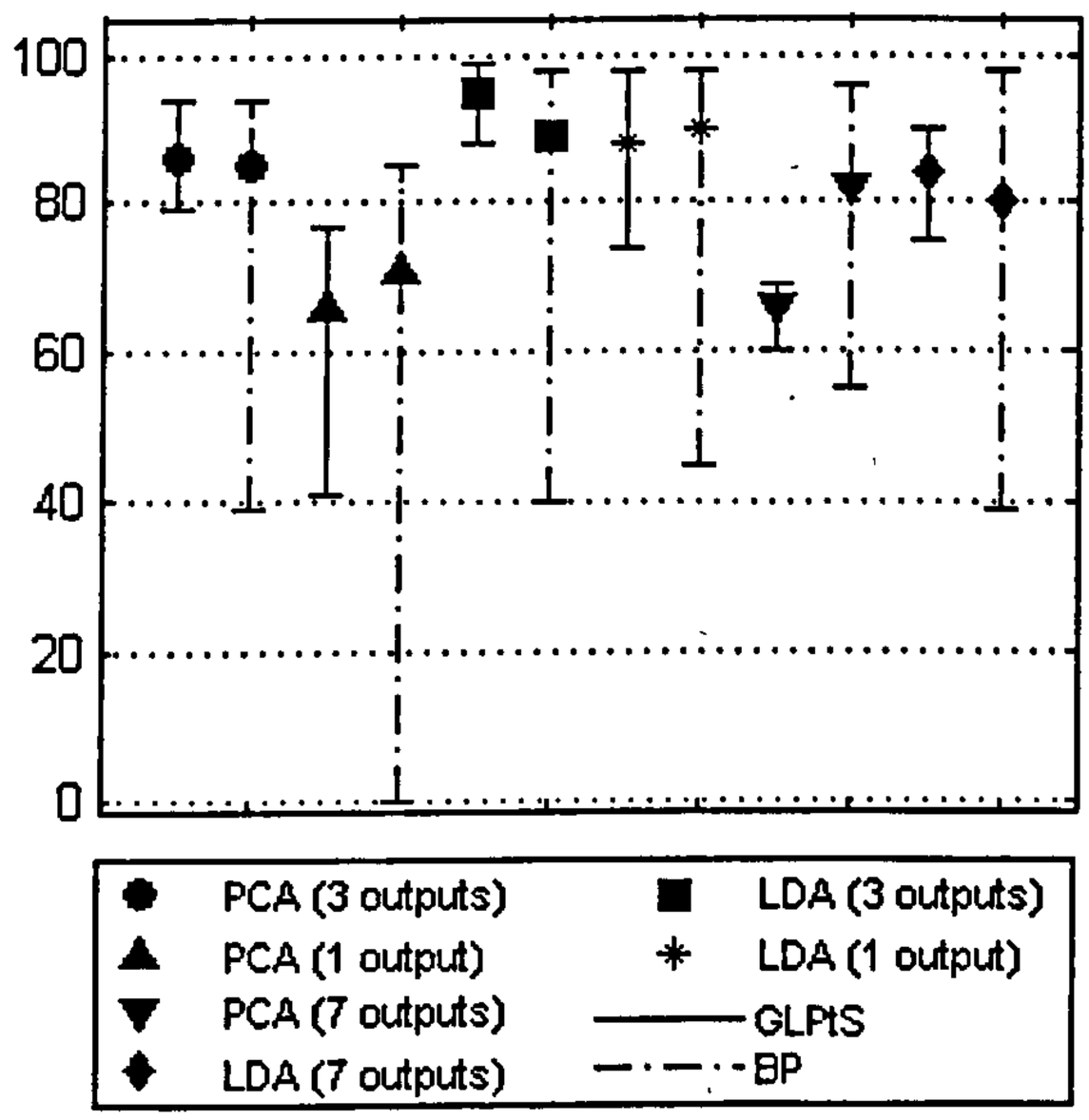


Figure 6.18. . Mean, min, and max test success rate (Table 6.5 and Table 6.6) for the experiments with different datasets, NN topologies, and learning approaches.

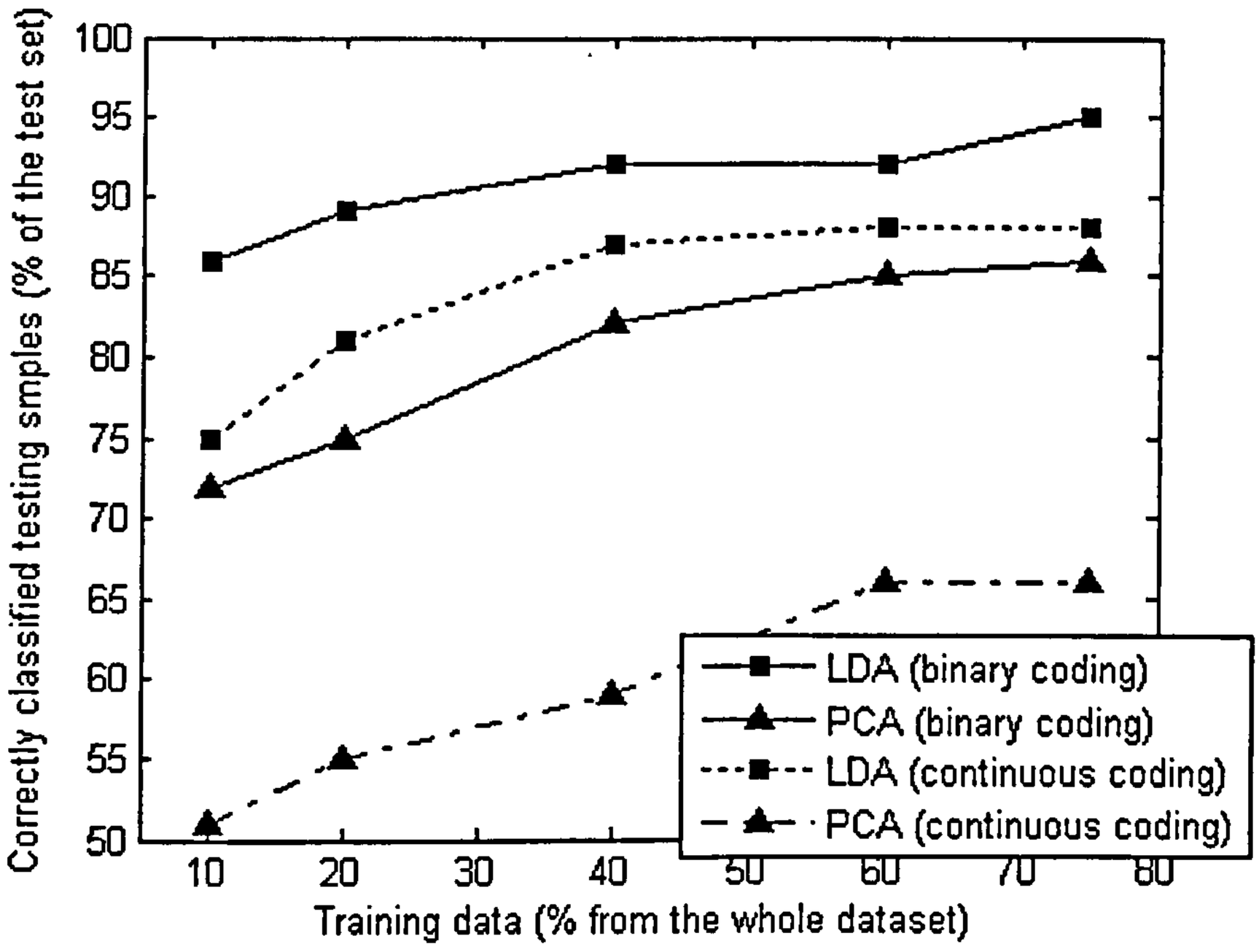


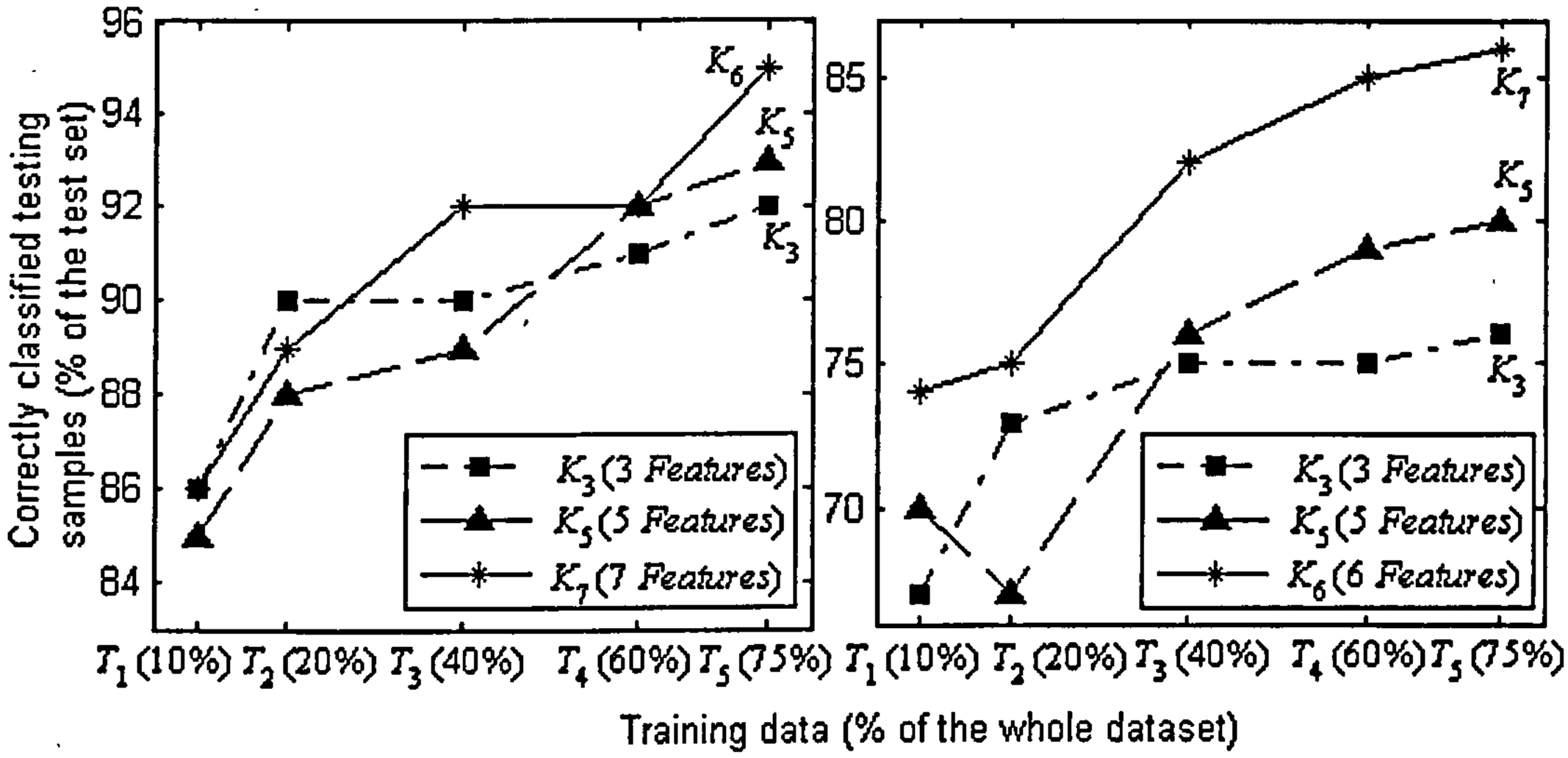
Figure 6.19. Test success rate for increasing number of samples in the training set. PCA and LDA feature sets are considered with binary and continuous coding of the classes.

Further on, we consider only the two cases with 3-binary and 1-continuous coding as well as NN trained with *GLPτS* as the most interesting and successful ones. Fig. 6.19 illustrates the testing success rate for the two NN topologies for both datasets (PCA and LDA) with respect to increasing number of training samples. The idea is to assess whether the number of samples and features used does give comprehensive and reliable information for the different cork classes. We use 25% of the whole data as an unseen testing subset and start increasing the percentage of used samples when training, keeping the NN topology unchanged. If the success rate increases proportionally to the increase of the training set size, then the features are considered reliable (Umbaugh, 2005). The results given in this figure are averaged over twenty runs. One can see from Fig. 6.19 that for both NN architectures LDA gives better generalisation results than PCA. One can also see that for all combinations (datasets and coding), the test rate graphs are ascendant, but the increase of number of training samples above 60% hardly brings any improvement of the test error success rate (with the exception of the LDA – binary architecture).

Number of Inputs

In Section 6.3.2 we discussed the LDA and PCA techniques and the way they transform the data. In this experiment we reduced the feature space significantly –

from 33 initial characteristics to 6 features as a result of applying LDA, and to 7 components in the PCA case. However, one may want to further reduce the feature space or to ask *how many* of the available ones after PCA and LDA transformations to be used? The following discussion is trying to give more detailed answer to this question.



(a) LDA feature set. (b) PCA feature set.

Figure 6.20. Test success rate for increasing number of samples in the training set and varying number of features. Only NN architectures with binary output are considered here.

If K is the number of inputs of the NN (the number of features), let us denote the cases of interest with $\{K_3, K_5, K_6, K_7\} = \{K = 3, K = 5, K = 6, K = 7\}$. The appropriate number of features needed for the NN training in the case of binary targets coding is empirically illustrated in Fig. 6.20. Fig. 6.20(a) shows the test success rate in the LDA case and Fig. 16.20(b) – the results for the PCA case (results are averaged over 20 runs). The NN generalisation abilities are assessed for varying number of features K and increasing number of training samples, denoted with $\{T_1, T_2, T_3, T_4, T_5\} = \{T = 10\%, T = 20\%, T = 40\%, T = 60\%, T = 75\%\}$, where the percentage is from the total dataset. One can see from Fig. 16.20 (a) that for T_1 and T_2 cases, K_3 outperforms both K_5 and K_6. This result shows that when the training set is *small*, the use of higher number of features might deteriorate the NN performance. As the size of the training set increases, the test rate for K_3 does not improve much (only from 90% to 92%), but

for K_5 and K_6 there is a significant improvement – from 85% to 95% for K_6 case, and from 88% to 93% for K_5 case. This observation confirms that as the number of training patterns increases, higher number of features reflects better the information available in the data. Finally, in the case of T_5, the performance rate is ordered accordingly to the number of features.

In the PCA case (Fig. 16.20(b)), the visible difference with the LDA (Fig. 16.20(a)) is the non-monotonic behaviour of K_5, which according to Umbaugh (2005) implies a need of higher number of features K. Otherwise, the overall performance of the graphs is similar, converging even earlier to the expected order (proportionally increasing test rate to the number of features).

The obtained results show that as the number of training samples increases, it is preferable to use higher number of features, in order to utilise better the information in the samples. However, if a smaller training set is available, it is reasonable to use *smaller* number K, since adding features does not improve the performance and might even deteriorate it (similar result is reported by Umbaugh (2005)).

Results from NN Training with Validation

The minimal MSE achieved is 0.0 in the case of LDA set and binary coding (Table 6.5). This shows that the exact learning of all training samples is possible in this case. We believe this is due to the very good discriminating properties of the obtained with LDA features, in combination with optimal NN architecture and learning technique. In order to avoid the *over-fitting* problem and to prevent the NN from *memorising* (rather than *generalising*), we perform training with validation, using 25% of the total data for this purpose (from the remaining part - 50% for training, and 25% for the testing discussed in previous section).

For each individual run of the NN learning process, we compute the test success rate with and without use of the validation set and afterwards compared and assessed the NN generalisation abilities for both cases. The following algorithm is employed:

1. At each iteration of the NN learning process, compute the MSE (given by (2.7)) for the validation set (MSEv);

2. Update MSEv_min which is the best MSEv achieved until the current iteration (update also the weights corresponding to MSEv_min);

3. Apply early stopping if MSEv $\geq$ (1 + t)*MSEv_min for more than *V* successive iterations (*t* is a validation error tolerance), save the weights and proceed to step 4, otherwise, repeat steps 1-3;

4. Compute the success rate *TRv* after testing the NN with the obtained weights from the early stopping (step 3);

5. Continue NN learning as if the early stopping is ignored until any of the other halting conditions is satisfied and save the weights (no validation);

6. Compute the success rate *TR* of the NN with the weights obtained in step 5.

We perform numerous experiments in order to select the most appropriate validation parameters – the validation error tolerance *t* and the number of successive iterations allowed without improvement of MSEv – *V*. Results from 100 runs for selected parameter combinations are given in Table 6.7. The first and second columns from this table show the values of *t* and *V* respectively. The third column shows the percentage of cases for which the validation condition from step 3 has been met. The next three columns compare the test success rate of both cases, with and without validation. It can be seen that in most of the cases *TRv* is lower than *TR*, showing that in our experiment the early stopping did not improve the success rate. The mean *TR* and *TRv* (of 100 runs) for all (*t*, *V*) combinations from Table 6.7 are illustrated graphically in Fig. 6.21.

Table 6.7. Neural Network Training with Validation – different values of the validation parameters. Results are out of 100.

t	*V*	Percentage of validation cases with:			
		Total	*TR* > *TRv*	*TR* < *TRv*	*TR* = *TRv*
t = 0.1	*V* = 10	89	63	16	10
	V = 20	67	38	10	19
	V = 50	22	14	5	3
t = 0.2	*V* = 10	70	46	14	10
	V = 20	61	34	10	17
	V = 50	12	6	4	2
t = 0.5	*V* = 10	33	23	6	4
	V = 20	28	15	7	6
	V = 50	4	1	1	2

TR – testing rate of the testing set; *TRv* – testing rate of the validation set; *t* - validation error tolerance; *V* - number of successive iterations allowed without improvement of MSEv (MSE of the validation set).

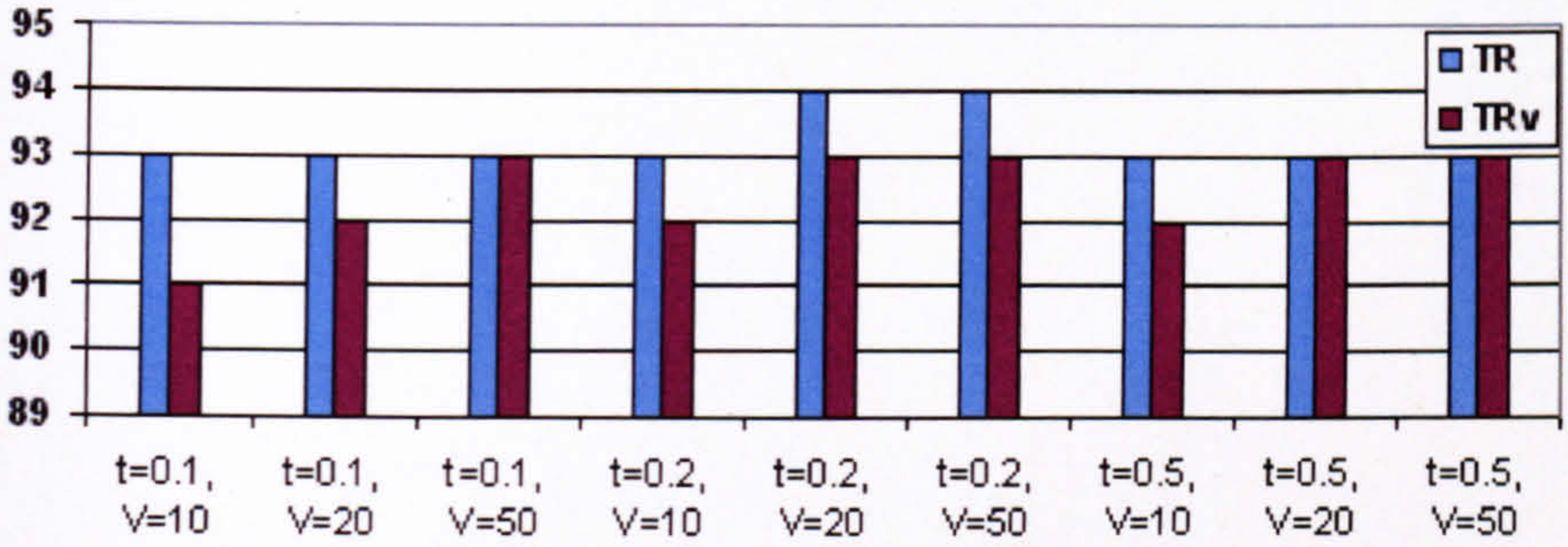
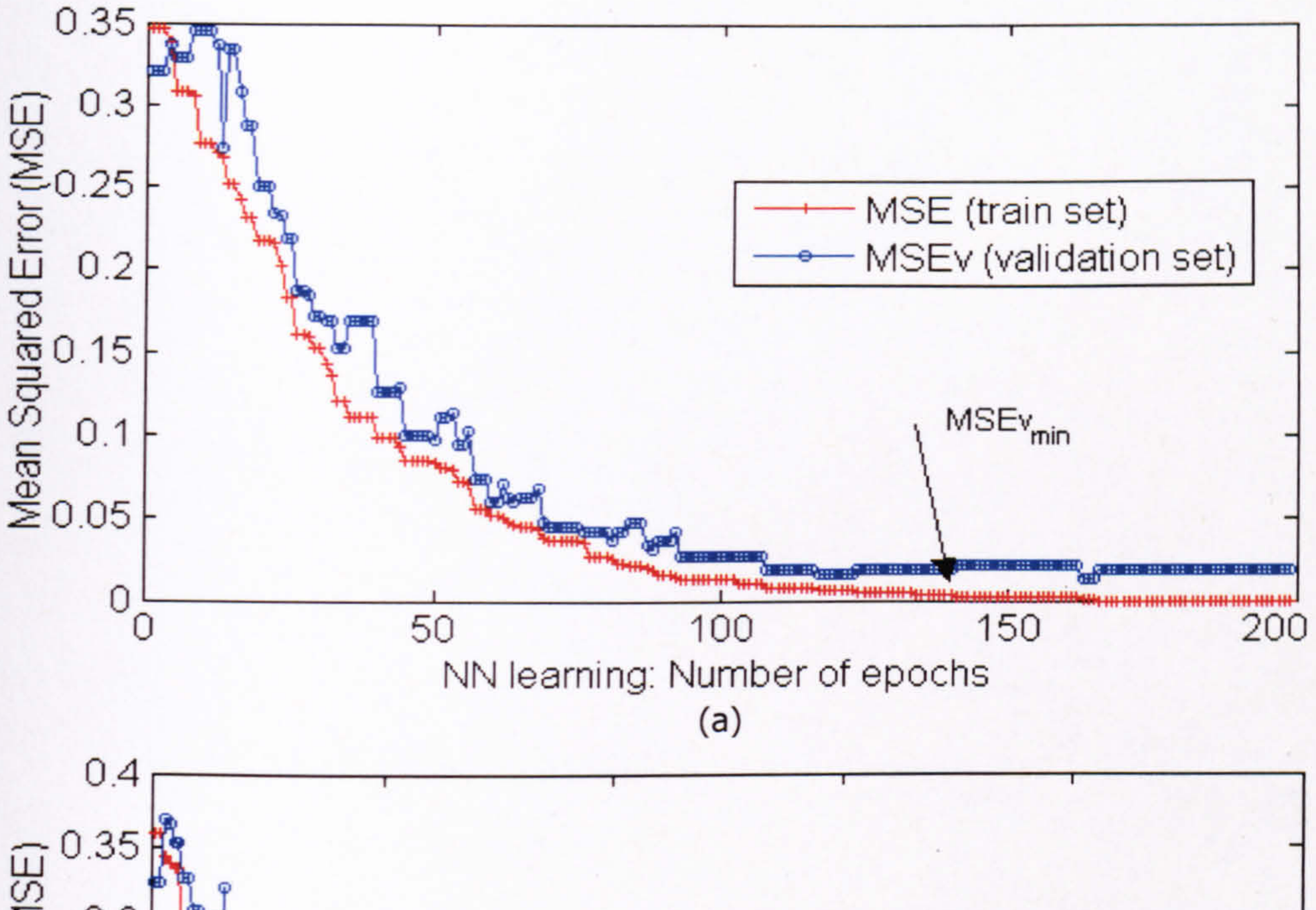


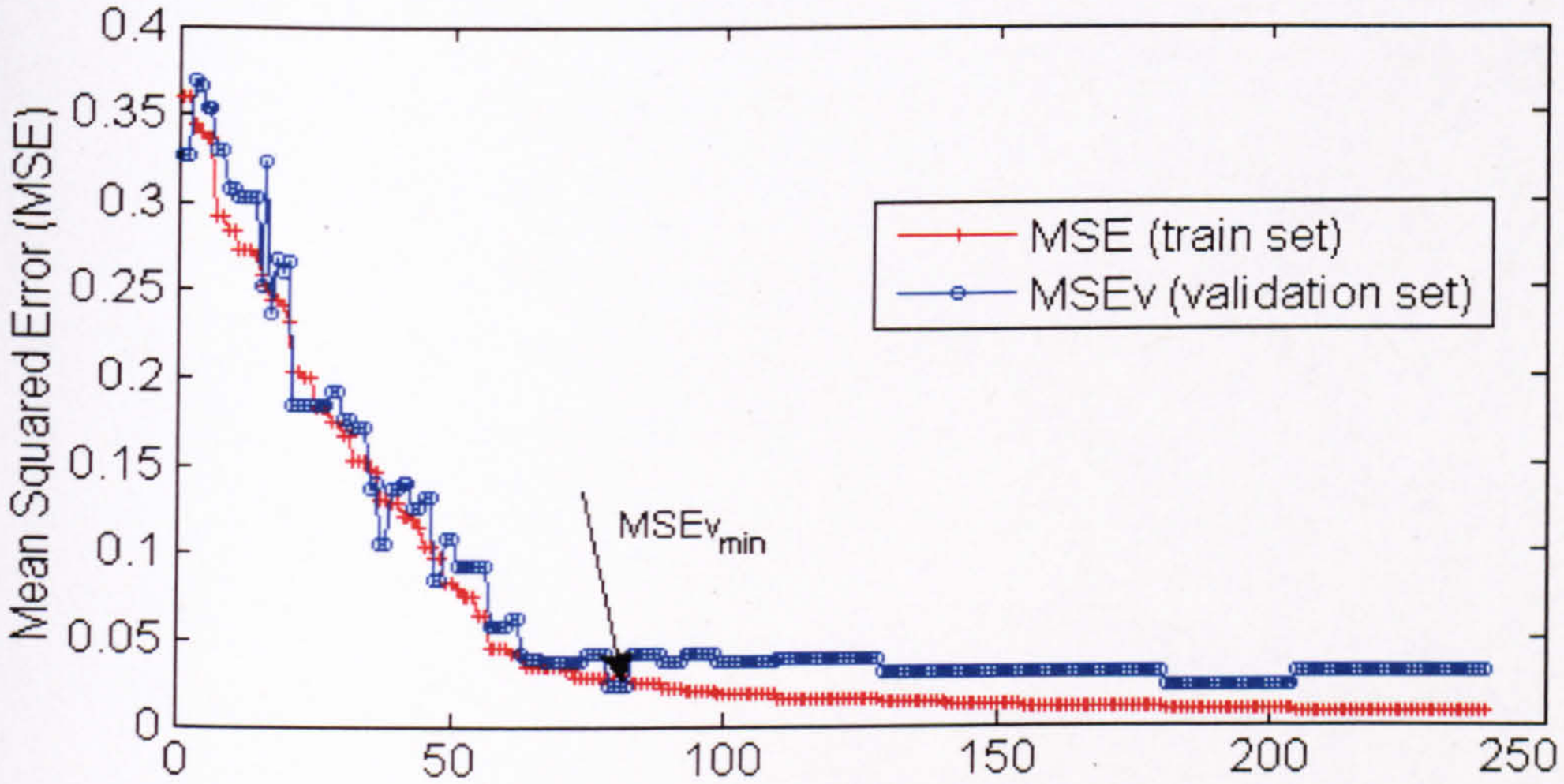
Figure 6.21. Test Rate for the weights obtained without validation (TR) and with validation (TRv). Different validation parameters are considered.

Since the minimisation process is stochastic and the NN performance is different for each independent training run - three demonstrative examples are shown in Fig. 6.22, ($t = 0.2$, $V = 20$). The arrow in this figure shows the iteration number for which the validation condition from step 3 is met. The NN performance is assessed in terms of MSE (training error) and MSEv (validation error). One can see from the cases shown in Fig. 6.22(a) and Fig. 6.22 (b), that after a certain iteration, the two graphs *separate* (not so clearly visible in Fig. 6.22(c)), with the MSEv graph starting to increase (indicating the beginning of the over-fitting process). In the first two cases, one might expect that the TRv would be better than TR. This is true for the NN from Fig. 6.22(a): $TR = 95\%$ and $TRv = 97\%$; but is not the case for the NN from Fig. 6.22(b), where $TR = 92\%$, and $TRv = 90\%$. For the third NN shown in Fig, 6.22(c), both test rates are equal, namely, $TR = TRv = 92\%$. We choose those cases only to demonstrate that there is no guarantee that the use of validation set and early stopping training technique will definitely improve the NN generalisation abilities in our experiment.

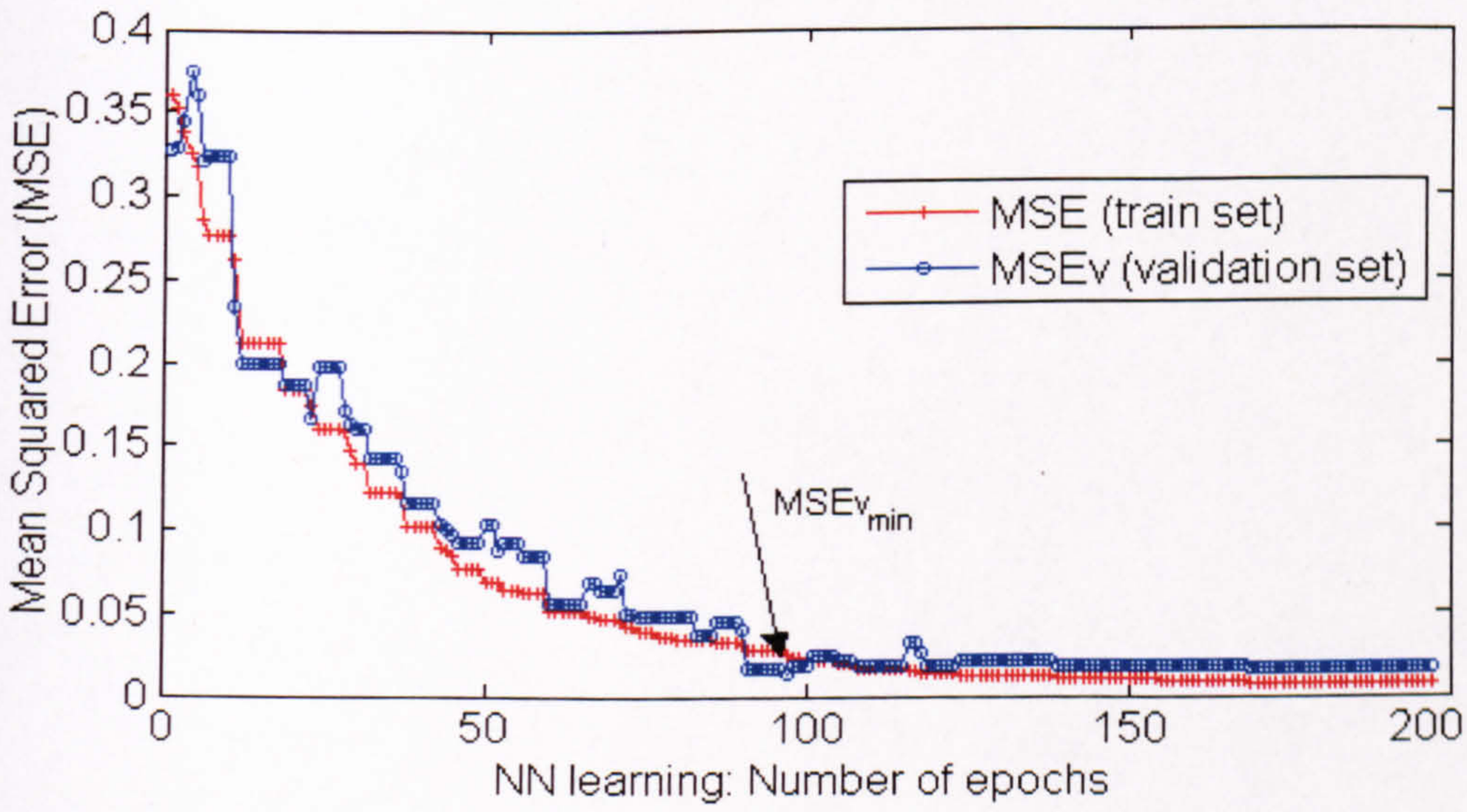
Results given in Table 6.5 and illustrated in Fig. 6.21 and Fig. 6.22, demonstrate that in some cases the use of a validation set and early stopping technique may improve the NN generalisation abilities, but this is not always the case. However, the use of early stopping approach in our case reduces significantly the computational effort without worsening the testing success rate significantly.



(a)



(b)



(c)

Figure 6.22. NN Learning – Mean Squared Error (MSE) for the train set and the validation set (MSEv).

Comparison with Results from Other Authors

As previously mentioned (Section 5.3), automated systems for cork products inspection have been developed only for cork stoppers and planks, but not for cork tiles. The work and results of other authors have been considered and discussed in detail in Section 5.3.

Straightforward comparison of our results with findings of other authors investigating similar cork inspection systems (discussed in Section 5.3), is a difficult task, because there are many differences in the parameters and techniques used. Some of the main differences are:

1. While natural cork stoppers are manufactured by punching a one-piece cork strip (and may have cracks and insect tunnels), cork tiles consist of various sizes of granules compressed together under high temperature (WWF/MEDPO, 2006), and cracks are not likely to be expected to appear. In Chang *et al.* (1997), Radeva *et al.* (2002), and Costa and Pereira (2006), the authors were looking mostly for pores, cracks and holes (and their sizes) in cork stoppers, whereas in our case gray density (texture) changes and overall appearance is of interest. We use feature generation techniques that capture the texture information of the images, while the other authors used features that aim to identify cracks and holes;

2. In Costa and Pereira (2006) the authors used only LDA as a classifier. In Chang *et al.* (1997), the investigation did not include any feature analysis techniques at all;

3. In our investigation, after using LDA and PCA to reduce the dimensionality of the problem space, we employ *GLPτS* method for optimal NN learning. This method has been investigated and developed in earlier stages of our research and has shown very good results for classification of other tasks (Georgieva and Jordanov, 2006; Jordanov and Georgieva, 2007). Other authors used different classifiers (Nearest Neighbor, Maximal Likelihood, Bayesian classifier (Radeva *et al.*, 2002), Fuzzy-neural networks (Chang *et al.*, 1997), and LDA (Costa and Pereira, 2006);

4. The size of training and testing datasets, as well as the size of the investigated images vary significantly in most of the cases.

Despite those differences, we could draw the following conclusions:

1. The test results of our Visual System (up to 95% success rate), significantly outperform the results (up to 58%), reported in Costa and Pereira (2006), discussed in Section 5.3;

2. Radeva *et al.* (2002) employed ICA (discussed in Section 5.3), for the data processing before the data is fed into a Bayesian classifier to achieve success rate of up to 98% for the classification of five types of cork stoppers. However, in our experiment we did not use ICA, but showed that LDA could reach up to 95% success rate for a task with seven classes, provided that the classifier is well designed and combined with a NN (trained with our *GLPτS* method). We claim that LDA is computationally efficient and very useful technique, provided that the other stages of the system process – feature generation and appropriate classifier design are thoroughly thought and investigated. On the other hand, ICA is not suitable for all types of data, since it imposes independence conditions on the features and also involves additional computational cost (Radeva *et al.*, 2002; Theodoridis and Koutroumbas, 2006).

3. Chang *et al.* (1997) investigated a classification task for eight different types of cork stoppers and reported only 6.7% misclassification rate, which is comparable with out results.

Considering the above-mentioned results, we can conclude that the intelligent inspection system trained with the proposed here *GLPτS*, has shown very good generalisation abilities.

6.4 Summary

In this chapter we have reported and discussed results from the investigation of an Intelligent Computer Vision System applied for recognition and classification of commercially available cork tiles. The system has been developed in three stages, named here as Experiment I, II, and III. In the final Experiment III, the system has been investigated in terms of statistical feature processing (dimensionality reduction techniques, number of features) and classifier design (NN target coding, architecture, complexity and performance). Testing success rate of up to 95% has been achieved, which in our view is due to several factors: combination of feature generation techniques; application of Principal Component Analysis and Linear Discriminant

Analysis, which appeared to be very efficient for preprocessing the data; and use of suitable NN design and learning method. The NNs have been trained with the proposed *GLPτS* Global Optimisation method and have demonstrated very good generalisation abilities when compared with the same NNs trained with *Backpropagation*. Early stopping technique has been also employed resulting in significant reduction of the computational load (although not improving the overall NN performance). The obtained and reported results have shown strongly competitive nature when compared with results from other authors investigating similar systems.

7 Conclusion and Future Directions

In this chapter the author's contributions are summarised, conclusion is given and possible future directions are presented.

7.1 Author's Contributions

The author's achievements have been the successful investigation, proposal, and application of novel meta-heuristic Global Optimisation techniques for supervised Neural Network learning. The author's specific contributions can be summarised as follows:

- Investigation, development, and proposal of novel Global Optimisation methods; Extensive evaluation of the proposed methods by testing on a number of multiminima multidimensional mathematical functions; Comparison of the obtained results with such obtained by other authors for a variety *state-of-the-art* Global Oprimisation methods;
- Application of the developed optimisation techniques to supervised Neural Network learning problems; Extensive testing on a number of benchmark classification and prediction problems, including real-world datasets;
- Design and implementation of an Intelligent Computer Vision System for automated inspection of cork tiles that involves three experiments with increasing complexity; Considering the following stages in the Computer Vision System: data acquisition; texture feature extraction and processing; classifier design and optimisation; system evaluation; Investigation of the system in terms of statistical feature processing (dimensionality reduction techniques, number of features) and classifier design (NN target coding, architecture, complexity and performance);
- Presenting the research results to five international conferences;
- Preparing five journal articles. One of them has been published in *IEEE Transactions on Neural Networks* and the other four are currently under review.

7.2 Conclusion

Initially, a novel Global Optimisation technique, called *LPτO*, has been proposed and investigated. It is based on *LPτ* Low-discrepancy Sequences and novel heuristic rules. *LPτO* has been discussed in detail and after the initial promising performance results, the technique has been hybridised with Nelder-Mead local search. The hybrid method, called *LPτNM*, has been investigated through a number of experiments, including stability with respect to the initial points, search domain, user-defined algorithm parameters, etc. The results from the testing on a number of mathematical functions and NN training benchmark problems have shown *LPτNM* to be reliable and promising GO technique. Nevertheless, with the increase of problems dimensionality, `computational load increases considerably and the performance of *LPτNM* deteriorates. To tackle this problem, a hybrid Global Optimisation method, called *GLPτS*, that combines Genetic Algorithms, *LPτO* method and Nelder-Mead simplex search, has been investigated and proposed. When compared with Genetic Algorithms, Evolutionary Programming, and Differential Evolution, *GLPτS* has demonstrated strongly competitive results in terms of both number of function evaluations and success rate. Subsequently, *GLPτS* has been applied for supervised NN training and tested on a number of benchmark problems. Based on the reported and discussed findings, it can be concluded that the investigated and proposed *GLPτS* technique is very competitive and demonstrates reliable performance.

Finally, an Intelligent Computer Vision System has been designed and investigated. It has been applied for a real-world problem for automated recognition and classification industrial products (in our case study – cork tiles). The classifier, employing supervised Neural Networks trained with *GLPτS*, has demonstrated reliable generalisation abilities. Early stopping technique has been also employed, resulting in significant reduction of the computational load. The obtained and reported results have shown strongly competitive nature when compared with results from other authors investigating similar systems.

7.3 Future Directions

Additional research could be very useful in the following directions:

- Further investigation and improvement of the proposed GO methods: algorithm parameters optimisation; further testing, investigation, and application;
- Further experiments with the Intelligent Computer Vision System: additional data processing and feature extraction techniques; further research of classifier design and system evaluation stages;
- Application of the Computer Vision System to problems for recognition and classification of other real-world objects, including 3D ones and colour images;
- Further research that would allow the Intelligent Computer Vision System to work online and be implemented in the real-world environment;

REFERENCES

[1] Adler D., "Genetic algorithms and simulated annealing: a marriage proposal", *IEEE International Conference on Neural Networks*, vol. 2, pp. 1104-1109, 1993.

[2] Alba E. and Chicano J.F., "Training neural networks with GA hybrid algorithms", *Lecture Notes in Computer Science*, vol. 3102, pp. 852-863, 2004.

[3] Ali M., Khompatraporn Ch., and Zabinsky Z., "A numerical evaluation of several stochastic algorithms on selected continuous global optimisation test problems", *Journal of Global Optimisation*, vol. 31, pp. 635-672, 2005.

[4] Ali M., Storey C., and Törn A., "Applications of stochastic global optimisation algorithms to practical problems", *Journal of Optimisation Theory and Applications*, vol. 95(3), pp. 545-563, 1997.

[5] Ali M. and Törn A., "Population set-based global optimisation algorithms: some modifications and numerical studies", *Computers and Operations Research*, vol. 31, pp. 1703-1725, 2004.

[6] Arora J.S., Elwakeil O.A., and Chahande A.I., "Global optimisation methods for engineering applications: a review", *Structural* Optimisation, vol. 9, pp/ 137-159, 1995.

[7] Bagirov A., Rubinov A., and Zhang J., "Local optimisation method with global multidimensional search", *Journal of Global Optimisation*, vol. 32, pp. 161-179, 2005.

[8] Barhen J., Protopopescu V., and Reister D., "TRUST: a deterministic algorithm for global optimisation", *Science*, vol. 276, pp. 1094-1097, 1997.

[9] Battiti R. and Tecchiolli G., "Training neural nets with the reactive tabu search", *IEEE Transactions on Neural Networks*, vol. 6, pp. 1185-1200, 1995.

[10] Baum E.B. and Haussler D., "What size net gives valid generalization", *Neural Computation*, vol. 1(1), 151-160, 1989.

[11] Bessaou M. and Siarry P., "A genetic algorithm with real-value coding to optimise multimodal continuous functions", *Structural and Multidisciplinary Optimisation*, vol. 23, pp. 63-74, 2001.

[12] Bianchini M. and Gori M., "Optimal learning in artificial neural networks: A review of theoretical results", *Neurocomputing*, vol. 13, pp. 313-346, 1996.

[13] Bilbro G.L. and Wesley S.E., "Optimisation of functions with many minima", *IEEE Transactions on Systems, Man, and Cybernetics*, vol. 21(4), pp. 840-49, 1991.

[14] Bishop C., *Neural networks for pattern recognition*, Clarendon Press Oxford, 1995.

[15] Björkman M. and Holmström K., "Global optimisation of costly nonconvex functions using radial basis functions", *Optimisation and Engineering*, vol. 1, pp. 373-397, 2000.

[16] Bortoletti A., Fiore C., Fanelli S., and Zellini P., "A new class of Quasi-Newtonian methods for optimal learning in MLP-Networks", *IEEE Transactions on Neural Networks*, vol. 14, pp. 263-273, 2003.

[17] Bratley P. and Fox B., "Implementation and tests of low-discrepancy sequences", *ACM Transactions on Modelling and Computer Simulation*, vol. 2, pp. 195-213, 1992.

[18] Burke L. and Ignizio J., "A practical overview of neural networks", Journal of Intelligent Manufacturing, vol. 8, pp. 157-165, 1997.

[19] Cetin B.C., Barhen J., and Burdick J.W., "Terminal repeller unconstrained subenergy tunnelling (TRUST) for fast global optimisation", *Journal of Optimisation Theory and Applications*, vol. 77(1), pp. 97-126, 1993.

[20] Chalup S. and Maire F., "A study of hill climbing algorithms for neural networks training", *Proc. 1999 Congress of Evolutionary Computations*, vol. 3, pp. 2014-2021, 1999.

[21] Chang J., Han G., Valverde J.M., Grisworld N.C., Duque-Carrillo J.-F., and Sanchez-Sinencio E. "Cork cork quality classification system using a unified image processing and fuzzy-neural network methodology", *IEEE Transactions in Neural Networks*, vol. 8, pp. 964-974, 1997.

[22] Chapelle O., Schoelkopf and Zien A. (eds.), *Semi-supervised learning*, MIT Press: Cambridge, MA, 2005.

[23] Chelouah R. and Siarry P., "A continuous genetic algorithm designed for the global optimisation of multimodal functions", *Journal of Heuristics*, vol. 6, pp. 191-213, 2000a.

[24] Chelouah R. and Siarry P., "Tabu search applied to global optimization", *European Journal of Operational Research*, vol. 123, pp. 256-270, 2000b.

[25] Chelouah, R. and Siarry, P., "Genetic and Nelder-Mead algorithms hybridiscd for a more accurate global optimisation of continuous multidimensional functions", *European Journal of Operational Research*, vol. 148, pp. 335-348, 2003.

[26] Chelouah R. and Siarry P., "A hybrid method combining continuous tabu search and Nelder-Mead simplex algorithms for the global optimisation of multiminima functions", *European Journal of Operational Research*, vol. 161, pp. 636-654, 2005.

[27] Chin R.T., "Automated visual inspection: 1981 to 1987", *Computer Vision, Graphics, and Image Processing*, vol. 41, pp. 346-381, 1988.

[28] Clerc M., *Particle swarm optimisation*, ISTE Publishing Company, 2006.

[29] Costa A. and Pereira H., "Decision rules for computer-vision quality classification of wine natural cork stoppers", *Am. J. Enology and Viticulture*, vol. 57, pp. 210-219, 2006.

[30] Davies E.R., *Machine vision: theory, algorithms, practicalities*. Morgan Kaufmann, 2005.

[31] De Jong, *Evolutionary computation*, MIT Press: Cambridge; 2006.

[32] Dillon W.R. and Goldstein M., *Multivariate analysis: methods and applications*, Wiley, New York, 2nd edition, 1984.

[33] Egmont-Petersen M., Ridder D., and Handels H., "Image processing with neural networks – a review", *Pattern recognition*, vol. 35, pp. 2279-2301, 2002.

[34] Engelbrecht A. P., *Computational intelligence*, John Wiley & Sons, Ltd, 2002.

[35] Faure H., "Discrépance de suites associées à un système de numération (en dimension *s*)", *Acta Arithmetica*, vol. 41, pp. 337-351, 1982.

[36] Fletcher R., *Practical methods of optimisation*, Wiley, 2nd edition, 2000.

[37] Feo T.A. and Resende M.G.C., "Greedy randomized adaptive search procedures", *Journal of Global Optimziation*, vol. 6, pp. 109-133, 1995.

[38] Festa P. and Resende M.G.C., "GRASP: an annotated bibliography", *AT&T Labs research Technical Report:* 00.1.1, 2000.

[39] Fischer M. and Gopal S, "Artificial neural networks: a new approach to modelling interregional telecommunication flows", *Journal of Regional Science*, vol. 34(4), pp. 503-527, 1994.

[40] Fogel D.B., Wasson E.C., Boughton E.M., Porto V.W., and Shively J., "Initial results of training neural networks to detect breast cancer using evolutionary programming," *Control and Cybernetics*, vol. 26(3), pp. 497-510, 1997.

[41] Georgieva A., Jordanov I., "A hybrid heuristic method for global optimisation", *Proc. Fifth International Conference on Hybrid Intelligent Systems, IEEE Computer Society*, pp. 503-505, 2005a.

[42] Georgieva A. and Jordanov I., "A hybrid method for stochastic global optimisation using low-discrepancy sequences of points", *Journal of Global Optimisation*, submitted 2005b.

[43] Georgieva A. and Jordanov I., "Supervised neural network training with hybrid global optimisation technique", *Proc. IEEE World Congress on Computational Intelligence*, Canada, pp. 6433-6440, 2006.

[44] Georgieva A., Jordanov I., "Global optimisation heuristic based on low-discrepancy sequences and genetic algorithms", *European Journal of Operational Research*, submitted 2007a.

[45] Georgieva A., Jordanov I., "Image processing techniques for cork tiles classification", *The 2007 IEEE International Conference on Signal Processing and Communications*, Dubai, 2007b (to appear).

[46] Georgieva A. and Jordanov I., "Intelligent visual recognition and product classification with neural networks", *IEEE Transactions on Neural* Networks, submitted 2007c.

[47] Georgieva A., Jordanov I., Rafik T., "Neural networks applied for cork tiles image classification", *Proc. IEEE Symposium on Computational Intelligence in Image and Signal Processing*, USA, pp. 232-239, 2007.

[48] Glover F., "Tabu search. Part I", *ORSA Journal on Computing*, vol. 1(3), pp. 190-206, 1989.

[49] Glover F., "Tabu search. Part II", *ORSA Journal on Computing*, vol. 2(1), pp. 4-32, 1990.

[50] Glover F., "A tamplate for scatter search and path relinking", In Hao, J.-K. *et al.* (eds.), *Artificial evolution*, Lecture Notes in Computer Science, 1363, Springer, pp. 13-54, 1998.

[51] Glover F., and Laguna M., *Tabu search*, Kluwer Academic Publishers, 1997.

[52] Glover F., Kochenberger G.A. (eds.), *Handbook of meta-heuristics*, Kluwer Academic Publishers, 2002.

[53] Goldberg D.E., Genetic algorithms *in search, optimisation; and machine learning*, Addison-Wesley, 1989.

[54] Gori M. and Tesi A., "On the problem of local minima in backpropagation", *IEEE Transactions on Pattern and Machine Intelligence*, vol. 14, pp. 76-85, 1992.

[55] Graves M. and Batchelor B. (eds.), *Machine vision for the inspection of natural products"*, Springer, 2003.

[56] Hagan M., Demuth H., and Beale M., *Neural network design*, Boston, MA: PWS Publishing, 1996.

[57] Hagan M. and Menhaj M., "Training feed-forward networks with the Marquardt algorithm", *IEEE Transactions on Neural Networks*, vol. 5(6), pp. 989-993, 1994.

[58] Haralick R.M., Shanmugam K., and Dinstein I., "Textural features for image classification", *IEEE Transactions on Systems, Man, and Cybernetics*, vol. 3, pp. 610-621, 1973.

[59] Hart W. E., Krasnogor N., and Smith J.E. (eds.), *Recent advances in memetic algorithms*, In Series: Studies in Fuzziness and Soft Computing, vol. 166, 2005.

[60] Haykin S., *Neural networks - a comprehensive foundation*, Prentice-Hall, Inc., 1999.

[61] Heaton J., *Introduction to neural networks*, Heaton Research Inc., 2005.

[62] Hedar A.R. and Fukushima M., "Minimizing multimodal functions by simplex coding genetic algorithm", *Optimisation Methods and Software*, vol. 18, pp. 265-282, 2003.

[63] Hebb D.O., *The organization of behaviour*, New York: Wiley, 1949.

[64] Ho S.Y., Shu L.S., and Chen J.H., "Intelligent evolutionary algorithms for large parameter optimisation problems", *IEEE Transactions on Evolutionary Computation*, vol. 8(6), pp. 522-541, 2004.

[65] Hogg R.V. and Ledolter J., *Applied statistics for engineers and physical scientists*, 2nd edition, 1992.

[66] Hohil M.E., Liu D., and Smith S.H., "Solving *N*-bit parity problem using neural networks", *Neural Networks*, vol. 12, pp. 1321-1323, 1999.

[67] Hole A., "Vapnik-Chervonenkis generalization bounds for real valued neural networks", *Neural Computation*, vol. 8, pp. 1277-1299, 1996.

[68] Holland J.H., *Adaptation in natural and artificial systems*, Ann Arbor, MI: University of Michigan Press, 1975.

[69] Hornik K., Stinchcombe M., and White H., "Multilayer feed-forward networks are universal approximators", *Neural Networks*, vol. 2(5), pp. 359-366, 1989.

[70] Houck C., Joine J., and Kay M., "A genetic algorithm for function optimisation: a Matlab implementation", *Technical Report: NCSU-IE-TR-95-09, North Caroline State* University, 1995.

[71] Hsieh P.-F., Wang D.-S., and Hsu Ch.-W., "A linear feature extraction for multiclass classification problems based on class mean and covariance discriminant function", *IEEE Transactions on Pattern Recognition and Machine Intelligence*, vol. 28(2), pp. 223-235, 2006.

[72] Huang G.-B., Zhu Q.-Y., and Siew C.-K., "Real-time learning capability of neural networks", *IEEE Transactions on Neural Networks*, vol. 17, pp. 863-878, 2006.

[73] Hush D., Horne B., and Salas J., "Error surfaces for multiplayer perceptrons", *IEEE Transactions Systems, Man, and Cybernetics*, vol. 22, pp. 1152-1161, 1992.

[74] Huyer W. and Neumaier A., "Global optimisation by multilevel coordinate search", *Journal of Global Optimisation*, vol. 14, pp. 331-355, 1999.

[75] Ingber L., "Very fast simulated annealing", *Mathematical and Computer Modelling*, vol. 12(8), pp. 967-973, 1989.

[76] Joins J. and Kay M., "Utilizing hybrid genetic algorithms", In Sarker *et al.* (eds.), *Evolutionary optimisation*, Kluwer Academic Publishers, pp. 199-228, 2002.

[77] Jones D.R., Perttunen C.D., and Stuckman B.E., "Lipschitzian optimisation without the Lipschitz constant", *Journal of Optimisation Theory and Application*, vol. 79, pp. 157-181, 1993.

[78] Jones D.R., Schonlau M., and Welch W.J., "Efficient global optimisation of expensive black-box functions", *Journal of Global* Optimziation, vol. 13, pp. 455-492, 1998.

[79] Jordanov I. and Georgieva A., "Global search approach in supervised neural networks training", *Proc. IADAT-micv2005*, pp. 39-43, 2005.

[80] Jordanov I., Georgieva A., "Neural network learning with global heuristic search", *IEEE Transactions on Neural Networks*, 18(3), pp. 937-942, 2007a.

[81] Jordanov I., Georgieva A., "Stochastic genetic algorithm (StGA) – a critical review", *IEEE Transactions on Neural Networks*, submitted, 2007b.

[82] Kirkpatrick S., Gelatt C.D., and Vecchi M.P., "Optimisation by simulated annealing", *Science*, vol. 220, pp. 671-680, 1983.

[83] Kucherenko S. and Sytsko Y. "Application of deterministic low-discrepancy sequences in global optimisation", *Computational Optimisation and Applications*, vol. 30, pp. 297-318, 2005.

[84] Laguna M. and Martí R., "Experimental testing of advanced scatter search designs for global optimisation of multimodal functions", *Journal of Global Optimisation*, vol. 33, pp. 235-255, 2005.

[85] Laws K.I., "Rapid texture identification", *SPIE - Image Proc. for Missile Guidance*, vol. 238, pp. 376-380, 1980.

[86] Lee D.W., Choi H.J., and Lee J., "A regularized line search tunnelling for efficient neural network learning", *Lecture Notes in Computer Science*, vol. 3173, pp. 239-243, 2004.

[87] Lee C.Y. and Yao X., "Evolutionary programming using mutations based Lévy probability distribution", *IEEE Transactions on Evolutionary Computation*, vol. 8(1), pp. 1-13, 2004.

[88] Leung C.S, Tsoi A., and Chan L.W., "Two regularizers for recursive least squared algorithms in feedforward multilayered neural networks", *IEEE Transactions Neural Networks*, vol. 12, pp. 1314-1332, 2001.

[89] Leung Y.W. and Wang Y., "An orthogonal genetic algorithm with quantization for global numerical optimisation". *IEEE Transactions on Evolutionary Computation*, vol. 5, pp. 41-53, 2001.

[90] Liberti L. and Kucherenko S. "Comparison of deterministic and stochastic approaches to global optimisation", *International Trans. in Operational Research*, vol. 12, pp. 263-285, 2005.

[91] Litnetski V.V., and Abramzon B.M., "MARS – a multistart adaptive random search method for global constrained optimisation in engineering applications", *Engineering Optimisation*, vol. 30, pp. 125-154, 1998.

[92] Liu X. and Wang D., "Texture classification using spectral histograms", *IEEE Transactions on Image Processing*, vol. 12(6), pp. 661-671, 2003.

[93] Ludemir T.B., Yamazaki A., and Zanchettin C., "An optimisation methodology for neural network weights and architectures", *IEEE Transactions on Neural Networks*, vol. 17(6), pp. 1452-1459, 2006.

[94] Magoulas G.D., Vrahatis M.N., and Androulakis G.S., "On the alleviation of the problem of local minima in Back-propagation", *Nonlinear Analysis, Theory and Applications*, vol. 30, pp. 4545-4550, 1997.

[95] Man Z., Wu H.R., Liu S., and You X., "A new adaptive backpropagation algorithm based on Lyapunov stability theory for neural networks", *IEEE Transactions on Neural Networks*, vol. 17(6), pp. 1580-1591, 2006.

[96] Mathias K.E. and Whitley D., "Changing representations during search: a comparative study of delta coding", *Evolutionary Computation*, vol. 2(3), pp. 249-278, 1994.

[97] McCulloch W. S. and Pitts W. H., "A logical calculus of the ideas immanent in neural nets", *Bulletin of Mathematical Biophysics*, vol. 5, pp. 115-133, 1943.

[98] Mitchell M., *An introduction to genetic algorithms*, MIT Press: Massachusetts, 2001.

[99] Munteanu C. and Rosa A.C., "Adaptive reservoir evolutionary algorithm: an evolutionary on-line adaptation scheme for global function optimisation", *Journal of Heuristics*, vol. 10, pp. 555-586, 2004.

[100] Nelder J. and Mead R., "A simplex method for function minimization", *The Computer Journal*, vol. 7, pp. 308-313, 1965.

[101] Niederreiter H., "On a measure of denseness for sequences", *Colloquia Mathematica Sociatatis Janos. Bolyai*, vol. 34, pp. 1163-1208, 1981.

[102] Niederreiter H., *Random number generation and Quasi-Mounte Carlo methods,* Society for Industrial and Applied Mathematics, Philadelphia, Pennsylvania, 1992.

[103] Noman N. and Iba H., "Enhancing differential evolution performance with local search for high dimensional function optimisation", *Genetic and Evolutionary Computation Conference*, pp. 967-974, 2005.

[104] Oblow E.M., "SPT: a stochastic tunnelling algorithm for global optimisation", *Journal of Global Optimisation*, vol. 20, pp. 195-212, 2001.

[105] Ojala T. and Pietikäinen M., "Multiresolution gray-scale and rotation invariant texture classification with local binary patterns", *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 24(7), pp. 971-987, 2002.

[106] Pardalos P.M., Romeijn H.E., and Tuy H., "Recent developments and trends in global optimisation", *Journal of Computational and Applied Mathematics*, vol. 124, pp. 209-228, 2000.

[107] Peng H., Long F., and Ding Ch., "Feature selection based on mutual information: criteria of max-dependency, max-relevance, and min-relevance", *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 27(8), 1226-1237, 2005.

[108] Pereira H., Lopes F., and Graca J., "The evaluation of the quality of cork planks by image analysis", *Holzforshhung*, vol. 50, pp. 111-115, 1996.

[109] Plaginakos V.P., Magoulas G.D., and Vrahatis M.N., "Supervised training using global search methods", *Advances in Convex Analysis and Global*

Optimisation, Kluwer Acad. Publishers, Dordrecht, The Netherlands, pp. 421-432, 2001.

[110] Portuguese Cork Association (APCOR), http://www.realcork.org.

[111] Prechelt L., "Proben 1: a set of neural network benchmark problems and benchmarking rules", *Fakultät für Informatik, Universität Karlsruhe, Germany, Technical Report 21/94*, 1994.

[112] Price K.V., Storn R., and Lampinen J., *Differential evolution – a practical approach to global optimisation*, Springer, 2005.

[113] Puig D. and Garcia M.A., "Automatic texture feature selection for image pixel classification", *Pattern Recognition*, vol. 39, pp. 1996-2009, 2006.

[114] Quinlan J.R., *C4.5: Programs for machine learning*, Morgan Kaufmann, San Mateo, CA, 1993a.

[115] Quinlan J.R., "Combining instance-based and model-based learning", *Proc. ML'93 (ed. P.E. Utgoff), San Mateo: Morgan Kaufmann*, pp. 236-243, 1993b.

[116] Rabunal J.R. and Dorrado J. (eds.), *Artificial neural networks in real-life applications*, Hershey PA : Idea Group Pub, 2006.

[117] Radeva P., Bressan M., Tobar A., and Vitrià J., "Bayesian classification for inspection of industrial products", *Proc. of the 5th Catalonian conference on AI* – M.T. Escrig et al. (Eds.), pp. 399-407, 2002.

[118] Randen T. and Husøy, "Filtering for texture classification: a comparative study", *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 21, pp. 291-310, 1999.

[119] Reeves C.R. and Rowe J.E, *Genetic algorithms: principles and perspectives*. Kluwer Academic Publishers, 2002.

[120] Ribeiro C. and Hansen P. (eds.), *Essays and surveys in metaheuristics*, Kluwer Academic Publishers, 2002.

[121] Rocha M., Cortez P., and Neves J., "Evolutionary neural networks learning", *Lecture Notes in Computer Science*, vol. 2902, pp. 24-28, 2003.

[122] Rosenblatt F., *Principles of neurodynamics: perseptrons and the theory of brain mechanisms*, Washington DC: Spartan, 1962.

[123] Rudolph G., "Convergence analysis of canonical algorithms", *IEEE Transactions on Neural Networks*, vol. 5(1), pp. 96-101, 1994.

[124] Rumelhart D.E., Hinton G.E., and Williams R.J., "Learning internal representations by error propagation". *Nature*, vol. 323, pp. 533-536, 1986.

[125] Sexton R.S., Alidaee B., Dorsey R.E., and Johnson J.D., "Global optimisation for artificial neural networks: a tabu search application", *European Journal of Operational Research*, vol. 106, pp. 570-584, 1998.

[126] Shapiro L.G. and Stockman G.C., *Computer vision*, Prentice Hall, 2001.

[127] Siarry P., Berthiau G., Durbin F., and Haussy J., "Enhanced simulated annealing for globally minimizing functions of many continuous variables", *ACM Transactions on Mathematical Software*, vol. 23, pp. 209-228, 1997.

[128] Sierra A., and Echeverría A., "Evolutionary discriminant analysis", *IEEE Transactions on Evolutionary Computation*, vol. 10(1), pp. 81-92, 2006.

[129] Smagt P., "Minimization methods for training feed-forward neural networks", *Neural Networks*, vol. 7, pp. 1-11, 1994.

[130] Sobol' I.M., "On the systematic search in a hypercube", *SIAM Journal of Numerical Analysis*, vol. 16(5), pp. 790-792, 1979.

[131] Sobol' I.M., "Uniformly distributed points in a multidimensional", *Moscow: Znanie ("Mathematics and Cybernetics" series in Russian)*, 1985.

[132] Solnon Ch., "Boosting local search with artificial ants", *Lecture Notes in Computer Science* (T. Welsh, ed.), vol. 2239, pp. 620-624, 2001.

[133] Sonka M, Hlavac V., and Boyle R., *Image processing, analysis, and machine vision*, Thomson-Engineering, 3rd edition, 2007.

[134] Storn R., and Price K.V., "Differential evolution – a simple and efficient adaptive scheme for global optimisation over continuous spaces", *Technical Report TR-95-012, ICSI*, 1995.

[135] Stuckman B. and Easom E., "A comparison of Bayesian/sampling global optimisation techniques", *IEEE Transactions on Systems, Man, and Cybernetics*, vol. 22(5), pp. 1024-1032, 1992.

[136] Sutton R. and Barto A., *Reinforcement learning: an introduction*, MIT Press, 1998.

[137] Szu H.H. and Hartley R.L., "Fast simulated annealing", *Physics Letters A*, vol. 122, pp. 157-162, 1987.

[138] Teh Y.S. and Rangaiah G.P., "Tabu search for global optimisation of continuous functions with application to phase equilibrium calculations", *Computers and Chemical Engineering*, vol. 27, pp. 1665-1679, 2003.

[139] Theodoridis S. and Koutroumbas K., *Pattern recognition*, Academic Press, 3rd edition, 2006.

[140] Törn A. and Viitanen S., "Topographical global optimisation using pre-sampled points", *Journal of Global Optimisation*, vol. 5, pp. 267-276, 1994.

[141] Törn A. and Žilinskas A., *Global optimization*, Springer-Verlag: New York, 1989.

[142] Tu Z. and Lu Y., "A robust stochastic genetic algorithm (StGA) for global numerical optimisation", *IEEE Transactions on Evolutionary Computation*, vol. 8(5), pp. 456-470, 2004.

[143] Umbaugh S., *Computer imaging: digital image analysis and processing*, The CRC Press, 2005.

[144] Wang X., Tang Z., Tamura H., Ishii M., and Sun W., "An improved backpropagation algorithm to avoid the local minima problem", *Neurocomputing*, vol. 56, pp. 455-460, 2004.

[145] Widrow B. and Hoff M. E., "Adaptive switching circuits", *IRE WESCON Convention Record*, vol. 4, pp. 96-104, 1960.

[146] Wolpert D. and Macready W.G., "No free lunch theorems for optimisation", *IEEE Transactions on Evolutionary Computation*, vol. 1(1), pp. 67-82, 1997.

[147] WWF Report, "Cork screwed? Environmental and economic impacts of the cork stoppers market", *World Wide Fund for Nature / Mediterranean Programme Office (MEDPO)*, 2006.

[148] Yao X., "Optimisation by genetic annealing", *Proc. Of Second Australian Conf. On Neural Networks* (M. Jarbi, ed.), pp. 94-97, 1991.

[149] Yao X., "A new simulated annealing", *International Journal of Computer Mathematics*, vol. 56, pp. 161-168, 1995.

[150] Yao X., "Evolving artificial neural networks", *Proc. IEEE*, vol. 87(9), pp. 1423-1447, 1999.

[151] Yao X., "Evolutionary computation: a gentle introduction", In Sarker *et al.* (eds.), *Evolutionary optimisation*, Kluwer Academic Publishers, pp. 27-53, 2002.

[152] Yao X. and Liu Y., "Fast evolution strategies", *Control and Cybernetics*, vol. 26(3), pp. 467-496, 1997.

[153] Yao X., Liu Y., and Lin G., "Evolutionary programming made faster", *IEEE Transactions on Evolutionary Computation*, vol. 3(2), pp. 82-102, 1999.

[154] Yao Y., "Dynamic tunnelling algorithm for global optimisation", *IEEE Transactions on Systems, Man, and Cybernetics*, vol. 19(5), pp. 1222-1231, 1989.

[155] Yiu K.F.C., Liu Y., and Teo K.L., "A hybrid descent method for global optimisation", *Journal of Global Optimisation*, vol. 28, pp. 229-238, 2004.

[156] Zheng R.T., Ngo N.Q., Shum P., and Tjin S.C., "A staged continuous tabu search algorithm for the global optimisation and its applications to the design of fibre bragg gratings", *Computational Optimisation and Applications*, vol. 30, pp. 319-335, 2005.

[157] Zielinski K., Weitkemper P., Laur R., and Kammeyer K.D., "Parameter study for differential evolution using a power allocation problem including interference cancellation", *Proc. IEEE Congress on Evolutionary Computation*, pp. 6748-6755, 2006.

APPENDIX A: GLOBAL OPTIMISATION TEST FUNCTIONS

(Part of the figures below are courtesy of Mr Lubomir Andreev and Mr Daniel Kirilov, Erasmus students form the Technical University of Sofia, Bulgaria)

A. TYPICAL LOWER DIMENSIONAL TEST FUNCTIONS (2–10 DIMENSIONS)

- *Shubert* (Fig. A1), dimensionality: $n = 2$.

$f(x, y) = \alpha(x)\beta(y)$, where $\alpha(x) = \sum_{i=1}^{5} i\cos[(i+1)x+i]$ and $\beta(y) = \sum_{i=1}^{5} i\cos[(i+1)y+i]$, x, $y \in [-10, 10]$.

Number of Local Minima (LM) = 760, Number of Global Minima (GM) = 18, f_{min} = -186.73091 in the points:

(-7.70838, -7.08351); (-7.08351, -7.70838); (-7.70838, 5.48285);
(5.48285, -7.70838); (-7.70838, -0.80033); (-0.80033, -7.70838);
(-7.08351, 4.85803); (4.85803, -7.08351); (-1.42513, -0.80033);
(-0.80033, -1.42513); (-1.42513, 5.48285); (5.48285, -1.42513);
(-1.42513, -7.08351); (-7.08351, -1.42513,); (-0.80033, 4.85803);
(4.85803, -0.80033,); (4.85803, 5.48285); (5.48285, 4.85803);

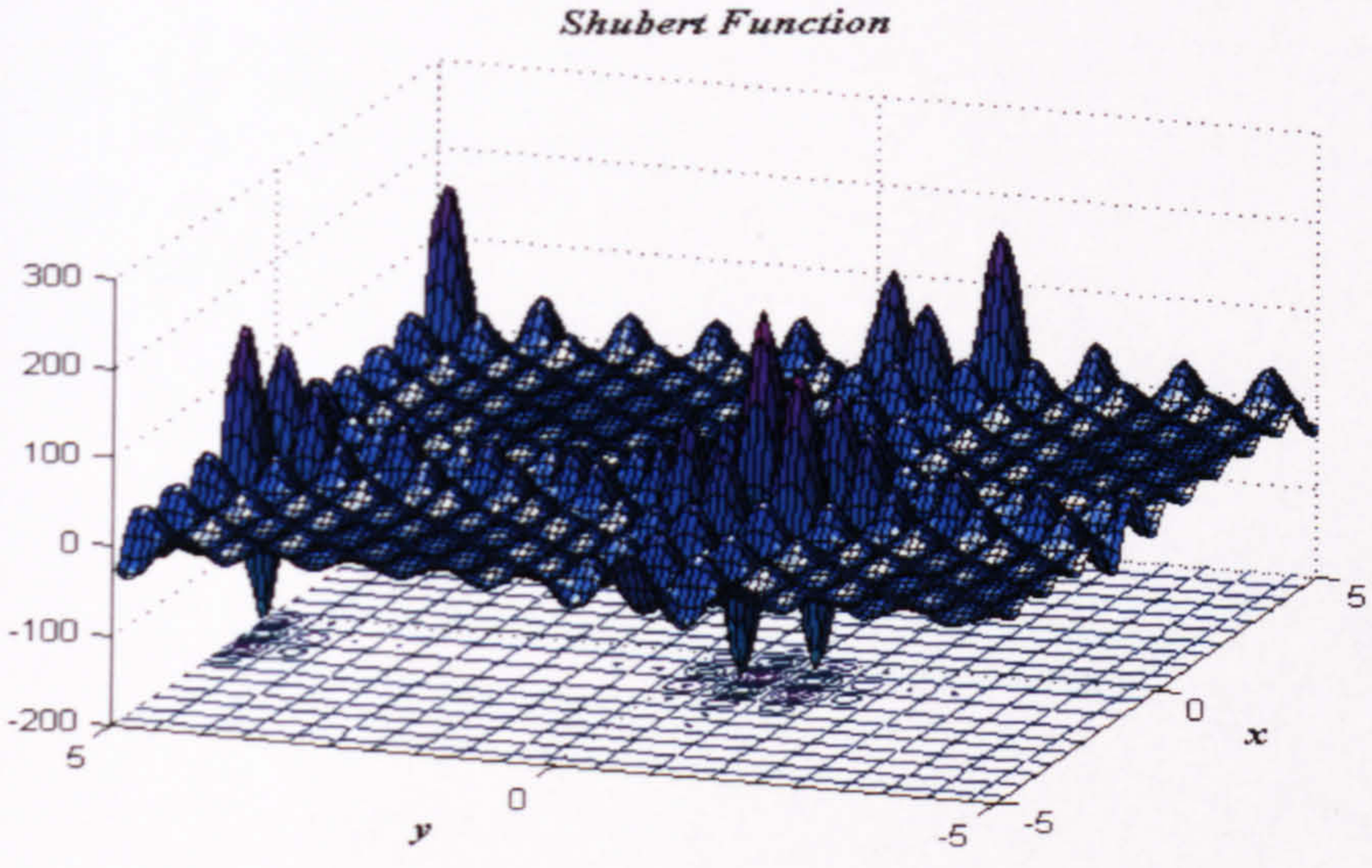


Figure A1. *Shubert* function in $[-5, 5]^2$.

- *Six-hump Camelback* (Fig. A2), dimensionality $n = 2$.

$$f(x,y) = \left(4 - 2.1x^2 + \frac{1}{3}x^4\right)x^2 + xy + \left(-4 + 4y^2\right)y^2, (x, y) \in [-3, 3]\text{x}[-2, 2].$$

LM = 6, GM = 2, f_{min} = -1.0316 in the points:
(–0.0898, 0.7126) and (0.0898, –0.7126).

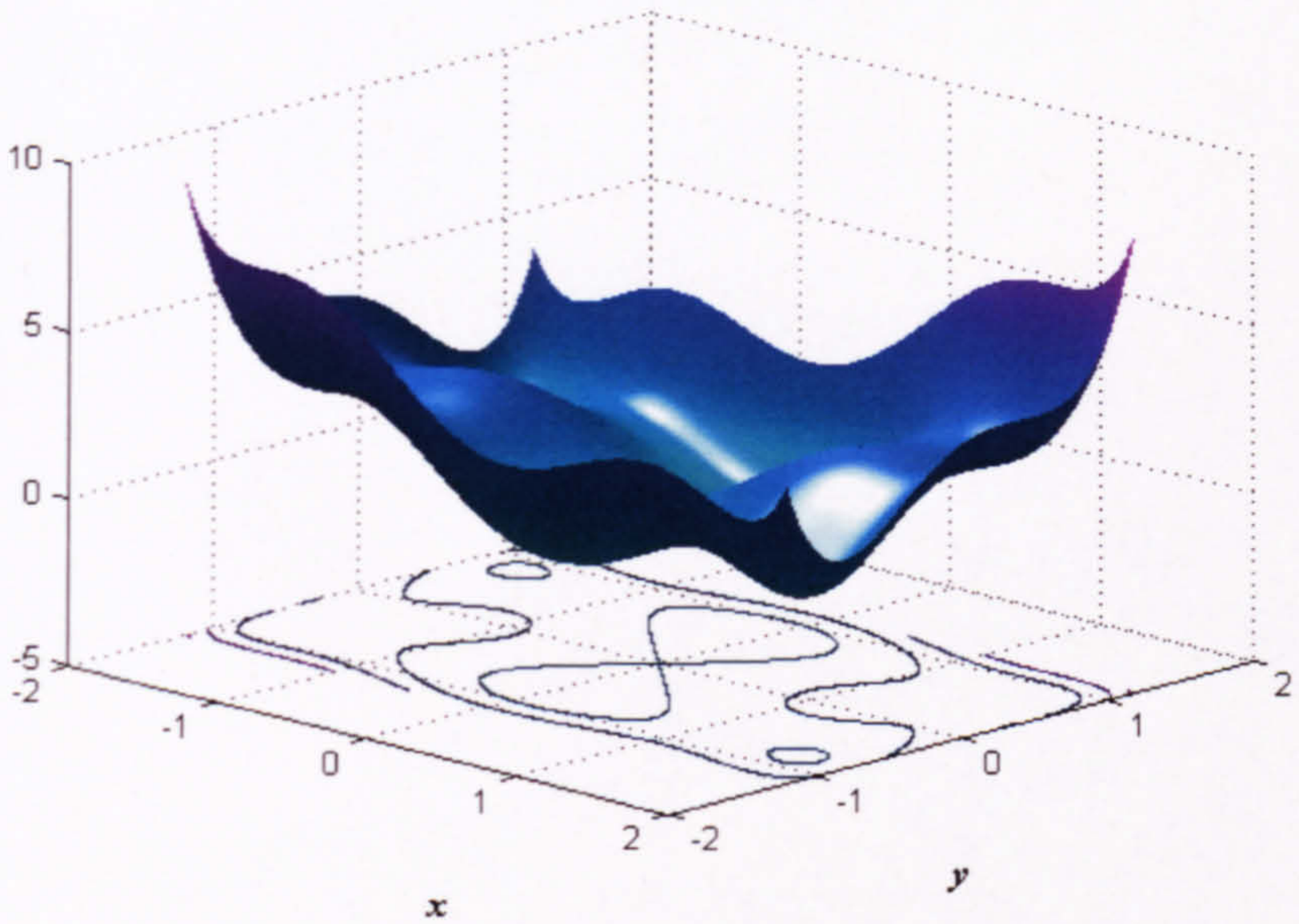


Figure A2. *Six-hump Camelback* function in $[-1.5, 1.5]^2$.

- *Goldstein and Price* (Fig. A3), dimensionality $n = 2$.

$f(x,y) = p(x,y)q(x,y)$, where
$p(x,y) = 1 + (x + y + 1)^2(19 - 14x + 3x^2 - 14y + 6xy + 3y^2)$ and
$q(x,y) = 30 + (2x - 3y^2)(18 - 32x + 12x^2 + 48y - 36xy + 27y^2)$, $x, y \in [-2, 2]$.

LM=4, GM=1, $f_{min} = 3$ in the point (0, -1).

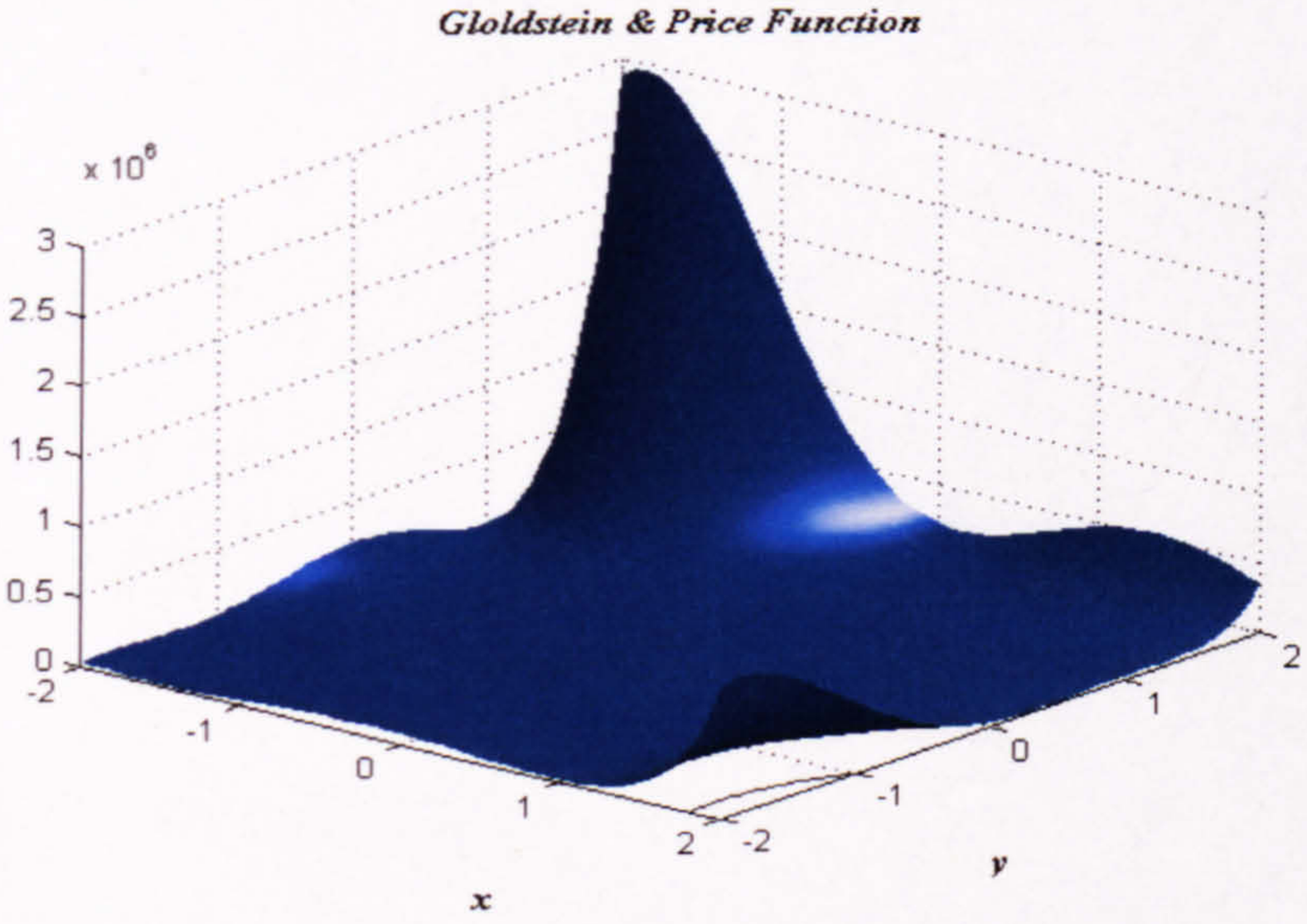


Figure A3. *Goldstein and Price* function in $[-2, 2]^2$.

- *Branin* (Fig. A4), dimensionality: $n = 2$.

$f(x, y) = a(y - bx^2 + cx - d)^2 + e(1 - f)\cos(x) + e$, where $a = 1$, $b = 5.1/(4\pi^2)$, $c = 5/\pi$, $d = 6$, $e = 10$, and $f = 1/(8\pi)$ in $x \in [-5, 10]$ and $y \in [0, 15]$.

LM=3, GM=3, $f_{\min} = 0.39789$ in the points:
$(-\pi, 12.275)$; $(-\pi, 2.275)$; $(9.4348, 2.475)$.

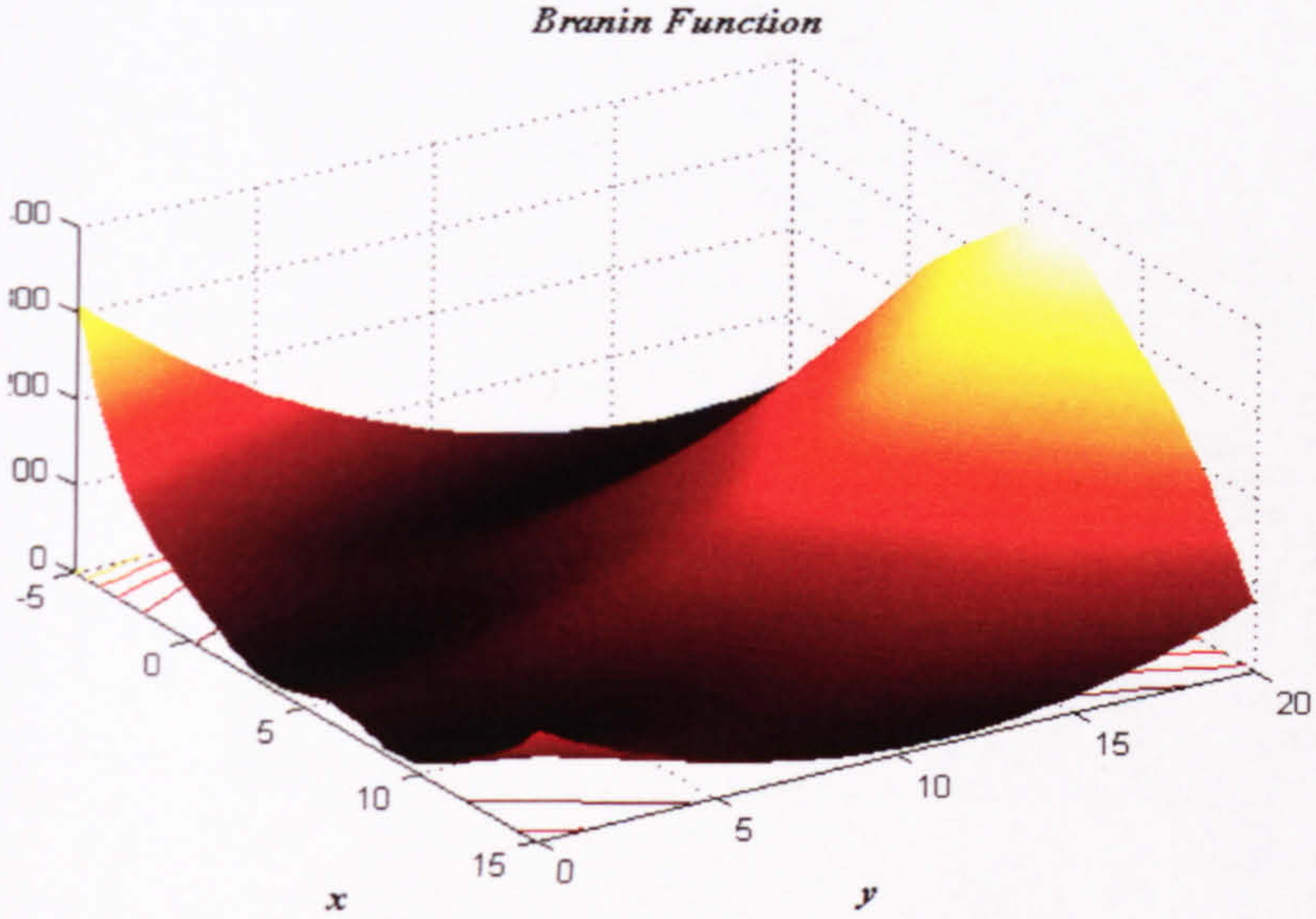


Figure A4. *Branin* function in [-5, 15]x[0, 20].

- *Hartman* (Fig. A5 and Fig. A6), dimensionality: $n = 3, 6$.

$f(x_1, \ldots, x_n) = -\sum_{i=1}^{m} c_i \exp[-\sum_{j=1}^{n} a_{ij}(x_j - p_{ij})^2]$, where a_{ij}, p_{ij} are given in Table A1 for the case $n = 3$ and in Table A2 and Table A3 for the case $n = 6$, $x_j \in [0, 1], j = 1, \ldots, n$.

For $n = 3$, LM=2, GM=1, $f_{\min} = -3.86278$ in the point (0.11477, 0.55556, 0.85254).

For the case $n = 6$, LM=4, GM=1, $f_{\min} = -3.32237$ in the point (0.20169, 0.150011, 0.47687, 0.275332, 0.311652, 0.6573).

Table A1. The *Hartman* function coefficients a, p_{ij} for $n = 3$.

i	$a_{i,1}$	$a_{i,2}$	$a_{i,3}$	c_i	$p_{i,1}$	$p_{i,2}$	$p_{i,3}$
1	3.0	10.0	30.0	1.0	0.3689	0.1170	0.2673
2	0.1	10.0	35.0	1.2	0.4699	0.4387	0.7470
3	3.0	10.0	30.0	3.0	0.1091	0.8732	0.5547
4	0.1	10.0	35.0	3.2	0.03815	0.5743	0.8828

Table A2. The *Hartman* function coefficients a_{ij} for $n = 6$.

$i \setminus j$	1	2	3	4	5	6
1	10.0	3.0	17.0	3.5	1.7	8.0
2	0.05	10.0	17.0	0.1	8.0	14.0
3	3.0	3.5	1.7	10.0	17.0	8.0
4	17.0	8.0	0.05	10.0	0.1	14.0

Table A3. The *Hartman* function coefficients p_{ij} for $n = 6$.

$i \setminus j$	1	2	3	4	5	6
1	0.1312	0.1696	0.5569	0.0124	0.8283	0.5886
2	0.2329	0.4135	0.8307	0.3736	0.1004	0.9991
3	0.2348	0.1451	0.3522	0.2883	0.3047	0.6650
4	0.4047	0.8828	0.8732	0.5743	0.1091	0.0381

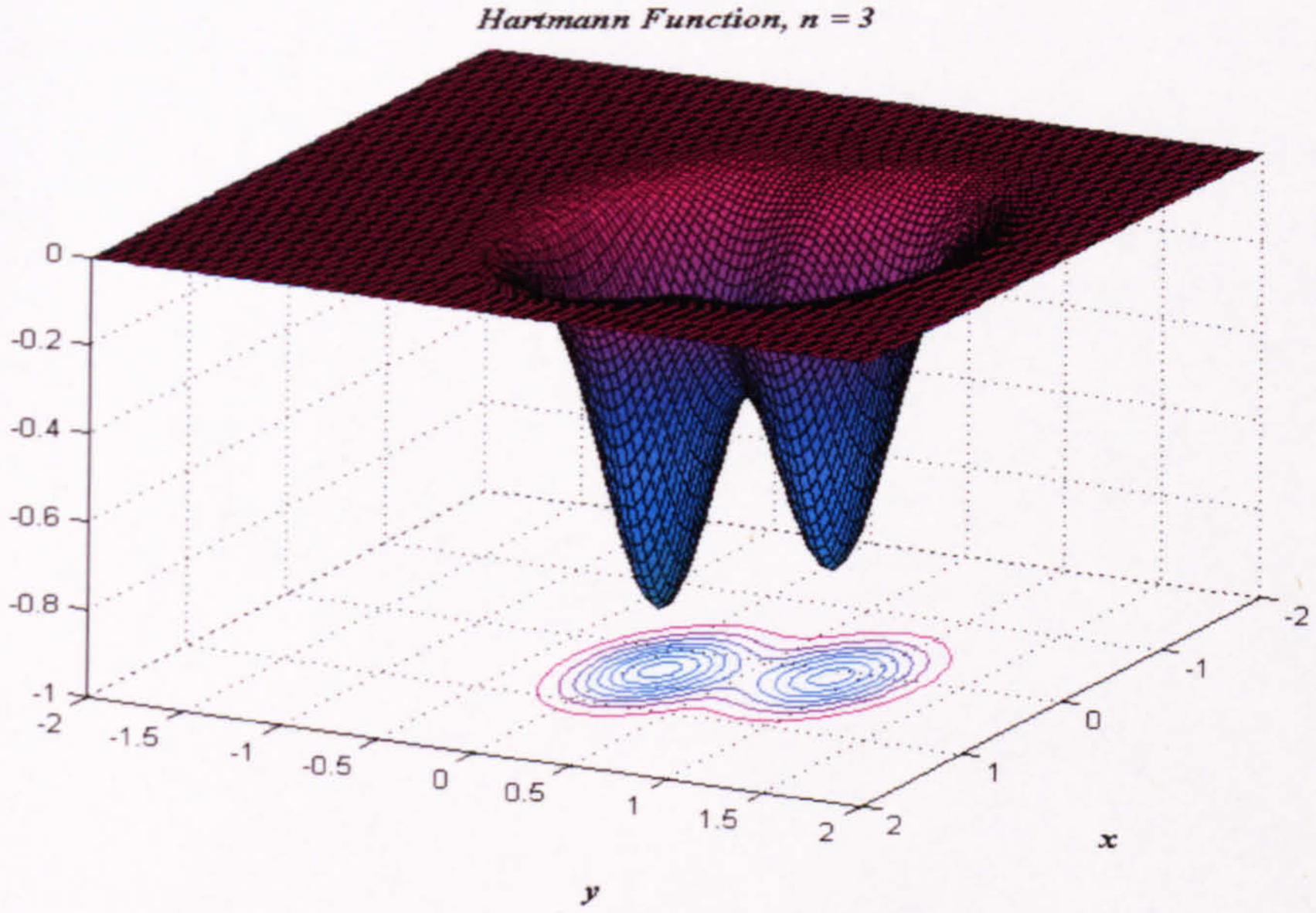


Figure A5. *Hartman* function, $n = 3$, $x_3 = 0.34$.

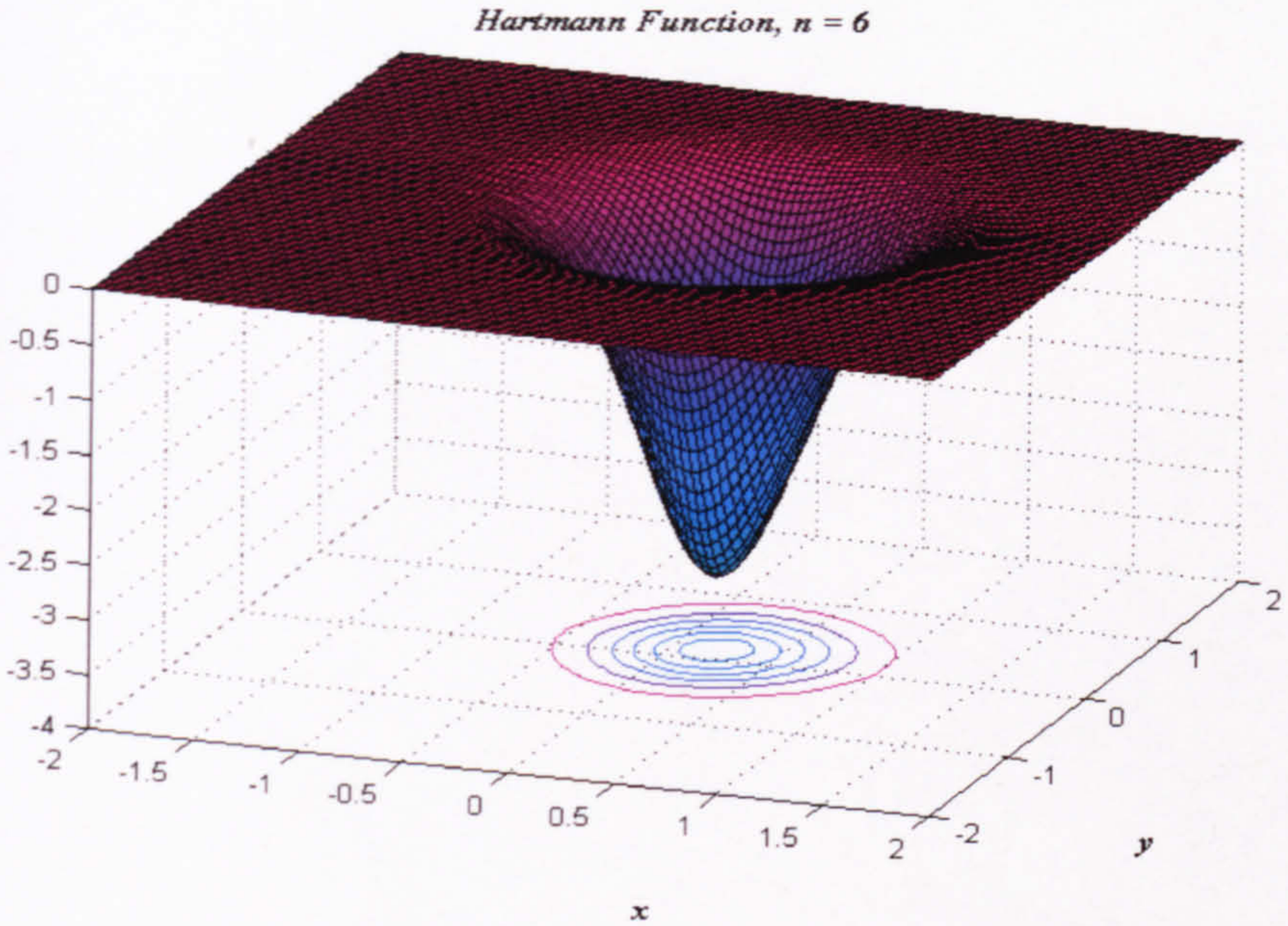


Figure A6. *Hartman* function, $n = 6$, $x_3 = 0.476874$; $x_4 = 0.275332$; $x_5 = 0.311652$; $x_6 = 0.6573$.

- *Easom* (Fig. A7), dimensionality: $n = 2$.

$f(x, y) = -\cos(x)\cos(y)e^{-[(x-\pi)^2+(y-\pi)^2]}$, $x, y \in [-100, 100]$.

LM = 1, GM=1, $f_{\min} = -1$ in the point $(-\pi, \pi)$.

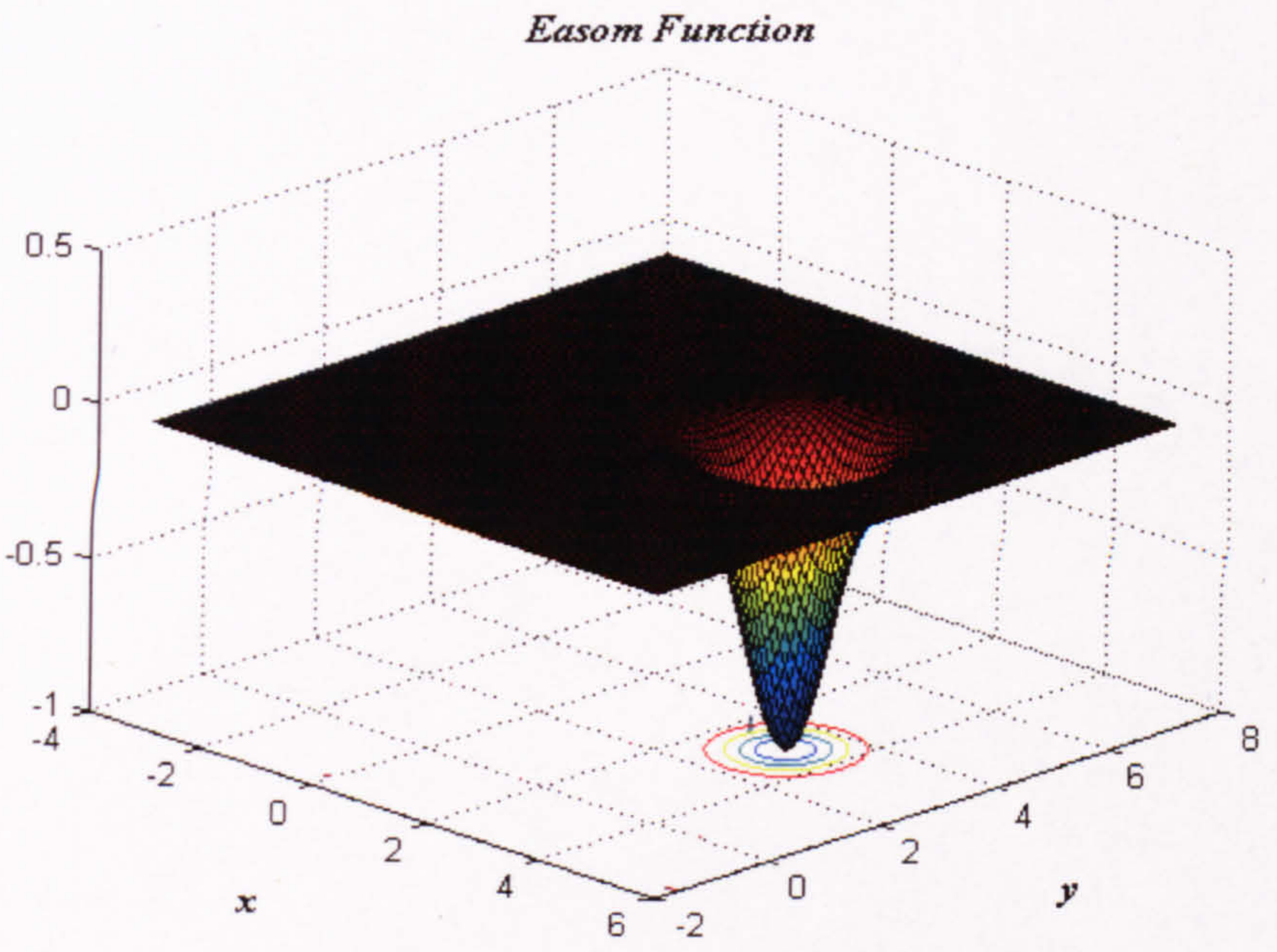


Figure A7. *Easom* function in [-3, 6]x[-2, 7].

- *Zakharov* (Fig. A8), dimensionality n = 2, 5, 10.

$$f(x_1, \ldots, x_n) = \sum_{j=1}^{n} x_j^2 + (\sum_{j=1}^{n} 0.5 j x_j)^2 + (\sum_{j=1}^{n} 0.5 j x_j)^4, \; x_j \in [-5, 10], j = 1, \ldots, n.$$

LM = 1, GM=1, $f_{\min} = 0$ in the point (0, …, 0).

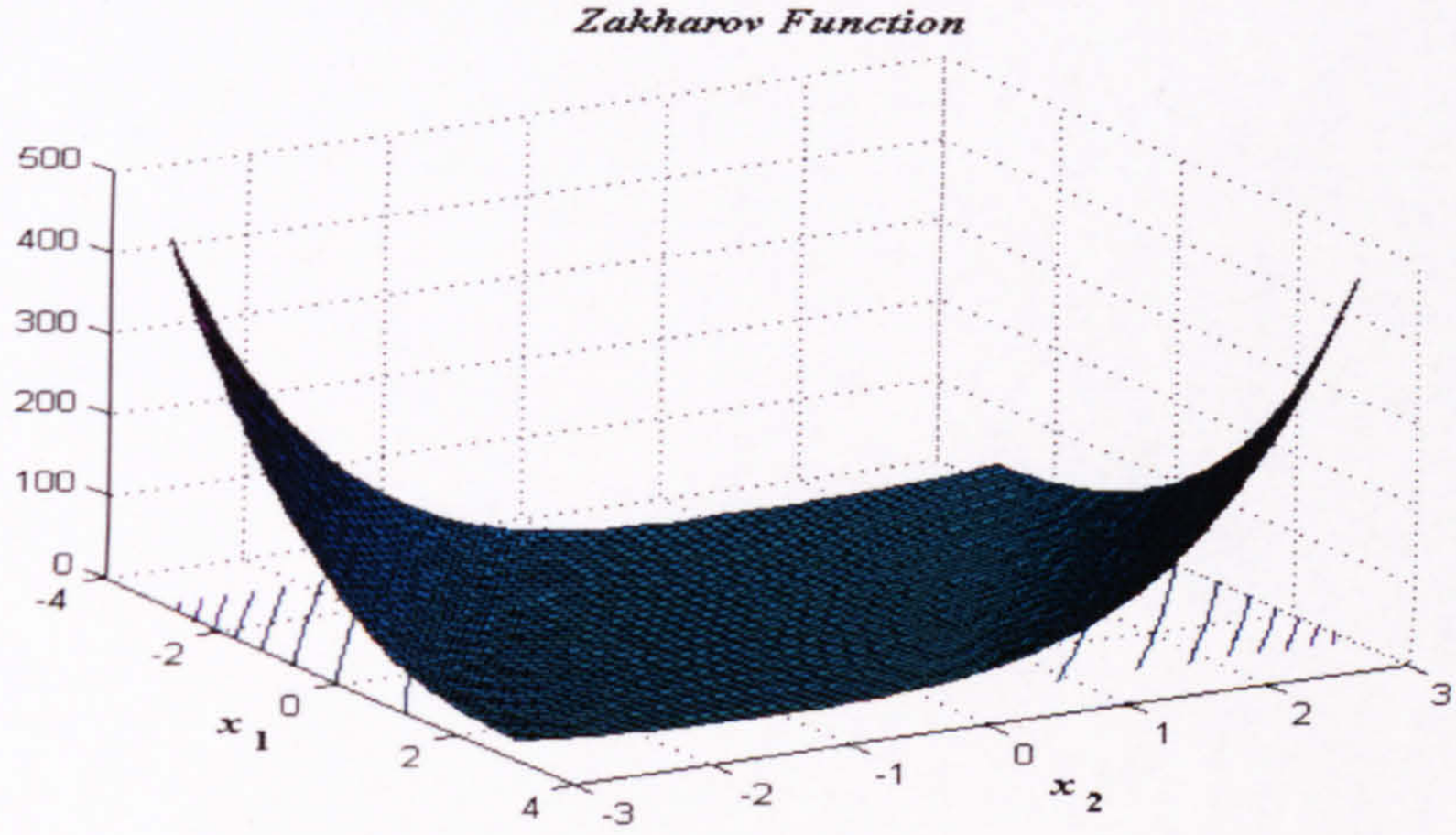


Figure A8. *Zakharov* function in $[-3, 3]^2$.

- *Shekel* ^{m} (Fig. A9-A11), dimensionality: n = 4, 10.

$$f(x_1, \ldots, x_n) = -\sum_{i=1}^{m} [\sum_{j=1}^{n} (x_j - a_{ij})^2 + c_i]^{-1}, \; x_j \in [0, 10], j = 1, \ldots, n.$$

Three 4-dimensional cases are considered when m = 5, 7, 10 and one 10-dimensional case when m = 30. The coefficients a_{ij} and c_j are given in Table A4 for the cases of m = 5, 7, and 10, and in Table A6 – for the case when m = 30.

Numerous LM and GM=1. The corresponding global minima are given in Table A5.

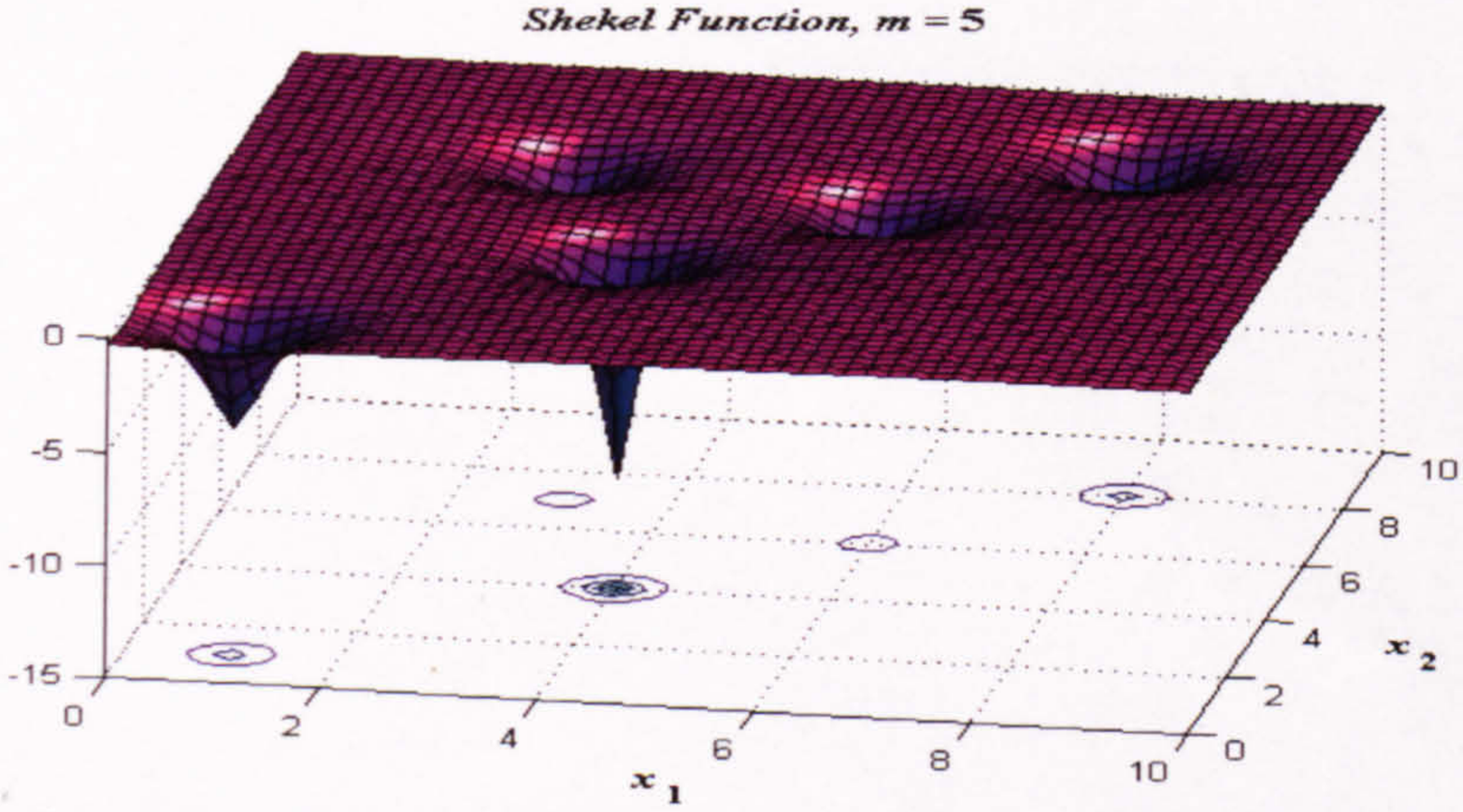


Figure A9. *Shekel* function, m = 5 in $[0, 10]^4$ ($x_3 = x_1, x_4 = x_2$).

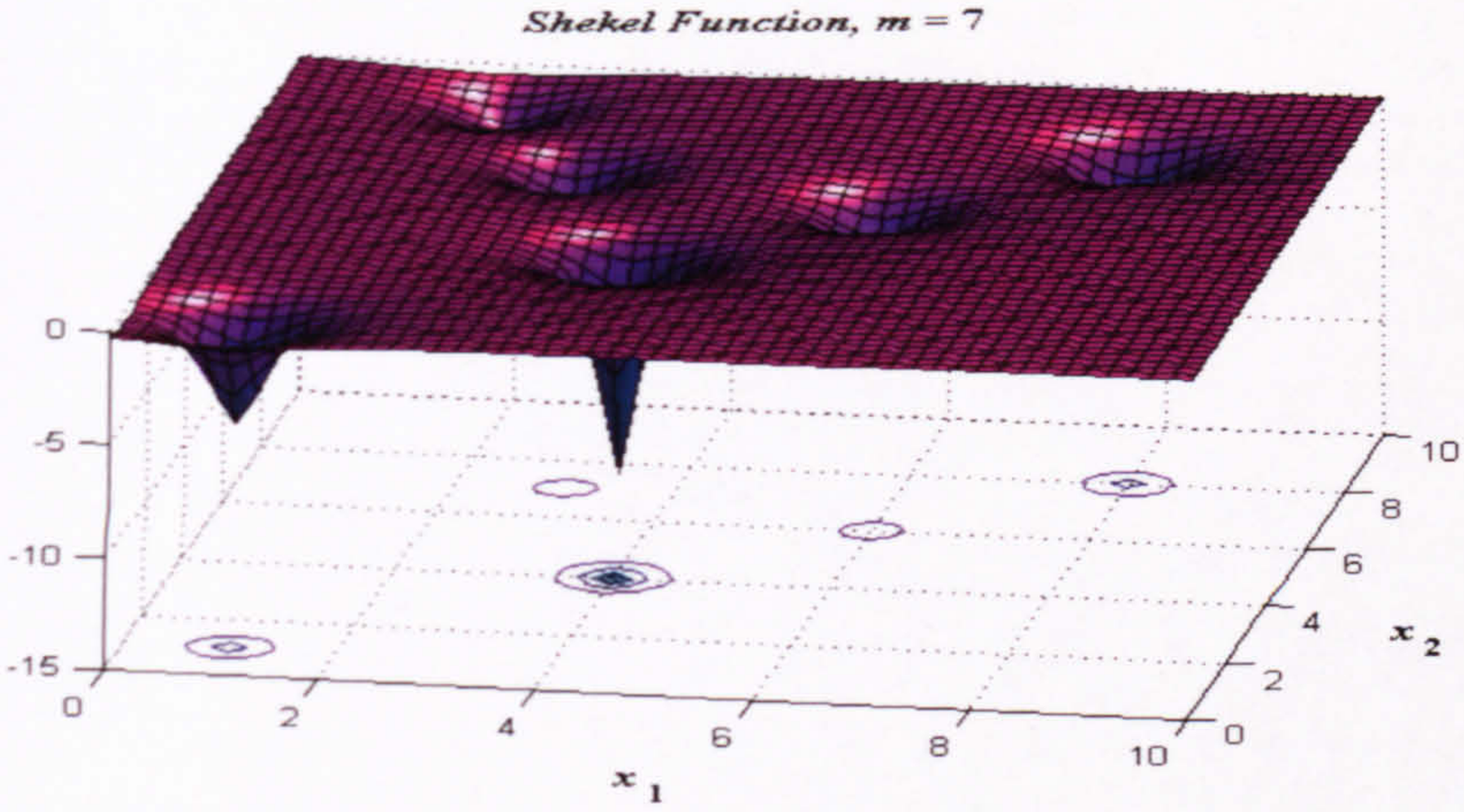


Figure A10. *Shekel* function, $m = 7$ in $[0, 10]^2$ ($x_3 = x_1, x_4 = x_2$).

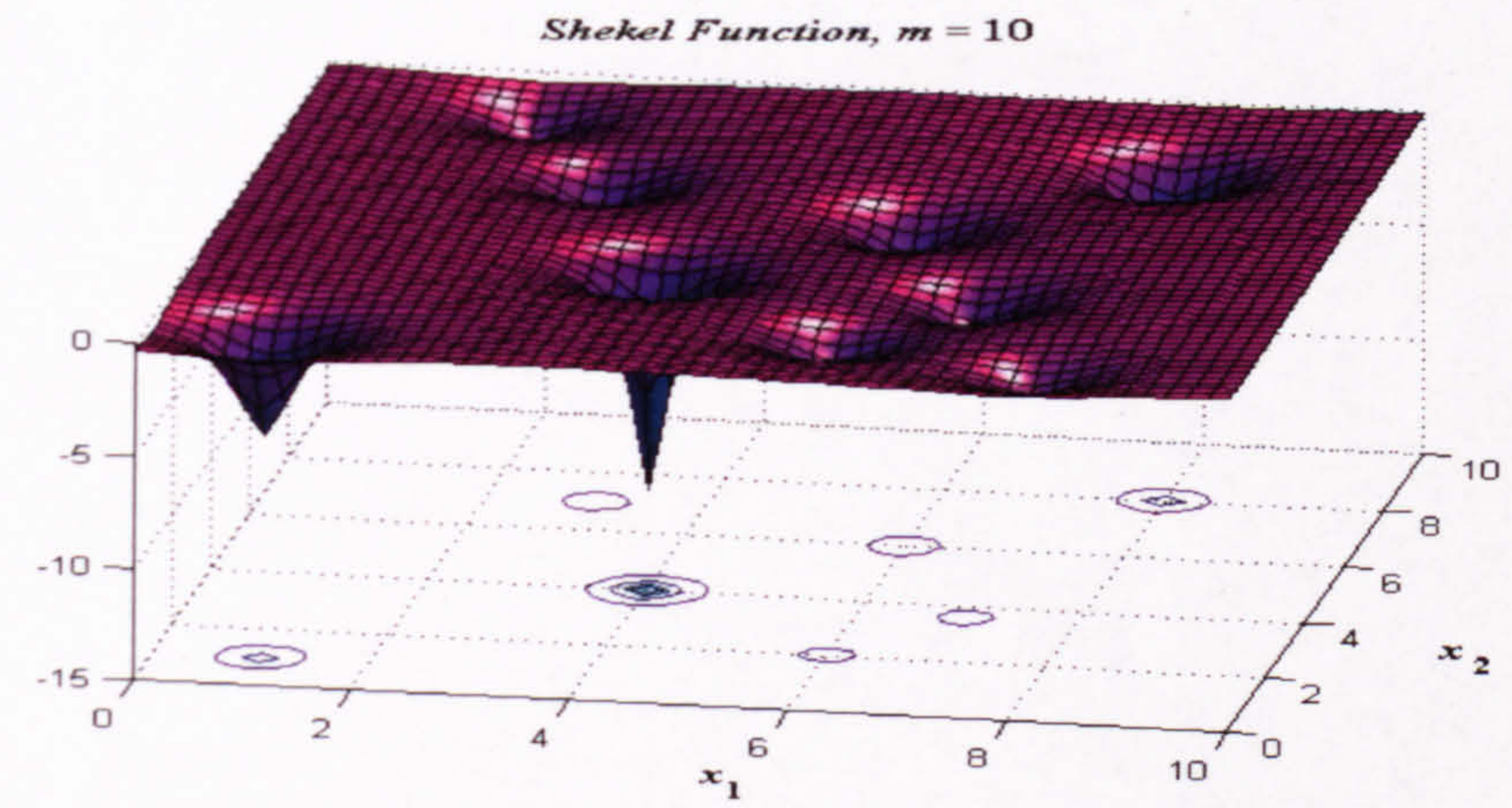


Figure A11. *Shekel* function, $m = 10$ in $[0, 10]^2$ ($x_3 = x_1, x_4 = x_2$).

Table A4. The *Shekel* functions coefficients a_{ij} and c_i for $m = 5$, 7, 10.

i	$a_{i,1}$	$a_{i,2}$	$a_{i,3}$	$a_{i,4}$	c_i
1	4.0	4.0	4.0	4.0	0.1
2	1.0	1.0	1.0	1.0	0.2
3	8.0	8.0	8.0	8.0	0.2
4	6.0	6.0	6.0	6.0	0.4
5	3.0	7.0	3.0	7.0	0.4
6	2.0	9.0	2.0	9.0	0.6
7	5.0	5.0	3.0	3.0	0.3
8	8.0	1.0	8.0	1.0	0.7
9	6.0	2.0	6.0	2.0	0.5
10	7.0	3.6	7.0	3.6	0.5

Table A5. The *Shekel* GM $m = 5$, 7, 10.

m	x	$f_{\min}$
5	(4, 4, 4, 4)	-10.1532
7	(4, 4, 4, 4)	-10.40294
10	(4, 4, 4, 4)	-10.53387
30	The vector $a_{3,j}$ (j=1, …, 10) from Table A6.	-10.2088

Table A6. The *Shekel* functions coefficients a_{ij} and c_i for $m = 30$ and $n = 10$.

i	$a_{i,1}$	$a_{i,2}$	$a_{i,3}$	$a_{i,4}$	$a_{i,5}$	$a_{i,6}$	$a_{i,7}$	$a_{i,8}$	$a_{i,9}$	$a_{i,10}$	c_i
1	9.681	0.667	4.783	9.095	3.517	9.325	6.544	0.211	5.122	2.020	0.806
2	9.400	2.041	3.788	7.931	2.882	2.672	3.568	1.284	7.033	7.374	0.517
3	8.025	9.152	5.114	7.621	4.564	4.711	2.996	6.126	0.734	4.982	0.100
4	2.196	0.415	5.649	6.979	9.510	9.166	6.304	6.054	9.377	1.426	0.908
5	8.074	8.777	3.467	1.863	6.708	6.349	4.534	0.276	7.633	1.567	0.965
6	8.074	8.777	3.467	1.863	6.708	6.349	4.534	0.276	7.633	1.567	0.669
7	7.650	5.658	0.720	2.764	3.278	5.283	7.474	6.274	1.409	8.208	0.524
8	1.256	3.605	8.623	6.905	4.584	8.133	6.071	6.888	4.187	5.448	0.902
9	8.314	2.261	4.224	1.781	4.124	0.932	8.129	8.658	1.208	5.762	0.531
10	0.226	8.858	1.420	0.945	1.622	4.698	6.228	9.096	0.942	7.637	0.876
11	7.305	2.228	1.242	5.928	9.133	1.826	4.060	5.204	8.713	8.247	0.462
12	0.652	7.027	0.508	4.876	8.807	4.632	5.808	6.937	3.291	7.016	0.491
13	2.699	3.516	5.874	4.119	4.461	7.496	8.817	0.690	6.593	9.789	0.463
14	8.327	3.897	2.017	9.570	9.825	1.150	1.395	3.885	6.354	0.109	0.714
15	2.132	7.006	7.136	2.641	1.882	5.943	7.273	7.691	2.880	0.564	0.352
16	4.707	5.579	4.080	0.581	8.698	8.542	8.077	8.515	9.231	4.670	0.869
17	8.304	7.559	8.567	0.322	7.128	8.392	1.472	8.524	2.277	7.826	0.813
18	8.632	4.409	4.832	5.768	7.050	6.715	1.711	4.323	4.405	4.591	0.811
19	4.887	9.112	0.170	8.967	9.693	9.867	7.508	7.770	8.382	6.740	0.828
20	2.440	6.686	4.299	1.007	7.008	1.427	9.398	8.480	9.950	1.675	0.964
21	6.306	8.583	6.084	1.138	4.350	3.134	7.853	6.061	7.457	2.258	0.789
22	0.652	2.343	1.370	0.821	1.310	1.063	0.689	8.819	8.833	9.070	0.360
23	5.558	1.272	5.756	9.857	2.279	2.764	1.284	1.677	1.244	1.234	0.369
24	3.352	7.549	9.817	9.437	8.687	4.167	2.570	6.540	0.228	0.027	0.992
25	8.798	0.880	2.370	0.168	1.701	3.680	1.231	2.390	2.499	0.064	0.332
26	1.460	8.057	1.336	7.217	7.914	3.615	9.981	9.198	5.292	1.224	0.817
27	0.432	8.645	8.774	0.249	8.081	7.461	4.416	0.652	4.002	4.644	0.632
28	0.679	2.800	5.523	3.049	2.968	7.225	6.730	4.199	9.614	9.229	0.883
29	4.263	1.074	7.286	5.599	8.291	5.200	9.214	8.272	4.398	4.506	0.608
30	9.496	4.830	3.150	8.270	5.079	1.231	5.731	9.494	1.883	9.732	0.326

- *HE – Hyper-Ellipsoid* (Fig. A12), dimensionality: $n = 10$.

$$f(x_1, \ldots, x_n) = \sum_{j=0}^{n-1} 2^j x_j^2 \,, (x_1, \ldots, x_n) \in [-100, 100]^n.$$

LM = 1, GM = 1, $f_{\min} = 0$ in the point $(0, \ldots, 0)$.

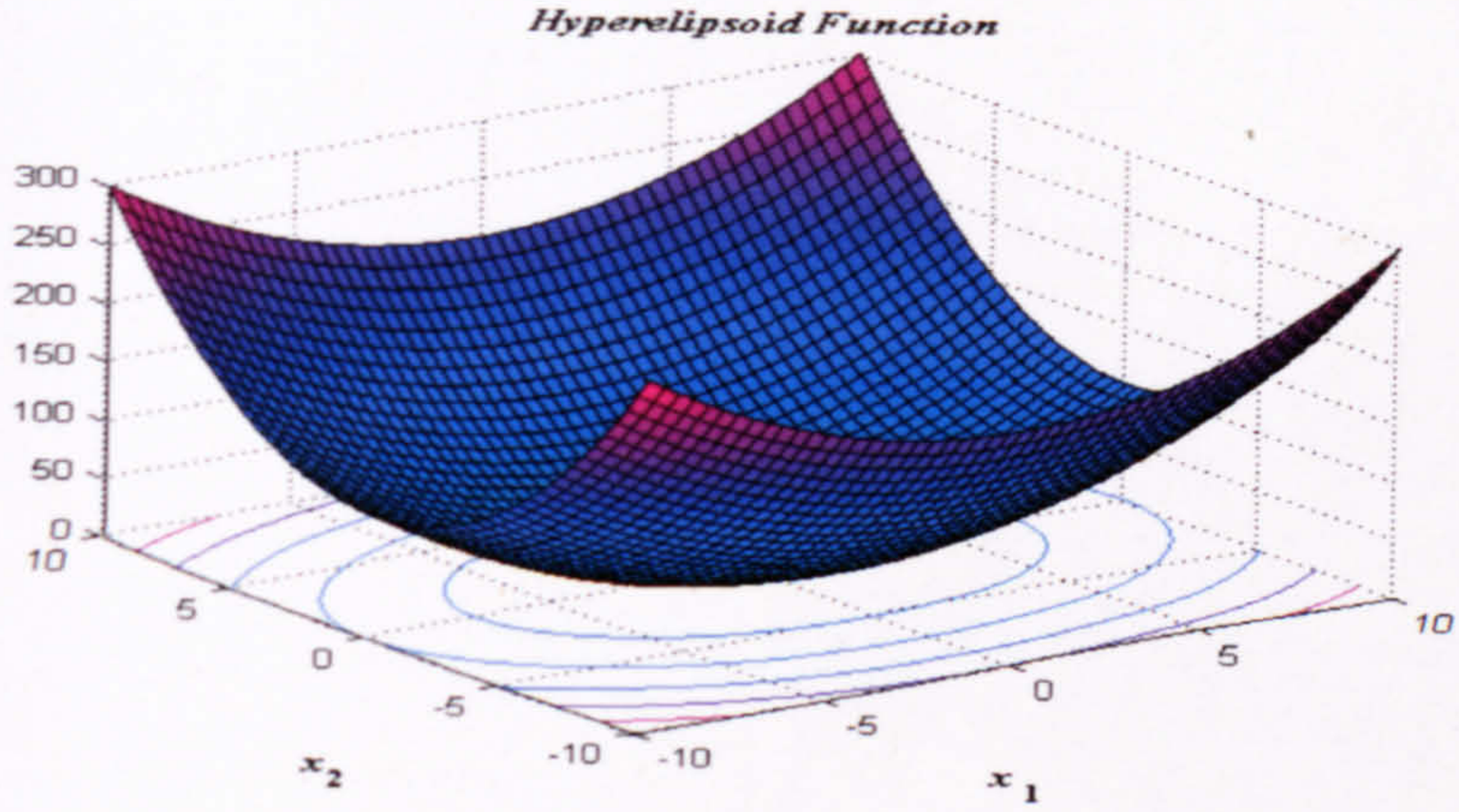


Figure A12. *Hyper Ellipsoid* function in $[-10, 10]^2$.

- *Langerman* (*modified*), (Fig. A13), dimensionality: $n = 10$.

$$f(x_1, \ldots, x_n) = \sum_{k=0}^{m-1} c_k \left(\exp\left(\frac{-\| x - A_k \|^2}{\pi} \right) \cos\left(\pi \| x - A_k \|^2 \right) \right),\ m = 30,\ \|x - \mathrm{A}_k\|^2 = \sum_{j=0}^{n-1} (x_j - a_{k,j})^2,\ k = 0, \ldots, m-1,\ c = (c_k),\ A = (a_{k,j}),\ (x_1, \ldots, x_n) \in [0, 10]^n,$$

where the coefficients in c and A are the same as the one of the Shekel function for $m = 30$, given in Table A6.
Multiple LM, GM = 1, $f_{\min} = -0.965$ in the point
(8.074, 8.777, 3.467, 1.867, 6.708, 6.349, 4.534, 0.276, 7.633, 1.567).

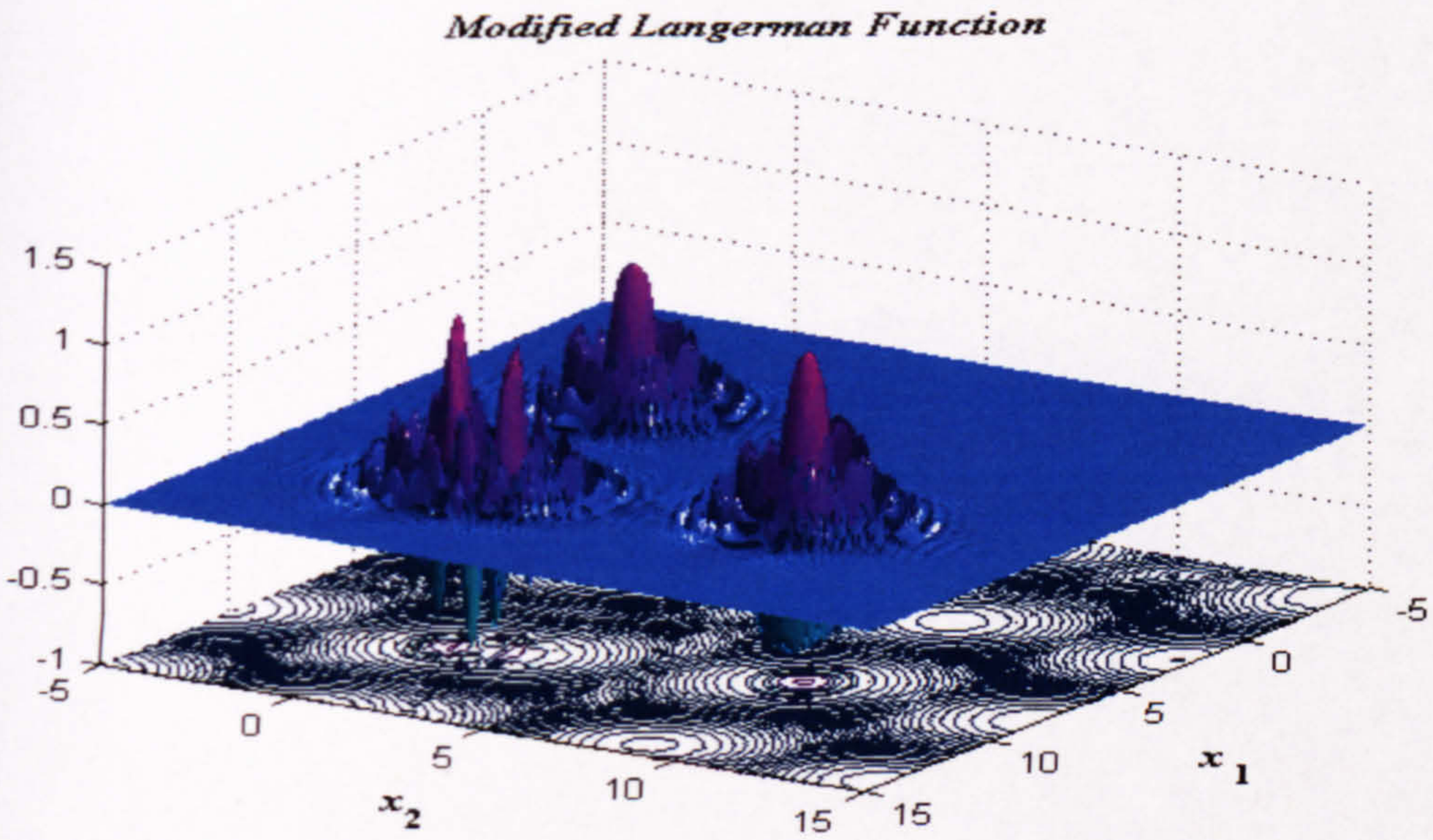


Figure A13. *Modified Langerman* function in $[-5, 15]^2$.

- *Whitley* (Fig. A14-A15), dimensionality: $n = 10$.

$$f(x_1, \ldots, x_n) = \sum_{k=0}^{n-1}\sum_{j=0}^{n-1}\left(y_{j,k}^2 / 4000 - \cos(y_{j,k}) + 1\right),\ y_{j,k} = 100(x_k - x_j^2)^2 + (1 - x_j)^2, \quad (x_1, \ldots, x_n) \in [-100, 100]^n,$$

Multiple LM, GM =1, $f_{\min} = 0$ in the point (1, 1, …, 1).

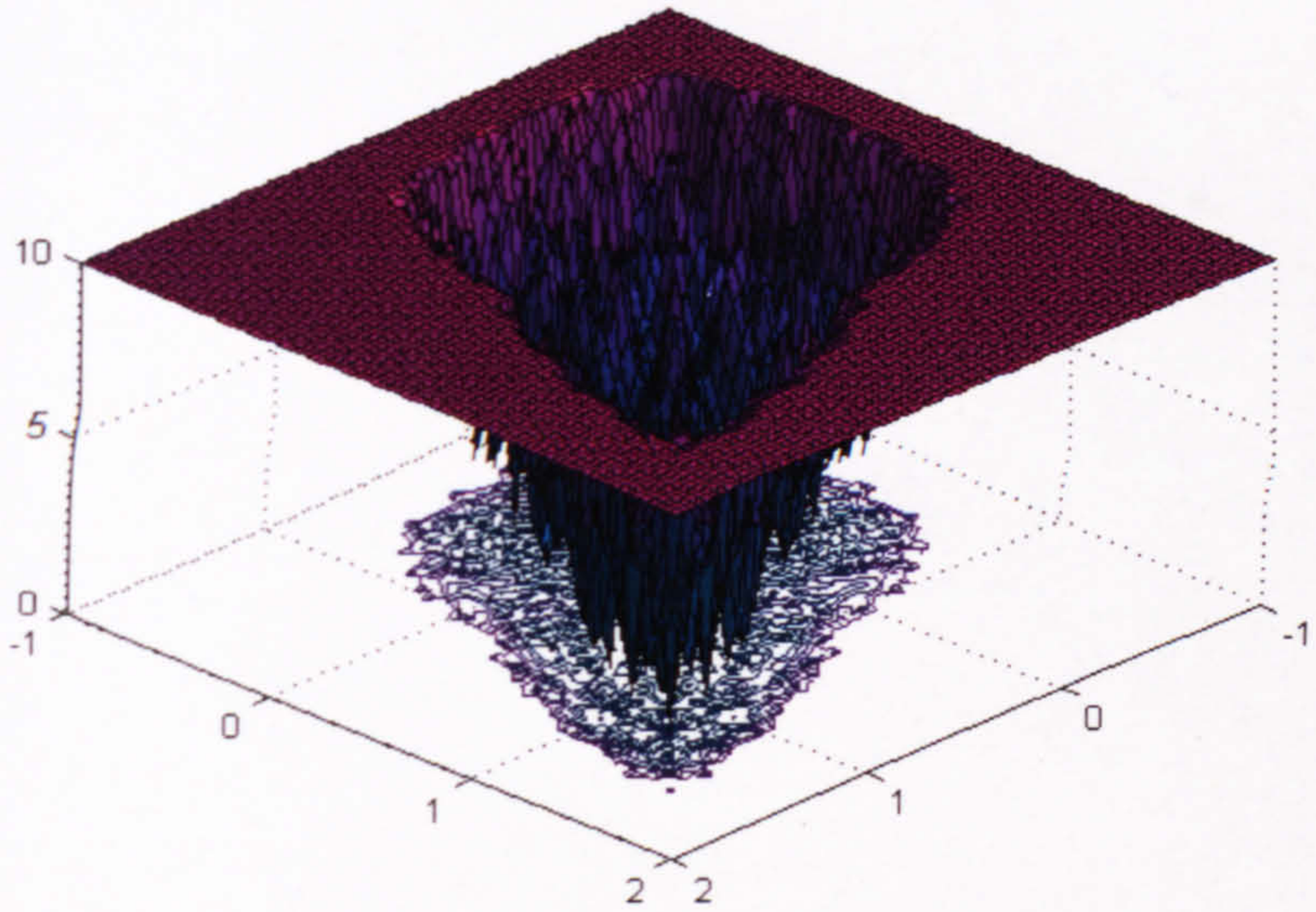


Figure A14. *Whitley* function in $[-1, 2]^2$. For visual purposes, all function values higher than 10 are clipped.

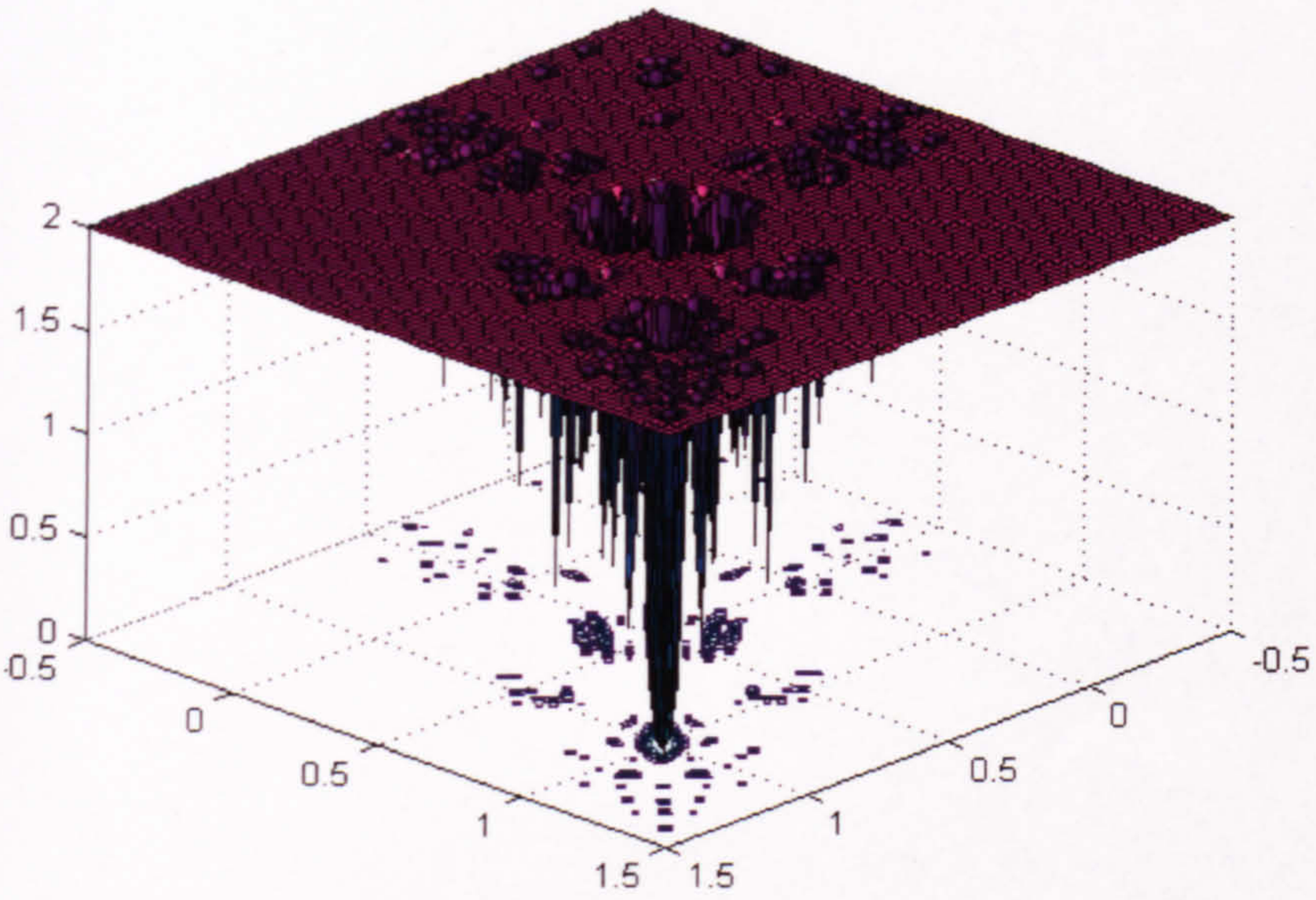


Figure A15. *Whitley* function in $[-0.5, 1.5]^2$. For visual purposes, all function values higher than 2 are clipped.

- *LJ – Lenard and Jones*, dimensionality: $n = 15$.

$$f(x_1, \ldots, x_n) = \sum_{i=0}^{p-2} \sum_{j=i+1}^{p-1} \left(\frac{1}{d_{i,j}^2} - \frac{2}{d_{i,j}} \right), \; d_{i,j} = \left(\sum_{k=0}^{2} \left(x_{3i+k} - x_{3j+k} \right)^2 \right)^3, \text{ for } n = 3p,$$

$(x_1, \ldots, x_n) \in [-2, 2]^n$.

Multiple LM and Multiple GM, $f_{\min} = -9.103852$.

B. FUNCTIONS USED WITH BOTH LOW AND HIGH DIMENSIONS

- *Rastrigin* (Fig. B1), dimensionality: $n = 2, 10, 30, 100, 150$.

$$f(x_1, \ldots, x_n) = \sum_{i=1}^{n} [x_i^2 - 10\cos(2\pi x_i) + 10], \; (x_1, \ldots, x_n) \in [-5.12, 5.12]^n.$$

Multiple LM and GM = 1 in the point $(0, \ldots, 0)$.

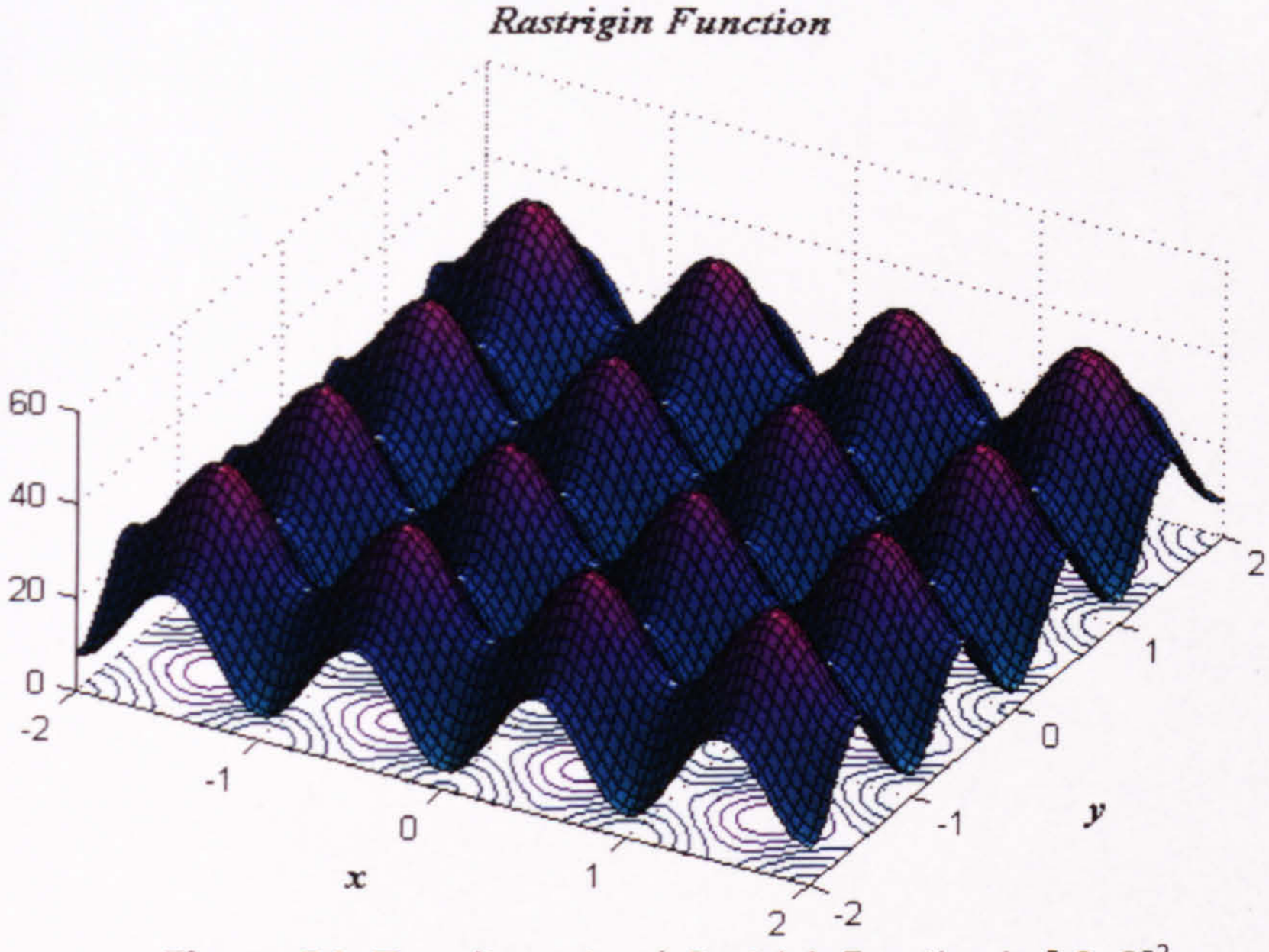


Figure B1. Two dimensional *Rastrigin* Function in $[-2, 2]^2$.

- *NDTF – N-Dimensional Test* (Fig. B2), dimensionality: $n = 2, 100, 150$.

$$f(x_1, \ldots, x_n) = \frac{1}{n} \sum_{i=1}^{n} (x_i^4 - 16x_i^2 + 5x_i), \; (x_1, \ldots, x_n) \in [-5, 5]^n.$$

Multiple LM and GM=1, $f_{\min} = -78.33236$ in the point $(-2.9, \ldots, -2.9)$.

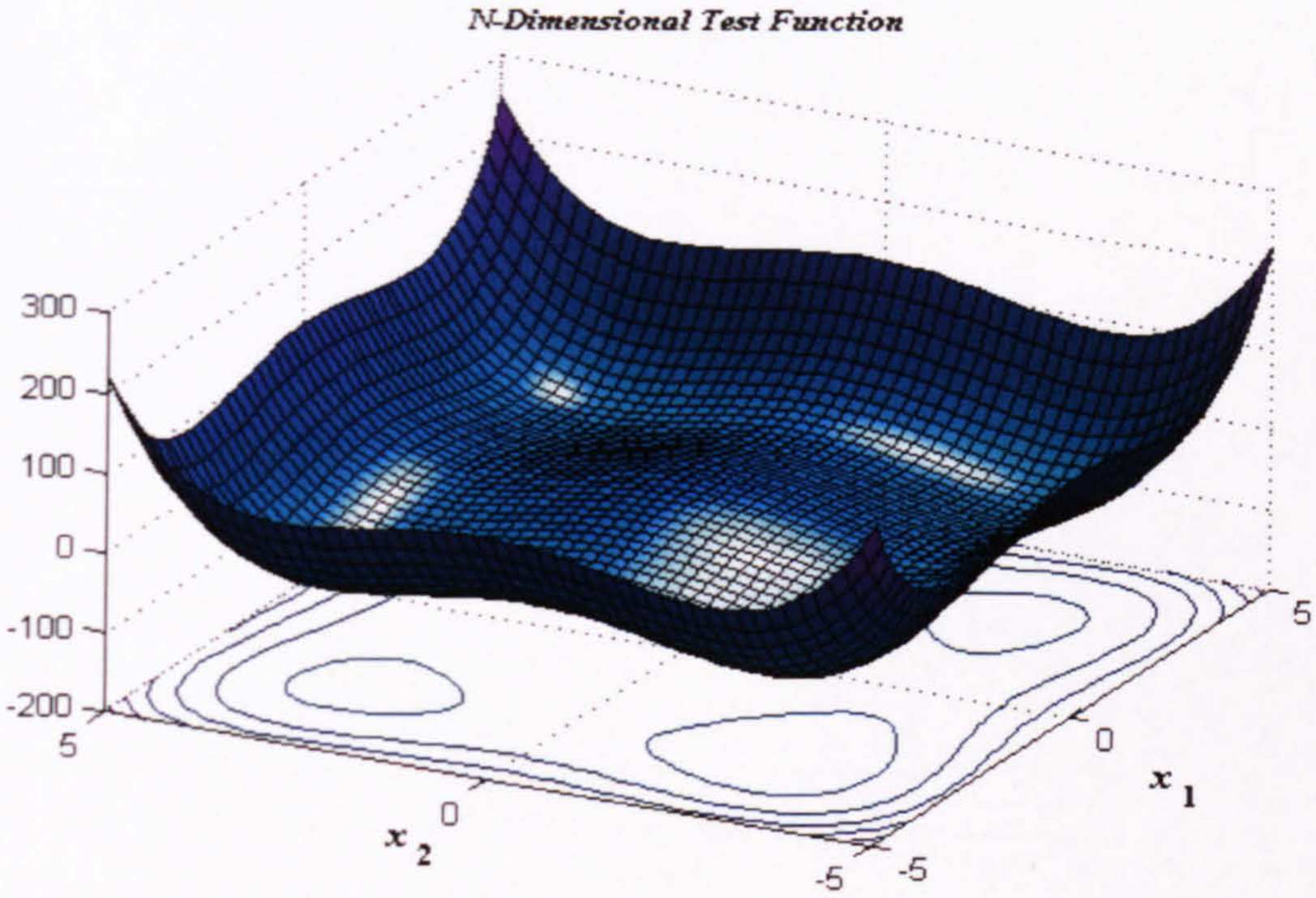


Figure B2. *N-Dimensional Test* Function for $N = 2$ in $[-5, 5]^2$.

- *Rosenbrock* (Fig. B3), dimensionality: $n = 2, 5, 10, 30$.

$$f(x_1, \ldots, x_n) = \sum_{j=1}^{n-1} [100(x_j^2 - x_{j+1})^2 + (x_j - 1)^2], \ (x_1, \ldots, x_n) \in [-5, 10]^n.$$

LM = 1, GM=1, $f_{\min} = 0$ in the point (1, …, 1).

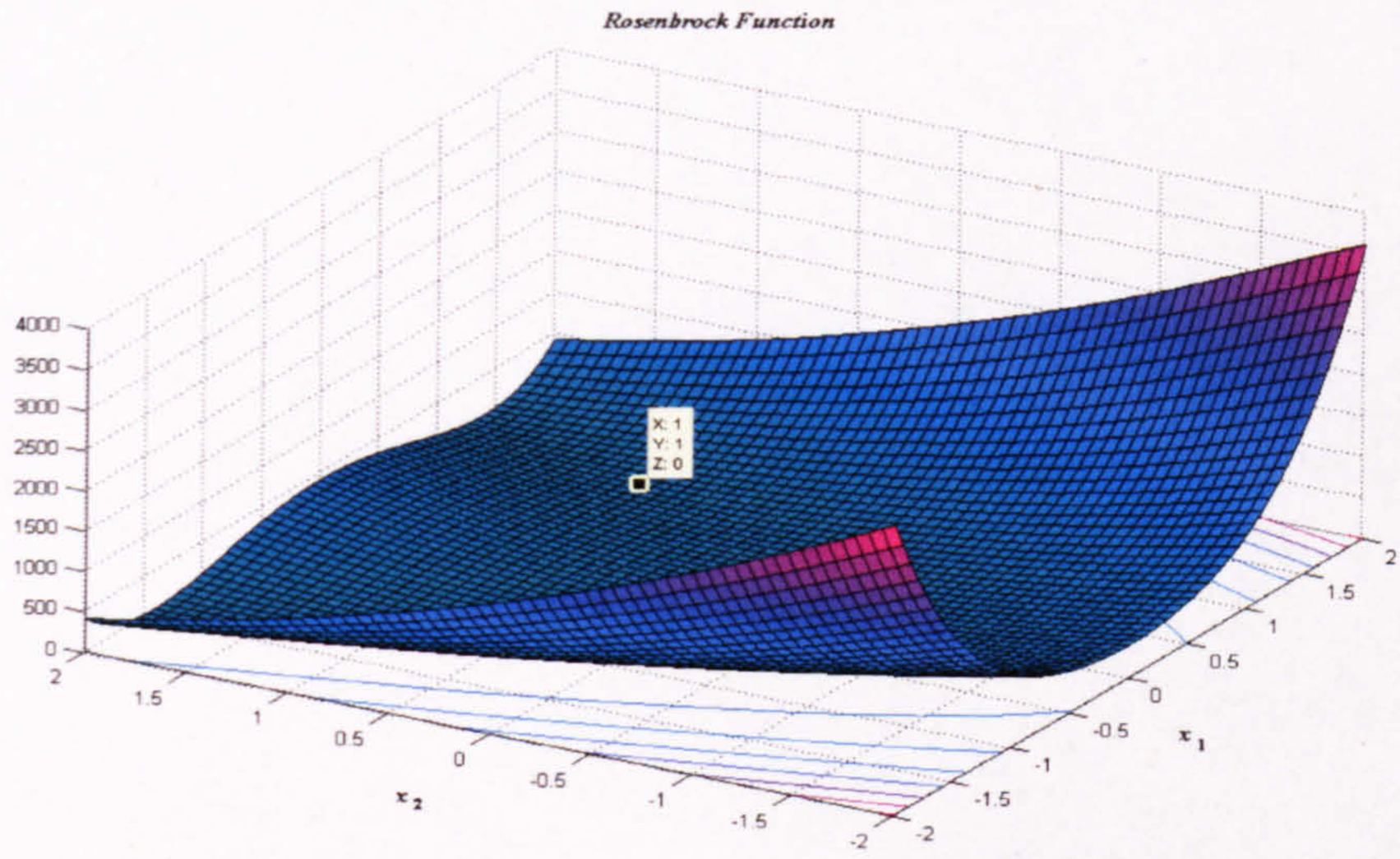


Figure B3. Two dimensional *Rosenbrock* Function in $[-2, 2]^2$.

- *Sphere* (Fig. B4), dimensionality $n = 3,\ 30, 100, 150$.

$$f(x_1, \ldots, x_n) = \sum_{i=1}^{n} x_i^2,\ (x_1, \ldots, x_n) \in [-120, 80]^n.$$

LM = GM = 1, $f_{\min} = 0$ in the point (0, …, 0).

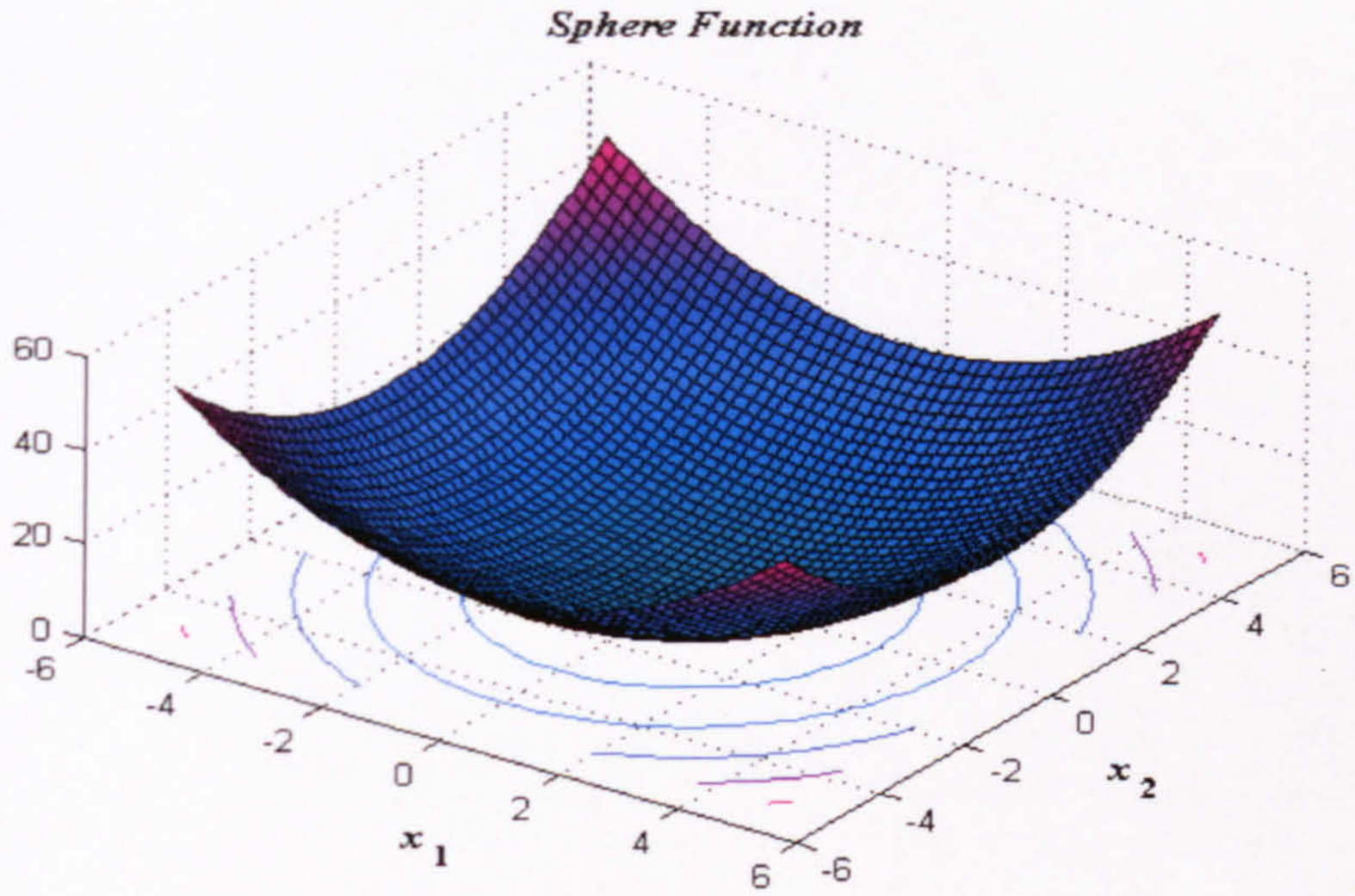


Figure B4. Two dimensional *Sphere* Function in $[-6, 6]^2$.

- *Ackley* (Fig. B5), dimensionality n = 10, 30, 100, 150.

$$f(x_1, \ldots, x_n) = -20\exp\left\{-0.2\sqrt{\frac{1}{n}\sum_{i=1}^{n} x_i^2}\right\} - \exp\left\{\frac{1}{n}\sum_{i=1}^{n}\cos(2\pi x_i)\right\} + 20 + e, \qquad (x_1, \ldots, x_n) \in [-22, 42]^n.$$

Multiple LM, GM = 1, $f_{\min} = 0$ in the point (0, …, 0).

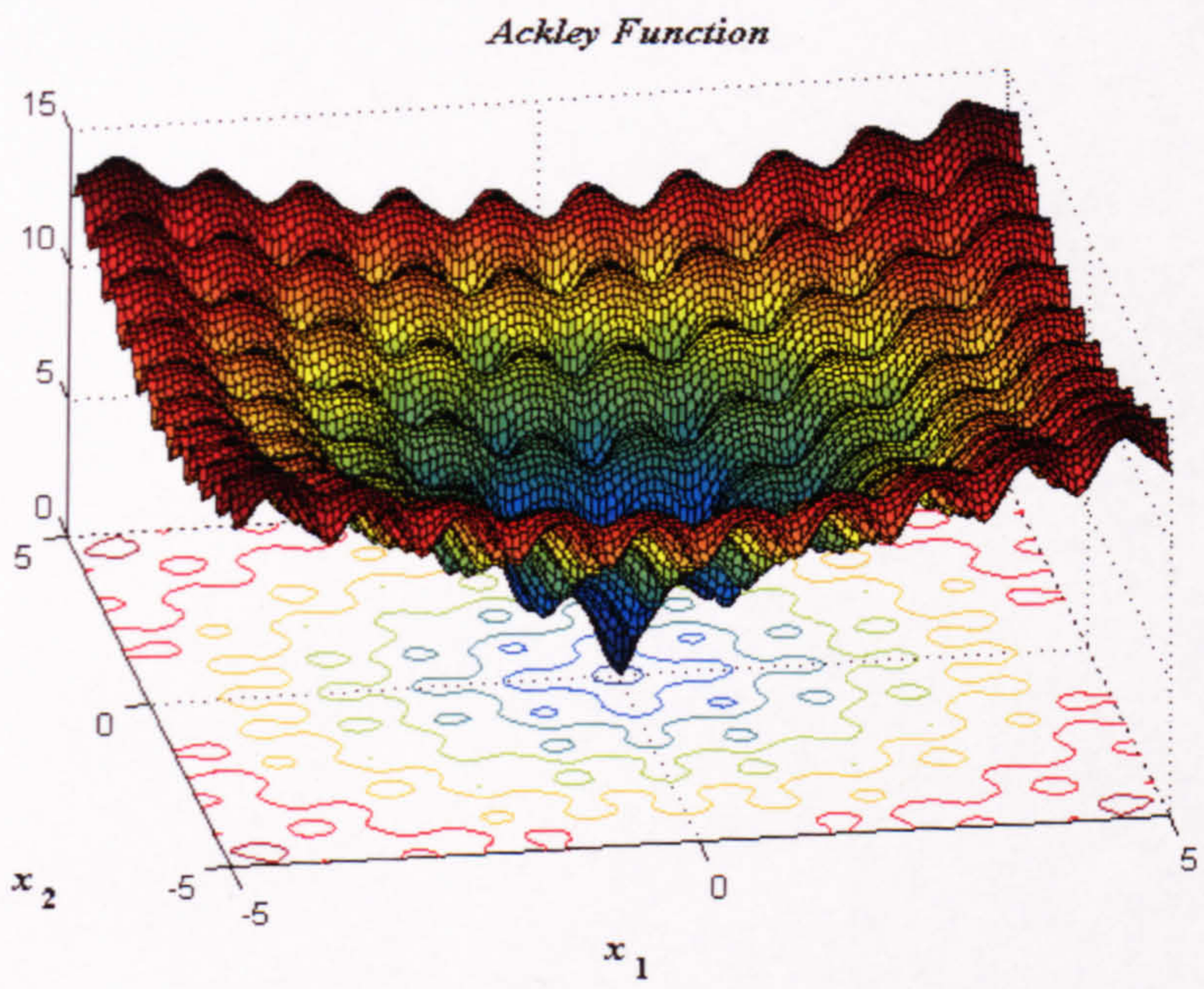


Figure B5. Two dimensional *Ackley* Function in $[-5, 5]^2$.

- *Griewank* (Fig. B6), dimensionality n = 10, 30, 100, 150.

$$f(x_1, \ldots, x_n) = \frac{1}{4000}\sum_{i=1}^{n} x_i^2 - \prod_{i=1}^{n}\cos\left(\frac{x_i}{\sqrt{i}}\right)+1, \quad (x_1, \ldots, x_n) \in [-500, 700]^n.$$

Multiple LM, GM = 1, $f_{\min} = 0$ in the point (0, …, 0).

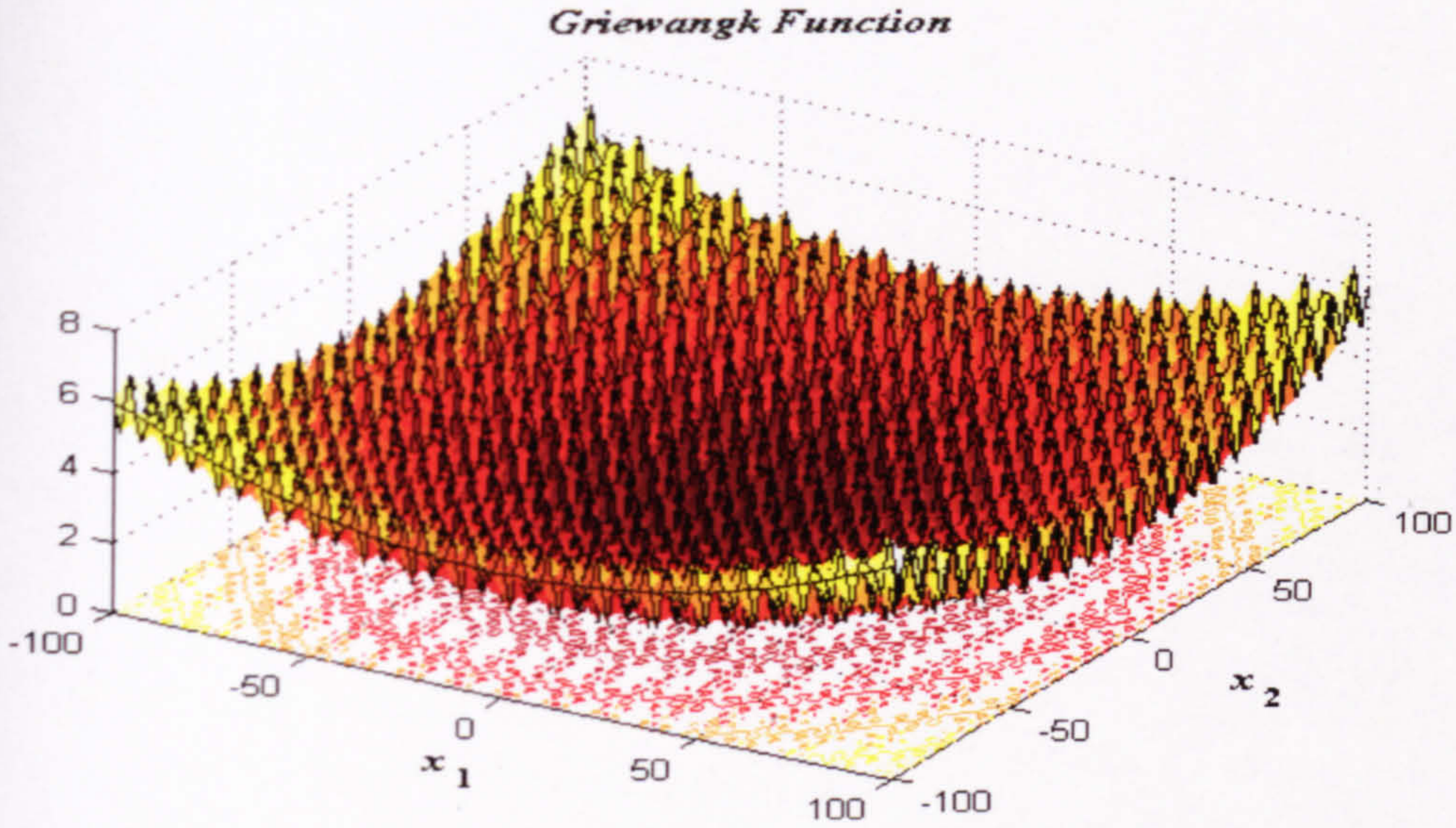


Figure B6. Two dimensional *Griewank* Function in $[-100, 100]^2$.

C. TYPICAL HIGHER DIMENSIONAL FUNCTIONS (20–150 DIMENSIONS)

- *Levy* (Fig. C1), dimensionality n = 20.

$$f(x_1, \ldots, x_n) = \frac{\pi}{n}\left\{10\sin^2(\pi y_1) + \sum_{i=1}^{n-1}(y_i - 1)^2[1+10\sin^2(\pi y_{i+1})] + (y_n - 1)^2\right\}, \text{ where}$$

$y_i = 1 + 0.25(x_i - 1)$ and $(x_1, \ldots, x_n) \in [-10, 10]^n$.

Multiple LM, GM = 1, $f_{\min} = 0$ in the point (1, …, 1).

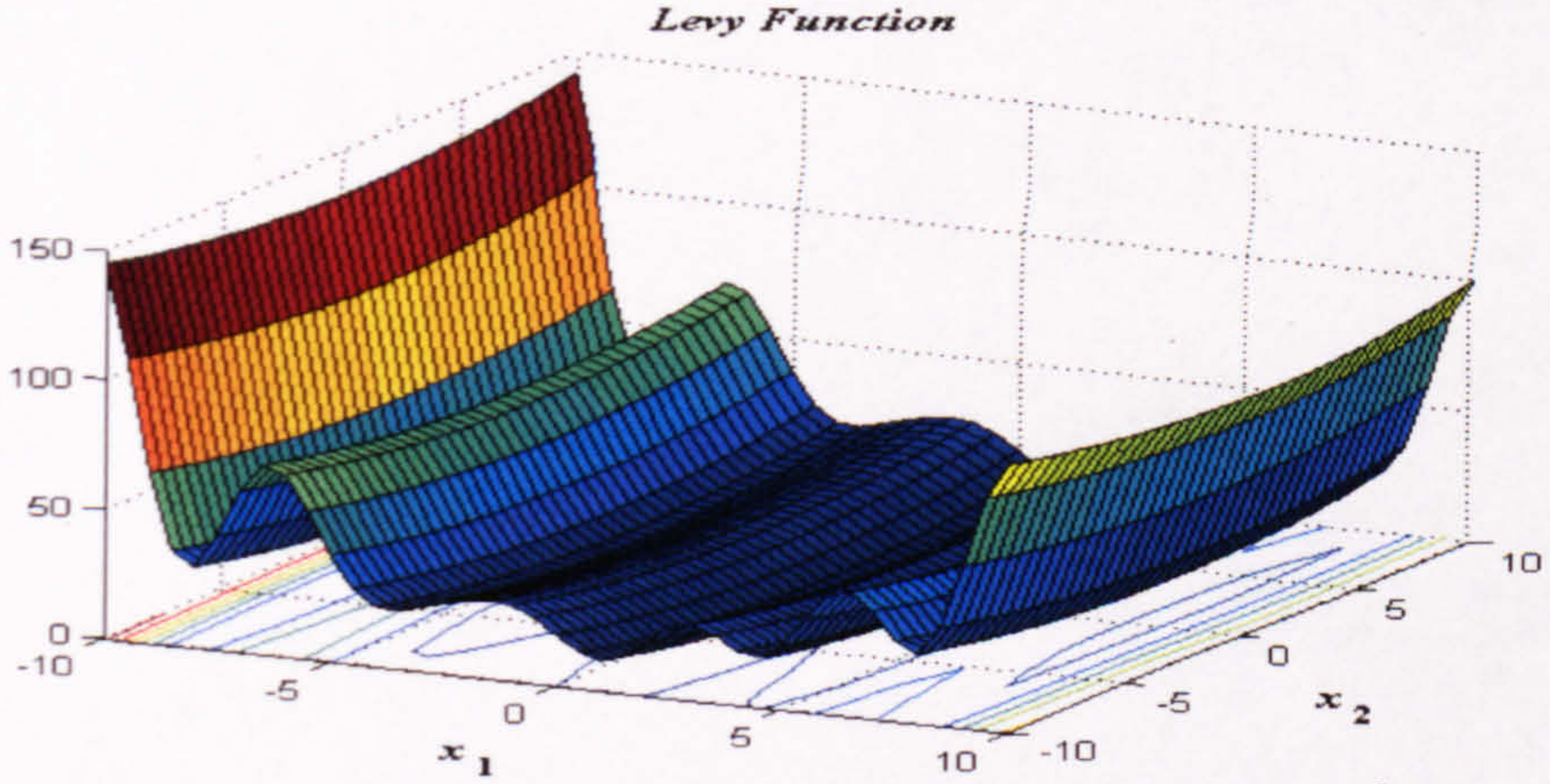


Figure C1. Two dimensional *Levy* Function in $[-10, 10]^2$.

$$f(x_1, \ldots, x_n) = \sum_{i=1}^{n-1} [(x_i^2)^{(x_{i+1}^2+1)} + (x_{i+1}^2)^{(x_i^2+1)}],\ (x_1, \ldots, x_n) \in [-1, 4]^n.$$

Multiple LM, GM=1, $f_{\min} = 0$ in the point (0, …, 0).

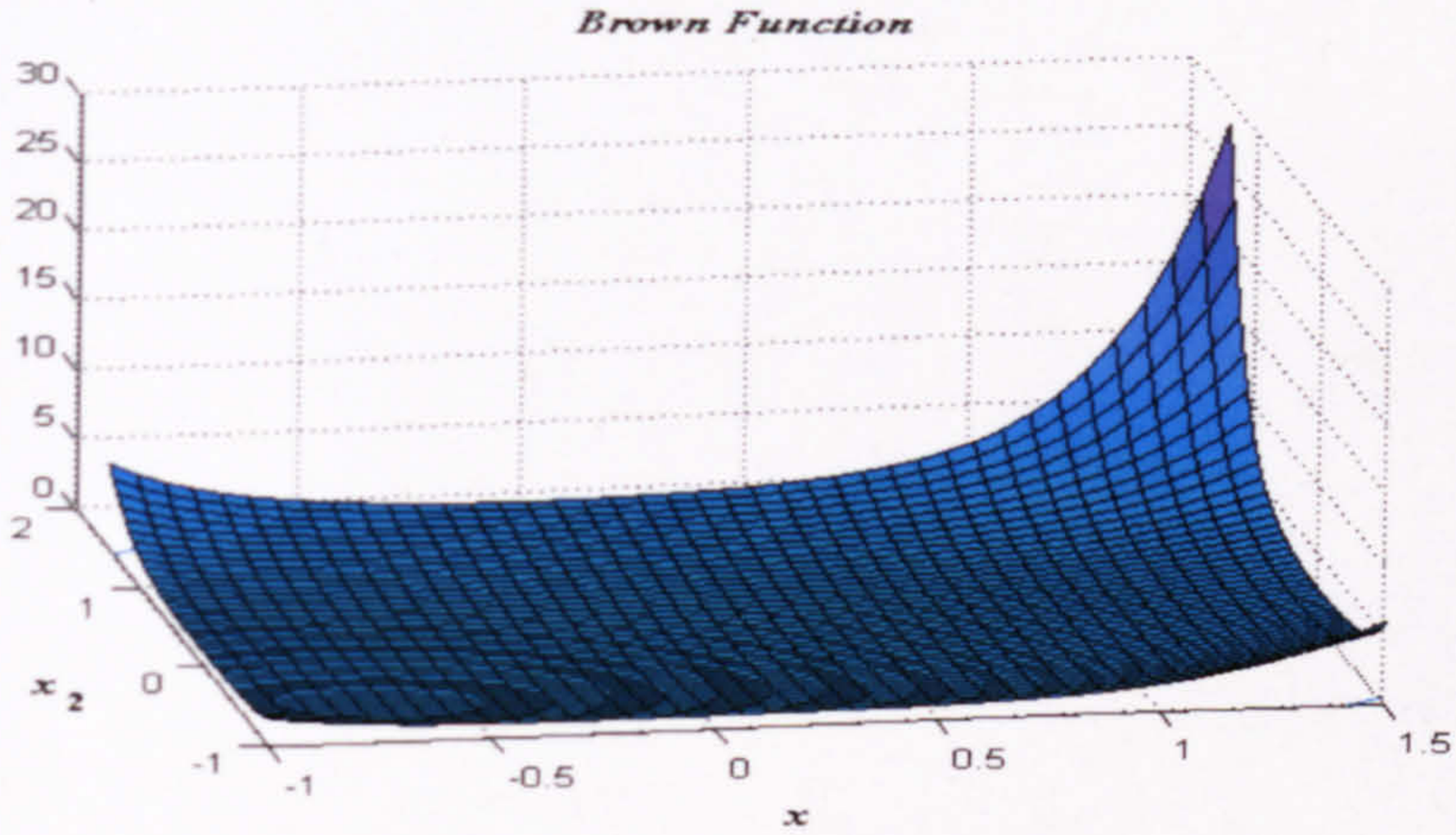


Figure C2. Two dimensional *Brown* Function in $[-1, 1.5]^2$.

- *Schwefel I* (Fig. C3), dimensionality n = 30, 100, 150.

$$f(x_1, \ldots, x_n) = \sum_{i=1} -x_i \sin(\sqrt{|x_i|}),\ (x_1, \ldots, x_n) \in [-500, 500]^n.$$

Multiple LM, GM=1, $f_{\min} = -12569.5$ in the point (420.968, …, 420.968).

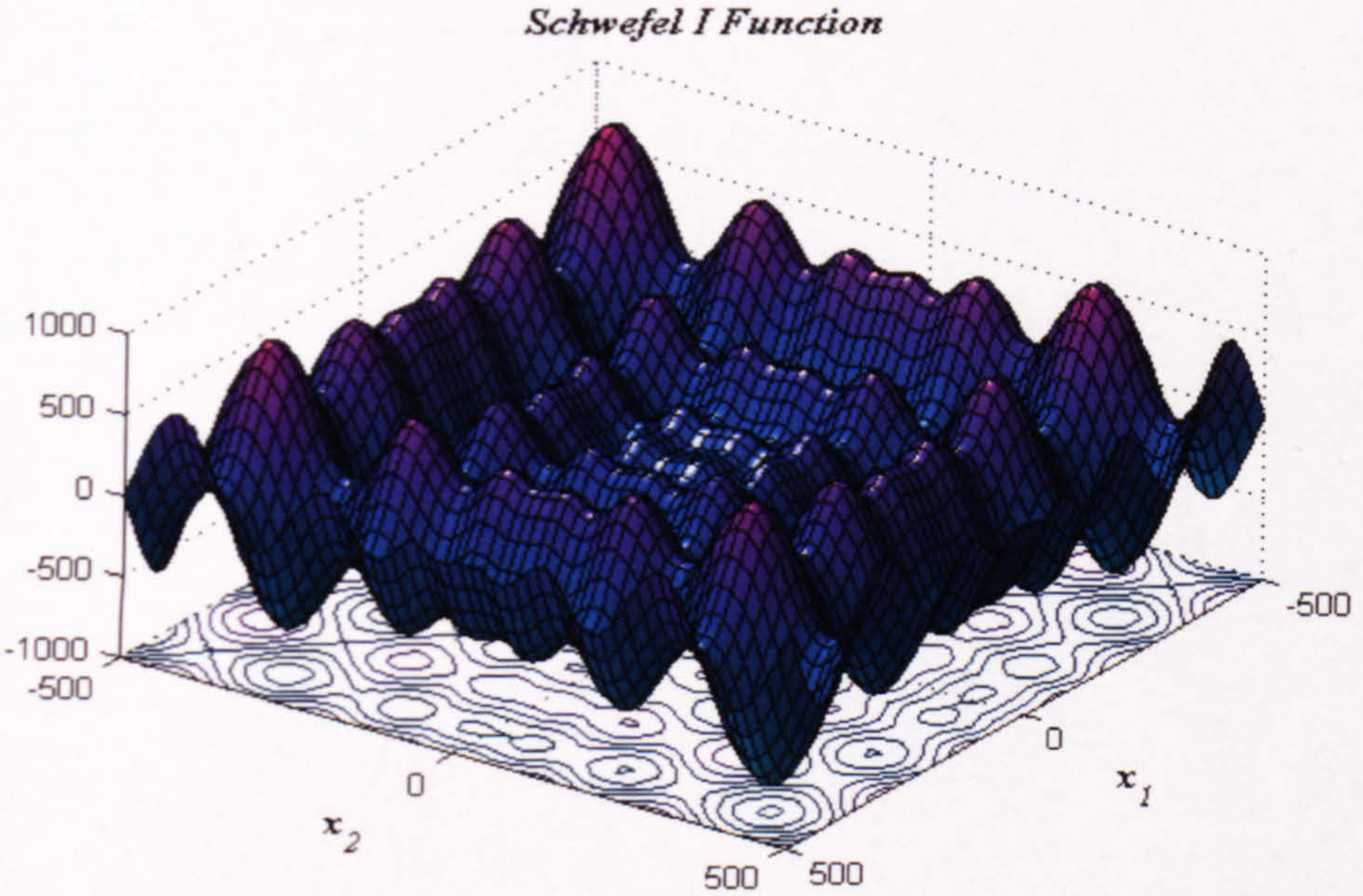


Figure C3. Two dimensional *Schwefel I* Function in $[-1, 1.5]^2$.

- *Schwefel II* (Fig. C4), dimensionality n = 30, 100, 150.

$$f(x_1, \ldots, x_n) = \sum_{i=1}^{n} |x_i| + \prod_{i=1}^{n} |x_i|, \ (x_1, \ldots, x_n) \in [-8, 12]^n.$$

LM = 1, GM = 1, $f_{\min} = 0$ in the point (0, …, 0).

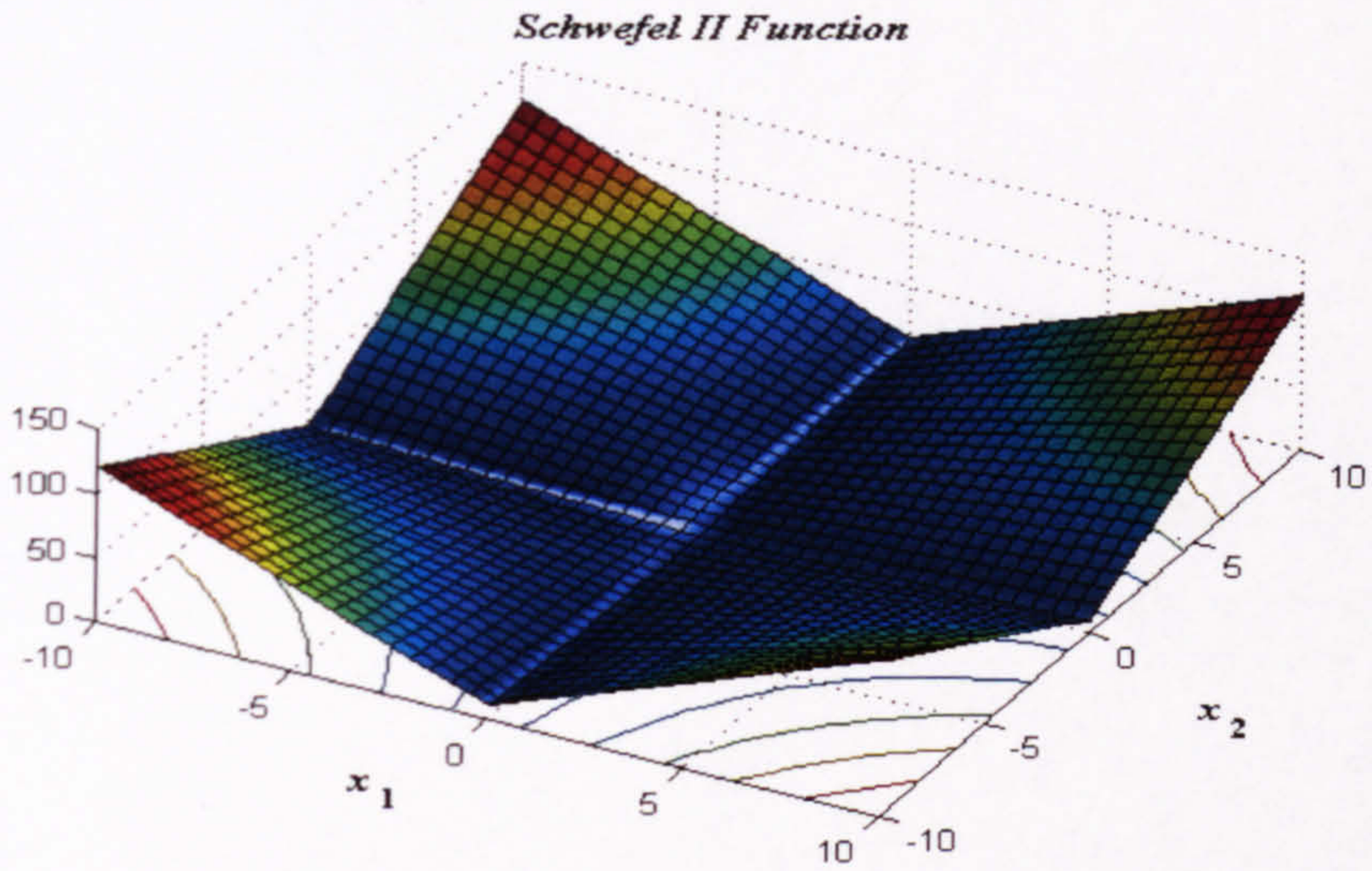


Figure C4. Two dimensional *Schwefel II* Function in $[-10, 10]^2$.

- *PLMI – Penalised Levy and Montalvo I* (Fig. C5), n = 30, 100, 150.

$$f(x_1, \ldots, x_n) = \frac{\pi}{n}\left\{10\sin^2(\pi y_1) + \sum_{i=1}^{n-1}(y_i - 1)^2[1 + 10\sin^2(\pi y_{i+1})] + (y_n - 1)^2\right\} +$$

$$+ \sum_{i=1}^{n} u(x_i, 10, 100, 4), \text{ where } y_i = 1 + \frac{1}{4}(x_i + 1) \text{ and}$$

$$u(x_i, a, k, m) = \begin{cases} k(x_i - a)^m, & x_i > a, \\ 0, & -a \le x_i \le a, \\ k(-x_i - a)^m, & x_i < -a. \end{cases} \quad (x_1, \ldots, x_n) \in [-30, 70]^n.$$

Multiple LM, GM = 1, $f_{\min} = 0$ in the point (–1, …, –1).

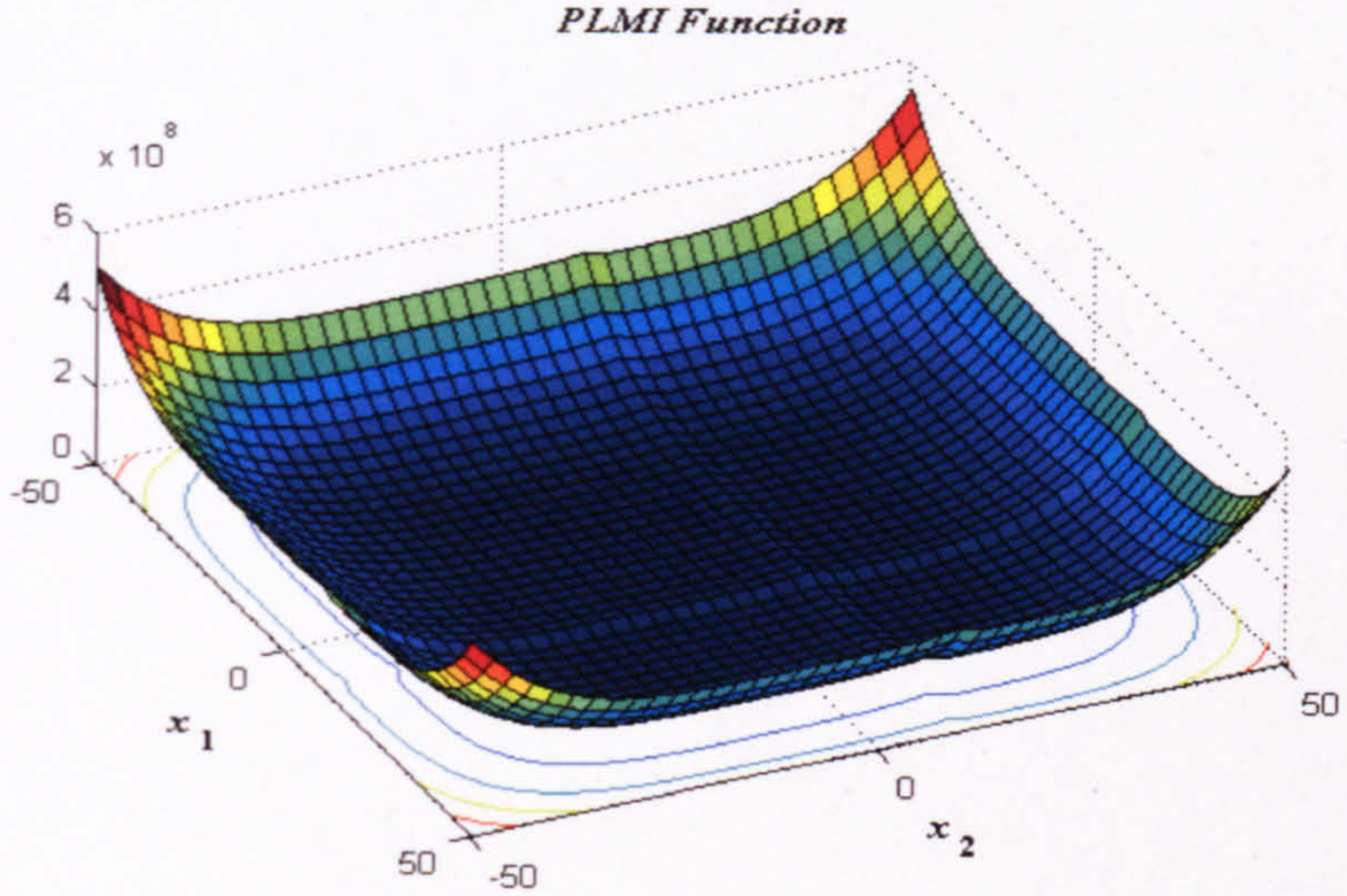


Figure C5. Two dimensional *PLMI* Function in $[-50, 50]^2$.

- *PLMII – Penalised Levy and Montalvo II* (Fig. C6), $n = 30, 100, 150$.

$$f(\boldsymbol{x}) = 0.1\left\{\sin^2(3\pi x_1) + \sum_{i=1}^{n-1}(x_i - 1)^2[1+\sin^2(3\pi x_{i+1})] + (x_n - 1)^2[1+\sin^2(2\pi x_n)]\right\} +$$

$$+ \sum_{i=1}^{n} u(x_i, 5, 100, 4), \ \boldsymbol{x} = (x_1, \ldots, x_n) \in [-70, 30]^n.$$

Multiple LM, GM = 1, $f_{\min} = 0$ in the point $(1, \ldots, 1)$.

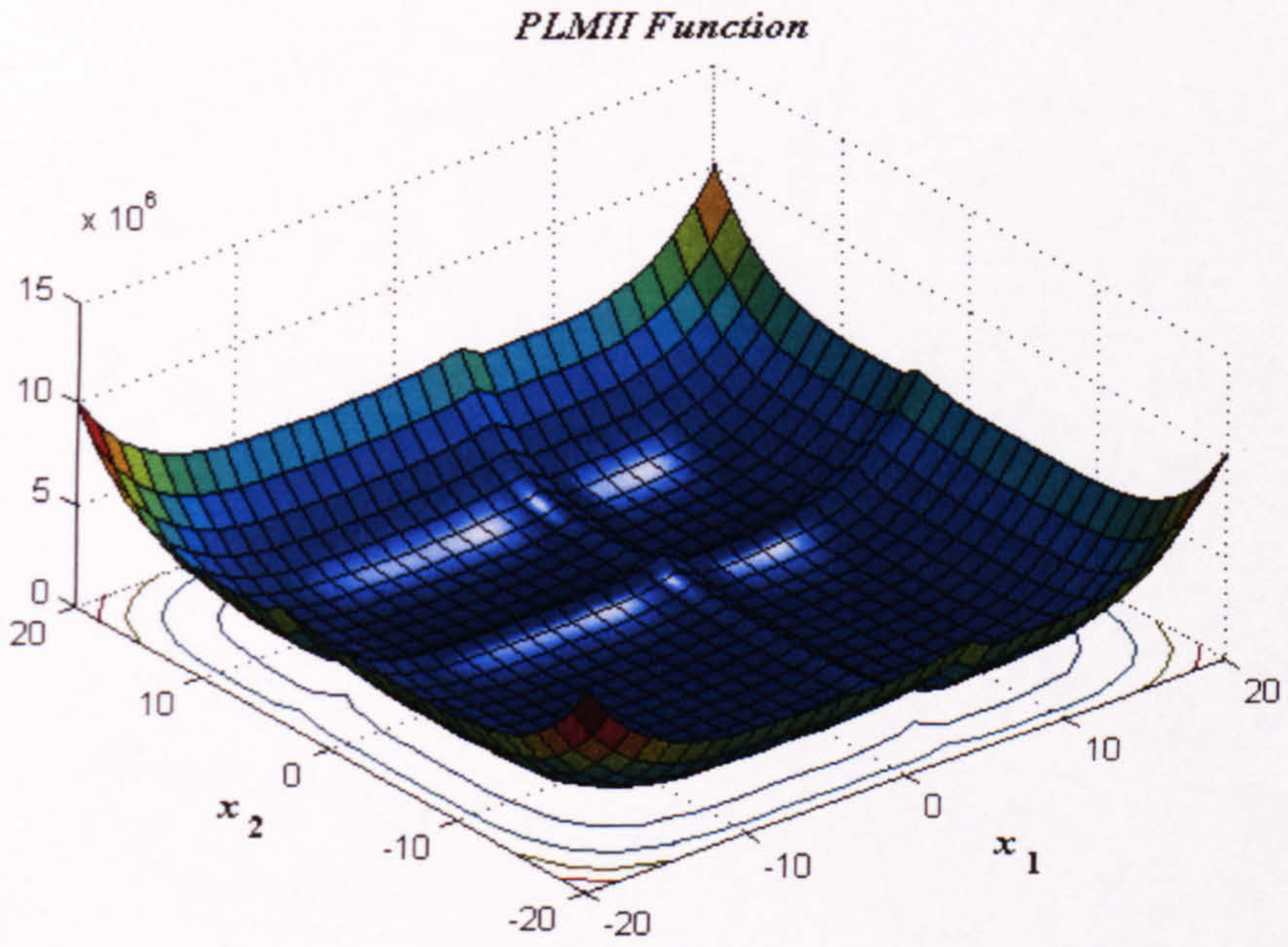


Figure C6. Two dimensional *PLMII* Function in $[-20, 20]^2$.

- *Michalevicz* (Fig. C7), dimensionality $n = 100, 150$.

$$f(x_1, \ldots, x_n) = -\sum_{i=1}^{n} \sin(x_i) \sin^{20}\left(\frac{ix_i^2}{\pi}\right), \ (x_1, \ldots, x_n) \in [0, \pi]^n.$$

LM = $n!$, GM = 1, $f_{\min} = -99.2784$, as reported in Leung and Wang, (2001).

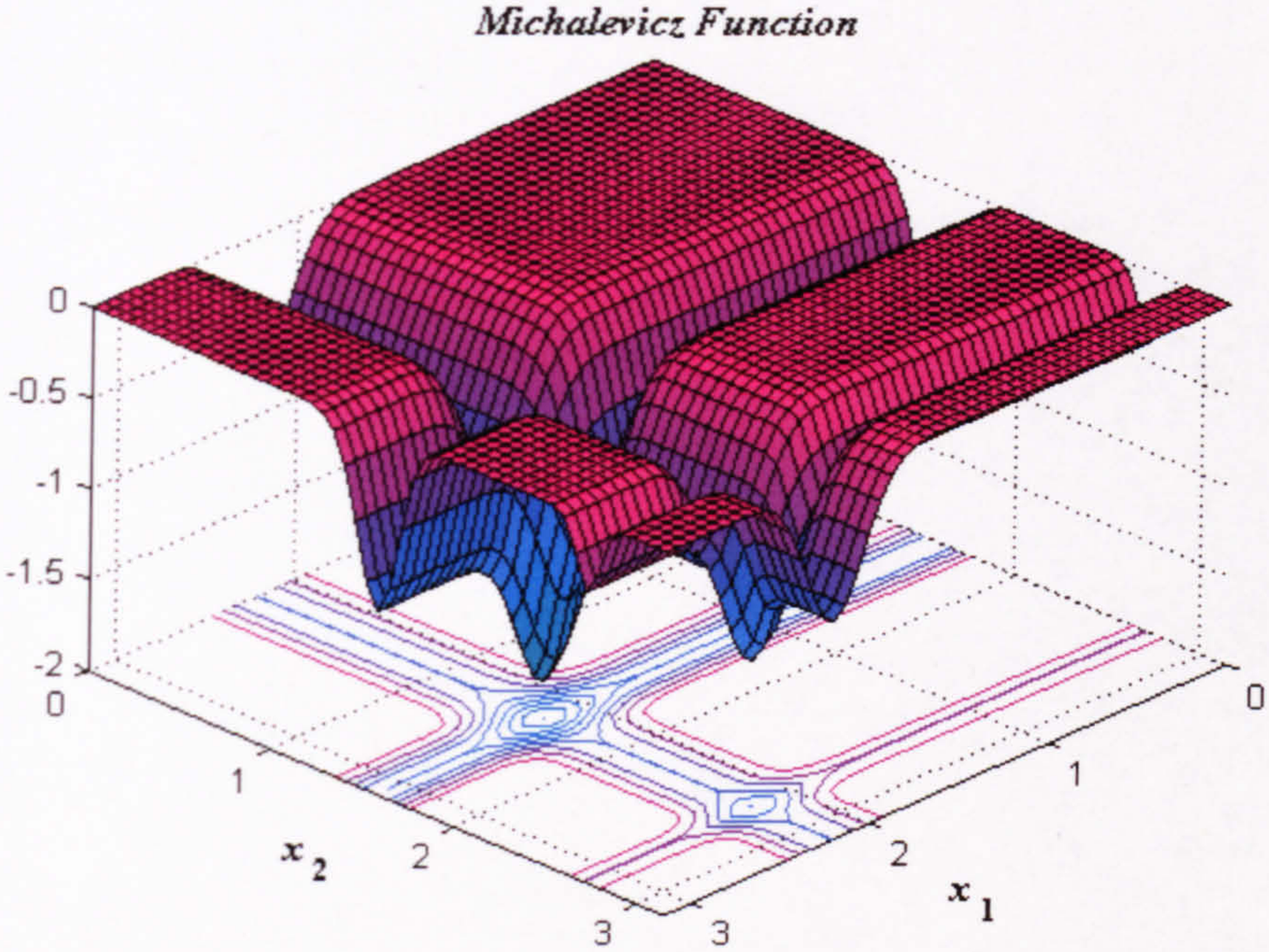


Figure C7. Two dimensional *Michalevicz* Function in $[0, \pi]^2$.

APPENDIX B: SOURCE CODE

(The full source code is attached to this thesis in a CD)

A. GLOBAL OPTIMISATION

The *LPτO* technique

For demonstrative purposes the four dimensional Shekel[7] function is used in this example. Screenshot of the output window in C++ is shown in Fig. AI and the source code (separated into three different *.cpp files) is given below:

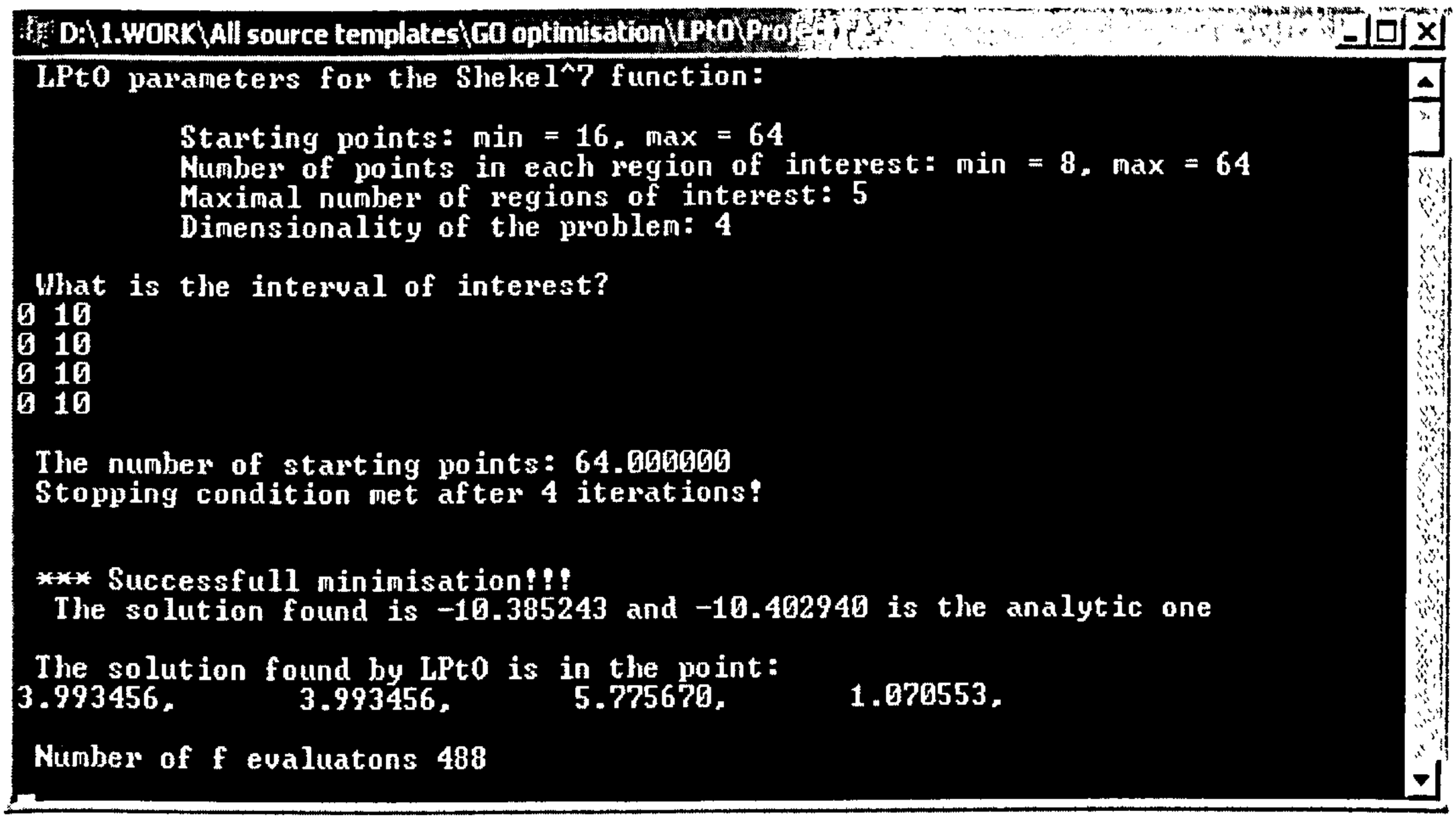


Figure AI. Optimisation of the *Shekel* [7] function with the *LPτO* method. The source code is written in C++.

- Main source code file

```
#include <stdio.h>
#include "LptO.cpp"
int min_n = 16, max_n = 64, min_nl = 8, max_nl = 64;   //LPtO input parameters
void main()
{
    //  Declare variables
        double max_int, temp;
        double min, x_min[POINTDIM]; //return parameters: minimal value and coordinates
    //  The initial interval of search is inputted by the user
        printf(" LPtO parameters for the Shekel^7 function: \n\n\t Starting points: min = %i, max = %i
",min_n, max_n);
        printf(" \n\t Number of points in each region of interest: min = %i, max = %i ",min_nl,
max_nl);
        printf("\n\t Maximal number of regions of interest: %i", NESTS);
        printf("\n\t Dimensionality of the problem: %i", POINTDIM);
        printf("\n\n What is the interval of interest?");
        for (int j=0; j<POINTDIM; j++)
            for (int i=0; i<2; i++)
```

```
            scanf("%lf", &minmax[j][i]);
//  The maximal interval length of any of the n dimensions is found and stored in max_int
      for (int i=0;i<POINTDIM;i++)
            interval[i]=minmax[i][1]-minmax[i][0];
                  max_int=interval[0];
      for (int i=1;i<POINTDIM;i++)
            if (max_int<interval[i])
                  max_int=interval[i];
//  The LPtO method is called with input parameters: min and max number
//  of initial points; min and max number of points in the successive iterations
//  x_min and min will hold the coordinates and function value of the minimum found by LPyO
       lpt(min_n, max_n, min_nl, max_nl, x_min, min);
//  Depending on the analytical minimum, the solution is considered as successfull or nonsuccessful
      if (anal_min == 0)
            temp=1.0;
        else
            temp =fabs(anal_min);
       if( fabs(min - anal_min)<0.01*temp) {
            printf("\n\n *** Successfull minimisation!!!\n  The solution found is %f and %f is the
analytic one",min,anal_min);
       }
       else{
            printf("\n\n *** Optimisation failed!!! \n The solution found is %f, but %f is the analytic
one",min,anal_min);
       }
       printf("\n\n The solution found by LPtO is in the point: \n");
       for (int i = 0; i<POINTDIM; i++)
            printf("%f, \t",x_min[i]);
       printf("\n\n Number of f evaluatons %i\n",lpt_f_eval);
       getchar();
 }
```

- File with the function to be optimised

```
//example here is Shekel^7 function
#define POINTDIM    4  // dimensionality of problem
#define anal_min  -10.40294  // The analytic minimum to be reached. This value is
                     // used only for evaluation of the result, but not as
                     // a stopping criteria.
double function (double x[POINTDIM]){
//this function works for only dim=4, where x_4=x_2 and x_1=x_3
     int i;
     double s,e;
     double c[10]={0.1,0.2,0.2,0.4,0.4,0.6,0.3,0.7,0.5,0.5};
     double A[10][2]={{4, 4 }, {1 ,1},{8, 8 },{6 ,6 },{3, 7},{2, 9},
                {5, 3},{8, 1},{ 6, 2},{7 ,3.6 }};
   s=0;
   for (i=0;i<7;i++){
         e=0;
         for (int j=0;j<2;j++){
               e+=(x[j]-A[i][j])*(x[j]-A[i][j]);
         }
         s+=-1/(2*e+c[i]);
   }
return(s);}
```

- *LPτ* Optimisation

```
/**
  OptimiSation by LPtO of a multidimensional function.
   Algorithms parameters are:
   1. Function to be optimised: Source, dimension and analitic minimum
   should be given in "function_to_be_optimised.cpp"
   2.  Nests - this is the MAXIMAL number of sub-regions to continue the search in
   3.    The min, max initial points as well as the min, max successive points
   are inputted in the main function.
**/
#include <stdio.h>
#include <math.h>
#include <stdlib.h>
#include "lp_directing.cpp"  // employed for the generation of LPt points
#include "function_to_be_optimised.cpp" // function to be optimised
// (including dimensionalitty and analytic minimum of the function to be optimised)
#define MaxNumber 2147483647   // 2**31 - 1
#define TRUE   1
#define FALSE   0

//LPtau constants
#define NUMPOINTS 1024*1024*1024     // working maximal constant
#define NESTS          5              // maximal number of subregions of interest
#define NACTIVE        2*NESTS       // Since the trail points are ALWAYS kept in
//order with nondescending function values, nowhere in the algorithm, the points that are
//at the back of the list will be used. Therefore, we always store only the first NACTIVE =
// 2*NESTS points in order to reduce memory usage
double interval[POINTDIM];   //search interval lenghts for each variable
double minmax[POINTDIM][2]; // search interval's parameters for each variable

//LPtO working variables
int m=0; //m dynamically changes and shows the current length of f
double start; //starting points
double nest_R[NESTS]; //radii of each nest
int label[NACTIVE];  // label of each nest, updated at each iteration
double f_stop,x_stop=0;     //stopping condition variables
int  lpt_f_eval=0;          // function evaluations counter
/*-------------------------------------------------------------------
                The function for generating points
-------------------------------------------------------------------*/
int lptau(long i, int n, double Vector[NUMPOINTS])
{
  long  BitString;
  int   j, k;
  /* trap for a wrong subroutine call */
  if (i > MaxNumber || i<0 || n > MaxDimension || n < 1) return TRUE;
  if((PrevNum+1) != i || n > PrevDim)
{  /*generation of the point from "i" and "n"  */
  for(j = 0; j < n; j++) MaskVect[j] = 0;
  for(k = 0, BitString = i^(i>>1); BitString; BitString>>=1, k++)
    if(BitString & 1)
      for(j = 0; j < n; j++)
        MaskVect[j] ^= Directing[k][j];
    for(j = 0; j < n; j++)
      Vector[j] = MaskVect[j] * scale;
}
else    /* recurrent generation of the point          */
{      /* k - position of the most right "1" in binary
          representation of the number i             */
  for(k=0, BitString=i; (BitString & 1)==0; k++, BitString>>=1);
  for(j = 0; j < n; j++)  {
```

```
        MaskVect[j] ^= Directing[k][j];
        Vector[j] = MaskVect[j] * scale;
      }
    }
   PrevNum = i;
   PrevDim = n;
   return 0;
}
/*-------------------------------------------------------------------------
                          Opening a file
-------------------------------------------------------------------------*/
FILE *OpenFile(char* fname, char* mode)
{ FILE *fp;
      if( !(fp=fopen(fname, mode))) {
            printf("\n Error in opening \"%s\"!", fname);
   getchar();
            exit(1);
      }
      return(fp);
}
/*-------------------------------------------------------------------------
                          Printout
-------------------------------------------------------------------------*/
// prints out the values of the arrey sf from 0 to n-1-th place
void printout(double sf[], int n){
      printf("\n\n The function values are:\n");
      for(int i=0;i<n;i++)
            printf("%f\t",sf[i]);
}
/*-------------------------------------------------------------------------
                          Radius vector
-------------------------------------------------------------------------*/
double compute_radius(double x[POINTDIM]){
//input is a point with POINTDIM coordinates and this function returns the
// radius sqrt(x1^2+x2^2+..._xn^2)
    double temp=0;
    for (int i=0;i<POINTDIM;i++)
          temp += x[i]*x[i];
    return sqrt(temp);
} .
/*-------------------------------------------------------------------------
                    THE NUMBER R
-------------------------------------------------------------------------*/
double Rr(double minmax[][2],int u){
// inputs are the area of interest and the number of points in it
//this function returns the metric R of the average distance between
//any 2 neighbouring points

double s,td=0.0,interval[POINTDIM];
double ttR;
for (int j=0; j<POINTDIM; j++) {
   interval[j]=fabs(minmax[j][1]-minmax[j][0]);
   }
   td=1.0;
   for (int i=0;i<POINTDIM;i++) //evaluating the volume
   td*=interval[i];

   s=POINTDIM;
   td/=u; //dividing by N
   ttR=pow(td,1/s)*sqrt(POINTDIM); //taking the n-th root,multiplying by sqrt(n)
```

```
    return(ttR);
 }
/* ----------------------------------------------------------------------
        Generating lp points and corresponding F values
----------------------------------------------------------------------*/
void fValues(double* s,double xx[][POINTDIM],int ww,int e,double mm[][2]){
/**
  Here new LPt points are generated one by one, the function value computed,
  and each point is put in the right place of the array, so the values are in
  nondecreasing order
**/
      int i,j,k;
      double tq[POINTDIM];
      double temp;
      double inter[POINTDIM];
      double tx[POINTDIM];

      for (i=0;i<POINTDIM;i++)
          inter[i]=mm[i][1]-mm[i][0];
      for (i=ww ; i<e; i++) {
//new point
          lptau(i, POINTDIM, tq);      // call lptau()
          for (j=0; j<POINTDIM; j++){
               tx[j]=mm[j][0]+ inter[j]*tq[j];
           }
           temp=function(tx);
           for (k=0;k<=m;k++){
//find a place for the new one
             if (temp<s[k]){
//initially s is filled with HUGE_VALs and the condition temp<s[k] will be aways held before
// NACTIVE values have been already computed.
//rearange old values
                      for (int u=m+1;u>k;u--){ //values in s and xx are rearanged
                           s[u]=s[u-1];
                           for (j=0;j<POINTDIM;j++)
                                xx[u][j]=xx[u-1][j];
                       }
//put in the new one
                      s[k]=temp;
                      for (j=0;j<POINTDIM;j++)
                           xx[k][j]=tx[j];
                      if (m<NACTIVE-2)
                           m++;
                      break;
                 }//closes last if
            }//closes k cycle
      }//stops generating new points
}
/*----------------------------------------------------------------------
                    STOPPING CONDITION
----------------------------------------------------------------------*/
 int stop(double* f,double tx[POINTDIM]){
// The algorithm will exit if the current best function value f_stop is not updated
  double rr;
  rr=compute_radius(tx);
  int flag=TRUE;
  if (fabs(fabs(f_stop)-fabs(f[0]))< 0.01 & fabs(rr-x_stop)<0.01)
    flag=FALSE;
   return(flag);
```

```
 }
/*-------------------------------------------------------------------------
                        DISTANCE
-------------------------------------------------------------------------*/
double distance(double x1[POINTDIM],double x2[POINTDIM]){
// returns the Eucledian distance between two points
      double s,ee;
      s=0;ee=0;
      for (int j=0;j<POINTDIM;j++){
          ee=(x1[j]-x2[j]);
          s+=ee*ee;
      }
      return sqrt(s);
}
    /*-------------------------------------------------------------------------
                  Selection of the successive number of points
    -------------------------------------------------------------------------*/
    int Number(double* ttf,double tx[][POINTDIM], double mm[][2], int NMINL,int NMAXL){
    /**
      In each region of interest the bumber of points to be generated is selected here
      in a range between NMINL and NMAXL which are inputted by the main function
    **/
    int n;
    double tempr,s,e;
   int far,grow,change=TRUE,tempm,ttR,ww=0;//ww remembers which point to start from if
//CHANGE
     n=tempm=NMINL;
     m=0;
     for (int i=0;i<=NACTIVE;i++)
        ttf[i]=HUGE_VAL;
while(tempm<2*NMAXL & change) {
      change=FALSE;
      fValues(ttf,tx,ww,n,mm);
      for (int i=1;i<2*POINTDIM;i++){
        tempr=distance(tx[0],tx[i]);
        ttR=Rr(mm,n);
        s=fabs(ttf[i]-ttf[0])/fabs(ttf[0]);
        if (tempr<0.5*ttR)
          far=FALSE;
        else
          far=TRUE;
        if (s<0.25)
          grow=FALSE;
        else
          grow=TRUE;
        if (far&grow){
         tempm *= 2;
          change=TRUE;
          ww=n;
          break;
         }
       }
       n=tempm;
 }
if (tempm==2*NMAXL)
     n=NMAXL;

return(n);}
/*-------------------------------------------------------------------------
                        Iterations
```

```
---------------------------------------------------------------------- */
 void cycle(double* tf,double tx[][POINTDIM],double minmax[][2],int NMINL,int NMAXL){
// The main loop which coordinates the algorithm at each iteration
    double f1[NACTIVE+1];
    double x1[NACTIVE+1][POINTDIM];
    double tempx[NESTS][POINTDIM];
    double mm[POINTDIM][2];
    int tlabel[NESTS],z=2,c=0,p[NESTS],far,grow; //c is a helping counter
//working variables
    double t1[NESTS],interval[POINTDIM],r[NESTS];
    double s,ee,tempr;
    double tempbb;
    int  k,a=0,MNew,e=0;
    for (int j=0;j<NESTS;j++)
       r[j]=0;
// the current best minimum is updated
    t1[0]=f_stop=tf[0];
    x_stop=compute_radius(tx[0]);
    MNew=1;
    for (int i=0;i<POINTDIM;i++)
        tempx[0][i]=tx[0][i];

    for (int i=1;i<NESTS;i++){
        tempr=distance(tx[0],tx[i]);
        s=fabs(tf[i]-tf[0])/fabs(tf[0]);
        z=label[0];
        if (tempr<0.5*nest_R[z])
            far=FALSE;
        else
            far=TRUE;
    if (s<0.25)
        grow=FALSE;
    else
        grow=TRUE;
    if (!far){
        if (grow){
            if (r[0]==0)
                r[0]=0.5*nest_R[z]/(2*sqrt(POINTDIM)); //small
        }
        else {
            if (r[0]==0)
                r[0]=0.5*nest_R[z]/(2*sqrt(POINTDIM)); //
            MNew+=1;
            r[MNew-1]=0.25*nest_R[label[i]]/(2*sqrt(POINTDIM));  //small;
            for (int j=0;j<POINTDIM;j++)
                tempx[MNew-1][j]=tx[i][j];
            t1[MNew-1]=tf[i];
        }
    }
    else {
        if(grow){
            r[0]=1.33*nest_R[z]/(2*sqrt(POINTDIM));  //big
        }
        else {
            MNew+=1;
            r[0]=0.66*nest_R[z]/(2*sqrt(POINTDIM));  //big
            r[MNew-1]=0.66*nest_R[label[i]]/(2*sqrt(POINTDIM));  //big
            for (int j=0;j<POINTDIM;j++)
                tempx[MNew-1][j]=tx[i][j];
            t1[MNew-1]=tf[i];
```

```
            }
        }
    }
    for (int i=0;i<=NACTIVE;i++)
       tf[i]=HUGE_VAL; //old values of tf are lost,new values are placed instead

    for (int i=0;i<MNew;i++){
       for (int j=0;j<POINTDIM;j++){
            mm[j][0]=tempx[i][j]-r[i];
            mm[j][1]=tempx[i][j]+r[i];
       }
       p[i]=Number(f1,x1,mm,NMINL,NMAXL);
//after execution of Number() we have renewed m
// For each nest, corresponds local tR[i]
      nest_R[i]=Rr(mm,p[i]);
      for(int ii=0;ii<m;ii++){
        a=c+ii; // a counts the actual total number of new points.
            // If they are more than NACTIVE, a hold the value of NACTIVE. If it is
            // less than NACTIVE, a shows the total number
        if (a>NACTIVE)
           a=NACTIVE;
        for ( k=0;k<=a;k++){
           if (f1[ii]<tf[k]){ //find a place in tf for the new value
                for (int u=a;u>k;u--){ //values in tf and tx are rearranged
                     tf[u]=tf[u-1];
                     label[u]=label[u-1];
                     for (int j=0;j<POINTDIM;j++)
                        tx[u][j]=tx[u-1][j];
                }
                tf[k]=f1[ii];
                label[k]=i;
                     //on k-th place the new point and value are placed
                for (int j=0;j<POINTDIM;j++)
                     tx[k][j]=x1[ii][j];
                break;  //place is found no need to go further
           }
        }
    }
    c=a+1; //c only helps it is not used later anywhere else
}
    a=0;
    for (int i=0;i<MNew;i++)
          a+=p[i];
    if (a<NACTIVE)
          m=a;   // m is updated and is now showing the current length of the array tf
    else
          m=NACTIVE;

       lpt_f_eval+=a;
}

/*-------------------------------------------------------------------------
                        STARTING POINTS
-------------------------------------------------------------------------*/
void startP(double* f,double x[][POINTDIM],double mm[][2], int NMIN,int NMAX){
//the number of initial points is determined in the user defined range NMIN and NMAX
  int far,grow,change=TRUE,tempm,ww=0;//ww remembers which point to start from if CHANGE
  double ee,s;
  tempm=start=NMIN;
  m=0;
```

```
for (int i=0;i<NACTIVE;i++)
  f[i]=HUGE_VAL;
while(start<=NMAX & change) {
    change=FALSE;
    fValues(f,x,ww,start,mm);
    nest_R[0]=Rr(mm,start);
    for (int i=1;i<POINTDIM;i++){
      ee=distance(x[0],x[i]);
      s=fabs(f[i]-f[0])/fabs(f[0]);
      if (ee<(0.5*nest_R[0]))
        far=FALSE;
      else
          far=TRUE;
      if (s<0.25)
        grow=FALSE;
      else
        grow=TRUE;
      if (far&grow){
       ww=tempm;
       tempm *= 2;
       change=TRUE;
       break;
      }
    }
    start=tempm;
}//end of while
if(tempm==2*NMAX)
    start=NMAX;
for (int i=1;i<NESTS;i++){
    nest_R[i]=nest_R[0];
  }
  for (int i=0;i<NACTIVE;i++)
      label[i]=0;}
  /*-------------------------------------------------------------------
                        LPtO Algorithm
  -------------------------------------------------------------------*/
  void lpt( int NMIN, int NMAX,int NMINL, int NMAXL, double xmin[POINTDIM],double
&min){
      int c=0; //c counts the number of cycles and e counts the function evaluations
      double f[NACTIVE+1]; //store the fun values
      double x[NACTIVE+1][POINTDIM];
      int i,j;
      startP(f,x,minmax, NMIN, NMAX);
      printf("\n The number of starting points: %f ",start);
      lpt_f_eval=start;
      f_stop=HUGE_VAL;
//maximal number of iterations is given here
   for (i=0;i<30;i++){
        if (stop(f,x[0])||c==1){
            c++;
            cycle(f,x,minmax,NMINL,NMAXL);
        }
        else
        {
            printf("\n Stopping condition met after %i iterations! \n",c);
            break;
        }
  }
  for (j=0;j<POINTDIM;j++)
        xmin[j]=x[0][j];
```

```
min=f[0];
getchar();}
```

The hybrid *GLPrS* technique

For demonstrative purposes the 30 dimensional *PLMI* function is used in this example. Screenshot of the output window in C++ is shown in Fig. AII and the source code could be found in the attached CD:

```
D:\1.WORK\G+LPSS_Paper\Project1.exe

***** This is run Number 1 *****

Minimum found by GA: 744667.303121

*****************************

LptO search in the GA individual number 0
Radius of search=5.100000
LPtO stopping condition met on iterqation 13!

Minimal value found by this LPtO is : min=3.798017
*****************************

LptO search in the GA individual number  2
LPtO stopping condition met on iterqation 19!

Minimal value found by this LPtO is : min=161.366933
Radius of search = 4.129829

*****************************

LptO search in the GA individual number  4
LPtO stopping condition met on iterqation 19!

Minimal value found by this LPtO is : min=208.846610
Radius of search = 4.119740

*****************************

Best function value found by NM simplex = 0.000097

***** This is run Number 2 *****

Minimum found by GA: 0.024142

*****************************

LptO search in the GA individual number 0
Radius of search=5.100000
LPtO stopping condition met on iterqation 2!

Minimal value found by this LPtO is : min=0.024142
*****************************

Best function value found by NM simplex = 0.000000

Rate of success: 2 out of 2

Average f value from GA -> 191371.408361 and from LPtO-> 4.911229

Mean number of f evaluations from GA 5859.500000, LPtO 56084.500000, and NM 136
01.000000 and Initial 64

Total mean number of f evaluations :75545.000000

Mean number of LPtO searches: 2.000000

Mean function value=0.000097 <0.000097>
```

Figure AII. Optimisation of the *PLMI* function with the *GLPrS* method. The source code is written in C++.

B. NEURAL NETWORK TRAINING

For demonstrative purposes, the case of Experiment III NN learning with LDA data set and binary coding of targets is shown here. Screenshots from the output window in C++ is given in Fig. BI. It shows part of the testing patterns, together with desired, actual output, and error for each pattern. At the bottom, the average results of 50 runs are presented.

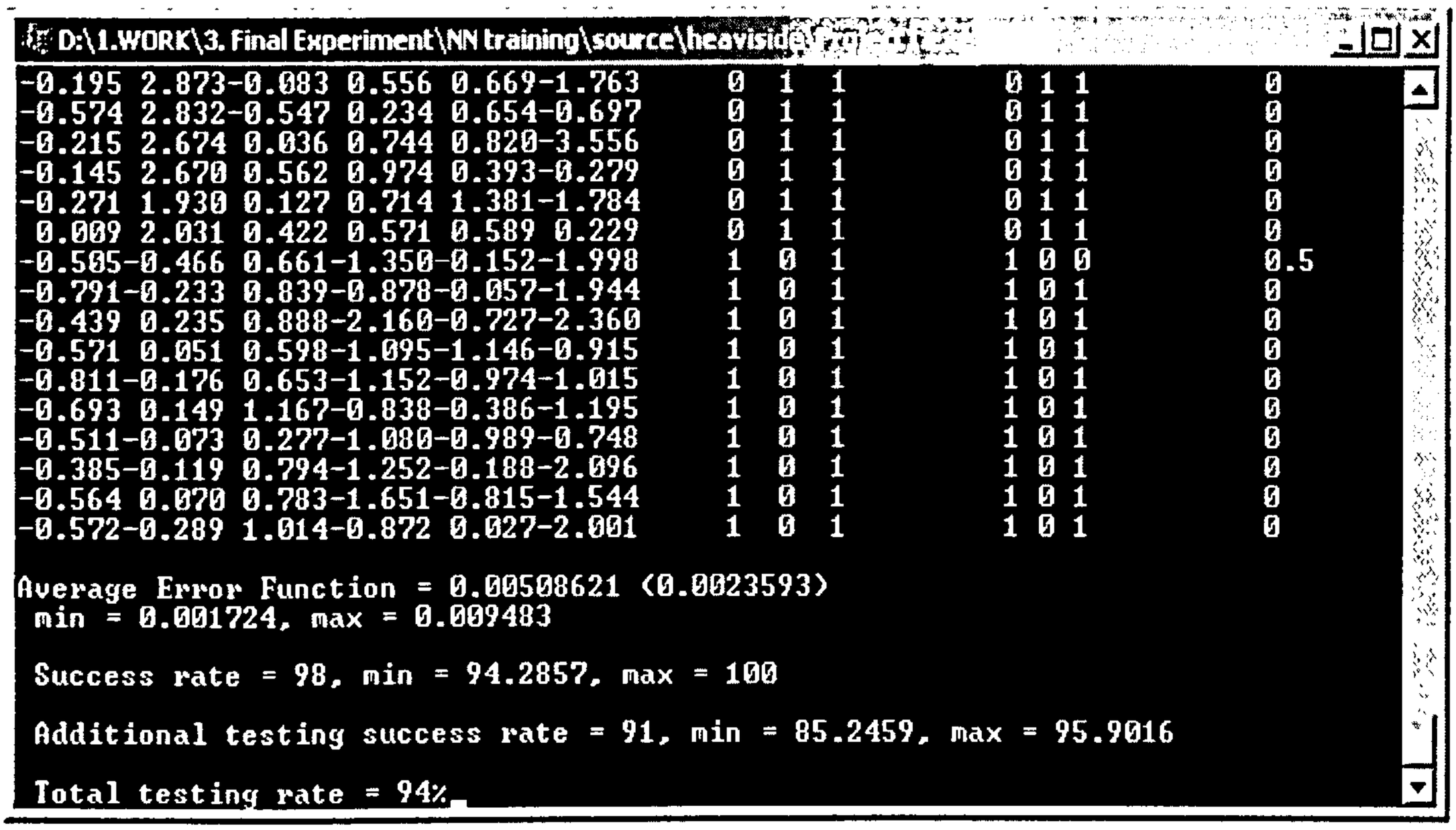


Figure BI. Neural Network training and evaluation Experiment III (LDA dataset and binary targets coding).

Some of the specific functions for handling the NN learning and evaluation are shown below:

```
/*-------------------------------------------------------------------
                    The error function
-------------------------------------------------------------------*/
 double F(double* tw){
   double temp;
  //loads the weight values in the array w
   for (int i=0;i<POINTDIM;i++)
       w[i]=tw[i];
   LptauWeights(&Net);
   temp=TrainNetAverage(&Net);
   return(temp);
}
/*-------------------------------------------------------------------
                    Loading weights into the net
-------------------------------------------------------------------*/
void LptauWeights(NET* Net)
{
```

```
  // w must be initialized - either reading values from file or randomly
  int i,j,k = 0;
//if the weights have been initialized, it is not possible that both first coordinates are zero
  if (w[0] == 0 & w[1] == 0)
      printf( "Weight have not been initialized!!!! Do not consider results as relevant!!");
  for (k=1; k<N_LAYERS; ++k)
       for (i=1; i<=Net->Layer[k]->Neurons; ++i)
                 for (j=0; j<=Net->Layer[k-1]->Neurons; ++j)
                Net->Layer[k]->Weight[i][j]=w[n++];
}

/*-------------------------------------------------------------------
          PROPAGATING SIGNALS
-------------------------------------------------------------------*/
void PropagateLayer (LAYER* Lower, LAYER* Upper)
{
  int  i,j;
  double Sum;
  for (i=1; i<=Upper->Neurons; ++i) {
   Sum = 0.0;
    for (j=0; j<=Lower->Neurons; ++j)
        Sum += Upper->Weight[i][j] * Lower->Output[j];
    Upper->Output[i] = 1.0/(1.0 + exp(-Sum));
  }
}

void PropagateLast (LAYER* Lower, LAYER* Upper)
{
      //this is used when the function in the output layer
      //is not sigmoid but treshold
  int  i,j;
  double Sum;
  for (i=1; i<=Upper->Neurons; ++i) {
       Sum = 0.0;
  for (j=0; j<=Lower->Neurons; ++j)
       Sum += Upper->Weight[i][j] * Lower->Output[j];
    if (Sum <= 0)
Upper->Output[i] = 0;
    else if (Sum > 0)
        Upper->Output[i] = 1;
  }
}

void PropagateNet (NET* Net)
{
  int k;
// N1=N_LAYERS-1
  for (k=0; k<N1-1;  ++k)
      PropagateLayer(Net->Layer[k], Net->Layer[k+1]);
      PropagateLast(Net->Layer[1], Net->Layer[2]);
}
/*-------------------------------------------------------------------
            SIMULATING THE NET
-------------------------------------------------------------------*/
void SimulateNet (NET* Net, ROW Input, double Target[])
{
  SetInput(Net, Input);
  PropagateNet(Net);
  ComputeOutputError(Net, Target);
}
```

```
double TrainNetAverage (NET* Net){
 /* for each pattern simulates the net and finds the average error */
       int  i;
       double Output[M], error=0.0;
  for (i=0; i<N_Pat; ++i) {
  SimulateNet(Net, dx[i],dy[i]);
    error += Net->Error;
  }
  return (error/N_Pat);
}
```

C. IMAGE AND DATA PROCESSING

Feature Extraction

The feature extraction stage is demonstrated here with two images – 25 Laws features are obtained together with 8 co-occurrence ones. The sourceis written entirely in Matlab.

- Main functions

```
clear all
% myList1 contains all the available images
myList1 = [cellstr('images\im1.bmp'),cellstr('images\im2.bmp')];
N = length(myList1);
vals1 = [];
for i = 1:1:N
   [I1,I2] = cut(str2mat(myList1(i)));
% 'cut.m' converts the image from RGB to Grayscale mode and returns the
%image at position i in myList1 cut into two 460x340 pxls subimages.
   vals1 = [vals1;imageproc(I1)];
   % imageproc3 returns an array containing 33 feature values (double)
   %for the subimage I1
   vals1 = [vals1;imageproc(I2)];
end
% the array vals1 contains N*2 rows and NumOfFeatures (33) columns which gives us the data
% table for the features. They are saved in a file by the
% next line:
save features.dat vals1 -ascii -tabs
```

```
function f = imageproc(I)
global N M
M = length(I(1,:));
N = length(I(:,1));
S = 15;
% S is the size of the  window that we use for the averaging.
% averaging is recommended by Laws, 1980; Shapiro and Stockman (2001); Umbaugh, 2005(on
page 277);
A1 = window_avm(I,S);
% features are extracted:
f = [cooc_process(A1),laws_process(A1)];
```

- The co-occurrence feature extraction function

```
function f = cooc_process(A)
% first we compute the co-occurence matrix for the near neighbours in the
% four directions
M1 = graycomatrix(A,'Offset',[0 1;-1 1; -1 0; -1 -1],'NumLevels',15); % this is a Matlab function
% then compute the features:
stats1 = graycoprops(M1,'Contrast Correlation Homogeneity Energy');
% and use the mean of the values in all directions so our features become
% rotation invariant (Randen and Hus?y, 1999).
f = [mean(stats1.Contrast), mean(stats1.Correlation), mean(stats1.Homogeneity),
mean(stats1.Energy)];
% we compute the co-occurence matrix for the more distant (5 pixels) members in the
% four directions
M1 = graycomatrix(A,'Offset',[0 5;-5 5; -5 0; -5 -5],'NumLevels',15);
% then compute the features
stats1 = graycoprops(M1,'Contrast Correlation Homogeneity Energy');
% and use the mean of the values in all directions - rotation invariant (Theodoridis and Kotroumbas,
2006)
f = [f, mean(stats1.Contrast), mean(stats1.Correlation), mean(stats1.Homogeneity),
mean(stats1.Energy)];
```

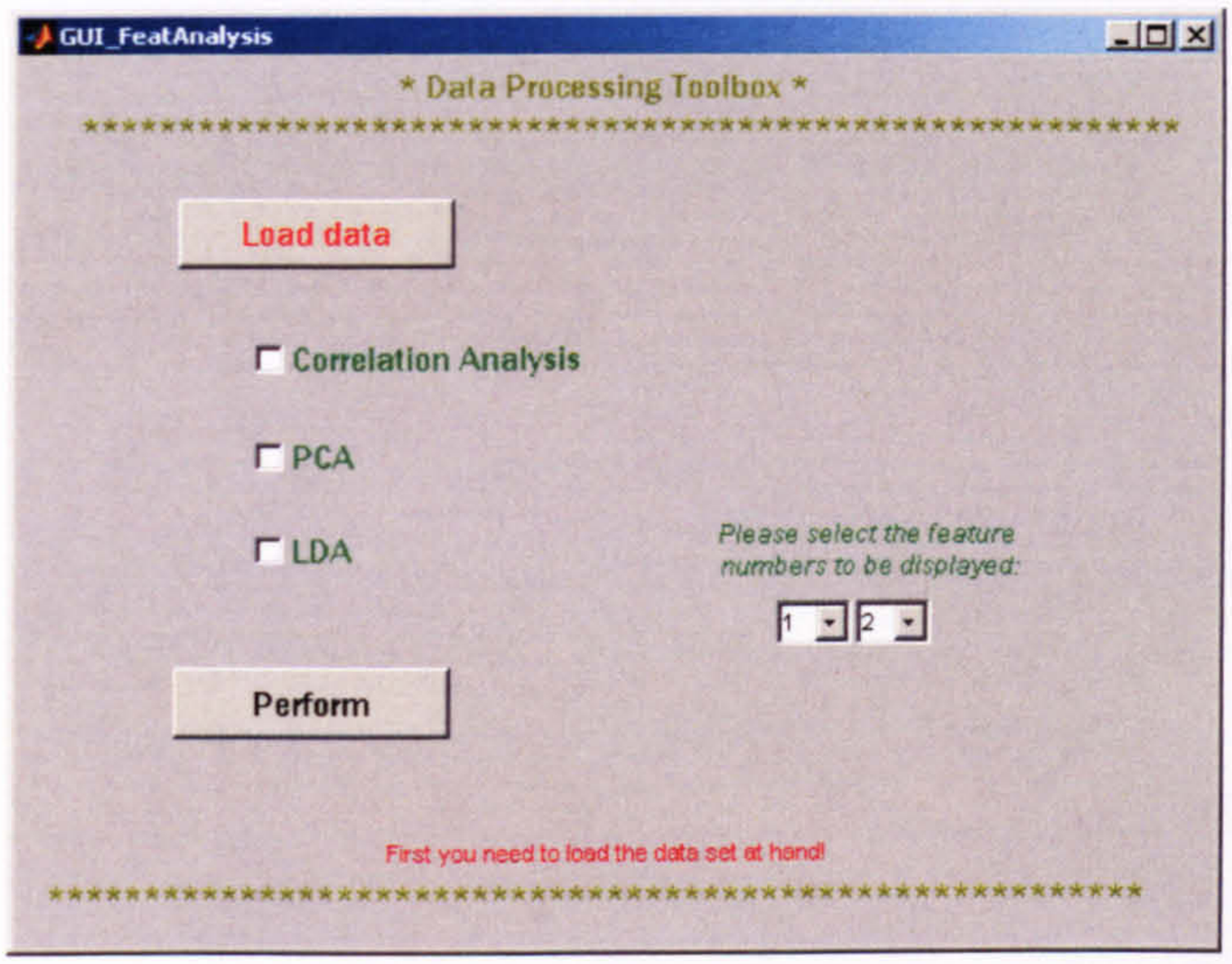
- The Laws feature extraction function

```
function f = laws_process(I,N,M)
 global N M

% 25 features are extracted by using 25 Laws masks and a 'power' averaging.
[A, E] = gen_laws(); % Generates the 25 Laws masks and they are accessible through the variableA
% E is the size of Laws Masks, defined in gen_laws(). By default they are 5x5 (E = 5), but they
%could be 3x3 as well, see Davies E.R.(2005).
 % Convolving with all masks
% B contains all convolutions of the image I and the 25 masks
B = conv_all(I, A, E);
 % Energy map, for details refer to Laws (1980).
a = [];
N1 = N+E-1;
M1 = M+E-1;
for i = 1:M1:E*E*M1
   I2 = B(1:N1,i:i+M1-1);
   a = [a,energy(I2)];
end
 f = a; % a is a list of 25 numbers

% This function is generating Laws 5x5 masks and holding them all in the
% matrix A. Returns also the size of the masks.
function [A,E] = gen_laws();
E = 5;    % the size of Laws Masks. By default they are 5x5, but they
%could be 3x3 as well, see Davies E.R. (2005) and Sonka et al. (2007).
 % The base vectors:
 a = [1, 4, 6, 4, 1];
b = [-1, -2, 0, 2, 1];
c = [-1, 0, 2, 0, -1];
d = [1, -4, 6, -4, 1];
e = [-1, 2, 0, -2, -1];
 % Forming the masks
A1 = transpose(a)*a;
A2 = transpose(a)*b;
```

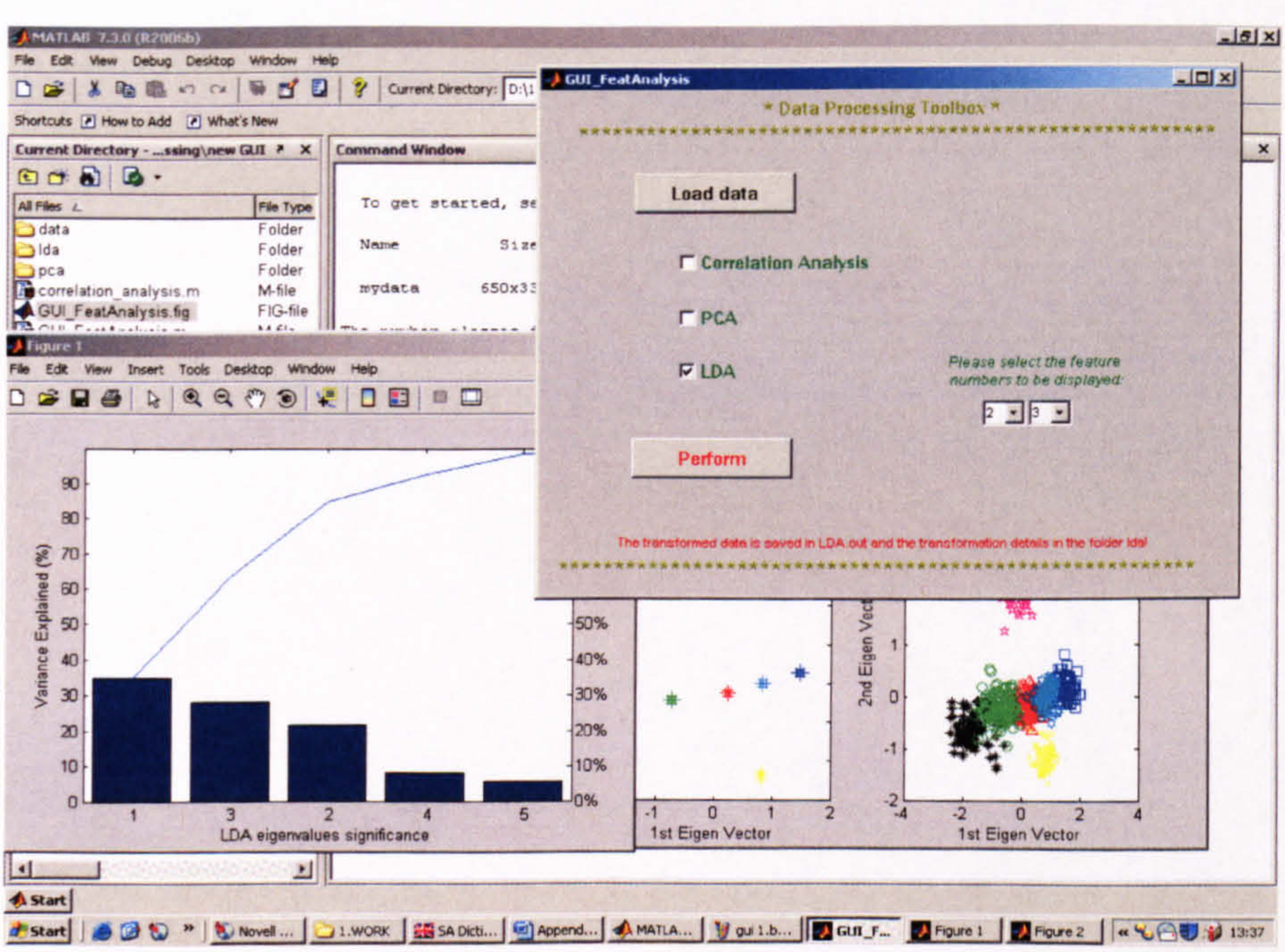
```
A3 = transpose(a)*c;
A4 = transpose(a)*d;
A5 = transpose(a)*e;
A6 = transpose(b)*a;
A7 = transpose(b)*b;
A8 = transpose(b)*c;
A9 = transpose(b)*d;
A10 = transpose(b)*e;
A11 = transpose(c)*a;
A12 = transpose(c)*b;
A13 = transpose(c)*c;
A14 = transpose(c)*d;
A15 = transpose(c)*e;
    A16 = transpose(d)*a;
    A17 = transpose(d)*b;
    A18 = transpose(d)*c;
    A19 = transpose(d)*d;
    A20 = transpose(d)*e;
    A21 = transpose(e)*a;
    A22 = transpose(e)*b;
    A23 = transpose(e)*c;
    A24 = transpose(e)*d;
    A25 = transpose(e)*e;
     % Storing them in one matrix for easy access.
    A = [A1, A2, A3, A4, A5, A6, A7, A8, A9, A10, A11, A12, A13, A14, A15, A16, A17, A18, A19,
A20, A21, A22, A23, A24, A25];


    function f = conv_all(I, A, E)
    % I image to be processed
% I is convolved subsequently with the 25 masks that are stored in the array
% A and have size ExE.
B = [];
for i = 1:E:E*E*E
   B = [B, conv2(double(I),double(A(1:E,i:i+E-1)))];
end
f = B;

function f = energy(I)
global N M
L = 7;% map size i 2*L+1

N = length(I(:,1));
M = length(I(1,:));
 for i = 1:1:N
   for j = 1:1:M
      e =  sum(sum(abs(I(max(1,i-L):min(i+L,N),max(j-L,1):min(j+L,M)))));
      Ibuf(i,j) = e; % Ibuf contains the energy map
   end
end
% f is indeed the 'power' function of the energy map Ibuf. For more details
% refer to Umbaugh, 2005 (page 272).
f = mean(mean(Ibuf.^2));
```

Feature analysis

The conducted major data processing is demonstrated here. A small Graphical User Interface (GUI) is developed in Matlab (Fig. CI). It handles correlation analysis, Principal Component Analysis (PCA), and Linear Discriminant Analysis (LDA). Part of the corresponding source is given below:



(a) Before the data is loaded.



(b) Linear Discriminant Analysis

Figure CI. GUI for the Data Analysis stage – processing and visualisation.

- GUI function (only the loading of data is shown)

```
function pushbutton1_Callback(hObject, eventdata, handles)
% hObject    handle to pushbutton1 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
handles = guihandles(hObject);
flag = 0;
val1 = get(handles.checkbox1,'Value');
val2 = get(handles.checkbox2,'Value');
val3 = get(handles.checkbox3,'Value');
    f1 = get(handles.popupmenu1,'Value');
    f2 = get(handles.popupmenu2,'Value');

    if (val1 == 1)
       correlation_analysis();
       flag = 1;
    end
    if (val2 == 1)
       myPCA(f1,f2);
       flag = 1;
       set(handles.text7,'String','The transformed data is saved in PCA.out and the transformation details
in the folder pca!');
    end
    if (val3 == 1)
       myLDA(f1,f2);
       flag = 1;
       set(handles.text7,'String','The transformed data is saved in LDA.out and the transformation
details in the folder lda!');
    end
    if (flag == 0)
       set(handles.text7,'String','Did you forget to choose?');
       menupop(hObject, eventdata, handles);
    end
```

Loading the data GUI extracts:

```
  function flag = load_data()
 global mydata N Nin C l
% The data files needs to be a matrix and the the samples are presented as
% rows and the features as columns.
 n = menu('Data is assumed to have samples in the rows and features in the columns! Make sure you
have filled in the corresponding data files!','OK','Cancel');
       if n ==1
          m = menu('Are labels included in the data? If yes they need to be in the last column and be
integers','Yes','No');
          if m ==1
             labels = 1;
          elseif m == 2
             labels = 0;
          end
          mydata = load('data/data.m');
          N = load('data/num_samples.m'); % this is a file that contains the number of samples from each
class.
          %It is assumed that the data is ordered, e.g. first n samples are all from
          %class 1, the following m samples are from class 2, etc.
```

```
%N = [124, 92, 124, 72, 82, 84, 72];
 C = length (N);% number of classes
Nin = [N(1)]; % this is a working variable that is needed in the visualisation
for i = 2:1:C
   Nin = [Nin, N(i)+Nin(i-1)];
end
 %Nin = [124, 216, 340, 412, 494, 578, 650];
if labels == 1
   l = length(mydata(1,:))-1; % number of features by default is read from the data
   % if the labels are incluced in the last column we extract one
   mydata = mydata(:,1:l);
else
   l = length(mydata(1,:));
      end
      whos mydata
      disp('The number classes is')
      C
      disp('The number of samples in each class is:')
      N
      flag = 1;
      Nin;
 elseif n == 2
    disp('Action cancelled!');
    flag = 0;
end
```

- PCA analysis and visualisation

```
function myPCA(f1,f2)
global mydata N Nin C l
%N - array with the sample numbers of each class
%Nin just a working variable
%C - number of classes
%l - number of features (excl. labels)
 % the original features considered here for the plots
 % normalization of data - each column becomes with zero mean and unit variance
[as,meana,stda] = prestd(mydata');
 % Performing PCA
[new_data,transMat] = prepca(as,0);
[M,Q] = size(new_data);
new_data = new_data';
% save the transformation data, so it could be used in future
save pca/meana.m meana -ascii -tabs
save pca/stda.m stda -ascii -tabs
save pca/myPCA.trans transMat -ascii -tabs
whos transMat
whos new_data
% the cluster diagrams of the classes according to principal components 1
% and 2
 hpca = figure;
subplot(1,2,1)
col = 'red'; mark = '^';
plotclust(new_data(1:Nin(1),f1),new_data(1:Nin(1),f2),mark,col)
col = 'black'; mark = '*';
plotclust(new_data(Nin(1)+1:Nin(2),f1),new_data(Nin(1)+1:Nin(2),f2),mark,col)
col = 'green'; mark = 'o';
plotclust(new_data(Nin(2)+1:Nin(3),f1),new_data(Nin(2)+1:Nin(3),f2),mark,col)
col = 'blue'; mark = 's';
plotclust(new_data(Nin(3)+1:Nin(4),f1),new_data(Nin(3)+1:Nin(4),f2),mark,col)
```

```
 col = 'yellow'; mark = 'x';
plotclust(new_data(Nin(4)+1:Nin(5),f1),new_data(Nin(4)+1:Nin(5),f2),mark,col)
 col = 'magenta'; mark = 'p';
plotclust(new_data(Nin(5)+1:Nin(6),f1),new_data(Nin(5)+1:Nin(6),f2),mark,col)
 col = 'cyan'; mark = 'h';
plotclust(new_data(Nin(6)+1:Nin(7),f1),new_data(Nin(6)+1:Nin(7),f2),mark,col)
xlabel('1st Principal Component');
ylabel('2nd Principal Component');
% the data itself
subplot(1,2,2)
col = 'red'; mark = '^';
plot(new_data(1:Nin(1),f1),new_data(1:Nin(1),f2),mark,'color',col,'MarkerSize',4)
hold on
col = 'black'; mark = '*';
plot(new_data(Nin(1)+1:Nin(2),f1),new_data(Nin(1)+1:Nin(2),f2),mark,'color',col,'MarkerSize',4)
hold on
col = 'green'; mark = 'o';
plot(new_data(Nin(2)+1:Nin(3),f1),new_data(Nin(2)+1:Nin(3),f2),mark,'color',col,'MarkerSize',4)
hold on
col = 'blue'; mark = 's';
plot(new_data(Nin(3)+1:Nin(4),f1),new_data(Nin(3)+1:Nin(4),f2),mark,'color',col,'MarkerSize',4)
hold on
col = 'yellow'; mark = 'x';
plot(new_data(Nin(4)+1:Nin(5),f1),new_data(Nin(4)+1:Nin(5),f2),mark,'color',col,'MarkerSize',4)
hold on
col = 'magenta'; mark = 'p';
plot(new_data(Nin(5)+1:Nin(6),f1),new_data(Nin(5)+1:Nin(6),f2),mark,'color',col,'MarkerSize',4)
hold on
col = 'cyan'; mark = 'h';
plot(new_data(Nin(6)+1:Nin(7),f1),new_data(Nin(6)+1:Nin(7),f2),mark,'color',col,'MarkerSize',4)
  xlabel('1st Principal Component');
ylabel('2nd Principal Component');
 % On a new figure a pareto diagram with the variance in the principal
% components is displayed
figure
varplot(new_data,'principal components')
 save PCA.out new_data -ascii -tabs
```

- LDA analysis and visualisation

```
function myLDA(f1,f2)
% this function follows the notationa in Dillon and Goldstein,
% "Multivariate Analysis", 1984, p. 400
 global mydata N Nin C l
%N - array with the sample numbers of each class
%Nin just a working variable, Note here it is modified and then returned to normal
Nin = [0,Nin];
 %C - number of classes
%l - number of features (excl. labels)
 P = 650; % number of all samples
 m = mean(mydata); % the global mean vector
a2 = mydata - repmat(m,P,1);
T = zeros(l);
for i=1:1:P
   t = a2(i,:);
   T = T+ t'*t;
end
% compute the within-class scatter matrix
W = zeros(l);
```

```
for i=1:1:C
   if (Nin(i)+1-Nin(i+1) ~= 0)
      m = mean(mydata(Nin(i)+1:Nin(i+1),:)); % the mean vector for the i-th class
%   elseif (Nin(i)+1-Nin(i+1) == 0)
 %      m = mydata(Nin(i)+1,:);
   end
 for j = 1:1:N(i)
      t = mydata(Nin(i)+j,:)-m;
      W = W + t'*t;
   end
end
 Nin = Nin(2:C+1);
 % compute the between-class scatter matrix
B = T - W;
% [V,D] = eig(A,B) produces a diagonal matrix D of generalized eigenvalues
% and a full matrix V whose columns are the corresponding eigenvectors so
% that A*V = B*V*D.
%rank(B)
%rank(W)
[V, D] = eig(B,W,'qz');
q = sum(D);
s = sum(sum(D));
sum(D)/s*100;
 figure
pareto(sum(D)/s*100);
xlabel('LDA eigenvalues significance')
ylabel('Variance Explained (%)')

V=V(:,1:C-1);
c = mydata*V;
[as,meana,stda] = prestd(c');
as = as';
save lda/meana.m meana -ascii -tabs
save lda/stda.m stda -ascii -tabs
save lda/myLDA.trans  V -ascii -tabs
 save LDA.out as -ascii -tabs
 data = as;
% Number of samples in each class
 % the original features considered here for the plots
 % the cluster diagrams
hlda = figure;
subplot(1,2,1)
temp = 1;
 col = 'red'; mark = '^';
plotclust(data(1:Nin(1),f1),data(1:Nin(1),f2),mark,col)
 if C >= 2
   col = 'black'; mark = '*';
   plotclust(data(Nin(1)+1:Nin(2),f1),data(Nin(1)+1:Nin(2),f2),mark,col)
end
if C >= 3
   col = 'green'; mark = 'o';
   plotclust(data(Nin(2)+1:Nin(3),f1),data(Nin(2)+1:Nin(3),f2),mark,col)
end
if C >= 4
   col = 'blue'; mark = 's';
   plotclust(data(Nin(3)+1:Nin(4),f1),data(Nin(3)+1:Nin(4),f2),mark,col)
end
if C >= 5
   col = 'yellow'; mark = 'x';
   plotclust(data(Nin(4)+1:Nin(5),f1),data(Nin(4)+1:Nin(5),f2),mark,col)
```

```
end
if C >= 6
   col = 'magenta'; mark = 'p';
   plotclust(data(Nin(5)+1:Nin(6),f1),data(Nin(5)+1:Nin(6),f2),mark,col)
end
if C >=7
   col = 'cyan'; mark = 'h';
   plotclust(data(Nin(6)+1:Nin(7),f1),data(Nin(6)+1:Nin(7),f2),mark,col)
else
   disp ('We are sorry but we do not support more than seven classes')
end
xlabel('1st Eigen Vector');
ylabel('2nd Eigen Vector');
% the data itself
subplot(1,2,2)
col = 'red'; mark = '^';
plot(data(1:Nin(1),f1),data(1:Nin(1),f2),mark,'color',col)
if C >= 2
   hold on
   col = 'black'; mark = '*';
   plot(data(Nin(1)+1:Nin(2),f1),data(Nin(1)+1:Nin(2),f2),mark,'color',col)
end
if C >= 3
   hold on
   col = 'green'; mark = 'o';
   plot(data(Nin(2)+1:Nin(3),f1),data(Nin(2)+1:Nin(3),f2),mark,'color',col)
end
if C >= 4
   hold on
   col = 'blue'; mark = 's';
   plot(data(Nin(3)+1:Nin(4),f1),data(Nin(3)+1:Nin(4),f2),mark,'color',col)
end
if C >= 5
   hold on
   col = 'yellow'; mark = 'x';
   plot(data(Nin(4)+1:Nin(5),f1),data(Nin(4)+1:Nin(5),f2),mark,'color',col)
end
if C >= 6
   hold on
   col = 'magenta'; mark = 'p';
   plot(data(Nin(5)+1:Nin(6),f1),data(Nin(5)+1:Nin(6),f2),mark,'color',col)
end
if C >= 7
   hold on
   col = 'cyan'; mark = 'h';
   plot(data(Nin(6)+1:Nin(7),f1),data(Nin(6)+1:Nin(7),f2),mark,'color',col)
else
   disp ('We are sorry but we do not support more than seven classes')
end
xlabel('1st Eigen Vector');
ylabel('2nd Eigen Vector');
```

- Plotting functions

```
% plots the variances in each column in a pareto chart, X is a amtrix
%variable is a string that contains description of the variables considered
function varplot(X, variable)
 M = length(X(1,:));
 variances = [];
```

```
for i=1:1:M
   variances = [variances;var(X(:,i))];
end
percent_explained = 100*variances/sum(variances);
pareto(percent_explained);
xlabel(variable)
ylabel('Variance Explained (%)')
% this function plots one cluster with the mean and 95% confidence
% intervals
% the input are the two vectors, the mark we want to use for the mean
% and the colour.

 function plotclust(x,y,mark,col)
    tvalues = [12.706 4.303 3.182 2.776 2.571 2.447 2.365 2.306 2.262 2.228 2.201 2.179 2.16 2.145
2.131 2.12 2.11 2.101 2.093 2.086 2.08 2.074 2.069 2.064 2.06 2.056 2.052 2.048 2.045 2.042 2.04
2.037 2.035 2.032 2.03 2.028 2.026 2.024 2.023 2.021 2.02 2.018 2.017 2.015 2.014 2.013 2.012 2.011
2.01 2.009 2.008 2.007 2.006 2.005 2.004 2.003 2.002 2.002 2.001 2 2 1.999 1.998 1.998 1.997 1.997
1.996 1.995 1.995 1.994 1.994 1.993 1.993 1.993 1.992 1.992 1.991 1.991 1.99 1.99 1.99 1.989 1.989
1.989 1.988 1.988 1.988 1.987 1.987 1.987 1.986 1.986 1.986 1.986 1.985 1.985 1.985 1.984 1.984
1.984];
    m1 = mean(x);
    %std1 = std(x);
    L1 = length(x);
    if L1<100
       L2 = L1;
    else
       L2 = 100;
    end
    std1 = tvalues(L2-1)*std(x)/sqrt(L1);
    tvalues(L2-1);
    m2 = mean(y);
    %std2 = std (y);
    std2 = tvalues(L2-1)*std(y)/sqrt(L1);
    tvalues(L2-1);
     plot(m1,m2,mark,'color',col,'MarkerSize',4);
     x1 = [m1,m1];
    y1 = [m2-std2,m2+std2];
    hold on
    plot(x1,y1,'b:+','color',col)
    x1 = [m1-std1,m1+std1];
    y1 = [m2,m2];
    hold on
    plot(x1,y1,'b:+','color',col)
    hold on
```

D. STOCHSTIC GENETIC ALGORITHM

The code for our implementation of the Stochastic GA (Tu and Lu, 2004) in Matlab is given below:

- The main functions

```
clear all
z = load('parameters.m');
a_min = z(9,:);
```

```
mean_f = [];
succ_rate = 0;
for i = 1:1:20
   mean_f(i)  = stGA_RunMeFile
   if abs(mean_f(i)-a_min)<0.01
      succ_rate = succ_rate+1;
   end
end
succ_rate
mean_error_f = mean(mean_f)
std_f = std(mean_f)

min(mean_f)
max(mean_f)



function answ = stGA_RunMeFile

% StGA method; ref: "A robust stochastic genetic algorithm (StGA) fpr
% global numerical optimisation", IEEE Trans. Evolutionary Computation,
% Vol. 8 (5), 2004.
% Load parameters values from file 'parameters.m'
z = load('parameters.m');
D = z(1,:);  % dimensionality of the problem
Np = z(2,:); % population size NP
Ns = z(3,:); % Number of points in each region during local selection
Tn = z(4,:); % Selection pressure in the Tournament selection Tn
Pcr = z(5,:);
Replacement = z(6,:);
max_iter = z(7,:);
Pm = z(8,:);
B = z(10,:);
Bl = load('f_low.m');
Bu = load('f_up.m');
f_eval = 0;
gene_length = B*D;
 %Initialization
rand('state',sum(100*clock));
 % chromosomes
for i = 1:1:Np
   for j = 1:1:D
      M(i,j) = Bl(j)+(Bu(j)-Bl(j))*rand;
   end
   f_M(i) = f7(M(i,:));
   f_eval = f_eval +1;
end
%f_M = f_M
% variations
for i = 1:1:D
   Rl(i) = 0.0083*(Bu(j)-Bl(j));
   Ru(i) = 0.0125*(Bu(j)-Bl(j));
end

for k = 1:1:Np
   for i= 1:1:D
      V(k,i) = Rl(i) + rand*(Ru(i)-Rl(i));
   end
end
%initial_variation=V
```

```
% delta - step for increasing/decreasing the variations
for i =1:1:D
   delta_l(i) = 0.02*V(1,i);
   delta_u(i) = 0.05*V(1,i);
   delta(i) = delta_l(i)+(delta_u(i)-delta_l(i))*rand;
end
 count = 0;
while (count < max_iter)
   count = count + 1;

%Local Selection
   randn('state',sum(100*clock));
   for k = 1:1:Np
      for i = 1:1:Ns
         for j = 1:1:D
            ch_t(i,j) = M(k,j)+randn*V(k,j);
         end
         f_ch_t(i) = f7(ch_t(i,:));
         f_eval = f_eval +1;
      end
%choose a representative
      [f_ch, ch] = order(f_ch_t,ch_t, Ns);
      if f_ch(1)<f_M(k)
         f_M(k) = f_ch(1);
         for j = 1:1:D
            M(k,j) = ch(1,j);
 %decrease the variation Vk for each coordinate j
            V(k,j) = V(k,j)-V(k,j)*delta(j);
         end
      else
         for j = 1:1:D
 %increase the variation Vk for each coordinate j
            V(k,j) = V(k,j)+V(k,j)*delta(j);
         end
      end
   end
%f_M = f_M
%Global Selection - tournament
   rand('state',sum(100*clock));
   mate_index = [];

   i = 1;
   while i <= Np
      temp2 = [];% keeps the finess values of the members
      temp3 = [];% keeps the indeces of the members
      for j=1:1:Tn
         a = ceil(Np.*rand(1,1));% random member of the population is withdrawn
         temp2(j) = f_M(a);
         temp3(j) = a;
      end
      if (rand<Pcr)
         i = i+1;
         mate_index = [mate_index, temp3(findmin(temp2))];
      end
   end
 %  mate_index
ch = [];
%Encoding
   coded = [];
   for i=1:1:length(mate_index)
```

```
      coded = [coded,bin2(M(mate_index(i),:))];
   end

% individually
%Crossover
   ch_f=[];
   for i = 1:2:length(mate_index)-1
      rand('state',sum(100*clock));
      cr = ceil((gene_length-1)*rand(1,1));%cutting point cr in the gene is randomly selected
% check if the cutting point is between different coordinates or in a
% coordinate

      temp1 = str2mat(coded(i));
%       decode(temp1)
      temp2 = str2mat(coded(i+1));
 %      decode(temp2)
      ch_t1 = [];
      ch_t2 = [];
      tt = cr/B;
      if (rem(cr,B) == 0)
% cross-over is between coordinates
         for j = 1:1:cr
            ch_t1 = [ch_t1, temp1(j)];
            ch_t2 = [ch_t2, temp2(j)];
         end

         for j = 1:1:tt
            Vch(i,j) = V(mate_index(i),j);
            Vch(i+1,j)= V(mate_index(i+1),j);
         end
      for j = cr+1:1:gene_length
         ch_t1 = [ch_t1, temp2(j)];
         ch_t2 = [ch_t2, temp1(j)];
      end
      for j = tt+1:1:D
         Vch(i,j) = V(mate_index(i+1),j);
         Vch(i+1,j)= V(mate_index(i),j);
      end
   else
      fl = floor(tt);
      for j = 1:1:cr
         ch_t1 = [ch_t1, temp1(j)];
         ch_t2 = [ch_t2, temp2(j)];
      end
      for j = cr+1:1:gene_length
         ch_t1 = [ch_t1, temp2(j)];
         ch_t2 = [ch_t2, temp1(j)];
   end
   for j = 1:1:fl
      Vch(i,j) = V(mate_index(i),j);
      Vch(i+1,j)= V(mate_index(i+1),j);
   end
   for j = fl+2:1:D
      Vch(i,j) = V(mate_index(i),j);
      Vch(i+1,j)= V(mate_index(i+1),j);
   end
   rand('state',sum(100*clock));
   r = rand(1);
   V(i, fl+1) = r*V(i,fl+1)+(1-r)*V(i+1,fl+1);
   V(i+1, fl+1) = r*V(i+1,fl+1)+(1-r)*V(i,fl+1) ;
```

```
      end

   %   chhh1 = decode(ch_t1)
    %   chhh2 = decode(ch_t2)
%Mutation
      for j = 1:B:gene_length
         for k = j:1:B
            r = rand(1);
            if (r < Pm)
               ch_t1(k) = mut(ch_t1(k));
            end
            r = rand(1);
            if (r < Pm)
               ch_t2(k) = mut(ch_t2(k));
            end
         end
      end
   %   chhh1 = decode(ch_t1)
    %   chhh2 = decode(ch_t2)

% Decoding
      ch(i,:) = decode(ch_t1);
      ch(i+1,:) = decode(ch_t2);
% use decode(coded(i)) to get each vector that has D coordinates
% actually is decode('21*D 1 or 0')
% Fitness value of children
      ch_f = [ch_f, f7(ch(i,:)),f7(ch(i+1,:))];
      f_eval = f_eval +2;
   end
  % children = ch_f

 %Replacement
 [o_f_ch, o_ch] = order(ch_f,ch, length(ch_f));
 %oordered_children = o_f_ch
 y = round(Replacement*Np);
 [o_f_p, o_p] = order(f_M,M,Np);
% ordered_parents = o_f_p
 for i = 1:1:y
    o_f_ch(Np-y+i) = o_f_p(i);
    for j=1:1:D
          o_ch(Np-y+i,j) = o_p(i,j);
    end
 end
 %orderdChildrPlusRepl = o_f_ch
 [o_f_ch, o_ch] = order(o_f_ch,o_ch,Np);
 %orderdNewGeneration = o_f_ch
    o_f_ch(1);
    o_ch(1,:);
    f_M = o_f_ch;
    M = o_ch;
%    count
 end
 %f_eval
 answ = o_f_ch(1);

% parameters for the minimization with StGA.m

% Dimensionality of problem D =
30;
% Number of population vectors Np =
```

```
30;% Should be bigger than 5 since otherwise there is no replacement
% Number of local selection children Ns =
5;
% Selection pressure in the Tournament selection Tn =
2;
% Probability for cross-over Pcr =
0.85;
% Replacement rate =
0.1; % Note that if Np is less than 5, there will not be replacement unless the rate is lowered.
% Maximal number of iterations max_iter =
170;
% Probability for mutation Pm =
0.02;
% Analitic minimum a_min =
0;
% flexible bit string length, total number of bits B (1 for the sign and
% (B-1)/2 for each the whole and fractional parts. If (B-1) is odd, than
% B/2 for the whole and (B-2)/2 for the fractional
8;
```

- The coding and de-coding functions

```
function [w,d] = w_d_parts(a)
b = a - round(a);
if (b<0)
   w = floor(abs(a));
   d = (1-abs(b));
else
   if (b == 0)
      w = abs(a);
      d = 0;
   else
      if (b>0)
         w = round(a);
         d = b;
      end
   end
end


%returns the binary representation of the fractional part of a real number
function s = d_dec2bin(x,d_length)
a = '';
re = [];

if floor(x)>1
   a = '!!!there is a problem in d_dec2bin: values bigger than 1 has been sent!!!'
   re = 'NAN';
end

if x == 0
   for i=1:1:d_length
      re = [re,'0'];
   end
else
   for i=1:1:d_length
      b = x*2;
      if floor(b) == 1
         re = [re,'1'];
```

```
        x = b-1;
      else
        re = [re,'0'];
        x = b;
      end
    end
end
s = re;

function a = d_bin2dec(x,d_length)
temp = 0;
for i = 1:1:d_length
   if (x(i)=='1')
      temp = temp +2^-i;
   end
end
a = temp;

 % x is a B*D gene we need to know where the sign is and where the whole
% part finishes
function w = decode(x)
a=[];
z = load('parameters.m');
D = z(1,:);
B = z(10,:);
k=1;
gene_length = B*D;
if rem(B,2) ~= 0
   w_length = (B-1)/2;
   d_length = (B-1)/2;
else
   w_length = B/2;
   d_length = (B-2)/2;
end
% even if B is bigger we don't want to have more than 5 bits for the
% fractional part, so we use them for the whole part
if B>=13
   w_length = w_length + d_length -5;
   d_length = 5;
end
while k < gene_length
   %decode each coordinate;
   if x(k) == '1'
      temp = 1;
   else
      if x(k) == '0'
         temp = -1;
      end
   end
y = [];
for i = k+1:1:k+w_length
   y = [y,x(i)];
end
temp2 = bin2dec(y);
y = [];
for i = k+w_length+1:1:k+B-1
   y = [y,x(i)];
end
temp3 = d_bin2dec(y,d_length);
k = k+B;
```

```
a = [a, temp*(temp2+temp3)];
end
w = a;

 % x is a real number and returns B charcater string z
function z = conv(x)

z = load('parameters.m');
B = z(10,:);% B is the lenght of the binary string

if rem(B,2)~= 0 %if B is odd
   w_length = (B-1)/2;
   d_length = (B-1)/2;
else  %if B is even
   w_length = B/2;
   d_length = (B-2)/2;
end
% even if B is bigger we don't want to have more than 5 bits for the
% fractional part, so we use them for the whole part
if B>=13
   w_length = w_length + d_length -5;
   d_length = 5;
end
% finding the sign
% + is 1
% - is 0
%assign sign as the first bit in the array. If x is equal to zero, the
%whole array becomes full of zeroes
if (x == 0)
   z = '0';
      for i=1:1:B-1
         z = [z,'0'];
      end
   else
      if x>0
         z = '1';
      else
         z = '0';
      end
      [w,d] = w_d_parts(abs(x));% returns the whole and the fractional parts of the number
       temp = dec2bin(w,w_length);% converts the whole part to a binary string with at least w_length
bits
      if length(temp)>w_length
         temp = takefirst(temp,w_length);% we need only the first w_length ones
      end
      z = [z,temp,d_dec2bin(d,d_length)];
end

%z = '0';

% x is one point with D coordinates - here each coordinate is converted to
% a binary string and all of them are joined together to form one gene
function q = bin2(x)
z = load('parameters.m');
D = z(1,:);
a=[];
for i = 1:1:D
   a = [a,conv(x(i))];
end
q = cellstr(a);
```