Faculty of Electrical Engineering, Computer Science and Telecommunication
University of Zielona Góra

Monographs

Volume 3

Andrzej OBUCHOWICZ
Institute of Control and Computation Engineering
University of Zielona Góra
Ul. Podgórna 50
65-246 Zielona Góra, Poland
e-mail: A.OBUCHOWICZ@ISSI.UZ.ZGORA.PL

Referees:

- Roman GALAR, Wrocław University of Technology
- Robert SCHAEFER, Jagiellonian University in Cracow

Typesetting: Camera-ready copy from the author's LaTeX2_ε files
The cover design by Julita KŁOSIŃSKA

Financial support by the State Committee for Scientific Research
in Poland (KBN)

ISBN 83-88317-02-4

Printed and bound by University of Zielona Góra Press, Poland.

CONTENTS

Notation

Chapters 2–5

$f(\cdot)$	objective function
$\mathcal{U}$	space of solutions for the global optimization problem
$\mathcal{G}$	genotype space
$\mathcal{D}$	phenotype space
$\boldsymbol{a} \in \mathcal{G}$	genotype of an individual
$\boldsymbol{x} \in \mathcal{D}$	phenotype of an individual
$P(t)$	population in the epoch (iteration) t
η	population size
$\boldsymbol{a}_k^t$	k-th individual (element) of the population $P(t)$
$\zeta : \mathcal{D} \to \mathcal{G}$	coding function
$\xi : \mathcal{G} \to \mathcal{D}$	decoding function
$\Phi : \mathcal{D} \to \boldsymbol{R}$	fitness function
$\phi_k^t = \Phi(\xi(\boldsymbol{a}_k^t))$	fitness of the k-th individual of $P(t)$
$t_{\max}$	maximum number of epochs
$\langle\cdot\rangle$	expectation operator or the mean operator
$\|\cdot\|$	Euclidesian norm in $\boldsymbol{R}^n$
$U(0,1)$	uniformly distributed random number from the interval $[0,1)$
$N(m,\sigma)$	normally distributed random number with the expectation value m and the standard deviation σ
$\boldsymbol{N}(\boldsymbol{m},\mathbb{C})$	normally distributed random vector with the expectation vector $\boldsymbol{m}$ and the covariance matrix $\mathbb{C}$
$C(u,\tau)$	random value obtained according to the one-dimensional Cauchy distribution with parameters u and τ
$\mathcal{T}$	set of terms dedicated for the tree representation in the GP algorithm
$\mathcal{F}$	set of operators dedicated for the tree representation in the GP algorithm
$\mathrm{diag}(\boldsymbol{z})$	diagonal matrix with diagonal elements taken from the vector $\boldsymbol{z}$
t_c	average number of epochs which is needed to cross the saddle by a given algorithm taken over a given number of this algorithm runs

p_c	percentage of given algorithm runs in which the global optimum has been found in a given number of epochs $t_{\max}$
$\Phi_{best}(t)$	fitness of the best individual in the population at the time t
$\Phi_{opt}(t)$, $\boldsymbol{x}_{opt}(t)$	optimum fitness in the search space at the time t and its argument
$\Phi_{acpt}(t)$	minimum acceptable fitness at the time t

Chapter 6

NN	neural network
NA	architecture of the NN
$\mathcal{A}$	space of neural network architectures
$\boldsymbol{v}$	vector of NN parameters
$\mathcal{V}$	space of neural network parameters
$\boldsymbol{w}$	vector of weights
V_m	m-th layer of the NN
$s_m = \operatorname{card} V_m$	number of neurons in the m-th layer
$M+1$	number of layers (indexed from 0 to M)
$\mathcal{E}$	set of connections between neurons in the NN
$\boldsymbol{a}_s^m$, $\boldsymbol{b}_s^m$	feedback and feedforward path weights of the IIR of the s-th DNM unit in the m-th layer
λ_s^m	slope parameter of the activation function of the s-th DNM unit in the m-th layer
o_s^m	IIR dynamic order of the s-th DNM unit in the m-th layer
$\boldsymbol{u}$	input vector
$\boldsymbol{y}_{NA,\boldsymbol{v}}$	output vector of the NN
$\boldsymbol{y}$	desired output vector
Φ_L	set of learning patterns (pairs $(\boldsymbol{u}, \boldsymbol{y})$)
Φ_T	set of testing patterns
$J_T\left(\boldsymbol{y}_{NA,\boldsymbol{v}}, \boldsymbol{y}\right)$	testing cost function
$J_L\left(\boldsymbol{y}_{NA,\boldsymbol{v}}, \boldsymbol{y}\right)$	learning cost function
$\mathcal{S}(\boldsymbol{c}, \boldsymbol{r})$	spherical surface with the center in $\boldsymbol{c}$ and the radius $\boldsymbol{r}$
$\mathcal{K}(\boldsymbol{c}, \boldsymbol{r})$	sphere with the center in $\boldsymbol{c}$ and the radius $\boldsymbol{r}$
κ	compression index

$\eta(NA)$	number of free parameters of the network with the architecture NA
$G(\mathcal{A})$	graph of the DMLP architectures
$\Gamma(NA)$	expansion operator which creates all successors of the NA in $G(\mathcal{A})$

Chapter 7

$\boldsymbol{x}_k, \hat{\boldsymbol{x}}_k \in \mathbb{R}^n$	state vector and its estimate
$\boldsymbol{y}_k, \hat{\boldsymbol{y}}_k \in \mathbb{R}^m$	output vector and its estimate
$\varepsilon_k \in \mathbb{R}^m$	output error
$\boldsymbol{u}_k \in \mathbb{R}^r$	input vector
$\boldsymbol{d}_k \in \mathbb{R}^q$	unknown input vector ($q \leq m$)
$\mathcal{E}_k$	unknown input distribution matrix
$\boldsymbol{w}_k, \boldsymbol{v}_k$	process and measurement noises
$\mathcal{Q}_k, \mathcal{R}_k$	covariance matrices of $\boldsymbol{w}_k$ and $\boldsymbol{v}_k$
$\boldsymbol{p}$	parameter vector
$\boldsymbol{f}_k \in \mathbb{R}^s$	fault vector
$\mathcal{L}_{1,k}, \mathcal{L}_{2,k}$	fault distribution matrices
$f(\cdot), g(\cdot), h(\cdot)$	non-linear functions
$\mathcal{K}$	gain matrix

Abbreviations

EA	Evolutionary Algorithm
GA	Genetic Algorithm;
SGA	Simple GA;
GP	Genetic Programming algorithm;
EP	Evolutionary Programming algorithm;
ES	Evolutionary Strategy algorithm;
ESSS	Evolutionary Search with Soft Selection algorithm;
ESSS-SVA	ESSS algorithm with Simple Variance Adaptation;
ESSS-VPS	ESSS algorithm with Varying Population Size;
ESSS-FDM	ESSS algorithm with Forced Direction of Mutation;
ESSS-LS	ESSS algorithm with Local Selection;
ESSS-MS	ESSS algorithm with Mixed Selection;
ESSS-ALS	ESSS algorithm with Adaptive radius of Local Selection;
ESSS-IP	ESSS algorithm with Impatience and Polarization;
ESSS-DOF	ESSS algorithm with Deterioration (erosion) of the Objective Function;
ESSS-G	ESSS algorithm with the standard Gaussian mutation;
ESSS-GN	ESSS algorithm with the standard Gaussian mutation and the standard deviation $\sigma/\sqrt{n}$;
ESSS-GO	ESSS algorithm with the modified Gaussian mutation;
ESSS-C	ESSS algorithm with the non-spherical Cauchy mutation;
ESSS-CO	ESSS algorithm with the spherical Cauchy mutation;
CEP	EP algorithm with the standard Gaussian mutation;
CEPS	EP algorithm with the modified Gaussian mutation;
FEP	EP algorithm with the non-spherical Cauchy mutation;
FEPS	EP algorithm with the spherical Cauchy mutation;
ANN	Artificial Neural Network;
MLP	Multilayer Perceptron;
DNM	Dynamic Neural Model;
IIR	Infinite Impulse Responce filter;
DMLP	Dynamic MLP;
BP	Back-Propagation algorithm;
EDBP	Extended Dynamic BP algorithm;

NARMAX	Nonlinear Auto Regressive Moving Average with eXogenous inputs;
GESA	Genetic-Evolutionary Search Algorithm;
SA	Simulated Annealing algorithm;
FDI	Fault Detection and Isolation system;
ARS	Adaptive random Search algorithm;
EKF	Extended Kalman Filter;
UIO	Unknown Input Observer;
EUIO	Extended UIO.

INTRODUCTION

The optimization problem, in general, can be formulated as follows:

$$\boldsymbol{x}^\star = \arg\max_{\boldsymbol{x}\in\mathcal{U}} f(\boldsymbol{x})\big|\big(c_i(\boldsymbol{x}) \leq 0, i = 1,\ldots,m,\big), \tag{0.1}$$

where $\boldsymbol{x}^\star$ is searched optimum solution of an objective function $f(\boldsymbol{x})$, $c_i(\boldsymbol{x})$ denotes an i^{th} constraint and $\mathcal{U}$ is a space of solutions.

Most of conventional optimization methods are based on, so called, *hard selection*, where new base points for further exploration are generated basing on the best obtained points. Such a strategy usually ends trapped in the local optimum, and there is almost no chance to leave this area and achieve better results, so the global optimization ability is strongly limited. There are many algorithms proposed in the literature which try to overcome this problem. Most of them can be assigned to two classes of algorithms: enumerative methods and stochastic methods.

First of them is dedicated to discrete and finite sets of possible solutions, and usually consists in examination of all solutions in order to choose the best one. This is, of course, an inefficient technique, especially in the case of large systems, where a full review is impossible to do in a reasonable time. Thus, methods of heuristic search can be used (c.f. (Schalkoff 1990)). Their efficacy, when applied to particular problems, is often highly dependent on the way they exploit the domain – specific knowledge since in and of themselves they are unable to overcome the combinatorial explosion to which search processes are so vulnerable.

Stochastic optimization methods are mainly applied to searching for the global optimum of the multi-modal and multi-dimensional objective functions, for which the derivative is difficult or impossible to compute. The simple computer implementation is also an advantage of these methods, however, they are time-consuming. Most of the stochastic methods are composed of two parts: global and local one. First of them determines starting points for the local search. Usually these points are randomly chosen from the domain. The local optimization is carried out applying classical gradient methods or stochastic methods, which randomly modifies current points usually using the normal distribution. Among stochastic methods one can list several classes of these techniques, e.g., the pure random search (Monte Carlo methods), multiple random start, clustering methods, random direction methods, search concentration methods (c.f. (Birge and Louveaux 1997, Zieliński and Neumann 1983)). The main disadvantage of stochastic methods lies on their chaotic manner, which does not take into consideration information contained in previously evaluated points.

An alternative way for global optimum finding are Evolutionary Algorithms (EAs). The evolution is the natural way of development. Species acquire their properties and abilities by the natural selection, seemingly a blind process, which allows mainly well fitted individuals to survive and procreate. This mechanism allows to transfer the profitable features to next generations, thus, we have some kind of "intelligent" selection. But, the nature does not restrict itself to select only the best individuals in the population. Weakly fitted individuals have a chance to introduce their offspring to the next generation, too. Their descendants are often gifted with attributes unknown in the current population, and which can be useful in the future. Therefore, it is luring to introduce to optimization techniques the *soft selection* rule instead of the *hard* one, i.e. there is a possibility of choosing worse points as base points for further search. It occurs that the soft selection accelerates the probability of escaping from a local optimum trap.

The soft selection is the base rule in the EAs, the extremely effective technique of the computation intelligence systems applied to the global optimization. A very rich bibliography (c.f. (Angeline and Kinnear 1996, Arabas 2001, Bäck 1995, Bäck *et al.* 1997, Dasgupta and Michalewicz 1997, Davis 1987, Fogel 1995, Fogel 1998, Galar 1990, Goldberg 1989, Holland 1992, Michalewicz 1996, Mitchel 1996, Osyczka 2002, Schwefel 1995) proves this mimicked search process of natural evolution is a very robust and effective direct algorithm of the global optimization or, rather, adaptation.

The aim of this monograph is to present selected basic properties of evolutionary algorithms in the global optimization and chosen applications in the neural networks design problem and the fault diagnosis of industrial processes.

The book is parted into six main chapters which are preceded by introduction and ended by conclusions and two appendices.

Chapter 1 contains a description of the general outline of the evolutionary algorithm, presents the "classical" forms of the most known EA representatives: Genetic Algorithms (GA), Genetic Programming (GP), Evolutionary Programming (EP), and Evolutionary Strategies (ES). Emphasis is put on the Evolutionary Search with Soft Selection (ESSS) algorithm, which is the subject of majority of research studies described in this book. The ESSS algorithm is based on probably the simplest selection–mutation model of Darwinian evolution. The n-dimensional real space is the search process domain, on which some non-negative fitness function is defined. At the beginning the population of points is selected from the domain and, next, it is iteratively transformed by the selection and mutation operations. As a selection operator the well–known proportional selection (roulette method) is chosen. Coordinates of selected parents are mutated by adding normally-distributed random values.

Next chapters contain results which are based on the author's research.

Techniques which accelerate the exploration abilities of the ESSS algorithm are the subject of Chapter 2. These techniques are parted into three classes: methods adapting the algorithm parameters, methods modifying evolutionary operators, and methods, which modify the objective function during the searching process. The first class contains algorithms which adapt the standard deviation of Gaussian mutation (ESSS with Simple Variance Adaptation, ESSS-SVA) and population size (ESSS with Varying Population Size, ESSS-VPS). The ESSS-SVA algorithm is based on a concept of an evolutionary trap. When population fluctuates around a local peak of an adaptation landscape, the standard deviation of the Gaussian mutation increases. This fact decreases the mean fitness of the population and facilitates the population escape towards a neighbouring peak. The idea of the ESSS-VPS algorithm is similar to the GAVaPS algorithm (Arabas *et al.* 1994), each individual contains additional parameter: a life time, which depends on the relation between the individual fitness and the mean fitness of the current population. The second class of techniques contains the ESSS algorithm with Forced Direction of Mutation (ESSS-FDM) and algorithms with local selections. The ESSS-FDM is distinct from other algorithms of the ESSS family that the expected vector of the Gaussian mutation is not equal to zero, its direction is parallel to the latest drift of population. In the case of the local selection, each individual competes with elements which are located a given radius away. Depending on a method of a surrounding radius determination three algorithms are defined: the ESSS with Local Selection (ESSS-LS), ESSS with Mixed Selection (ESSS-MS), and ESSS with Adaptive radius of Local Selection (ESSS-ALS). The third class of exploration techniques is represented by the ESSS algorithm with Impatience and Polarization mechanism (ESSS-IP), which has been proposed by Galar and Kopciuch (1999), and erosion mechanism (ESSS with Deterioration of the Objective Function, ESSS-DOF). The erosion mechanism consists in gradual deterioration of a currently occupied local peak of the adaptation landscape by the population. Parameters of the erosion factor depend on the current location and distribution of the population. This idea was firstly introduced by Beasley *et al.* (1993), but it has not been developed yet. The simulation analysis of effectiveness of considered exploration mechanisms ends the chapter.

Chapter 3 concerns the analysis of Gaussian and Cauchy mutation, their influence on the effectiveness of phenotypic evolutionary algorithms. The author distinguishes two effects: *surrounding effect* and *symmetry effect*, which affect the EAs efficacy. Modified versions of Cauchy and Gaussian mutations are proposed, implemented in the ESSS and EP algorithms, and compared using simulating experiments.

The evolutionary adaptation in non-stationary environments is the subject of Chapter 4. There is an attempt at classification of non-stationary adaptation tasks taking into account different criteria. As far as the intensity of changes is considered three types of environment changes are distinguished: adiabatic, indirect and turbulent changes. In the case of adiabatic changes, classical methods of local optimization are effective. Along with increasing intensity of changes, usefulness of evolutionary algorithms increases. In the case of turbulent changes, the evolutionary process could not keep up with a running global optimum location and searches an ascent of some function averaged over some time interval. The adaptation tasks in non-stationary environments can be also classified taking account of the problem specification, e.g. tracing processes, an optimization in a mega-epoch, keeping solutions on an acceptable level, and so on. Different problem specifications require different quality measures for applied adaptation algorithms. A short analysis of algorithms quality measures known from literature and proposed by the author is presented and illustrated.

The problem of neural models design is considered in Chapter 5. The Dynamic Multi–Layer Perceptron (DMLP), whose units are based on Ayoubi'es Dynamic Neural Model (DNM) and organized into the standard feedforward architecture, focuses the author's attention. The problem of a neural network design can be viewed as a pair of optimization tasks: a learning process and an architecture optimization. Both tasks have different nature and need different optimization methods. The learning process belongs to continuous optimization tasks, which is usually connected with nonlinear, multi-modal objective functions, especially in the case of the DMLP. Therefore, techniques based on the gradient-descent method, like the extended dynamic back-propagation learning, are ineffective. Evolutionary algorithms approaches, especially based on the ESSS-FDM algorithm, turn out very effective learning methods. The space of neural networks architectures is discrete and can be represented by an infinite digraph. However, there are many instances of evolutionary approaches to the optimal neural model architecture allocation in the literature, they are not so efficient as presented heuristic search methods: the $A^\star$ algorithm and Tabu Search technique.

Applications of evolutionary algorithms to the fault diagnosis systems design is discussed in Chapter 6. The diagnosis of industrial processes has been intensively studied by the research group of Institute of Control and Computation Engineering of University of Zielona Góra for the last ten years. This chapter is based on results of the author and co–workers' research. Designing the fault diagnosis systems for complex dynamic systems is usually connected with the lack of a mathematical model, or with the fact that such a model is unsatisfactory. Recently, artificial intelligence methods have attracted researches' attention. It is worth noticing that the process of designing fault diagnosis systems, using both analytical and artificial intelligence methods, can be reduced to a set of complex optimization problems. They are usually nonlinear, multimodal and, not so rarely, multi-objective. So, the conventional algorithms are insufficient to solve them. Evolutionary algorithms seem to be an attractive tool for searching an optimal solution. Although there are few applications of evolutionary algorithms to fault diagnosis systems, a discussion of existing solution is presented. The emphasis is put on genetic programming approaches to residual generation module of a Fault Detection and Isolation (FDI) system.

It is a pleasure to express my sincere thanks to a number of people. First of all, I am grateful to professor Józef Korbicz for his continuous support and advice. I wish to thank professors Roman Galar and Robert Schaefer for their active interest in my research and many stimulating suggestions, professor Dariusz Uciński, dr. Krzysztof Patan, dr. Krzysztof Trojanowski, and dr. Marcin Witczak for discussions and co–operation, which bears many joint publications.

I wish to express my special gratitude to my wife Beata for her continuous patience, understanding and support in hard times during preparing this book and over the common years.

Zielona Góra, December 2002 *Andrzej Obuchowicz*

Chapter 1

EVOLUTIONARY ALGORITHMS

Evolutionary algorithms (EAs) are a broad class of stochastic adaptation algorithms, inspired by biological evolution, the process that allows populations of organisms to adapt to their surrounding environment. The concept of evolution was introduced in XIX century by Charles Darwin and Johann Gregor Mendel and, complemented with further details, are still widely acknowledged as valid.

In 1859, Darwin published his theory of natural selection or survival of the fittest. The idea is: not every organism can be held, only those which can adapt and win the competition for food and shelter are able to survive. Almost at the same time (1865) Mendel published short monograph about experiments in plants hybridisation. He observed how traits of different parents are combined in an offspring by sexual reproduction. Darwinian evolutionary theory and Mendel investigations of heredity in plants became foundations of evolutionary search methods.

Structure and properties of evolutionary algorithms are discussed in several books (Angeline and Kinnear 1996, Bäck 1995, Bäck *et al.* 1997, Dasgupta and Michalewicz 1997, Davis 1987, Fogel 1995, Fogel 1998, Galar 1990, Goldberg 1989, Holland 1992, Michalewicz 1996, Mitchel 1996, Schwefel 1995). The articles concerned with evolutionary computation are published in many scientific journals. There are at least 20 international conferences closely connected with evolutionary methods. Due to a large number of available publications it is impossible to present all of plenty of different evolutionary algorithms and their components, where their authors tried to improve the algorithm efficiency in the case of given problem to be solved. In this chapter, the main components of evolutionary algorithms are reminded and different basic forms of them briefly discussed.

1.1. Basic concepts of evolutionary search

In nature, individuals in a population compete with each other for resource such as food, water and shelter. Also, members of the same species often compete to attract a mate. Those individuals which are most successful in surviving and attracting mates will have relatively larger numbers of offspring. Poorly performing individuals will produce few or even no offspring at all. This means that the information (genes), slightly mutated, from the highly adapted individuals will spread to an increasing number of individuals in each successive generation. In this way, species evolve to became more and more suited to their environment.

In order to describe a general outline of the evolutionary algorithm let us introduce few useful concepts and notations (Atmar 1992, Fogel 1999). An evolutionary algorithm is based on the collective learning process within a *population* $P(t) = \{\boldsymbol{a}_k \in \mathcal{G} \mid k = 1, 2, \ldots, \eta\}$ of η *individuals*, each of which represents a genotype (an underlying genetic coding), a search point in a, so called, *genotype space* $\mathcal{G}$. The environment delivers a quality information (fitness value) of the individual dependent on its phenotype (the manner of response contained in the behavior, physiology and the morphology of the organism). The fitness function $\Phi : \mathcal{D} \to \boldsymbol{R}$ is defined on a *phenotype space* $\mathcal{D}$. So, each individual can be viewed as a duality of its genotype and phenotype, and some decoding function, *epigenesis*, $\xi : \mathcal{G} \to \mathcal{D}' \subset \mathcal{D}$ is needed.

At the beginning, a population is arbitrary initialized and evaluated (Tab. 1.1). Next, the randomized processes of *reproduction*, *recombination*, *mutation* and *succession* are iteratively repeated until a given termination criterion $\iota : \mathcal{G}^\eta \to \{\text{true}, \text{false}\}$ are satisfied. *Reproduction*, called also *preselection*, $s^p_{\theta_p} : \mathcal{G}^\eta \to \mathcal{G}^{\eta'}$ is a randomized process (deterministic in some algorithms) of η' parents selection from η individuals of the current population. This process is controlled by a set θ_p of parameters. *Recombination* mechanism (omitted in some realization) $r_{\theta_r} : \mathcal{G}^{\eta'} \to \mathcal{G}^{\eta''}$, controlled by additional parameters θ_r, allows the mixing of parental information while passing it to their descendants. *Mutation* $m_{\theta_m} : \mathcal{G}^{\eta''} \to \mathcal{G}^{\eta''}$ introduces innovation into current descendants, θ_m is again a set of control parameters. *Succession*, also called *postselection* $s^n_{\theta_n} : \mathcal{G}^\eta \times \mathcal{G}^{\eta''} \to \mathcal{G}^\eta$ is applied to choose a new generation of individuals from parents and descendants.

1.2. Standard evolutionary algorithms

Despite similarities of various evolutionary computation techniques, there are also many differences between them. It is generally accepted that any evolutionary algorithm to solve a problem must have five basic components (Davis 1987, Michalewicz 1999):

- a representation of solutions to the problem,
- a way to create an initial population of solution,
- an evaluation function, rating solution in terms of their fitness,
- selection and variation operators that alter the composition of children during reproduction and mutation,
- values for the parameters (population size, probabilities of applying variation operators, etc.)

The duality of the genotype and the phenotype suggest two main approaches to simulated evolution dedicated to global optimization problems in $\boldsymbol{R}^n$: genotypic and phenotypic simulations (Fogel 1999). In genotypic simulations, attention is focused on genetic structures. The candidate solutions are described as being

Tab.1.1. The outline of an evolutionary algorithm

I. Initiation

A. Random generation

$P(0) = \{\boldsymbol{a}_k^0 \mid k = 1, 2, \ldots, \eta\}.$

B. Evaluation

$P(0) \rightarrow \Phi\big(P(0)\big) = \big\{\phi_k^0 = \Phi\big(\xi\big(\boldsymbol{a}_k^0\big)\big) \mid k = 1, 2, \ldots, \eta\big\}.$

C. $t = 1.$

II.Repeat:

A. Reproduction

$P'(t) = s_{\theta_p}^p\big(P(t)\big) = \big\{\boldsymbol{a'}_k^t \mid k = 1, 2, \ldots, \eta'\big\}.$

B. Recombination

$P''(t) = r_{\theta_r}\big(P'(t)\big) = \big\{\boldsymbol{a''}_k^t \mid k = 1, 2, \ldots, \eta''\big\}.$

C. Mutation

$P'''(t) = m_{\theta_m}\big(P''(t)\big) = \big\{\boldsymbol{a'''}_k^t \mid k = 1, 2, \ldots, \eta''\big\}.$

D. Evaluation

$P'''(t) \rightarrow \Phi\big(P'''(t)\big) = \big\{\phi_k^t = \Phi\big(\xi\big(\boldsymbol{a'''}_k^t\big)\big) \mid k = 1, 2, \ldots, \eta''\big\}.$

E. Succession

$P(t+1) = s_{\theta_n}^n\big(P(t) \cup P'''(t)\big) = \big\{\boldsymbol{a}_k^{t+1} \mid k = 1, 2, \ldots, \eta\big\}.$

F. $t = t + 1.$

Until $\big(\iota\big(P(t)\big) = \text{true}\big).$

---

analogous to chromosomes. All the searching process is provided in the genotype space $\mathcal{G}$. However, in order to calculate the individual fitness, its chromosome must be decoded to its phenotype. Two main streams of instances of such evolutionary algorithms, can nowadays be identified:

- Genetic Algorithms (GA) (De Jong 1975, Goldberg 1989, Grefenstette 1986, Holland 1975, Michalewicz 1996),
- Genetic Programming (GP) (Kinnear 1994, Koza 1992).

In the phenotypic simulations, attention is focused on the behaviors of the candidate solutions in a population. All searching operations, selection, reproduction and mutation, are constructed in the phenotype space $\mathcal{D}$. This type of simulations characterizes a strong behavioral link between a parent and its offspring. Nowadays, there are two main streams of instances of "phenotypic" evolutionary algorithms:

- Evolutionary Programming (EP) (Fogel *et al.* 1991, Fogel 1992, Fogel 1999, Fogel *et al.* 1966, Yao and Liu 1999),
- Evolutionary Strategies (ES) (Rechenberg 1965, Schwefel 1981).

In this book emphasis is put on an Evolutionary Search with Soft Selection algorithm (ESSS) (Galar 1985, Galar 1989, Galar 1990, Galar and Karcz-Dulęba 1994), which is some simplified version of the ES. Basic ideas of GA, GP, EP and ES algorithms are presented below. The ESSS algorithm is the subject of the next section.

1.2.1. Genetic algorithms

GAs are probably the best know evolutionary algorithms, receiving remarkable attention all over the world. The basic principles of GAs were first laid down rigorously by Holland (1975), and are well described in many texts (e.g. (Bäck and Schwefel 1993, Beasley *et al.* 1993a, Beasley *et al.* 1993b, Dasgupta and Michalewicz 1997, Davis 1987, Davis 1991, Goldberg 1989, Grefenstette 1986, Grefenstette 1990, Michalewicz 1996).

Previously proposed form of GAs (De Jong 1975, Holland 1975), called Simple GAs (SGAs) (Goldberg 1989) or canonical GAs (Bäck and Schwefel 1993), operate on binary strings of fixed-length l, i.e. the genotype space $\mathcal{G}$ is a l-dimensional Hamming cube $\mathcal{G} = \{0,1\}^l$. SGAs are a natural technique of solving discrete problems, especially in the case of finite cardinality of possible solutions. Such a problem can be transformed to a pseudo-boolean fitness function, where GAs can be used directly. In the case of continuous domains of optimization problems, the function $\zeta : \mathcal{D} \to \mathcal{G}$ that encodes the variables of the solving problem into a bit string, so called, a chromosome, is needed. The encoding function ζ is non-invertible and there does not exist the inverse function ζ^{-1}. A decoding function $\xi : \mathcal{G} \to \mathcal{D}_l \subset \mathcal{D}$ generates only 2^l representatives of solutions. This is a strong limitation of SGAs.

The parent selection s^p is carried out by, so called, *proportional method* (*roulette method*):

$$s^p\big(P(t)\big) = \{\boldsymbol{a}_{h_1}, \boldsymbol{a}_{h_2}, \ldots, \boldsymbol{a}_{h_\eta}\} : \quad h_k = \min\left\{h : \frac{\sum_{l=1}^{h}\phi_l^t}{\sum_{l=1}^{\eta}\phi_l^t} > \chi_k\right\}, \tag{1.1}$$

where $\{\chi_k = U(0,1) \mid k = 1, 2, \ldots, \eta\}$ are uniformly-distributed, independent random numbers from the interval $[0,1)$. In this type of selection, the probability that a given chromosome will be chosen as a parent is proportional to its fitness. Because sampling is carried out with returns, it can be expected that well-fitted individuals insert a few of their copies in the temporary population $P'(t)$.

Chromosomes from $P'(t)$ are recombined. In the case of SGA the *crossover* is the recombination operation. Chromosomes from $P'(t)$ are joined into pairs. The decision that a given pair will be recombined is taken with the given probability θ_r. If the decision is positive, an i-th position in the chromosome is randomly chosen and the information from the position $(i+1)$ to the end of chromosomes is exchanged in the pair:

$$\left\{\begin{array}{c}(a_1, a_2, \ldots, a_l)\\(b_1, b_2, \ldots, b_l)\end{array}\right\} \to \left\{\begin{array}{c}(a_1, \ldots, a_i, b_{i+1}, \ldots, b_l)\\(b_1, \ldots, b_i, a_{i+1}, \ldots, a_l)\end{array}\right\}.$$

New obtained temporary population $P''(t)$ is mutated. The individuals mutation m_{θ_m} is done separately for each bit in a chromosome. The bit value is changed to the opposite one with the given probability θ_m. Obtained population is the population of a new generation.

Historically, the first attempt to the formal description of the asymptotic characteristics of the SGA was proposed by Holland (1975). The combined effect of selection, crossover and mutation give so-called reproductive schema growth equation (Schaefer 2002):

$$\langle\eta(S, t+1)\rangle \geq \eta(S,t)\frac{\Phi(S,t)}{\bar{\Phi}(t)}\Big(1 - \theta_r\frac{\delta(S)}{l-1}\Big)(1-\theta_m)^{o(S)}, \tag{1.2}$$

where S is a schema defined over the alphabet of 3 symbols ('0','1',and '$\star$' of length l; each schema represents all strings which match it on all positions other than '$\star$'); $\eta(S,t)$ denotes the number of strings in a population at the time t matched by schema S; $\langle\cdot\rangle$ is a symbol of an expectation value; $\delta(S)$ is the defining length of the schema S — the distance between the first and the last fixed string positions; $o(S)$ denotes the order of the schema S — the number of 0 and 1 positions presented in the schema; $\Phi(S,t)$ is defined as the average fitness of the all strings in the population at the time t matched by the schema S; and $\bar{\Phi}(t)$ is the average fitness taken over all individuals in the population at the time t.

The equation (1.2) tells us about the expected number of strings matching a schema S in the next generation as a function of the actual number of strings matching the schema, the relative fitness of the schema, and its defining length and order. It is clear that above-average schemata with short defining length and low-order would still be sampled at exponentially increased rates.

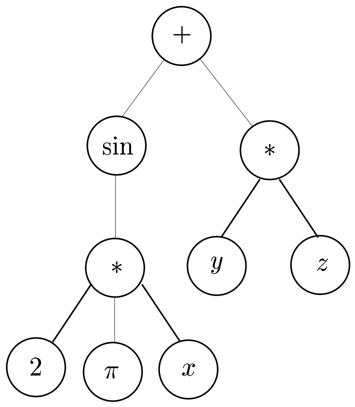


Fig. 1.1 . The sample of the tree which represents the function $f(x, y, z) = yz+\sin(2\pi x)$.

The above approach, which was criticized many times (see (Grefenstette 1993, Schaefer 2002)), can be treated as an attempt to evaluation of a numerical force increasing of a population (Whitley 1994). Vose (1999) proves that Markov processes, which model the genetic algorithms processing, are ergodic. This fact implies the asymptotic correctness in the probabilistic sense and the asymptotic guarantee of success (Schaefer 2002).

1.2.2. Genetic programming

Many trends of the SGA development are connected with the change of an individual representation. One of them deserves particular attention: each individual is a tree (Koza 1992). This little change in the GA gives evolutionary techniques possibility of solving problems, which are not early efforted to solve. This type of the GA is called the Genetic Programming (GP).

Two sets are needed to be defined before the GP starts: the set of terms $\mathcal{T}$ and the set of operators $\mathcal{F}$. In the initiation step, the population of trees is randomly chosen. For each tree leaves are chosen from the set $\mathcal{T}$ and other nodes are chosen from the set $\mathcal{F}$. Depending on $\mathcal{T}$ and $\mathcal{F}$ definitions a tree can represent a polyadic function, a logical sentence or a part of a programme code in a given programming language. Figure 1.1 presents the sample tree for $\mathcal{T} = \{x, y, z, 2, \pi\}$ and $\mathcal{F} = \{*, +, -, sin\}$. The new type of an individual representation needs new definitions of crossover and mutation operators, both of them are explained in Fig. 1.2.

1.2.3. Evolutionary programming

Evolutionary programming resides in the “phenotypic" category of simulations. It was devised by L.G. Fogel *et al.* (1966) in the mid-sixties for the evolution of finite state machines in order to solve prediction tasks. The environment was described

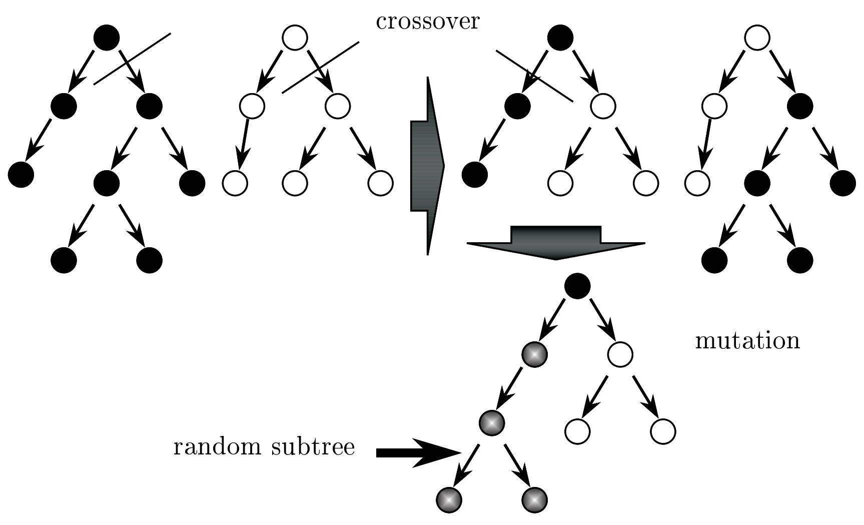


Fig. 1.2 . Genetic operators for the GP.

as a sequence of symbols (from a finite alphabet) and a finite state machine had to create a new symbol. The output symbol had to maximize some profit function, which was a measure of prediction accuracy. There are not preselection and recombination operators. Each machine of the current population generates an offspring by random mutation. There are five possible modes of random mutation that naturally result from the description of the finite state machine: change an output symbol, change a state transition, add a state, delete a state, or change the initial state. Mutation are chosen with respect to a uniform distribution. The best half number of parents and offspring were selected to survive.

The EP was extended by D.B. Fogel (1991, 1992) to work on real-valued object variables based on normally distributed mutations. This algorithm was called the meta-EP (Fogel *et al.* 1991) or the Classical EP (CEP) (Yao and Liu 1999). The description shown in Table 1.2 is based on (Bäck and Schwefel 1993, Yao and Liu 1999).

In the meta-EP, an individual is represented by pair $\boldsymbol{a} = (\boldsymbol{x}, \boldsymbol{\sigma})$, where $\boldsymbol{x} \in \mathbb{R}^n$ is a real-valued phenotype, $\boldsymbol{\sigma} \in \mathbb{R}_+^n$ is a self-adapted standard deviation vector for Gaussian mutation. For initialization, the EP assumes a bounded initial domains $\Omega_x = \prod_{i=1}^n [u_i, v_i] \subset \mathbb{R}^n$ and $\Omega_\sigma = \prod_{i=1}^n [0, c] \subset \mathbb{R}_+^n$ with $u_i < v_i$ and $c > 0$. However, the search domain is extended to $\mathbb{R}^n \times \mathbb{R}_+^n$ during the algorithm processing. As a mutation operator a Gaussian mutation with a standard deviation vector ascribed to an individual is used. All elements in the current population are mutated. Individuals from both parent and offspring populations participate in the new generation selection process. For each individual $\boldsymbol{a}_k$, q individuals are chosen at random from $P(t) \cup P'(t)$ and compared to $\boldsymbol{a}_k$ with respect to their fitness values. w_k is a number, how many of the q individuals are worse than $\boldsymbol{a}_k$.

Tab.1.2. The outline of the EP algorithm

```
I. Initiation
  A. Random generation
    $P(0) = \{\boldsymbol{a}_k^0 = (\boldsymbol{x}_k^0, \boldsymbol{\sigma}_k(0)) \mid k = 1,2,\dots,\eta\}$.
    $\boldsymbol{x}_k^0 = RANDOM(\Omega_x)$, $\boldsymbol{\sigma}_k^0 = RANDOM(\Omega_\sigma)$,
    $\Omega_x \subset \mathbb{R}^n$, $\Omega_\sigma \subset \mathbb{R}_+^n$.
  B. Evaluation
    $P(0) \to \Phi(P(0)) = \{\phi_k^0 = \Phi(\boldsymbol{x}_k^0) \mid k = 1,2,\dots,\eta\}$.
  C. $t = 1$.
II.Repeat:
  A. Mutation
    $P'(t) = m_{\tau,\tau'}(P(t)) = \{\boldsymbol{a'}_k^t \mid k = 1,2,\dots,\eta'\}$.
    ${x'}_{ki}^t = x_{ki}^t + \sigma_{ki}^t N_i(0,1)$, ${\sigma'}_{ki}^t = \sigma_{ki}^t \exp(\tau' N(0,1) + \tau N_i(0,1))$,
    $i = 1,2,\dots,n$,
    where $N(0,1)$ denotes a normally distributed one-dimensional random
    number with mean zero and standard deviation one, $N_i(0,1)$ indicates
    that the random number is generated anew for each component $i$.
  B. Evaluation
    $P'(t) \to \Phi(P'(t)) = \{{\phi'}_k^t = \Phi(\boldsymbol{x'}_k^t) \mid k = 1,2,\dots,\eta\}$.
  C. Selection of new generation
    $P(t+1) = s_{\theta_n}^n(P(t) \cup P'(t)) = \{\boldsymbol{a}_k^{t+1} \mid k = 1,2,\dots,\eta\}$.
    $\forall \boldsymbol{a}_k^t \in P(t) \cup P'(t)$,
    $\boldsymbol{a}_k^t \to \{\boldsymbol{a}_{kl}^t = RANDOM(P(t) \cup P'(t)) \mid l = 1,2,\dots,q\}$,
    $w_k^t = \sum_{l=1}^q \theta(\Phi(\boldsymbol{x}_k^t) - \Phi(\boldsymbol{x}_{kl}^t))$, $\theta(\alpha) = \begin{cases} 0 & \text{for } \alpha < 0 \\ 1 & \text{for } \alpha \geq 0 \end{cases}$,
    $P(t+1) \leftarrow \eta$ individuals with the best score $w_k^t$.
  D. $t = t + 1$.
 Until $(\iota(P(t)) = \text{true})$.
```

η individuals having highest score w_k are selected from 2η parents and offspring to form new population $P(t+1)$.

The analysis of the classical EP algorithm (Fogel 1992) giving a proof of the global convergence with probability one for the resulting algorithm, and the result is derived from defining a Markov chain over the discrete state space that is obtained from a reduction of the abstract search space $\boldsymbol{R}^n$ to the finite set of numbers representable on the digital computer.

1.2.4. Evolutionary strategies

The second well-known „phenotypical" algorithms are Evolutionary Strategies which have been introduced in mid-sixties by Rechenberg (1965) and Schwefel (1981). The description of the ES presented in this subsection is based on the article (Bäck and Schwefel 1993). The general form of the ES relies on the individual representation in the form of a pair: $\boldsymbol{a} = (\boldsymbol{x}, \mathbb{C})$, where $\boldsymbol{x} \in \boldsymbol{R}^n$ is a point in a searching space, and the fitness value of the individual $\boldsymbol{a}$ is calculated directly from the objective function: $\Phi(\boldsymbol{a}) = f(\boldsymbol{x})$. $\mathbb{C}$ is the covariance matrix for the n-dimensional normal distribution $\boldsymbol{N}(\boldsymbol{0}, \mathbb{C})$, having probability density function

$$p(\boldsymbol{z}) = \sqrt{\frac{1}{(2\pi)^n \det(\mathbb{C})}} \exp\left(-\frac{1}{2}\boldsymbol{z}^T \mathbb{C}^{-1} \boldsymbol{z}\right), \tag{1.3}$$

where $\boldsymbol{z} \in \boldsymbol{R}^n$. To assure positive-definiteness of the $\mathbb{C}$, it is described by two vectors: vector of standard deviations $\boldsymbol{\sigma}$ ($c_{ii} = \sigma_i^2$) and vector of rotation angles $\boldsymbol{\alpha}$ ($c_{ij} = \frac{1}{2}(\sigma_i^2 - \sigma_j^2) \tan 2\alpha_{ij}$). So, $\boldsymbol{a} = (\boldsymbol{x}, \boldsymbol{\sigma}, \boldsymbol{\alpha})$ is used to denote a complete individual.

There is no separated operation of selection of parents in ESs, this selection is strongly connected with the recombination mechanism. Different recombination mechanisms can be used in ESs to create λ new individuals. Recombination rules of determining an individual $\boldsymbol{a}' = (\boldsymbol{x}', \boldsymbol{\sigma}', \boldsymbol{\alpha}')$ have the following form:

$$a_i' = \begin{cases} a_{p,i} & \text{without recombination} \\ a_{p,i} \text{ or } a_{s,i} & \text{discrete recombination} \\ a_{p,i} + \chi(a_{s,i} - a_{p,i}) & \text{intermediate recombination} \\ a_{p_i,i} \text{ or } a_{s_i,i} & \text{global discrete recomb.} \\ a_{p_i,i} + \chi_i(a_{s_i,i} - a_{p_i,i}) & \text{global intermediate recomb.} \end{cases} \tag{1.4}$$

where indices p and s denote two parent individuals selected at random from $P(t)$, and $\chi \in [0,1]$ is uniform random variable. For global variants, for each component of $\boldsymbol{a}$ the parents p_i, s_i as well as χ_i are determined anew. Empirically, discrete recombination on object variables and intermediate recombination on strategy parameters have been observed to give best results (Bäck and Schwefel 1993).

Each recombined individual $\boldsymbol{a}'$ is subject to mutation. Firstly, strategy parameters are mutated, and then new object variables are calculated using new standard deviations and rotation angles:

$$\begin{aligned}
\boldsymbol{\sigma}'' &= \{\sigma_i' \exp(\tau' N(0,1) + \tau N_i(0,1)) \mid i = 1,2,\ldots,n\} \\
\boldsymbol{\alpha}'' &= \{\alpha_j' + \beta N_j(0,1) \mid j = 1,2,\ldots,n(n-1)/2\} \qquad (1.5) \\
\boldsymbol{x}'' &= \boldsymbol{x}' + \boldsymbol{N}(\boldsymbol{0},\boldsymbol{\sigma}'',\boldsymbol{\alpha}''),
\end{aligned}$$

where factors τ', τ and β are rather robust exogenous parameters, which are suggested to set as follows: $\tau' \approx 1/\sqrt{2\sqrt{n}}$, $\tau \approx 1/\sqrt{2n}$ and $\beta \approx 0.0873$ (5^o in radians).

Selection in ESs are completely deterministic. There exist two possible strategies:

- $(\mu + \lambda)$-ES — selecting μ best individuals out of the union of μ parents and λ offsprings;
- (μ, λ)-ES — selecting μ best individuals out of the set of λ offsprings ($\lambda > \mu$).

Although, the $(\mu + \lambda)$-ES is elitist and guarantees a monotonously improving performance, the effectiveness of global optimum searching is worse than in the case of (μ, λ)-ES, therefore the second one is recommended nowadays.

Under some restrictions it is possible to prove the convergence theorem for the evolutionary strategies (Bäck *et al.* 1991). Let the covariance matrix $\mathbb{C}$ be reduced to the standard deviation vector which possesses all components identical, i.e. $\boldsymbol{\sigma} = \{\sigma, \ldots, \sigma\}$ and $\sigma > 0$, and remains unchanged during the process. If the optimization problem with $\Phi_{opt} > -\infty$ (minimization) or $\Phi_{opt} < \infty$ (maximization) is regular then the evolutionary process converges to the global optimum in infinite limit of time with probability one.

1.3. Evolutionary search with soft selection (ESSS)

The ESSS algorithm was introduced by Galar (1989) relying on probably the simplest model of the Darwinian phenotypical evolution (Galar 1985). This selection-mutation process is executed in a multi-dimensional real space, on which fitness function is defined. At the beginning, a population of points is randomly chosen from the searching space and is iteratively changed by selection and mutation operators. As a selection operator the well-known proportional selection is used. Selected elements are mutated by adding a normally distributed random vector.

1.3.1. Phenotypic model of evolution

A basic phenotype evolution model was proposed by Galar (1985). The foundations of this model are as follows:

- There exists an environment of invariant properties which have a limited capacity.

- There exists a population of reproducing elements (individuals of the same species). The elements of the population are characterized by a set of features (phenotype quantitative features). The set of feature values determines the type of an element (phenotype). Each type is assigned to its fitness.
- The assumption that each element occupies only one place in the environment is also made. The elements "live" in the environment for some length of time (generation), and then a new generation is produced out of the actual one (reproduction).
- The new generation is created by selecting parent elements from the actual generation and changing their features (*asexual reproduction*).
- The choice of parents is accomplished by soft selection being a random process. Each parent element has a chance of allocating a descendant in the environment with probability proportional to the element quality.
- The descendant elements are not perfect copies of the parent elements. Type differences result from clear random mutation.

Basing on the above assumptions, the evolution is a motion of individuals in the phenotype space called also the adaptation landscape. This motion is caused by selection and mutation processes. Selection leads to concentration of the individuals around the best ones, but mutation introduces diversity of phenotypes and disperses the population in the landscape.

1.3.2. ESSS algorithm

Assumptions described above can be formalized by the algorithm presented in Table 1.3. A real, n-dimensional, searching space (an adaptation landscape) $\boldsymbol{R}^n$ is given. A fitness function Φ to be maximized is also defined on this adaptation landscape. Previously, an initial population $P(0)$ of η elements is randomly generated. If the ESSS algorithm is used to solve the optimization problem in $\boldsymbol{R}^n$ without constrains, the concept that an initial population has to be 'uniformly distributed' in the search space has no sense. One of the possible and rational solution is to create an initial population by adding η times a normally-distributed random vector to a given initial point $\boldsymbol{x}_0^0 \in \boldsymbol{R}^n$. The fitness $\phi_k^0 = \Phi(\boldsymbol{x}_k^0)$ is calculated for each element $\boldsymbol{x}_k^0$ of the population ($k = 1, 2, \ldots, \eta$). The searching process consists in generating a sequence of η-element populations. A new population $P(t+1)$ is created based only on the previous population $P(t)$. In order to generate a new element $\boldsymbol{x}_k^{t+1}$, a *parent* element is selected and mutated. Both selection and mutation are random processes. Each element $\boldsymbol{x}_k^t$ can be chosen as a parent with a probability proportional to its fitness ϕ_k^t (the *roulette method* (1.1)). A new element $\boldsymbol{x}_k^{t+1}$ is obtained by adding a normally-distributed random value to each entry of the selected parent:

$$\left(\boldsymbol{x}_k^{t+1}\right)_i = \left(\boldsymbol{x}_{h_k}^t\right)_i + N(0,\sigma) \quad i = 1, \ldots, n, \tag{1.6}$$

Tab.1.3. The outline of the ESSS algorithm

```
Input data
    $\eta$ – population size;
    $t_{\max}$ – maximum number of iterations (epochs);
    $\sigma$ – standard deviation of mutation;
    $\Phi : \boldsymbol{R}^n \to \boldsymbol{R}_+$ – non-negative fitness function, $n$ — number of features;
    $\boldsymbol{x}_0^0$ – initial point.
1. Initialize
  (a) $P(0) = \{\boldsymbol{x}_1^0, \boldsymbol{x}_2^0, \ldots, \boldsymbol{x}_\eta^0\} : \ (\boldsymbol{x}_k^0)_i = (\boldsymbol{x}_0^0)_i + N(0,\sigma)$
      $i = 1,2,\ldots,n; \quad k = 1,2,\ldots,\eta$
  (b) $\phi_0^0 = \Phi(\boldsymbol{x}_0^0)$
2. Repeat:
  (a) Evaluation
      $\Phi(P(t)) = \{\phi_1^t, \phi_2^t, \ldots, \phi_\eta^t\}$ where $q_k^t = \Phi(\boldsymbol{x}_k^t), \quad k = 1,2,\ldots,\eta.$
  (b) Selection
      $\{h_1, h_2, \ldots, h_\eta\}$ where $h_k = \min\left\{h : \frac{\sum_{l=1}^{h} \phi_l^t}{\sum_{l=1}^{\eta} \phi_l^t} > \zeta_k\right\}$
      and $\{\zeta_k\}_{k=1}^{\eta}$ are random numbers uniformly distributed in $[0,1)$.
  (c) Mutation
      $P(t) \to P(t+1);$
      $(\boldsymbol{x}_k^{t+1})_i = (\boldsymbol{x}_{h_k}^t)_i + N(0,\sigma), \quad i = 1,2,\ldots,n; \quad k = 1,2,\ldots,\eta.$
  Until $\quad (\iota(P(t)) = \text{true}).$
```

where the standard deviation σ is a parameter to be selected. It is important to note that there is no recombination (crossover) operator in the ESSS. However, the recombination operator is biologically motivated (Mendel's experiments) and possesses great importance in EAs based on the genotypic representation of individuals, in the case of phenotype simulations of evolution, which are based on floating point representation of individuals, the mutation seems to be the crucial operator of the evolutionary process (Fogel 1995, Fogel 1999, Galar 1989).

Numerical tests of the ESSS algorithm (Galar 1989) have proved essential advantages of soft selection in a global optimum finding in comparison with *hard selection* in which only the best individuals are chosen and only local optima are attained. The ESSS algorithm does not constitute an optimization algorithm in the sense of reaching extrema with a desired accuracy. The evolution process is not asymptotically convergent to an optimum and the interpolation effectiveness of soft selection is rather weak. The evolution leads next generations to an elevated response surface, rather than to maxima. In spite of that, search advantages of the ESSS algorithm suggest that this algorithm can be of real practical use in numerical packages for global optimization, especially when combined with local optimization algorithms.

First attempt at the ESSS convergence analysis was presented in (Karcz-Dulęba 1992, Karcz-Dulęba 1997, Karcz-Dulęba 2001a), where dynamics of infinite populations in a landscape of unimodal and bimodal fitness functions is considered. Galar and Karcz-Dulęba (1994) propose to consider the evolution dynamics in the state space of the population. The population state space is $n\eta$-dimensional. Because the evolution dynamics is independent on the elements' sequence in the population, the population state space does not cover all the $\boldsymbol{R}^{n\eta}$ space but only some convex, compact and multi-lateral subspace of $\boldsymbol{R}^{n\eta}$. Analytical results for the population of two elements, obtained by using population state space description, are presented in (Karcz-Dulęba 2001).

1.4. Summary

The evolutionary algorithm is distinguished by two main characteristics. Unlike other classes of optimization algorithms the EA operates on the population of individuals. In this way the knowledge about the environment is discovered simultaneously by many individuals, verifies information inherited from ancestors and is passed down from generation to generation. Species acquire their individual characteristics due to the survival of well fitted ones, that is seemingly a blind mechanism where only individuals well adapted to presence can survive and procreate. However, the nature does not select only the best individuals to procreate, sometimes even a weakly adapted one has a possibility of creating an offspring which can possess a feature without parallel in the population. This is the second evolution characteristic, called *soft selection*. If we give up the hard selection and use the soft one instead, assuming that weakly-adapted points (in the sense of the values of the objective function) can be selected to create offsprings, the possibility of the global optimum finding increases.

Although evolutionary algorithms have been successfully implemented to many practical problems, there have a number failures as well, and there is little understanding of what features of these domain make them appropriate or inappropriate for these algorithms. Because of the simple form of the ESSS algorithm, it seems to be useful material for transformation and analysis, which will be helpful to understand the nature of evolution algorithms. This is a task for the next chapters of this part of the book.

Chapter 2

NATURAL EXPLORATION MECHANISMS

If the adaptation landscape is composed of multi-dimensional hills, valleys, saddles and ridges, it is easy to prove that the Darwinian-type evolution has a cyclic nature (Galar 1989). Each cycle consists of two phases: active and latent. In relative short-lived active phases, the population of individuals climbs an adaptation slope to a neighbourhood of a local peak. The latent phase is a quasi-stationary state with sporadic fluctuations, such a phenomenon is known in biology as so called „Müller's catch" (Müller 1964): the population is trapped around the local optimum of the fitness, almost all mutations give worse fitted offsprings. If the occupied hill possesses a higher neighbour, the fluctuations can contribute to cross a saddle and a new active phase starts. The cyclic nature of evolution is consistent with the theory of „punctuated equilibria" (Eldredge and Gould 1972) which claims that the evolution is not evolve with steady motion but irregular — stepwise.

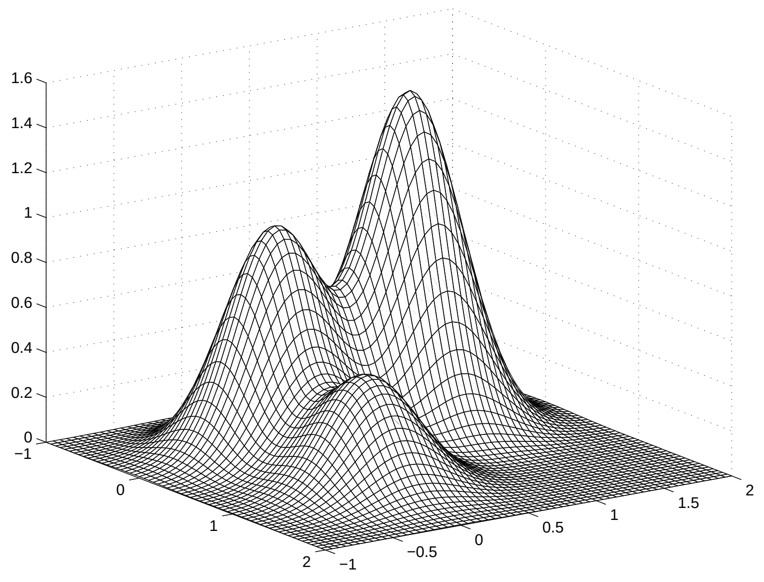
In order to illustrate the ESSS process, a sum of three two-dimensional Gaussian peaks

$$\begin{aligned}\Phi(\boldsymbol{x}) \quad = \quad & \frac{1}{2}\exp\Big(-5\big((1-x_1)^2+x_2^2\big)\Big)+\exp\Big(-5\big(x_1^2+x_2^2\big)\Big)+ \\ & +\frac{3}{2}\exp\Big(-5\big(x_1^2+(1-x_2)^2\big)\Big) \qquad (2.1)\end{aligned}$$

was chosen as a fitness function (Fig. 2.1a). The searching process can be split into two cyclically interchanged phases: an active phase (exploitation) and a latent phase (exploration) (Fig. 2.1b). In the short-lived active phase the concentrated population moves toward a local pick of the fitness. In the long-standing latent phase the trapped population fluctuates around the top in the search for a saddle of the adaptation landscape.

Long-time execution of the ESSS algorithm is caused, among other things, by long time intervals of the latent phases which result from the fact that the selection process prefers new offsprings allocated in well-exploited areas around the occupied peak. This is, of course, a drawback to this approach in the context of the effectiveness of the global optimization process. In order to overcome this problem, a natural idea is to exclude the neighbourhood of the occupied peak in the exploration process and to propose and analyze some mechanisms which

(a)



(b)

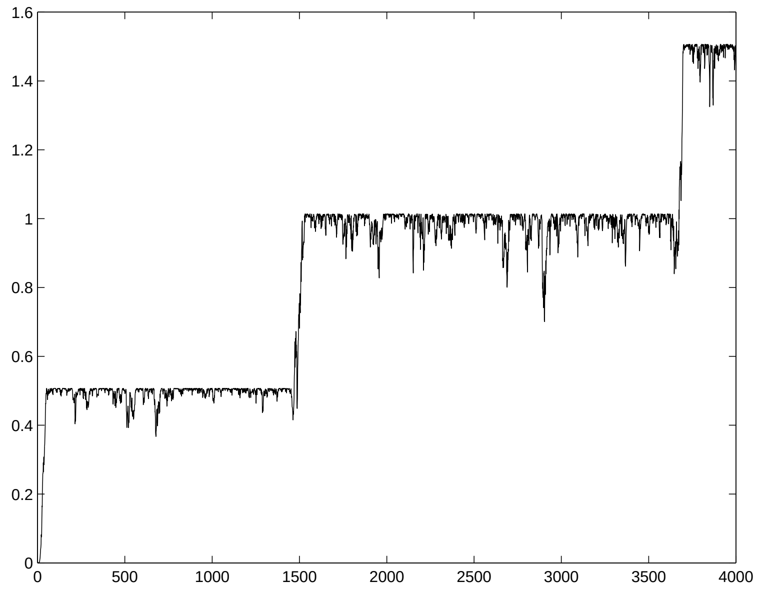


Fig. 2.1 . The 2D adaptation landscape used in the test (a); the fitness of the best element in the population vs. epochs (b). These results were obtained for $\eta = 15$, $\sigma = 0.03$, $t_{max} = 4000$ and $\boldsymbol{x}_0^0 = [2, -1]$.

would try to accelerate the saddle crossing and shorten the time spent on the latent phase. There are many instances of this idea in the specialized literature (Bäck *et al.* 1997, Goldberg 1989, Michalewicz 1996, Schaefer 2002).

There are three ways of influence on the ESSS algorithm processing (Obuchowicz 2003a):

- adaptation of algorithm parameters;
- modification of evolutionary operators;
- inclusion of the population influence on the fitness function.

In this chapter a few methods, which increase the ESSS exploration abilities, divided into three above classes, are presented.

2.1. Adaptation of algorithm parameters

The efficiency of the ESSS algorithm depends on values of input parameters: the standard deviation of mutation σ and the population size η (Table 1.3). Both parameters can be adapted during the algorithm execution. This idea is not novelty. The adaptation of the mutation standard deviations are included in the standard versions of the ES (Bäck and Schwefel 1993, Schwefel 1981) and the EP (Fogel *et al.* 1991). Standard deviations in the ES and the EP algorithms are self-adapted and changed randomly and continuously being subject to the same low as in the case of variables of the objective function. Unlike these algorithms, the adaptation of the standard deviation in the modified ESSS algorithm, called the ESSS-SVA (the ESSS with Simple Variance Adaptation), is controlled by the actual state of population (Obuchowicz and Patan 1997a). There is proposed a completely new mechanism, called *trap test*, which monitored whether the population is trapped around a local peak of the fitness function. The varying population size was firstly proposed for the GA algorithm by Arabas and coworkers in their GAVaPS algorithm (Arabas *et al.* 1994). Each individual is extended by a new parameter: *the life-time*, the value of which depends on the individual fitness. This technique, slightly modified, is implemented in the ESSS-VPS (the ESSS with Varying Population Size) algorithm (Obuchowicz and Korbicz 1999).

2.1.1. Adaptation of the standard deviation of mutation

Idea. When the population is trapped around a local peak, the standard deviation of mutation increases. This fact results in a larger variance of the population and a worse mean fitness. In this way, the mean fitness of the population decreases to a saddle level and the possibility of saddle crossing increases.

ESSS-SVA algorithm. When compared to ESSS, the ESSS-SVA algorithm is enriched by an additional mechanism which consists of three new procedures:

1. *Trap test.* The objective of this procedure is to determine whether the population quality changed substantially for a given number of epochs t_T. The

test is positive if the population displacement for the last t_T epochs is of the same order as the mutation variance σ_t.

2. *Adaptation of the mutation variance.* This procedure is started if an evolutionary trap is detected. The variance of the normal distribution used in mutation is multiplied by a constant $\alpha > 1$.

3. *Return to the initial variance* – If no evolutionary trap is detected, the variance of the normal distribution is set to the initial, relatively low value.

The ESSS-SVA algorithm can be written in the following form (see Table 1.3):

```
1. Initiation
2. Repeat
   (a) Estimation;
   (b) Choice of the best element in the history;
   (c) If Trap Test then Adaptation of the mutation variance else Return to
       the initial variance;
   (d) Selection;
   (e) Mutation;
   Until $t > t_{\max}$.
```

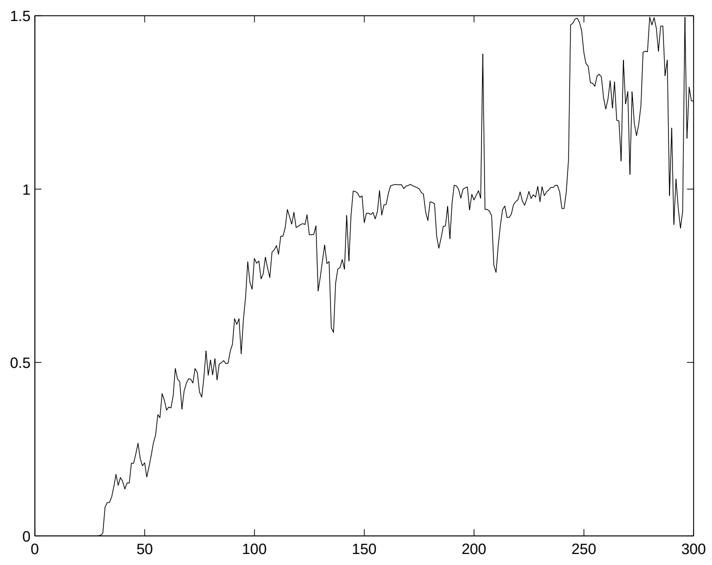
Illustrative example. In order to validate the performance of the ESSS-SVA process, let us consider again the sum of three two-dimensional Gaussian peaks as a fitness function (2.1). From the results shown in Fig. 2.2 it is easy to see that the applied SVA mechanism accelerates the effectiveness of saddle crossing.

2.1.2. Adaptation of the population size

Idea. When the population fluctuates around a local peak of the fitness function, individuals weakly fitted but geographically allocated closely to the saddle, seem to have a greater chance to create descendants in the other side of the saddle. The probability that such an individual will be selected as a parent increases when the population size is low, in other words, it has fewer rivals with better fitness. On the other hand the fluctuations of small population around a local peak are higher and the efficiency of local fitness maximum allocation is low. Therefore, the following hypothesis can be advanced that if the population size η is large, the process possesses a high quality of local fitness maximum allocation, but its ability of saddle crossing is poor in comparison with the ESSS algorithm with small value of η.

ESSS-VPS algorithm. The value of η is adapted in the ESSS-VPS algorithm. An individual element $\boldsymbol{x}_k^t$ in the ESSS-VPS algorithm is extended by adding one component: the life-time τ_k^t, i.e. the number of epochs in which the element $\boldsymbol{x}_k^t$ exists in population. The life-time τ_k^t is specified at the moment of an element

(a)



(b)

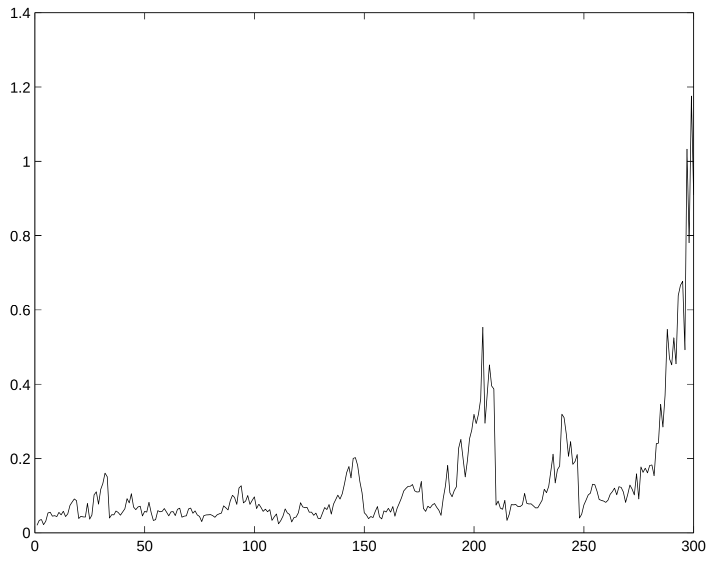


Fig. 2.2. The results of the ESSS-SVA searching process for the adaptation landscape (2.1). Figures (a) and (b) illustrate relations between the fitness of the best element in the population and the population variance, respectively, vs. epochs. These results were obtained for $\eta = 15$, $\sigma_0 = 0.03$, $\alpha = 1.1$, $t_T = 20$ and $t_{\max} = 300$.

Tab.2.1. Parameters used in simulations

Parameters	Values
maximum of epoch (t_{max})	3000
initial population size (η_0)	$5; 10; 20; 50; 100$
maximum life-time (τ_{max})	4
standard deviation (σ)	$0.005; 0.01; 0.05; 0.1; 0.5$

birth. It depends on the relation between an element's fitness $\Phi\left(\boldsymbol{x}_k^t\right)$ and the fitness history of population. A few methods of determining τ_k^t have been tested. The following one was chosen in (Obuchowicz and Korbicz 1999):

$$\tau_k^t = \left\lceil \tau_{max} \frac{\phi_k^t}{\phi_0^{t+1}} \right\rceil, \tag{2.2}$$

where τ_{max} is an exogenous parameter, $\lceil z \rceil$ returns the minimal integer value that is still greater then z. Unlike genetic algorithms, where the varying population size is a technique to avoid precocious convergence (Arabas *et al.* 1994), the population size decreases when average fitness of the population increases, in the ESSS-VPS algorithm the population size increases with average fitness of population and decreases, when the population is trapped around the local optimum or average population fitness decreases.

Illustrative example. Let us consider three two-variable functions from Appendix B: the „drop wave" function $f_4(x_1, x_2)$ (B4), Michalewicz's function $f_5(x_1, x_2)$ (B5), and Rastringin's function $f_7(x_1, x_2)$ (B7). All these functions are strongly non-linear and multimodal. The fitness function has been chosen in the form :

$$\Phi\left(\boldsymbol{x}_k^t\right) = f\left(\boldsymbol{x}_k^t\right) - f_{min}^t + \left(\frac{1}{\eta^t}\right)^2, \tag{2.3}$$

where $f_{min}^t = min\left(f\left(\boldsymbol{x}_k^t\right) | k = 1, \ldots, \eta^t\right)$ is the minimal value of f taken over all elements in the actual population of a size η^t, and f is a given objective function which has to be maximized. Such a fitness function is non-negative and its relative values in the actual population make the proportional selection effective.
Simulations have been done several times for all possible combinations of input ESSS-VPS parameters contained in the Table 2.1. When the best set of parameters was allocated, several starting points $\left(\boldsymbol{x}_0^0\right)$ were tested. The best results have been compared with the ESSS algorithms. Table 2.2 presents the percentages of 500 algorithm courses which have found the global optimum finding for three chosen objective functions. Simulations show that the ESSS-VPS algorithm is slightly better than the standard ESSS algorithm in localization of a global optimum of a given objective function.

Tab.2.2. Percentages of algorithms courses which have found the global optimum: $t_{max} = 3000$, $\eta_0 = 20$, $\tau_{max} = 4$, starting points: (9,9) for f_4, (3,3) for f_5, (4.5,4.5) for f_7.

alg.	ESSS			ESSS-VPS		
σ	0.01	0.05	0.1	0.01	0.05	0.1
f_4	0	0	8	0	0	10
f_5	0	10	88	4	46	96
f_7	0	8	98	0	18	96

2.2. Modification of evolutionary operators

The ESSS algorithm is composed of two evolutionary operators: selection and mutation. Many selection and mutation techniques known from the literature can be implemented (cf. (Bäck *et al.* 1997)), but methods presented below do not violate the models paradigm (see section 1.3.1).

2.2.1. Forced direction of mutation

Idea. The ESSS with Forced Direction of Mutation (ESSS-FDM) algorithm, firstly proposed by Obuchowicz and Korbicz (1998), has been designed as an adaptation algorithm in a time-varying landscape (Obuchowicz 1999b). The idea of FDM mechanism is following: if natural conditions existing in the environment favor some direction of alteration in the phenotype space, this direction is preferred not only by selection, but by mutation, too.

ESSS-FDM algorithm. The ESSS-FDM algorithm differs from the standard ESSS one only in the modification step. The elements selected are mutated by adding to each component i a normally-distributed random variable with expectation $m_i \neq 0$. This is unlike the ESSS algorithm, where $m_i = 0$ (see Table 1.3). Accordingly we have

(c) *Mutation*

$$
\begin{aligned}
&P(t) \to P(t+1);\\
&\left(\boldsymbol{x}_k^{t+1}\right)_i = \left(\boldsymbol{x}_{h_k}^{t+1}\right)_i + N\left(m_i^t, \sigma\right); \quad i = 1, 2, \ldots, n; \quad k = 1, 2, \ldots, \eta;\\
&m_i^t = \mu\sigma \frac{\langle x_i^t\rangle - \langle x_i^{t-1}\rangle}{\|\langle x_i^t\rangle - \langle x_i^{t-1}\rangle\|};\\
&\langle x_i^t\rangle = \frac{1}{\eta}\sum_{k=1}^{\eta}\left(\boldsymbol{x}_k^t\right)_i.
\end{aligned}
\tag{2.4}
$$

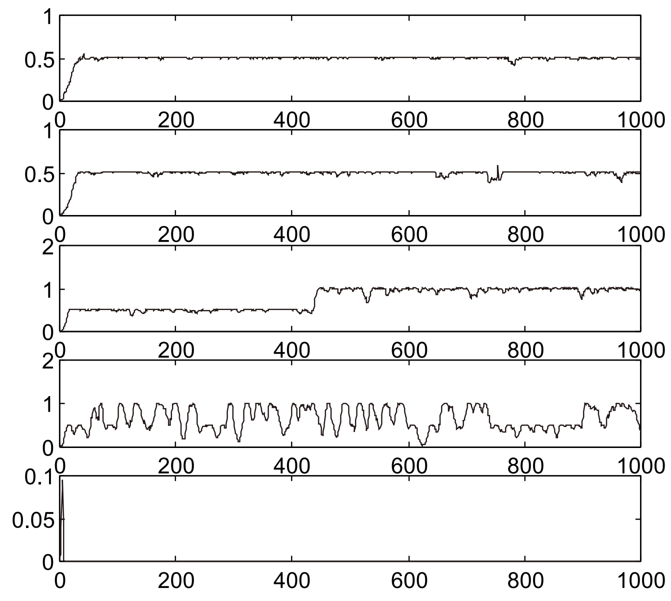


Fig. 2.3 . The fitness of the best element in the population vs. epochs; these results were obtained for $\eta = 20$, $\sigma = 0.03$, $t_{\max} = 1000$ and $\mu = 0$, $\mu = 0.5$, $\mu = 1$, $\mu = 2$, $\mu = 10$, respectively from the top to the bottom panels.

Illustrative example. Let us consider the fitness function $\Phi(\boldsymbol{x}) = f_1(\boldsymbol{x})$ (B1) from Appendix B. The initial point is chosen as $\boldsymbol{x}_0^0 = [-1, 2]$. Results for different set of the ESSS-FDM algorithm parameters are presented in Fig. 2.2.1. The mutation expectation vector $\boldsymbol{m}^t$ depends on standard deviation of normal distribution σ and is parallel to the latest trends of the population drift. The exogenous parameter μ, which is called the momentum, determines the proportion between the standard deviation σ and the length of the vector $\boldsymbol{m}^t : \mu = \|\boldsymbol{m}^t\|/\sigma$. If μ is too small, there is essentially no difference between the ESSS and the ESSS-FDM searching. In the case of a very large μ $\left(\|\boldsymbol{m}^t\| \gg \sigma\right)$, there is no possibility of changing the direction of the population drift, which was chosen in the beginning of the searching process.

2.2.2. Local selection

Idea. Almost all known evolutionary algorithms use a global selection, i.e., all individuals in the current population compete with each other for placing as many offspring individuals in the next generation as possible. In nature, such a selection is impossible for population dispersed on a wide area. The natural selection is local selection, where an individual only competes with rivals in its "ecological niche".

The idea of the local selection is as follows (Obuchowicz 2002a). Each individual of the current population is a centre of a sphere with a given radius ρ. One parent is selected from each of η spheres in accordance to the proportional selection (the roulette method), i.e., individuals located inside a given sphere compete with each other to become a parent. In this way the parent population of η individuals are created. There are proposed three variants of evolutionary algorithms which use the local selection.

ESSS algorithm with Local Selection (ESSS-LS). The ESSS-LS algorithm differs from the ESSS algorithm (see Table 1.3) only in *Selection* step. At first η sets of individuals are constructed

$$\mathcal{S}_j^t = \{\boldsymbol{x}_i^t \in P(t) : \|\boldsymbol{x}_i^t - \boldsymbol{x}_j^t\| < \rho\}, \quad j = 1, 2, \ldots, \eta. \tag{2.5}$$

The set $\mathcal{S}_j^t$ contains the individual $\boldsymbol{x}_j^t$ and its neighbours located in the sphere centered on $\boldsymbol{x}_j^t$ and the radius ρ, which is an input parameter of the algorithm. It easy to see, that $\boldsymbol{x}_i^t \in \mathcal{S}_j^t \Leftrightarrow \boldsymbol{x}_j^t \in \mathcal{S}_i^t$. From each set $(\mathcal{S}_j^t \mid j = 1, 2, \ldots, \eta)$ one parent is randomly chosen. The probability p_{ij}^t that the individual $\boldsymbol{x}_i^t \in \mathcal{S}_j^t$ will be chosen as a parent has the form

$$p_{ij}^t = \frac{\phi_i^t}{\sum_{\boldsymbol{x}_l^t \in \mathcal{S}_j^t} \phi_l^t}. \tag{2.6}$$

ESSS algorithm with Mixed Selection (ESSS-MS). In the ESSS-MS algorithm, local and global selection operators are applied alternately. At first, the local selection is used over t_l iterations and next the global selection is used over t_g iterations. Both time intervals t_l and t_g are input parameters.

ESSS algorithm with Adapted Local Selection (ESSS-ALS). The selection in the ESSS-ALS is local and almost the same as in the ESSS-LS. The ESSS-ALS differs from the ESSS-LS only in representation of individual and definition of $\mathcal{S}_j^t$. An individual in the ESSS-ALS algorithm is a pair $(\boldsymbol{x}_j^t, \rho_j^t)$, where ρ_j^0 is initially chosen at random from a given interval $(0, c)$ with uniform distribution. Then the equation (2.5) has a new form:

$$\mathcal{S}_j^t = \{\boldsymbol{x}_i^t \in P(t) : \|\boldsymbol{x}_i^t - \boldsymbol{x}_j^t\| < \rho_j^t\} \quad j = 1, 2, \ldots, \eta, \tag{2.7}$$

and hence the relation $\boldsymbol{x}_i^t \in \mathcal{S}_j^t \Leftrightarrow \boldsymbol{x}_j^t \in \mathcal{S}_i^t$ is not still valid.

The mutation in the ESSS-ALS operates not only on the phenotype $\boldsymbol{x}_j^t$, but also on the local radius ρ_j^t. A new radius is obtained as follows

$$\rho_j^{t+1} = |\rho_j^t + \sigma N(0, 1)|. \tag{2.8}$$

The experiment which compares described above techniques is presented in Section 2.4.1.

2.3. Population influence on the fitness

There are strong interaction between populations of individuals and the environment in nature. On the one hand, the population fitness depends on an available amount of food and water. On the other hand, individuals can influence on the environment in order to improve the living conditions or, sometimes, they destruct it when the environment cannot provide too large number individuals of a given species.

2.3.1. Impatience and polarization

Idea. Individuals weakly fitted but geographically allocated closely to the saddle, seem to have a greater chance to create descendants in the other side of the saddle. Thus, when the population does not achieve better values of the fitness function, an impatience operator is activated. This operator modifies the fitness of individuals so that the remote individuals from the centre of the population are rewarded. In this way the population is dispersed like in the *sharing* method (Goldberg 1989). In the case of the ESSS algorithm with the impatience operator new unexpected effect occurs: the „polarization". Dispersed population assembles in two clusters located on either side of the population centre and rotating around it. If the adaptation landscape is regular then this phenomenon accelerates the saddle crossing.

Impatience operator. The impatience operator has been proposed by Galar and Kopciuch (1999). It transforms the original fitness $\Phi(\boldsymbol{x}_j)$ of the j-th individual to the effective fitness $\Phi_e(\boldsymbol{x}_j)$ as follows:

$$\Phi_e(\boldsymbol{x}_j) = \left(\frac{d_j}{d_A} + c\right) \Phi(\boldsymbol{x}_j), \tag{2.9}$$

where $d_j = \|\boldsymbol{x}_j - \langle \boldsymbol{x} \rangle\|$ and $d_A = \frac{1}{\eta}\sum_{k=1}^{\eta} d_k$.

The efficiency of the impatience and polarization (IP) effect has been tested on the bimodal fitness function composed of the sum of two Gaussian peaks (Galar and Kopciuch 1999). Obtained results suggest that:

- the IP effect affects on the decreasing of the number of iterations needed to cross a saddle;
- the IP mechanism increases the saddle crossing efficiency for large populations, almost no effect has been noticed in the case of very small populations;
- the IP effect is profitable in the case of low dimensional landscapes.

2.3.2. Erosion

Idea. The great efficiency of saddle crossing of the ESSS algorithm and all its modification described in this section is not a sufficient condition for scouring a wide area of the searching space. There is a possibility that population of searching individuals will fluctuate between two or more neighbouring local optima. There exists natural phenomenon, which influences the rate of saddle crossing and prevents searching individua from coming back to the previously inspected areas. This phenomenon is known as the landscape deterioration by the evolutionary trapped population. Therefore, the population decreases its fitness itself indirectly searching the escape way from the trap.

ESSS algorithm with Deterioration of the Objective Function (ESSS-DOF). The ESSS-DOF influences on the topology of an objective function. It contains an additional step which is composed of the following procedures (Obuchowicz 1997):

- *Trap test* – the objective of this procedure is to determine whether the population quality changed substantially for a given number of epochs.
- *Erosion* – this procedure transforms the objective function $\Phi(\boldsymbol{x})$ as follows:

$$\Phi(\boldsymbol{x}) = \begin{cases} \Phi(\boldsymbol{x}) - G(\boldsymbol{x}) & \text{for } \Phi(\boldsymbol{x}) \geq G(\boldsymbol{x}), \\ 0 & \text{for } \Phi(\boldsymbol{x}) < G(\boldsymbol{x}), \end{cases} \tag{2.10}$$

  where $G(\boldsymbol{x})$ is the deterioration peak chosen in the Gaussian form

$$G(\boldsymbol{x}) = h \exp\left(-\frac{1}{2}(\boldsymbol{x} - \boldsymbol{\chi})^T \mathbb{T}^{-1} (\boldsymbol{x} - \boldsymbol{\chi}) \right), \tag{2.11}$$

  where h, $\boldsymbol{\chi}$ and $\mathbb{T}$ are a height, a central point and a correlation matrix of the Gaussian deterioration peak, respectively.

The deterioration peak (2.11) has to approximate the currently occupied local quality peak. If the population is trapped around the local optimum, it can be assumed that the population distribution approximates the shape of this peak. Thus, the parameters of the deterioration peak can be chosen in the form (Obuchowicz 1997):

$$h = \phi^t_{\max}, \tag{2.12}$$

$$\boldsymbol{\chi} = \langle \boldsymbol{x}(t) \rangle, \tag{2.13}$$

$$\mathbb{T} = \mathbb{C}_t, \tag{2.14}$$

where $\phi^t_{\max}$ is the fitness of the best individual in the actual population $P(t)$, $\langle \boldsymbol{x}(t) \rangle$ and $\mathbb{C}_t$ are the expectation vector and the covariance matrix of $P(t)$, respectively.

Although, the ESSS-DOF algorithm is the most efficient in saddle crossing in comparison with other methods based on ESSS, it possesses one main disadvantage. The deterioration function (2.11) does not approximate current quality peak with the sufficient accuracy. If an evolutionary trap is detected, the modified quality peak looks like a crater with steep slopes. The deterioration mechanism should be performed several times until the population starts scouring another landscape area. A large number of deterioration peaks used by the algorithm influences the computation time and space complexity. In (Obuchowicz 2000b) a modified ESSS-DOF algorithm, named ESSS-DOF* has been proposed. Basing on analytical consideration contained in (Karcz-Dulęba 2001a) it can be shown that if the fitness function is chosen in the form of Gaussian peak:

$$\Phi(\boldsymbol{x}) = \prod_{i=1}^{n} \exp\left(-\frac{x_i^2}{2\tau_i^2}\right), \tag{2.15}$$

an infinite population in the latent phase can be modelled by the Gaussian density function with the variance:

$$\nu_{\infty,i}^2 = \frac{1}{2}\sigma^2\left(1+\sqrt{1+\left(\frac{2\tau_i}{\sigma}\right)^2}\right), \qquad i = 1,2,\ldots,n. \tag{2.16}$$

The covariance matrix of the deterioration Gaussian peak (2.11) is approximated using (2.16). This part of algorithm consists of four steps:

1. Calculate the covariance matrix $\mathbb{C}_t$ of the actual population;
2. Find all eigenvectors and eigenvalues of the matrix $\mathbb{C}_t$ in order to define an orthonormal matrix $\mathbb{U}$ and a diagonal matrix $\operatorname{diag}(\nu_{ti}^2|i = 1,2,\ldots,n)$ such that:

$$\mathbb{C}_t = \mathbb{U}\operatorname{diag}(\nu_{ti}^2|i = 1,2,\ldots,n)\mathbb{U}^T; \tag{2.17}$$

3. Calculate the variances of the deterioration peak (2.16):

$$\tau_i^2 = \nu_{ti}^2\left(\frac{\nu_{ti}^2}{\sigma^2}-1\right); \tag{2.18}$$

4. Calculate the covariance matrix $\mathbb{T}$ of the deterioration peak:

$$\mathbb{T} = \mathbb{U}\operatorname{diag}(\tau_i^2|i = 1,2,\ldots,n)\mathbb{U}^T. \tag{2.19}$$

Illustrative example. In order to validate the performance of the ESSS-DOF* algorithm, let us consider the sum of three two-dimensional Gaussian peaks as a fitness function (2.1). The ESSS-DOF* algorithm has a much greater convergence rate than other algorithms from the ESSS family (Fig. 2.4). If the population gets stuck in an evolutionary trap, the process of local peak erosion is started. This effect decreases the average fitness of the population. The population fitness reduces to a saddle level, and running away towards other quality peak is possible. The deteriorated peak will never be attractive for the searching population. Two disadvantages of ESSS-DOF* should be noted. First, if the algorithm approximates well the peak shape, then the fitness function after the erosion procedure vanishes in the area occupied by the population. Consequently, the evolutionary algorithm works like a typical stochastic one and its effectiveness in locating a new peak substantially decreases. The other problem is that the population composed of a finite number of individuals fluctuates around the local peak and may non

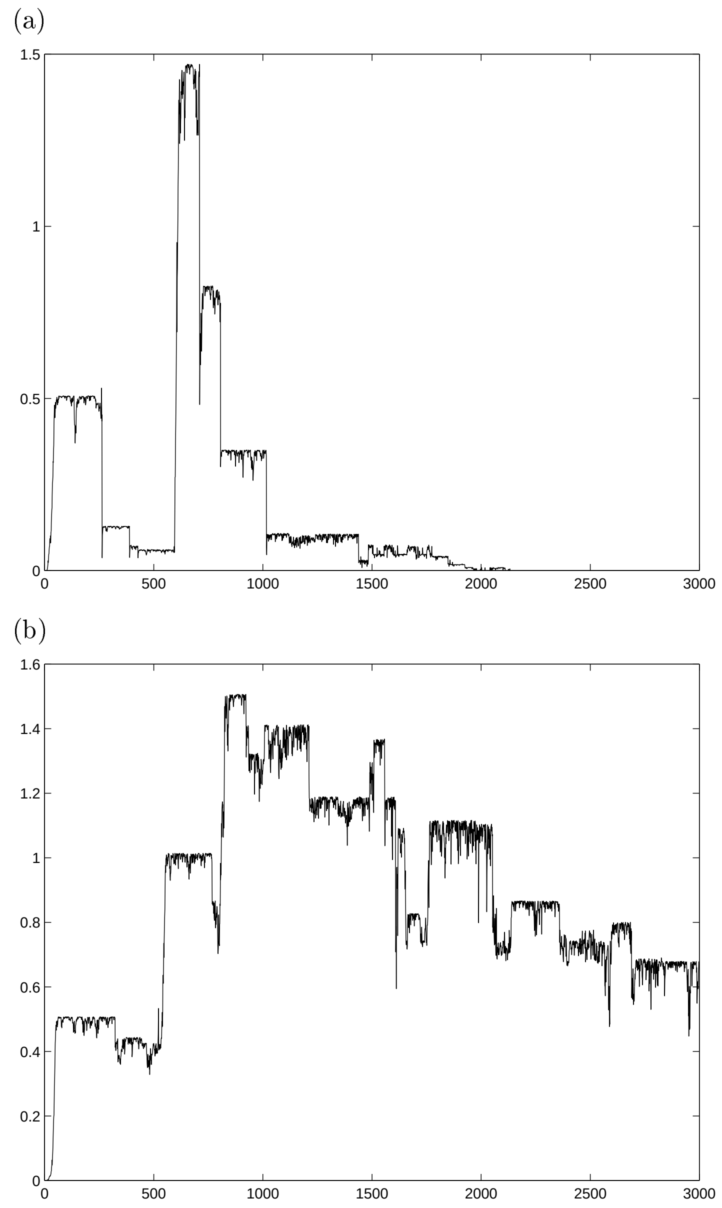


Fig. 2.4. An exemplary ESSS-DOF search in the adaptation landscape (2.1): (a) the fitness of the best element in the population vs. epochs for the ESSS-DOF*, (b) the fitness of the best element in the population vs. epochs for the ESSS-DOF; these results were obtained for $\eta = 15$, $\sigma = 0.03$, $t_{\max} = 3000$ and $t_T = 20$ epochs.

uniformly surround it at the time moment the erosion procedure fires. As a result, the neighbouring peaks may be deteriorated. Figure 2.4a shows that the medium peak of the adaptation landscape was aggravated by the lowest one. Fortunately, the highest one remained intact. Taking account of the numerical complexity and above disadvantages of the ESSS-DOF* algorithm, the ESSS-DOF algorithm seems to be more lucrative.

2.4. Experimental comparisons

2.4.1. Local selection contra global selection

Many numerical simulations (about 800) with two two-variable objective functions have been carried out. First function, Schwefel's problem 2.22

$$\begin{aligned} &f_1(x_1, x_2) = |x_1| + |x_2| + |x_1||x_2|, \\ &\min f_1 = 0, \quad \arg\min f_1 = (0, 0), \end{aligned} \tag{2.20}$$

is unimodal, while the second one, Rastringin's function

$$\begin{aligned} f_2(x_1, x_2) = x_1^2 + x_2^2 - 10(\cos(2\pi x_1) + \cos(2\pi x_2) + 20, \\ \min f_2 = 0, \quad \arg\min f_2 = (0, 0), \end{aligned} \tag{2.21}$$

is multimodal. The fitness function $\Phi(\boldsymbol{x}_k^t)$ is calculated from the objective function f_i, which has to be minimized, using the formulae similar to (2.3)

$$\Phi(\boldsymbol{x}_k^t) = f_{i,\max}^t - f_i(\boldsymbol{x}_k^t) + \frac{1}{\eta^2}, \tag{2.22}$$

where $f_{i,\max}^t = \max(f_i(\boldsymbol{x}_k^t) \mid k = 1, 2, \ldots, \eta)$ is the maximum value of f_i taken over all elements in the current population.

Schwefel's problem 2.22. Population in the ESSS algorithms (Fig. 2.5) is focused around some centre $\langle \boldsymbol{x}_k^t \rangle$ with the standard deviation

$$\sqrt{\langle (\boldsymbol{x}_k^t - \langle \boldsymbol{x}_k^t \rangle)^2 \rangle} \approx \sigma.$$

This algorithm is very effective in the optimum finding problem, the standard deviation of mutation σ controls the convergence rate of the algorithm.

The ESSS-LS algorithm performance strongly depends on the radius ρ of the neighbourhood sphere. If $\rho \ll \sigma$ there is no selection. We get clear stochastic expansion of the population independently of the objective function. If $\rho \gg \sigma$ then the ESSS-LS algorithm reduces to the ESSS algorithm. The most interesting case is $\rho \approx \sigma$, where population is divided into few subpopulations which sporadically exchange individuals and are explored on a wide area. However, the local selection decreases the effectiveness of the algorithm in the case of an unimodal objective function like the Schwefel's problem 2.22.

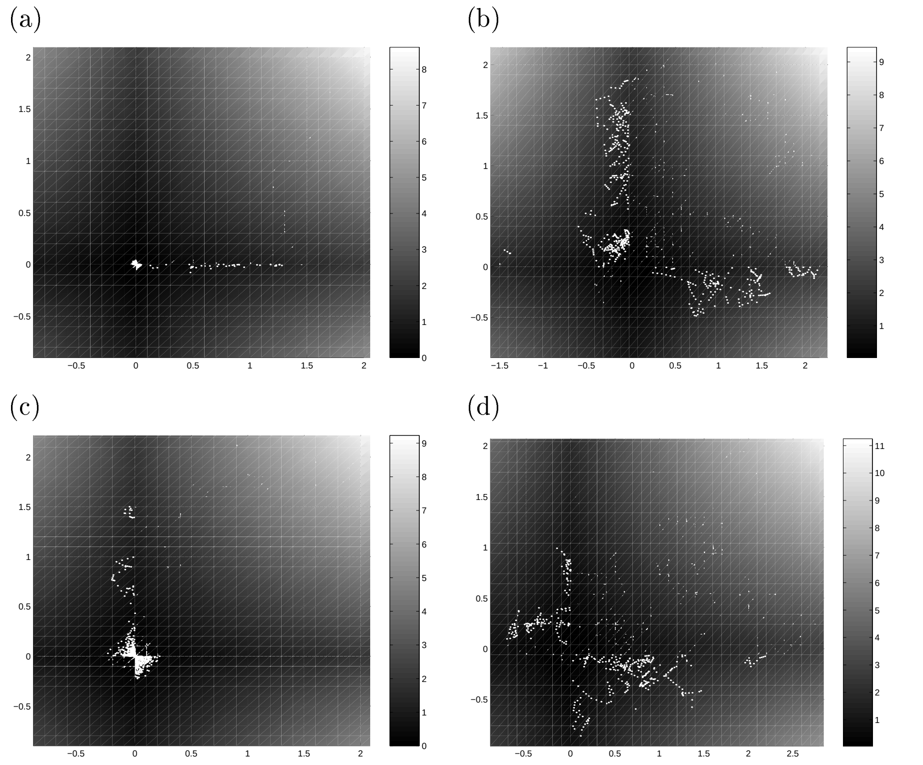


Fig. 2.5. Schwefel's problem 2.22. Traces of the best elements locations of the ESSS (a), the ESSS-LS (b), the ESSS-MS (c) and the ESSS-ALS (d) algorithms — typical results ($\eta = 20$, $\sigma = 0.03$, $\boldsymbol{x}_0^0 = [2,2]$, $\rho = 0.07$, $t_T = 20$, $t_{\max} = 1000$).

Mixed selection mechanism in the ESSS-MS algorithm joins the advantages of both the local and global selection. Firstly it uses the local selection and allows the population to divide it into small subpopulations and explore on a wide area. After some generations the selection is changed from local to global and population is focused on one or few subpopulation around the best obtained points. In the case of the Schwefel's problem the ESSS-MS effectiveness can be comparable with the ESSS one.

The ESSS-ALS algorithm effectiveness is placed between the ESSS-LS and ESSS-MS ones. During its processing the population autonomously divides into few large subpopulations, which possess a high exploitation rate, and many small ones with high exploration rate. Unlike the ESSS-MS algorithm, such a division is not forced by exchanging local and global selection, which is controlled by a researcher, but results from the implemented local selection with neighbourhood sphere radius self-adapted during the algorithm processing.

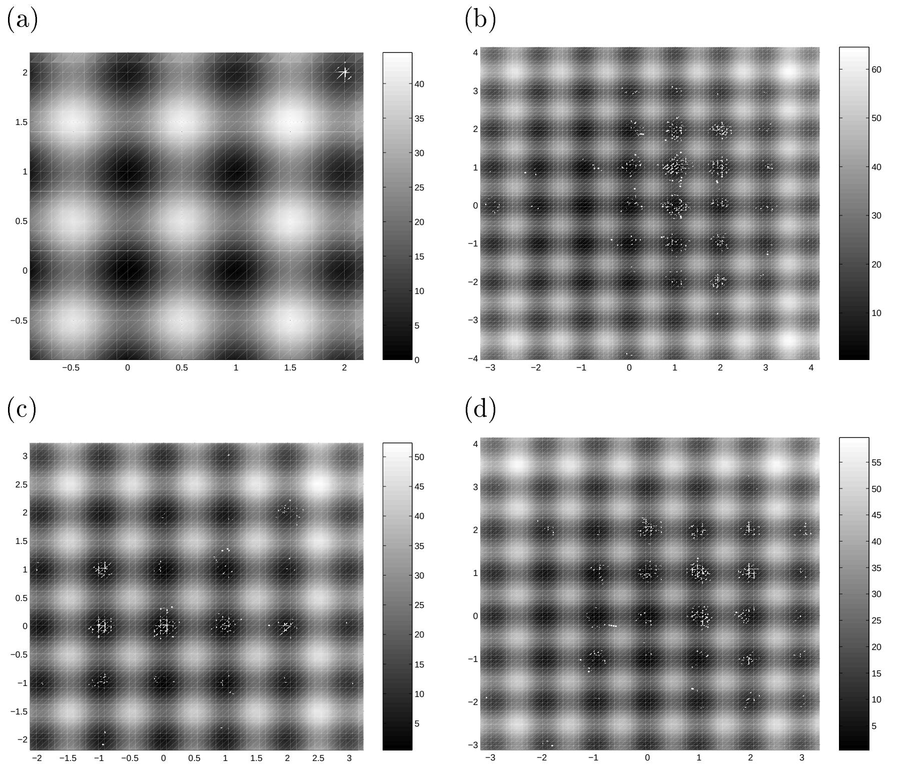


Fig. 2.6 . Rastringin's function. Traces of the best elements locations of the ESSS (a), the ESSS-LS (b), the ESSS-MS (c) and the ESSS-ALS (d) algorithms — typical results ($\eta = 20$, $\sigma = 0.03$, $\boldsymbol{x}_0^0 = [2, 2]$, $\rho = 0.1$, $t_T = 20$, $t_{\max} = 3000$).

Rastringin's function. However, three, proposed in this work, local selection implementations rather interfere in quick local optimum allocation, their high exploration rate makes them very effective in the case of multimodal objective functions.

Figures 2.6 present the algorithms processing for the same standard deviation of mutation σ and the same population size η. The ESSS algorithm cannot leave the local valley, in which the initial population has been created. The ESSS-LS, ESSS-MS and ESSS-ALS algorithms successively explore consecutive valleys and find the global optimum. The exploration rate of the ESSS-LS is the highest one and the population disperses in many local valleys and none of them is preferred. In the case of the ESSS-MS and the ESSS-ALS the population dispersion is guided in the direction of global optimum.

Tab.2.3. Parameter values used in the simulations

Algorithm	Parameter	Value
all	$t_{\max}$	3000
all	η	$10, 20, 50$
all	σ	$0.015,\ 0.03, 0.05, 0.1$
all	$\boldsymbol{x}_0^0$	$[1, 0, \ldots, 0]$
ESSS-SVA	α	1.1
ESSS-SVA, ESSS-DOF	t_t	10
ESSS-FDM	μ	0.3

2.4.2. Efficiency of saddle crossing

2.4.2.1. Experiment

Let us consider the problem described in Appendix A.

The aim of the experiment is to compare the effectiveness of the four evolutionary algorithms (ESSS, ESSS-SVA, ESSS-FDM, and ESSS-DOF) in saddle crossing problem. Two parameters are chosen as measures of this effectiveness. The first one is the average number t_c of epochs which is needed to cross the saddle by a given algorithm. The second one p_c is the percentage of runs in which the global optimum was found in a given time $t_{\max}$. In order to make the Gaussian mutation to be independent on the landscape dimension (see the next chapter), the mutation (1.6) is substituted by the following

$$\left(\boldsymbol{x}_k^{t+1}\right)_i = \left(\boldsymbol{x}_{h_k}^t\right)_i + N(0, \frac{\sigma}{\sqrt{n}}) \quad i = 1, \ldots, n, \tag{2.23}$$

where the standard deviation σ is a parameter to be selected.

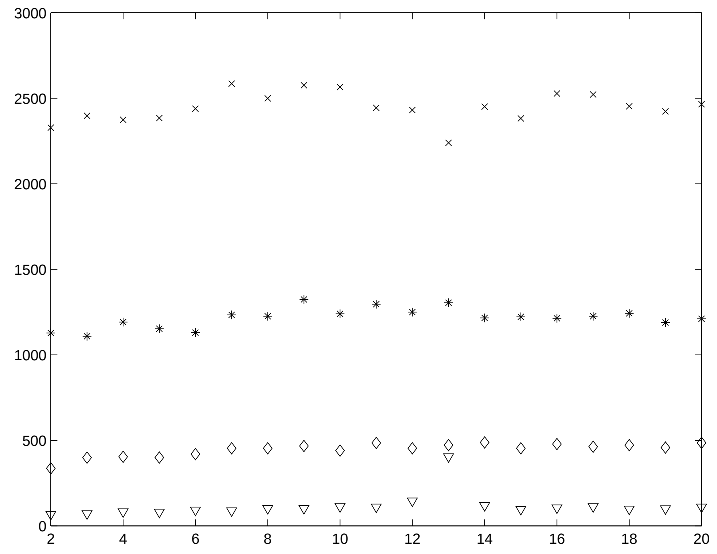
The algorithms were processed over 400000 times: 500 times for 19 dimensions ($n = 2, 3, \ldots, 20$) and 12 combinations of the population size η and the standard deviation of mutation σ (see Table 2.3). Because of the numerical complexity of the ESSS-DOF algorithm, the number of times it was processed its processing is limited to the following combinations of η and σ : $\eta = 20$ and $\sigma = 0.015, 0.03, 0.05, 0.1$; $\sigma = 0.05$ and $\eta = 10, 20, 50$.

2.4.2.2. Results for ESSS

The results obtained for the ESSS algorithm are comprehensively presented in Figs. 2.7, 2.8 and 2.9. Analyzing those, the following conclusions can be drawn:

- The saddle crossing effectiveness of the ESSS algorithm is independent of the adaptation landscape dimension. This result opposes the results presented in (Galar 1989). The reason is different mutation operators in both approaches.

(a)



(b)

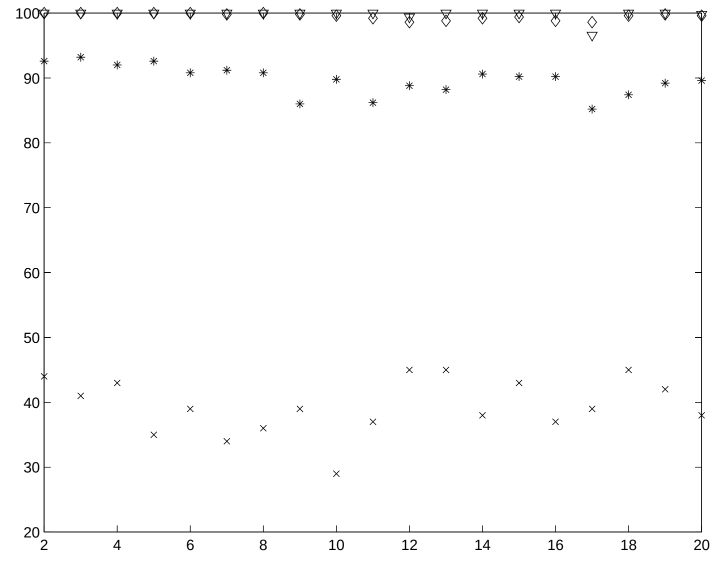


Fig. 2.7. The ESSS algorithm with $\eta = 10$: the mean number of epochs t_c needed to cross the saddle (a) and percentages p_c of the algorithm runs in which the global optimum was found in 3000 (b) vs. the dimension of the adaptation landscape n; ($\sigma = 0.015$: crosses, $\sigma = 0.03$: stars, $\sigma = 0.05$: diamonds, $\sigma = 0.1$: triangles).

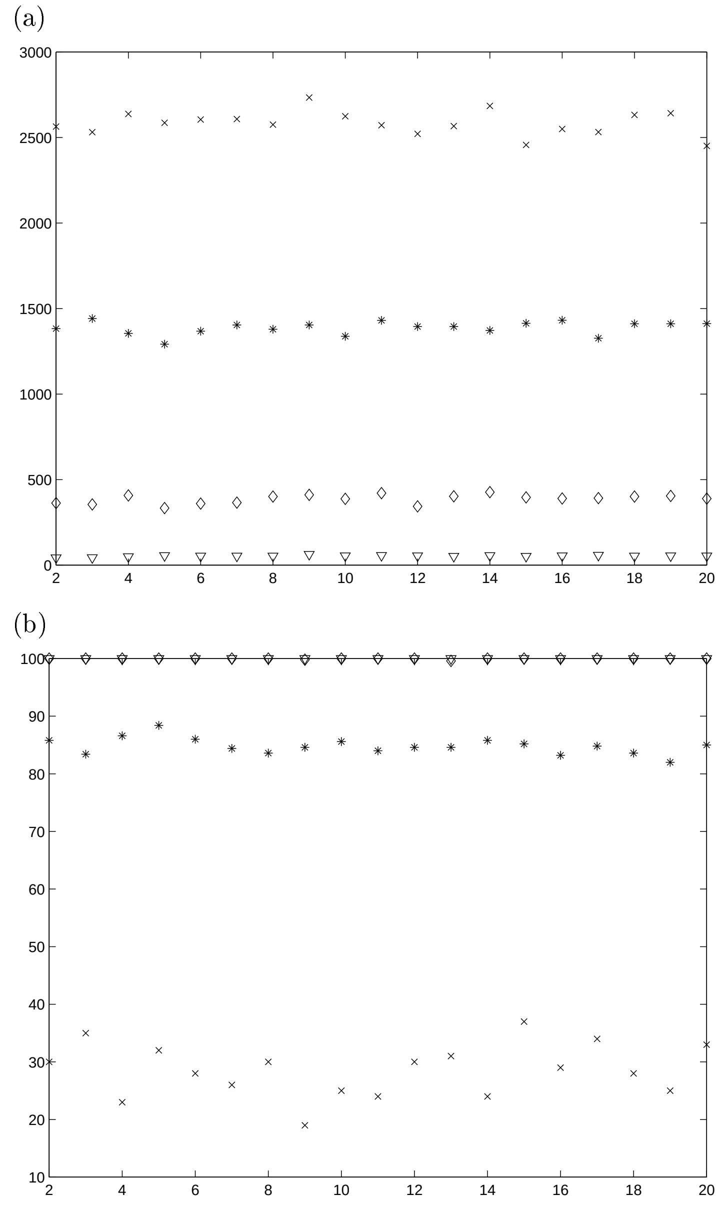
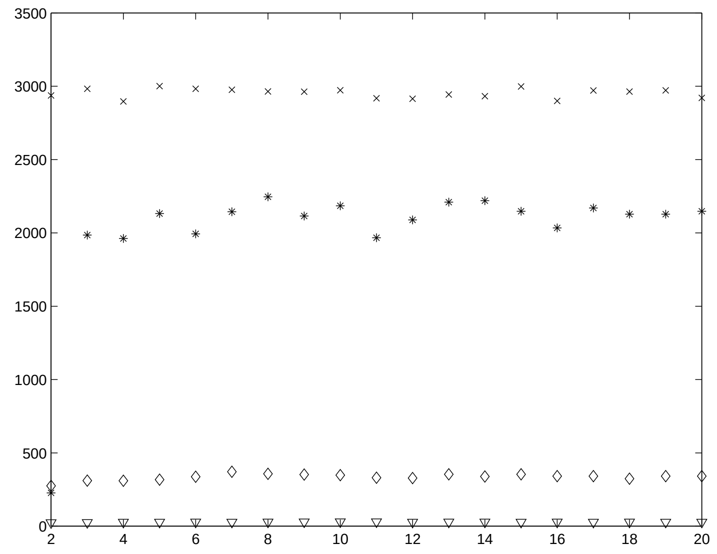


Fig. 2.8 . The ESSS algorithm with $\eta = 20$: the mean number of epochs t_c needed to cross the saddle (a) and percentages p_c of the algorithm runs in which the global optimum was found in 3000 (b) vs. the dimension of the adaptation landscape n; ($\sigma = 0.015$: crosses, $\sigma = 0.03$: stars, $\sigma = 0.05$: diamonds, $\sigma = 0.1$: triangles).

(a)



(b)

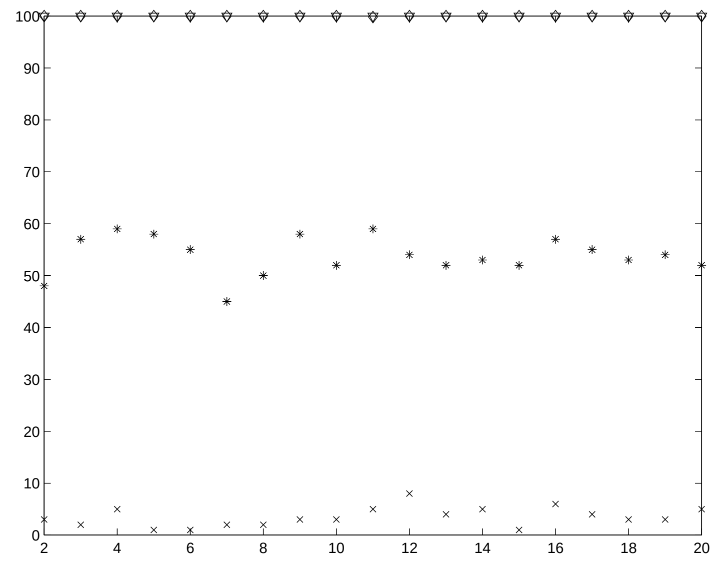


Fig. 2.9 . The ESSS algorithm with $\eta = 50$: the mean number of epochs t_c needed to cross the saddle (a) and percentages p_c of the algorithm runs in which the global optimum was found in 3000 (b) vs. the dimension of the adaptation landscape n; ($\sigma = 0.015$: crosses, $\sigma = 0.03$: stars, $\sigma = 0.05$: diamonds, $\sigma = 0.1$: triangles).

Galar (1989) modifies a selected parent according to the formulae (1.6) and the mutation radius is dependent on the landscape dimension. In this work the equation (2.23) is used.

- It is not surprising that the saddle crossing effectiveness decreases with the standard deviation σ. If one chooses the standard deviation as a length unit, the saddle width relatively increases when σ decreases.
- A small population is better than a large one. For small values of σ, i.e. when percentage of successful algorithm runs $p_c < 100\%$, p_c decreases and t_c increases when the population size η increases. For large σ, t_c is smaller in the case of large populations, but the number of fitness function evaluations is still larger.

The good ability of the ESSS algorithm with small population size in the saddle crossing problem was reported in (Galar and Karcz-Dulęba 1994), where an extremely small population — of two individuals — was considered. On the other hand, a small population, in comparison to a large one, localizes optimum points with lower accuracy. Fitting of both input parameters η and σ requires some research experience.

2.4.2.3. Results for ESSS-SVA

Figs. 2.10, 2.11 and 2.12 present results obtained for the ESSS-SVA algorithm. One can notice that the value of the initial standard deviation σ_0 does not influence significantly the effectiveness of the algorithm.

Unlike in the case of the ESSS algorithm, there exists a dependence between the ESSS-SVA algorithm's effectiveness and the dimension of the adaptation landscape. It is significant especially for small populations. The algorithm produces satisfactory results in low-dimensional landscapes. When the dimension increases, the algorithm's effectiveness violently decreases and fluctuates around some steady level for high dimensions. The researcher who wants apply the ESSS-SVA algorithm to a given problem has to fit the size of the population several times higher than the dimension of the searching space.

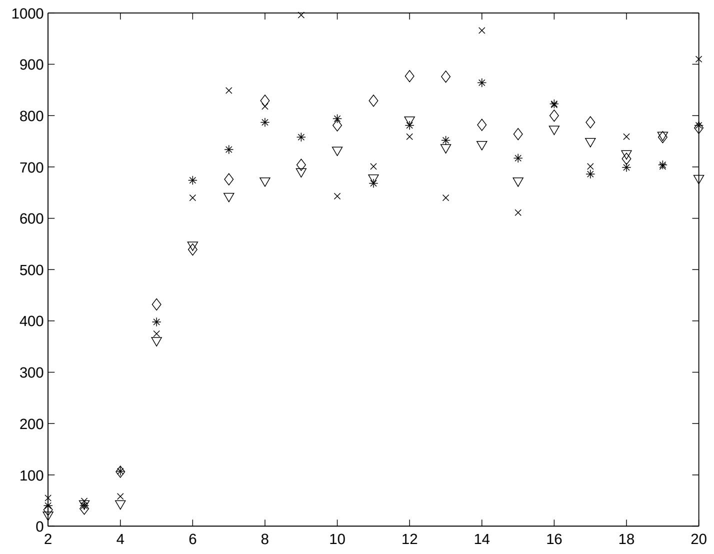
2.4.2.4. Results for ESSS-DOF

The most interesting simulating result for the ESSS-DOF algorithm is that the population has to be at least a simplex in the n-dimensional searching space (Fig. 2.13):

$$\eta \geq n + 1. \tag{2.24}$$

If the above relation is not satisfied, then $p_c = 0\%$. No exceptions was noticed during the simulations. This feature can be explained by the fact that one needs at least $n+1$ points in order to approximate a convex of a fitness function. If only the expression (2.24) is satisfied, the population size does not influence significantly the ESSS-DOF algorithm's effectiveness.

(a)



(b)

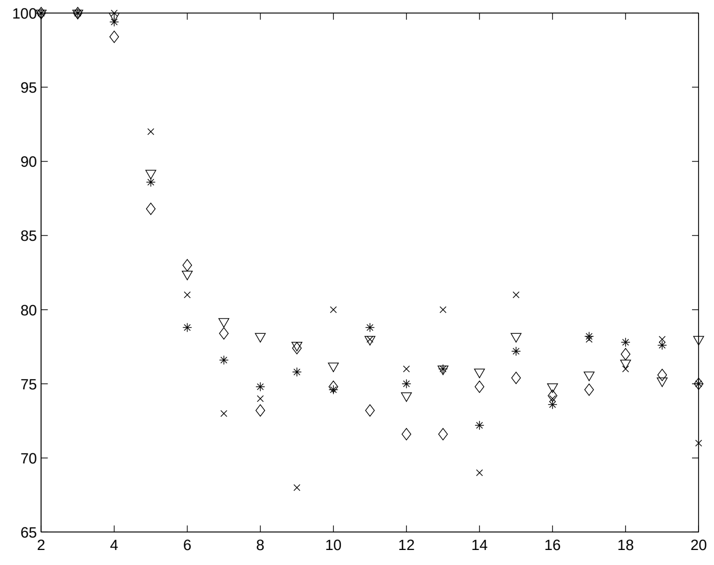
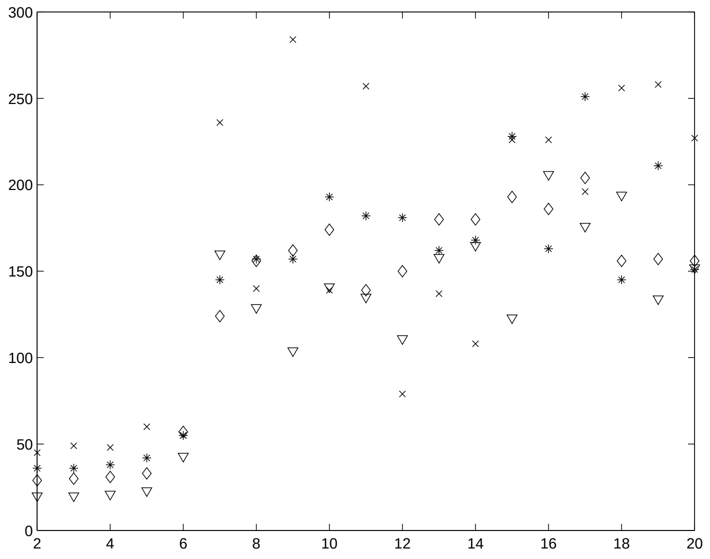


Fig. 2.10 . The ESSS-SVA algorithm with $\eta = 10$: the mean number of epochs t_c needed to cross the saddle (a) and percentages p_c of the algorithm runs in which the global optimum was found in 3000 (b) vs. the dimension of the adaptation landscape n; ($\sigma_0 = 0.015$: crosses, $\sigma_0 = 0.03$: stars, $\sigma_0 = 0.05$: diamonds, $\sigma_0 = 0.1$: triangles).

(a)



(b)

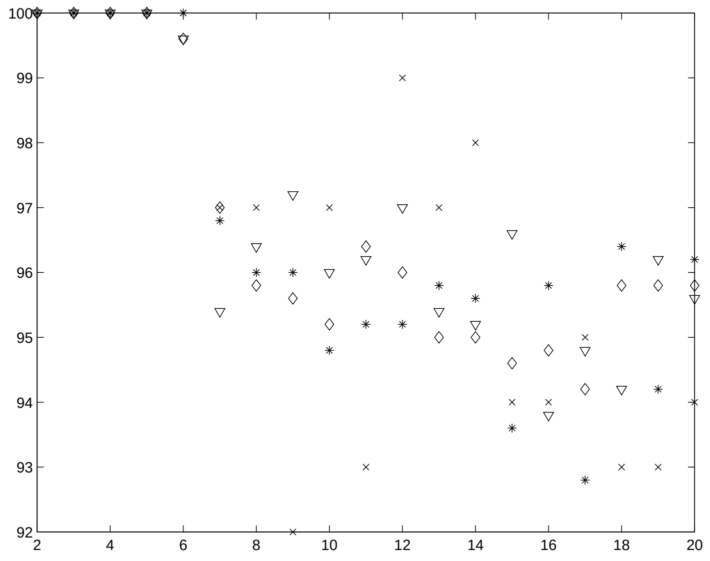
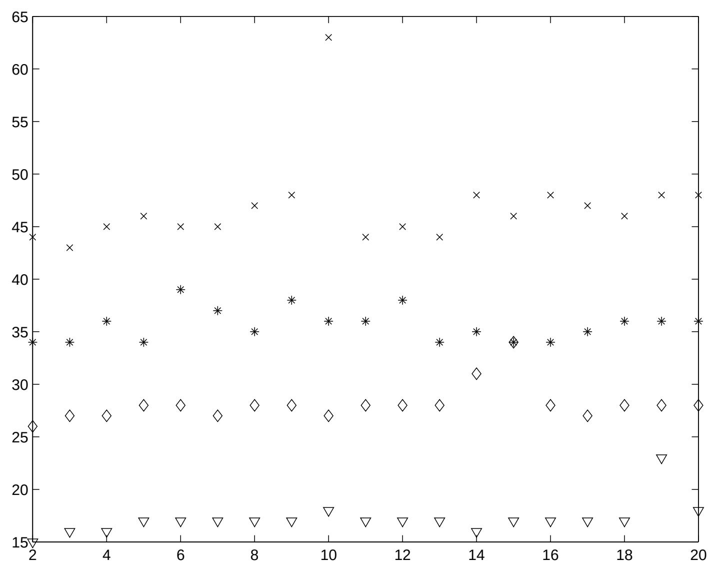


Fig. 2.11 . The ESSS-SVA algorithm with $\eta = 20$: the mean number of epochs t_c needed to cross the saddle (a) and percentages p_c of the algorithm runs in which the global optimum was found in 3000 (b) vs. the dimension of the adaptation landscape n; ($\sigma_0 = 0.015$: crosses, $\sigma_0 = 0.03$: stars, $\sigma_0 = 0.05$: diamonds, $\sigma_0 = 0.1$: triangles).

(a)



(b)

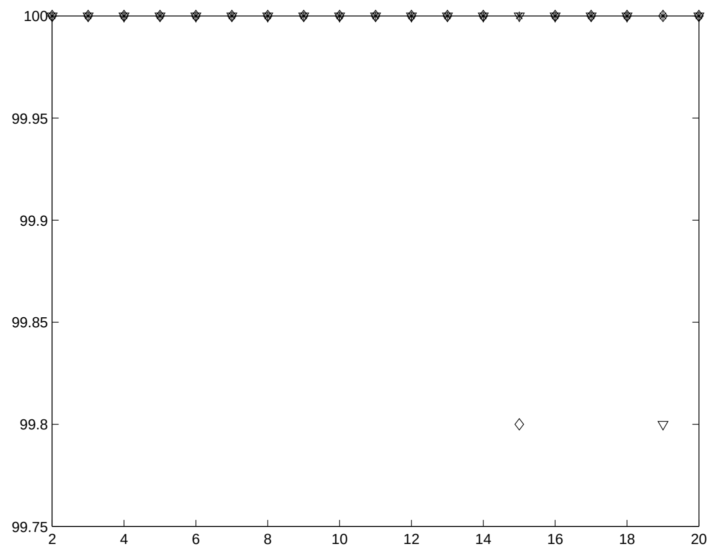


Fig. 2.12 . The ESSS-SVA algorithm with $\eta = 50$: the mean number of epochs t_c needed to cross the saddle (a) and percentages p_c of the algorithm runs in which the global optimum was found in 3000 (b) vs. the dimension of the adaptation landscape n; ($\sigma_0 = 0.015$: crosses, $\sigma_0 = 0.03$: stars, $\sigma_0 = 0.05$: diamonds, $\sigma_0 = 0.1$: triangles).

Analyzing the relation between the ESSS-DOF algorithm's effectiveness and the standard deviation σ (Fig. 2.14) one can distinguish three ranges. The first one is the range of low standard deviations, where the ESSS-DOF algorithm effectiveness of saddle crossing is pure. The second range of high values of σ describes a very effective algorithm searching. There exists some intermediate range for which the algorithm effectiveness cannot be clearly characterized. Both t_c and p_c stochastically fluctuate from low to high values and these results are not recurrent.

2.4.2.5. Results for ESSS-FDM

The best results of the ESSS-FDM algorithm in the saddle crossing problem were obtained for large values of the standard deviation σ and the population size η (Figs. 2.15, 2.16 and 2.17). Relationships between the algorithm's effectiveness and both of the input parameters, σ and η, are not trivial or correlated. If $\sigma = 0.1$, then the best results are obtained for $\eta = 50$, but if $\sigma = 0.015$, then the effectiveness of the large population is worse.

One can say that the population size is small or large only in comparison with the landscape dimension. The dependence of the algorithm's effectiveness on the searching space dimension (Fig. 2.15a) clearly illustrates that the ESSS-FDM algorithm works well if only $\eta > n$ and the value of the σ is sufficiently high.

2.4.2.6. Comparison

The graphs presented in Figs. 2.18 $\div$ 2.29 comprehensively illustrate the comparative characteristics of algorithms considered in this work. The following conclusions are worth to noticing:

- Generally, the ESSS-SVA algorithm seems to be the most effective one for the saddle crossing problem. If it finds the global optimum, it needs the shortest time to do it in comparison with other considered algorithms. However, in some cases, described below, its effectiveness is very poor.

- In the case of small population sizes, the saddle crossing ability of the ESSS-SVA and ESSS-FDM algorithms becomes worse with an increase in the searching space dimension. Extremely, the SVA and FDM mechanisms seem to disturb ESSS when input parameters are well adjusted (Figs. 2.19, 2.21b, 2.24b, 2.25 and 2.27).

- The effectiveness of the ESSS-DOF algorithm is high in low dimensional landscapes (Figs. 2.19a, 2.20,2.21, 2.23a, 2.25a, 2.26,2.27, 2.29a). Otherwise, its values are like the effectiveness of the ESSS algorithm. The advantage of the ESSS-DOF algorithm is that it does not return to previously occupied peaks, but tries to explore new, unknown areas. This feature cannot occur in the case of the chosen fitness function (A1).

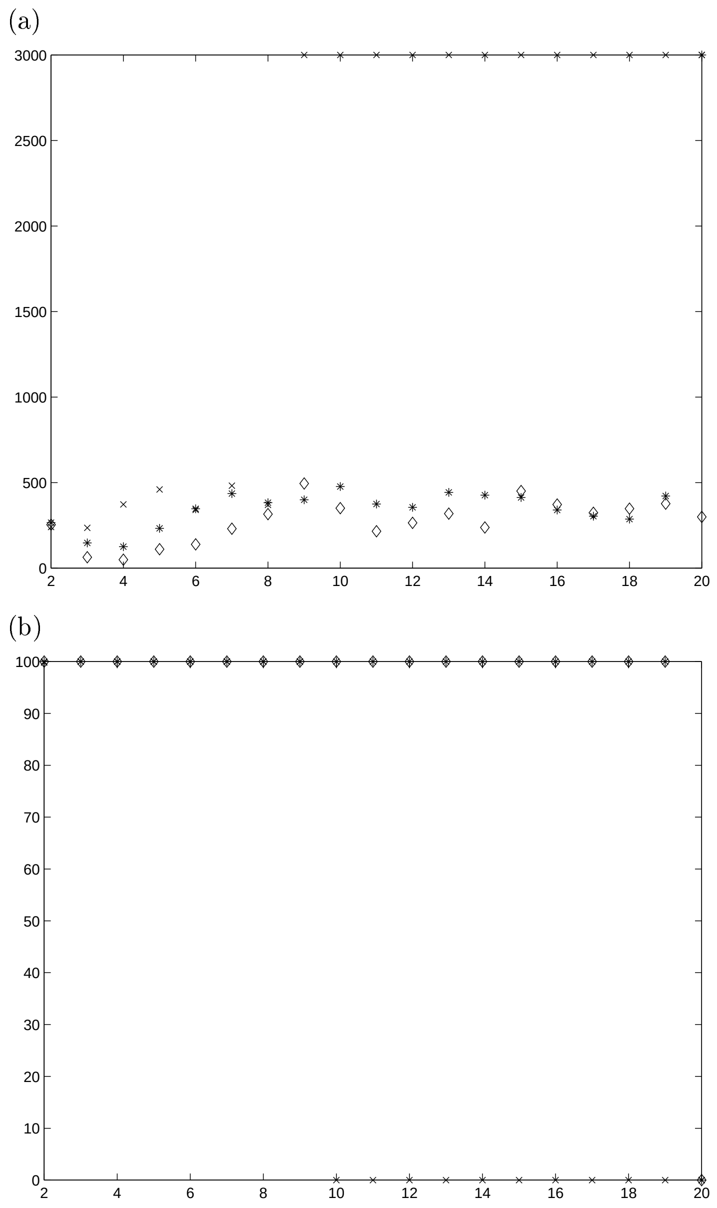
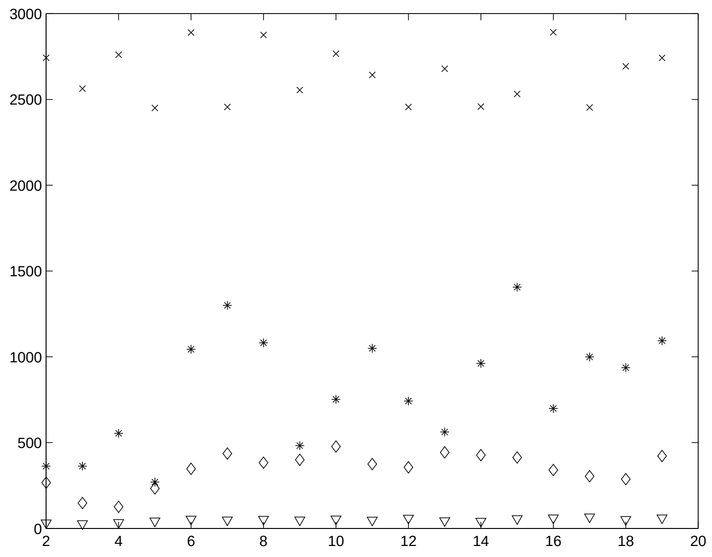


Fig. 2.13 . The mean number of epochs t_c needed to cross the saddle by the ESSS-DOF algorithm vs. the dimension of the adaptation landscape n (a). Percentages p_c of the ESSS-DOF algorithm runs in which the global optimum was found in 3000 epochs vs. the dimension of the adaptation landscape n (b). ($\sigma = 0.05$ and $\eta = 10$: crosses, $\eta = 20$: stars, $\eta = 50$: diamonds).

(a)



(b)

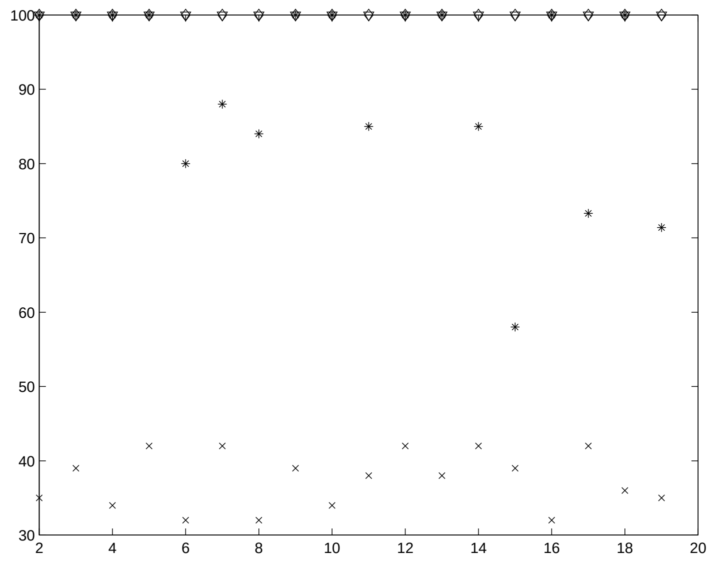


Fig. 2.14 . The mean number of epochs t_c needed to cross the saddle by the ESSS-DOF algorithm vs. the dimension of the adaptation landscape n (a). Percentages p_c of the ESSS-DOF algorithm runs in which the global optimum was found in 3000 epochs vs. the dimension of the adaptation landscape n (b). ($\eta = 20$ and $\sigma = 0.015$: crosses, $\sigma = 0.03$: stars, $\sigma = 0.05$: diamonds, $\sigma = 0.1$: triangles).

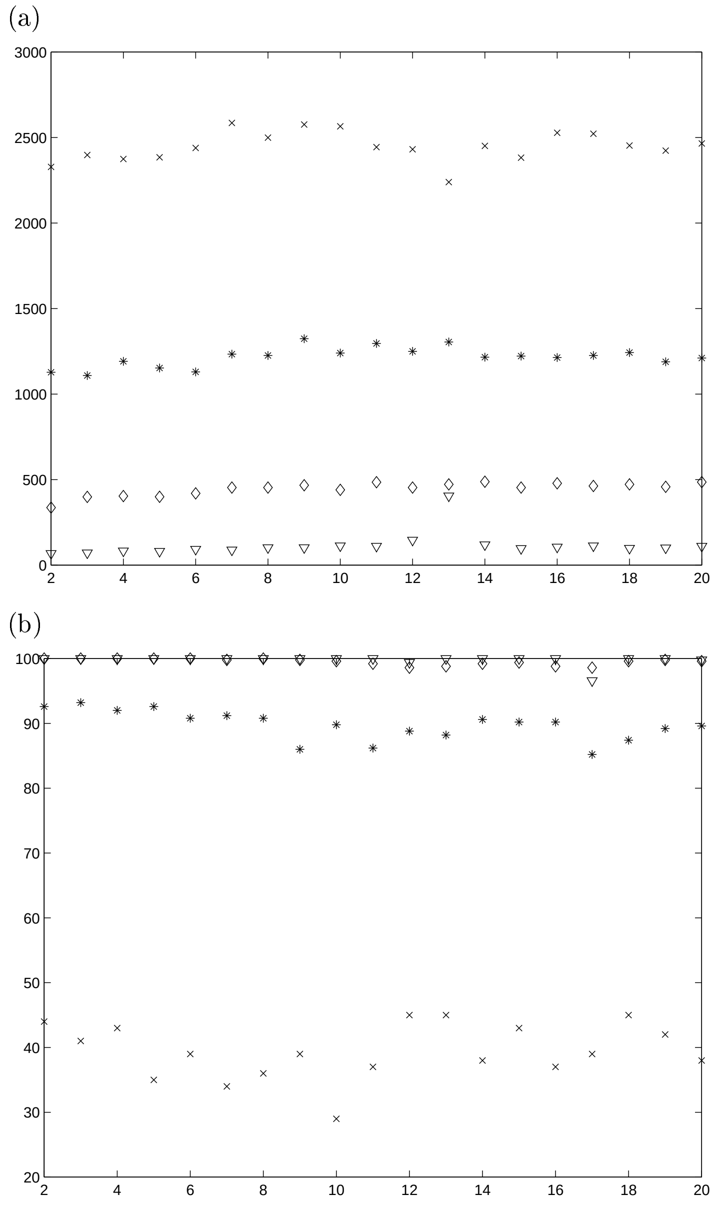


Fig. 2.15 . The ESSS-FDM algorithm with $\eta = 10$: the mean number of epochs t_c needed to cross the saddle (a) and percentages p_c of the algorithm runs in which the global optimum was found in 3000 (b) vs. the dimension of the adaptation landscape n; ($\sigma = 0.015$: crosses, $\sigma = 0.03$: stars, $\sigma = 0.05$: diamonds, $\sigma = 0.1$: triangles).

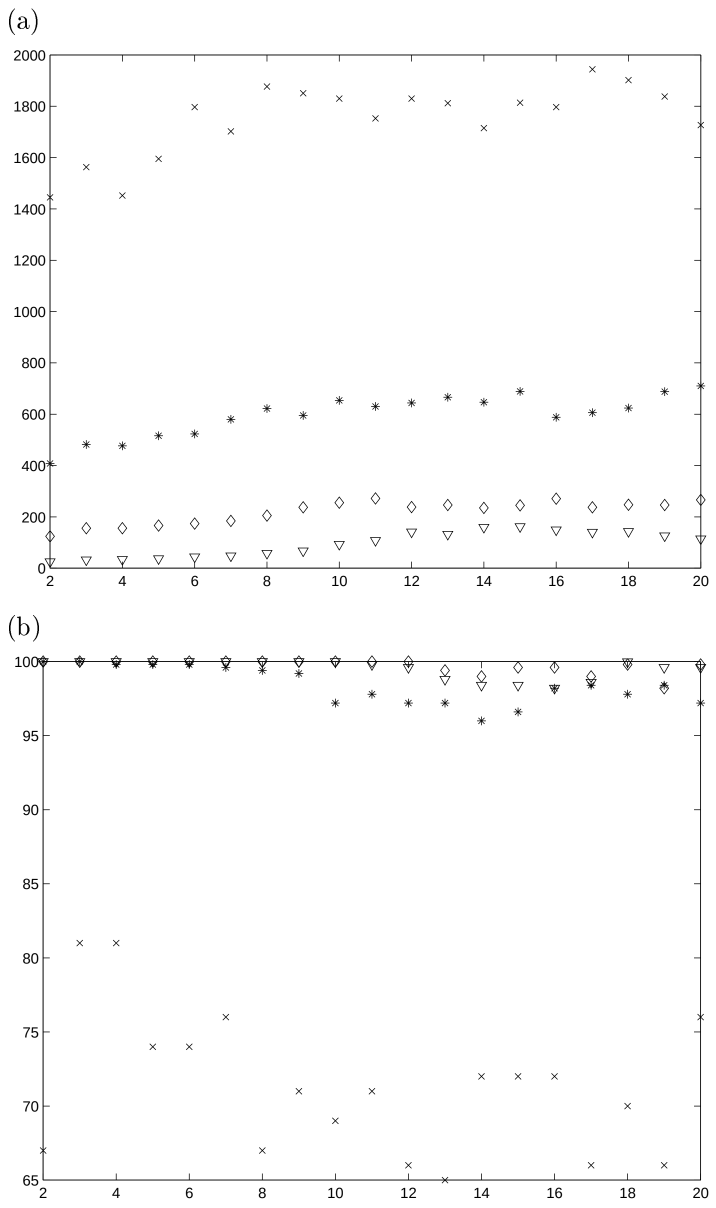
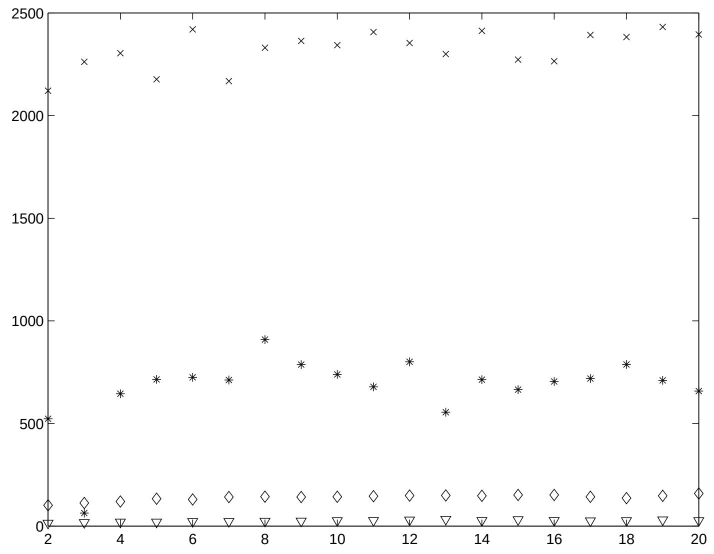


Fig. 2.16 . The ESSS-FDM algorithm with $\eta = 20$: the mean number of epochs t_c needed to cross the saddle (a) and percentages p_c of the algorithm runs in which the global optimum was found in 3000 (b) vs. the dimension of the adaptation landscape n; ($\sigma = 0.015$: crosses, $\sigma = 0.03$: stars, $\sigma = 0.05$: diamonds, $\sigma = 0.1$: triangles).

(a)



(b)

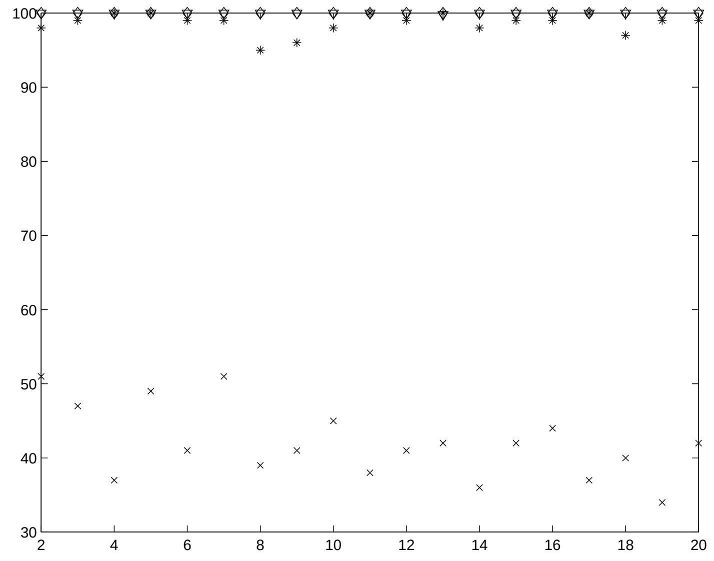
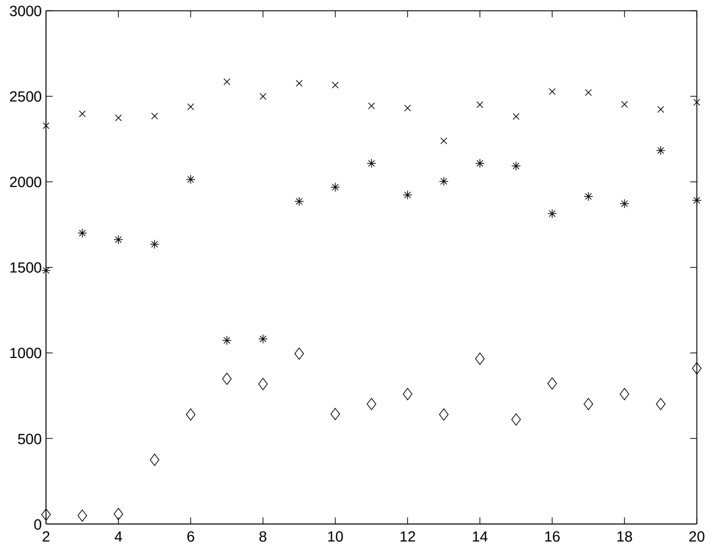


Fig. 2.17. The ESSS-FDM algorithm with $\eta = 50$: the mean number of epochs t_c needed to cross the saddle (a) and percentages p_c of the algorithm runs in which the global optimum was found in 3000 (b) vs. the dimension of the adaptation landscape n; ($\sigma = 0.015$: crosses, $\sigma = 0.03$: stars, $\sigma = 0.05$: diamonds, $\sigma = 0.1$: triangles).

(a) $\sigma = 0.015$



(b) $\sigma = 0.03$

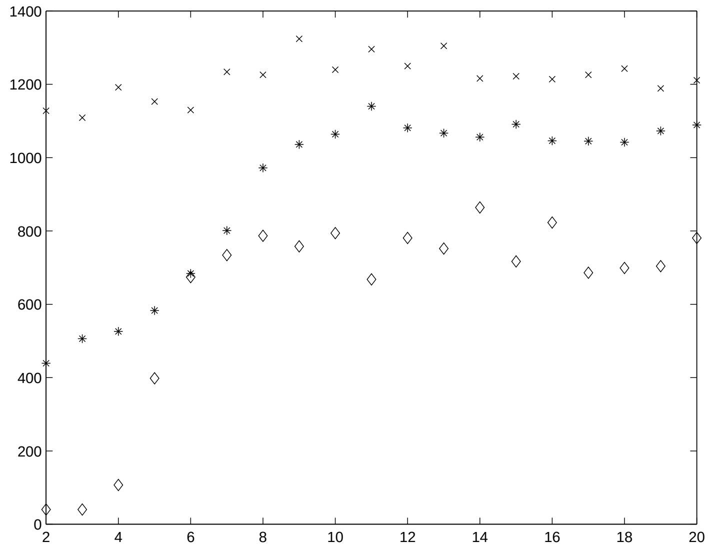
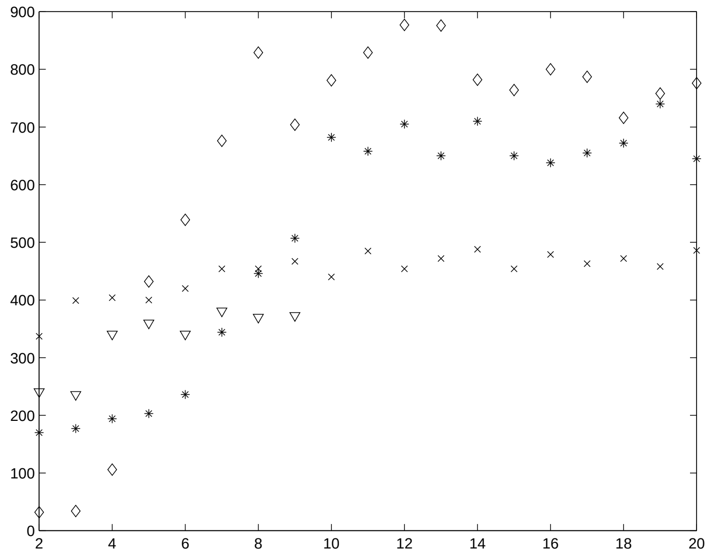


Fig. 2.18 . The mean number of epochs t_c needed to cross the saddle by the algorithms considered in the work vs. the dimension of the adaptation landscape n for $\eta = 10$: (a) $\sigma = 0.015$, (b) $\sigma = 0.03$; (ESSS: crosses, ESSS-FDM: stars, ESSS-SVA: diamonds, ESSS-DOF: triangles).

(a) $\sigma = 0.05$



(b) $\sigma = 0.1$

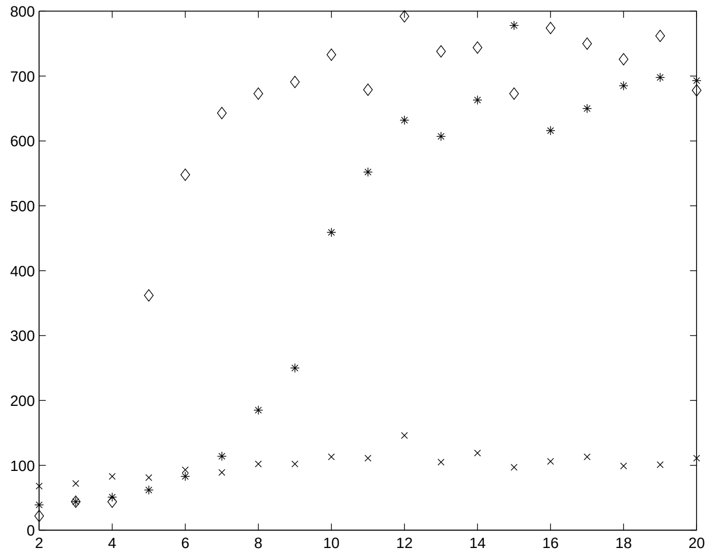
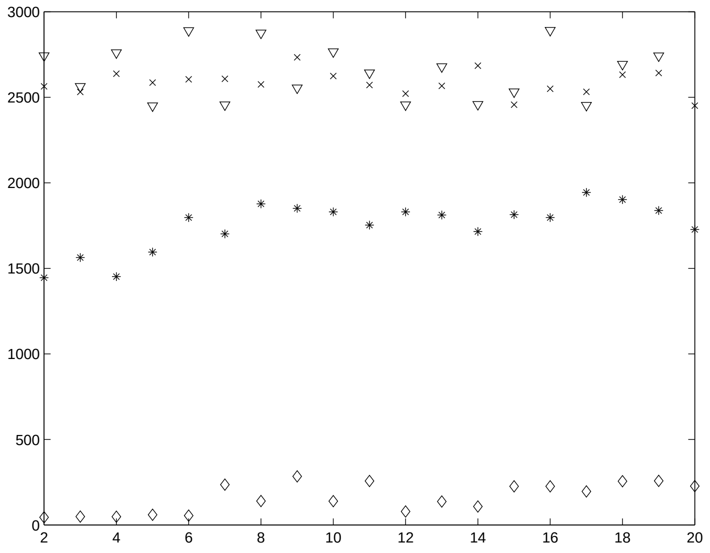


Fig. 2.19 . The mean number of epochs t_c needed to cross the saddle by the algorithms considered in the work vs. the dimension of the adaptation landscape n for $\eta = 10$: (a) $\sigma = 0.05$, (b) $\sigma = 0.1$; (ESSS: crosses, ESSS-FDM: stars, ESSS-SVA: diamonds, ESSS-DOF: triangles).

(a) $\sigma = 0.015$



(b) $\sigma = 0.03$

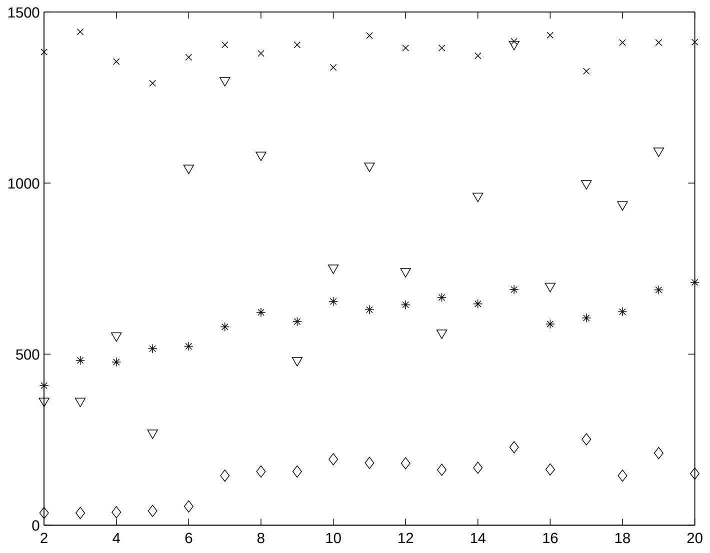


Fig. 2.20 . The mean number of epochs t_c needed to cross the saddle by the algorithms considered in the work vs. the dimension of the adaptation landscape n for $\eta = 20$: (a) $\sigma = 0.015$, (b) $\sigma = 0.03$; (ESSS: crosses, ESSS-FDM: stars, ESSS-SVA: diamonds, ESSS-DOF: triangles).

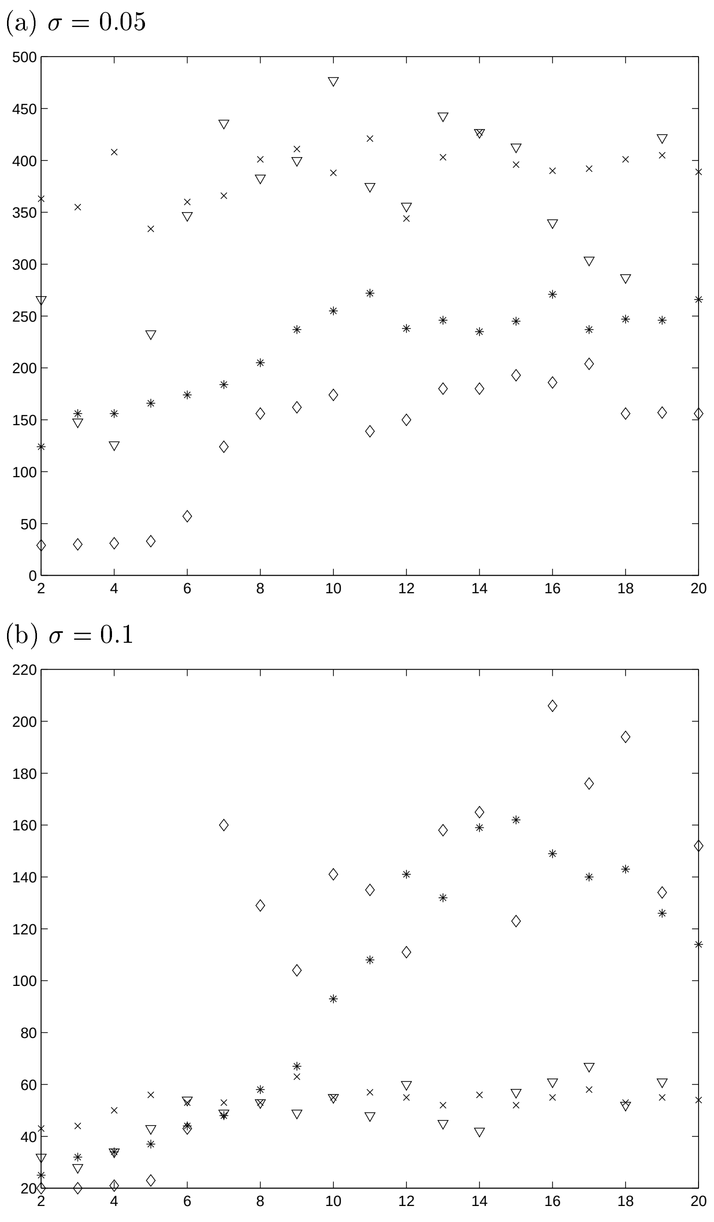
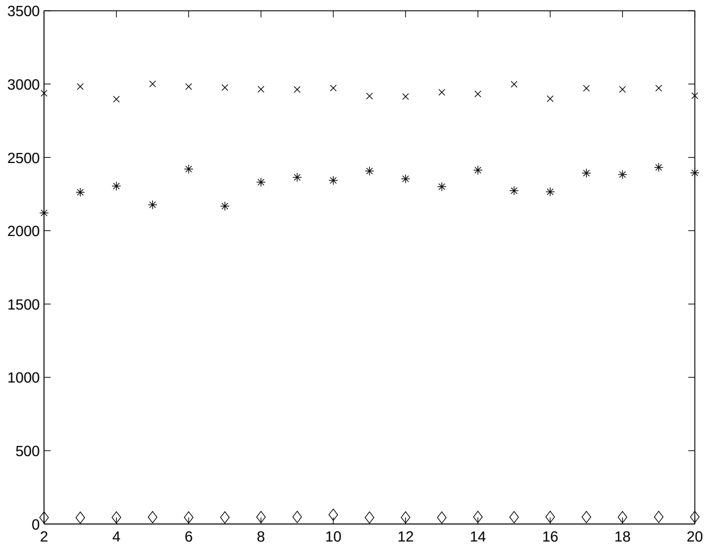


Fig. 2.21 . The mean number of epochs t_c needed to cross the saddle by the algorithms considered in the work vs. the dimension of the adaptation landscape n for $\eta = 20$: (a) $\sigma = 0.05$, (b) $\sigma = 0.1$; (ESSS: crosses, ESSS-FDM: stars, ESSS-SVA: diamonds, ESSS-DOF: triangles).

(a) $\sigma = 0.015$



(b) $\sigma = 0.03$

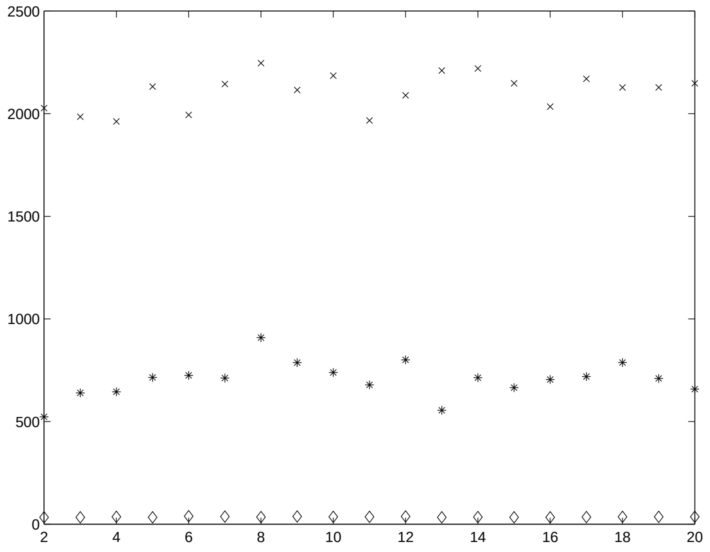
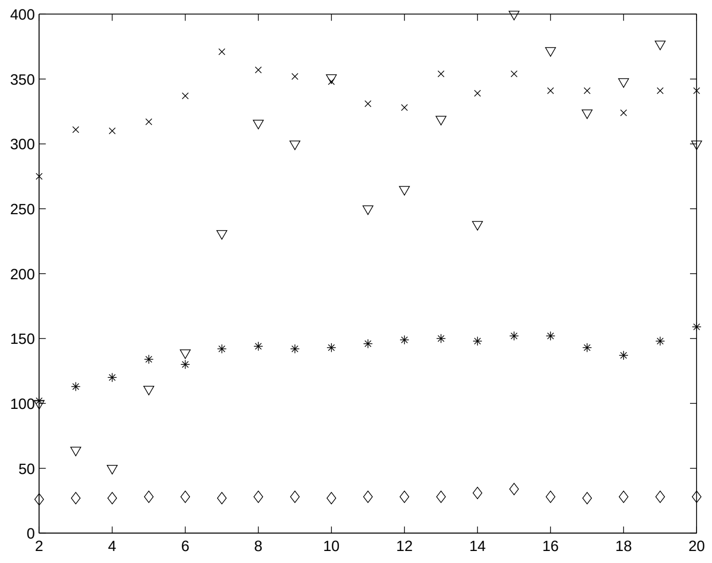


Fig. 2.22 . The mean number of epochs t_c needed to cross the saddle by the algorithms considered in the work vs. the dimension of the adaptation landscape n for $\eta = 50$: (a) $\sigma = 0.015$, (b) $\sigma = 0.03$; (ESSS: crosses, ESSS-FDM: stars, ESSS-SVA: diamonds, ESSS-DOF: triangles).

(a) $\sigma = 0.05$



(b) $\sigma = 0.1$

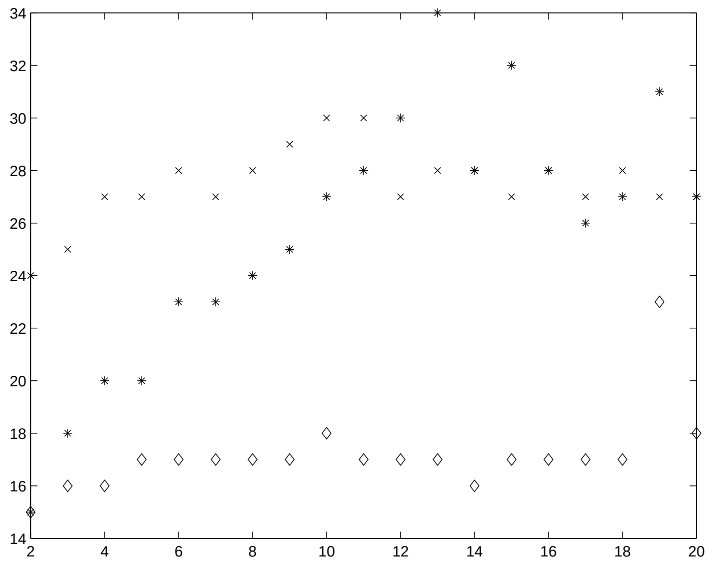
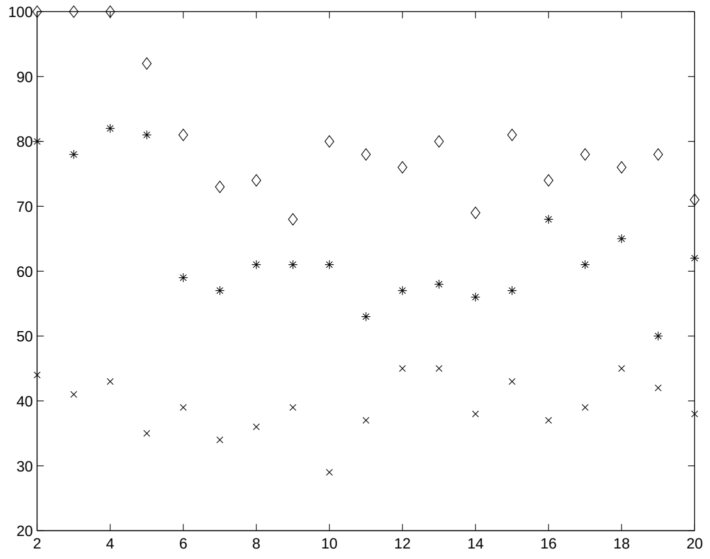


Fig. 2.23 . The mean number of epochs t_c needed to cross the saddle by the algorithms considered in the work vs. the dimension of the adaptation landscape n for $\eta = 50$: (a) $\sigma = 0.05$, (b) $\sigma = 0.1$; (ESSS: crosses, ESSS-FDM: stars, ESSS-SVA: diamonds, ESSS-DOF: triangles).

(a) $\sigma = 0.015$



(b) $\sigma = 0.03$

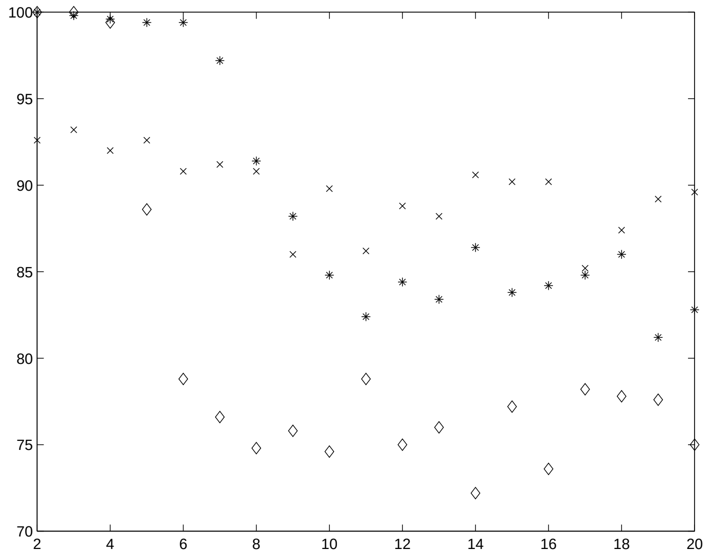
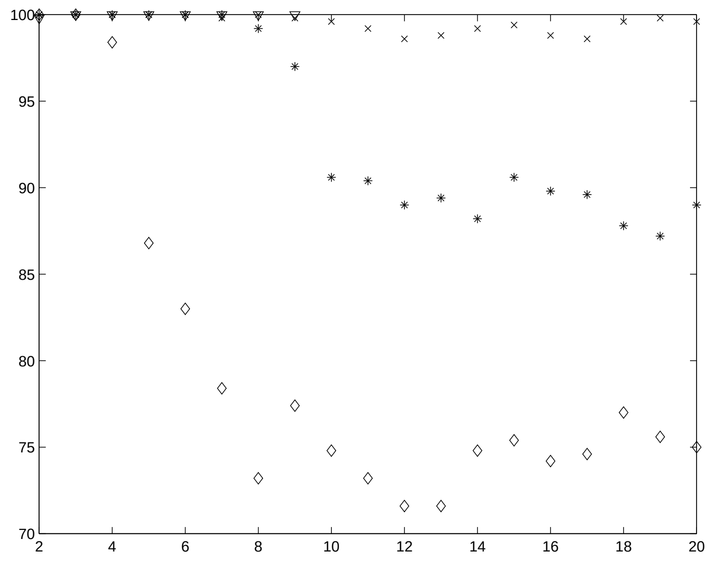


Fig. 2.24 . Percentages p_c of algorithm runs in which the global optimum was found in 3000 epochs vs. the dimension of the adaptation landscape n for $\eta = 10$: (a) $\sigma = 0.015$, (b) $\sigma = 0.03$; (ESSS: crosses, ESSS-FDM: stars, ESSS-SVA: diamonds, ESSS-DOF: triangles).

(a) $\sigma = 0.05$



(b) $\sigma = 0.1$

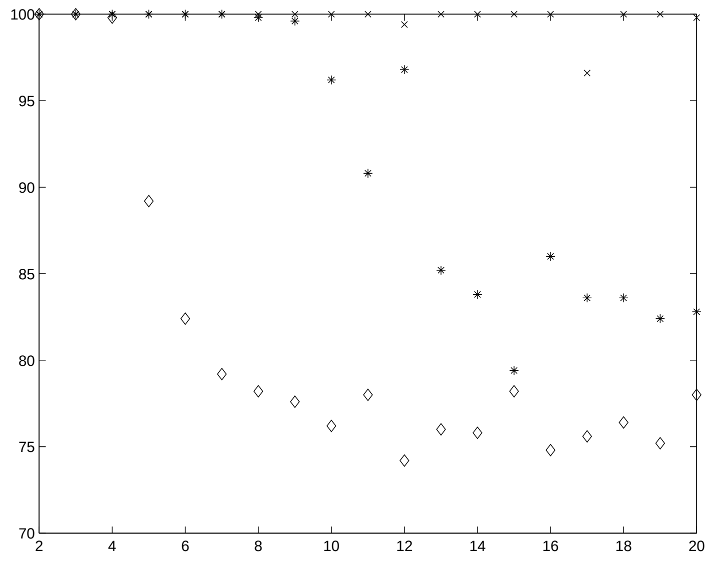
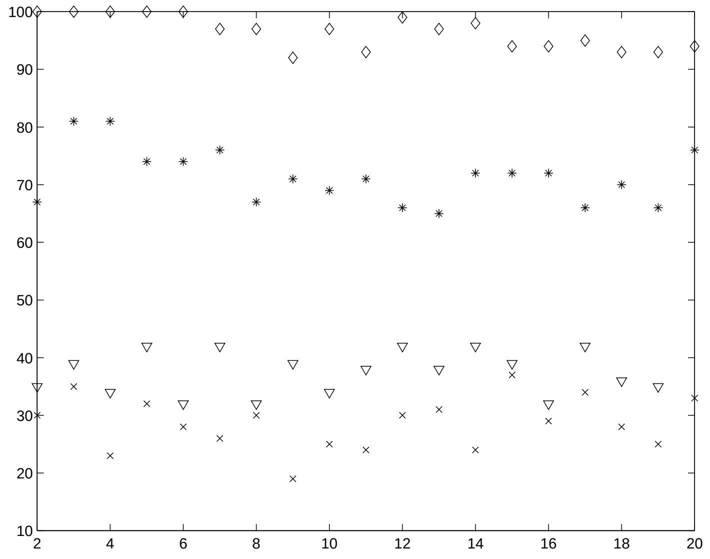


Fig. 2.25 . Percentages p_c of algorithm runs in which the global optimum was found in 3000 epochs vs. the dimension of the adaptation landscape n for $\eta = 10$: (a) $\sigma = 0.05$, (b) $\sigma = 0.1$; (ESSS: crosses, ESSS-FDM: stars, ESSS-SVA: diamonds, ESSS-DOF: triangles).

(a) $\sigma = 0.015$



(b) $\sigma = 0.03$

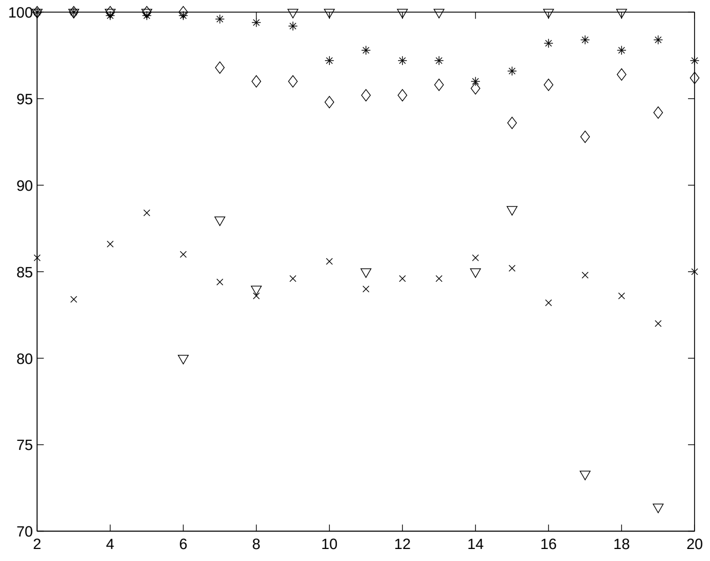
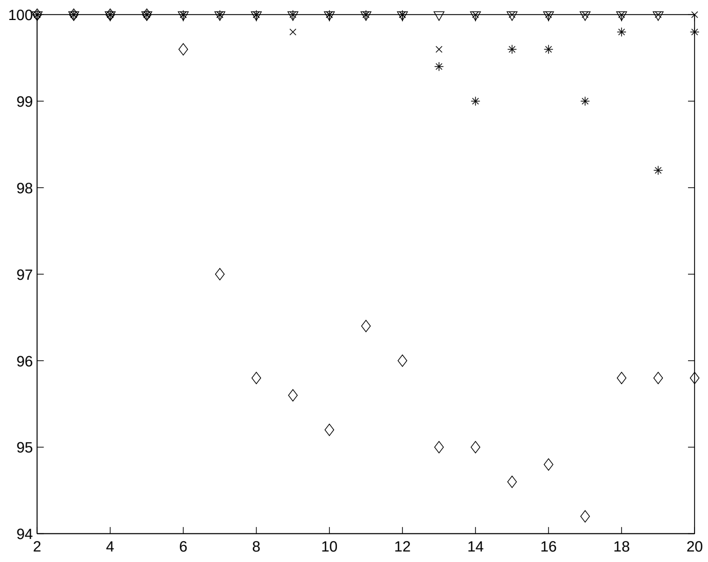


Fig. 2.26 . Percentages p_c of algorithm runs in which the global optimum was found in 3000 epochs vs. the dimension of the adaptation landscape n for $\eta = 20$: (a) $\sigma = 0.015$, (b) $\sigma = 0.03$; (ESSS: crosses, ESSS-FDM: stars, ESSS-SVA: diamonds, ESSS-DOF: triangles).

(a) $\sigma = 0.05$



(b) $\sigma = 0.1$

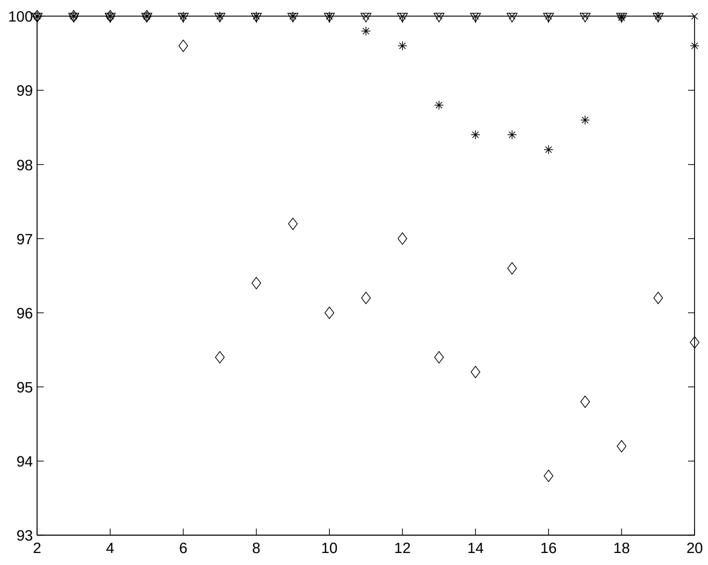
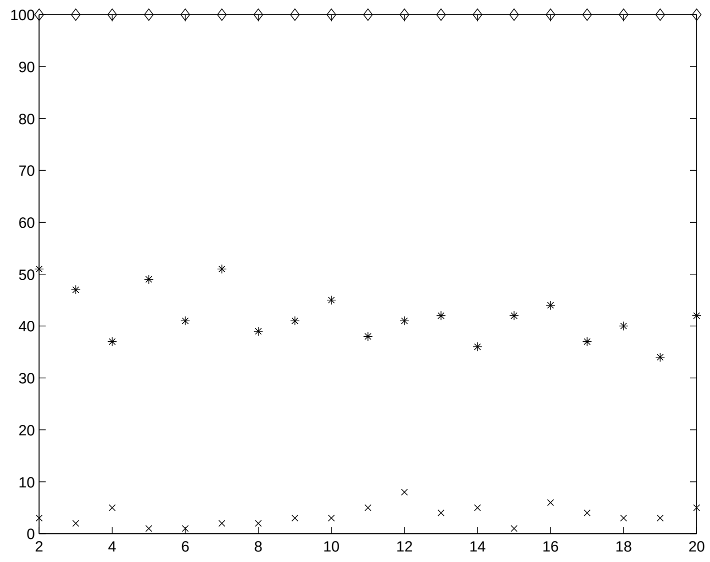


Fig. 2.27 . Percentages p_c of algorithm runs in which the global optimum was found in 3000 epochs vs. the dimension of the adaptation landscape n for $\eta = 20$: (a) $\sigma = 0.05$, (b) $\sigma = 0.1$; (ESSS: crosses, ESSS-FDM: stars, ESSS-SVA: diamonds, ESSS-DOF: triangles).

(a) $\sigma = 0.015$



(b) $\sigma = 0.03$

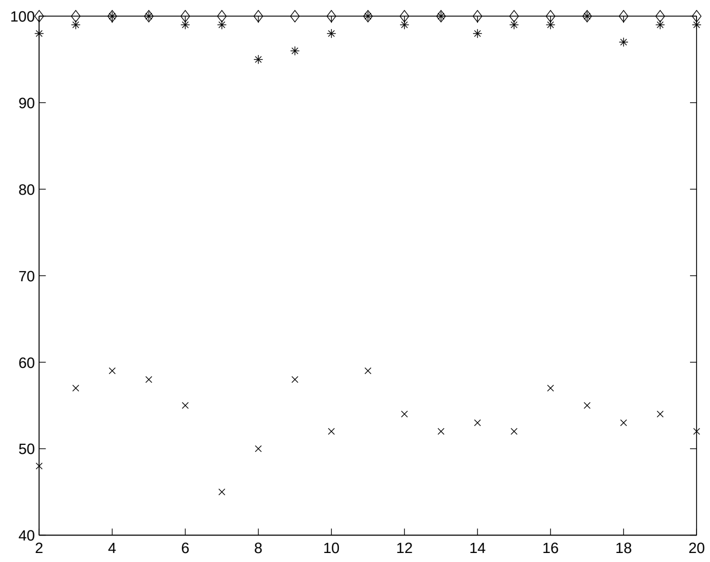
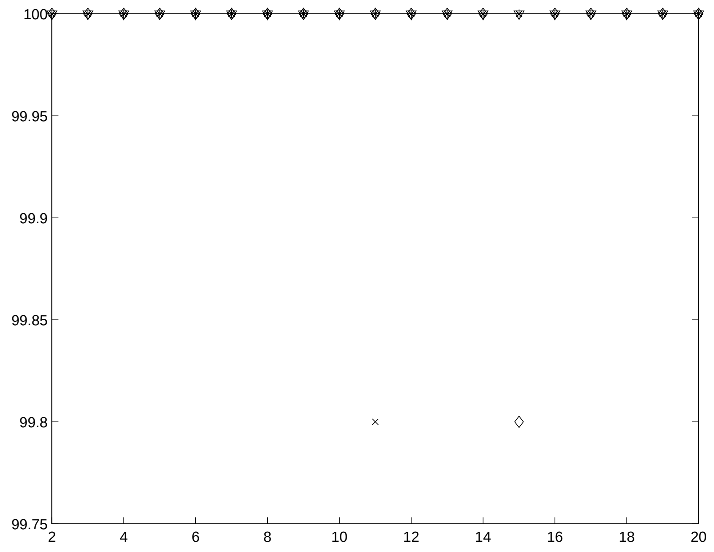


Fig. 2.28 . Percentages p_c of algorithm runs in which the global optimum was found in 3000 epochs vs. the dimension of the adaptation landscape n for $\eta = 50$: (a) $\sigma = 0.015$, (b) $\sigma = 0.03$; (ESSS: crosses, ESSS-FDM: stars, ESSS-SVA: diamonds, ESSS-DOF: triangles).

(a) $\sigma = 0.05$



(b) $\sigma = 0.1$

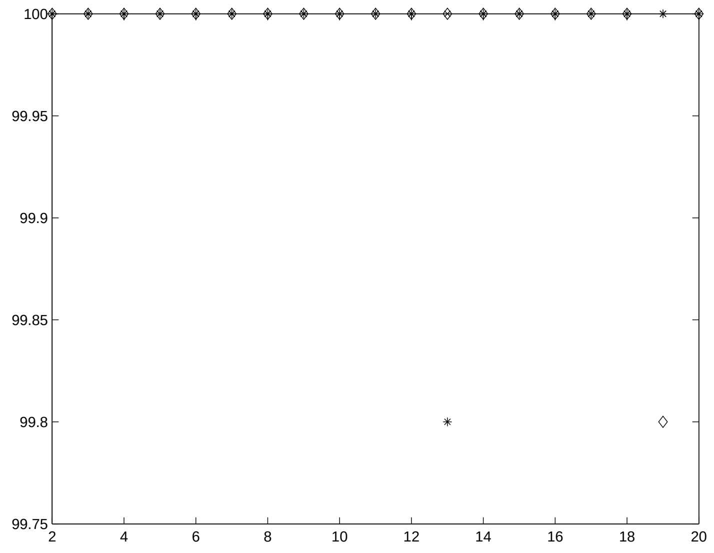


Fig. 2.29 . Percentages p_c of algorithm runs in which the global optimum was found in 3000 epochs vs. the dimension of the adaptation landscape n for $\eta = 50$: (a) $\sigma = 0.05$, (b) $\sigma = 0.1$; (ESSS: crosses, ESSS-FDM: stars, ESSS-SVA: diamonds, ESSS-DOF: triangles).

Tab.2.4. Parameter values used in the simulations

Algorithm	Parameters	Values
all	$t_{\max}$	1000
	η	20
	σ	0.05
ESSS-SVA	α	1.1
ESSS-SVA and ESSS-DOF	t_t	10
ESSS-FDM	μ	0.3

2.4.3. Optimization of the chosen multi-dimensional functions

2.4.3.1. Experiment

Many simulations (about 1600) with eight two-variable objective functions were carried out. Test functions used during simulations are listed below:

- function f_1 (sum of two Gaussian peaks) (B1),
- function f_2 (De Jong's function F2) (B2),
- function f_3 (De Jong's function F5) (B3),
- function f_4 (the „drop wave" function) (B4),
- function f_5 (Michalewicz's function) (B5),
- function f_6 (Shubert's function) (B6),
- function f_7 (Rastringin's function) (B7),
- function f_8 (Acley's function) (B8)

All those functions have to be maximized and are strongly non-linear and multimodal. The fitness function was chosen in the form (2.3), where $\eta^t = \eta = const$.

At first, simulations were carried out several times for different sets of input parameters. When the best set of parameters was allocated for each algorithm (see Table 2.4), several starting points were tested.

2.4.3.2. Results

The results are compared in Table 2.5. Analysis of the results shows that all mechanisms (SVA, FDM and DOF) applied to the standard ESSS algorithm accelerate the crossing of the objective function saddles and increase the effectiveness of the global optimum finding. Two algorithms, ESSS-SVA and ESSS-DOF, compete

Tab.2.5. Percentages of runs in which the global optimum was found

function	f_1	f_2	f_3	f_4	f_5	f_6	f_7	f_8
ESSS	38	53	0	0	12	22	26	0
ESSS-SVA	100	88	42	37	27	98	59	69
ESSS-FDM	87	79	58	0	13	81	74	0
ESSS-DOF	100	100	0	0	41	39	23	13

to be the best. ESSS-SVA seems to be the most effective algorithm. It wins with other algorithms in the case of almost all tested objective functions. But ESSS-DOF wins in the case of a fitness function which consists of a group of concentrated local optimum peaks and other distant peaks (Michalewicz's function – f_5). If the population in ESSS-SVA starts in the area of this local group, it cyclically moves from peak to peak of the group and cannot achieve a remote one. ESSS-DOF erodes peaks in turn and slowly, but consequently, leads toward the global optimum.

2.5. Summary

Techniques of exploration in the ESSS algoritm can be divided into three classes: techniques which adapt algorithm parameters (the ESSS-SVA and ESSS-VPS), methods which modify evolutionary operators (the ESSS-FDM, ESSS-LS, ESSS-MS and ESSS-ALS) and which modify the fitness function (the ESSS-IP and ESSS-DOF).

However, most of proposed methods possess their equivalents in the literature, there are some new proposals. First of all the trap-test procedure (the impatience mechanism) is an original idea. This procedure decides on turning off or on the exploration mechanism. This decision is dependent on the actual state of the evolutionary process. The forced direction of mutation technique (ESSS-FDM) is an original method which has not an equivalent in the literature. However, the idea of the erosion technique can be found in (Beasley *et al.* 1993), the wide simulation analysis of the proposed ESSS-DOF algorithm is firstly included in this book.

The results of three types of comparison experiments were presented in this work.

The aim of first experiment was to analyze a local selection mechanism, which seems to be specific to natural selection, in the global parameter optimization. Three variants of the local selection were implemented in the standard ESSS algorithm: ESSS-LS, ESSS-MS and ESSS-ALS, and were compared with the ESSS in the saddle crossing problem and two 2D optimization problems: the Schwefel's problem 2.22 and the Rastringin's function. Simulation experiments show that the 'local' selection mechanisms are effective only for low landscape dimensions (in comparison to the size of population). The ESSS-ALS algorithm possesses the highest exploration ability. Although, the local selection mechanism rather decreases effectiveness of the evolutionary search in an optimum allocation in the case of unimodal Schwefel problem, it accelerates the exploration rate of the algorithm, which is helpful in global optimum searching of the Rastringin's function. Especially, the ESSS-MS and ESSS-ALS algorithms, which join the good exploitation rate of the ESSS and the exploration rate of the ESSS-LS, seem to be good tools for technical applications. Proposed method is close to the idea of diffusion model. In the diffusion model, the number of competitors is constant and equal for each individual in the base population. Here, some kind of niching is used. The questions arise: is the ball-shaped neighbourhood the most appropriate for all possible fitness functions (e.g., one might expect a negative answer when the shapes of the attraction around the local maxima are highly deformed ellipsoids) and what is the nature of the *transition effect* which is still incomprehensible.

The performance analysis of the algorithms mentioned above for the problem of multi-dimensional saddle crossing was the subject of the second experiment. Emphasis is put on the relation between the effectiveness of the algorithm and the dimension of the adaptation landscape. Simulation results reveal that all modified algorithms are usually better than the standard ones. In the case of a low population size, the performance of ESSS-FDM, ESSS-SVA, and ESSS-DOF became worse for an increasing standard deviation of mutation and was lost with the standard ESSS algorithm. It is worth noticing that in the case of the ESSS-DOF algorithm the population has to be a simplex, i.e. the size of the population has to be larger than the problem dimension.

The aim of the third experiment was the effectiveness analysis of the algorithms in the global parameter optimization. All modified algorithms are more effective than standard ESSS. Especially, ESSS-SVA and ESSS-DOF seem to be useful in technical applications.

Chapter 3

MULTI-DIMENSIONAL MUTATIONS IN EAS BASED ON REAL-VALUED REPRESENTATION

Most applications of evolutionary algorithms (EAs), which use the floating point representation of population individuals, use the Gaussian mutation as a mutation operator (Bäck and Schwefel 1993, Fogel *et al.* 1966, Fogel 1994, Galar 1985, Michalewicz 1996, Rechenberg 1965). A new individual $\boldsymbol{x}$ is obtained by adding a normally distributed random value to each entry of a selected parent $\boldsymbol{y}$:

$$x_i = y_i + N(0, \sigma_i), \quad i = 1, \dots, n. \tag{3.1}$$

The choice is usually justified by the central limit theorem. Mutations in nature are caused by a variety of physical and chemical influences that are not identifiable or measurable. These influences are considered as independent and identically distributed (i.i.d.) random perturbations whose normed sum approaches a Gaussian random variable in the limit (Rudolph 1997). If the Lindeberg condition is obeyed, i.e. the first two absolute moments are finite, then the Gaussian distribution is the only limit distribution for normed sums of i.i.d. random variables. Taking into consideration also other distributions, which have finite absolute moments $\beta (0 < \beta < 2)$, the limit distribution for normed i.i.d. variables may be generally expressed as (Gutowski 2001, Mantegna and Stanley 1994):

$$L(x) = \frac{1}{\pi} \int_0^\infty \exp\big(-\gamma q^\beta\big) \cos qx dq, \tag{3.2}$$

and is known as *the symmetrical Lévy stable distribution of index β and scale factor $\gamma (\gamma > 0)$*. The special case of (3.2) for $\beta = 1$ (and $\gamma = 1$ for simplicity) is the Cauchy distribution with the probability density function (pdf) in the form

$$g(x) = \frac{1}{\pi} \frac{\tau}{\tau^2 + (x-u)^2}. \tag{3.3}$$

While the univariate Cauchy distribution has a unique definition, there exist at least two multivariate versions of the Cauchy distribution: the spherically symmetric Cauchy distribution (Obuchowicz 2001b, Shu and Hartley 1987), and the

Cauchy distribution with independent univariate Cauchy random variables in each dimension. In recent years, the latter Cauchy mutation has been successfully applied in the various evolutionary algorithms (Bäck *et al.* 1997, Kappler 1998, Yao and Liu 1996, Yao and Liu 1997, Yao and Liu 1999). In these cases, the normally distributed random value $N(0, \sigma_i)$ (3.1) is substituted by a random variable of the one-dimensional Cauchy distribution. The Cauchy pdf shape resembles that of the Gaussian one, but it approaches the axis very slowly, increasing the probability of the so called macro-mutations and the local optimum leaving.

Rudolph (1997) analytically analyzes the local convergence of simple $(1+1)ES$ and $(1+\lambda)ES$ with Gaussian, spherical and non-spherical Cauchy mutations. It has been proved that the order of local convergence is identical for Gaussian and spherical Cauchy distributions, whereas non-spherical Cauchy mutations lead to slower local convergence. There are no comparing results of the saddle crossing ability of EAs with spherical and non-spherical Cauchy mutations in the literature. The influence of the choice of the reference frame on the effectiveness of EAs in global optimization tasks is a very important problem that should in particular be analyzed (the *symmetry effect*).

Another problem which seems to be imperceptible by the researches is related to the probability that the distance from the mutated point $\boldsymbol{x}$ and its offspring $\boldsymbol{y}$ will be in the range $\|\boldsymbol{x}-\boldsymbol{y}\| \in [r, r+dr]$. Although the pdfs of multivariate Gaussian and non-spherical Cauchy mutations of the type (3.1) have their optimum in the mutated point, it is easy to prove (Obuchowicz 2001a, Obuchowicz 2001b) that the most probable location of the offspring is the nearest neighbourhood of the parent individual only in the case of the one-dimensional mutation. In the case of n-dimensional one, the most probable location moves from the center of mutation to the "ring" of the radius proportional to the norm of the standard deviation vector of mutation and to $\sqrt{n-1}$ (the *surrounding effect*).

The aim of this chapter is to present the results of simulation experiments which compare the effectiveness of evolutionary algorithms with multivariate Gaussian and Cauchy mutations (Obuchowicz 2003b). Four types of mutations are considered, namely, spherical and non-spherical Cauchy mutations, and the Gaussian mutation in its classical form (3.1) and in the new form, in which spherically symmetric random vector is decomposed on the uniformly distributed random direction and a normally distributed random radius. It is important to note that the *surrounding effect* does not obligate in the case of multivariate spherical Cauchy and modified Gaussian distributions. Implemented EAs are based on two types of evolutionary models: the ESSS (Tab. 1.3) and EP (Tab. 1.2). The main difference between these two types of EAs is that the standard deviation of mutation is adapted in EP but not in ESSS. Thus, it is possible to analyze whether the adapted standard deviation reduces the *surrounding effect* or not.

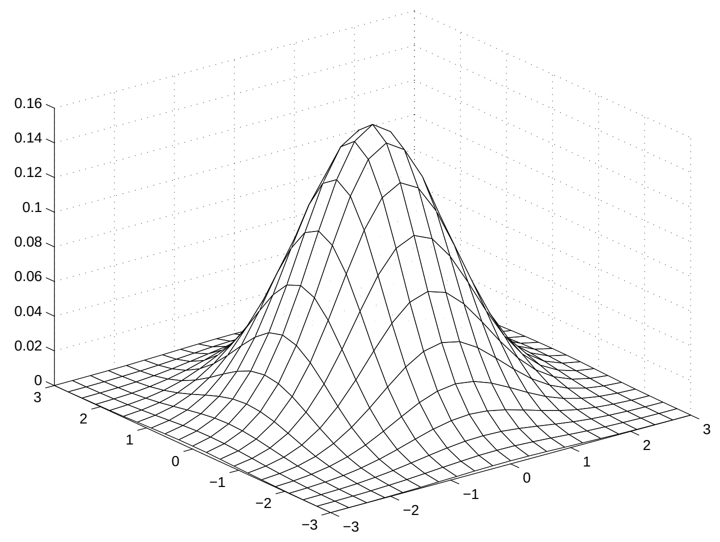


Fig. 3.1 . Two-dimensional Gaussian density function, $\sigma = 1$.

3.1. Surrounding and symmetry effects

A mutation function which describes the transformation of an element $\boldsymbol{y}$ into $\boldsymbol{x}$ in accordance with the Gaussian mutation (3.1) has the form

$$g_G(\boldsymbol{x}-\boldsymbol{y}) = \prod_{i=1}^{n} \frac{1}{\sqrt{2\pi}\sigma_i} \exp\left(-\frac{(x_i-y_i)^2}{2\sigma_i^2}\right). \tag{3.4}$$

Let us scale the reference frame using σ_i $(i = 1, 2, \ldots, n)$ as a unit length in the ith direction; then the length of the vector

$$\boldsymbol{r} = \left\{ r_i = \frac{x_i - y_i}{\sigma_i} \middle| i = 1, 2, \ldots, n \right\} \tag{3.5}$$

can represent the distance between the base and offspring vectors. Thus, the equation (3.4) has the following form:

$$g_G(\boldsymbol{x}-\boldsymbol{y}) = \left(\frac{1}{\sqrt{2\pi}\sigma}\right)^n \exp\left(-\frac{1}{2}r^2\right), \tag{3.6}$$

where $r = \|\boldsymbol{r}\|$ and $\sigma = \left(\prod_{i=1}^{n} \sigma_i\right)^{1/n}$. The two-dimensional version of (3.6) with $\boldsymbol{y} = 0$ and $\sigma = 1$ is presented in Fig. 3.1.

The probability dP_G that the point obtained after mutation will be located in the volume $([x_i, x_i + dx_i]^n | i = 1, \ldots, n)$ is equal to

$$dP_G = g_G(\boldsymbol{x}-\boldsymbol{y})d\omega, \tag{3.7}$$

where $d\omega = dx_1 dx_2 \ldots dx_n$.

Let us consider the relationship between the probability dP_G and the distance between the base and mutated points. In order to do it, the n-spherical reference frame with the origin in the base point $\boldsymbol{y}$ will be introduced. Transformation equations have the following forms:

$$\begin{aligned} r_1 &= r\cos(\alpha_1), \\ r_2 &= r\sin(\alpha_1)\cos(\alpha_2), \\ r_3 &= r\sin(\alpha_1)\sin(\alpha_2)\cos(\alpha_3), \\ &\dots \\ r_n &= r\sin(\alpha_1)\sin(\alpha_2)\dots\sin(\alpha_{n-1}), \end{aligned} \tag{3.8}$$

where $\alpha_{n-1} \in [0, 2\pi)$ and $(\alpha_i \in [0,\pi)|i = 1,\dots,n-2)$.

Using (3.8) in (3.7), the probability dP_G can be obtained from the equation

$$dP_G = \left(\frac{1}{\sqrt{2\pi}}\right)^n \exp\left(-\frac{1}{2}r^2\right) r^{n-1} dr d\Omega = g_G(r) dr d\Omega, \tag{3.9}$$

where $d\Omega = \prod_{i=1}^{n-1}\left(\sin^{n-(i+1)}(\alpha_i)d\alpha_i\right)$ is the infinitely small n-dimensional solid angle. Due to non-negative values of the radius and the sinus function in the interval $[0,\pi)$, the magnitude operator of the transformation Jacobian can be omitted. It is very interesting that for a small r the value of the probability dP_G is low. The most probable distance $r^\star$ is not equal to 0 but

$$r^\star(n) = \arg\max g_G(r) = \sqrt{n-1} \tag{3.10}$$

and $r^\star(n) \to \infty$ as $n \to \infty$. Therefore, in the case of the n-dimensional ($n \geq 2$) Gaussian mutation, the probability that the offspring will be located closely to its parent is low and decreases with n. This fact influences the exploitation effectiveness of EAs in the case of large landscape dimensions (the *surrounding effect*).

In order to confirm the above results, a simulation experiment is done. 10^6 points are generated in accordance to (3.1) for dimensions $n = 2, 3, 4, 5$, $\sigma_1 = \sigma_2 = \cdots = \sigma_n = 1$ for each one-dimensional mutation and the base point $\boldsymbol{y} = 0$. Histograms of the distances between the base and points mutated according to (3.1) (Fig. 3.2a) show that the probability of point location in the nearest neighbourhood of the base point is low and decreases with n. Maximum points of histograms are obtained in $r^\star(n) = \sqrt{n-1}$.

The presented effect of the multi-dimensional Gaussian mutation is caused by the following fact. The volume of the subspace $d\omega' = dr d\alpha_1 d\alpha_2 \dots d\alpha_{n-1}$ depends on the radius r: $dV = dSdr \sim r^{n-1}dr$, where dS is the area of the n-dimensional spherical sector (Fig. 3.3). The probability that the mutated point $\boldsymbol{x} \in d\omega'$ is proportional to the probability density $g_G(r)$ in this subspace and to the dV. Fig. 3.4 illustrates the result of this product in the 2D landscape.

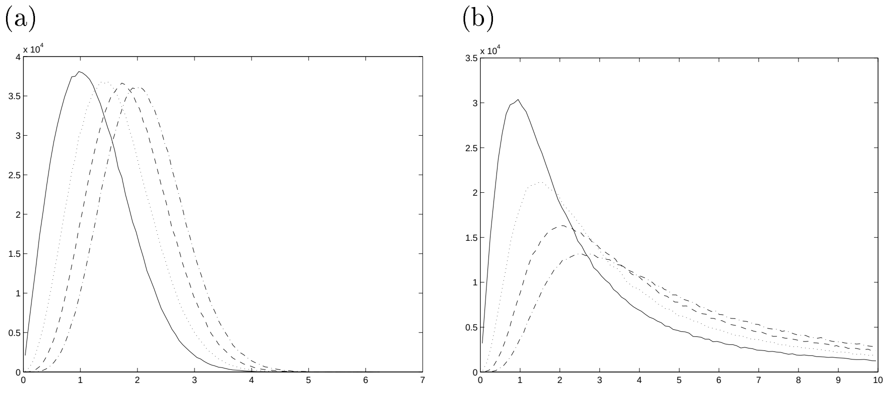


Fig. 3.2. Histograms of the distances between the base and 10^6 points mutated according to Gaussian (a) and non-spherical Cauchy (b) mutations; $n = 2$ — solid line, $n = 3$ — dotted line, $n = 4$ — dashed line, $n = 5$ — dash-dotted line, other characteristics in the text.

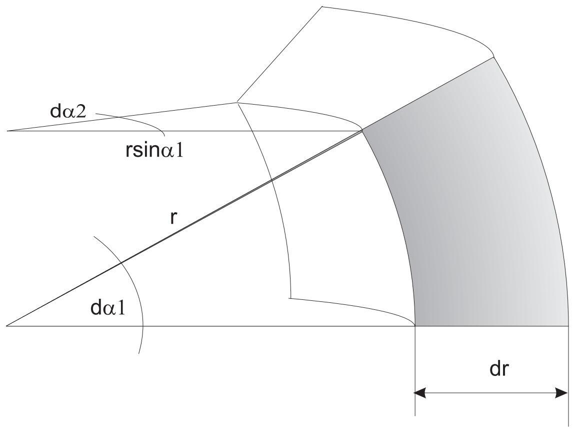


Fig. 3.3. The subspace $d\omega'$ in 3D landscape. It is ease to calculate that $dV = r^2 \sin(\alpha_1) dr d\alpha_1 d\alpha_2$.

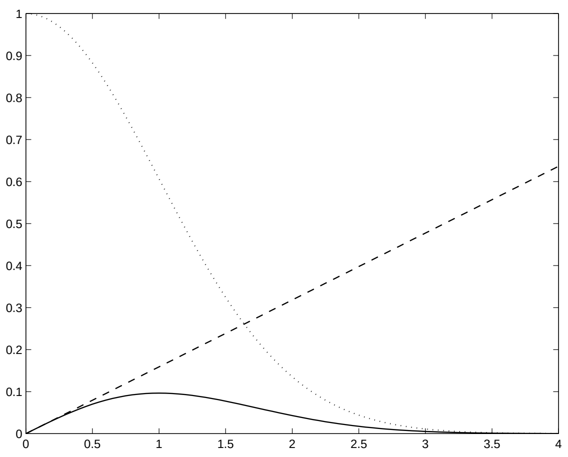


Fig. 3.4. Linear increase of perimeter with r (dashed line) and the Gaussian probability density function (dotted line) as factors creating the relationship between the probability that $\|\boldsymbol{x}-\boldsymbol{y}\| \in [r, r+dr]$ vs. r (solid line) in the case of the two-dimensional landscape,($\sigma = 1$).

Similar results are obtained in the case of the multi-variate non-spherical Cauchy mutation (Fig. 3.2b). Here, a new individual $\boldsymbol{x}$ is obtained by adding a random value to each entry of a selected parent $\boldsymbol{y}$:

$$x_i = y_i + C(0, \tau_i), \quad i = 1, \ldots, n, \tag{3.11}$$

where $C(u, \tau)$ is a random value obtained according to the one-dimensional Cauchy mutation with the pdf defined by (3.3). The shape of the one-dimensional Cauchy pdf is centered at u and resembles that of the Gaussian density function, but approaches the axis so slowly that the variance is infinite and an expectation does not exist. The comparison between one-dimensional Cauchy and Gaussian density functions is presented in Fig. 3.5a. A mutation function describing the transformation of the vector $\boldsymbol{y}$ into $\boldsymbol{x}$ in accordance with the multi-dimensional non-spherical Cauchy mutation has the form

$$g_C(\boldsymbol{x}-\boldsymbol{y}) = \pi^{-n} \prod_{i=1}^{n} \frac{\tau_i}{\tau_i^2 + (x_i - y_i)^2}. \tag{3.12}$$

The Cauchy mutation of type (3.12) has a non-spherical symmetry (Fig. 3.5b) and prefers directions parallel to the axis of the reference frame. Therefore the effectiveness of evolutionary algorithms, which uses the mutation described by (3.12), depends on the choice of the reference frame (the *symmetry effect*).

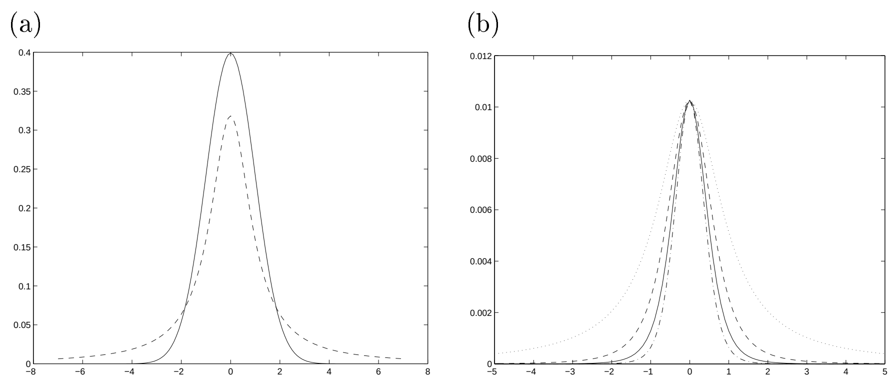


Fig. 3.5. (a) One-dimensional Gaussian (solid line) and Cauchy (dashed line) probability density functions ($\sigma = 1$, $\tau = 1$ and $u = 0$). (b) Four one-dimensional sections of the four-dimensional Cauchy density function, along the directions [1,0,0,0] (dotted line), [1,1,0,0] (dashed line), [1,1,1,0] (solid line) and [1,1,1,1] (dash-dotted line) ($\tau = 1$ and $u = 0$).

3.2. Spherically Symmetric Distributions

Fang et al. (Fang *et al.* 1990) prove that a spherically symmetric random vector $\boldsymbol{Z}$ can be decomposed via

$$\boldsymbol{Z} = r\boldsymbol{U}, \tag{3.13}$$

where $\boldsymbol{U}$ is uniformly distributed on the surface of an n-dimensional hyperball (e.i. a random direction), and r is a random variable representing the random radius of the hyperball. The vector $\boldsymbol{U}$ can be obtained from the formulae

$$\begin{aligned} U_1 &= \cos(\alpha_1), \\ U_2 &= \sin(\alpha_1)\cos(\alpha_2), \\ U_3 &= \sin(\alpha_1)\sin(\alpha_2)\cos(\alpha_3), \\ &\dots \\ U_{n-1} &= \sin(\alpha_1)\dots\sin(\alpha_{n-2})\cos(\alpha_{n-1}), \\ U_n &= \sin(\alpha_1)\dots\sin(\alpha_{n-2})\sin(\alpha_{n-1}), \end{aligned} \tag{3.14}$$

and α_{n-1} is a uniformly distributed random angle ($\alpha_{n-1} = U[0, 2\pi]$), and other angles $\{\alpha_{n-k-1} \in [0,\pi) | k = 1, \dots, n-2\}$ are randomly chosen with the pdfs

$$f(\alpha_{n-k-1}) = K_k \sin^k(\alpha_{n-k-1}), \quad k = 1, 2, \dots, n-2, \tag{3.15}$$

where

$$K_k = \frac{1}{2}\frac{3\cdot 5\cdot \ldots \cdot(2l+1)}{2\cdot 4\cdot \ldots \cdot 2l} \quad \text{for} \quad k = 2l+1,$$

$$K_k = \frac{1}{\pi}\frac{2\cdot 4\cdot \ldots \cdot 2l}{3\cdot 5\cdot \ldots \cdot(2l-1)} \quad \text{for} \quad k = 2l.$$

In order to obtain a new modified Gaussian mutation or a spherical Cauchy mutation, the random variable r can be chosen as

$$r = N(0,\sigma) \tag{3.16}$$

or

$$r = C(0,\tau), \tag{3.17}$$

respectively.

The probability dP_{NO} corresponding to the modified Gaussian mutation can be calculated from the following formulae:

$$dP_{NO} = \frac{K}{\sqrt{2\pi}\sigma}\exp\left(-\frac{r^2}{2\sigma^2}\right)\sin^{n-2}(\alpha_1)\sin^{n-3}(\alpha_2)\ldots\sin(\alpha_{n-2})d\omega', \tag{3.18}$$

and for the spherical Cauchy mutation:

$$dP_{CO} = \frac{K}{\pi}\frac{\tau}{\tau^2+r^2}\sin^{n-2}(\alpha_1)\sin^{n-3}(\alpha_2)\ldots\sin(\alpha_{n-2})d\omega', \tag{3.19}$$

where $d\omega' = dr d\alpha_1 d\alpha_2 \ldots d\alpha_{n-1}$ and

$$K = \prod_{k=1}^{n-2} K_k = \begin{cases} \frac{\left(\frac{n-2}{2}\right)!}{\pi^{\frac{n-2}{2}}}, & \text{if } n \text{ is even,} \\ \frac{(n-1)!}{\left(\frac{n-1}{2}\right)!2^n\pi^{\frac{n-1}{2}}}, & \text{if } n \text{ is odd.} \end{cases}$$

Fig. 3.6 presents histograms of distances $\|\boldsymbol{x}-\boldsymbol{y}\|$ for 10^6 points generated in accordance with both the proposed mutation and dimensions $n = 2, 7$. The probability of point location decreases with r and is independent of the landscape dimension. The same histograms can be obtained for whatever n.

3.3. Effectiveness of EA vs. mutation type: experimental studies

3.3.1. Evolutionary algorithms used in simulations

Two classes of evolutionary algorithms are used in simulation experiments. The first one is based on the ESSS algorithm, and the following four evolutionary algorithms of this class are considered:

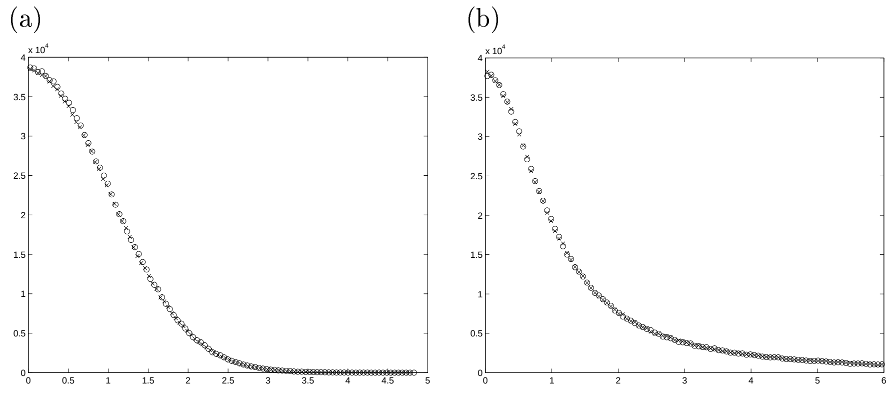


Fig. 3.6 . Histograms of the distances between the base and 10^6 mutated points generated in accordance with the modified Gaussian mutation (a) and the spherical Cauchy mutation (b); $n = 2$ — crosses, $n = 7$ — circles, $\sigma = 1$ and $\tau = 1$.

- ESSS-G : the ESSS algorithm with the standard Gaussian mutation (1.6);
- ESSS-GN : the ESSS algorithm with the Gaussian mutation and the standard deviation $\sigma/\sqrt{n}$ (2.23);
- ESSS-GO : the ESSS algorithm with the modified Gaussian mutation (3.13) (3.16);
- ESSS-C : the ESSS algorithm with the non-spherical Cauchy mutation (3.11);
- ESSS-CO : the ESSS algorithm with the spherical Cauchy mutation (3.13) (3.17).

The second class contains algorithms based on the EP algorithm

- CEP : the classical evolutionary programming algorithm with the Gaussian mutation (Fogel *et al.* 1991);
- CEPS : the CEP algorithm with the modified Gaussian mutation (3.13) (3.16);
- FEP : the fast EP algorithm with the non-spherical Cauchy mutation (3.11) (Yao and Liu 1997);
- FEPS : the FEP algorithm with the spherical Cauchy mutation (3.13) (3.17).

Apart from the different selection technique, the EP-class algorithms posses the adaptation mechanism of standard deviation of the mutation operator.

3.3.2. Local optimum allocation

Let us choose the 'quadratic function with noise' (Yao and Liu 1999) as a unimodal objective function to be minimized:

$$f_u(\boldsymbol{x}) = \sum_{i=1}^{n} i x_i^4 + random[0,1). \tag{3.20}$$

Algorithms based on the ESSS algorithm (ESSS-G, ESSS-GO, ESSS-C and ESSS-CO) localized the optimum so quickly that differences in their efficiency are on the level of a statistical error. This fact is mainly caused by the proportional selection, which prevails over mutation operators in the sense of their influence on the local optimum localization. So, the attention is focused on the four variants of the EP-class algorithms: CEP, CEPS, FEP and FEPS. The following parameters are used in the simulations: the population size $\eta = 50$, the initial area for variances $\Omega_\sigma = \prod_{i=1}^{5}[0, 0.5]$, the initial area for population $\Omega_x = \prod_{i=1}^{5}[-0.3, 0.3]$, the maximum number of epochs $t_{\max} = 5000$ and the number of sparing partners $q = 10$. Each algorithm is started 50 times.

The surrounding effect in the EP-class algorithms clearly manifests itself in this experiment. The CEPS and FEPS algorithms reach the optimum surroundings significantly faster than their classical originals. Figure 3.7b presents the epochs of the first success, i.e. the first epoch in which the objective function value of one of the population elements is lower than 0.1. For clarity of graphs, samples have been sorted. The advantage of the CEPS and FEPS algorithms over the CEP and FEP algorithms is significant. As it has been anticipated the surrounding effect increases with the landscape dimension (Fig. 3.8). In the case of the 5D landscape, average courses of all algorithms considered are similar (Fig. 3.8a). Disproportion between the pairs CEPS–FEPS and CEP–FEP enlarges in the case of the 30D landscape (Fig. 3.8b). This experiment illustrates that offspring in the CEPS and FEPS algorithms are located closely to their parents with higher probability than in the case of CEP and FEP (Fig. 3.7), for which attractiveness of the parents' surroundings decreases with the landscape dimension (Fig. 3.8).

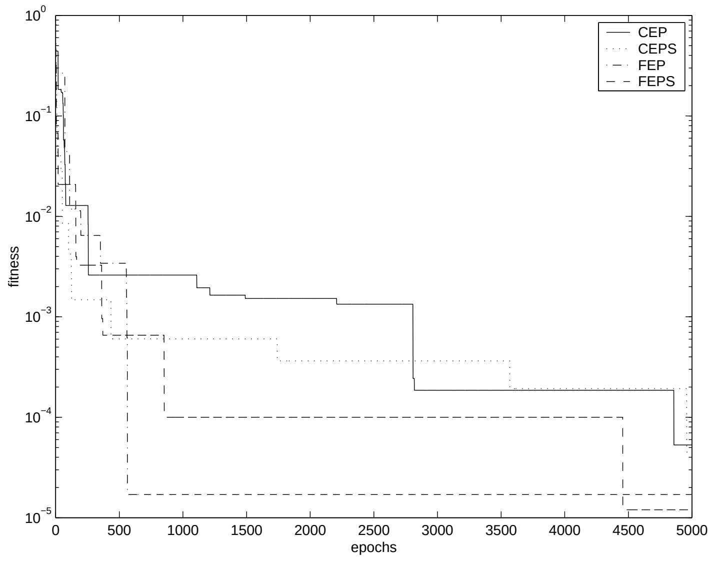
3.3.3. Sensitivity to narrow peaks

Let us consider the following five-dimensional fitness function:

$$\Phi_{np}(\boldsymbol{x}) = \frac{1}{2}\exp\left(-5\sum_{i=1}^{5} x_i^2\right) + \exp\left(-100\Big((x_1-0.4)^2 + \sum_{i=2}^{5} x_i^2\Big)\right). \tag{3.21}$$

The two-dimensional equivalent of this function is presented in Fig. 3.9. It consists of two Gaussian peaks. The first one is high and slim, the second one is low and wide. The distance between both peaks is not very large in comparison with the standard deviation of mutation chosen in the experiment and fixed as $\sigma = 0.05$. Three algorithms are tested: ESSS-G, ESSS-GN and ESSS-GO. All of them start with a population generated by an η-time mutation of an initial point

(a)



(b)

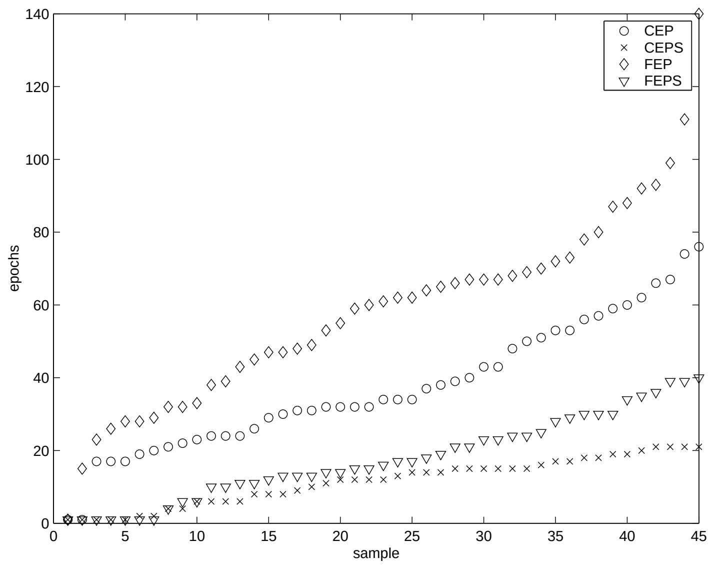
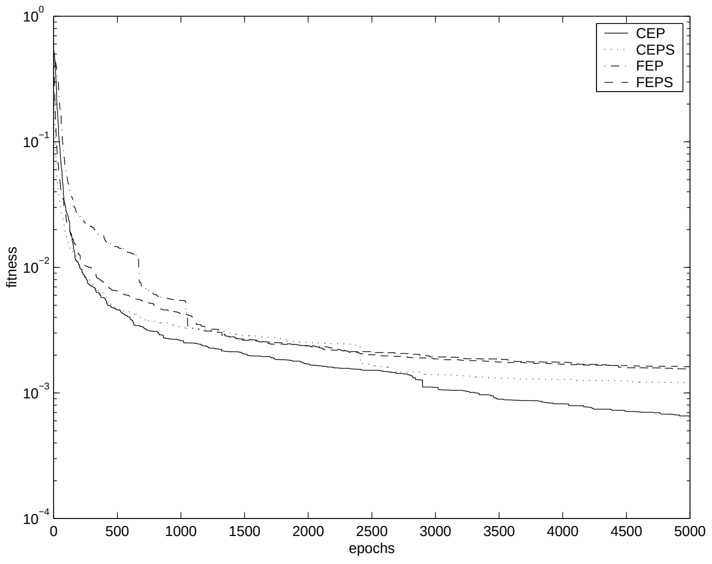


Fig. 3.7. The objective function value of the best element in the current population vs. epochs for the best realizations of the EP-class algorithms (a), and epochs of the first success ($f_u(\boldsymbol{x}) < 0.1$) for 45 runs of algorithms (sorted) (b) — results obtained for 5D version of (3.20)

(a)



(b)

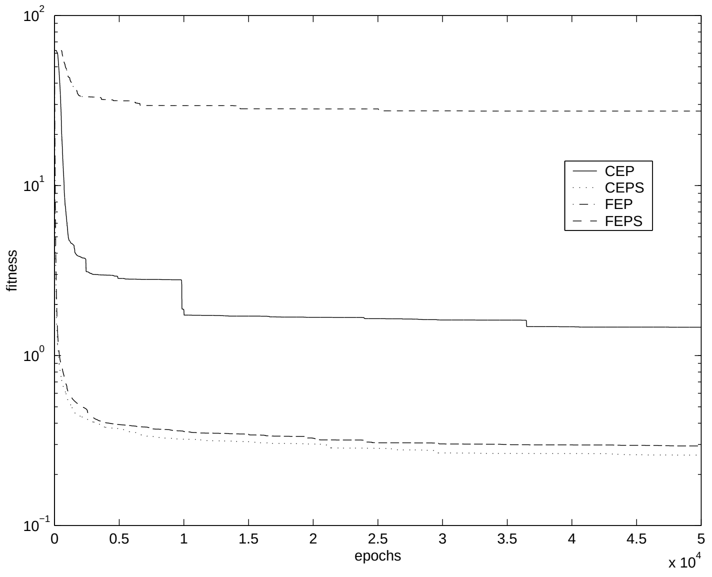


Fig. 3.8 . The objective function value of the best element in the current population vs. epochs, results averaged over 50 samples for 5D (a) and 30D (b) versions of (3.20).

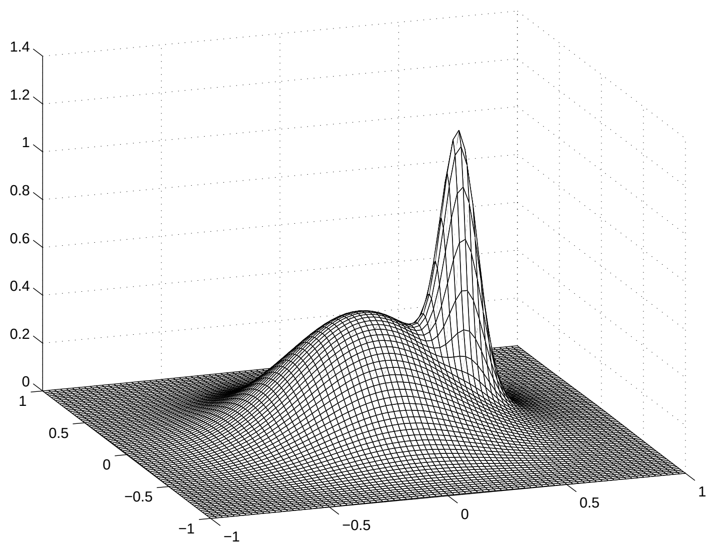


Fig. 3.9 . The two-dimensional version of the fitness function (3.21).

$\boldsymbol{x}_0^0 = [-1, -1, -1, -1, -1]$. Other algorithm parameters are chosen as follows: the population size $\eta = 20$, the maximum number of epochs $t_{\max} = 2000$.

All algorithms are processed 500 times. Typical algorithm proceedings are illustrated in Fig. 3.10. The standard ESSS-G algorithm has trouble finding the global optimum. If there is a dominant element in the population located in the narrow peak, its successors are generated outside this peak. It follows from the fact that the most probable distance between parent and successor elements $r^\star = \sigma\sqrt{n-1} = 0.1$ (3.10) is of the same order as v_h. In the case of the ESSS-GN algorithm, $r^\star = \sigma\sqrt{(n-1)/n} \approx 0.045$. The evolved population does not locate an element in the higher peak so easily as in the case of the ESSS-G algorithm. But if it does, this peak is occupied for a short time. The ESSS-GO algorithm is most effective on the narrow peak localization. It finds it quickly and does not lose it.

The presented numerical experiment shows that the standard Gaussian mutation decreases the evolutionary algorithm's sensitivity on narrow peaks. It is caused by the surrounding effect. The evolutionary algorithm with the proposed modified Gaussian mutation ends in full success.

In order to analyze the efficiency of the evolutionary algorithm, which adapts its mutation parameters during its processing, the four variants of the EP algorithm are tested: CEP, CEPS, FEP and FEPS. The following parameters are used in the simulations: the population size $\eta = 50$, the initial area for variances $\Omega_\sigma = \prod_{i=1}^{5}[0, 0.5]$, the initial area for population $\Omega_x = \prod_{i=1}^{5}[-0.2, 0.2]$, the maximum number of epochs $t_{\max} = 2000$ and the number of sparing partners $q = 10$. Each algorithm is started 500 times. Figure 3.11 presents the mean fitness of the

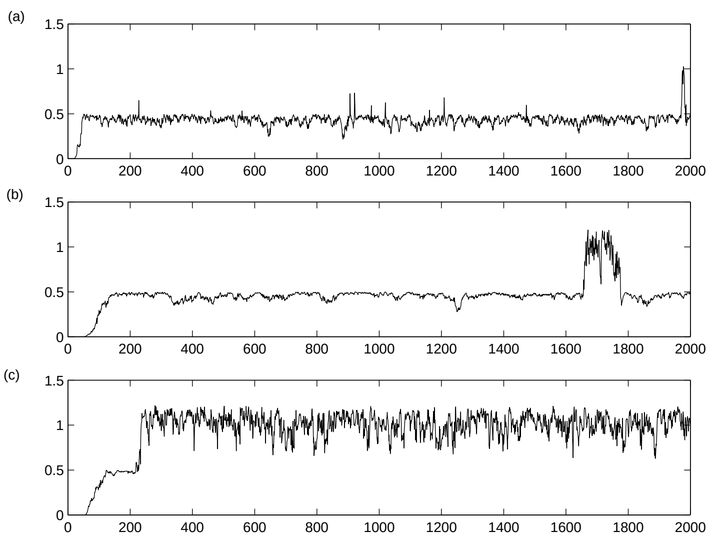
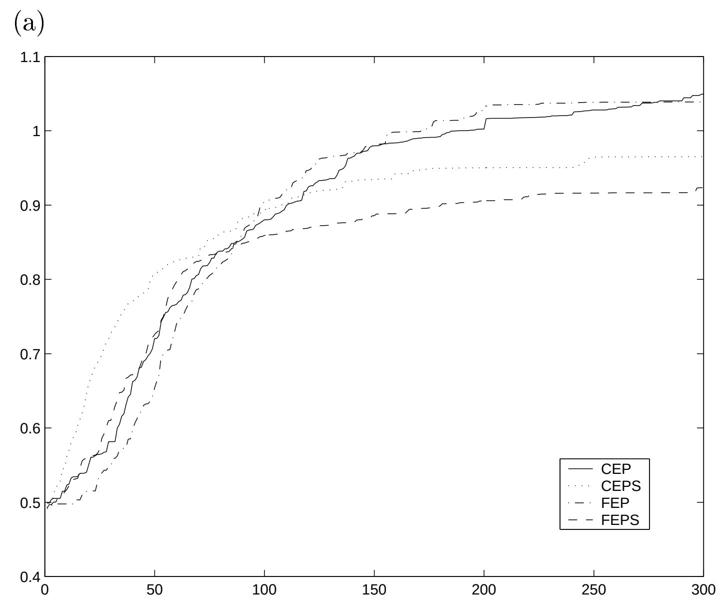


Fig. 3.10 . The fitness (3.21) of the best element in the population vs. epochs; ESSS-G (a), ESSS-GN (b) and ESSS-GO (c).

Tab.3.1. Percentages of successful algorithm runs

algorithm	successes [%]
CEP	74
CEPS	64
FEP	78
FEPS	60

best element in the population vs. epochs. Unlike the ESSS-class algorithms, selection in the case of EP keeps the best elements from generation to generation. This is the reason for the monotonic character of the curves in Fig. 3.11, which presents the relation between the fitness of the best element in the population and epochs. The population fluctuates around the best elements and only a single macro-mutation can put offspring in the area of the higher peak. So, there are many algorithm runs which end without success in $t_{\max} = 2000$ epochs, as against to the ESSS-GO algorithm, which has found global optimum in all tests. In the case of the fitness function (3.21), the percentages of successful runs of the algorithms under consideration are described in Tab. 3.1. It can be seen that the CEPS and FEPS algorithms, which use the spherical mutation (3.13) are less effective than the CEP and FEP algorithms. This fact suggests that in the case of the mutation (3.13) a new formulae for adaptation mechanism is needed instead



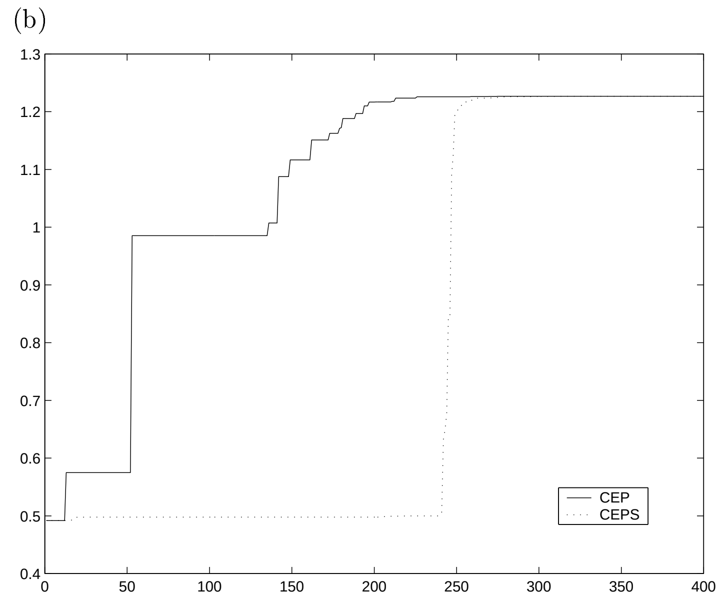


Fig. 3.11 . The mean fitness (3.21) of the best element in the population vs. epochs for four variants of the EP algorithm (a) and typical realizations of the CEP and CEPS algorithms (b).

of that presented in Table 1.2. The same conclusion is obtained via the theoretical consideration by Rudolph (1997). The surrounding effect manifests itself only in the character of the curve slopes in Fig. 3.11b. In the case of the CEPS algorithm, when some element of population is located in the narrow global peak, the extreme point is localized very quickly. Descendants of the element considered are generated closely to their parent. In the case of the CEP algorithm, this process proceeds more slowly. Most of the descendants are located outside the narrow global peak.

3.3.4. Efficiency of saddle crossing

In order to analyze the mutation influence on the saddle crossing efficiency, let us consider the problem described in Appendix A.

Firstly, let us consider the ESSS-G, ESSS-GN, ESSS-GO, ESSS-C and ESSS-CO algorithms. An initial population is obtained by η mutations of the local optimum point of the lower peak $\boldsymbol{x}_0^0 = [1, 0, \ldots, 0]$. The goal is to cross the saddle between both peaks. We assume that it is done when the weight mean of the population $E_t(\boldsymbol{x})$ is located at the higher peak, i.e. $\langle E(x_i \mid i = 1, 2, \ldots, n) \rangle < 0.35$. Other algorithm parameters are chosen as follows: the population size $\eta = 20$, the maximum number of epochs $t_{\max} = 10^5$, $\sigma = \tau = 0.05$ (for Gaussian and Cauchy mutations, respectively). All algorithms are tested for a set of dimensions of the adaptation landscape $n = 2, 4, 6, \ldots, 40$.

Relations between the mean number of epochs neccessary to cross the saddle taken over 100 runs of the algorithms and the dimension of the adaptation landscape for all algorithms are presented in Fig. 3.12. In the case of low dimensions, algorithms with standard Gaussian and non-spherical Cauchy mutations are substantially better than their modified versions. The most probable distances r_0 between parent points and successors are less than the peak thickness v. The ESSS-G and ESSS-C algorithms create more dispersed populations and their ability of saddle crossing is greater. The ESSS-C algorithm is the most effective algorithm for low dimensions, especially since the direction between both local and global fitness optima is parallel to the direction of the axe of the reference frame and is preferred by the non-spherical Cauchy mutation. The spherical Cauchy mutation (the ESSS-CO algorithm) evenly hands out directions to mutated points and its effectiveness initially decreases very quickly with n.

The surrounding effect manifests itself in a quick decrease in the ESSS-G and ESSS-C algorithms' efficiency. It is clearly visible in the case of the ESSS-C algorithm, which is still the best for $n = 10$ and becomes the worst for $n = 14$. The efficiency of the ESSS-GN and ESSS-GO algorithms is similar. Just in the very high dimensions, ESSS-GO becomes better. The most interesting results, especially for high landscape dimensions, are obtained for the spherical Cauchy mutation.

The four versions of the EP algorithm (CEP, CEPS, FEP and FEPS) are also tested in order to find the global optimum of the function (A1). All algorithm parameters are the same as in the previous experiment concerning sensitivity to the narrow peak apart from the initial area for population $\Omega_x =$

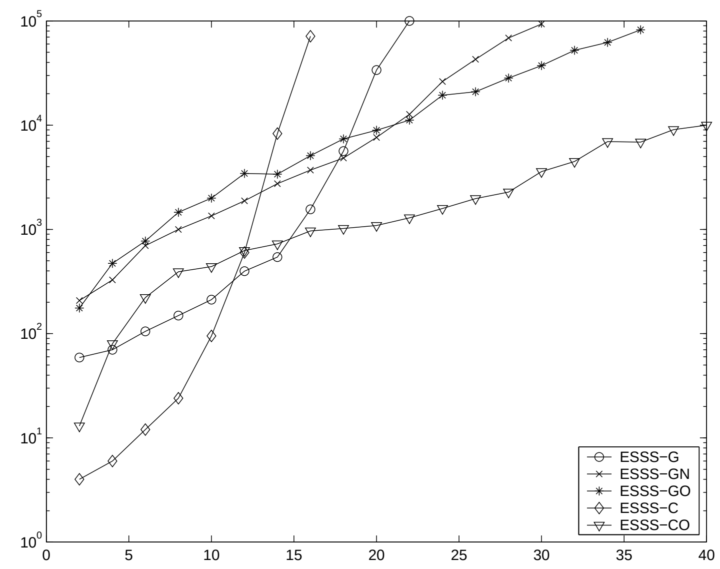


Fig. 3.12 . The mean number of epochs needed to cross the saddle vs. the dimension of the adaptation landscape.

$[0.8, 1.2] \times \prod_{i=2}^{5}[-0.2, 0.2]$. Each algorithm is started 100 times for each dimension of $n = 1, 2, \dots, 20$.

As it has been noted in the previous subsection, the saddle crossing efficiency of the EP-class algorithms is closely related to their ability to perform macro-mutations. Surprisingly, the searching process either finds the global peak in relatively short time (Fig. 3.13b), or it does not find it at all in $t_{\max} = 100000$ epochs (Fig. 3.13a). This is a disadvantage of the EP-class algorithms in comparison with the ESSS ones. Unexpectedly, the efficiency of the CEPS algorithm is much higher than that of the CEP algorithm for low dimensions. This disproportion disappears along with the dimension growth, when the saddle crossing efficiency of both the CEPS and FEPS algorithms rapidly decreases. The FEP algorithm seems to be least unreliable in the saddle crossing problem. It owes its success to its non-spherical mutation, which prefers directions parallel to the axis of the reference frame. The surrounding effect manifests itself in an exponential increase in the mean number of epochs needed to cross a saddle by the CEP and FEP algorithms with the landscape dimension (Fig. 3.13b). This relation is weaker in the case of the CEPS and FEPS algorithms.

3.3.5. Symmetry problem

Let us consider the following series of four-dimensional fitness functions:

$$\Phi_l(\boldsymbol{x}) = \frac{1}{2}\exp\big(-5\|\boldsymbol{x}\|^2\big) + \exp\big(-5\|\boldsymbol{x} - \boldsymbol{a}_l\|^2\big), \quad l = 1, 2, 3, 4, \tag{3.22}$$

where $\boldsymbol{a}_1 = [1,0,0,0]$, $\boldsymbol{a}_2 = [1/\sqrt{2}, 1/\sqrt{2}, 0, 0]$, $\boldsymbol{a}_3 = [1/\sqrt{3}, 1/\sqrt{3}, 1/\sqrt{3}, 0]$ and $\boldsymbol{a}_4 = [1/2, 1/2, 1/2, 1/2]$ are global optimum locations. Distances between both local and global optima are the same in all Φ_l and equal to unity.

The ESSS-C and ESSS-CO algorithms are tested in this experiment. The goal is the same as in the previous simulations: to cross the saddle between both peaks. Other algorithm parameters are chosen as follows: the population size $\eta = 20$, the maximum number of epochs $t_{\max} = 10^3$, $\tau = 0.05$ and the initial point of searching $\boldsymbol{x}_0^0 = [0,0,0,0]$.

Fig. 3.14 shows the relation between the mean number of epochs needed to cross a saddle taken over 10^3 algorithms' processing. It is easy to see that the ESSS-C algorithm's efficiency strongly depends on the direction of the global peak location. In the case of the ESSS-CO algorithm the saddle crossing efficiency is independent of this direction.

The symmetry effect in the EP-class algorithms is tested using five-dimensional Ackley's ($n = 5$):

$$\Phi_A(\boldsymbol{x}) = 20 + e - 20\exp\left(-\frac{1}{5}\frac{\|\boldsymbol{x}\|}{n}\right) - \exp\left(\frac{\sum_{i=1}^{n}\cos(2\pi x_i)}{n}\right) \tag{3.23}$$

and generalized Rastringin's function ($n = 5$):

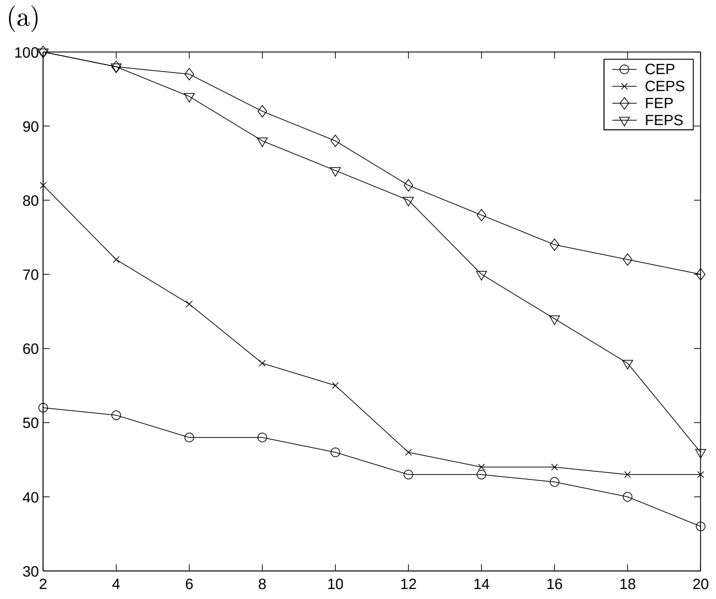
$$\Phi_R(\boldsymbol{x}) = \sum_{i=1}^{n}\left(x_i^2 - 10\cos(2\pi x_i) + 10\right). \tag{3.24}$$

Both functions have to be minimized. Next two functions Φ_{Ar} and Φ_{Rr} are rotated versions of Φ_A and Φ_R, i.e. both are obtained from Φ_A and Φ_R after rotation of the reference frame in the plane (x_1, x_2) through an angle equal to $\pi/4$, and in the plane (x_2, x_3) through an angle equal to $\pi/4$, too. Both Ackley's and Rastringin's functions are multimodal, but Rastringin's function characterizes the higher amplitude of changes and its valleys are deeper. Local optima of both functions Φ_A (3.23) and Φ_R (3.24) are located in the nodes of the 5D-cubic net, whose edges are parallel to the axes of the reference frame. This property is disturbed in the case of Φ_{Ar} and Φ_{Rr}.

The following parameters are used in the simulations: the population size $\eta = 50$, the maximum number of epochs $t_{\max} = 10000$, the number of sparing partners $q = 10$ and the initial area for variances $\Omega_\sigma = \prod_{i=1}^{5}[0,3]$. The initial areas for population are $\Omega_x = \prod_{i=1}^{5}[-5.12, 5.12]$ in the case of Φ_A and Φ_{Ar}, and $\Omega_x = \prod_{i=1}^{5}[-32, 32]$ in the case of Φ_R and Φ_{Rr}. Each algorithm is started 50 times.

The CEP and FEP algorithms reveal their advantage over the CEPS and FEPS algorithms in the case of Φ_A (Fig. 3.15a). The surrounding effect in the CEP algorithm makes it easier for the population to cross shallow saddles of Φ_A. The high effectiveness of the FEP algorithm follows from three main facts:

- high probability of macro-mutations (in the sense of phenomenon, not a new operator) using the Cauchy distribution,



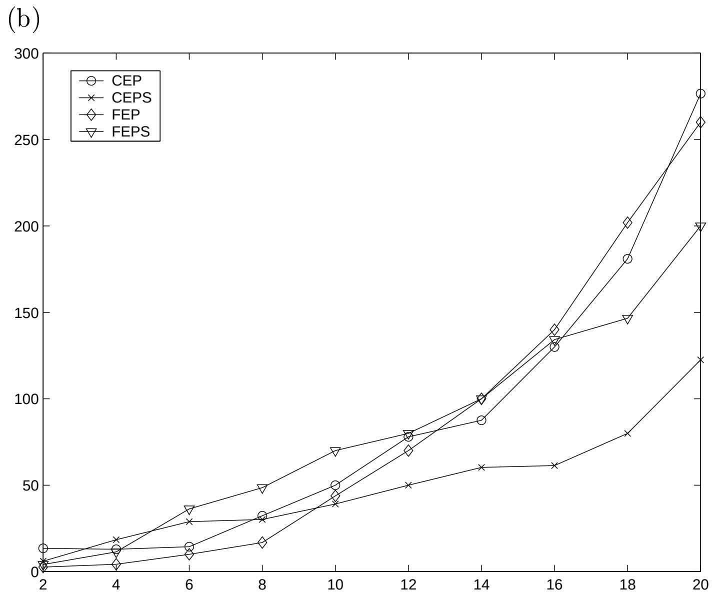


Fig. 3.13 . Percentages of the successful runs (a) and the mean number of epochs needed to cross the saddle taken over all successful runs (b) of four versions of the EP algorithm vs. the landscape dimension.

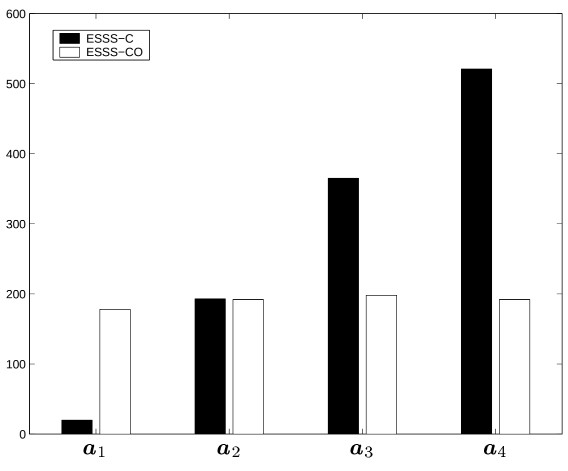


Fig. 3.14 . The mean number of epochs needed to cross the saddle vs. the global optimum location.

- Cauchy mutation preference of the direction parallel to axes of the reference frame, and
- the surrounding effect.

If the second property is desired in the case of the fitness function Φ_A, it becomes inconvenient in the case of Φ_{Ar} (Fig. 3.15b), where only efficiency of the CEPS algorithm is kept on the same level as in the case of Φ_A. The dependence of the FEP algorithm on the choice of the reference frame also manifests itself if one compares results obtained for both versions Φ_R and Φ_{Rr} of Rastringin's function (Fig. 3.16).

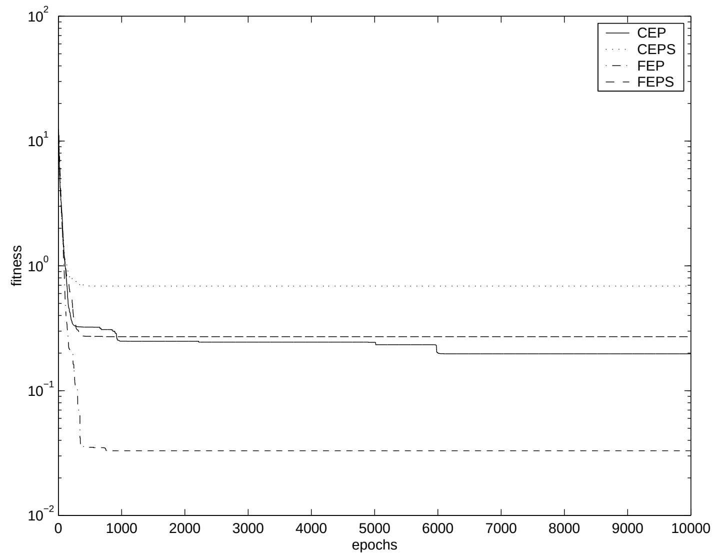
3.4. Summary

Two important properties of the Gaussian and Cauchy mutations, called *the surrounding effect* and *the symmetry effect*, are considered in details in this chapter. Both of them are overcome in the modified versions of Gaussian and Cauchy mutations. Here the direction of the mutation is first randomly chosen with uniform distribution and then the distance between the base and mutated points is randomly chosen with the one-dimensional Gaussian or Cauchy distribution.

Four experiments are reported on in this chapter. As examples of evolutionary algorithms Evolutionary Search with Soft Selection and Evolutionary Programming are used. The first one is probably the simplest selection-mutation model of evolution. The second one is the well-known Evolutionary Programming, proposed by Fogel (1966, 1992).

Convergence to a local optimum is analyzed in the first experiment, where four algorithms of the EP-class are tested. The performed simulations prove the influ-

(a)



(b)

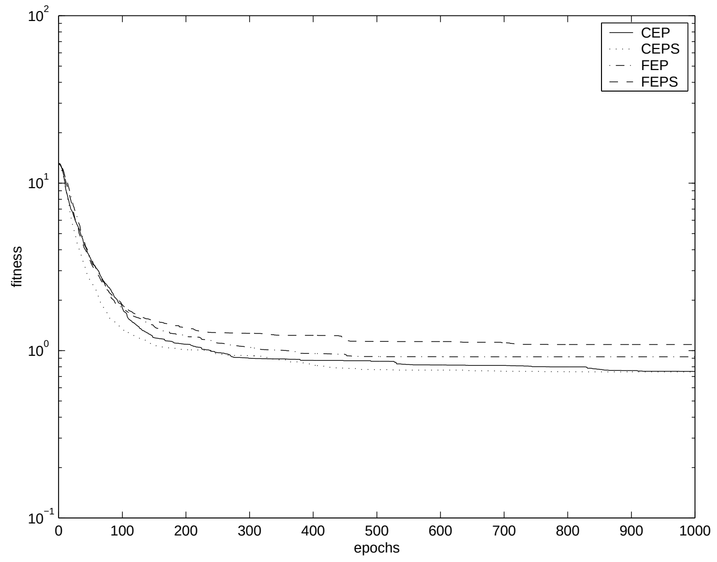
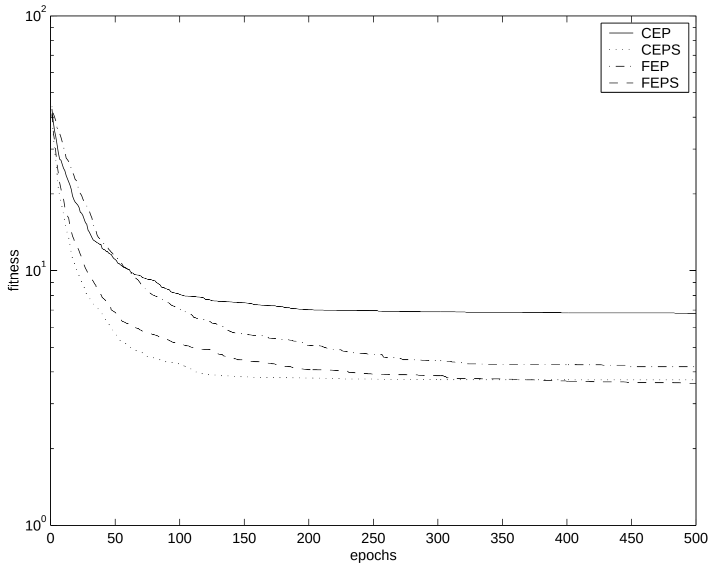


Fig. 3.15 . The fitness of the best element in the current population vs. epochs; results averaged over 50 samples for 5D Ackley's function Φ_A (a) and its rotated version Φ_{Ar} (b).

(a)



(b)

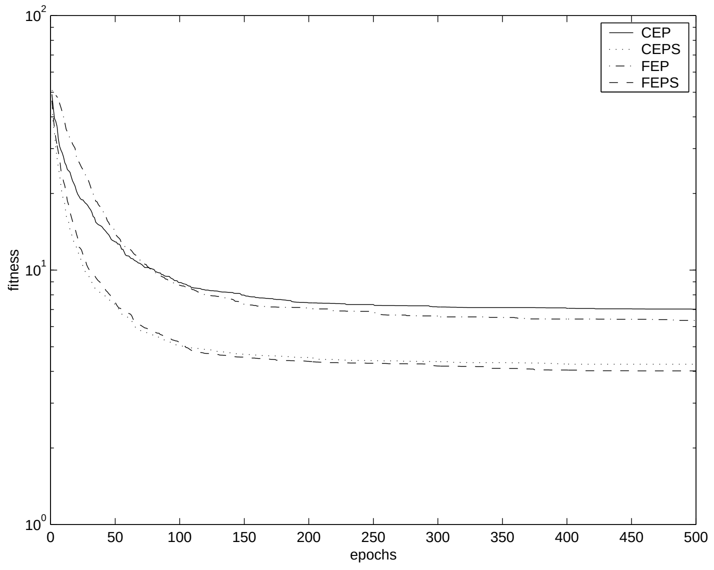


Fig. 3.16 . The fitness of the best element in the current population vs. epochs; results averaged over 50 samples for 5D Rastringin's function Φ_R (a) and its rotated version Φ_{Rr} (b).

ence of the surrounding effect on the convergence rate of evolutionary algorithms based on classical Gaussian and Cauchy mutations.

The second experiment illustrates the exploitation performance of the evolutionary algorithms considered. Evolutionary search with standard Gaussian mutation is least effective in localizing narrow peaks because of the surrounding effect. Only application of the modified Gaussian mutation proposed in this work guarantees success. In order to analyze the efficiency of the evolutionary algorithm, which adapts its mutation parameters during its processing, the four variants of the EP algorithm are tested. It can be seen that the CEPS and FEPS algorithms, which use the spherical mutation (3.13), are less effective than the CEP and FEP algorithms. This suggests that in the case of the mutation (3.13), a new formulae for the adaptation mechanism is needed instead of that applied in the classical form of the EP algorithm.

The third experiment presents the influence of the landscape dimension on the exploration efficiency of the algorithms. The measure of this efficiency is the mean number of generations needed to cross a saddle between two Gaussian peaks. It is not surprising that, in the case of the ESSS-class algorithms, standard Gaussian and Cauchy mutations give the best results in the case of low dimensions. The surrounding effect accelerates their capability of saddle-crossing. Unfortunately, the efficiency of the ESSS-G and ESSS-C algorithms rapidly decreases when the landscape dimension increases. Gaussian peaks become too narrow for these algorithms. Application of modified Gaussian and spherical Cauchy mutations again successfully overcomes this problem, as those are the most effective algorithms in high landscape dimensions. However, their efficiency in low dimensions is poor in comparison with standard mutations. When evolutionary algorithms with adapted mutation parameters are used, the disadvantage of the classical form of Gaussian and Cauchy mutations disappears. The surrounding effect is decreased by the adaptation mechanism.

The last experiment discloses the influence of the selection of the reference frame on the global optimization effectiveness of evolutionary algorithms which use the non-spherical Cauchy mutation.

The presented simulation results do not prove the advantage of multi-dimensional Gaussian and Cauchy mutations in their modified forms over their usually used classical versions. One can only say that these are different types of mutation operators, and each of them can be preferred for a different class of problems.

Chapter 4

EVOLUTIONARY ADAPTATION IN NON-STATIONARY ENVIRONMENTS

In recent years the problem of adaptation in time-varying landscapes has been intensively studied by many groups of researches. The number of publications successively grows. This domain of research is important and current from point of view of many technical branches, e.g. the optimal control, the learning process of neural networks, the fault detection in dynamic systems. Unfortunately, diverse methodology and terminology make most of research solutions incomparable. In this chapter some proposal of ordered view on optimization and adaptation problems in non-stationary environments are introduced. Some taxonomy of non-stationary environments as well as measures of adaptation algorithms quality are also proposed.

4.1. Non-stationary environments

A non-stationary optimization problem in general can be formulated as follows:

$$\max f(\boldsymbol{x},t)\big|\big(c_i(\boldsymbol{x},t)\leq 0, i=1,\ldots,m, \boldsymbol{x}\in\mathcal{U}(t)\big), \tag{4.1}$$

where $f(\boldsymbol{x},t)$ is an objective function, $c_i(\boldsymbol{x},t)$ denotes an i^{th} constraint and $\mathcal{U}(t)$ is a space of solutions.

Non-stationary problems can be classified under a number of criteria. The first one is a physical structure of the space of solutions $\mathcal{U}(t)$: is it discrete or continuous? A domain structure determines a class of possible measures of evolutionary algorithms performance.

In general $f(\boldsymbol{x},t)$, $c_i(\boldsymbol{x},t)$ and $\mathcal{U}(t)$ can be time varying simultaneously. But physically it occurs very seldom. The first attempt to classification of all possible cases, which elements of the sequence $(f,\{c_i\}_{i=1}^m,\mathcal{U})$ are varying in time, is provided in (Trojanowski and Michalewicz 1999b). An extension of this classification is proposed in (Trojanowski and Obuchowicz 2001) (Table 4.1).

Changes of the domain, e.g. changes of the number of dimensions or of dimensions' boundaries significantly modify the nature of the problem. For example, for binary representation of solutions, changes of the domain modify resolution and

Tab.4.1. Classification of changes in a process. The symbol $\emptyset$ denotes the case where the set of constrains is empty; *static* — there are no change in time; *varying* — there are some changes in time.

No.	*objective function*	*constrains*	*space of solutions*
$\mathcal{M}_1$	*static*	$\emptyset$	*static*
$\mathcal{M}_2$	*static*	*static*	*static*
$\mathcal{M}_3$	*static*	*varying*	*static*
$\mathcal{M}_4$	*varying*	$\emptyset$	*static*
$\mathcal{M}_5$	*varying*	*static*	*static*
$\mathcal{M}_6$	*varying*	*varying*	*static*
$\mathcal{M}_7$	*static*	$\emptyset$	*varying*
$\mathcal{M}_8$	*static*	*static*	*varying*
$\mathcal{M}_9$	*static*	*varying*	*varying*
$\mathcal{M}_{10}$	*varying*	$\emptyset$	*varying*
$\mathcal{M}_{11}$	*varying*	*static*	*varying*
$\mathcal{M}_{12}$	*varying*	*varying*	*varying*

precision of the algorithm so there is a need for modification of individual representation. Thus, in the case of such change, we usually have to re-start the search procedure and to tune the optimization tool to the new problem after the change has occurred. For the sake of that, we assume that the domain is constant and do not discuss this form of changes in further text.

There are a number of criteria along which non-stationary environments can be categorized (Branke 1999):

- frequency of changes;
- severity of changes;
- predictability of changes;
- regularity of changes.

Frequency of changes. The environment can change with different frequency, from continuous in time to very rare sudden changes which are preceded by stationary state of the environment. An example of a environment with the continuous changed can be the optimal control problem in the case of a real system affected by an ageing process, or a time optimal trajectory planning for mobile robots in the case of moving obstruction.

Severity of changes. The nature of changes can manifest in their speed and range. The classification under this criteria is difficult because of estimation subjectivity whether changes are sudden or adiabatic, wide or local. In the case of model $\mathcal{M}_4$ (Tab. 4.1), where the fitness function is varying in time, some measure of changes in a given subspace $\Omega \subset \mathcal{U}$ and a given time interval T can be introduced (Trojanowski and Obuchowicz 2001)

- continuous domain
  (let $f(\boldsymbol{x},t) \in L_2(\Omega), \forall t$)

$$M(\Omega,T,t) = \frac{1}{T}\frac{\int\cdots\int_\Omega \left(f(\boldsymbol{x},t) - f(\boldsymbol{x},t-T)\right)^2 d\omega}{\int\cdots\int_\Omega f^2(\boldsymbol{x},t)d\omega}, \tag{4.2}$$

  where $d\omega = dx_1dx_2\ldots dx_n$;

- discrete domain

$$M(\Omega,T,t) = \frac{1}{T}\frac{\sum_{\boldsymbol{x}_i\in\Omega} \left(f(\boldsymbol{x}_i,t) - f(\boldsymbol{x}_i,t-T)\right)^2}{\sum_{\boldsymbol{x}_i\in\Omega} f^2(\boldsymbol{x}_i,t)}. \tag{4.3}$$

The measure $M(\Omega,T,t)$ describes the average speed of relative fitness changes in subspace Ω taken over the time interval T which can be considered as a sampling interval, i.e. the time interval between two successive calculations of the fitness function. One may define two constants Θ_a and Θ_c ($\Theta_a < \Theta_c$) for given searching problem in order to classify changes of the fitness function:

- $M(\Omega,T,t) < \Theta_a$ — the adiabatic changes ($\mathcal{S}_1$), which guarantee approximately stationary state of evolutionary search. The population “keeps up" with the changed optimum. There are usually no quality differences between this problem and stationary problems.

- $\Theta_a < M(\Omega,T,t) < \Theta_c$ — the indirect changes ($\mathcal{S}_2$). It is the most interesting case. An effectiveness of the searching process significantly depends on a chosen searching strategy and its input parameters.

- $M(\Omega,T,t) > \Theta_c$ — the turbulent changes ($\mathcal{S}_3$). In this case, usually the search procedure have to be restarted and tuned to the completely new problem after the change has occurred.

Parameters Θ_a and Θ_c have, rather, informal nature and are not well defined. The ability of classifying, to which of changes type: $\mathcal{S}_1, \mathcal{S}_2$ or $\mathcal{S}_3$, a given problem belongs, allows to choose a class of optimization methods to solve the problem and choose a measure of a given methods effectiveness. If a given problem belongs to the class $\mathcal{S}_1$, global optimum is usually moved to such a point which is close enough to be found again without a risk of becoming trapped in a local optimum. Then it is possible to use standard optimization methods, like gradient methods, to follow the optimum point during all the processing time. Here, evolutionary computation method is computationally rather too expensive to use.

From an evolutionary point of view, $\mathcal{S}_2$ is the most interesting class. The changes are too difficult and therefore computationally too expensive for the classical optimization methods, but not too difficult for evolutionary methods, which may solve the problem because of their softness and concurrent searching, especially when we are satisfied even if the solution is suboptimal only.

The turbulent changes $\mathcal{S}_3$ are usually unable to control (e.g. the changes of square error function in the on-line neural network training process where a sequence of training patterns is randomly chosen from a training set). Any optimization process can not keep up the optimum peak track. Applied adaptation algorithms usually find hills of a form of objective function averaged over searching time

Predictability of changes. If changes of the problem components appear in real-world optimization tasks continuously or at least periodically, then values of proposed solutions vary in time and thus a continuous search process is needed. In general, non-stationary optimization task belongs to one of four main groups (Mothes 1967):

1. *Deterministic situations*, where full information about the values of environment parameters now and in the future is available.

2. *Probabilistic situations*, where the values of environment parameters are not known, however they are predictable, because probability distributions of these parameters are known.

3. *Uncertain situations*, where environment parameters are unknown and unpredictable.

4. *Conflict situations*, where the environment parameters are controlled by our antagonists (cases, where a game theory is used).

The subject of our interest is the third group of situations. A large number of real world problems belongs to this group, and examples of such non-stationary and unpredictable problems can be easily found around us (Trojanowski and Michalewicz 1999b, Trojanowski and Michalewicz 1999c).

Regularity of changes. Investigating properties of optimization tools applied to problems varying in time, it is also necessary to study and classify different forms of changes. Changes of the problem components can be classified in many ways, for example (Trojanowski and Michalewicz 1999b):

— regularity of changes (i.e. cyclic and non-cyclic ones);

— continuous vs. discrete changes in time;

— continuous vs. discrete changes in the search space.

Obviously, not every problem varying in time can be optimized with evolutionary algorithms, e.g. situations where immediate reaction is needed, but there is still a large group of cases (like e.g. control and management of electric energy sources by a dispatcher day by day).

4.2. Quality rates for adaptation algorithms

In evolutionary computation community, some measures for obtained results have been proposed; these measures exploit the iterational nature of the search process and the presence of continuously modified and improved population of solutions. One of the first measures were on-line and off-line performance proposed by De Jong (1975).

- **Off-line performance** — is the best value in the current population averaged over the entire run. It represents the efficiency of the algorithm in a given time of run.
- **On-line performance** — is the average of all evaluation of the entire run. It shows the impact of the population on the focus of the search.

These two measures, although designed for static environments, were employed in experiments with non-stationary ones (Bäck 1998, Grefenstette 1992, Vavak and Fogarty 1996, Vavak *et al.* 1997).

In other publications, authors visually compared graphs of the best objective function value measured during the entire search process (or graphs of the mean value obtained from series of experiments) (Angeline 1997, Bäck and Schutz 1996, Branke 1999, Cedeno and Vemuri 1997, Cobb and Grefenstette 1993, Dasgupta and McGregor 1992, Ghosh *et al.* 1998, Goldberg and Smith 1987, Grefenstette 1992, Grefenstette 1999, Lewis *et al.* 1998, Mori *et al.* 1997, Mori *et al.* 1996, Mori *et al.* 1998, Ng and Wong 1995, Vavak and Fogarty 1996). In some papers graphs of average values of all individuals or of the worst individual in the population were also analyzed (Cobb and Grefenstette 1993, Goldberg and Smith 1987, Dasgupta and McGregor 1992, Mori *et al.* 1997, Mori *et al.* 1996, Mori *et al.* 1998). Both these methods were based on the measures of off-line and on-line performance.

Before a form of a quality rate for searching algorithm in the non-stationary environments is chosen, a researcher has to decide what kind of results will be satisfying, what type of searching process should be applied. Four main types of searching processes can be distinguished. Let us assume that the fitness function is equal to the objective function $\Phi(\boldsymbol{x}, t) = f(\boldsymbol{x}, t)$.

$\mathcal{C}_1$: **A tracing process.** — This type of the searching process is dedicated mainly to adiabatic problems ($\mathcal{S}_1$). The goal of the searching process of the type $\mathcal{C}_1$ is to keep solutions closed to the optimum one as well as possible. Most of publications of the non-stationary optimization consider such a type of searching process. Applied measures of searching algorithms are usually based on measures for stationary environments.

An interesting measure based on the off-line performance is an *adaptation performance* described in (Mori *et al.* 1997, Mori *et al.* 1996). It was evaluated according to the formula:

$$\mu_{ad} = \frac{1}{t_{max}} \sum_{t=1}^{t_{max}} \frac{\Phi_{best}(t)}{\Phi_{opt}(t)}, \tag{4.4}$$

where: t_{max} is the length of the entire search process, $\Phi_{best}(t)$ — the fitness of the best individual in the population at the time t, $\Phi_{opt}(t)$ — the optimum fitness in the search space at the time t.
This formula was later modified slightly to:

$$\mu'_{ad} = \frac{1}{t_{max}} \sum_{t=1}^{t_{max}} \alpha \frac{\Phi_{best}(t)}{\Phi_{opt}(t)}, \tag{4.5}$$

where (for the maximization problem):

$$\alpha = \begin{cases} 1, & \text{if } \Phi_{best}(t) = \Phi_{opt}(t), \\ 0.5, & \text{if } \Phi_{best}(t) < \Phi_{opt}(t). \end{cases}$$

In (Feng *et al.* 1998), two benchmarks measuring relative closeness of the best found solution to the global optimum were proposed: *optimality* μ_{op} and *accuracy* μ_{ac}. *optimality* μ_{op} represents closeness of the value of the best obtained solution $\Phi(\boldsymbol{x}_0)$ to the value of optimum Φ_{opt}, e.g. for the maximization problem, we have the following formulae:

$$\mu_{op} = \frac{\Phi(\boldsymbol{x}_0) - \Phi_{min}}{\Phi_{opt} - \Phi_{min}}, \tag{4.6}$$

where $\Phi_{min} = \min_{\boldsymbol{x}\in\mathcal{U}} \Phi(\boldsymbol{x})$. The *accuracy* μ_{ac} represents the relative closeness of the found solution $\boldsymbol{x}_0$ to the global optimum solution $\boldsymbol{x}_{opt}$ and it is defined with following formula:

$$\mu_{ac} = 1 - \frac{\rho(\boldsymbol{x}_{opt}, \boldsymbol{x}_0)}{\rho(\boldsymbol{x}_{max}, \boldsymbol{x}_{min})}. \tag{4.7}$$

where $\boldsymbol{x}_{opt} = \arg\max_{\boldsymbol{x}\in\mathcal{U}} \Phi(\boldsymbol{x})$, $\boldsymbol{x}_{max}$ and $\boldsymbol{x}_{min}$ are the lower and upper bounds of the search range, and $\rho(\boldsymbol{a}, \boldsymbol{b})$ is a distance measure in $\mathcal{U}$, e.g. if $\mathcal{U} \subset \mathbb{R}^n$ then $\rho(\boldsymbol{a}, \boldsymbol{b}) = \|\boldsymbol{a} - \boldsymbol{b}\|$.
Although authors did not use these measures to non-stationary optimization evaluation, the closeness to the optimum during the search process is an interesting value which seems to be helpful in comparisons between applications and is easy to control in experiments. The evolutionary approach to non-stationary optimization presented in (Obuchowicz 1999b) uses measure which idea is closely related to the measure μ_{ac}:

$$\mu'_{ac} = \frac{1}{t_{max}} \sum_{t=1}^{t_{max}} \rho\big(\boldsymbol{x}_{opt}(t), \boldsymbol{x}_0(t)\big), \tag{4.8}$$

where $\boldsymbol{x}_0(t)$ is the best point of population in the time t and $\boldsymbol{x}_{opt}(t) = \arg\max_{\boldsymbol{x}\in\mathcal{U}} \Phi(\boldsymbol{x}, t)$.

$\mathcal{C}_2$: **An optimization in a mega-epoch.** — This type of searching process concerns tasks, in which consecutive changes in the environment are significant but occur rather seldom (ones in a 'mega-epoch'). The goal is to find the optimum before a new change occurs.

For estimations of non-stationary optimization results, the following two measures: *accuracy* (μ_{acc}) and *adaptability* (μ_{ada}) were proposed in (Trojanowski and Michalewicz 1999a, Trojanowski and Michalewicz 1999c). They are based on a measure proposed by De Jong (1975): *off-line performance* but evaluate the difference between the value of the current best individual and the optimum value instead of evaluation of the value of just the best individual. *Accuracy* is a measure dedicated exclusively to dynamic environments. It is a difference between the value of the current best individual in the population of the "just before the change" generation and the optimum value averaged over the entire run:

$$\mu_{acc} = \frac{1}{K}\sum_{i=1}^{K}(err_{i,\tau-1}). \tag{4.9}$$

Adaptability measures a difference between the value of the current best individual of each generation and the optimum value averaged over the entire run:

$$\mu_{ada} = \frac{1}{K}\sum_{i=1}^{K}\left[\frac{1}{\tau}\sum_{j=0}^{\tau-1}(err_{i,j})\right], \tag{4.10}$$

where: $err_{i,j}$ is the difference between the value of the current best individual in the population of the j-th generation after the last change ($j \in [0, \tau-1]$), and the optimum value for the fitness landscape after the i-th change ($i \in [0, K-1]$), τ — the number of generations between two consecutive changes, K — the number of changes of the fitness landscape during the run.
Clearly, the lower values of measure (for both *accuracy* and *adaptability*) correspond to the better results. In particular, a value of 0 for *accuracy* means that the algorithm found the optimum every time before the landscape was changed (i.e. τ generations were sufficient to track the optimum). On the other hand, a value of 0 for *adaptability* means that the best individual in the population was at the optimum for all generations, i.e. the optimum was never lost by the algorithm.

$\mathcal{C}_3$: **Keeping solutions on an acceptable level.** — In many real technological problems, e.g. in the *on-line* training of a dynamic neural networks (Korbicz *et al.* 1998), in control systems (Bryson and Ho 1975) or in many problems of the operational research, the optimal solution is not so necessary as the solution of an acceptable quality. This problems usually are of the type $\mathcal{S}_2$. One has to be sure that the fitness of the actual best known solution will not be worse than a given assumed level during all time long of a searching process. This acceptable level may, for example, describe non conflict run of concurrent processes, guarantee the stability of the controlled dynamic system.

The *acceptability* is a measure of unsatisfied realization in a quality space — this is a mean deviation of the best fitness in time t below the acceptable level (Trojanowski and Obuchowicz 2001)

$$\mu_{acpt} = \frac{1}{t_{max}} \sum_{t=1}^{t_{max}} \theta\big(\Phi_{acpt}(t) - \Phi_{best}(t)\big), \tag{4.11}$$

where: t_{max} is the length of the entire search process, $\Phi_{best}(t)$ — the fitness of the best individual in the population at the time t, and $\Phi_{acpt}(t)$ is the minimum satisfying value of the fitness (it can be varying in time) and (Trojanowski and Obuchowicz 2001)

$$\theta(a-b) = \begin{cases} 0, & \text{if } a \leq b, \\ a-b, & \text{if } a > b. \end{cases}$$

The *acceptability distance* μ_{acd} is an equivalent of μ_{acpt} in searching space and is defined as follows:

$$\mu_{acd} = \frac{1}{t_{max}} \sum_{t=1}^{t_{max}} \theta\big(\rho(\boldsymbol{x}_{opt}(t), \boldsymbol{x}_0(t)) - r\big), \tag{4.12}$$

where r is, so called, the acceptability radius which describes the maximum acceptable distance between the best known solution and the actual optimum point.

$\mathcal{C}_4$: **A process with averaged acceptability.** — This type of searching process is dedicated to turbulent problems ($\mathcal{S}_3$). A searching process is unable to follow the optimum as well as to guarantee the acceptable solutions during the algorithm processing. The only measure of the adaptation process is its ability to find the solution with the best average fitness over all realization of $\Phi(\boldsymbol{x},t)(t = 1, 2, \ldots, t_{\max})$. This measure can be expressed in the following form

$$\mu_{avac} = \frac{1}{t_{max}} \sum_{t=1}^{t_{max}} \rho\big(\boldsymbol{x}^\star, \boldsymbol{x}_0(t)\big), \tag{4.13}$$

where

$$\boldsymbol{x}^\star = \arg\max \left(\frac{1}{t_{max}} \sum_{t=1}^{t_{max}} \Phi(\boldsymbol{x}, t) \right).$$

4.3. Illustrative simulations

4.3.1. Properties of the ESSS in the adiabatic and turbulent cases of landscape non-stationarity

Let us consider the following one-dimensional non-stationary adaptation landscape composed by two Gaussian peaks

$$\begin{aligned}\Phi(x,t) &= (0.5+0.5\cos(\pi t/s))e^{-5(x-0.5)^2}\\ &\quad + (0.5-0.5\cos(\pi t/s))e^{-5(x+0.5)^2},\end{aligned} \tag{4.14}$$

where t denotes time, s is a given positive parameter controlling the rate of change of peaks highs.

Adaptation process is controlled by the ESSS algorithm (Trojanowski and Obuchowicz 2001). Figure 4.1 illustrates the adiabatic function changes. The best point of the population follows the global optimum during all time. The measures for the obtained results dedicated for stationary problems can be applied. In the case of the turbulent changes (Fig. 4.2) the global optimum is not monitored. The population fluctuates around the point of the smallest changes of the objective function. This feature is well known in the *on-line* training process of the artificial neural networks (Korbicz *et al.* 1994)

4.3.2. Comparison of four algorithms from ESSS family

Let us consider the following time-varying 2D adaptation landscape:

$$\Phi(\boldsymbol{x}) = h\exp\left[\left(\boldsymbol{x}-\boldsymbol{z}^t\right)^T\mathbb{C}^{-1}\left(\boldsymbol{x}-\boldsymbol{z}^t\right)\right], \tag{4.15}$$

$$\boldsymbol{z}^t = \left[z_1^t, z_2^t\right] = \left[1+\cos\left(\frac{2\pi t}{360}\right), \sin\left(\frac{2\pi t}{360}\right)\right] \tag{4.16}$$

where $\Phi(\boldsymbol{x})$ represents a Gaussian peak with expectation vector $\boldsymbol{z}^t$, which moves around the circumference of unity radius, and covariance matrix $\mathbb{C}$.

Four algorithms are used in the adaptation process: ESSS, ESSS-FDM, ESSS-VPS and the last ESSS-FV containing both FDM and VPS mechanisms. Simulations have been carried out for various sets of input parameters, several times for each set. Representative realizations of the considered algorithms are presented in Figs. 4.3–4.6.

As a quality measure, the average distance between the best element of the population and the top of the Gaussian peak is chosen (4.8). For all the considered algorithms, there exists a set of input parameters, which usually realizes a satisfying quality factor of the adaptation process, although, the sensitivity of this factor to slight changes in the optimal input parameters is different for each tested algorithm.

The ESSS-FDM algorithm turns out to be least sensitive to input parameter disturbances. Thus, the search time for the optimal input parameters set

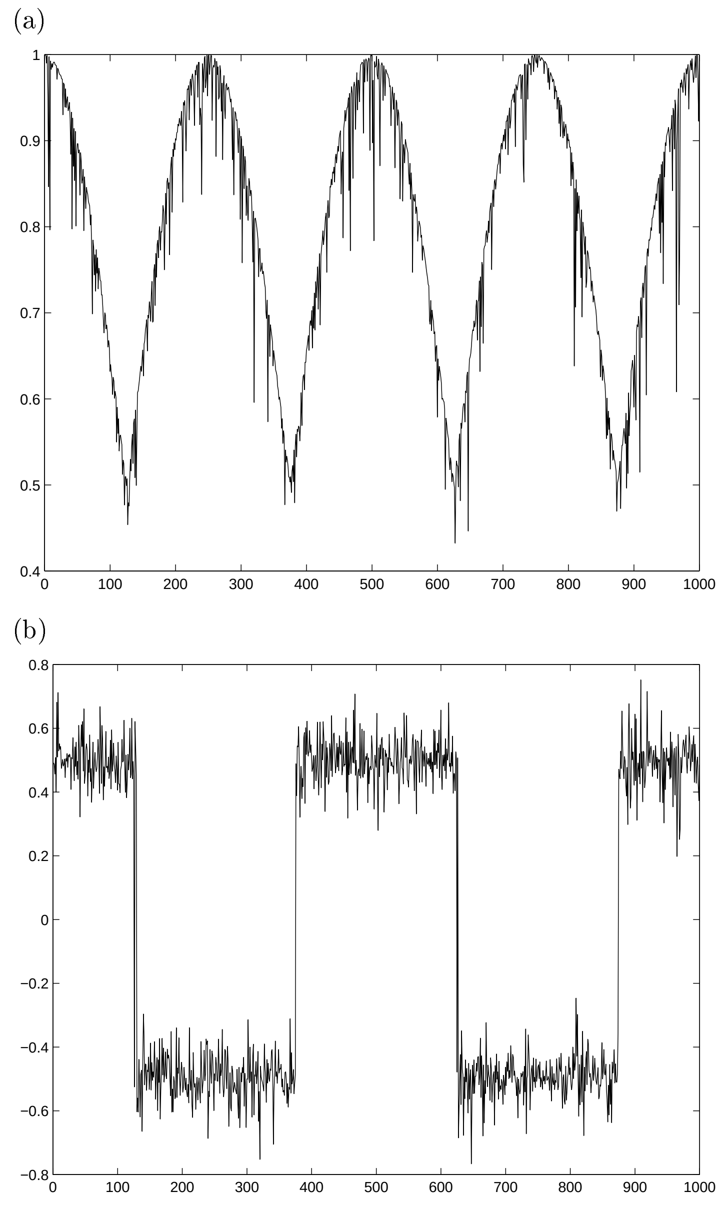
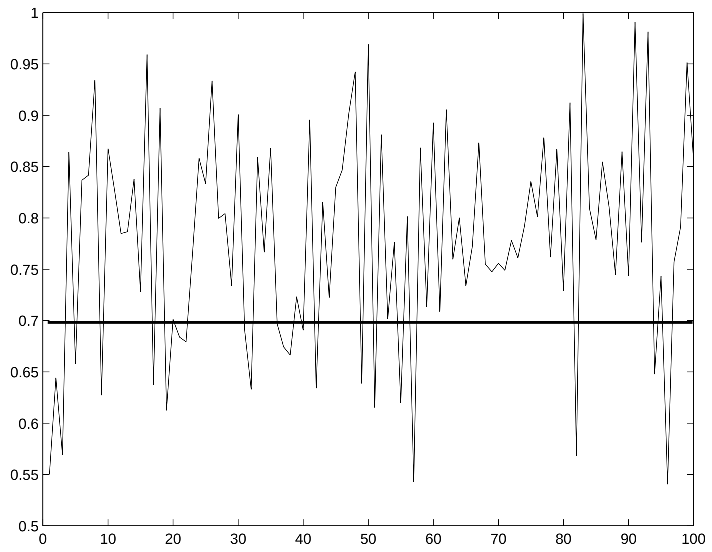


Fig. 4.1 . Adiabatic changes $M([-2,2],1) \approx 2.86 \times 10^{-9}$. The fitness (a) and location (b) of the best element in the population vs. time, ($\eta = 20$, $\sigma = 0.05$, $s = 250$).

(a)



(b)

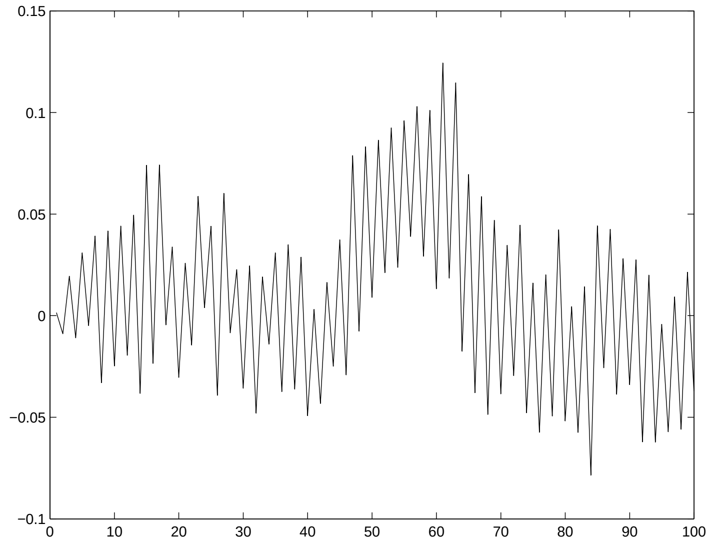


Fig. 4.2 . Turbulent changes $M([-2,2],1) \approx 1.84$. The fitness (a) of the best element in the population vs. time, and location of the mean point in the population vs. time (b),($\eta = 20$, $\sigma = 0.01$, $s = 1$).

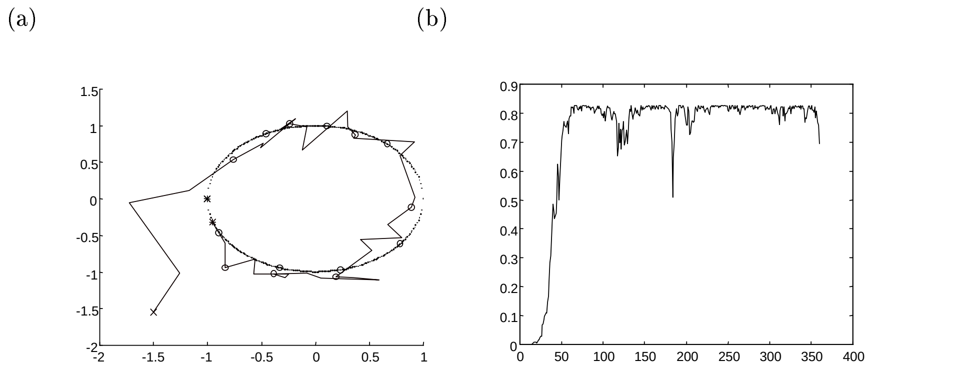


Fig. 4.3 . Realization of the ESSS algorithm in the time-varying landscape (4.15); input parameters: $\eta = 20$, $\sigma = 0.5$; the quality factor: $\mu'_{ac} = 0.345$; (a) evaluation of the Gaussian peak (dotted line) and the best element of the population (solid line); consecutive circles represent the results obtained every 50 iterations; (b) fitness of the best element in population vs. iterations.

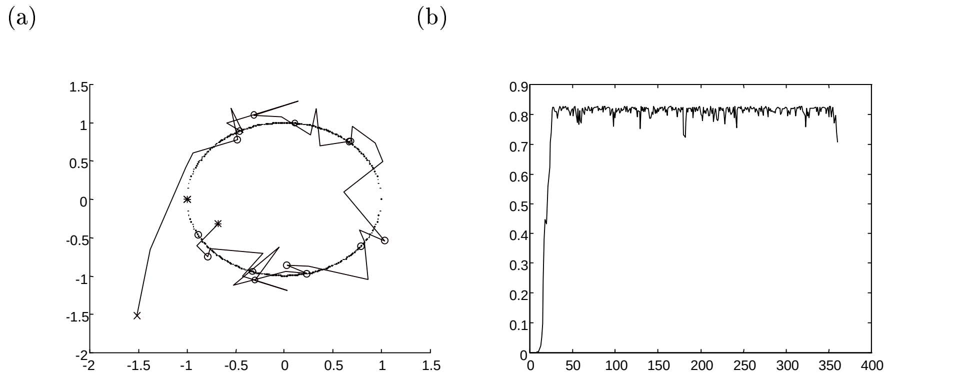


Fig. 4.4 . Realization of the ESSS-FDM algorithm in the time-varying landscape (4.15); input parameters: $\eta = 20$, $\sigma = 0.8$; $\mu = 0.125$; the quality factor: $\mu'_{ac} = 0.206$; (a) evaluation of the Gaussian peak (dotted line) and the best element of the population (solid line); consecutive circles represent the results obtained every 50 iterations; (b) fitness of the best element in population vs. iterations.

was shortest in comparison with the remaining algorithms. This property of the ESSS-FDM algorithm is used in dynamic neural networks learning process (see section 5.4.2).

(a) (b)

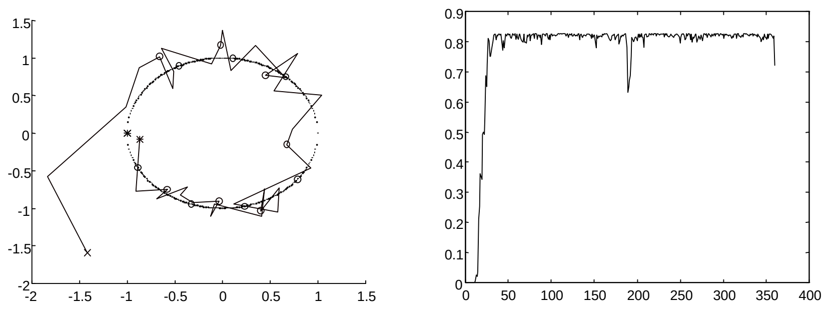


Fig. 4.5 . Realization of the ESSS-VPS algorithm in the time-varying landscape (4.15); input parameters: $\eta = 20$, $\sigma = 0.8$; the quality factor: $\mu'_{ac} = 0.280$; (a) evaluation of the Gaussian peak (dotted line) and the best element of the population (solid line); consecutive circles represent the results obtained every 50 iterations; (b) fitness of the best element in population vs. iterations.

(a) (b)

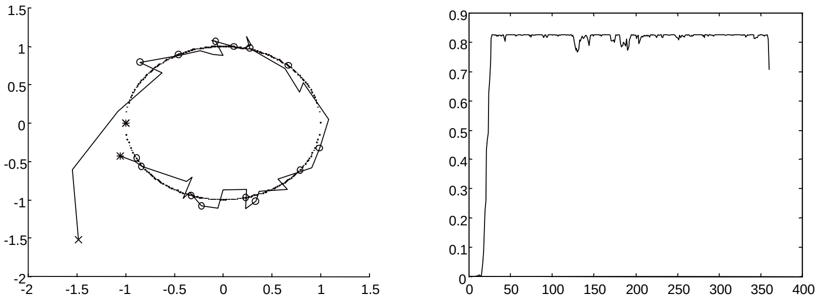


Fig. 4.6 . Realization of the ESSS-FV algorithm in the time-varying landscape (4.15); input parameters: $\eta = 20$, $\sigma = 0.8, \mu = 7.5$; the quality factor: $\mu'_{ac} = 0.303$; (a) evaluation of the Gaussian peak (dotted line) and the best element of the population (solid line); consecutive circles represent the results obtained every 50 iterations; (b) fitness of the best element in population vs. iterations.

4.4. Summary

For years, the evolutionary algorithms were applied mostly to the group of static problems. A set of satisfying procedures for tuning and comparisons between different approaches was established during that time. However, nowadays a wider group of problems, including non-stationary cases, is optimized with evolutionary algorithms but still with the same old methods and procedures of results evaluation and algorithms comparison. Because of the extension of optimized problems range, these measures seem to be insufficient, and new measures for optimization tool quality and for the non-stationary problem difficulty should be proposed.

In this chapter, an analysis and classification of these problems, review of the existing measures and some propositions of new ones as well as the simulation study of algorithms from the ESSS family in the non-stationary environment are presented.

Chapter 5

OPTIMIZATION TASKS IN NEURAL MODELS DESIGNING

Artificial Neural Networks (ANN) provide an excellent mathematical tool for dealing with non-linear problems. They have an important property, according to which any continuous non-linear relationship can be approximated with arbitrary accuracy using a neural network with suitable architecture and weight parameters (Korbicz *et al.* 1994). Their another attractive property is the self learning ability. A neural network can extract the system features from historical training data using the learning algorithm, requiring a little or no *a priori* knowledge about the process. This provides modelling of non-linear systems a great flexibility (Fausett 1994, Hertz *et al.* 1991, Korbicz *et al.* 1994). These properties make the ANN a very attractive tool in modelling and identification of dynamic processes, adaptive control systems (Hunt *et al.* 1992, Miller *et al.* 1990), time series prediction problems (Zhang and Man 1998), and diagnostics of industrial processes (Frank and Köppen-Seliger 1997, Koivo 1994).

The application of ANNs in modelling and identification of dynamic processes has been intensively studied for the last two decades (cf. (Korbicz *et al.* 1998, Narendra and Parthasarathy 1990, Zhu and Paul 1995)). Attractiveness of ANNs results from the fact that they are useful when there are no mathematical models of an investigated system, hence, analytical models and parameter-identification algorithms cannot be applied. As opposed to a lot of ANN effective applications, e.g. in the pattern recognition (cf. (Looney 1997, Sharkey 1999)) or in the approximation of the non-linear function (cf. (Hornik *et al.* 1989)), the application of ANNs in modelling requires taking into consideration the dynamics of the investigated processes.

One of the possible solutions is the application of recurrent neural networks (Draye *et al.* 1996, Tsoi and Back 1994). The most general architecture of recurrent networks was proposed by Williams and Zipser (1989), where connections between any neurons are permitted. Unfortunately, a practical realization of such a network structure is very limited, mainly due to their instability and a very slow convergence of the training process. The Elman recurrent network has less general character but better characteristics of practical applications (Elman 1990). It is worth noting that the standard recurrent neural networks are built using the static McCulloch-Pitts neuron model (McCulloch and Pitts 1943), and their relatively good dynamic properties are achieved by introduction of global feedbacks. Generally, such networks suffer from stability problems during training and require complicated learning algorithms.

The alternative solution is the application of a neural network of the Multilayer Perceptron (MLP) structure but composed of dynamic neurons. In general case, dynamic neuron models can be obtained by introducing one of the following feedbacks: synapse feedback (Gupta and Rao 1993), output feedback (Fasconi *et al.* 1992) or activation feedback (Tsoi and Back 1994) to the static McCulloch-Pitts model. Another solution can be obtained by extension of a static model by adding memory elements (Sastry *et al.* 1974). One of the most interesting solutions of dynamic system modelling problem is the application of the Dynamic Neural Model (DNM) which consists of an adder module, a linear dynamic system - Infinite Impulse Response (IIR) filter, and a non-linear activation module ((Ayoubi 1994, Patan and Korbicz 1996)). The dynamic neuron models render it possible to design a neural network with a structure similar to the well-known MLP. Taking into account the fact that this structure has no feedbacks between neurons, one can train it in a simpler way than the globally recurrent networks (Campolucci *et al.* 1999).

The construction process of an ANN, which has to solve a given problem, usually consists of four steps (Obuchowicz 2000a). First, a set of pairs of input and output patterns, which should represent characteristics of a problem as well as possible, is selected. Next, an architecture of the ANN, the number of units, their ordering into layers or modules, synaptic connections and other structure parameters, are defined. At the third step, free parameters of the ANN (e.g. weights of synaptic connections, slope parameters of activation functions) are automatically trained using a set of training patterns. Finally, the obtained ANN is evaluated in accordance with a given quality measure. The above process is repeated until the quality measure of the ANN is satisfied. Therefore, two optimization processes can be distinguished in the ANN construction process: an optimal architecture designing and optimal ANN parameters allocation (learning process).

The relatively complex DNM allows to build an effective Dynamic MLP (DMLP). The DMLP can have the same architecture as the MLP. The calculated output error is propagated back to the input layer through hidden layers containing dynamic filters, similarly as in the standard Back-Propagation (BP) algorithm (Werbos 1974, Korbicz *et al.* 1994). As a result, the Extended Dynamic Back-Propagation algorithm may be defined (Patan 2000, Patan and Korbicz 1996). This algorithm adjusts connection weights as well as IIR filter parameters. Unfortunately, the training process of an DMLP which has to identify a dynamic system, seems to be an optimization problem which is intrinsically related to a very rich topology of the sum square-error function (Korbicz *et al.* 1998). The EDBP algorithm usually finds one of the local unsatisfactory optima. Therefore, global optimization methods, like stochastic algorithms (Patan and Obuchowicz 1999) or evolutionary algorithms (Obuchowicz 1999a, Patan and Jesionka 1999), should be implemented. High performance of a dynamic system neural modelling, which has been trained by an evolutionary algorithm, has been observed by Obuchowicz (1999a).

However, there are effective methods of a training patterns selection, learning and an evaluation of the ANN, researchers usually allocate the ANN architecture rather on a basis of their intuition and experience than using an automatic procedure. Experienced researcher has, usually, no problems with architecture design in the case of the MLP applied to the approximation of a given low-dimensional non-linear function, however, there are many propositions of the automatic MLP structure allocation methods (Alippi *et al.* 1997, Ash 1989, Bornholdt and Graudenz 1991, Chauvin 1989, Doering *et al.* 1997, Fahlman and Lebierre 1990, Frean 1990, Harp *et al.* 1989, Hassibi and Stork 1993, Koza and Rice 1991, LeCun *et al.* 1990, Marshall and Harrison 1991, Mezard and Nadal 1989, Miller *et al.* 1989, Obuchowicz 1998, Obuchowicz 2000a, Wang *et al.* 1994). But, in the case of more complex modeled systems, high-dimensional or dynamic, automatic algorithms become indispensable.

The significance of network architecture optimization increases when the DMLP is taken into considerations. Dynamic nature of DMLP is very sensitive to changes in the network structure and then a suitable selection of the DMLP architecture is very important. The methods of the artificial intelligence searching, like the simulated annealing (Obuchowicz 1998), the tabu search (Obuchowicz and Patan 2003), the $A^\star$ algorithm (Obuchowicz 1999c) and evolutionary algorithms (Obuchowicz and Politowicz 1997) seem to be very attractive for this task.

5.1. Considered neural networks

The ANN is represented by a ordered pair $NN = (NA, \boldsymbol{v})$ (Doering *et al.* 1997, Obuchowicz 2000a). NA denotes the ANN architecture:

$$NA = (\{V_i \mid i = 0, \dots, M\}, \mathcal{E}). \tag{5.1}$$

$\{V_i \mid i = 0, \dots, M\}$ is a family of $M + 1$ sets of neurons, called *layers*, including at least two non-empty sets V_0 and V_M that define $s_0 = \mathrm{card}(V_0)$ input and $s_M = \mathrm{card}(V_M)$ output units, respectively, $\mathcal{E}$ is a set of connections between neurons in the network. The vector $\boldsymbol{v}$ contains all free parameters of the network, among which the set of weights of synaptic connections $\boldsymbol{w} : \mathcal{E} \to \mathbb{R}$ are.

In general, sets $\{V_i \mid i = 0, \dots, M\}$ have not to be disjunctive, thus, there can be input units which are also outputs of the NN. Units which do not belong to either V_0 or V_M are called *hidden* neurons. If there are cycles of synaptic connections in the set $\mathcal{E}$, then we have a dynamic network.

5.1.1. Multi-Layer Perceptron

The most popular type of the neural network $NN = (NA, \mathbf{v})$ is the MLP. The MLP is based on the McCulloch-Pitts neurons (McCulloch and Pitts 1943) and its architecture possesses following properties:

$$\forall i \neq j \qquad V_i \cap V_j = \emptyset, \tag{5.2}$$

$$\mathcal{E} = \bigcup_{i=0}^{M-1} V_i \times V_{i+1}. \tag{5.3}$$

Layers in the MLP are disjunctive. The main task of the input units of the layer V_0 is preliminary input data processing $\boldsymbol{u} = \{u_p \mid p = 1, 2, \ldots, P\}$ and passing them onto units of the hidden layer. Data processing can comprise e.g. scaling, filtering or signal normalization. Fundamental neural data processing is carried out in hidden and output layers. It is necessary to notice that links between neurons are designed in such a way that each element of the previous layer is connected with each element of the next layer. There are no feedback connections. Connections are assigned with suitable weight coefficients, which are determined, for each separate case, depending on the task the network should solve.

The fundamental training algorithm for the MLP is the BP algorithm (Rumelhart *et al.* 1986, Werbos 1974). This algorithm is of iterative type and it is based on minimization of a sum-squared error utilizing optimization gradient descent method. Unfortunately, the standard BP algorithm is slowly convergent, however, is widely used and in a few recent years its numerous modifications and extensions have been proposed (Świąć and Bilski 2000), e.g.: Chan's and Fallside's algorithm (1997), the delta-delta algorithm (Jacobs 1988), Quickprop algorithm (Fahlman 1988), Silva's and Almeida's algorithm (1990), Park's,Yun's and Kim's algorithm (1992), RPROP algorithm (Riedmiller and Braun 1992), and Levenberg-Marquardt algorithm (Hagan and Menhaj 1994).

Neural networks with the MLP architecture owe their popularity to many effective applications, e.g. in the pattern recognition problems (Looney 1997, Sharkey 1999) and approximation of the non-linear functions (Hornik *et al.* 1989). It is proved that using the MLP with only one hidden layer and suitable number of neurons, it is possible to approximate any non-linear static relation with arbitrary accuracy (Cybenko 1989, Hornik *et al.* 1989). Thus, taking relatively simple algorithms applied to the MLP learning into consideration, this type of networks becomes a very attractive tool for building models of static systems.

5.1.2. Dynamic neural model

In this chapter, the general structure of a neuron model proposed by Ayoubi (1994) is considered. Dynamics is introduced to the neuron in such a way that the neuron activation depends on its internal states. It is done by introducing a linear dynamic system – IIR filter – to the neuron structure (Fig. 5.1) which is called the Dynamic Neuron Model. Three main operations are performed in this dynamic structure. First of all, the weighted sum of inputs is calculated according to the formula :

$$\phi(k) = \boldsymbol{w}^T \boldsymbol{u}(k), \tag{5.4}$$

where $\boldsymbol{w} = \{w_p \mid p = 1, 2, \ldots, P\}$ denotes the input weights vector, P is the number of inputs, and $\boldsymbol{u}(k) = \{u_p(k) \mid p = 1, 2, \ldots, P\}$ is the input vector. The weights perform a similar role as in static feed-forward networks. The weights together with the activation function are responsible for approximation properties

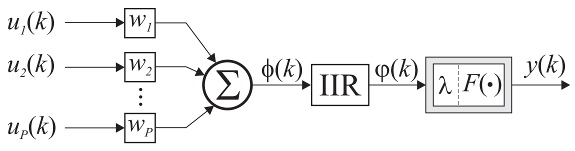


Fig. 5.1 . General structure of DNM with P inputs.

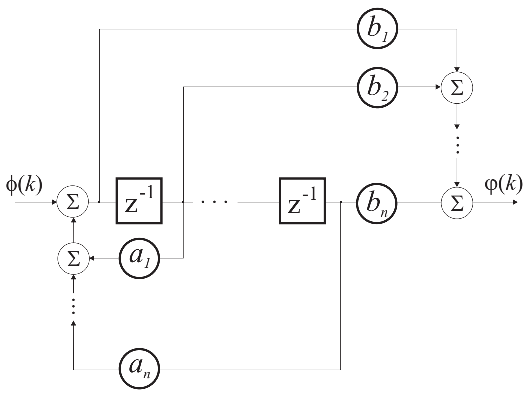


Fig. 5.2 . Block scheme of nth order IIR filter.

of the model. Then this calculated sum $\phi(k)$ is passed to the IIR filter. Here, the filters under consideration are linear dynamic systems of different orders, viz. the first, second and third order. The general structure of the n-th order IIR filter is shown in Fig. 5.2. This filter consists of delay elements (denoted by z^{-1}) and feedback and feedforward paths weighted by the vector weights $\boldsymbol{a} = \{a_i \mid i = 1, 2, \ldots, n\}$ and $\boldsymbol{b} = \{b_i \mid i = 1, 2, \ldots, n\}$, respectively. The behaviour of this linear system can be described by the following difference equation:

$$\begin{aligned}\varphi(k) = {} & b_0\phi(k) + b_1\phi(k-1) + \cdots + b_n\phi(k-n) \\ & - a_1\varphi(k-1) - \cdots - a_n\varphi(k-n),\end{aligned} \tag{5.5}$$

where $\phi(k)$ is the filter input, $\varphi(k)$ is the filter output, and k is the discrete-time index. Finally, the neuron output can be described by:

$$y(k) = F\big(\lambda\varphi(k)\big), \tag{5.6}$$

where $F(\cdot)$ is a non-linear activation function that produces the neuron output $y(k)$, and λ is the slope parameter of the activation function. In the dynamic neuron the slope parameter λ as well as weights $\boldsymbol{w}$ and feedback $\boldsymbol{a}$ and feed-forward $\boldsymbol{b}$ filter weights are trained during a learning process.

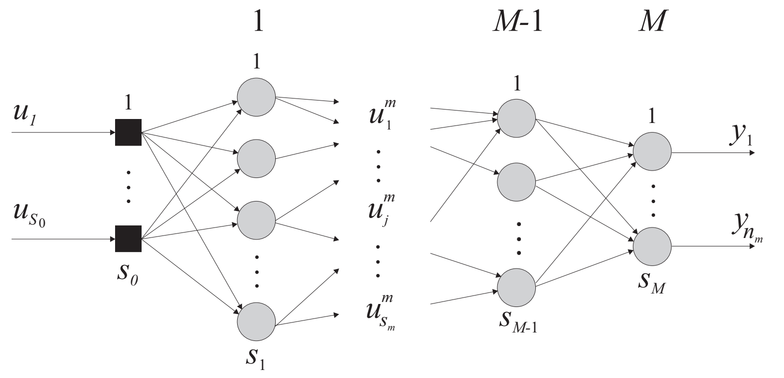


Fig. 5.3 . Architecture of the DMLP.

5.1.3. Dynamic MLP

Considering the dynamic neuron model described above, one can design more complex structure - a neural network. Using the well-known MLP structure with DNM units as nodes a network of dynamic neurons, called Dynamic MLP (DMLP), is defined (Fig. 5.3). The DMLP presents an ordered pair $(NA, \boldsymbol{v})$. NA denotes the network architecture (5.1)

$$\begin{aligned} NA =& \big(\{V_m \mid m = 0, 1, \ldots, M\}, \\ & \{o_s^m \mid m = 1, 2, \ldots, M; s = 1, 2, \ldots, s_m\}, \mathcal{E}\big), \end{aligned} \tag{5.7}$$

where $\{V_m \mid m = 0, 1, \ldots, M\}$ is a family of $M + 1$ layers of DNM units. $\{o_s^m \mid m = 1, 2, \ldots, M; s = 1, 2, \ldots, s_m\}$ is a set of natural numbers, o_s^m denotes the IIR dynamic order of the s-th DNM unit from the m-th layer, $s_m = \operatorname{card}(V_m)$. $\mathcal{E} = \bigcup_{m=0}^{M-1} V_m \times V_{m+1}$ is a set of edges that define the connections between units in the network. The vector of network parameters $\boldsymbol{v}$ can be expressed in the following way:

$$\boldsymbol{v} = \big(\boldsymbol{w}, \{(\boldsymbol{a}_s^m, \boldsymbol{b}_s^m, \lambda_s^m) \mid m = 1, 2, \ldots, M; s = 1, 2, \ldots, s_m\}\big), \tag{5.8}$$

where the set of weights $\boldsymbol{w}$ assigns a real value to each connection, $\{(\boldsymbol{a}_s^m, \boldsymbol{b}_s^m, \lambda_s^m) \mid m = 1, 2, \ldots, M; s = 1, 2, \ldots, s_m\}$ describes feedback and feedforward IIR synaptic vectors and the slope parameter, respectively, of the s-th DNM unit from the m-th layer.

It can be proved by applying the Leontaritis and Bilings theorem (1985) that the DMLP is a universal identifier. They proved that under some assumptions, any nonlinear, discrete and time-invariant system can be represented by a simplified version of the NARMAX model (Nonlinear Auto Regressive Moving Average with eXogenous inputs).

5.2. Problem statement

Let us consider the network which has to approximate a given function $f(\boldsymbol{u})$. Let $\Phi = \{(\boldsymbol{u}, \boldsymbol{y})\}$ be a set of all possible (usually uncountably many) pairs of vectors from the domain $\boldsymbol{u} \in \mathbb{D} \subset \boldsymbol{R}^{s_0}$ and from the range $\boldsymbol{y} \in \mathbb{D}' \subset \boldsymbol{R}^{s_M}$ which realize the relation $\boldsymbol{y} = f(\boldsymbol{u})$. The goal is to construct a *NN* with an architecture NA^{opt} and a set of parameters $\boldsymbol{v}^{opt}$, which fulfills the relation $\boldsymbol{y}_{NA,\boldsymbol{v}} = f_{NA,\boldsymbol{v}}(\boldsymbol{u})$, that a given cost function $\sup_{(\boldsymbol{u},\boldsymbol{y})\in\Phi} J_T(\boldsymbol{y}_{NA,\boldsymbol{v}}, \boldsymbol{y})$ will be minimized. So, the following pair has to be found

$$\left(NA^{opt}, \boldsymbol{v}^{opt}\right) = \arg\min \left[\sup_{(\boldsymbol{u},\boldsymbol{y})\in\Phi} J_T\left(\boldsymbol{y}_{NA,\boldsymbol{v}}, \boldsymbol{y}\right)\right]. \tag{5.9}$$

Practically, the solution of the above problem is not possible to obtain because of the infinite cardinality of the set Φ. Thus, in order to estimate the solution, two finite sets $\Phi_L, \Phi_T \subset \Phi$ are selected. The set Φ_L are used in the learning process of the network of the architecture *NA*:

$$\boldsymbol{v}^{\star} = \arg\min_{\boldsymbol{v}\in\mathcal{V}} \left[\max_{(\boldsymbol{u},\boldsymbol{y})\in\Phi_L} J_L\left(\boldsymbol{y}_{NA,\boldsymbol{v}}, \boldsymbol{y}\right)\right], \tag{5.10}$$

where $\mathcal{V}$ is the space of network parameters. In general, cost functions of the learning $J_L(\boldsymbol{y}_{NA,\boldsymbol{v}}, \boldsymbol{y})$ and testing $J_T(\boldsymbol{y}_{NA,\boldsymbol{v}}, \boldsymbol{y})$ processes can have different definitions. The set Φ_T is used in searching process of $NA^{\star}$, for which

$$NA^{\star} = \arg\min_{NA\in\mathcal{A}} \left[\max_{(\boldsymbol{u},\boldsymbol{y})\in\Phi_T} J_T\left(\boldsymbol{y}_{NA,\boldsymbol{v}^{\star}}, \boldsymbol{y}\right)\right], \tag{5.11}$$

where $\mathcal{A}$ is the space of neural network architectures. Obviously, the solutions of both tasks, (5.10) and (5.11), need not necessary be unique. Than a definition of an additional criterion is needed.

There are many definitions of the selection of the best neural network architecture. The most popular are (Obuchowicz 1998):

- *minimization of the number of network free parameters.* In this case, the subset

$$\mathcal{A}_\delta = \left\{NA : J_T\left(\boldsymbol{y}_{NA,\boldsymbol{v}^{\star}}, \boldsymbol{y}\right) \le \delta\right\} \subset \mathcal{A} \tag{5.12}$$

  is looked for. The network with architecture $NA \in \mathcal{A}_\delta$ and the smallest number of training parameters is considered to be optimal. This criterion is crucial, when the VLSI implementation of the neural network is planned.

- *maximization of the network generalization ability.* The sets of training Φ_L and testing Φ_T patters have to be disjunctive $\Phi_L \cap \Phi_T = \emptyset$. Then, J_T is the conformity measure between network reply on testing patterns and desired outputs. Usually, both quality measures J_L and J_T are similarly defined

$$J_{L(T)}\left(\boldsymbol{y}_{NA,\boldsymbol{v}}, \boldsymbol{y}\right) = \sum_{k=1}^{\mathrm{card}(\Phi_{L(T)})} \left(\boldsymbol{y}_{NA,\boldsymbol{v}} - \boldsymbol{y}\right)^2. \tag{5.13}$$

Restriction of the number of training parameters is the minor criterion in this case. The above criterion is important for approximating networks or neural models.

- *maximization of the noise immunity.* This criterion is applied in networks applied in classification or pattern recognition problems. The quality measure is the maximal noise level of the pattern which is still recognized by the network.

Two first criterions are correlated. Gradually decreasing number of hidden neurons and synaptic connections causes the drop of non-linearity level of the network mapping, and then the network generalization ability increases. The third criterion needs some redundancy of the network parameters. This fact usually clashes with previous criterions. For the most part of publications, the second criterion is chosen.

The quality of the estimates obtained via neural networks strongly depends on selection finite training Φ_L and testing Φ_T sets. Small network structures may not be to able to approximate the desired relation between inputs and outputs with the satisfying accuracy. On the other hand, if the number of network free parameters is to large (in comparison with $\mathrm{card}(\Phi_L)$), then the function $f_{NA^\star,\boldsymbol{v}^\star}(\boldsymbol{u})$ realized by the network strongly depends on the actual set of training patterns (*the bias/variance dilemma*, (Geman *et al.* 1992)).

It is very important to note that the efficiency of the method of the neural network architecture optimization strongly depends on the used learning algorithm. In the case of multi-modal topology of the network error function, the effectiveness of the classical learning algorithms based on the gradient descent method (e.g. the BP algorithm and its modifications) is limited. These methods usually localize some local optimum and the superior algorithm searching for the optimal architecture receives wrong information about the learned network quality.

The problem of the optimal network design, described by relations (5.9), (5.10) and (5.11), is applied to the stationary case and can be simply extended to the problem of the design of the optimal dynamic neural model.

Let

$$\boldsymbol{y}(k) = f\big(\boldsymbol{u}(k), \boldsymbol{u}(k-1), \ldots, \boldsymbol{u}(k-n), \boldsymbol{y}(k), \boldsymbol{y}(k-1), \ldots, \boldsymbol{y}(k-n')\big) \quad (5.14)$$

is the response of a non-linear dynamic system $f(\cdot)$ on an input signal $\boldsymbol{u}(k)$. Let $\Phi = \{\boldsymbol{u} : K \to \boldsymbol{R}^{s_0}\}$ is a family of all possible maps (infinitely many) from the set K of discrete time moments to the space $\boldsymbol{R}^{s_0}$ of input signals. The ultimate goal of the construction of a neural model (with an architecture NA and a set of network parameters $\boldsymbol{v}$)

$$\begin{aligned} \boldsymbol{y}_{NA,\boldsymbol{v}}(k) \quad = \quad & f_{NA,\boldsymbol{v}}\big(\boldsymbol{u}(k), \boldsymbol{u}(k-1), \ldots, \boldsymbol{u}(k-n_{NA}), \\ & \boldsymbol{y}_{NA,\boldsymbol{v}}(k), \boldsymbol{y}_{NA,\boldsymbol{v}}(k-1), \ldots, \boldsymbol{y}_{NA,\boldsymbol{v}}(k-n'_{NA})\big) \end{aligned} \quad (5.15)$$

of a dynamic system (5.14) is the minimization of a cost function $\sup_{\boldsymbol{u}(k)\in\Phi} J_T\big(\boldsymbol{y}_{NA,\boldsymbol{v}}(k), \boldsymbol{y}(k) \mid k \in K\big)$. Thus one has to determine the following pair :

$$(NA^{opt}, \boldsymbol{v}^{opt}) = \arg\min \left[\sup_{\boldsymbol{u}(k)\in\Phi} J_T\big(\boldsymbol{y}_{NA,\boldsymbol{v}}(k), \boldsymbol{y}(k) \mid k \in K\big)\right]. \tag{5.16}$$

Similarly as in the static case, the solution of the (5.16) cannot be achieved in real systems, because of infinite size of Φ. Hence, two finite subsets $\Phi_L, \Phi_T \subset \Phi : \Phi_L \cap \Phi_T = \emptyset$ are separated. The set Φ_L is used to determine the best vector of parameter of a given network architecture *NA*:

$$\boldsymbol{v}^* = \arg\min_{\boldsymbol{v}\in\mathcal{V}} \left[\sup_{\boldsymbol{u}(k)\in\Phi_L} J_L\big(\boldsymbol{y}_{NA,\boldsymbol{v}}(k), \boldsymbol{y}(k) \mid k \in K\big)\right]. \tag{5.17}$$

The set Φ_T is used to determine the network architecture *NA* that realizes the minimal cost within the set of all network architectures $\mathcal{A} = \{NA\}$:

$$NA^* = \arg\min_{NA\in\mathcal{A}} \left[\sup_{\boldsymbol{u}(k)\in\Phi_T} J_T\big(\boldsymbol{y}_{NA,\boldsymbol{v}^*}(k), \boldsymbol{y}(k) \mid k \in K\big)\right]. \tag{5.18}$$

5.3. ESSS algorithms in the MLP learning process

However, the emphasis of this part is put on the dynamic neural models design, it is interesting to check the effectiveness of algorithms from the ESSS family (Chapters 1 and 2), in the problem of the MLP learning. The MLP, which is usually learned by BP algorithm, is a type of ANNs used the most often by researches and engineers. The surface topology of the error function of the MLP is multimodal and knowledge about it is limited. The BP algorithm, which is based on the gradient-descent method, usually gets stuck in local minimum and is terminated too early. Thus, global optimization algorithms, which are able to cross saddles of the error function surface, may be an interesting solution. There is a reach bibliography on genetic algorithms applications to neural network training. They are used in feed-forward networks (Kwaśnicka and Szerszon 1997, Montana and Davis 1989, Muselli and Ridella 1991, Reeves and Steele 1992, Rutkowska *et al.* 1997, Yao 1993), and Kohonen networks (Harp and Samad 1992).

The ESSS algorithm has been successfully applied to learn a simple MLP for the XOR problem (Makuch *et al.* 1996). The aim of this section is to analyze the efficiency of the ESSS and ESSS-SVA algorithms as learning methods of the MLP with much complex structure than that of the XOR network (Obuchowicz and Patan 1997b). An approximation of a two-variable version of the De Jong's function F3 (De Jong 1975) is chosen as an exemplary problem for an MLP. This function has the form

$$f(u_1, u_2) = \alpha\big(\lfloor u_1 \rfloor + \lfloor u_1 \rfloor\big), \tag{5.19}$$

where $u_1, u_2 \in (-5.12, 5.12)$, $\lfloor\cdot\rfloor$ rounds a real number to the nearest integer towards $-\infty$, and $\alpha = 0.08$ is a normalization factor such that $f(u_1, u_2) \in (-1, 1)$ for argument values in the considered region.

Tab.5.1. Efficiency of four tested learning methods of the MLP considered in the sense of factor J_T (5.21). Percentage of simulations with J_T in a given interval.

algorithm	BP	BPA	ESSS	ESSS-SVA
$J_T \leq 0.1$	0	47	4	32
$0.1 < J_T \leq 0.25$	0	29	94	68
$J_T > 0.25$	100	24	2	0

The trained MLP consists of two input units, one output neuron, *bias* unit and two hidden layers of seven and four neurons, respectively. A hyperbolic tangent function is chosen as the activation function of each neuron. The initial weights are chosen from a uniform distribution on the interval $(-1, 1)$. Similarly, a hundred training pairs are chosen from a uniform distribution on the considered region of u_1, u_2.

The MLP is trained independently by four algorithms: BP (Rumelhart *et al.* 1986), BP with adaptive learning rate (BPA) (Demuth and Beale 1993), the ESSS and ESSS-SVA algorithms, in the off-line course, i.e. the sum of squared errors of all training pairs is minimized. The fitness function for ESSS and ESSS-SVA has to be non-negative and is chosen in the form:

$$\Phi(u_1, u_2) = n|b-a| - \sum_{i=1}^{n} \Big(y_{NA,\boldsymbol{v}}(u_1^{(i)}, u_2^{(i)}) - f(u_1^{(i)}, u_2^{(i)}) \Big)^2, \tag{5.20}$$

where n is the number of training pairs, $(a, b)\big(=(-1,1)\big)$ is the interval of output values, $\{((u_1^{(i)}, u_2^{(i)}), f(u_1^{(i)}, u_2^{(i)})) \mid i = 1, 2, \ldots, n\}$ is a set of training pairs, and $\{y_{NA,\boldsymbol{v}}(u_1^{(i)}, u_2^{(i)}) \mid i = 1, 2, \ldots, n\}$ is a set of network answers.
In order to compare the learning methods, a factor J_T is defined:

$$J_T = \frac{\iint_\Delta \Big(y_{NA,\boldsymbol{v}}(u_1, u_2) - f(u_1, u_2) \Big)^2 du_1 du_2}{\iint_\Delta f^2(u_1, u_2) du_1 du_2}, \tag{5.21}$$

where Δ is the input vector space.

In order to compare the efficiency of the proposed algorithms with the existing approaches, a number (about 400 experiments for each algorithm) of computational experiments were carried out. The corresponding results are listed in Table 5.1.

The BP algorithm cannot teach the network approximating the considered function in all samples. It gets stuck in local minima. If the MLP is learned successfully by the BPA algorithm, the optimal point in the weight space is usually reached perfectly. But about a quarter of simulations gives bad results. In practice, the ESSS and ESSS-SVA algorithms stop with success. In the case of the ESSS, the accuracy of the global optimum location is weak. It results from the fact that the value of the standard deviation σ of the normal distribution used in parents' modification (Table 1.3) is too large for a high accuracy of extreme fitting. But decreasing of σ decreases the efficiency of saddle crossing. The σ-adaptation mechanism used in the ESSS-SVA algorithm improves the searching accuracy.

Unfortunately, the ESSS and ESSS-SVA algorithms possess some critical defect, which is the feature of all evolutionary methods. This is extremely long time of searching the solution. They operate on populations of power from several dozens to hundreds points of the multi-dimensional weight space. These feature of the ESSS and ESSS-SVA methods suggests that a hybrid method, which combines them with a method of local optimization, can be more efficient.

5.4. Learning techniques for DMLP

Let us consider an M-layered network with dynamic neurons described by differentiable activation functions $F(\cdot)$. The activity $u_s^m(k)$ of the s-th neuron in the m-th layer is defined by

$$u_s^m(k) = F\left(\lambda_s^m\left(\sum_{i=0}^{n} b_{is}^m \sum_{p=1}^{S_{m-1}} w_{sp}^m u_p^m(k-i) - \sum_{i=1}^{n} a_{is}^m \varphi_s^m(k-i)\right)\right) \tag{5.22}$$

The main objective of the learning process is to adjust all the unknown network parameters $\boldsymbol{v}$ (5.8) based upon a given training set of input-output pairs. The MLP structure of the DMLP suggests that some kind of the BP algorithm can be implemented. However, if an internal recurrence is presented (Fig. 5.1), the localized calculation of the gradient becomes difficult, because the present output of the network $\boldsymbol{y}_{NA,\boldsymbol{v}}(k)$ depends on the past outputs. In order to solve this problem, the Dynamic BP algorithm (DBP) (Baldi 1995) with extension for the DMLP, so called Extended DBP (EDBP) (Patan and Korbicz 1997, Korbicz *et al.* 1998), will be discussed. The EDBP algorithm, similarly to the classical BP, usually finds one of the local unsatisfactory optima. A multi-start version of the EDBP algorithm very seldom ends up successfully (Obuchowicz and Patan 1998). Thus, algorithms of global optimization should be implemented. Three types of the global optimization algorithms were used in the DMLP learning process: genetic algorithms (Patan and Jesionka 1999), stochastic algorithms (Patan and Obuchowicz 1999), and the ESSS-FDM (Obuchowicz 1999a). The last two will be discussed in this section.

5.4.1. Stochastic algorithms

In this section, an attempt to apply stochastic algorithms to training of the dynamic neural network is undertaken. A similar study for a simple multi-layer perceptron was presented by Tu *et al.* (1995). In that work, however, simulation was performed for a very simple XOR problem. In the case of modelling of dynamic non-linear processes, a sum-squared error surface is more complex and multimodal. Pure stochastic algorithms to the DMLP training process was proposed by Patan and Obuchowicz (1999) as a simple technique, which allows to avoid the problem of the local character of the EDBP algorithm. In this section, algorithms which have iterative character are considered. Assuming that the sequence $\boldsymbol{v}^0, \boldsymbol{v}^1, \ldots, \boldsymbol{v}^k$ is already appointed, a way of achieving next point $\boldsymbol{v}^{k+1}$ is formulated (Zieliński and Neumann 1983). All the algorithms described in this section are characterized by a very simple structure.

5.4.1.1. Stochastic algorithms with randomly chosen direction of searching (Algorithm A)

The principle of operation of these algorithms is that the direction of searching is chosen in a random manner, and after that in this direction a step of suitable length is performed. Two cases can be distinguished here: 1) the step rate is established and 2) the step rate is changed during optimization procedure. Let there be given two sequences $(a_k)_{k\geqslant 0} : a_k > 0$ and $(\varepsilon_k)_{k\geqslant 0} : \varepsilon_k > 0 \wedge \varepsilon_k \downarrow 0$. Let a vector sequence $(\boldsymbol{\xi}^k)_{k\geqslant 0} : \boldsymbol{\xi}^k \in U(\mathcal{S}(0,1))$, each $\boldsymbol{\xi}^k$ is randomly chosen from the spherical surface of radius equal to unity with the uniform distribution. The stochastic algorithm with a given learning rate is defined by the formula:

$$\boldsymbol{v}^{k+1} = \begin{cases} \boldsymbol{v}^k + a_k\boldsymbol{\xi}^k & J_L(\boldsymbol{v}^k + a_k\boldsymbol{\xi}^k) < J_L(\boldsymbol{v}^k) - \varepsilon_k, \\ \boldsymbol{v}^k & \text{otherwise,} \end{cases} \tag{5.23}$$

where $\boldsymbol{v}$ is the vector of all network parameters, and $J_L(\boldsymbol{v}^k)$ is the performance index in the form of sum-square error:

$$J_L(\boldsymbol{v}^k) = \sum_{p=1}^{P} \left(y_{NA,\boldsymbol{v}^k}(p) - y(p)\right)^2. \tag{5.24}$$

Taking into account equation (5.23), in the first case one can speak about "success", and in the other case, – about "failure". A new, different from previous, point $\boldsymbol{v}^{k+1}$, only in the case of success is obtained. A point $\boldsymbol{v}^k + a_k\boldsymbol{\xi}^k$ can be treated as a *test point*. The quantity a_k is the step rate, and ε_k is the "improvement threshold".

5.4.1.2. Stochastic algorithms with an estimation of a gradient (Algorithm B)

The stochastic algorithm with an estimation of a gradient is defined according to the formula:

$$\boldsymbol{v}^{k+1} = \boldsymbol{v}^k - a_k\hat{\varphi}\left(\boldsymbol{v}^k\right), \tag{5.25}$$

where $(a_k)_{k\geqslant 0}$ is a given sequence, and $\hat{\varphi}\left(\boldsymbol{v}^k\right)$ is an estimator of $\nabla\varphi\left(\boldsymbol{v}^k\right)$, where $\varphi\left(\boldsymbol{v}^k\right) = J_L(v^k)/\|J_L(v^k)\|$. This estimator can be expressed in the form:

$$\hat{\varphi}\left(\boldsymbol{v}^k\right) = \frac{\hat{J}_L\left(\boldsymbol{v}^k\right)}{\|\hat{J}_L\left(\boldsymbol{v}^k\right)\|}, \tag{5.26}$$

where $\hat{J}_L\left(\boldsymbol{v}^k\right) \neq 0$ is the estimator of the ∇J_L at the point $\boldsymbol{v}^k$. Let $(c_k)_{k\geqslant 0}$: $c_k > 0$ be a sequence of positive numbers, and $(\boldsymbol{\xi}^k)_{k\geqslant 0}$: $\boldsymbol{\xi}^k \in U\left(\mathcal{S}(0,1)\right)$ be a vector sequence of independent random directions with uniform distribution. The estimator $\hat{J}_L\left(\boldsymbol{v}^k\right)$ can be described as:

$$\hat{J}_L\left(\boldsymbol{v}^k\right) = \frac{J_L\left(\boldsymbol{v}^k + c_k\boldsymbol{\xi}^k\right) - J_L\left(\boldsymbol{v}^k\right)}{c_k}\boldsymbol{\xi}^k. \tag{5.27}$$

It is necessary to note that in this case:

$$\hat{\varphi}(\boldsymbol{v}^k) = \operatorname{sgn}\left[J_L\left(\boldsymbol{v}^k + c_k\boldsymbol{\xi}^k\right) - J_L\left(\boldsymbol{v}^k\right)\right]\boldsymbol{\xi}^k \tag{5.28}$$

and the algorithm under consideration can be transformed to the formula:

$$\boldsymbol{v}^{k+1} = \begin{cases} \boldsymbol{v}^k - a_k\boldsymbol{\xi}^k & \text{if } J_L(\boldsymbol{v}^k + c_k\boldsymbol{\xi}^k) > J_L(\boldsymbol{v}^k), \\ \boldsymbol{v}^k + a_k\boldsymbol{\xi}^k & \text{if } J_L(\boldsymbol{v}^k + c_k\boldsymbol{\xi}^k) < J_L(\boldsymbol{v}^k), \\ \boldsymbol{v}^k & \text{otherwise.} \end{cases} \tag{5.29}$$

In this case, the point $\boldsymbol{v}^k + c_k\boldsymbol{\xi}^k$ can be treated as a *test point*. This algorithm admits points with better as well as worse quality (compare with (5.23)). Thus, it takes the feature of crossing saddles in sum-square error landscape.

5.4.1.3. Stochastic algorithms with randomly chosen sample points (Algorithm C)

Let there be two sequences $(r_k)_{k\geqslant 0}$: $r_k > 0$ and $(\varepsilon_k)_{k\geqslant 0}$: $\varepsilon_k > 0 \wedge \varepsilon_k \downarrow 0$. Let a vector sequence $(\boldsymbol{\xi}^k)_{k\geqslant 0}$: $\boldsymbol{\xi}^k \in U\left(\mathcal{K}(0, r_k)\right)$, each $\boldsymbol{\xi}^k$ is randomly chosen from the sphere of radius equal to r_k with the uniform distribution. The stochastic algorithm with a given learning rate is defined by the formula:

$$\boldsymbol{v}^{k+1} = \begin{cases} \boldsymbol{\xi}^k & J_L(\boldsymbol{\xi}^k) < J_L(\boldsymbol{v}^k) - \varepsilon_k, \\ \boldsymbol{v}^k & \text{otherwise.} \end{cases} \tag{5.30}$$

5.4.1.4. Illustrative example

Let us consider the following identification problem of a nonlinear dynamic system described by the equation (Narendra and Parthasarathy 1990):

$$y(k) = \frac{y(k-1)y(k-2)y(k-3)u(k-1)\big(y(k-3)-1\big) + u(k)}{1 + y^2(k-2) + y^2(k-3)}. \tag{5.31}$$

To solve this problem, the DMLP with one input and one output units, and one hidden layer with five units has been applied (Patan and Korbicz 1996). All the neurons have the second order IIR filter (the network of the $\mathcal{N}^2_{1,5,1}$ class), so the considered network has 46 learning parameters. The network structures and

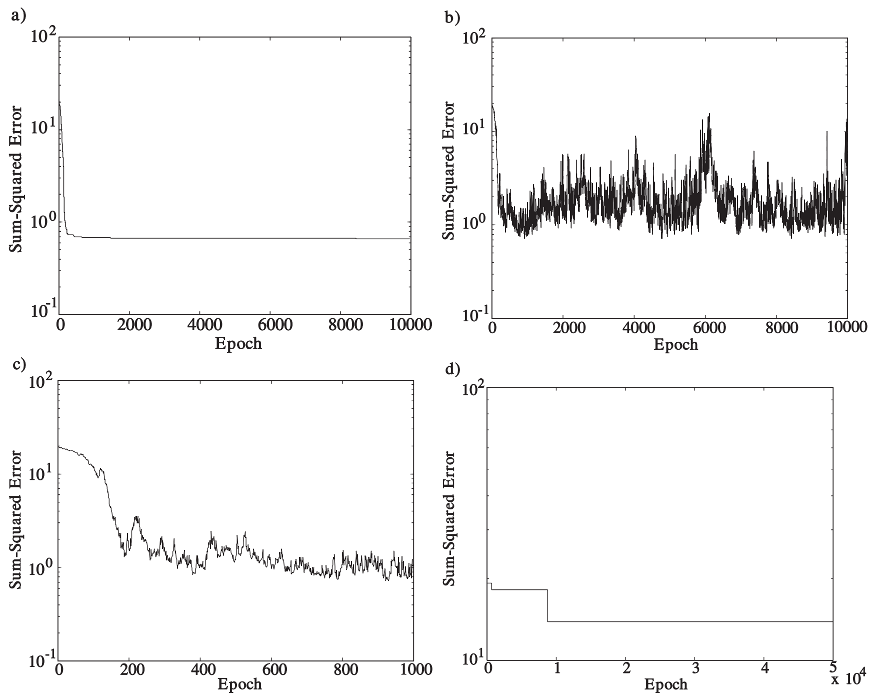


Fig. 5.4. Evolution of training: (a) algorithm A: $a_k = 0.1, \varepsilon_k = 0, k = 1, \ldots, 10^4$; (b) and (c) algorithm B: $a_k = 0.1, c_k = 0.1, k = 1, \ldots, 10^4$; (d) algorithm C: $r_k = 1, \varepsilon_k = 0, k = 1, \ldots, 5 \cdot 10^4$

learning parameters have been chosen experimentally. Training of the network was carried out using, by turns: the stochastic algorithm with a randomly chosen search direction (algorithm A), the stochastic algorithm with gradient estimation (algorithm B), and the stochastic algorithm with randomly chosen sample points (algorithm C) (Patan and Obuchowicz 1999). Illustrative examples of training processing are presented in Fig. 5.4. As it can be seen, none of the proposed algorithms is able to train the network accurately. In the best case the sum-squared error was approximately equal to 0.65. This large error value does not assure a high modelling quality. In spite of the fact that these algorithms fail, some interesting remarks can be formulated.

Both algorithms A and B solve the training problem with an equivalent accuracy. Training with the algorithm A seems to be more stable. It results from the fact that, unlike in the algorithm B, only the best vectors are accepted as a base for the next searching in the algorithm A. The algorithm B sporadically crosses saddles in sum-square error landscape, but benefits following from this fact are not to be noticeable. The efficiency of the algorithm C is the worst. However, in the case of the multi-layer perceptron training, it usually gives the best results. It probably follows from the fact that there is a very rich topology of the sum-square error function, in the case of chosen non-linear system (5.31), and fitting a suitable algorithm parameters is very difficult. However, if an adaptation of algorithm parameters is implemented, the fast stochastic algorithms might train the DMLP with better efficiency. A very important remark is that the stochastic and gradient algorithms can be combined. Thus, a hybrid method is obtained, within which the stochastic algorithm can act as a mechanism which secures running away from local minima.

5.4.2. Evolutionary algorithms

The idea of a soft algorithm application to DMLP learning process was first introduced by Patan and Jesionka (2000, 1999). They used the genetic algorithm to learn the DMLP which had to identify a dynamic system. In order to implement the GA to the DMLP training, the chromosome should contain information about the weights $\boldsymbol{w}$ of synaptic connections between neurons, the feedback $\boldsymbol{a}$ and the feed-forward $\boldsymbol{b}$ parameters for each neural IIR filter and the slope parameters λ for each DNM unit. The chromosome length depends on the number of bits, which code each network parameter. The DMLP considered in (Patan and Jesionka 1999) belongs to the class $\mathcal{N}^1_{1,5,1}$ (one input, five hidden and one output units with the first order IIR). Thus we have 40 network parameters to adjust. Let all the parameters be from the range [-1,1]. If each parameter is to be represented with accuracy equal to 10^{-5}, then we need 18 bits per network parameter and 720 bits per chromosome. If the size of population is equal to 200, then the algorithm operates on 144 thousand bits per epoch (!).

The results obtained are not satisfying (Patan 2000). The GA is not an optimization algorithm in the sense of reaching an optimum with a desired accuracy. It is not asymptotically convergent to an optimum. Thus, the best element obtained in the history of a GA processing is proposed to be the initial one for the local optimization method, e.g. the EDBP algorithm. The resulting DMLP reveals a high quality of system identification. This technique, which combines the GA and EDBP algorithms gives a very good neural model. In order to evaluate the modelling results the discrete version of the quality index (5.21) is defined in the form (Obuchowicz and Patan 1998):

$$J_T = \frac{\sum_{p=1}^{P} \left(y_{NA,\boldsymbol{v}}(p) - y(p)\right)^2}{\sum_{p=1}^{P} y^2(p)}, \tag{5.32}$$

where P is the size of the testing set. The quality indices J_T (5.32) obtained for the DMLP models learned using the GA and EDBP algorithms are lower several times then these DMLP models, which have been learned by the EDBP algorithm only (Patan 2000).

Besides unquestionable advantages, the GA has several drawbacks. First of all, this algorithm is numerically complex, and training time is very long. Moreover, training using the GA can be performed only *off-line*. Therefore, the GAs should be applied in very difficult optimization problems, where classical methods fail, and a very high modelling quality is required.

Some of the GA disadvantages which occur in the DMLP learning process have been overcome using the ESSS-FDM algorithm (Obuchowicz 1999a, Korbicz *et al.* 1999). Unlike the GA, the ESSS-FDM algorithms works using floating point representation and there are no problems with the long chromosomes. The ESSS-FDM algorithm has been successfully implemented in the case of a time-varying performance index (Obuchowicz 1999b), thus the attempt at applying this algorithm to the *on-line* learning process ends in success (Obuchowicz 1999a).

To model Narendra's dynamic system (5.31), a DMLP network belonging to class $\mathcal{N}^2_{1,5,1}$ was chosen (one hidden layer with five neurons). Each neuron contains a second-order IIR filter. Hence, 52 adaptable parameters have to be adjusted in the training process. The parameters of the ESSS-FDM algorithm were as follows: the size of population $\eta = 20$; the momentum $\mu = 0.0545$; the maximum number of iterations $t_{\max} = 5000$; the variance of modification $\sigma = 0.075$ for $t \leq 200$ and $\sigma = 0.015$ for $t > 200$. A set of 500 training patterns for the on-line training process was generated. Figure 5.5 shows the system and neural model outputs for different chosen inputs.

In the case of the DMLP network trained with the ESSS-FDM algorithm, a very good quality ($J_T = 0.0058$) was obtained for the testing signal

$$u(k) = \begin{cases} \sin(2\pi k/250) & \text{for} \quad k \leq 250, \\ 0.8\sin(2\pi k/250) + 0.2\sin(2\pi k/25) & \text{for} \quad k > 250. \end{cases} \tag{5.33}$$

The obtained result is better than all known results for the Narendra's system (5.31) in the literature (Narendra and Parthasarathy 1990, Specht 1991, Patan and Korbicz 1996, Obuchowicz and Patan 1998). From Fig. 5.5 it follows that the performance of the system modelling is high for different inputs. Unfortunately, the ESSS-FDM algorithm is more time-consuming than the EDBP one. Based on this approach a good quality model of the dynamic process can be designed. It was found that the DMLP network with ESSS-FDM learning algorithm can be applied in the cases where high modelling quality is required and the learning time does not matter. In other cases, the EDBP algorithm is recommended to train the DMLP network.

5.5. The MLP architecture optimization

Let us consider the MLP network with two hidden layers and units with sigmoid activation function (Fig. 5.6).

Four spaces can be distinguished: the input space $\mathcal{U}$, and its successive patterns $\mathcal{Y}_{h1} = R_{h1}(\mathcal{U})$, $\mathcal{Y}_{h2} = R_{h2}(\mathcal{Y}_{h1})$ and $\mathcal{Y} = R_o(\mathcal{Y}_{h2})$, where R_{h1}, R_{h2} and R_o are mappings realized by both hidden and output layers, respectively. Numbers of input and output units are defined by dimensions of input and output spaces. The number of hidden units in both hidden layers depends on an approximation problem solved by a network. Further consideration are based on the following theorem (Wang *et al.* 1992):

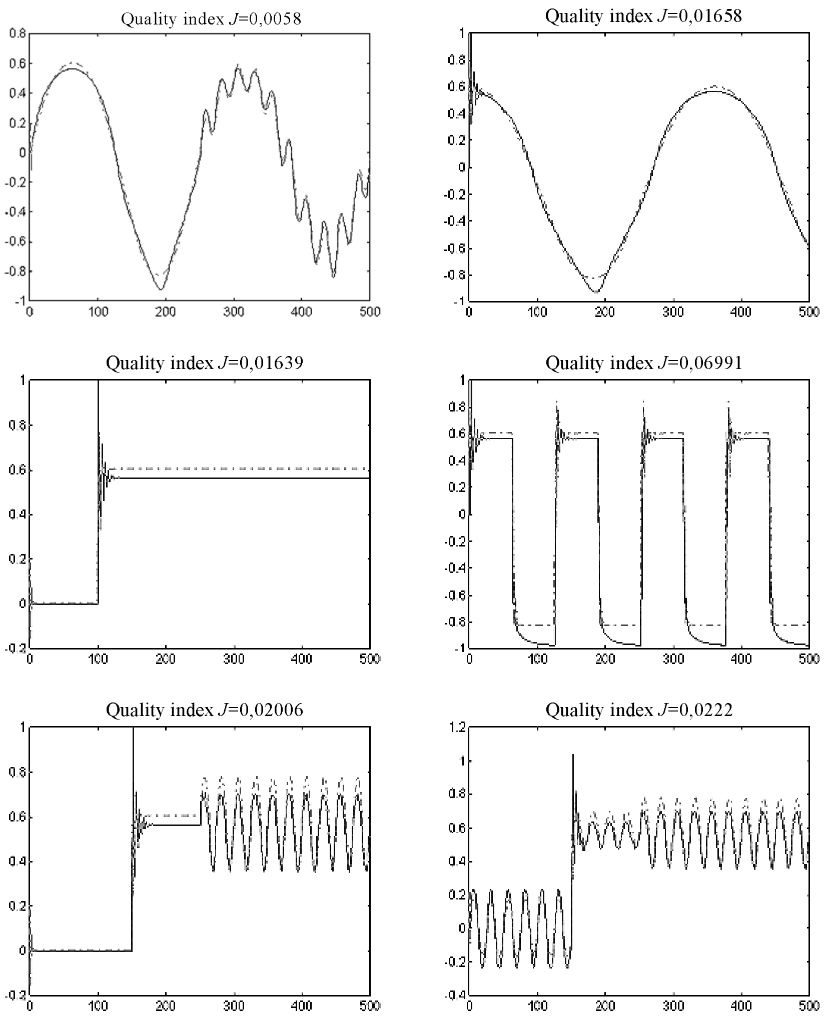
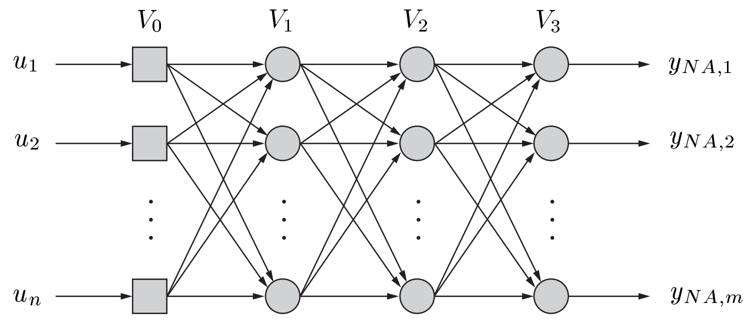


Fig. 5.5 . The system output (solid line) and the DMLP output (dotted line) to a set of chosen inputs.



$$\begin{tikzcd}\mathcal{U} \arrow[r, "R_{h1}"] & \mathcal{Y}_{h1} \arrow[r, "R_{h2}"] & \mathcal{Y}_{h2} \arrow[r, "R_o"] & \mathcal{Y}\end{tikzcd}$$

Fig. 5.6 . The MLP network with three layers

Theorem 5.5.1. *Let Φ_L be a finite set of training pairs associated with finite and compact manifolds. Let f be some continuous function. Taking into account the space of three-level MLPs, there exists an unambiguous approximation of the canonical decomposition of the function f, if and only if the number of hidden neurons in each hidden layer is equal to the dimension of subspace of the canonical decomposition of the function f.*

Theorem 5.5.1 gives necessary and sufficient conditions for existing of the MLP approximation of the canonical decomposition of any continuous function. These conditions are following. The $\mathcal{U}$ and $\mathcal{Y}$ must be fully represented by the training set Φ_L. The network contains more than two hidden layers, which are enough for implementation of the considered approximation of canonical decomposition of any continuous function. The goal of the first hidden layer is to map the n-dimensional input space $\mathcal{U}$ into the space $\mathcal{Y}_{h1} = R_{h1}(\mathcal{U})$, which is inverse image of the output space in the sense of the function f. Thus, the mapping $\mathcal{Y}_{h1} \to \mathcal{Y}$ is invertible. The number of units in the first hidden layer $\text{card}(V_1)$ is equal to a dimension of the minimal space, which still fully represent input data, and is, in general, lower than the dimension of input vectors.

Theorem 5.5.1 guarantees, that an approximation of the canonical form of function f exists and is unambiguous. If $\text{card}(V_1)$ is higher than the dimension of the canonical decomposition space of the function f, the network does not approximate the canonical decomposition, but can still be the best approximation of the function f. However, such an approximation is not unambiguous, and depends on the initial condition of the learning process. On the other hand, if the number $\text{card}(V_1)$ is too low, the obtained approximation is not optimal. So, both deficiency and excess of neurons in the first hidden layer lead to poor approximation.

As it has been pointed out above, the first layer reduces the dimension of the actual input space to the level sufficient for the optimal approximation. Next two layers, second hidden and output, are sufficient for realization such an approximation (Cybenko 1989, Hornik *et al.* 1989). The number of units in the second hidden layer card(V_2) is determined by an assumed error of the approximation. Lowest error needs higher card(V_2). The crucial tradeoff one has to make is between the learning capability of the MLP and fluctuations due to the finite sample size. If card(V_2) is too small, network might not be able to approximate well enough the functional relationship between the input and target output. If card(V_2) is too great (compared to the number of training samples), the realized network function will depend too much on the actual realization of the training set (Geman *et al.* 1992).

The above consideration suggests, that the MLP can be used to approximation of a canonical decomposition of any function specified on the compact topological manifold. The following question comes to mind: why is the canonical decomposition needed? Usually, essential variables, which fully describe the input-output relation, are not precisely defined. Thus, the approximation of this relation can be difficult. The existence of the first layer allows to transform a real data to the form of the complete set of variables of an invertible mapping. If the input space agrees with the inverse image of the approximated mapping, the first hidden layer is unnecessary.

5.5.1. Methods classification

Procedures, which search the optimal ANN architecture, have been studied for dozen or so years. Especially an escalation of papers took place in 1989–1991. At that time almost all standard solutions were published. In subsequent years the number of publications significantly decreases. Most of proposed methods were dedicated to specific types of neural networks. But new results are still needed.

There are very rich bibliography items and various methods to solve this problem. Recently, a variety of architecture optimization algorithms have been proposed. They can be divided into three classes (Doering *et al.* 1997, Obuchowicz 2000a):

- bottom-up approaches,
- top-down approaches,
- discrete optimization methods.

Starting with a relatively small architecture, bottom-up procedures increase the number of hidden units and thus increase the power of the growing network. One of the first methods was proposed by Mezard and Nadal (1989). Their *tiling algorithm* is dedicated for the MLP, which have to map Boolean functions of binary inputs. Creating subsequent layers neuron by neuron the tiling algorithm successively reduces the number of training patters, which are not linearly separable. Similar approach was introduced by Frean (1990). Both algorithms give MLP architectures in a finite time, and this architectures aspire to be almost optimal. In (Hirose *et al.* 1991) the extension of the back-propagation algorithm has been proposed. This algorithm allows to add or reduce hidden units depending on an actual position of the training process. Ash (1989), Setiono and Chi Kwong Hui (1995) proposed that the training process of the sequential created networks is initiated using values of parameters from previous obtained networks. Wang and co-workers (1994) built an algorithm based on the their Theorem 5.5.1 (Wang *et al.* 1992) , which describes necessary and sufficient conditions that there exists a neural network approximation of an canonical decomposition of any continuous function. The cascade-correlation algorithm (Fahlman and Lebierre 1990) builds an ANN of original architecture. The bottom-up methods prove to be the most flexible approach, though computationally expensive, complexity of all known algorithms is exponential. Several bottom-up methods have been reported to train even hard problems with a reasonable computational effort. The resulting network architectures can hardly be proven to be optimal. But, a further criticism concerns the insertion of hidden neurons as long as elements of the training set are misclassified. Thus the resulting networks posses a poor generalization performance and are disqualified for many applications.

Most of neural networks applications uses the neural model of binary, bipolar, sigmoid or hyperbolic tangent activation function. A single unit of this type represents a hyperplane which separates its input space into two subspaces. Through the serial-parallel units connections in network, the input space is divided into subspaces which are polyhedral sets. The idea of the top-down methods is gradual reduction of the hidden unit number in order to simplify shapes of the division of the input space. In this way the generalization property can be improved. Three classes of top-down application may be distinguished:

- sensitive methods,
- penalty function methods, and
- covariance analysis methods.

A sensitivity of an synaptic connection is a measure of the influence of this connection reduction on a quality measure of the network. First definitions of the sensitivity measure was proposed by Mozer and Smolensky (1989) and Karnin (1990), but the most known sensitivity algorithms are Optimal Brain Damage (LeCun *et al.* 1990) and its extension: Optimal Brain Surgeon (Hassibi and Stork 1993). The idea of the penalty function methods is modification of the quality measure of an ANN by adding a factor which penalizes a network for the excess of architecture elements (Chauvin 1989, Hertz *et al.* 1991). The topological optimization may be provided by the distribution analysis of the eigenvalues of the covariance matrix of output signals of hidden units. It is assumed that as many hidden units can be reduced as there are eigenvalues negligently small (Weigend and Rumelhart 1991). Alippi and co-workers (1997) transform the covariance matrix of output signals of hidden units into diagonal matrix using, so called, virtual layer, and than the generalization performance is improved by reducing virtual neurons with insignificant output signals. The top-down approaches inherently assume knowledge of a sufficiently complex network architecture that can always be provided for finite size training samples. Because the algorithms presented up to now can only handle special cases of redundancy reduction in a network architecture, they are likely to result in a network that still oversized. In this case the cascade-reduction method (Obuchowicz 1999d), where the obtained architecture using a given top-down method is an initial architecture for next searching process, can be a good solution.

The space of ANN architectures is an infinite discrete space and there are very rich bibliography items about implementation of discrete optimization methods to solve the ANN architecture optimization problem. In particular, evolutionary algorithms, especially genetic algorithms, seem to have gained a strong attraction within this context (c.f. (Bornholdt and Graudenz 1991, Harp *et al.* 1989, Kitano 1990, Koza and Rice 1991, Marshall and Harrison 1991, Miller *et al.* 1989, Nagao *et al.* 1993, Obuchowicz and Politowicz 1997)). Nevertheless, implementations of the $A^\star$ algorithm (Doering *et al.* 1997, Obuchowicz 1999c) and the Simulating Annealing (Obuchowicz 2000a) deserve an attention.

One of the most interesting approaches, proposed by Doering and co-workers (1997), where the crucial point certainly is the efficient use of information already gained during training a sequence of network architectures. The $A^\star$-algorithm is applied. It is known that it uses heuristic information in an optimal way and thus is superior to all other algorithms working with the same heuristic information, i.e., it finds the optimal architecture by exploring the smallest possible subset of the search space.

5.5.2. Evolutionary algorithms approach to ANN architecture optimization

Application of the evolutionary algorithms to the construction process of neural tools has just a history of a dozen or so years. Evolutionary algorithms can be used in three types of problems:

- learning process of an ANN with fixed architecture;
- searching for an optimal ANN architecture, the learning process is done using another method, e.g. the BP algorithm;
- solving both above problems simultaneously.

The first type of problems was considered in Section 5.3, the others are the subject of this point. Among all known EAs, genetic algorithms seem to be the most natural tool for searching a discrete space of ANN architectures. This fact results from the classical structure of a chromosome — a string of elements from a discrete set, e.g. a binary set.

The most popular representation of the ANN architecture is a binary string (Bornholdt and Graudenz 1991, Harp *et al.* 1989). At first, an initial architecture $NA_{\max}$ must be chosen. This architecture must be sufficient to realization of a desired input-output relation. The $NA_{\max}$ defines the upper limit of searching architectures complexity. Next, all units of the $NA_{\max}$ have to be numbered from 1 to N. In this way, the searching space of ANN architectures is limited to class of all digraphs of N nodes. Any architecture NA (a graph) of this class is represented by its connection matrix $\mathbb{V}$ of elements equal to 0 or 1. If $V_{ij} = 1$ then there exists the sinaptic connection from i-th unit to j-th one, $V_{ij} = 0$ otherwise. A chromosome is built by rewriting the matrix $\mathbb{V}$ row by row to a bit string of length N^2. Using such a representation of an ANN architecture a standard GA algorithm can be used (see Section 1.2.1).

It is easy to see, that the above representation can describe an ANN of any architecture: the feedforward networks as well as recurrent ones. If a class of considered networks is limited to the MLP then the matrix $\mathbb{V}$ contains many elements equal to 0 and cannot be changed during a searching process. Such a limitation complicates genetic operations and occupies a wide memory in a computer. Thus, the passing over this elements in the representation is sensible (Obuchowicz and Politowicz 1997).

Usually, an ANN has from hundreds to thousands synaptic connections in practical applications, and a binary code representing such an ANN architecture is very long. This fact causes that standard genetic operations are not effective. The convergence of the genetic process deteriorates with increasing the complexity of the ANN architecture. Miller and coworkers (1989) propose genetic representation of the ANN architecture in the form of the connection matrix $\mathbb{V}$. The crossover operator is defined as an exchange of randomly chosen rows or columns between two matrices. In the case of the mutation, each bit is turned with some (very small) probability.

Presented above methods of the genetic representation of the ANN architecture are called *direct encoding* (Kitano 1990). This term informs that each bit represents one synaptic connection in the ANN structure. The disadvantage of these methods is too slow convergence of the genetic process, or lack of convergence in the limit of very large architectures. Furthermore, if the initial architecture $NA_{\max}$ is very complex, the result of such genetic searching process is not so optimal as can be characterized by some compression level. The measure of the method efficiency can be, so called, *the compression index* (Obuchowicz and Politowicz 1997) defined by the form:

$$\kappa = \frac{\eta^\star}{\eta_{max}} \times 100\%, \tag{5.34}$$

where $\eta^\star$ is a number of synaptic connections in the resulted architecture, $\eta_{\max}$ is the maximal number of connections which is acceptable in a given architecture representation.

In order to illustrate this compression ability of the genetic approach with the direct encoding, let us consider the MLP, which implements logical conjunction (AND), inclusive OR and exclusive OR (XOR) of two bits. This problem is described in the work (Obuchowicz and Politowicz 1997). The number of input and output units is defined by the dimension of the input vector $\boldsymbol{u} \in \boldsymbol{R}^2$ and the output vector $\boldsymbol{y} \in \boldsymbol{R}^3$. For simplification the network architecture is limited to one hidden layer (Fig. 5.7). The problem is reduced to the number ν of hidden neurons and the number η of synaptic connections determination. The definition

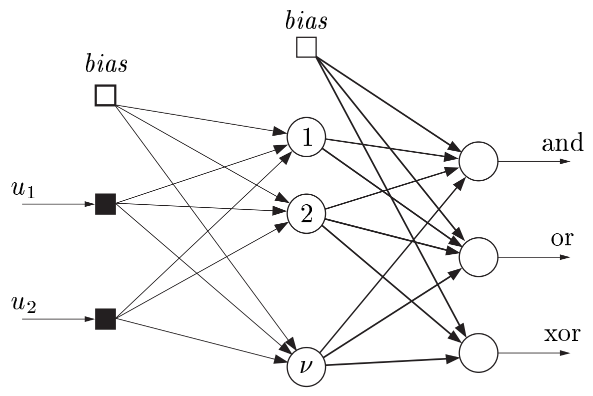


Fig. 5.7. The architecture of the network implementing three logical functions of two bits.

of the quality criterion is defined as follows

$$J_T = \beta\eta(NA) + \gamma t(NA), \tag{5.35}$$

where $t(NA)$ is the discrete learning time, β and γ are weight parameters deciding, which factor, the number of connection $\eta(NA)$ or the learning time $t(NA)$ is more essential.

Two evolutionary algorithms are used. First of them is the classical GA, which works on the binary string of the length l, i.e. an individual belongs to the space $I = \{0,1\}^l$. The maximized fitness function is non-negative:

$$\Phi(NA) = \alpha - J_T(NA), \tag{5.36}$$

where the cost J_T is defined by (5.35). As a stop condition, a maximum number of iteration is set. The selection $s : I^\mu \to I^\mu$ generates two parent elements using the *roulette method.* The one-point crossover occurs with the probability $\theta_r = 0.6$. The mutation sporadically exchange one bit in the string with probability $\theta_m = 0.033$ for each bit. The second algorithm, called Genetic-Evolutionary Search Algorithm (GESA), has been proposed in (Obuchowicz and Politowicz 1997). It differs from the classical GA in three facts. Unlike the GA, μ parent elements are selected from actual population of μ elements by the *roulette method.* The crossover operator is omitted. The probability of mutation θ_k of the k-th element is not constant, but:

$$\theta_k = 0.1\sigma_m \ln\left(\frac{2}{\xi_k} - 1\right), \tag{5.37}$$

where σ_m is an input parameter, and $\xi_k \in [0,1]$ is a uniformly distributed random number. The learning process for each testing architecture is done using the BP algorithm. Figure 5.8 illustrates the compression ability of both algorithms.

The alternative class of genetic representations of an ANN architectures is the *indirect encoding* (Kitano 1990, Koza and Rice 1991, Marshall and Harrison 1991).

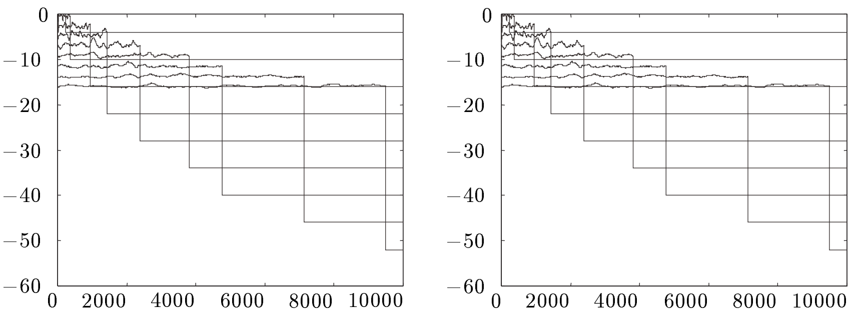


Fig. 5.8 . Fitness of the best individual in time in the case of the GA (a) and GESA (b) for different values of $\nu = 2, 3, \ldots, 10$ ($\nu = 2$ – the top curve, and $\nu = 10$ – the bottom curve). For the efficiency comparison $\alpha = 0$ in the equation (5.36). The high of the curve fault is proportional to the compressing factor κ (5.34)

In the work (Marshall and Harrison 1991) an individual contains binary encoded parameters of an MLP architecture (the number of hidden layers, the number of hidden neurons in each layer, etc.) and parameters of the BP algorithm used for learning this MLP (the learning factor, the momentum factor, the desired accuracy, the maximal number of iterations, etc.). A discrete finite set of values is defined for each parameter, the cardinality of this set depends on the number of bits assigned for a given parameter. In this case the genetic process searches not only for the optimal architecture but for optimal training process, too.

The other proposition (Kitano 1990) is a graph-based encoding. Let the searching space be limited to architectures, which contain 2^{h+1} units at the most. Then, the connection matrix can be represented by a tree of h levels, and each node of this tree possesses four successors of is the leaf. Each leaf is one of the 16 possible matrices 2×2 of binary elements. Four leaves of a given node of the level $h - 1$ define a 4×4 matrix, etc. In this way the root of the tree represents the whole connection matrix. Crossover and mutation operators are defined in the same way as in GP method (Fig. 1.2). Koza and Rice (1991) apply the GP algorithm (see Section 1.2.2) for neural network design.

5.6. Optimization of the DMLP architecture

The significance of network architecture optimization increases when the DMLP is taken into considerations. The number of free network parameters rapidly increases when one substitutes standard McCulloch-Pitt's neurons by the DNM units. Thus, there is some quality difference between architecture allocation of the MLP and the DMLP. Apart from setting an appropriate number of hidden layers and the number of neurons in each of these layers, the dynamic order of each particular neuron has to be established in the DMLP.

5.6.1. Simulated annealing with cascade-reduction technique

5.6.1.1. Simulated annealing algorithm

Simulated Annealing (SA) (Kirkpatrick *et al.* 1992) is based on the observation of the crystal annealing process, which has to reduce crystal defects. The system state is represented by a point S in the space of feasible solutions of a given optimization problem. The neighbouring state S' of the state S differs from S only in one parameter. The minimized objective function E is called the energy by the physical analogy, and the control parameter T is called the temperature. The SA algorithm is following:

1. Choose the initial state $S = S_0$ and the initial temperature $T = T_0$.
2. If the stop condition is satisfied then stop with the solution S else go to 3.
3. If the equilibrium state is achieved, go to 8 else go to 4.
4. Choose randomly a new neighbouring state S' of the state S.
5. Calculate $\Delta E = E(S') - E(S)$.
6. If $\Delta E < 0$ or $\chi < \exp(-\Delta E/T)$, where χ is uniformly distributed random number from the interval $[0, 1)$, then $S = S'$.
7. Go to 3.
8. Update T and go to 2.

The non-negative temperature ($T > 0$) allows to choose the state S', whose energy is higher than the energy of the actual state S, as a base state for the further search, and then, there is a chance to avoid getting stuck in a local optimum. Dislocations , which deteriorate the system energy, are controlled by the temperature T. Their range and occurring frequency decrease with $T \to 0$. As the equilibrium state can be chosen a state,when the energy almost does not change (with a given accuracy, which is a function of temperature) in a given time interval. This criterion is relatively strong and cannot be accomplished. So, usually, a number of iteration is fixed for a given temperature. The initial temperature is the measure of the maximal "thermal" fluctuations in the system. Usually, it is assumed that the chance of achieving any system energy should be high in the beginning of the searching process. The linear decreasing of the temperature is not recommended. The linear *annealing strategy* causes the exponential decrease of "thermal" fluctuations and the searching process usually gets stuck in a local optimum. Two *annealing strategies* are recommended:

$$T(t_n) = \begin{cases} \dfrac{T_0}{1 + \ln t_n} \\ \alpha T(t_{n-1}) \end{cases}, \tag{5.38}$$

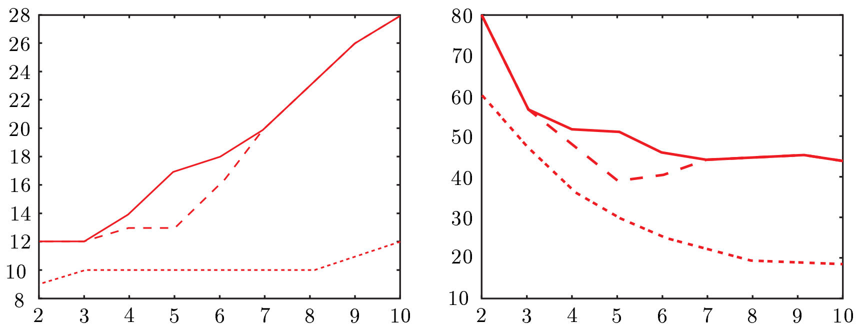


Fig. 5.9. (a) The minimal number of synaptic connections obtained using the GA (solid line), the GESA (dashed line) and the SA (dotted line) as a function of maximal number of hidden units ν (see Fig. 5.7); (b) The relationship between the compression index κ (5.34) and ν.

where t_n is the number of temperature updating, and $\alpha \in [0, 1]$ is a given constant. The annealing strategy determines the stop condition. If the strategy (5.38) is used, the process is stopped when the temperature is almost equal to 0 ($T < \varepsilon$).

Firstly, the simulated annealing algorithm was applied as a recreation process of the Boltzmann machine (Korbicz *et al.* 1994). The SA implementations in the learning of the MLP have not significant successes. But, it is very promising algorithm in the case of searching for the optimal ANN architecture. In the work (Obuchowicz 1998), the SA algorithm was compared with the GA and GESA (see Section 5.5.2). Let the MLP architecture be represented in the same way like in the GA and GESA, i.e. by a bit string, where each bit represents presence (1) or absence (0) of the corresponding synaptic connection. A randomly generated bit string is chosen as an initial solution S. The neighbouring solution S' differs from the S only by one bit. The annealing strategy is conducted in order to the second formulae of (5.38), where $\alpha = 0.9$ and the initial temperature $T_0 = 20$. The algorithm ends if $T < 0.05$. Simulation experiments show that the SA algorithm is more effective than the GA and GESA. Figure 5.9a presents the relationship between the minimal numbers of connections (the global minimum $\eta_{opt} = 9$) obtained by the GA, GESA, and SA, and the maximal permissible number of hidden neurons $\nu = 2, 3, \ldots, 10$ ($15 < \eta_{\max} < 63$). Figure 5.9b presents the relationship between the compression index κ (5.34) and ν. The domination of the SA algorithm is clearly seen.

5.6.1.2. Cascade reduction with simulated annealing

The main idea of cascade reduction is following (Obuchowicz 1999d). We start with a network structure that is supposed to be sufficiently complex and reduce it using a given algorithm. Thus, we obtain a network with $\eta^\star(0)$ parameters from $\eta_{\max}(0)$. In the next step we assume that $\eta_{\max}(1) = \eta^\star(0)$ and apply reduction again. This process is repeated until $\eta^\star(k) = \eta_{\max}(k) (= \eta^\star(k-1))$.

Let us consider the dynamic system described by (5.31), which is modeled by the DMLP network. The set of learning signals consists only one type of the input signal — the white noise. The squared error

$$J_L = \frac{1}{2}\Big\langle \big(y_{DMLP}(k) - y(k)\big)^2 \Big\rangle, \tag{5.39}$$

is chosen as the cost function of the *on-line* learning process, which is proceeded by the EDBP algorithm. The training ends up if $J_L < 0.01$ or the assumed maximum number of iterations ($k_{\max} = 10000$) is achieved. The map (5.33) is chosen as a testing signal. The testing cost J_T is defined by (5.32).

Taking into account results obtained in previous Section (Fig. 5.9) the SA algorithm seems to be the best choice as a reduction process. An architecture of the DMLP network is encoded into a bit string, each bit represents absence or presence of one free parameter of the DMLP network (weight of the synaptic connection, the feedback ar feedforward parameter of the IIR filters, etc.). The parameters of the SA algorithm are chosen as: $\alpha = 0.95$, $T_0 = 0.5$. For simplicity, we assume that all DNM units possess second-order IIR filter. We start with the network architecture $\mathcal{N}^2_{1:10:10:1}$ (one input unit, 10 units in the first hidden layer, 10 units in the second hidden layer, and one output unit — 246 free parameters). The sequence of network structures obtained after each SA process is as follows:

$$\mathcal{N}^2_{1:10:10:1}(246) \to \mathcal{N}^2_{1:5:3:1}(77) \to \mathcal{N}^1_{1:5:1}(46) \to \mathcal{N}^1_{1:1:1}(14). \tag{5.40}$$

The number of free parameters is described in brackets. Figure 5.10 presents the responses of the dynamic system (5.31) and the resulting DMLP on the testing signal (5.33). Unfortunately, the above DMLP architecture has been obtained only a couple of times for dozens experiments. This fact proves that the EDBP algorithm used to the network training usually finds unsatisfactory local minima in the space of DMLP parameters.

5.6.2. Cascade network of dynamic neurons

The basic idea of the cascade-correlation algorithm is to reduce iteratively the output error by inserting hidden units that correlate (or anti-correlate) well with the error. By freezing the network while optimizing the new hidden unit candidate the algorithm avoids the moving targets problem of the standard BP algorithm (Fahlman and Lebierre 1990). Below, the cascade-correlation algorithm adapted to the dynamic network and used in this work is described.

This algorithm starts without any hidden units. The direct input-output connections are trained on-line using the gradient descent method. At this stage, the EDBP algorithm reduced to the version of one DNM unit learning can be applied. If the network performance is satisfactory, the procedure is stopped, otherwise it attempts to reduce the residual errors further by adding a new hidden DNM unit to the network. The unit creation process begins with a candidate unit that receives trainable input connections from all of the networks external inputs and from all pre-existing hidden units. The output of this candidate unit is not yet

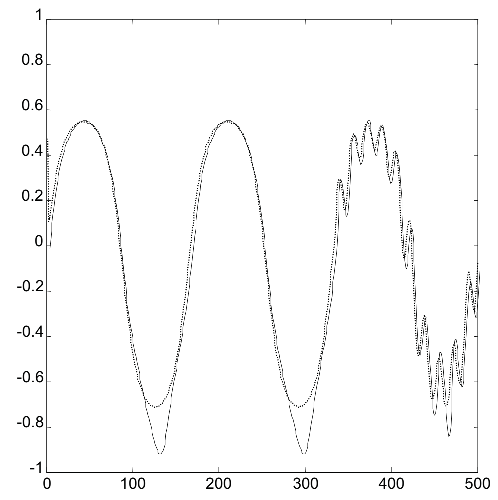


Fig. 5.10 . Responses of the dynamic system (5.31) (solid line) and the DMLP of $\mathcal{N}^1_{1:1:1}$ structure (dotted line) on the testing signal (5.33) ($J_T = 0.011$ (5.32))

connected to the active network. The adjusting of the candidate unit input weights and its IIR parameters is performed to maximize the following performance index:

$$\Psi = \sum_{i=1}^{s_M} \left| \sum_{p=1}^{P} (V_p - \langle V \rangle)(E_{pi} - \langle E_i \rangle) \right| \tag{5.41}$$

where s_M is the number of output units, P is the number of training patterns, $\langle V \rangle = (\frac{1}{P}) \sum_{p=1}^{P} V_p$ and $\langle E_i \rangle = (\frac{1}{P}) \sum_{p=1}^{P} E_{pi}$, V_p denotes the response of the candidate on the input u_p and E_{pi} is the output error on the input u_p.

When a new DNM unit is added to the network, its adjusted parameters are frozen, and all the output neurons parameters are trained again using the gradient descent method. This cycle repeats until the output network error is acceptable. In Fig. 5.11 an example of the neural network with two inputs and two outputs is shown. It is called the Cascade Network of Dynamic Neurons (CNDN) (Patan *et al.* 1999). Black dots denote adaptable weights between neurons. This is a feed-forward series-parallel structure. Each neuron receives signals from all inputs and all hidden neurons. Such a structure has some advantages in contradistinction to the standard feed-forward networks. The first advantage is preventing moving target problem. This problem often occurs in the standard feed-forward networks, where each neuron adapts its parameters in a constantly changing environment receiving only the input and output network data of small sizes. In fact, instead of a quick adjustment of its parameters, the hidden neurons engage in a complex dance

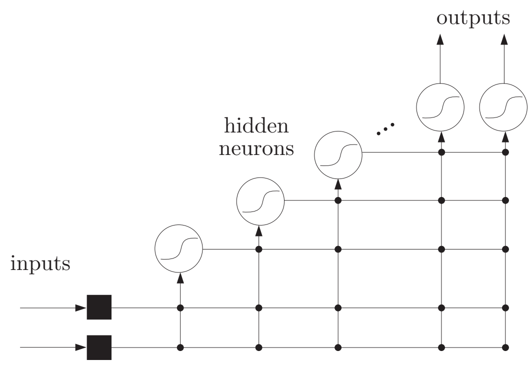


Fig. 5.11 . Examples of cascade network with two inputs and two outputs.

around the constantly moving target. The more hidden neurons there are, the harder it is to achieve a good learning quality. In a cascade network, each hidden neuron is trained separately. Thus, it receives all input and output learning data and that is why it can adjust its parameters in a correct way. The other advantage is an optimal character of this structure. The hidden neurons are added to the network one by one until the output network error is acceptable. In this way an optimal neural network, in the sense of modelling quality, can be designed. It is necessary to note that the proposed network is not optimal in the sense of the number of hidden neurons or number of parameters either.

In order to illustrate the effectiveness of the neural model based on the CNDN, let us consider the *Two-Tank System*, which consists of two cylindrical tanks with identical cross sections being filled with water and with a delay spiral pipeline (see Fig. 5.12). The nominal outflow Q_n is located at Tank 2. The pump driven by a DC motor supplies Tank 1, where Q_1 is the inflow of the liquid through pump to Tank 1. Both the tanks are equipped with sensors for measuring the level of the liquid (h_1, h_2). Valves V_1, V_2, V_3, V_4 and V_E are electronic switching ones. The aim of the two-tank system control is to keep up the water level in Tank 2 constant.

The high modelling quality has been obtained for relatively small CNDN architecture N_{3--1} (the CNDN consists of 3 hidden DNM units and one output DNM unit). Figure 5.13 compares the measured and model liquid levels in Tank 2. Basing on the CNDN models for a set of possible faults in the two-tank system, the effective fault diagnosis system has been proposed (Korbicz *et al.* 1999, Korbicz *et al.* 2001).

5.6.3. Graph of the DMLP structures

The optimum DMLP architecture searching process can be more effective if the space of the DMLP architectures will be ordered. Doering and coworkers (1997) propose the ordering of the MLP architectures in the infinite graph (Fig. 5.14).

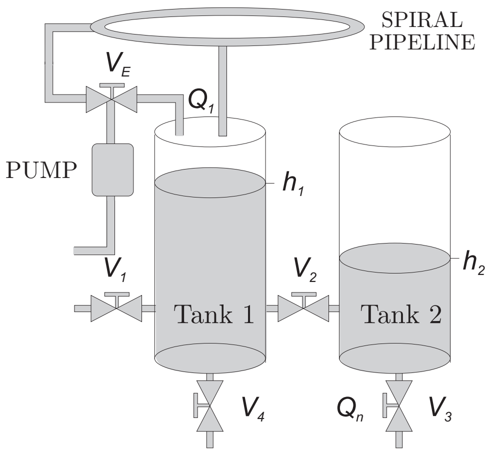


Fig. 5.12 . Two-tank system with a delay spiral pipeline.

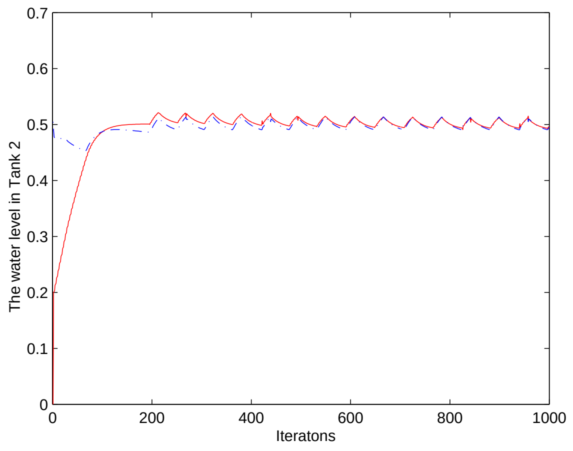


Fig. 5.13 . The measured (solid line) and modelled by the CNDN (dotted line) liquid levels in Tank 2

The notation $\upsilon 1 - \upsilon 2$ used in Fig. 5.14 denotes that the network consists of $\upsilon 1$ neurons in the 1st hidden layer and $\upsilon 2$ neurons in the 2nd hidden layer. The number of units in the input and output layers are defined by the dimensions of the input and output spaces, respectively.

In the case of the space of the DMLP architectures the corresponding graph $G(\mathcal{A})$ is much complicated because of IIR filters existence in the DNM units (Obuchowicz 1999c). The graph $G(\mathcal{A})$ can be described by definition of the expansion operator $\Gamma(NA)$, which generates all successor of a given architecture NA.

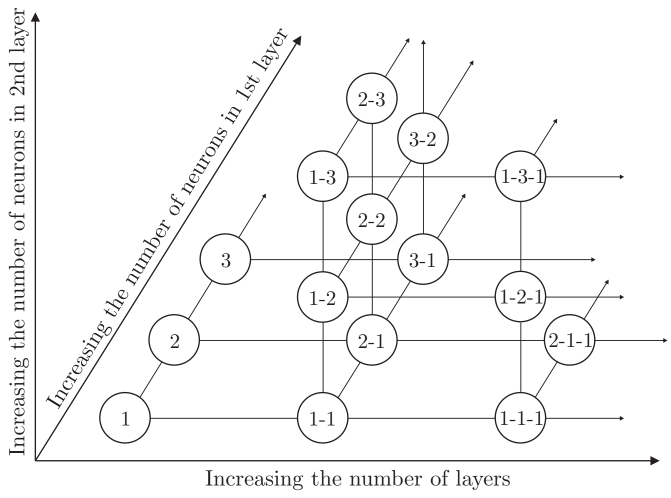

Increasing the number of neurons in 2nd layer
Increasing the number of neurons in 1st layer
Increasing the number of layers
1
2
3
1-1
1-2
1-3
2-1
2-2
2-3
3-1
3-2
1-1-1
1-2-1
1-3-1
2-1-1

Fig. 5.14 . The graph of the MLP structures.

The expansion operator $\Gamma(NA)$ creates the following successors.

1. *Varying the number of hidden layers.* Assume the DMLP architecture NA (5.7). The architecture $NA^{(1)}$ with inserted hidden layer with one DNM unit v' of zero order

$$\begin{aligned}
NA^{(1)} &= \big(\{V_m^{(1)} \mid m = 0,1,\ldots,M+1\}, \\
&\quad \{o_s^{(1)m} \mid m = 1,2,\ldots,M+1; s = 1,2,\ldots,s_m\}, \mathcal{E}^{(1)}\big); \\
V_m^{(1)} &= V_m, \qquad \text{for} \quad m = 0,1,\ldots,M-1; \\
V_{M+1}^{(1)} &= V_M; \\
V_M^{(1)} &= \{v'\}; \\
o_s^{(1)m} &= o_s^m, \qquad \text{for} \quad m = 0,1,\ldots,M-1, \quad s = 1,2,\ldots,s_m; \\
o_1^{(1)M} &= 0; \\
o_s^{(1)M+1} &= o_s^M, \qquad \text{for} \quad s = 1,2,\ldots,s_{M+1}(=s_M); \\
\mathcal{E}^{(1)} &= \mathcal{E} \cup \big\{(v,v') \big| v \in V_{M-1}^{(1)}\big\} \\
&\quad \cup \big\{(v',v) \big| v \in V_{M+1}^{(1)}\big\} \setminus \big\{(v_-,v_+) \big| v_- \in V_{M-1}^{(1)}, v_+ \in V_{M+1}^{(1)}\big\}
\end{aligned} \tag{5.42}$$

is the successor of NA.

2. *Varying the number of units in a hidden layer.* Assume the architecture NA (5.7) that has at least one hidden layer ($M \geq 2$). Then, all architectures $NA^{(2)}$ with an inserted unit v'' in a i-th layer for $i = 1, 2, \ldots, M-1$

$$\begin{aligned}
NA^{(2)} &= \big(\{V_m^{(2)} \mid m = 0, 1, \ldots, M\}, \\
&\qquad \{o_s^{(2)m} \mid m = 1, 2, \ldots, M; s = 1, 2, \ldots, s_m\}, \mathcal{E}^{(2)}\big); \\
V_m^{(2)} &= V_m, \qquad \text{for} \quad m \neq i; \\
V_i^{(2)} &= V_i \cup \{v''\}; \\
o_s^{(2)m} &= o_s^m, \qquad \text{for} \quad m \neq i, \quad s = 1, 2, \ldots, s_m; \\
o_{s_i+1}^{(2)i} &= 0; \\
\mathcal{E}^{(2)} &= \mathcal{E} \cup \big\{(v, v'') \big| v \in V_{i-1}^{(2)}\big\} \cup \big\{(v'', v) \big| v \in V_{i+1}^{(2)}\big\},
\end{aligned} \tag{5.43}$$

are successors of NA.

3. *Varying the IIR order of the DNM unit.* Assume the architecture NA (5.7). Then, all architectures $NA^{(3)}$ with increased order of the IIR filter in the j-th DNM unit of the i-th layer $i = 1, 2, \ldots, M$, $j = 1, 2, \ldots, s_i$

$$\begin{aligned}
NA^{(3)} &= \big(\{V_m \mid m = 0, 1, \ldots, M\}, \\
&\qquad \{o_s^{(3)m} \mid m = 1, 2, \ldots, M; s = 1, 2, \ldots, s_m\}, \mathcal{E}\big); \\
o_s^{(3)m} &= o_s^m, \qquad \text{for} \quad m \neq i, \quad s = 1, 2, \ldots, s_m; \\
o_s^{(3)i} &= o_s^i, \quad s \neq j; \\
o_j^{(3)i} &= o_j^i + 1
\end{aligned} \tag{5.44}$$

are successors of NA.

Thus, $\Gamma(NA)$ maps an architecture NA with $M-1$ hidden layers and N processing DNM units onto $M+N$ successors.

The searching process on the $G(\mathcal{A})$ can be carried out by many ways. The SA algorithm, the $A^\star$ algorithm, and ***tabu search*** seem to be the most interesting approaches.

5.6.4. Simulated annealing approach

Taking into account the encouraging results of the simulated annealing in the cascade-reduction method, it is first proposed method for searching the optimal DMLP architecture $NA^\star$ in the graph $G(\mathcal{A})$ (Obuchowicz and Patan 1998). Starting with the random node S, the graph is searched by the SA algorithm (see

Section 5.6.1). A node, which is directly connected with an actual base node S, is treated as a neighbouring architecture S'.

Similarly as in Section 5.6.1, the dynamic system described by (5.31) is considered. The set of learning signals consists of only one type of the input signal: the white noise. The squared error (5.39) is chosen as the cost function of the *on-line* learning process, which is proceeded by the EDBP algorithm, for which 10000 learning patters have been generated. The training ends up if $J_L < 0.01$ or the assumed maximum number of iterations ($k_{\max} = 10000$) is achieved. As a testing signal the map (5.33) is chosen. The testing cost J_T defined by (5.32) is treated as a energy function for the SA (Section 5.6.1).

The optimization process has been set in motion a few dozen times for different initial DMLP architectures, different learning parameters, and different annealing strategies. Unfortunately, obtained results are not reproducible. This problem results from the fact, that the EDBP algorithm for a given network architecture and a given set of learning parameters does not give the same results, but gets stuck in different for each run local optimum of the square error function. The most often obtained resulting architectures are $\mathcal{N}^2_{1:4:1}$ or $\mathcal{N}^2_{1:5:1}$ ($J_T \cong 0.012 \div 0.019$). However, it happens that the structures $\mathcal{N}^2_{1:10:1}$ or $\mathcal{N}^2_{1:4:3:1}$ ($J_T \cong 0.025 \div 0.035$) have been treated as "optimal".

5.6.5. $A^\star$ and Tabu Search approaches do DMLP architecture optimization

5.6.5.1. $A^\star$ algorithm

The $A^\star$ algorithm, first described in (Hart *et al.* 1968, Nilsson 1980), is a way to implement best-first search to a problem graph. The algorithm will operate by searching a directed graph in which each node n_i represents a point in the problem space. Each node will contain, in addition to a description of the problem state it represents, an indication of how promising it is, a parent link that points back to the best node from which it came, a list of the nodes that were generated from it. The parent link will make it possible to recover the path to the goal once the goal is found. The list of successors will make it possible, if a better path is found to an already existing node, to propagate the improvement down to its successors.

A heuristic function $f(n_i)$ is needed that estimates the merits of each generated node. In the $A^\star$ algorithm this cost function is defined as a sum of two components:

$$f(n_i) = g(n_i) + h(n_i), \tag{5.45}$$

where $g(n_i)$ is the cost of the best path from the start node n_0 to the node n_i and it is known exactly to be the sum of the cost of each of the rules that were applied along the best path from n_0 to n_i, and $h(n_i)$ is the estimation of the addition cost getting from the node n_i to the nearest goal node. The function $h(n_i)$ contains the knowledge about the problem.

The outline of the $A^\star$ algorithm is described in many handbooks from the domain of Artificial Intelligent. In this work the algorithm included in (Rich 1983) is implemented.

5.6.5.2. Tabu Search

The tabu search metaheuristic has been proposed by Glover (1986). This algorithms models processes existing in the human memory. This memory is implemented as a simple list of solutions explored recently. The algorithm starts from a given solution x_0, which is treated as actually the best solution $x^* \leftarrow x_0$. The tabu list is empty $T := \emptyset$. Next, the set of neighbouring solutions are generated excluding solutions noted in the tabu list, and the best solution of this set is chosen and is chosen as a new base point. If x' is better than x^* then $x^* \leftarrow x'$. The actual base point x' is added to the tabu list. This process is iteratively repeated until a given criterion is satisfied.

There are many implementations of the Tabu Search idea, which differ between each other in the method of the tabu list managing, e.g. Tabu Navigation Method (TNM), Cancellation Sequence Method (CSM), Reverse Elimination Method (REM). Particular description of these methods can be found in (Glover and Laguna 1997).

5.6.5.3. Implementations

In order to apply the $A^\star$ and Tabu Search algorithms to an architecture optimization of the DMLP we have to define (Obuchowicz and Patan 2003):

- the optimization criterion — which is chosen in the form

$$J_T\big(\boldsymbol{y}_{NA,\boldsymbol{v}^\star}(k), \boldsymbol{y}(k) \mid k \in K\big) = \frac{\sum_{k\in K}\big(\boldsymbol{y}_{NA,\boldsymbol{v}^\star}(k) - \boldsymbol{y}(k)\big)^2}{\sum_{k\in K}\boldsymbol{y}^2(k)}, \quad (5.46)$$

  where $\boldsymbol{y}_{NA,\boldsymbol{v}^\star}(k)$ and $\boldsymbol{y}(k)$ are the output of the learned DMLP and desired output, respectively.

- an expansion operator $\Gamma(NA) : \mathcal{A} \to 2^{\mathcal{A}}$ defined be equations (5.42)–(5.44), which maps any network architecture $NA \in \mathcal{A}$ onto a set of successors.

Moreover, the following functions have to be defined for the $A^\star$ algorithm

- the cost function $g(NA, NA')$ assigned to each expansion operation:

$$g(NA, NA') = \begin{bmatrix} \gamma(NA') - \gamma(NA) \\ \delta(NA') - \delta(NA) \end{bmatrix}, \quad (5.47)$$

  where $\gamma(NA)$ is the number of free parameters in the DMLP architecture NA, and $\delta(NA)$ is the number of hidden layers in NA;

- the heuristic function $h(NA)$

$$h(NA) = \begin{bmatrix} \dfrac{J_T\big(\boldsymbol{y}_{NA,\boldsymbol{v}^\star}(k), \boldsymbol{y}(k) \mid k \in K\big)}{J_T\big(\boldsymbol{y}_{NA_0,\boldsymbol{v}^\star}(k), \boldsymbol{y}(k) \mid k \in K\big)} \\ 0 \end{bmatrix}, \quad (5.48)$$

  where NA_0 denotes the initial architecture of searching.

Tab.5.2. Specification of the selected neural networks

CHARACTERISTICS	**Method**			
	Tabu Search – list length			$A^\star$
	3	5	10	
Network structure	$N^2_{1,1,1}$	$N^2_{1,4,1}$	$N^2_{1,4,1}$	$N^2_{1,5,1}$
1st layer filters orders	(2)	(2 1 1 0)	(2 1 1 0)	(2 1 0 2 1)
2nd layer filters orders	(0)	(1)	(1)	(2)
Modelling quality	0.145051	0.139015	0.139015	0.123431

Because both $g(NA, NA')$ and $h(NA)$ are vector functions, the relation $\boldsymbol{p}\dot{\leq}\boldsymbol{q}$ must be defined

$$\boldsymbol{p}\dot{\leq}\boldsymbol{q} \Leftrightarrow (p_1 \leq q_1)\mathrm{or}\big((p_1 = q_1)\mathrm{and}(p_2 \leq q_2)\big). \tag{5.49}$$

5.6.6. Experimental comparison of the $A^\star$ algorithm and Tabu Search

This section presents the experimental results achieved during selection of the optimal neural network structure using searching methods described in the previous sections (Obuchowicz and Patan 2003). The neural network composed of dynamic neuron models is used here to identify the dynamic non-linear process represented by the following difference equation:

$$y(k+1) = \frac{y(k)}{1+y(k)^2} + u(k)^3, \tag{5.50}$$

where $u(k)$ and $y(k)$ are the input and output of the process at the instant k, respectively. The learning process is carried out off-line for 500 steps using the Extended Dynamic Back-Propagation algorithm and a pseudo-random input uniformly distributed in the interval $[-2, 2]$. The learning set consists of 200 patterns and the learning rate is equal to 0.01. The training procedure of each examined network structure is repeated four times in order to decrease a chance to get stuck in local minima of an error function. Furthermore, each neuron in the network has the hyperbolic tangent activation function.

The selection of the optimal neural network structure is performed using two searching methods: the $A^\star$ algorithm and the Tabu Search method. The second algorithm is tested with different number of structures memorized, in turn 3, 5 and 10. Results achieved during experiments are presented in Table 5.2, where the notation $N^n_{r,v,s}$ denotes n-th layer neural network with r inputs, v hidden neurons and s outputs, and $(or1\ or2\ \ldots\ orn)$ denotes that 1st neuron possesses $or1$ order filter, 2nd one – $or2$ order filter and n-th neuron – orn filter order.

Both searching methods start with the minimal network structure consisting of the output neurons only. After that, the neural network is growing up. At each algorithm step one parameter can be changed: the number of hidden layers (maximum 2 hidden layers) or the filter order (maximum 2nd order) or the number of neurons in hidden layers. The Tabu Search algorithm makes is also possible to reduce the network size. As one can see in Table 5.2, the best results have been obtained using $A^\star$ algorithm. The optimal network structure selected with this method consists of two processing layers and five hidden neurons. It is worth noting here that this method has been run on three computers and results obtained in the each case are the same (the same network structure, quality and optimal path). One can conclude that the algorithm generates credible results. The optimal path generated with the $A^\star$ algorithm is presented in Table 5.3. In order to find the optimal neural network, 558 structures have been tested. Each next network has a bigger size than the previous one.

Tab.5.3. Optimal path generated with the $A_\star$ algorithm

Network No.	Network structure	Filters orders 1st layer (hidden)	Filters orders 2nd layer (output)	Modelling quality
0	$N^1_{1,1}$	–	(0)	0.267325
2	$N^1_{1,1}$	–	(1)	0.171869
3	$N^2_{1,1,1}$	(0)	(1)	0.144926
9	$N^2_{1,1,1}$	(1)	(1)	0.249667
12	$N^2_{1,1,1}$	(1)	(2)	0.239527
20	$N^2_{1,2,1}$	(1 0)	(2)	0.151304
28	$N^2_{1,3,1}$	(1 0 0)	(2)	0.152115
31	$N^2_{1,4,1}$	(1 0 0 0)	(2)	0.161490
118	$N^2_{1,4,1}$	(1 1 0 0)	(2)	0.168057
322	$N^2_{1,4,1}$	(1 1 0 1)	(2)	0.168039
426	$N^2_{1,5,1}$	(1 1 0 1 0)	(2)	0.208492
540	$N^2_{1,5,1}$	(1 1 0 1 1)	(2)	0.210782
543	$N^2_{1,5,1}$	(2 1 0 1 1)	(2)	0.220161
558	$N^2_{1,5,1}$	(2 1 0 2 1)	(2)	0.123431

In the case of the Tabu Search method the results are also interesting. When a short tabu list was used (length of 3), the algorithm demonstrated the periodic behaviour. After every 23 structures it generates the same optimal neural network of the $N^2_{1,1,1}$ class. To avoid such a periodic behaviour, the longer tabu lists have been applied (length of 5 and 10). Table 5.2 clearly shows that using longer tabu lists, better results can be obtained. Moreover, in both cases the identical optimal path has been achieved. The conclusion is, that further increasing size of the tabu list does not yield better results. In Table 5.4 one can see the optimal path generated with the Tabu Search method (list length equal to 3). First, the network

Tab.5.4. The optimal path generated with the Tabu Search (list length – 3)

Network No.	Network structure	Filters orders		Modelling quality
		1st layer (hidden)	2nd layer (output)	
0	$N^1_{1,1}$	–	(0)	0.267325
2	$N^1_{1,1}$	–	(1)	0.171863
3	$N^2_{1,1,1}$	(0)	(1)	0.163588
9	$N^2_{1,1,1}$	(1)	(1)	0.163725
28	$N^2_{1,1,1}$	(1)	(0)	0.147749
33	$N^2_{1,1,1}$	(2)	(0)	0.145051

is growing up, and for the network No. 28 the algorithm reduces the network size. This phenomenon is very attractive and can cause that Tabu Search may be more flexible method than the $A^\star$ algorithm. In turn, Table 5.5 shows the optimal path generated with the Tabu Search algorithm using the tabu list length of 10.

5.7. Summary

The problem of the neural model design has been considered in this chapter. Two optimization tasks are distinguished in order to solve the main problem: the learning process and the allocation of the optimal architecture of the ANN.

Locally recurrent neural networks, called the DMLP, are considered. This network is composed of the DNM units, which contain an addition module between the adder and activation modules – the IIR filter. Therefore, basing on the DNM units, one can build a dynamic neural network of the multilayer feedforward architecture.

It has been shown that evolutionary algorithms are a very effective tool for neural models learning both in the MLP and DMLP cases. Especially, results of algorithms from the ESSS family seem to be very promising. Unfortunately, the evolutionary approach to the ANN learning possesses some critical defect. This is extremely long time of searching the solution. A hybrid method, which combines EAs with a method of local optimization, like the EDBP, can be more efficient.

The neural model architecture optimization belongs to the class of discrete optimization problems. However, there are many proposals of genetic approaches to this tasks, experiments provide the conclusion that the heuristic search methods, like the simulated annealing, $A^\star$ algorithm and tabu search, effectively compete with GA implementations. Especially in the case of the DMLP network, where space of the of the network architectures is represented by a digraph, the results obtained for the GA algorithm are so pure in comparison with other algorithms that they are not presented in this chapter.

Tab.5.5. The optimal path generated with the Tabu Search (list length – 10)

Network No.	Network structure	Filters orders		Modelling quality
		1st layer (hidden)	2nd layer (output)	
0	$N^1_{1,1}$	–	(0)	0.267325
2	$N^1_{1,1}$	–	(1)	0.171863
3	$N^2_{1,1,1}$	(0)	(1)	0.163588
9	$N^2_{1,1,1}$	(1)	(1)	0.163725
38	$N^2_{1,1,1}$	(2)	(1)	0.152251
42	$N^2_{1,2,1}$	(2 0)	(1)	0.167976
49	$N^2_{1,2,1}$	(2 0)	(2)	0.167494
54	$N^2_{1,2,1}$	(2 1)	(2)	0.151649
61	$N^2_{1,2,1}$	(2 1)	(1)	0.151706
64	$N^2_{1,3,1}$	(2 1 0)	(1)	0.159347
74	$N^2_{1,3,1}$	(2 1 0)	(0)	0.159677
80	$N^2_{1,3,1}$	(2 0 0)	(0)	0.166961
84	$N^2_{1,3,1}$	(2 0 1)	(0)	0.150898
88	$N^2_{1,3,1}$	(2 0 1)	(1)	0.156903
93	$N^2_{1,3,1}$	(2 1 1)	(1)	0.161591
97	$N^2_{1,4,1}$	(2 1 1 0)	(1)	0.139015

The effectiveness of an algorithm of an ANN architecture optimization strongly depends on the effectiveness of a learning algorithm, which provides a quality information of an ANN of a given architecture. in the case of the DMLP networks, results of experiments presented in this chapter are obtained basing on the EDBP algorithm as a learning method. This is a drawback of the presented implementations, because of a low efficacy of the EDBP. Unfortunately, the application of an evolutionary learning algorithm instead of the EDBP causes the sudden increase of time complexity of considered architecture optimization methods.

Chapter 6

GENETIC PROGRAMMING APPROACH TO THE FDI SYSTEM DESIGN

There is an increasing demand for modern technological processes to become safer and more reliable. These requirements extend beyond normally accepted safety-critical systems of nuclear reactors, chemical plants or aircraft to new system such as autonomous vehicles or fast rail systems. The early detection of faults can help avoid systems shut-down, breakdown and even catastrophes involving human fatalities and material damage. Therefore, it is clear that the problem of *fault diagnosis* constitutes an important subject (Korbicz *et al.* 2002).

During the last two decades many investigations were carried out using analytical approaches, based on quantitative models. The idea is to generate signals, termed *residuals*, that reflect inconsistencies between nominal and faulty system operation. Such signals are usually generated using analytical approaches (Chen and Patton 1999, Patton *et al.* 2000). Requirements for precise and accurate analytical model imply that any resulting modelling error will affect the performance of the resulting Fault Detection and Isolation (FDI) scheme (Frank 1998, Frank and Köppen-Seliger 1997). This is particularly true for dynamically non-linear and uncertain systems, which represent the majority of real processes. Therefore, a number of researchers have seen artificial intelligence methods, like artificial neural networks (Himmelblau 1992, Korbicz *et al.* 1999, Korbicz *et al.* 2001, Patton *et al.* 1994, Sorsa and Koivo 1992), fuzzy logic or neuro-fuzzy systems (Calado *et al.* 2001, Kościelny *et al.* 1999a, Kościelny *et al.* 1999b, Pieczyński 1999), expert systems (Cholewa 2002, Fathi *et al.* 1992, Pieczyński 1999) as an alternative way to represent knowledge about faults.

However, there are many techniques of non-analytical models construction, all of them, sooner or later, reduce to a set of optimization problems, e.g. their structure optimization or parameter allocation. These problems are usually non-linear, multimodal, sometimes multi-criteria. And standard local optimization methods are insufficient. Evolutionary algorithms, especially, seem to be an attractive tool to solve these problems (Obuchowicz and Korbicz 2002).

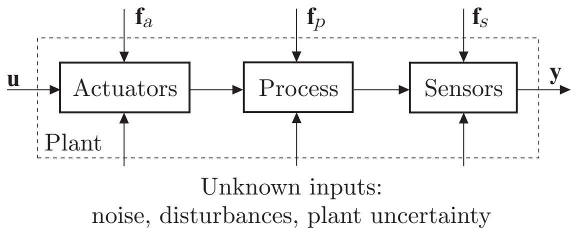


Fig. 6.1 . Automatic control system.

6.1. Basic concepts of fault diagnosis systems

A *fault* can generally be defined as an unexpected change in a system of interest, e.g a sensor malfunction. All the unexpected variations that tend to degrade the overall performance of a system can also be interpreted as faults. Contrary to the term *failure*, which suggests complete breakdown of the system, the term *fault* is used to denote a malfunction rather than a catastrophe.

Since a system can be split into three parts (Frank and Köppen-Seliger 1997) (Fig. 6.1): actuators, process components, and sensors, such a decomposition leads directly to three classes of faults. Actuators faults can be viewed as any malfunction of the equipment that actuate the system, e.g. a malfunction of an electro-mechanical actuator for a diesel engine (Blanke *et al.* 1994). Component faults can be interpreted as the case when some changes in the system make the dynamic relation invalid, e.g. a leak in a tank in the two tank system. Sensors faults can be viewed as serious measurements variations. The faults can commonly be described as inputs. In addition, there is always a modelling uncertainty due to unmodelled disturbances, noise and model mismatch. This may not be critical to the process behaviour, but may obscure the fault detection by rising false alarms.

The automatic fault detection and isolation can be viewed as a sequential process involving the symptom extraction and, basing on actual symptoms and/or additional knowledge, the decision making about a fault occurrence (detection) and its type, range and location (isolation) (Fig. 6.2). There are many fault diagnosis methods. The choice of the method for a given diagnosis problem depends on its type. Generally, two classes of the fault diagnosis systems can be distinguished (Fig. 6.3). The first is based on the pattern recognition principle (Fig. 6.3a). These methods are efficiently applied in the case of static diagnosed systems. Measured signals are initially processed in the symptom extraction step using, e.g., the time windows technique (Kowal and Korbicz 2000) or neural networks (Marciniak and Korbicz 1999).

The second class is model-based FDI systems (Fig. 6.3b), where the quality of the system strongly depends on the accuracy of its model. The residual generator has to form a suitable signal (residual signal) basing on outputs of the system and its model obtained for the same input signals. The appearance of any fault should affect on the residual signal value. Basing on the residual signals, the system state

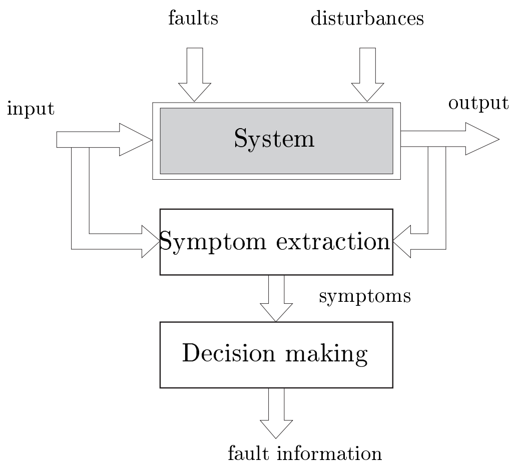


Fig. 6.2 . Schema of the FDI system

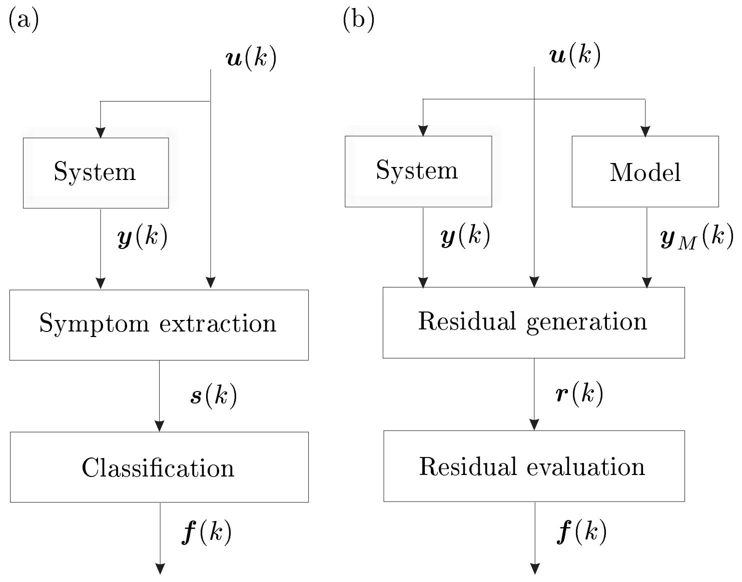


Fig. 6.3 . Schema of the FDI based on pattern recognition techniques (a) and the model-based FDI system (b)

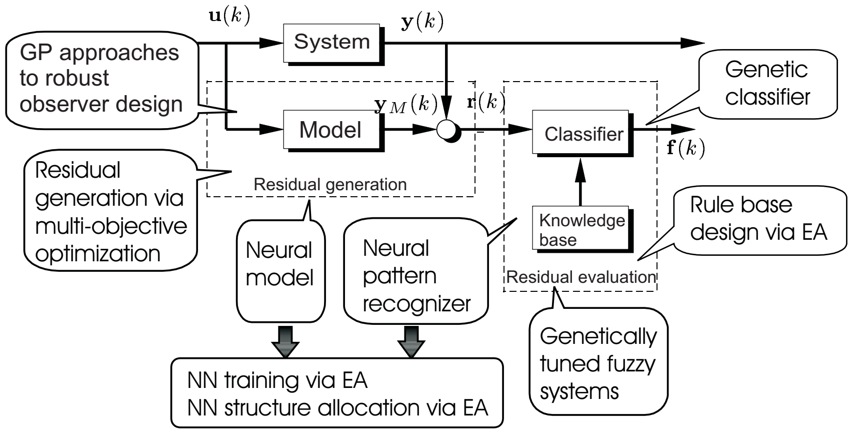


Fig. 6.4 . EAs in the FDI system design

is described in the residual evaluation module. In order to improve the isolation ability of the model-based FDI system, the bank of the models are used, where each model is sensitive to a different fault and of them represents the system in the nominal conditions.

If residual signals are properly generated, the fault detection becomes a relatively easy task. Since without fault detection it is impossible to perform fault isolation, all efforts regarding an improvement of residual generation seem to be justified. This is the main reason why the research effort of this work is oriented towards fault detection and especially towards residual generation.

6.2. EA in the FDI system design

There are relatively few publications of the EA applications to the FDI systems design. Proposed solutions (Chen and Patton 1999, Chen *et al.* 1996, Korbicz *et al.* 1998, Obuchowicz 1999a, Obuchowicz and Korbicz 2002, Witczak *et al.* 1999, Witczak *et al.* 2002) (Fig. 6.4) show the high efficiency of diagnosis systems which design has been aided by EAs.

Optimal residual generation via genetic algorithm was firstly proposed by Chen and coworkers (1996). The studied residual generator is based on full-order observer. The residual response is affected by faults, disturbances, sensor and input noises, and discrimination between them is very difficult. In order to make the residual become insensitive to modelling uncertainty and sensitive to sensor faults a number of performance indices, which are functions of gain and weighting matrices, are defined. The maximization of the first index becomes the residual generator the most sensitive of the faults in the required frequency range. Next indices describe the influence of the sensor noise effect, the disturbance and the initial condition effects and the input noise effect, respectively, on the residual signal and have to be minimized. Some indices are defined in the frequency domain to account for the fact that modelling uncertainty effects and faults occupy different frequency bands. The solution of the simultaneously provided optimizations procedures are obtained using the method of inequalities (see (Chen and Patton 1999)). Such a multi-objective optimization task cannot be solved by the conventional optimization techniques. The genetic algorithm has been successfully applied.

Among artificial intelligence methods applied to design fault diagnosis systems artificial neural networks are very popular, which are used for building of neural models as well as neural classifiers (Frank and Köppen-Seliger 1997, Köppen-Seliger and Frank 1999, Korbicz *et al.* 1998). But, the construction of the neural model is corresponded to two basic optimization problems: optimization of a neural network architecture and its training process, i.e. searching the optimal set of network free parameters. Evolutionary algorithms are a very useful tool to solve both problems, especially in the case of dynamic neural networks (Korbicz *et al.* 1998, Obuchowicz 1999a, Obuchowicz 2000a). Neural networks approaches to the FDI systems building and EAs approaches to the ANN construction are themes of the previous chapters in this book.

The main objective of residual evaluation is to decide whether and where a fault occurred with possible avoidance of wrong decisions causing false alarms. In this case, many techniques can be applied (Frank and Köppen-Seliger 1997), which can be further improved by using the global optimization via evolutionary algorithms, however, below shortly described opportunities of the EA approaches to the symptom evaluation process are only the author's proposals and they have not be implemented, yet.

Efficiency of fault detection systems in the case of multi-dimensional symptom vectors may be improved by pre-processing which leads to the partitioning of the symptom domain into subdomains (clusters). Among many well-known preprocessing methods, EAs characterize high clustering performance. Let us concern with multi-dimensional real data that form a set of the so-called training pairs

$$\mathcal{T}_d = \{\boldsymbol{p}_q = (\boldsymbol{x}^q, y^q) \in \boldsymbol{R} \mid q = 1, ..., p\}. \tag{6.1}$$

The goal is to perform an evolutionary cluster analysis of data in $\mathcal{T}_d$ to get at the end a partitioning of $\mathcal{T}_d$. The number of clusters is not known in advance. To evaluate each off-spring cluster in the population, different local fitness functions may be used. They could be the maximal distance of the training pair of the cluster from the cluster centroid, or a mean variation of all training pairs in the cluster. Based on the local fitness function of the cluster one can build a global fitness function (Kosinski *et al.* 1998)

A fuzzy inference system is often used as universal approximator for a problem of multi-dimensional data or as controller for some industrial applications (Köppen-Seliger and Frank 1999). A fuzzy modelling approach consists of two kinds of problem, configuring fuzzy rules and optimization of the shapes of membership functions, which are considered to be combinatorial and numerical optimization problems, respectively. The EA is able to be applied to both these problems. In many research works (cf. (Carse *et al.* 1996)), however, the EA is applied only to optimize the configuration of the fuzzy rules, while another opti-

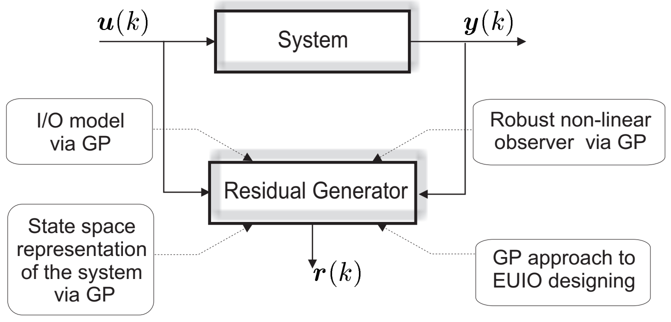

$u(k)$
System
$y(k)$
I/O model via GP
Robust non-linear observer via GP
Residual Generator
State space representation of the system via GP
$r(k)$
GP approach to EUIO designing

Fig. 6.5 . The GP implementation considered in this chapter

mization algorithm, such as steepest descent method, is applied to optimize the shapes of the membership functions.

The application of the artificial intelligence techniques leads to the concept of the fault diagnosis expert system where analytical and heuristic information as well as knowledge processing are combined (Frank and Köppen-Seliger 1997). The expert system for fault diagnosis consists a knowledge base which usually includes a rule base. The construction of the rule base is the main problem for the knowledge engineers, which has to implement, usually out of order, incomplete and heuristic knowledge of human expert. In this case the fuzzy techniques seem to be an effective tool to build the knowledge base. Unfortunately, there are many fault diagnosis problems for which the human expert knowledge is insufficient and the automatic optimal selection of the rule base is needed. Because of the exponential complexity of the problem of the optimal searching there are no possibilities of using a total review method. In this case, techniques of genetic algorithms and genetic programming (Koza 1992) may become very effective tools, assuming that decision rules are a set of complexes (Skowroński 1998). Each complex is a conjunction of selectors and each selector is a disjunction of the discrete attribute values. In this case, the population of individuals is built of vectors of selectors. The GA composes the rule base from the sets of attributes and their values. In order to use the GP to create the rule base, two sets have to be defined. The first one, terminal set, contains all possible premises and conclusions, the second one contains logic operators. Each rule is represented by a structured tree, and GP is used to find the best sets of rules. Contrary to the GA-based approach, where only simple rules (triples) are considered, the GP-based approach makes it possible to use arbitrary complex rules (Koza 1992)

The solutions proposed in this chapter are connected with two classes of the FDI systems, which are based on:

- input/output models, and
- state observers.

If the physical models are used, the identification problem reduces to an estimation of some parameters. This estimation does not seem to be a difficult problem because these parameters have usually physical interpretations. Unfortunately, the complexity of the modern industrial processes makes it impossible to construct sufficiently exact physical models. In these cases the models, which reflect the input/output behaviors of the system, are needed. There are many techniques, which can be used to build such a model. Especially, neural networks are very attractive and popular tool to non-linear system modelling (Korbicz 1997, Korbicz *et al.* 1999). These neural models are "black boxes" and give only qualitative information. An alternative approach is the genetic programming technique. The GP approaches to modelling of dynamic nonlinear systems: via choice of the gain matrix of the robust nonlinear observer (Witczak *et al.* 1999), searching for the MIMO NARX model (*Multi Input Multi Output Nonlinear AutoRegresive with eXogenous variable*) (Witczak and Korbicz 2000), selection of the state space representation of the system (Witczak *et al.* 2002), or via extended unknown input observer (EUIO) design (Witczak *et al.* 2002) (Fig. 6.5). This four GP application are presented in details in this chapter.

6.3. Tree representation of the function

All considered in this chapter GP approaches (Fig. 6.5) reduce to the problem of searching of analytical forms of some nonlinear relations between a given set of arguments $\boldsymbol{x}$ and output y

$$y = f(\boldsymbol{x}). \tag{6.2}$$

As it has already been mentioned (see section 1.2.2), a tree is the main ingredient underlying the GP algorithm. In order to adapt GP to searching the function (6.2) it is necessary to represent it as a tree, or a set of trees in the case of a vector function.

Firstly, two sets, the terminal $\mathcal{T}$ and function $\mathcal{F}$ sets, can be distinguished

$$\mathcal{T} = \{x_1, x_2, \dots, x_n, c_1, c_2, \dots, c_s\}, \quad \mathcal{F} = \{+, *, /, \xi_1(\cdot), \dots, \xi_l(\cdot)\}, \tag{6.3}$$

where $(c_i \mid i = 1, 2, \dots, s)$ is a set of constants, and $(\xi_i(\cdot) \mid i = 1, 2, \dots, l)$ is a set nonlinear univariate functions. The language of the trees in GP is formed by a user-defined function $\mathcal{F}$ and terminal $\mathcal{T}$ set, which form nodes of the trees. The function should be chosen so as they be *a priori* useful in solving the problem, i.e. any knowledge concerning the system under consideration should be included in the function set. This function set is very important and should be universal enough to be capable of representing a wide range of nonlinear systems. The terminals are usually variables or constants. Thus, the search space consists of all the possible compositions that can be recursively formed from the elements of $\mathcal{F}$ and $\mathcal{T}$. Selection of variables does not cause any problems, but the handling of numerical parameters (constants) seems very difficult. Even though no constant numerical values are in the terminal set $\mathcal{T}$, they can be implicitly generated, e.g.

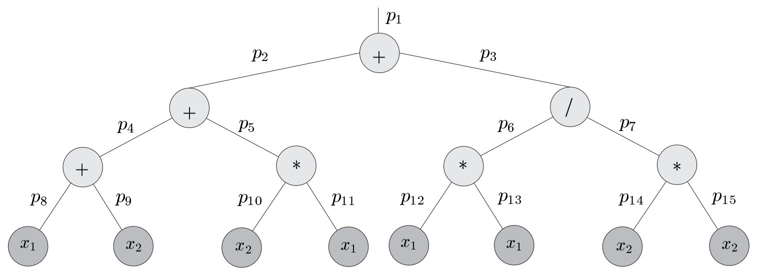


Fig. 6.6 . An exemplary tree of the two-variable function.

the number 0.5 can be expressed as $x/(x+x)$. Unfortunately, such an approach leads to an increase in both the computational burden and evolution time. Another way is to introduce a number of random constants into the terminal set, but this is also an inefficient approach. An alternative way of handling numerical parameters, which seems to be more suitable, is called *node gains* (Esparcia-Alcazar 1998). A node gain is a numerical parameter associated to a node, which multiplies its output value (see Fig. 6.6). Although this technique is straightforward, it leads to an excessive number of parameters, i.e. there are parameters which are not identifiable:

$$y = p_1\left(p_2\big(p_4(p_9x_1+p_8x_2)+p_5p_{10}p_{11}x_1x_2\big)\big)+p_3\frac{p_6p_{12}p_{13}x_2^2}{p_7p_{14}p_{15}x_1^2}\right). \tag{6.4}$$

Thus, it is necessary to develop a mechanism which prevents such situations happening. To tackle the parameters reduction problem, a few simple rules can be established (Obuchowicz and Witczak 2002, Witczak *et al.* 2002)

$*,/$: A node of type either $*$ or $/$ has always parameters set to unity on the side of its successors. If a node of the above type is a root node of a tree then the parameter associated with it should be estimated.

$+$: A parameter associated with a node of type $+$ is always equal to unity. If its successor is not of type $+$ then the parameter of the successor should be estimated.

ξ: If a successor of the node of type ξ is a leaf of a tree or is of type $*$ or $/$ then the parameter of the successor should be estimated. If a node of type ξ is a root of a tree then the associated parameter should be estimated.

As an example, consider the tree shown in Fig. 6.6. Following the above rules, the resulting parameter vector has only four elements $\boldsymbol{p} = (p_8, p_9, p_5, p_3)$ (Fig. 6.7),

$$y = p_9x_1 + p_8x_2 + p_5x_1x_2 + p_3x_2^2/x_1^2. \tag{6.5}$$

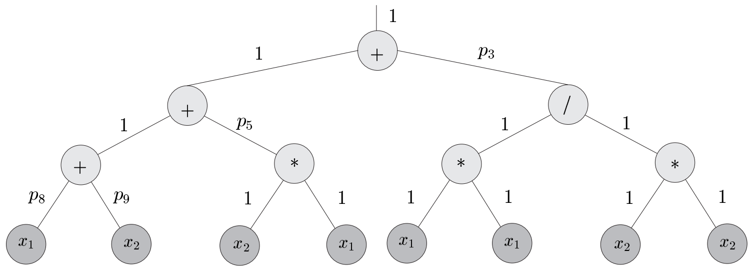


Fig. 6.7 . The tree of the function presented in Fig. 6.6 after reduction of the parameters number.

It is obvious that, although these rules are not optimal in the sense of parameter identifiability, their application reduces the dimension of the parameter vector significantly thus making the parameter estimation process much easier. Moreover, the introduction of parameterized trees reduces the terminal set to variables only, i.e. constants are no longer necessary, and hence the terminal set is given by

$$\mathcal{T} = \{x_1, x_2, \ldots, x_n\}. \tag{6.6}$$

In this way the evolutionary process of the GP searches only an optimal structure of the function (6.2) represented by a tree, whose parameters have to be estimated using another method. In the case of parameter estimation, many algorithms can be employed, more precisely, as the GP function are usually non-linear in their parameters, the choice reduces to one of non-linear optimization techniques. Unfortunately, because trees are randomly generated, they can contain linearly dependent parameters (even after the application of parameters reduction rules), and parameters which have very little influence on the model output. In many cases, this may lead to a very pure performance of the gradient-based algorithms. Owing to the above mentioned problems, the spectrum of possible non-linear optimization techniques reduces to the gradient-free techniques which usually require a large number of cost evaluations. On the other hand, the application of stochastic gradient-free algorithms, apart from their simplicity, decreases the chance to get stuck in a local optimum, and hence it may give more suitable parameter estimates. Based on numerous computer experiments, it was found that the extremely simple Adaptive Random Search (ARS) algorithm (Walter and Pronzato 1997) is especially well-suited for that purpose.

6.4. Input/output representation of the system via the GP

6.4.1. Problem statement

The characterization of a class of possible candidate models from which the system model will be obtained is an important preliminary task in any system identification procedure. Knowing that the system exhibits nonlinear characteristic, a choice of nonlinear model set must be made. In this section, the NARX was selected as the foundation for the identification methodology. The MIMO NARX model has the following form

$$\begin{aligned}\hat{y}_{i,k} = g_i(&\hat{y}_{1,k-1}, \ldots, \hat{y}_{1,k-n_{1,y}}, \ldots, \hat{y}_{m,k-1}, \ldots, \hat{y}_{m,k-n_{m,y}},\\ &u_{1,k-1}, \ldots, u_{1,k-n_{1,u}}, \ldots, u_{r,k-1}, \ldots, u_{r,k-n_{r,u}}, \boldsymbol{p}_i),\\ &i = 1, \ldots, m.\end{aligned} \tag{6.7}$$

Thus the system output is given by

$$\boldsymbol{y}_k = \hat{\boldsymbol{y}}_k + \boldsymbol{\varepsilon}_k, \tag{6.8}$$

where $\boldsymbol{\varepsilon}_k$ consists of a structural deterministic error, caused by the model-reality mismatch, and the measurement noise $\boldsymbol{v}_k$. The problem is to determine the set of models $\mathcal{M} = \{M_i = (g_i(\cdot), \boldsymbol{p}^i) \mid i = 1, 2, \ldots, m\}$, where $g_i(\cdot)$ are unknown functions and $\boldsymbol{p}^i$ are corresponding parameters vectors, which have to be estimated.

One of the best known of the criteria which can be employed to select the model structure and to estimate its parameters is the Akaike Information Criterion (AIC) (Walter and Pronzato 1997), where the following quality index is minimized

$$J_{AIC}(M_i) = \frac{1}{2} j(M_i(\hat{\boldsymbol{p}}^i)) + \frac{1}{n_T} \dim \boldsymbol{p}^i, \tag{6.9}$$

where

$$j(M_i(\boldsymbol{p}^i)) = \ln \det \sum_{k=1}^{n_T} \boldsymbol{\varepsilon}_k \boldsymbol{\varepsilon}_k^T, \tag{6.10}$$

$\hat{\boldsymbol{p}}^i = \arg\min_{\boldsymbol{p}^i} j(M_i(\boldsymbol{p}^i))$ are obtained using the identification data set of n_T pairs of input/output measurements

6.4.2. The GP approach

In order to adapt GP to system identification it is necessary to represent the model (6.7) as a tree, or a set of trees. Indeed, the MISO NARX model can be easily put in the form of a tree, and hence to build the MIMO model (6.7) it is necessary to use m trees. The function set $\mathcal{F}$ can be chosen in the form (6.3), the terminal set is given by

$$\begin{aligned}\mathcal{T} = \{&\hat{y}_{1,k-1}, \ldots, \hat{y}_{1,k-n_{1,y}}, \ldots, \hat{y}_{m,k-1}, \ldots, \hat{y}_{m,k-n_{m,y}},\\ &u_{1,k-1}, \ldots, u_{1,k-n_{1,u}}, \ldots, u_{r,k-1}, \ldots, u_{r,k-n_{r,u}}\}.\end{aligned}$$

The remaining problem is to select appropriate lags in the input and output signals of the model. For that purpose, it is possible to assume that each $n_y = n_u = n$.

Thus the problem reduces to finding, throughout experiments, such n for which the model is the best replica of the system.

If the terminal and function sets are given, populations of GP individuals (trees) can be generated, i.e. the set $\mathcal{M}$ of possible model structures is created. The algorithm works on a set of populations $\mathcal{P} = \{P_i \mid i = 1, \ldots, m\}$. Each of the above populations $P_i = \{b_{ij} \mid j = 1, \ldots, \eta\}$ is composed of a set of η trees b_{ij}. Thus, the GP searches concurrently each model M_i $(i = 1, \ldots, m)$ of the set $\mathcal{M}$ using m-th independent populations P_i of η trees.

Since the number of populations is given, the GP algorithm can be started (*initiation*) by randomly generating individuals (see section 1.2.2), i.e. η individuals are created in each population whose trees are of a desired depth n_d. Using (6.9), all considered models are estimated, estimation of the parameter vector $\boldsymbol{p}$ of each individual is performed, according to (6.10) using the ARS algorithm. If the model selected satisfies the prespecified requirements, the algorithm is stopped. In the second step, the *selection* process is applied to create a new intermediate population of "parent individuals". For that purpose, various approaches can be employed, e.g. proportional selection, rank selection, tournament selection (Koza 1992, Michalewicz 1996). The selection method used in this work is the tournament selection. The individuals for the new populations (the next generation) are produced through the application of *crossover* and *mutation*. To apply *crossover*, random couples of individuals which have the same position in each population are formed. Then, with a probability θ_c, each couple undergoes crossover, i.e. a random crossover point (node) is selected and then the corresponding sub-trees are exchanged. *Mutation* is implemented so that for each entry of each individual, a sub-tree at a selected point is removed with probability θ_m and replaced with a randomly generated tree.

The GP algorithm is repeated until the best suited model satisfies the prespecified requirements $\iota(\mathcal{P}(t))$, or until the number of maximum admissible iterations has been exceeded. It should also be pointed out that the simulation programme must ensure robustness to unstable models. This can be easily attained when (6.10) is bounded by a certain maximum admissible value. This means that each individual which exceeds the above bound is penalized by stopping the calculation of its fitness, and then $J_m(M_i)$ is set to a sufficiently large positive number. This problem is especially important in the case of input-output representation of the system. Unfortunately, the stability of the models resulting from this approach is very difficult to prove. However, this is a common problem with non-linear input-output models. To overcome this problem, an alternative state-space model structure is presented in the subsequent section.

Another reason for using state-space models in fault diagnosis tasks is that this kind of models can be employed together with robust observers, which makes it possible to increase the reliability of the entire FDI system by minimizing the influence of model uncertainty. This is, however, impossible to perform with the non-linear input-output model structure.

Tab.6.1. Specification of the process variables

$F51\_01$	Thin juice flow at the inlet of the evaporation station
$F51_02$	Steam flow at the inlet of the evaporation station
$LC51_03$	Juice level in the first section of the evaporation station
$P51_03$	Vapour pressure in the first section of the evaporation station
$P51_04$	Juice pressure at the inlet of the evaporation station
$T51_06$	Input steam temperature
$T51_07$	Vapour temperature in the first section of the evaporation station
$T51_08$	Juice temperature at the outlet of the first section of the evaporation station
$TC51_05$	Thin juice temperature at the outlet of the heater

6.4.3. System identification based on the data from the sugar factory

The real data from an industrial plant were employed to identify the input-output model of the chosen part of the plant. The plant to be considered is the evaporation station at the Lublin Sugar Factor S. A. (Poland) (Edelmayer 2000). Fig. 6.8 shows the scheme of the plant with all available process variables. These process

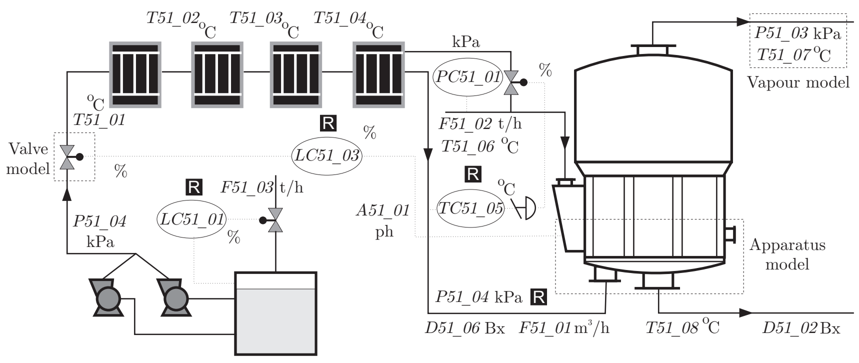


Fig. 6.8 . The scheme of the evaporation station.

variables are described in Tab. 6.1. The model to be obtained is the vapour model (cf. Fig. 6.8): the input and output vectors: $\boldsymbol{u}_k = (T51_07)$, $\boldsymbol{y}_k = (P51_03)$. The data used for the training and test sets were collected, from two different shifts, in November 1998. The data from the first one were used to the identification and the data from the second one formed the validation data set. Unfortunately, the data turned out to be sampled too fast (the sampling rate was 10s). Thus, every

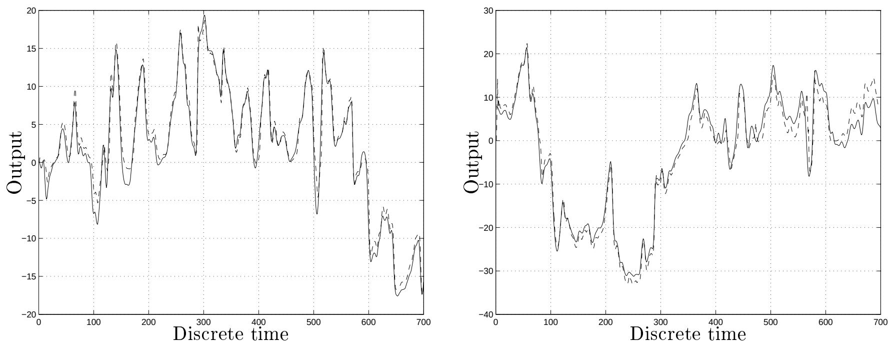


Fig. 6.9 . The system (solid line) and model (dashed line) output for the identification (left) and validation (right) data sets.

10-th value was picked, after proper prefiltering, resulting in the 700-th elements identification and validation data sets. After this the offset levels were removed with the use of MATLAB Identification Toolbox.

6.4.3.1. The vapour model

The objective of this section is to design the input-output vapour model using GP technique. The parameters used during the identification process are: the probability of the corssover $\theta_c = 0.8$, the probability of the mutation $\theta_m = 0.01$, the population size $\eta = 200$, the initial depth of trees $n_d = 10$, $\mathcal{F} = \{+, *, /\}$. The best model structure obtained is given by

$$\begin{aligned}\hat{y}_k = {} & ((p_2 u_{k-2} + p_1 \hat{y}_{k-2}) u_{k-1}^2 + (p_5 u_{k-2} \hat{y}_{k-1} + p_6 u_{k-2}^2 + p_3 \hat{y}_{k-1}^2 \\ & + p_4 \hat{y}_{k-1} u_{k-2} + p_9) u_{k-1} p_7 u_{k-2} \hat{y}_{k-1}^2 + p_8 \hat{y}_{k-1} u_{k-2}^2) / (p_{10} \hat{y}_{k-1} \\ & + p_{11} \hat{y}_{k-1}^2 + p_{12} \hat{y}_{k-1} u_{k-2} + p_{13}).\end{aligned} \tag{6.11}$$

The response of the model obtained for both the identification and validation data sets are given in Fig. 6.9. The comparative study performed for the ARX and GP (NARX) models shows that the GP model is superior to the ARX models (Witczak and Korbicz 2000). From this results it can be seen that the introduction of the nonlinear model has significantly improved modelling performance.

The main drawback to the GP-based identification algorithm concerns its convergence abilities. Indeed, it seems very difficult to establish the convergence conditions which can guarantee the convergence of the proposed algorithm. On the other hand, many examples treated in the literature, cf. (Esparcia-Alcazar 1998, Gray *et al.* 1998, Koza 1992) and the references therein, as well as the authors' experience with GP (Witczak *et al.* 2002) confirm its particular usefulness, in

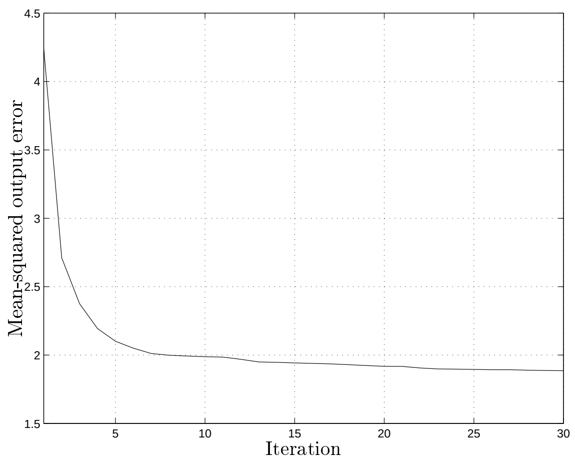


Fig. 6.10 . The average fitness for the 50 runs of the algorithm.

spite of the lack of the convergence proof. In the case of the presented example, the average fitness (mean-squared output error for the identification data set), Fig. 6.10, for the 50 runs of the algorithm confirms the modelling abilities of the approach.

Moreover, based on the fitness attained by each of the 50 models (resulting from 50 runs) it is possible to obtain the histogram representing the fitness values achieved (Fig. 6.11) as well as the fitness's confidence region. Let $\alpha = 0.99$ denote the confidence level then the corresponding confidence region can be defined as

$$\bar{J}_m \in \left[\bar{j}_m - t_\alpha \frac{s}{\sqrt{50}}, \bar{j}_m + t_\alpha \frac{s}{\sqrt{50}}\right] \tag{6.12}$$

where $\bar{j}_m = 1.89$ and $s = 0.64$ denote the mean and standard deviation of the fitness of the 50 models, $t_\alpha = 2.58$ is the normal distribution quantile. According to (6.12), the fitness's confidence region is $\bar{J}_m \in [1.65, 2.12]$, which means that there is 99% of probability that the true mean fitness $\bar{J}_m$ belongs to this region. On the other hand, owing to the multimodal properties of the identification index, it can be observed (Fig. 6.11) that there are two optima resulting in models of different quality. However, it should be pointed out that, on average (Fig. 6.10), the algorithm converges to the optimum resulting in models of better quality. The convergence abilities of the algorithm can be further increased by the application of various parameter, e.g.: θ_c, θ_m, control strategies (Eiben *et al.* 1999).

The above results confirm that, even if there is no convergence proof, the proposed approach can be successfully used to tackle the nonlinear system identification problem.

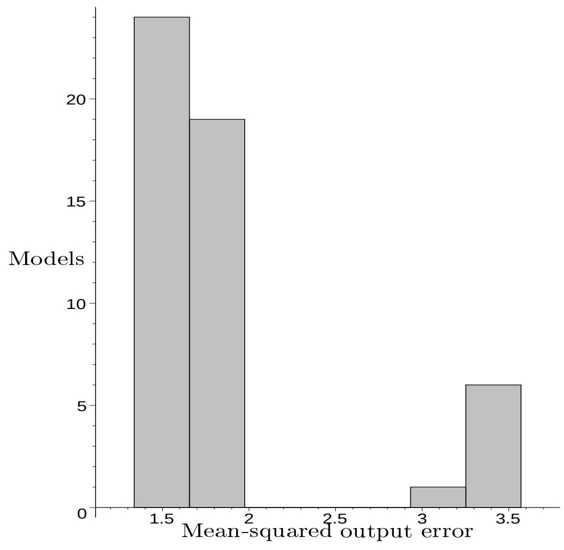


Fig. 6.11 . The histogram representing the fitness of 50 models.

6.5. Choice of the gain matrix for the robust non-linear observer via the GP

6.5.1. Problem formulation

Consider a non-linear discrete system

$$\begin{aligned} \boldsymbol{x}_{k+1} &= \boldsymbol{f}\left(\boldsymbol{x}_k, \boldsymbol{u}_k, \boldsymbol{w}_k\right), \\ \boldsymbol{y}_k &= \boldsymbol{h}\left(\boldsymbol{x}_k, \boldsymbol{v}_k\right), \end{aligned} \tag{6.13}$$

where $\boldsymbol{u}_k$ is the input, $\boldsymbol{y}_k$ is the output, $\boldsymbol{x}_k$ is the state, $\boldsymbol{w}_k$ and $\boldsymbol{v}_k$ represents the process and measurement noise, and $\boldsymbol{h}_(\cdot)$, $\boldsymbol{f}(\cdot)$ are non-linear functions.

The problem is to estimate the state $\boldsymbol{x}_k$ of the system (6.13), where a set of measured inputs and outputs and the model of the system are given. The classical methods using different kinds of an approximation are often applied (Anderson and Moore 1979, Korbicz and Bidyuk 1993) and can be given as follows

$$\begin{aligned} \hat{\boldsymbol{x}}_k &= \hat{\boldsymbol{x}}_k^- + \mathbb{K}_k \boldsymbol{\varepsilon}_k^-, \\ \boldsymbol{\varepsilon}_k^- &= \boldsymbol{y}_k - \boldsymbol{h}\left(\hat{\boldsymbol{x}}_k, \boldsymbol{0}\right), \end{aligned} \tag{6.14}$$

where $\boldsymbol{\varepsilon}_k^-$ denotes *a priori* output error, $\hat{\boldsymbol{x}}_k$ is the state estimate and $\mathbb{K}_k$ is the gain matrix.

The gain matrix $\mathbb{K}_k$ of the observer (6.14) can be searched by various methods (e.g.: the Kalman filter ($\boldsymbol{w}_k$ and $\boldsymbol{v}_k$ are assumed to be independent, white, and with normal probability distribution), the Luenberger's observer etc.) which, in large majority of them, consist of constant elements. In our approach the gain matrix is composed of certain functions, i.e. each entry of the gain matrix is a function, which depends on the *a priori* output error and the system input. Therefore, it can be written as follows

$$\hat{\boldsymbol{x}}_k = \hat{\boldsymbol{x}}_k^- + \mathbb{K}_k\left(\boldsymbol{\varepsilon}_k^-, \boldsymbol{u}_k\right)\boldsymbol{\varepsilon}_k^-. \tag{6.15}$$

Thus , the main goal is to obtain an appropriate form of $\mathbb{K}_k\left(\boldsymbol{\varepsilon}_k^-, \boldsymbol{u}_k\right)$ based on a set of measured outputs and inputs and the mathematical model of the system. Even if the mathematical model is uncertain and/or the initial state is far from its expected it seems possible to obtain such $\mathbb{K}_k\left(\boldsymbol{\varepsilon}_k^-, \boldsymbol{u}_k\right)$ to ensure the best fitness to the real system. For that purpose, the GP technique is exploited, where the gain matrix is obtained off-line from a randomly created population by means of evolutionary process.

6.5.2. Proposed algorithm

As it was mentioned in the previous section, each entry of the gain matrix is a function, and it can be represented easily as a tree in the sense of the GP formalism. It is important to note that the gain matrix consists of a list of trees. In order to apply the GP algorithm (Section 1.2.2) the sets of terms $\mathcal{T}$ and operators $\mathcal{F}$ must be defined:

$$\mathcal{T} = \{\boldsymbol{\varepsilon}_k^-, \boldsymbol{u}_k\} \quad \mathcal{F} = \{+, -, *, /\}.$$

Next, a fitness criterion must be determined. It is assumed that the fitness of the gain matrices can be represented by a sum of normalized output errors (i.e., the smaller sum the better fitness), which can be obtained by the following algorithm (Witczak *et al.* 1999)

A: Set an initial *a priori* estimate $\hat{\boldsymbol{x}}_0^-$ and set $k = 0$, $s = 0$.

B: Measurement update

$$\begin{aligned} \boldsymbol{\varepsilon}_k^- &= \boldsymbol{y}_k - \boldsymbol{h}\left(\hat{\boldsymbol{x}}_k^-, \boldsymbol{0}\right), \\ \hat{\boldsymbol{x}}_k &= \hat{\boldsymbol{x}}_k^- + \mathbb{K}_k\left(\boldsymbol{\varepsilon}_k^-, \boldsymbol{u}_k\right)\boldsymbol{\varepsilon}_k^-, \\ \hat{\boldsymbol{y}}_k &= \boldsymbol{h}\left(\hat{\boldsymbol{x}}_k, \boldsymbol{0}\right), \\ s &= s + \zeta\left(\boldsymbol{y}_k - \hat{\boldsymbol{y}}_k\right). \end{aligned}$$

C: Time update

$$\hat{\boldsymbol{x}}_{k+1}^- = \boldsymbol{f}\left(\hat{\boldsymbol{x}}_k, \boldsymbol{u}_k, \boldsymbol{0}\right)$$

If $k = n_T$ then STOP else set $k = k + 1$ and go to STEP 1.

Where $\hat{\boldsymbol{y}}_k$ denotes the system output estimate, $\zeta : \mathcal{D} \to \mathbb{R}_+ \cup \{0\}$, where $\mathcal{D}$ is the output space (e.g., $\zeta\left(\boldsymbol{y}_k - \hat{\boldsymbol{y}}_k\right) = \left(\boldsymbol{y}_k - \hat{\boldsymbol{y}}_k\right)^2$), n_t Ts the number of data points, and s is the sum of normalized output errors.

The structure of the algorithm used to obtain the gain matrix can be described as follows (Witczak *et al.* 1999)

A: Initiation

1. Choose n_d^0, θ_c, θ_m, n_{max}, $\hat{\boldsymbol{x}}_0^-$, s_{min}, n_p, and set $n = 0$.
2. Create a random population of the gain matrices

$$\left\{\mathbb{K}_i^0 = \mathrm{RANDOM}(\mathcal{T}, \mathcal{F}) \mid i = 1, 2, \ldots, \eta\right\},$$

(each entry in the gain matrix has an initial length (a number of nodes) n_d^0).

B: Calculate fitness
Using fitness calculation algorithm, for each gain matrix in the population compute the sum of normalized output errors

$$\{\mathbb{K}_i^n \to s_i^n \mid i = 1, 2, \ldots, \eta\},$$

if min $\{s_i^n \mid i = 1, 2, \ldots, \eta\} < s_{min}$ then STOP.
Set $n_{best} : s_{n_{best}}^n = \min\{s_i^n \mid i = 1, 2, \ldots, n_p\}$.

C: Selection

$$\{\mathbb{K}_i^n \mid i = 1, 2, \ldots, \eta\} \to \left\{\mathbb{K}_i^{n+1} \mid i = 1, 2, \ldots, \eta\right\}$$

where the probability of being extracted is proportional to the fitness.

D: Crossover
For each entry in each gain matrix, repeat:

1. select random independent couples of the same entries in the gain matrices $(K_{i,j,k}^{n+1}, K_{i,j,l}^{n+1})$, where i, j denote the (i, j)-th entry, $k, l \in [1, \ldots, \eta]$;
2. for each couple, select a random crossover point $l_{cross} \in U(2, \min\{\mathrm{length}(K_{i,j,k}^{n+1}), \mathrm{length}(K_{i,j,l}^{n+1})\})$;
3. exchange the sub-trees of the couple with probability θ_c.

E: Mutation

1. For each entry in each gain matrix, select a random mutation point $l_{mut} \in U(1, \mathrm{length}(K_{i,j}^{n+1}))$;
2. remove a sub-tree at the selected point l_{mut} and replace it with a randomly generated tree with probability θ_m;
3. If $n = n_{max} - 1$ then STOP else set $n = n + 1$ and go to step B.

where U denotes the uniform distribution, n_d^0 is the initial length of the trees, θ_c and θ_m are the crossover and mutation probabilities, respectively, n_{max} is the maximum number of iterations, l_{cross} and l_{mut} are realization of random independent variables with the uniform distribution (the crossover and mutation points), s_{min} is the desired fitness value, n_{best} is the index of the most fittest gain matrix and η is the population size.

6.5.3. Illustrative experiment

To illustrate the methodology of non-linear observers designing, consider a second order discrete system described by equations (Witczak *et al.* 1999)

$$\begin{aligned} x_{1,k+1} &= \tau\left(\frac{ax_{1,k}x_{2,k}}{x_{2,k}+b} - x_{1,k}\right) + x_{1,k}, \\ x_{2,k+1} &= \tau\left(-\frac{dax_{1,k}x_{2,k}}{x_{2,k}+b} + (c - x_{2,k})u_k\right) + x_{2,k}, \\ y_k &= (x_{1,k}+e)x_{2,k} + v_k, \end{aligned} \tag{6.16}$$

where $x_{1,k}$,$x_{2,k}$ denote states, y_k is the output, v_k is a realization of the random independent variable representing the measurement noise, u_k is the input signal, a, b, c, d, and e are system's parameters and τ is the sampling period.

The input signal is given by

$$u_k = 0.07\sin(0.31\tau k) + 0.38.$$

The output measurement is corrupted by a noise v_k with normal distribution $N(0, 0.0002)$. The nominal values of the model parameters are equal to $a = 0.55$, $b = 0.15$, $c = 0.8$, $d = 2.0$, $e = 0.01$ and $\tau = 0.5$. The initial state is $\boldsymbol{x}_0 = (0.21, 0.37)$ for the system to be observed, and $\hat{\boldsymbol{x}}_0^- = (2.1, 1.6)$ for the observer. For the sake of comparison, a usual Extended Kalman Filter (EKF) with the same initial condition $\hat{\boldsymbol{x}}_0^-$ is employed. Moreover, parameters eploited during an evolution of the gain matrices are: $n_d^0 = 30$, $\eta = 40$, $s_{min} = 0.001$, $\theta_c = 0.5$, $\theta_m = 0.0001$. The population was learned over a sample of $n_T = 200$ simulated measurements.

As shown in Fig 6.12, the estimated state x_2 approaches the real state for the proposed observer but not for the EKF. Further simulation results have shown that the proposed observer has a larger domain of attraction that the EKF, i.e., the initial estimation error may be larger. As it was mentioned, even if the initial state estimate is known there is still a problem of a model uncertainty, e.g. parameter uncertainty.

Reconsider the non-linear system (6.16) and assume that the values of the model parameters are slightly modified: $a = 0.53$, $b = 0.17$, $c = 0.78$, $d = 2.0$, $e = 0.0099$, and other parameters are the same as previously.

For the sake of comparison, it is assumed that the initial *a priori* state estimate is close to the real state so as to ensure the stability of the EKF, i.e. $\hat{\boldsymbol{x}}_0^- = 1$. As shown in Fig. 6.13, that state estimation error $\boldsymbol{e}_k = \boldsymbol{x}_k - \hat{\boldsymbol{x}}_k$ is closer zero for the proposed observer that for the EKF. Further simulation results have shown that the proposed observer is less sensitive to the model uncertainty that the EKF.

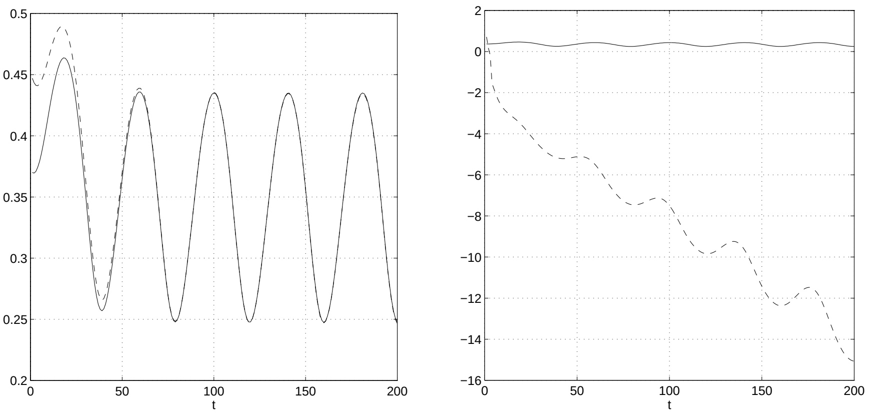


Fig. 6.12. The real state x_2 (solid line) and its estimates obtained by the proposed observer (left) and the EKF (right)

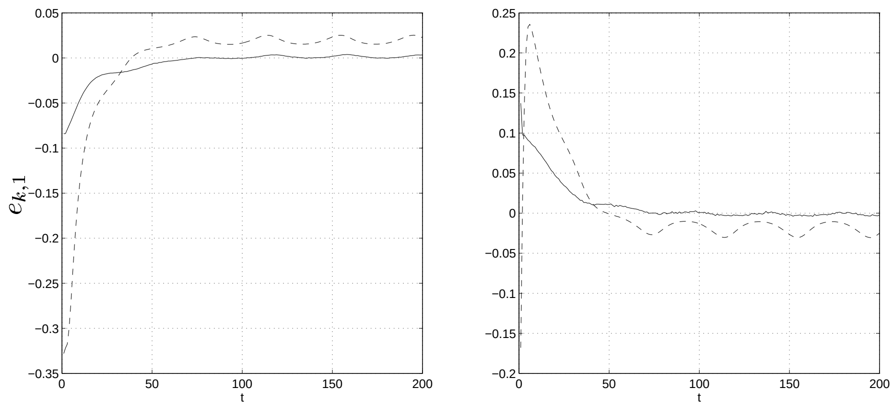


Fig. 6.13. State estimation errors for the proposed observer(solid line) and the EKF (dashed line)

6.6. The GP approach to state space representation of the system

6.6.1. Design of state space models

Let us consider the following class of nonlinear discrete-time systems

$$\begin{aligned} \boldsymbol{x}_{k+1} &= \boldsymbol{g}(\boldsymbol{x}_k, \boldsymbol{u}_k) + \boldsymbol{w}_k, \\ \boldsymbol{y}_{k+1} &= \mathbb{C}\boldsymbol{x}_{k+1} + \boldsymbol{v}_k. \end{aligned} \tag{6.17}$$

Assume that the function $\boldsymbol{g}(\cdot)$ has the form

$$\boldsymbol{g}(\boldsymbol{x}_k, \boldsymbol{u}_k) = \mathbb{A}(\boldsymbol{x}_k)\boldsymbol{x}_k + \boldsymbol{h}(\boldsymbol{u}_k). \tag{6.18}$$

Thus, the state-space model of the system (6.17) can be expressed as

$$\begin{aligned} \hat{\boldsymbol{x}}_{k+1} &= \mathbb{A}(\hat{\boldsymbol{x}}_k)\hat{\boldsymbol{x}}_k + \boldsymbol{h}(\boldsymbol{u}_k), \\ \hat{\boldsymbol{y}}_{k+1} &= \mathbb{C}\hat{\boldsymbol{x}}_{k+1}. \end{aligned} \tag{6.19}$$

Without loss of generality, it is possible to assume that

$$\mathbb{A}(\hat{\boldsymbol{x}}_k) = \mathrm{diag}[a_{i,i}(\hat{\boldsymbol{x}}_k) \mid i = 1, 2, \ldots, n]. \tag{6.20}$$

The problem reduces to identifying nonlinear functions $a_{i,i}(\hat{\boldsymbol{x}}_k), h_i(\boldsymbol{u}_k)$ $(i = 1, \ldots, n)$, and the matrix $\mathbb{C}$. Assuming $\max_{i=1,\ldots,n} |a_{i,i}(\hat{\boldsymbol{x}}_k)| < 1$ it can be shown (Witczak *et al.* 2002) that the model (6.19) is globally asymptotically stable. This implies that $a_{i,i}(\hat{\boldsymbol{x}}_k)$ should have the following structure

$$a_{i,i}(\hat{\boldsymbol{x}}_k) = \tanh(s_{i,i}(\hat{\boldsymbol{x}}_k)), i = 1, \ldots, n, \tag{6.21}$$

where $\tanh(\cdot)$ is a hyperbolic tangent function, and $s_{i,i}(\hat{\boldsymbol{x}}_k)$ is a function to be determined.

In order to identify $s_{i,i}(\hat{\boldsymbol{x}}_k), h_i(\boldsymbol{u}_k)$ $(i = 1, \ldots, n)$, and the matrix $\mathbb{C}$ the GP algorithm described in section 6.4 is applied. The fitness function is defined by (6.9).

6.6.2. The apparatus model

Let us consider the system described in Section 6.4.3. The objective of this section is to design the state-space apparatus model,
$\boldsymbol{u}_k = (T51_06, TC51_05, F51_01, F51_02)$, $\boldsymbol{y}_k = (T51_08)$ (Tab. 6.1), according to the approach described above. The parameters used in the GP algorithm are the same as in Section 6.4.3.1. The best model structure obtained is given by

$$\begin{aligned} \hat{x}_{1,k+1} &= \tanh(s_{1,1}) + h_1(\boldsymbol{u}_k), \\ \hat{x}_{2,k+1} &= \tanh(s_{2,2}) + h_2(\boldsymbol{u}_k), \end{aligned} \tag{6.22}$$

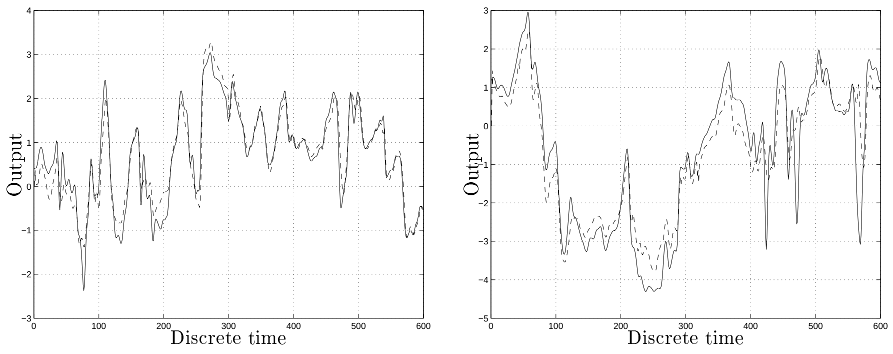


Fig. 6.14 . The system (solid line) and model (dashed line) output for the identification (left) and validation (right) data sets.

where

$$s_{1,1} = -0.13\hat{x}_{2,k}, \quad s_{2,2} = \frac{\hat{x}_{2,k}}{\hat{x}_{1,k}(\hat{x}_{2,k}\hat{x}_{1,k} + 2\hat{x}_{2,k}^2\hat{x}_{1,k}^2 + 1)},$$

$$\begin{aligned} h_1(\boldsymbol{u}_k) = \;& (u_{1,k} + (u_{1,k} + 2u_{4,k} + u_{4,k}u_{1,k})(u_{1,k} + u_{4,k} + u_{3,k} + \\ & + u_{4,k}u_{1,k}))u_{3,k} + u_{3,k} + (u_{1,k} + (u_{1,k} + u_{4,k} + u_{3,k} + \\ & + u_{4,k}u_{1,k})(u_{1,k} + \frac{u_{4,k}u_{3,k}u_{1,k}}{u_{4,k} + u_{2,k}} + 2u_{4,k})), \\ h_2(\boldsymbol{u}_k) = \;& u_{1,k} + u_{2,k}, \end{aligned}$$

and

$$\mathbb{C} = [0.21 * 10^{-5}, 0.51].$$

The response of the model obtained for both the identification and validation data sets is given in Fig. 6.14. The comparative study for the linear model and the GP model has been performed in (Witczak *et al.* 2002). from this results it can be seen that the proposed nonlinear state-space model identification approach can be effectively applied to various system identification tasks. The average fitness (mean-squared output error for the identification data set), Fig. 6.15, for the 50 runs of the algorithm confirms the modelling abilities of the approach. As previously, based on the fitness attained by each of the 50 models (resulting from 50 runs) it is possible to obtain the histogram representing the fitness values achieved (Fig. 6.16) as well as the fitness's confidence region. According to (6.12), the fitness's confidence region is $\bar{J}_m \in [0.06, 0.78]$ (for: $s = 0.2$, $\bar{\jmath}_m = 0.07$), which means that there is 99% of probability that the true mean fitness $\bar{J}_m$ belongs to this region. Similarly to the previous section, it can be observed (Fig. 6.16) that there are two optima in the space of models. However, on average, the algorithm is convergent to the optima resulting in models of better quality.

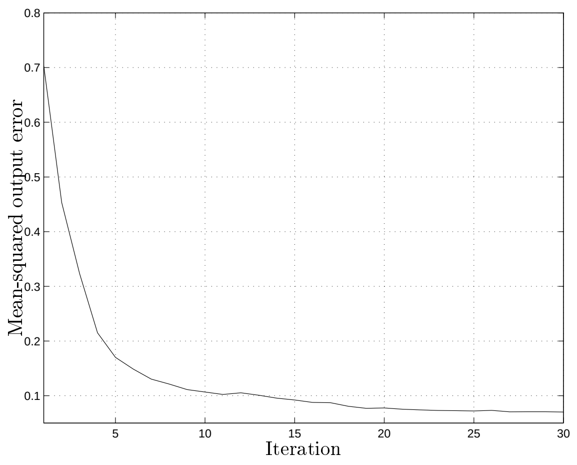


Fig. 6.15 . The average fitness for the 50 runs of the algorithm.

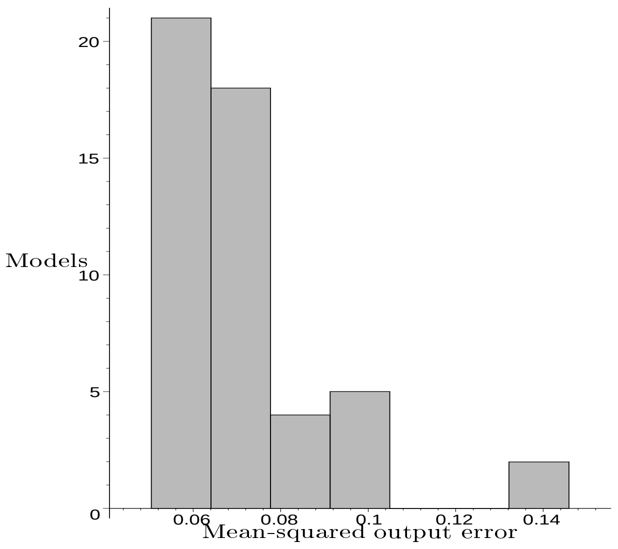


Fig. 6.16 . The histogram representing the fitness of 50 models.

6.7. GP approach to the EUIO design

6.7.1. Problem formulation

Let us consider a class of nonlinear system described by the following equations

$$\begin{aligned}\boldsymbol{x}_{k+1} &= \boldsymbol{g}(\boldsymbol{x}_k) + \boldsymbol{h}(\boldsymbol{u}_k + \mathbb{L}_{1,k}\boldsymbol{f}_k) + \mathbb{E}_k\boldsymbol{d}_k,\\ \boldsymbol{y}_{k+1} &= \mathbb{C}_{k+1}\boldsymbol{x}_{k+1} + \mathbb{L}_{2,k+1}\boldsymbol{f}_{k+1},\end{aligned} \tag{6.23}$$

where $\boldsymbol{g}(\boldsymbol{x}_k)$ is assumed to be continuously differentiable with respect to $\boldsymbol{x}_k$, $\boldsymbol{f}_k$ states for the fault signal, $\boldsymbol{d}_k$ is the unknown input, and $\mathbb{L}_{1,k}$, $\mathbb{L}_{2,k}$, and $\mathbb{E}_k$ are their distribution matrices. Similarly to EKF (Anderson and Moore 1979), the Unknown Input Observer (UIO) (Chen and Patton 1999) can be extended to the class of nonlinear systems (6.23). This leads to the following structure of the Extended UIO (EUIO) (Witczak *et al.* 2002):

$$\begin{aligned}\hat{\boldsymbol{x}}_{k+1/k} &= \boldsymbol{g}(\hat{\boldsymbol{x}}_k) + \boldsymbol{h}(\boldsymbol{u}_k),\\ \hat{\boldsymbol{x}}_{k+1} &= \hat{\boldsymbol{x}}_{k+1/k} + \mathbb{H}_{k+1}\boldsymbol{\varepsilon}_{k+1/k} + \mathbb{K}_{1,k+1}\boldsymbol{\varepsilon}_k,\end{aligned} \tag{6.24}$$

Witczak *et al.* (2002) performed a comprehensive convergence analysis with the Lyapunov method. As a result they obtained the following conditions

$$\bar{\sigma}\left(\boldsymbol{\alpha}_k\right) \le \gamma_1 = \frac{\underline{\sigma}\left(\mathbb{A}_k\right)}{\bar{\sigma}\left(\mathbb{A}_k\right)}\left(\frac{(1-\zeta)\underline{\sigma}\left(\mathbb{P}_k\right)}{\bar{\sigma}\left(\mathbb{A}_{1,k}\mathbb{P}'_k\mathbb{A}_{1,k}^T\right)}\right)^{\frac{1}{2}}. \tag{6.25}$$

and

$$\bar{\sigma}\left(\boldsymbol{\alpha}_k - \mathbb{I}\right) \le \gamma_2 = \left(\frac{\underline{\sigma}\left(\mathbb{C}_k^T\right)\underline{\sigma}\left(\mathbb{C}_k\right)}{\bar{\sigma}\left(\mathbb{C}_k^T\right)\bar{\sigma}\left(\mathbb{C}_k\right)}\frac{\underline{\sigma}\left(\mathbb{R}_k\right)}{\bar{\sigma}\left(\mathbb{C}_k\mathbb{P}_k\mathbb{C}_k^T + \mathbb{R}_k\right)}\right)^{\frac{1}{2}}. \tag{6.26}$$

where $\mathbb{P}_k$ is the state estimate covariance matrix, and

$$\mathbb{H}_{k+1} = \mathbb{E}_k\left[(\mathbb{C}_{k+1}\mathbb{E}_k)^T\mathbb{C}_{k+1}\mathbb{E}_k\right]^{-1}(\mathbb{C}_{k+1}\mathbb{E}_k)^T. \tag{6.27}$$

$$\mathbb{A}_k = \left.\frac{\partial \boldsymbol{g}(\boldsymbol{x}_k)}{\partial \boldsymbol{x}_k}\right|_{\boldsymbol{x}_k=\hat{\boldsymbol{x}}_k}. \tag{6.28}$$

Bearing in mind that $\boldsymbol{\alpha}_k$ is a diagonal matrix, the above inequalities can be expressed as

$$\max_{i=1,\ldots,n}|\alpha_{i,k}| \le \gamma_1 \quad \text{and} \quad \max_{i=1,\ldots,n}|\alpha_{i,k}-1| \le \gamma_2. \tag{6.29}$$

Since (Chen and Patton 1999)

$$\mathbb{P}_k = \mathbb{A}_{1,k}\mathbb{P}'_k\mathbb{A}_{1,k}^T + \mathbb{T}_k\mathbb{Q}_{k-1}\mathbb{T}_k^T + \mathbb{H}_k\mathbb{R}_k\mathbb{H}_k^T, \tag{6.30}$$

it is clear that an appropriate selection of the instrumental matrices $\mathbb{Q}_{k-1}$ and $\mathbb{R}_k$ may enlarge the bounds γ_1 and γ_2, and consequently the domain of attraction. Indeed, if the conditions (6.29) are satisfied then $\hat{\boldsymbol{x}}_k$ converges to $\boldsymbol{x}_k$.

The problem is to obtain an appropriate form of the instrumental matrices $\mathbb{Q}_{k-1}$ and $\mathbb{R}_k$ in such a way as to ensure the convergence of the observer or adequately to maximize the bounds of the diagonal elements of the matrix $\boldsymbol{\alpha}_k$.

First, let us define the identification criterion consisting a necessary ingredient of the $\mathbb{Q}_{k-1}$ and $\mathbb{R}_k$ selection process. Since the instrumental matrices should be chosen so as to satisfy (6.25), the selection of $\mathbb{Q}_{k-1}$ and $\mathbb{R}_k$ can be performed according to

$$(\mathbb{Q}_{k-1}, \mathbb{R}_k) = \arg \max_{q(\boldsymbol{\varepsilon}_{k-1}), r(\boldsymbol{\varepsilon}_k)} j_{\text{obs},1}(q(\boldsymbol{\varepsilon}_{k-1}), r(\boldsymbol{\varepsilon}_k)), \tag{6.31}$$

where

$$j_{\text{obs},1}(q(\boldsymbol{\varepsilon}_{k-1}), r(\boldsymbol{\varepsilon}_k)) = \sum_{k=0}^{n_t-1} \text{trace}\mathbb{P}_k. \tag{6.32}$$

On the other hand, owing to the FDI requirements, it is clear that the output error should be near zero in the fault free mode. In this case, one can define another identification criterion

$$(\mathbb{Q}_{k-1}, \mathbb{R}_k) = \arg \min_{q(\boldsymbol{\varepsilon}_{k-1}), r(\boldsymbol{\varepsilon}_k)} j_{\text{obs},2}(q(\boldsymbol{\varepsilon}_{k-1}), r(\boldsymbol{\varepsilon}_k)), \tag{6.33}$$

where

$$j_{\text{obs},2}(q(\boldsymbol{\varepsilon}_{k-1}), r(\boldsymbol{\varepsilon}_k)) = \sum_{k=0}^{n_t-1} \boldsymbol{\varepsilon}_k^T \boldsymbol{\varepsilon}_k. \tag{6.34}$$

Therefore, in order to join (6.31) and (6.33), the following identification criterion is employed

$$(\mathbb{Q}_{k-1}, \mathbb{R}_k) = \arg \min_{q(\boldsymbol{\varepsilon}_{k-1}), r(\boldsymbol{\varepsilon}_k)} j_{\text{obs},3}(q(\boldsymbol{\varepsilon}_{k-1}), r(\boldsymbol{\varepsilon}_k)) \tag{6.35}$$

where

$$j_{\text{obs},3}(q(\boldsymbol{\varepsilon}_{k-1}), r(\boldsymbol{\varepsilon}_k)) = \frac{j_{\text{obs},2}(q(\boldsymbol{\varepsilon}_{k-1}), r(\boldsymbol{\varepsilon}_k))}{j_{\text{obs},1}(q(\boldsymbol{\varepsilon}_{k-1}), r(\boldsymbol{\varepsilon}_k))} \tag{6.36}$$

6.7.2. Increasing the convergence rate via GP

Unfortunately, an analytical derivation of the $\mathbb{Q}_{k-1}$ and $\mathbb{R}_k$ matrices seems to be an extremely difficult problem. However, it is possible to set the above matrices as follows $\mathbb{Q}_{k-1} = \beta_1\mathbb{I}$, $\mathbb{R}_k = \beta_1\mathbb{I}$, with β_1 and β_1 large enough. On the other hand, it is well known that the convergence rate of such an EKF-like approach can be increased by an appropriate selection of the covariance matrices $\mathbb{Q}_{k-1}$ and $\mathbb{R}_k$, i.e.

the more accurate (near "true" values) covariance matrices the better convergence rate. This means that, in the deterministic case ($\boldsymbol{w}_k = \mathbf{0}$ and $\boldsymbol{v}_k = \mathbf{0}$), both the matrices should be zero ones. Unfortunately, such an approach usually leads to divergence of the observer as well as other computational problems. To tackle this problem a compromise between convergence and convergence rate should by established. This can be easily done by setting the instrumental matrices as

$$\begin{aligned} \mathbb{Q}_{k-1} &= \beta_1 \boldsymbol{\varepsilon}_{k-1}^T \boldsymbol{\varepsilon}_{k-1} \mathbb{I} + \delta_1 \mathbb{I}, \\ \mathbb{R}_k &= \beta_2 \boldsymbol{\varepsilon}_k^T \boldsymbol{\varepsilon}_k \mathbb{I} + \delta_2 \mathbb{I}, \end{aligned} \tag{6.37}$$

with β_1, β_2 large enough, and δ_1, δ_2 small enough. Although this approach is very simple, it is possible to increase the convergence rate further. Indeed, the instrumental matrices can be set as follows

$$\begin{aligned} \mathbb{Q}_{k-1} &= q^2(\boldsymbol{\varepsilon}_{k-1}) \mathbb{I}, \\ \mathbb{R}_k &= r^2(\boldsymbol{\varepsilon}_k) \mathbb{I}, \end{aligned} \tag{6.38}$$

where $q(\boldsymbol{\varepsilon}_{k-1})$ and $r(\boldsymbol{\varepsilon}_k)$ are nonlinear functions of the output error $\boldsymbol{\varepsilon}_k$ (the squares are used to ensure the positive definiteness of $\mathbb{Q}_{k-1}$ and $\mathbb{R}_k$).

Thus, the problem reduces to identifying the above functions. To tackle this problem the genetic programming can be employed. The unknown functions $q(\boldsymbol{\varepsilon}_{k-1})$ and $r(\boldsymbol{\varepsilon}_k)$ can be expressed as a tree. In the case of $q(\cdot)$ and $r(\cdot)$ the terminal sets are $\mathcal{T} = \{\boldsymbol{\varepsilon}_{k-1}\}$ and $\mathcal{T} = \{\boldsymbol{\varepsilon}_k\}$, respectively. In both the cases, the function set can be defined as $\mathcal{F} = \{+, *, /, \xi_1(\cdot), \ldots, \xi_l(\cdot)\}$, where $\xi_k(\cdot)$ is a nonlinear univariate function, and consequently the number of populations is $m = 2$. Since the terminal and function sets are given, the approach described in Section 6.4 can be easily adapted for the identification purpose of $q(\cdot)$ and $r(\cdot)$ using the identification criterion (6.36).

6.7.3. State estimation and fault diagnosis of an induction motor using EUIO

The numerical example considered here is the fifth-order two-phase nonlinear model of an induction motor which has already been the subject of a large number of various control design applications (see (Boutayeb and Aubry 1999) and the references therein). Moreover, the above model, unlike the model of Section 6.6.2, can be used by other researchers and hence a straightforward comparison to other approaches can be realized.

The complete discrete time model in stator fixed (a,b) reference frame is

$$\begin{aligned}
x_{1,k+1} &= x_{1,k} + h(-\gamma x_{1k} + \frac{K}{T_r}x_{3k} + Kpx_{5k}x_{4k} + \frac{1}{\sigma L_s}u_{1k}) + 0.01d_{1,k},\\
x_{2,k+1} &= x_{1,k} + h(-\gamma x_{2k} + Kpx_{5k}x_{3k} + \frac{K}{T_r}x_{4k} + \frac{1}{\sigma L_s}u_{2k}) + 0.01d_{1,k},\\
x_{3,k+1} &= x_{1,k} + h(\frac{M}{T_r}x_{1k} - \frac{1}{T_r}x_{3k} - px_{5k}x_{4k}) + d_{1,k},\\
x_{4,k+1} &= x_{1,k} + h(\frac{M}{T_r}x_{2k} - px_{5k}x_{3k} - \frac{1}{T_r}x_{4k}) + d_{1,k},\\
x_{5,k+1} &= x_{1,k} + h(\frac{pM}{JL_r}(x_{3k}x_{2k} - x_{4k}x_{1k}) - \frac{T_L}{J}),\\
y_{1,k+1} &= x_{1,k+1},\ y_{2,k+1} = x_{2,k+1}.
\end{aligned} \tag{6.39}$$

where $\boldsymbol{x}_k = (x_{1,k}, \dots, x_{n,k}) = (i_{\text{sak}}, i_{\text{sbk}}, \psi_{\text{rak}}, \psi_{\text{rbk}}, \omega_k)$ represents the currents, the rotor fluxes, and the angular speed, respectively. $\boldsymbol{u}_k = (u_{\text{sak}}, u_{\text{sbk}})$ is the stator voltages control vector, p is the number of pair of poles, T_L is the load torque. The rotor time constant T_r and the remaining parameters are defined as

$$T_r = \frac{L_r}{R_r},\ \sigma = 1 - \frac{M^2}{L_sL_r},\ K = \frac{M}{\sigma L_s L_r^2},\ \gamma = \frac{R_s}{\sigma L_s} + \frac{R_r M^2}{\sigma L_s L_r^2}, \tag{6.40}$$

where R_s, R_r and L_s, L_r are stator and rotor per-phase resistances and inductances, respectively, and J is the rotor moment inertia.
The numerical values of the above parameters are as follows: $R_s = 0.18\ \Omega$, $R_r = 0.15\ \Omega$, $M = 0.068$ H, $L_s = 0.0699$ H, $L_r = 0.0699$ H, $J = 0.0586$ kgm^2, $T_L = 10$ Nm, $p = 1$, and $h = 0.1$ ms. The initial condition for the observer and the system are $\hat{\boldsymbol{x}}_k = (200, 200, 50, 50, 300)$ and $\boldsymbol{x}_k = \boldsymbol{0}$. The unknown input distribution matrix is

$$\mathbb{E}_k = \begin{bmatrix} 0.01 & 0 & 1 & 0 & 0 \\ 0 & 0.01 & 0 & 1 & 0 \end{bmatrix}^T, \tag{6.41}$$

and hence, according to (6.27), the matrix $\mathbb{H}_k$ is

$$\mathbb{H}_k = \begin{bmatrix} 1 & 0 & 100 & 0 & 0 \\ 0 & 1 & 0 & 100 & 0 \end{bmatrix}^T, \tag{6.42}$$

The input signals are

$$u_{1,k} = 300\cos(0.03k), \quad u_{2,k} = 300\sin(0.03k). \tag{6.43}$$

The unknown input is defined as

$$d_{1,k} = 0.002\sin(0.5\pi k)\cos(0.3\pi k), \quad 0.005\sin(0.01k), \tag{6.44}$$

and $\mathbb{P}_0 = 10^3\mathbb{I}$.
Moreover, the following three cases concerning the selection of $\mathbb{Q}_{k-1}$ and $\mathbb{R}_k$ were considered

Case 1: Classical approach (constant values),i.e. $\mathbb{Q}_{k-1} = 0.1$, $\mathbb{R}_k = 0.1$

Case 2: Selection according to (6.37), i.e.

$$\mathbb{Q}_{k-1} = 10^3\varepsilon_{k-1}^T\varepsilon_{k-1}\mathbb{I} + 0.01\mathbb{I}, \quad \mathbb{R}_k = 10\varepsilon_k^T\varepsilon_k\mathbb{I} + 0.01\mathbb{I}, \tag{6.45}$$

Case 3: GP-based approach

In order to obtain the matrices $\mathbb{Q}_{k-1}$ and $\mathbb{R}_k$ using the GP-based approach (Case 3), a set of $n_t = 300$ input-output measurements was generated according to (6.39). As a result, the following form of the instrumental matrices were obtained

$$\begin{aligned}\mathbb{Q}_{k-1} &= \left(10^2\varepsilon_{1,k-1}^2\varepsilon_{2,k-1}^2 + 1012\varepsilon_{1,k-1} + 103.45\varepsilon_{1,k-1} + 0.01\right)^2\mathbb{I},\\ \mathbb{R}_k &= \left(112\varepsilon_{1,k}^2 + 0.1\varepsilon_{1,k}\varepsilon_{2,k} + 0.12\right)^2\mathbb{I}.\end{aligned} \tag{6.46}$$

The parameters used in the GP algorithm were the same as in Section 6.4.3.1. It should be also pointed out that the above matrices (6.46) are formed by simple polynomials. This, however, may not be the case for other applications.
Simulation results (for all the cases) are shown in Fig. 6.17. The numerical values of the optimization index (6.36) are as follows: Case 1 $\jmath_{\text{obs}} = 1.49 * 10^5$, Case 2 $\jmath_{\text{obs}} = 1.55$, Case 3 $\jmath_{\text{obs}} = 1.2 * 10^{-16}$. Both of the above results as well as the plots shown in Fig. 6.17 confirm the relevance of the appropriate selection of the gain matrices. Indeed, as it can be seen, the proposed approach is superior to the classical technique of selecting the instrumental matrices $\mathbb{Q}_{k-1}$ and $\mathbb{R}_k$.

6.8. Summary

Although there are few applications of evolutionary algorithms to fault diagnosis systems, a discussion of existing solutions and their possibilities as well as the possibilities of further development have been presented in this chapter. Emphasis has been put on genetic programming approaches to the residual generation module design. In particular, it has been shown how to represent various model structures as a parameterized trees and how to identify their structure as well as to estimate their parameters. Both the input-output NARX and state-space model structures are presented. Moreover, it has been proven that the proposed state-space model

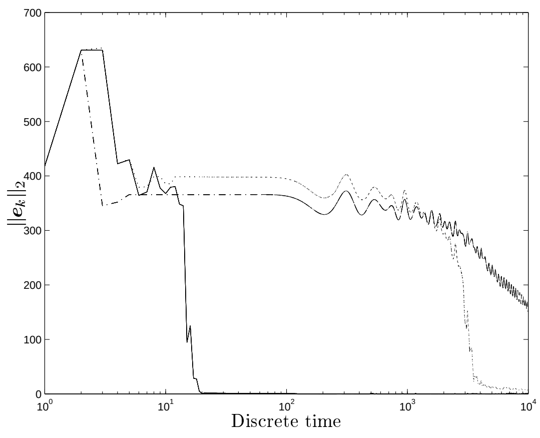


Fig. 6.17. The state estimation error norm $\|\boldsymbol{e}_k\|_2$ for Case 1 (dash-dotted line), Case 2 (dotted line) and Case 3 (solid line).

identification scheme provides asymptotically stable models. The experimental results, covering models construction of chosen parts of an evaporation station at the Lublin Sugar Factory S.A., confirm the reliability and effectiveness of the proposed framework. The main drawback to this approach is its computational cost resulting in a relatively long identification time. However, the model construction procedure is as usual realized *off-line* and hence the identification time is not very important.

Another GP approach concerns the concept of the Extended Unknown Input Observer. It can be shown, with the use of the Lyapunov approach, that the considered fault detection observer is convergent under certain conditions. Moreover, it has been shown that an appropriate selection of the instrumental matrices $\mathbb{Q}_{k-1}$ and $\mathbb{R}_k$ strongly influences the convergence properties. To tackle the instrumental matrices selection problem a genetic programming based approach has been proposed. It has been shown, by an example with an induction motor, that the proposed observer can be a useful tool for both state estimation and fault diagnosis problems of nonlinear deterministic systems. This is mainly because of the convergence properties of the observer which confirm its superiority to the classical approaches.

Chapter 7

CONCLUDING REMARKS

Since the beginning of the human civilization the technological development has been strongly inspired by the solutions existing in nature. A man building models of observed phenomena wants to detect the principles lying under them, and, simultaneously, searches for methods and algorithms for his problems solving. Especially, one work of nature is worth noticing: the evolution. The evolution is the creative strength of the animated nature. Thanks to this process the most incredible and complicated organisms, which are adapted to live in almost all, even the most extreme, conditions existing on Earth, have been created. The evolution owes its power to its parallel processing and soft selection rules, which, implemented in the algorithmic form, result in one of the most effective tool of *Computational Intelligence.*

The applicability of EAs in global optimization tasks, both in the continuous and discrete domain, is not questionable. Many instances of successful EA implementations are published every year (cf. (Bäck 1995, Bäck *et al.* 1997, Galar 1990, Osyczka 2002, Schaefer 2002)). The main disadvantage of EAs is their computational complexity that these algorithms are suggested to be optimization methods of "the last resort", when other, conventional, techniques have disappointed.

In this book, the emphasis is put on the ESSS algorithm, which, thanks to its simplicity, seems to be a very attractive subject for research analyzing the basic properties of the evolutionary concept in the global optimization problems. The simply selection–mutation model contains the principal idea of the phenotype evolution (Chapter 1). Basing on this model, properties of different techniques of natural exploration have been tested and analyzed in details.

Two classes of exploration techniques in EAs can be distinguish in the literature: *local* and *global* ones. The first class is based on the idea that individuals "live" independently and separated and only the natural selection and random mutation control the evolutionary exploitation and exploration phenomena, respectively. This rule is accepted in almost all known implementations of EAs. Most of the solutions proposed in Chapter 2 possess a different and original character. Population evolve like a herd. It is subject to the selection and mutation mechanisms as well as to some form of intelligence — "herd instinct" — which controls some global behaviors. The *trap test* for the SVA, IP and DOF mechanisms as well as the FDM technique can be treated as some forms of this idea implementations. However, the ideas of most mechanisms proposed in this book are known, but two of them: the erosion mechanism (ESSS-DOF) and the forced direction of mutation (ESSS-FDM) are new. In the ESSS-DOF algorithm, the population erodes a currently occupied peak. Owing to this fact, already exploited areas are just unattractive for the population, which runs away toward new territories. This method allows to search the widest area of the domain in a given number of iterations in comparison with other considered techniques. The ESSS-FDM algorithm has been mainly proposed for adaptation tasks in non-stationary environments. Basing on it, the extremely effective learning method for the DMLP network has been designed.

The multi-dimensional Gaussian mutation is the most popular mutation technique in evolutionary algorithms based on the floating point representation of individuals. In the case of a one-dimensional mutation, the most probable location of the offspring is the nearest neighbourhood of the parent individual. But in the case of n-dimensional one, the most probable location moves from the center of mutation to the "ring" of the radius proportional to the norm of the standard deviation vector, which increases with landscape dimension whenever the standard deviation of each entry is fitted.
In recent years, the multi-dimensional Cauchy mutation has attracted a lot of research attention. Evolutionary algorithms which use the Cauchy mutations seem to be more effective in comparison to algorithms with the Gaussian mutation, in the case of most global optimization problems. But the multi-dimensional Cauchy density function obtained as a product of n independent one-dimensional Cauchy density functions is not isotropic. The convergence of the density function shape to the zero value is different for different directions in the n-dimensional real space. However, the non-spherical symmetry of the Cauchy mutation is well-known in the statistical literature, the influence of the *symmetry effect* on a phenotype evolutionary algorithm efficiency needs detailed studies. The author has not found any mention of the *surrounding effect*, thus it is supposed that this problem is not commonly known. Both phenomenons are investigated in Chapter 3. Simulation experiments prove that both effects are profitable from the exploration property point of view, but their exploitation abilities decrease with the increase of the landscape dimension, especially in the case of narrow peaks. Proposed modified versions of the Gaussian and Cauchy mutation are deprived of this disadvantage.

Evolutionary algorithms, especially in their phenotype manner, are very interesting from economic analyzers point of view. There are many suggestions that so called "free market" is subject of the evolution principle (Galar 1990). Simulations of the simple phenotype evolution, like the ESSS algorithm, may be the source of the information about economic trends and behaviors. The main characteristic of the "economic environment" is its non-stationarity and continual interaction with economic entities. Recently, the problem of adaptation in the non-stationary environments as well as the models of a population–environment interaction seem to be very attractive for bigger and bigger number of researches. The number of publication successively increases. Unfortunately, the diverse methodology and terminology cause most of results being not comparable. An analysis and classification of these problems, review of the existing measures and some propositions of new ones are presented in Chapter 4. We hope, that they can bring better understanding of the considered problem and optimization tool behavior, and therefore provide more satisfying results.

The problem of the neural model designing is mainly connected with two optimization processes (see Chapter 5): a learning process and an optimal architecture allocation. The nature of the both processes is different. The learning process belongs to a class of global optimization in a multi-dimensional continuous domain. The space of neural networks architectures is discrete and infinite.
In order to model a dynamic systems, a dynamic neural networks should be used. One of the most interesting solution is application a locally recurrent neural network: the DMLP. Such a network is organized in the well-known MLP structure and dynamic is included in the particular DNM units. Similarly to the classical MLP, the DMLP allows to construct the learning algorithm based on the BP principle: the Extended Dynamic Back-Propagation algorithm. This algorithm trains the weights of synaptic connections as well as feedforward and feedback parameters of IIR filters contained in each DMN unit. The main disadvantage of the EDBP algorithm is its local manner. As well as all algorithms based on the "gradient descent" method, the EDBP usually gets stuck in one of local optima of a multimodal mean square error function. Thus, the methods of global optimization, like evolutionary algorithms, can improve the effectiveness of learning process.
The discrete nature of the ANN architectures space suggests that the methods of discrete optimization are a proper tool for allocation of the optimal neural model architecture. Four of them: Simulating Annealing, Genetic Algorithms, $A^\star$ algorithm and Tabu Search, are intensively studied in this book. Simulation experiments suggest that the $A^\star$ algorithm seems to be the most attractive tool for optimization of the DMLP architecture. All experiments prove that the efficiency of the optimal architecture searching algorithms is strongly dependent on the efficiency of the learning method, the result of which influences a quality index of a given architecture. If obtained set of trained network parameters is only a local optimal solution and far of the global one, the information about a quality of a considered network architecture can be falsified.

The last part of this book illustrates the applicability of EAs in the one of the most important domains of the modern industrial processes: the fault diagnosis. Presented examples prove that application of the EA technique, in particular the GP method, to design of FDI systems significantly improve their effectiveness, especially in the case when a non-linear dynamic system is diagnosed.

The following is a concise summary of the contribution provided by this work to the evolutionary computing theory and application:

- Proposes a new mechanism, called *trap test*, which allows to detect the end of the active phase of the evolutionary processing and to apply procedures, which accelerate the exploration ability of the ESSS algorithm.

- Presents a concept of a population–environment interaction in the form the erosion technique (ESSS-DOF). This procedure prevents a cyclically visiting of neighbouring peaks by a population.

- Introduces a new modified version of Gaussian mutation (applied in the ESSS-FDM algorithm) with the nonzero expectation vector, which is parallel to the latest population drift.
- Formulates and experimentally analyzes the geographically local selection operator.
- Investigates the influence of the surrounding and symmetry effects of the Gaussian and Cauchy mutations on the effectiveness of phenotype EAs. Proposes and analyzes modified version of Gaussian and Cauchy mutation deprived above effects.
- Systematizes problems of the adaptation in non-stationary environments taking into account both the problem specification and intensity of an environment changes criteria.
- Defines two new measures for algorithms processing in the non-stationary environments: the "acceptability" and "acceptability distance", which reward algorithms keeping the adaptation process on an acceptable level.
- Develops, basing on the ESSS-FDM algorithm, the *on-line* evolutionary learning method for the DMLP network.
- Proposes a cascade-reduction scheme for the genetic algorithm and simulated annealing approach, based on the direct representation, to the ANN architecture optimization.
- Extends the digraph representation of the space of MLP structures, proposed by Doering *et al.* (Doering *et al.* 1997), to the digraph representing the space of DMLP architectures.
- Investigates the applicability of the SA, $A^\star$ and Tabu Search algorithms to the DMLP architecture optimization through searching the digraph of DMLP structures.
- Systematizes existing application of the EAs in the FDI systems, and also proposes possible directions of this task development.
- Shows how to represent various model structures as a parameterized trees and how to identify their structure using the GP algorithm as well as to estimate their parameters.
- Presents the methodology of the input-output MIMO NARX and asymptotically stable state-space models design using above version of the GP approach.
- Basing on the fact that instrumental matrices strongly influence the convergence properties of the Extended Unknown Input Observer, proposes the GP approach to the these matrices selection problem.

A Saddle crossing problem

In this book the saddle crossing problem is defined as follows. Let us consider the sum of two Gaussian peaks

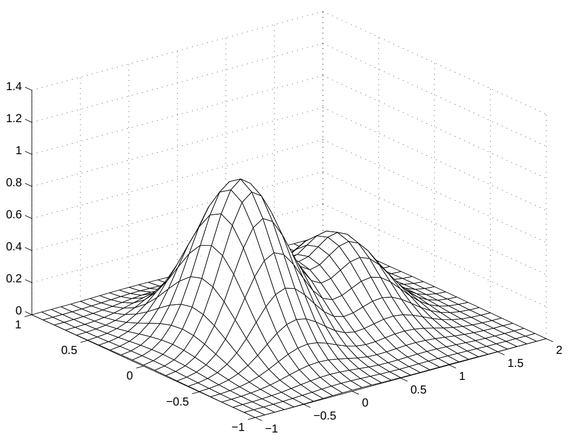


Fig. A.1 . Two dimensional version of the sum of two Gaussian peaks (A1)

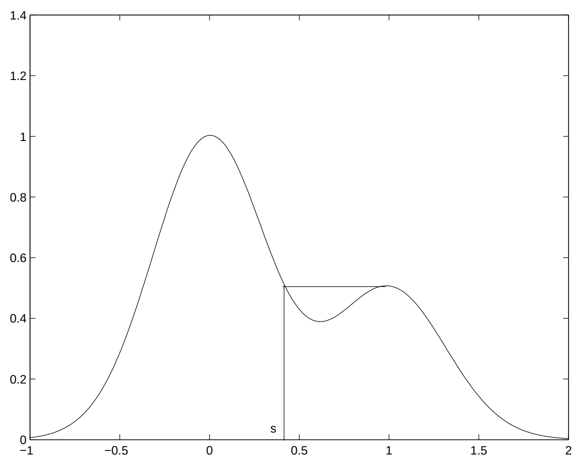


Fig. A.2 . The termination point of the saddle crossing problem

$$\Phi_{sc}(\boldsymbol{x}) = \exp\left(-5 \sum_{i=1}^{n} x_i^2 \right) + \frac{1}{2} \exp\left(-5\Big((1-x_1)^2 + \sum_{i=2}^{n} x_i^2 \Big) \right), \qquad \text{(A1)}$$

where n is the landscape dimension. The function $\Phi_{sc}(\boldsymbol{x})$ is composed of two Gaussian peaks. The lowest one possesses its optimum at the point $(1, 0, \ldots, 0)$. The global optimum is located at the point $(0, 0, \ldots, 0)$.

It was assumed that the saddle is crossed by the population if the mean value of the first entry taken over all elements

$$\langle x_1 \rangle = \frac{1}{n} \sum_{k=1}^{n} \left(\boldsymbol{x}_k \right)_1 < s \approx 0.42, \qquad \text{(A2)}$$

it means that most individuals are located on the higher peak (Fig. A.2).

The initial point of searching for the algorithm tested is chosen in the local optimum of the function $\Phi_{sc}(\boldsymbol{x})$. If the population did not crossed the saddle during a given processing time $t_{\max}$, then, in order to calculate ξ, the crossing time is fixed to $t_{\max}$.

B Benchmarks for the global optimization problem

Function f_1 — sum of two Gaussian peaks. (Fig. A.1)

$$f_1(x_1, x_2) = \exp(-x_1^2 - x_2^2) + \frac{1}{2}\exp(-(x_1 - 1)^2 - x_2^2), \tag{B1}$$

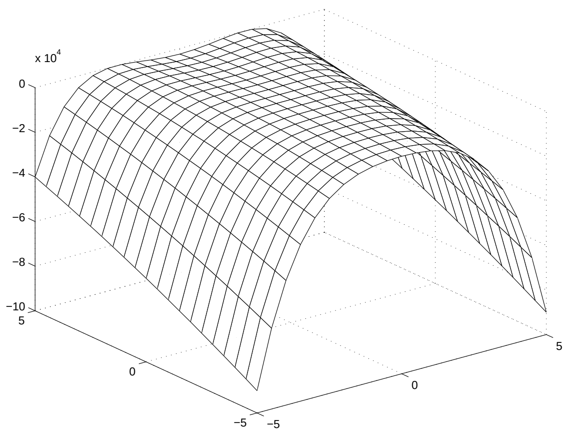


Fig. B.1 . De Jong's function F2: f_2 (B2)

Function f_2 — De Jong's function F2. (Fig. B.1)

$$f_2(x_1, x_2) = 3500 - 100(x_1^2 - x_2)^2 - (1 - x_1)^2, \tag{B2}$$

Function f_3 — De Jong's function F5. (Fig. B.2)

$$f_3(x_1, x_2) = 500 - \left\{0.002 + \sum_{j=1}^{25}\left[j + \sum_{i=1}^{2}(x_i - a_{ij})^6\right]^{-1}\right\}^{-1}, \tag{B3}$$

$$(a_{ij}) = \left(\begin{array}{cccccccccc} -32 & -16 & 0 & 16 & 32 & -32 & \dots & 0 & 16 & 32 \\ -32 & -32 & -32 & -32 & -32 & -16 & \dots & 32 & 32 & 32 \end{array}\right)$$

Function f_4 — the „drop wave" function. (Fig. B.3)

$$f_4(x_1, x_2) = \frac{1 + \cos\left(12\sqrt{x_1^2 + x_2^2}\right)}{\frac{1}{2}\left(x_1^2 + x_2^2\right) + 2}, \tag{B4}$$

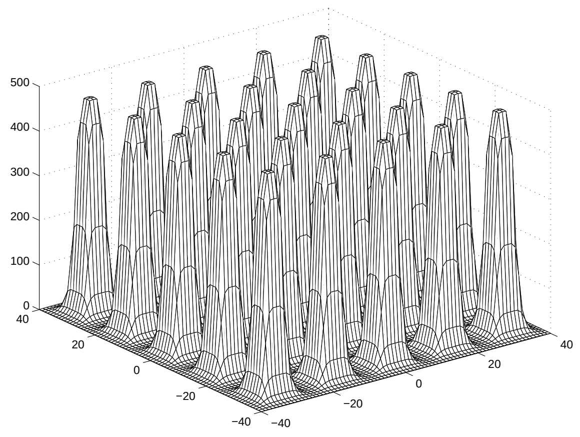


Fig. B.2 . De Jong's F5: f_3 (B3)

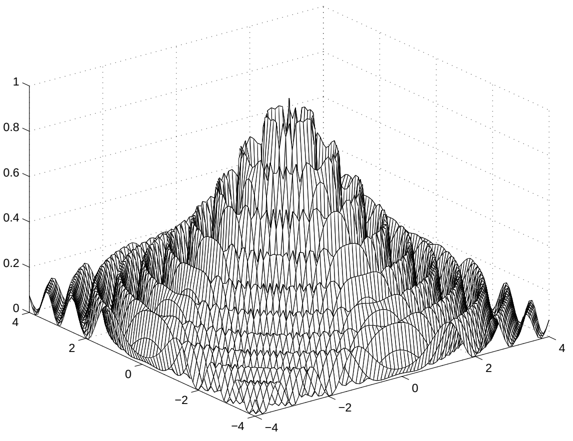


Fig. B.3 . The function f_4 (B4)

Function f_5 — Michalewicz's function. (Fig. B.4)

$$f_5(x_1, x_2) = \sin(x_1)\big(\sin(\frac{x_1^2}{\pi})\big)^{20} + \sin(x_2)\big(\sin(\frac{x_2^2}{\pi})\big)^{20}, \tag{B5}$$

Function f_6 — Shubert's function. (Fig. B.5)

$$f_6(x_1, x_2) = 200 + \sum_{i=1}^{5} i\cos\big((i+1)x_1 + 1\big)\sum_{i=1}^{5} i\cos\big((i+1)x_2 + 1\big), \tag{B6}$$

Function f_7 — Rastringin's function. (Fig. B.6)

$$f_7(x_1, x_2) = 100 - \left(x_1^2 + x_2^2\right) - 10\left(\cos(2\pi x_1) + \cos(2\pi x_2)\right), \tag{B7}$$

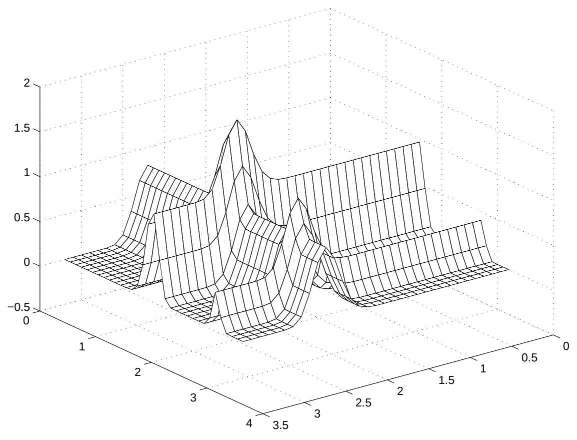


Fig. B.4 . Michalewicz's function f_5 (B5)

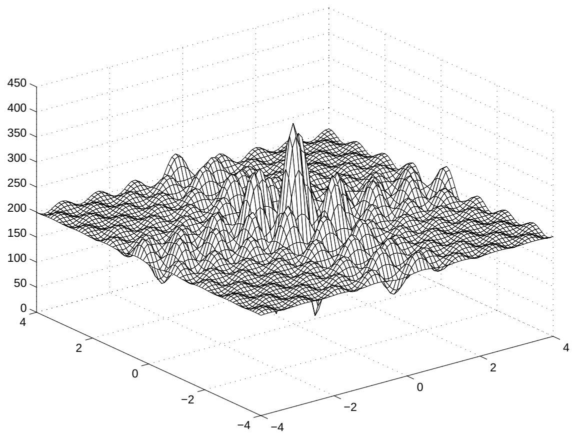


Fig. B.5 . Shubert's function f_6 (B6)

Function f_8 — Ackley's function. (Fig. B.7)

$$\begin{aligned} f_8(x_1, x_2) = & - e + 20 \exp\left(- \frac{1}{10}\sqrt{x_1^2 + x_2^2}\right) \\ & - \exp\left(\frac{\cos(2\pi x_1) + \cos(2\pi x_2)}{2}\right). \end{aligned} \tag{B8}$$

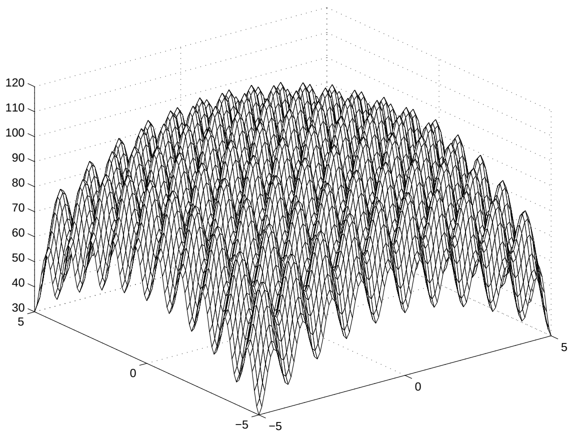


Fig. B.6 . Rastringin's function f_7 (B7)

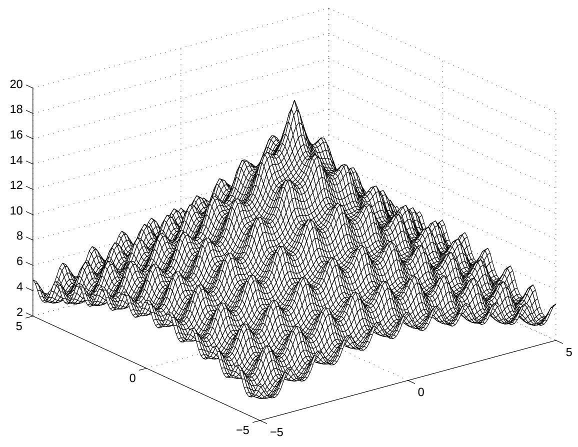


Fig. B.7 . Ackley's function f_8 (B8)

REFERENCES

Alippi, C., Petracca, R., and Piuri, V. (1995): *Off-line performance maximization in feedforward neural networks by applying virtual neurons and covariance transformations.* – In: *Circuits and Systems*, IEEE Int. Symp. ISCAS'95, Seattle WA, USA, pp.2197–2200.

Anderson, B.D.O., and Moore, J.B. (1979): *Optimal Filtering.* – Prentice-Hall, New Jersey.

Angeline, P. (1997): *Tracking extrema in dynamic environments.* – In: *Evolutionary Programming*, 6th Int.Conf.- EP'97, vol.1213 in LNCS, – Springer, NY, pp.335-346.

Angeline, P., and Kinnear, K.E. (1996): *Advances in Genetic Programming.* – MIT Press.

Arabas, J. (2001): *Lecturers on Evolutionary Algorithms.* – WNT, Warsaw, (in Polish).

Arabas, J., Michalewicz, Z., and Mulawka, J. (1994): *GAVaPS - a genetic algorithm with varying population size.* – In: *Evolutionary Computation*, 1st IEEE Int.Conf., Orlando, USA, Vol.2, pp.73–78.

Ash, T. (1989): *Dynamic node creation.* – Connection Sci., Vol.1, No.4, pp.365–375.

Atmar, W. (1992): *On the rules and nature of simulated evolutionary programming.* – In: D.B. Fogel and W. Atmar (Eds.), Proc. 1st Annual Conf. on *Evolutionary Programming* – Evolutionary Programming Society, LA Jolla, CA, pp.17–26.

Ayoubi, M. (1994): *Fault diagnosis with dynamic neural structure and application to turbo–charger.* – In: *Fault Detection Supervision and Safety for Technical Processes*, Int. Symp. SAFEPROCESS'94, Espoo, Finland, Vol.2, pp.618–623.

Bäck, T. (1995): *Evolutionary Algorithms in Theory and Practice.* – Oxford University Press.

Bäck, T. (1998): *On the behaviour of evolutionary algorithms in dynamic environments.* – In: *Evolutionary Computation*, Int.Conf. ICEC'98, – IEEE Publishing, pp.446-451.

Bäck, T., Hoffmeister, F., and Schwefel H.-P., (1991): *A survey of evolution strategies.* – In: R. Belew and L. Booker (Eds.), Proc. 4th Inter. Conf. on *Genetic Algorithms*, – Morgan Kauffmann Publishers, Los Altos, CA.

Bäck, T., Fogel, D.B., and Michalewicz, Z. (Eds.) (1997): *Handbook of Evolutionary Computation.* – Institute of Physics Publishing and Oxford University Press, NY.

Bäck, T., Schutz, M. (1996): *Intelligent mutation rate control in canonical genetic algorithm.* — ISMIS'96, vol.1079 in LNAI, – Berlin: Springer, pp. 158-167.

Bäck, T., and Schwefel, H.-P. (1993): *An overview of evolutionary algorithms for parameter optimization.* – Evolutionary Computation, Vol.1, No.1, pp.1–23.

Baldi, P. (1995): *Gradient descent learning algorithm overview. A general dynamical systems perspective.* – IEEE Trans. Neural Networks, Vol.6, No.1, pp.182–195.

Beasley, D., Bull, D.R., and Martin, R.R. (1993): *A sequential niche technique for multimodal function optimization.* – Evolutionary Computation, Vol.1, No.2, pp.101–125.

Beasley, D., Bull, D.R., and Martin, R.R. (1993a): *An overview of gentic algorithms: part 1, fundamentals.* – University Computing, Vol.15, No.2, pp.58–69.

Beasley, D., Bull, D.R., and Martin, R.R. (1993b): *An overview of gentic algorithms: part 2, research topics.* – University Computing, Vol.15, No.4, pp.170–181.

Birge, J., and Louveaux, F. (1997): *Introduction to Stochastic Programming.* – Springer–Verlag, New York.

Blanke, M., Bogh, S., Jorgenson, R.B., and Patton, R.J. (1994): *Fault detection dor diesel engine actuator – a benchmark for FDI.* – In: *Fault Detection Supervision and Safety for Technical Processes*, Int. Symp. SAFEPROCESS'94, Espoo, Finland, Vol.2, pp.498–506.

Bornholdt, S., and Graudenz, D. (1991): *General assymetric neural networks and-structure design by genetic algorithms.* – Deutsches Elektronen-Synchotron 91–046, May 1991.

Boutayeb, M., and Aubry, D. (1999): *A strong tracking extended Kalman observer for nonlinear discrete-time systems.* – IEEE Trans. Automat. Contr., Vol.44, No.8, pp.1550-1556.

Branke, J. (1999): *Memory enhanced evolutionary algorithm for changing optimisation problems.* – In: *Evolutionary Computation*, IEEE Congres CEC'99, – IEEE Publishing, pp.1875-1882.

Bryson, A.E., and Ho, C. (1975): *Applied Optimal Control.* – A halsted Press Book, New York.

Calado, J.M.F., Korbicz, J., Patan, K., Patton, R.J., and Sa da Costa, J. (2001): *Soft computing approaches to fault diagnosis for dynamic systems.* – European Journal of Control, Vol.7, pp.248–286.

Campolucci, P., Uncini A., Piazza, F., and Rao, B.D. (1999): *On-line learning algorithms for locally recurrent neural networks.* – IEEE Trans. Neural Networks, Vol.10, pp.253–271.

Carse, B., Fogarty, T.C., and Munro, A. (1996): *Envolving fuzzy rule based controllers using genetic algorithms.* – Fuzzy Sets and Systems, Vol.80, No.3, pp.273–293.

Cedeno, W., and Vemuri, V.R. (1997): *On the use of niching for dynamic landscapes.* – In: *Evolutionary Computation*, Int.Conf. ICEC'97, – IEEE Publishing, pp.361–366.

Chan, L.W., and Fallside, F. (1987): *An adaptive training algorithm for backpropagation networks.* – Computer Speech and Language, Vol.2, pp.205–218.

Chauvin, Y. (1989): *A back-propagation algorithm with optimal use of hidden units.* – In: D.S. Touretzky (Ed.) *Advances in Neural Information Processing Systems 1*, – Morgan Kaufmann, San Mateo, CA, pp.519–526.

Chen, J., and Patton, R.J. (1999): *Robust Model Based Fault Diagnosis for Dynamic Systems.* – Kluwer Academic Publishers.

Chen, J., Patton, R.J., and Lui, G.P. (1996): *Optimal residual design for fault diagnosis using multiobjective optimization and genetic algorithms.* – Int. J. Syst. Sci., Vol.27, No.6, pp.567–576.

Cholewa, W. (2002): *Advisory systems in technical diagnosis.* – In: J. Korbicz, J.M. Kościelny, Z. Kowalczuk, and W. Cholewa (Eds.) *Process Diagnosis.* WNT, Warszawa, pp. 543–580, (in Polish).

Cobb., H.G., and Grefenstette, J.J., (1993): *Genetic Algorithms for Tracking Changing Environments.* – In: *Genetic Algorithms*, 5th Int.Conf. ICGA'93 – Morgan Kauffman, pp.523–530.

Cybenko, G. (1989): *Approximation by superpositions of a sigmoidal function.* – Mathematics of Control, Signals, and Systems, Vol.2, pp.303–314.

Dasgupta, D., and McGregor, D.R. (1992): *Non–stationary function optimisation using the structured genetic algorithm.* – In: *Parallel Problem Solving from Nature* – Elsevier Science Publishers B.V., pp.145–154.

Dasgupta, D., and Michalewicz, Z. (Eds.) (1997): *Evolutionary Algorithms for Engineering Applications.* – Springer–Verlag.

Davis, L. (Ed.) (1987): *Genetic Algorithms and Simulated Annealing.* – Morgan Kaufmann.

Davis, L. (Ed.) (1991): *Handbook of Genetic Algorithms.* – Van Nostrand Reinhold.

De Jong, K. (1975): *An analysis of the behaviour of a class of genetic adaptive systems.* – Ph.D. thesis, University of Michigan, Ann Arbor, MI.

Demuth, H., and Beale, M. (1993): *Neural Network Toolbox for Use with MATLAB.* – The MathWorks Inc.

Doering, A., Galicki, M., and Witte, H. (1997): *Structure optimization of neural networks with the $A^{\star}$–algorithm.* – IEEE Trans. Neural Networks, Vol.8, No.6, pp.1434–1445.

Draye, J.P.S., Pavisic, D.A., Cheron, G.A., and Libert, G.A. (1996): *Dynamic recurrent neural networks: a dynamical analysis.* – IEEE Trans. Systems, Man, and Cybernetics, Part B: Cybernetics, Vol.26, pp.692–706.

Edelmayer, A. (Ed.) (2000): *Research of Quantitative and Qualitative FDI Methods Based on Data From Lublin Sugar Factory. Special Session of the IFAC Symp. SAFEPROCESS'2000.* – Prep. IFAC Symp. *Fault Detection, Supervision and Safety of Technical Processes: SAFEPROCESS'2000*, June 14-16, Budapest, Hungary, pp. 331-358.

Eiben, A.E., Hinterding, R., and Michalewicz, Z. (1999): *Parameter control in evolutionary algorithms.* – IEEE Trans. on Evolutionary Computation, Vol.3, No.2, pp.124-141.

Eldredge, N., and Gould, S.J. (1972): *Punctuated equlibria: an alternative to phylogenetics gradualism.* – In: T.J.M. Schopt (Ed.), *Models in Paleobiology* – San Francisco Freeman.

Elman, J.L. (1990): *Finding structure in time.* – Cognitive Science, Vol.14, pp.179–211.

Esparcia-Alcazar, A.I. (1998): *Genetic Programming for Adaptive Digital Signal Processing.* – Ph.D. dissertation. Glasgow University, U.K.

Fang, K.-T., Kotz, S., and Ng, K.-W. (1990): *Symmetric Multivariate and Related Distributions.* – Chapman and Hall, London.

Fahlman, S.E. (1988): *Fast–learning variations on backpropagation.* – In: Proc. *Connectionist Models Summer Scholl* – Morgan Kaufmann, Los Atos.

Fahlman, S.E., and Lebierre, C. (1990): *The cascade-correlation learning architecture.* – In: D.S. Touretzky (Ed.) *Advances in Neural Information Processing Systems 2* – Morgan Kaufmann, San Mateo CA, pp.524–532.

Fasconi, P., Gori, M., and Soda, G. (1992): *Local feedback multilayered networks.* – Neural Computation, Vol.4, pp.120–130.

Fathi, Z., Ramirez, W.F., and Korbicz, J. (1992): *Fault detection in coal fired power plants using non–linear filtering.* – Applied Mathematics and Computer Science, Vol.2, No.1, pp.87–118.

Fausett, L. (1994): *Fundamentals of Neural Networks.* – Prentice Hall, Englewood Cliffs, N.J.

Feng, W., Brune, T., Chan, L., Chowdhury, M., Kuek, C.K., Li, Y. (1998): *Benchmarks for testing evolutionary algorithms.* – In: *Measurement & Control*, 3rd Asia-Pacific Conf., Dunhuang, China.

Fogel, D.B., Fogel, L.J., and Atmar, J.W. (1991): *Meta-evolutionary programming.* – In: *Signals, Systems & Computers*, 25th Asilomar Conf., – Maple Press, San Jose, CA, pp.540–545.

Fogel, D.B. (1992): *An analysis of evolutionary programming.* – In: *Evolutionary Programming*, 1st Annual Conf. – Evolutionary Programming Society, LA Jolla, CA, pp.43–51.

Fogel, D.B. (1994): *An introduction to simulated evolutionary computation.* – IEEE Trans. Neural Networks, Vol.5, pp.3–14.

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

Fogel, D.B. (1998): *Evolutionary Computation: The Fossil Record.* – IEEE Press, NY.

Fogel, D.B. (1999): *An overview of evolutionary programming.* – In: L.D. Davis, K. De Jong, M.D. Vose, and L.D. Whitley (Eds.) *Evolutionary Algorithms* – Springer–Verlag, Heidelberg, pp.89–109.

Fogel, L.J., Owens, A.J., and Walsh, M.J. (1966): *Artificial Intelligence through Simulated Evolution.* – Wiley, NY.

Frank, P.M. (1998): *AI methods in fault diagnosis.* – In: *System–Modelling–Control*, Int. Conf., Zakopane, Poland, (published on CD-ROM).

Frank, P.M., and Köppen-Seliger, B. (1997): *New developments using AI in fault diagnosis.* – Artificial Intelligence, Vol.10, No.1, pp.3–14.

Frean, M. (1990): *The upstart algorithm: a method for constructing and training feedforward neural networks.* – Neural Computation, Vol.2, pp.198–209.

Galar, R. (1985): *Handicapped individua in evolutionary processes.* – Biological Cybernetics, Vol.51, pp.1–9.

Galar, R. (1989): *Evolutionary search with soft selection.* – Biological Cybernetics, Vol.60, pp.357–364.

Galar, R. (1990): *Soft Selection in Random Global Adaptation in R^n. A Biocybernetic Model of Development.* – Technical University of Wrocław Press, Wrocław (in Polish).

Galar, R., and Karcz-Dulęba, I. (1994) *The evolution of two: an example of space of states approach.* – In: *Evolutionary Programming*, 3rd Annual Conference – World Scientific, San Diego CA, pp.261–268.

Galar, R., and Kopciuch, R. (1999): *Impatience and polarisation in evolutionary processes.* – In: *Evolutionary Algorithms and Global Optimization*, 3nd Conference at Potok Złoty, Poland – Warsaw University of Technology Press, Warsaw, pp.115–122.

Geman, S., Bienenstack, E., and Doursat, R. (1992): *Neural networks and the bias/variance dilemma.* – Neural Computation, Vol.4, No.1, pp.1–58.

Ghosh, A., Tsutsui, S., Tanaka, H. (1998): *Function optimisation in nonstationary environment using steady–state genetic algorithms with aging of individuals.* – In: *Evolutionary Computation*, 5th IEEE Int. Conf. – IEEE Publishing, pp.666–671.

Glover, F. (1986): *Future Paths for Integer Programming and Links to Artificial Intelligence.* – Computers and Operations Research, Vol.5.

Glover, F., and Laguna, M. (1997): *Tabu Search.* – Kluwer Academic Publishers.

Goldberg, D.E. (1989): *Genetic Algorithms in Search, Optimization and Machine Learning.* – Addison–Wesley, Reading, MA.

Goldberg, D.E., and Smith, R.E. (1987): *Nonstationary function optimisation using genetic algorithms with dominance and diploidy.* – In: *Genetic Algorithms*, Int.Conf. – Lawrence Erlbaum Associates, pp.59–68.

Gray, G.J, Murray-Smith, D.J., Li, Y., Sharman, K.C., and Weinbrenner, T. (1998): *Nonlinear model structure identification using genetic programming.* – Control Eng. Practice, Vol.6. pp.1341-1352.

Grefenstette, J.J. (1986): *Optimization od control parameters for genetic algorithms.* – IEEE Trans. on System, Man and Cybernetics, Vol.16, No.1, pp.122–128.

Grefenstette, J.J. (1990): *Genetic algorithms and their applications.* – In: A. Kent, and J.G. Williams (Eds) *Encyclopedia of Computer Science and Technology* – Marcel Dekker, pp.139–152.

Grefenstette, J.J. (1992): *Genetic algorithms for changing environments.* In: *Parallel Problem Solving from Nature* – Elsevier Science Publishers B. V., pp.137-144.

Grefenstette, J.J. (1993): *Deception considered harmful.* – In: G. Rawlings (Ed.) *Foundations of Genetic Algorithms* – Morgan Kauffman.

Grefenstette, J.J. (1999) *Evolvability in dynamic fitness landscapes: a genetic algorithm approach.* – In: *Evolutionary Computation*, IEEE Congress CEC'99 – IEEE Press, pp.2031-2038.

Gupta, M.M., and Rao, D.H. (1993): *Dynamic neural units with application to the control of unknown nonlinear systems.* – Journal of Intelligent and Fuzzy Systems, Vol.1, pp.73–92.

Gutowski, M. (2001): *Levy flights as an underlying mechanism for global optimization algorithm.* – In: *Evolutionary Algorithms and Global Optimization*, 5th Conf., Jastrzębia Góra, Poland – Warsaw University of Technology Press, pp.79–86.

Hagan, M.T., and Menhaj, M.B. (1994): *Training feedforward networks with the Marquardt algorithm.* – IEEE Trans. Neural Networks, Vol.5, pp.989–993.

Harp, S.A., Samad, T., and Guha, A, (1989): *Designing application-specific neural networks using genetic algorithms.* – In: D.S. Touretzky (Ed.) *Advances in Neural Information Processing Systems 2* – Morgan Kaufmann, San Mateo CA, pp.447–454.

Harp, S.A., and Samad, T. (1992): *Genetic optimization of self-organizing feature maps.* – In: *Neural Networks*, Int. Joint Conf. IJCNN'91, Seattle, USA.

Hart, P.E., Nilsson, N.J., and Raphael, B. (1968) *A formal basis for the heuristic determination of a minimum cost paths.* – IEEE Trans. Systems, Man, Cybernetics, Vol. 4.

Hassibi, B., and Stork, D. (1993): *Second order derivaties for network prunning: optimal brain surgeon.* – In: D.S. Touretzky (Ed.) *Advances in Neural Information Processing Systems 5* – Morgan Kaufmann, San Mateo CA, pp.164–171.

Hertz, J., Krogh, A., and Palmeer, R.G. (1991): *Introduction to the Theory of Neural Computation.* – Addison-Wesley Publishin COmpany, Inc, Reading.

Himmelblau, D.M. (1992): *Use of artificial neural networks in monitor faults and for troubleshooting in the process industries.* – In: *On–line Fault Detection and Supervision in the Chemical Process Industries*, IFAC Symp., Newark, Delaware, USA, pp.144–149.

Hirose, Y., Yamashita, K., and Hijiya, S. (1991): *Back-propagation algorithm which varies the number of hidden units.* – Neural Networks, Vol.4, No.1, pp.61–66.

Holland, J.H. (1975): *Adaptation in natural and artificial systems.* – The University of Michigan Press, Ann Arbor, MI.

Holland, J.H. (1992): *Adaptation in Natural and Artificial Systems.* – MIT Press, Cambridge, MA.

Hornik, K., Stinchcombe, M., and White, H. (1989): *Multilayer feedforward networks are universal approximators.* – Neural Networks, Vol.2, pp.359-366.

Hunt, K.J., Sbarbaro, D., Zbikowski, R., and Gathrop, P.J. (1992): *Neural networks for control systems — a survey.* – Automatica, Vol.28, pp.1083–1112.

Jacobs, R.A. (1988): *Increased rates of convergence through learning rate adaptation.* – Neural Networks, Vol.1, pp.295–307.

Kappler, C. (1996): *Are evolutionary algorithms improved by large mutations?* – In: H.-M. Voigt, W. Ebeling, I. Rechenberg and H.-P. Schwefel (Eds.) *Parallel Problem Solving from Nature (PPSN) IV*,(Lecture Notes in Computer Science, vol. 1141) – Springer–Verlag, Berlin, pp.388–397.

Karcz-Dulęba, I. (1992): *Simulation of evolutionary processes as a tool of global optimization in $\mathbb{R}^n$.* – Ph.D. Thesis, Technical University of Wrocław, Wrocław (in Polish).

Karcz-Dulęba, I. (1997): *Some convergence aspects of evolutionary search with soft selection method.* – In: *Evolutionary Algorithms and Global Optimization*, 2nd Conference at Rytro, Poland – Warsaw University of Technology Press, Warsaw, pp.113–120.

Karcz-Dulęba, I. (2001): *Evolution of two-element population in the space of population states: equilibrium states for assymetrical fitness functions.* – In: *Evolutionary Algorithms and Global Optimization*, 5th Conference at Jastrzębia Góra, Poland – Warsaw University of Technology Press, Warsaw, pp.106–113.

Karcz-Dulęba, I. (2001): *Dynamics of infinite populations envolving in a landscape of uni- and bimodal fitness functions.* – IEEE Trans. Evolutionary Computation, Vol.5, No.4, pp.398–409.

Karnin, E.D. (1990): *A simple procedure for prunning back-propagation trained neural networks.* – IEEE Trans. Neural Networks, Vol.1, pp.239–242.

Kinnear, J.R., (Ed) (1994): *Advances in Genetic Programming.* – The MIT Press, Cambridge, MA.

Kirkpatrick, S., Gellat, C.D., and Vecchi, M.P. (1983): *Optimization by simulated annealing.* – Science, Vol.220, pp.671–680.

Kitano, H. (1990): *Designing neural networks using genetic algorithms with graph generation system.* – Complex System, Vol.4, pp.461–486.

Koivo, M.H. (1994): *Artificial neural networks in fault diagnosis and control.* – Control Engineering and Practice, Vol.2, No.7, pp.89–101.

Köppen-Seliger, B., and Frank, P.M. (1999): *Fuzzy logic and neural networks in fault detection.* – In: L. Jain, and N. Martin (Eds.) *Fusion of Neural networks, Fuzzy Sets, and Genetic Algorithms* – CRC Press, New York, pp.169–209.

Korbicz, J. (1997): *Neural networks and thier application in fault detection and diagnosis.* – In: *Fault Detection, Supervision and Safety of Technical Processes*, IFAC Symp. SAFEPROCESS'97, Hull, UK, Vol.1, pp.377–382.

Korbicz, J., and Bidyuk, P.I. (1993): *State and Parameter Estimation. Digital and Optimal Filtering, Applications.* – Technical University of Zielona Góra Press, Zielona Góra.

Korbicz, J., Kościelny, J.M., Kowalczuk, Z., and Cholewa, W. (Eds.) (2002): *Process Diagnosis.* WNT, Warszawa, (in Polish).

Korbicz, J., Obuchowicz, A., Patan, K. (1998): *Network of dynamic neurons in Fault Detection Systems.* – In: *System, Man, Cybernetics*, IEEE Int.Conf., San Diego, USA, IEEE Publishing, pp.1862–1867.

Korbicz, J., Obuchowicz, A., and Uciński, D. (1994): *Artificial Neural Networks. Foundations and Applications.* – Academic Publishing Hause, Warszawa, (in Polish).

Korbicz, J., Patan, K., and Obuchowicz, A. (1999): *Dynamic neural networks for process modelling in fault detection and isolation systems.* – Int. Journ. Applied Mathematics and Computer Science, Vol.9, No.3, pp.510–546.

Korbicz, J., Patan K., and Obuchowicz, A. (2001): *Neural network fault detection system for dynamic processes.* – Bulletin of the Polish Academy of Sciences, Technical Sciences, Vol.49, No.2, pp.301–321.

Kosiński, W., Michalewicz, Z., Weigl, M., and Koleśnik, R. (1998): *Genetic algorithms for preprocessing of data for universal approximators.* – In: M. Kłopotek, M. Michalewicz, and Z.W. Raś (Eds.) *Intelligent Information Systems*, 7th International Symposium at Malbork, Poland – Polish Academy of Sciences Press, Warsaw, pp.320–331.

Kościelny, J.M., Sędziak, D., and Zakroczymski, K. (1999a): *Fuzzy logic fault isolation in large scale systems.* – Int. Journ. Applied Mathematics and Computer Science, Vol.9, No.3, pp.637–652.

Kościelny, J.M., Syfert, M., and Bartyś, M. (1999b): *Fuzzy logic fault diagnosis of industrial process actuators.* – Int. Journ. Applied Mathematics and Computer Science, Vol.9, No.3, pp.653–666.

Kowal, M., and Korbicz, J. (2000): *Neuro-fuzzy detector for industrial process.* – In: R. Mampel, M. Wagenknecht, and M. Chacker (Eds) *Fuzzy Control Theory and Practice* – Physice–Verlag, Heidelberg.

Koza, J.R. (1992): *Genetic Programming: On the Programming of Computers by Means of Natural Selection.* – The MIT Press, Cambridge, MA.

Koza, J.R., and Rice, J.P. (1991): *Genetic generation of both the weights and architecture for a neural network.* – In: *Neural Networks*, Int. Joint Conf. IJCNN'91, Sattle, USA, Vol.2, pp.397–404.

Kwaśnicka, H., and Szerszon, P (1997): *NetGen – evolutionary algorithms in designing artificial neural networks.* – In: *Neural Networks and Their Applications*, 3rd Conf., Kule, Poland, pp.671–676.

LeCun, Y., Denker, J., and Solla, S. (1990): *Optimal brain damage.* – In: D.S. Touretzky (Ed.) *Advances in Neural Information Processing Systems 2* – Morgan Kaufmann, San Mateo CA, pp.598–605.

Leontaritis, I.J., and Bilings, S.A. (1985): *Input-output parametric models for nonlinear systems.* – Int. J. Control, Vol.41, pp.303–344.

Lewis, J., Hart, E., Ritche, G., (1998): *A comparison of dominance mechanisms and simple mutation on non–stationary problems.* – In: *Parallel Problem Solving from Nature*, Vol.1498 in LNCS – Springer, pp.139-148.

Looney, C.G. (1997): *Pattern Recognition Using Neural Networks.* – Oxford University Press.

Makuch, M., Patan K., and Obuchowicz, A. (1996): *Evolutionary search with soft selection in artificial neural network learning process.* – In: *Evolutionary Algorithms*, 1st Conference at Murzasichle, Poland – Warsaw University of Technology Press, Warsaw, pp.87–93, (in Polish).

Mantegna, R.N., and Stanley, H.E. (1994): *Stochastic Process with Ultraslow Convergence to a Gaussian: The truncated Lévy Flights.* – Phys. Rev. Lett., Vol.73, No.22, pp.2946–2947.

Marciniak, A., and Korbicz, J. (1999): *Diagnosis of the dynamical non–linear systems using static neural networks.* – In: *Automatics*, XII National Conf., Opole, Poland, pp.185–190, (in Polish).

Marshall, S.J., and Harrison, R.F. (1991): *Optimization and training of feedforward neural networks by genetic algorithms.* – In: *Artificial Neural Networks*, Int. Conf., Boumemouth, UK, pp.39–42.

McCulloch, W.S., and Pitts, W. (1943): *A logical calculus of ideas immanent in nervous activity.* – Bull. Math. Biophysis, Vol.5, pp.115–133

Mezard, M., and Nadal, J.-P. (1989): *Learning feedforward layered networks: the tiling algorithm.* – J. Phys. A, Vol.22, pp.2191–2204.

Michalewicz, Z. (1996): *Genetic Algorithms + Data Structures = Evolution Programs.* – Springer–Verlag, Heidelberg.

Michalewicz, Z. (1999): *The significance of the evaluation function in evolutionary algorithms.* – In: L.D. Davis, K. De Jong, M.D. Vose, and L.D. Whitley (Eds.) *Evolutionary Algorithms* – Springer–Verlag, Heidelberg, pp.151–166.

Miller, W.T., Sutton, R.S., and Werbos, P.J. (1990): *Neural Networks for Control.* – MIT Press, Cambridge, MA.

Miller, G.,Todd, P., and Hedge, S. (1989): *Desinging neural networks using genetic algorithms.* – In: *Genetic Algorithms*, Int. Conf., pp.379–384.

Mitchel, M. (1996): *An Introduction to Genetic Algorithms.* – MIT Press, Cambridge, MA.

Montana, D.J., and Davis, L. (1989): *Training feedforward neural networks using genetic algorithms.* – BNN Systems and Technologies, Cambridge, Mass., USA.

Mori, N., Imanishi, S., Kita, H., Nishikawa, Y. (1997): *Adaptation to a changing environments by mMeans of the memory based thermodynamical genetic algorithm.* – In: *Genetic Algorithms*, 7th IEEE Int.Conf. – Morgan Kauffman, pp.299-306.

Mori, N., Kita, H., Nishikawa, Y. (1996): *Adaptation to a changing environments by means of the thermodynamical genetic algorithm.* – In: *Parallel Problem Solving from Nature*, vol. 1141 in LNCS – Springer, pp.513-522.

Mori, N., Kita, H., Nishikawa, Y. (1998): *Adaptation to changing nvironments by means of the feedback thermodynamical genetic algorithm.* – In: *Parallel Problem Solving from Nature*, vol. 1498 in LNCS – Springer, pp.149-157.

Mothes, J. (1967): *Incertitudes et décisions industrielles* – Dunod.

Mozer, M., and Smolensky, P. (1989): *Skeletonization - a technique for trimming the fat from a network via relevance assessment.* – In: D.S. Touretzky (Ed.) *Advances in Neural Information Processing Systems 1* – Morgan Kaufmann, San Mateo CA, pp.107–115.

Müller, H.,J. (1964) *The relation of recombinational to mutational variance.* – Mutat. Res. Vol.1, pp.2–9.

Muselli, M., and Ridella, S. (1991): *Supervised learning using a genetic algorithm.* – In: *Neural Networks*, Int. Conf., Paris, France, Vol.2, pp.790.

Nagao, T., Agui, T., and Nagahashi, H. (1993): *Structural evolution of neural networks having arbitrary connections by a genetic method.* – IEICE Trans. Inf. & Syst., Vol.E76–D, No.6, pp.689–697.

Narendra, K.S., and Parthasarathy, K. (1990): *Identification and control of dynamical systems using neural networks.* – IEEE Trans. Neural Networks, Vol.1, p.12–18.

Naudts, B., and Kallel, L. (2000): *A cmparison of predictive measures of problem difficulty in evolutionary algorithms.* – IEEE Transactions on Evolutionary Computation, Vol.4, No.1, pp.1-15.

Ng, K.P., and Wong, K.C. (1995): *A new diploid scheme and dominance change mechanism for non–stationary function optimisation.* – In: *Genetic Algorithms*, 6th Int.Conf. – Morgan Kauffman, pp.159-166.

Nilsson, N.J. (1980): *Principles of Artificial Intelligence.* – Springer-Verlag, New York.

Obuchowicz, A. (1997) *Evolutionary search with soft selection and deterioration of the objective function.* – In: M. Kłopotek, M. Michalewicz, and Z.W. Raś (Eds.) *Intelligent Information Systems*, 6th Int. Symp., Zakopane, Poland – Polish Academy of Sciences Press, Warsaw, pp.288–295.

Obuchowicz, A. (1998): *Optimization of the artificial neural networks structure.* – In: *Computer Aided Scientific Research*, 5th Conf., Polanica Zdrój, Poland, pp.451–456, (in Polish).

Obuchowicz, A. (1999a): *Evolutionary search with soft selection in training a network of dynamic neurons.* – In: M. Kłopotek, and M. Michalewicz, (Eds.) *Intelligent Information Systems*, 6th Int. Symp., Ustroń, Poland – Polish Academy of Sciences Press, Warsaw, pp.214–223.

Obuchowicz, A. (1999b): *Adaptation in a time-varying landscape using an evolutionary search with soft selection.* – In: *Evolutionary Algorithms and Global Optimization*, 3nd Conference, Potok Złoty, Poland – Warsaw University of Technology Press, Warsaw, pp.245–251.

Obuchowicz, A. (1999c) *Architecture optimization of a network of dynamic neurons using the $A^\star$ algorithm.* – In: *Intelligent Techniques and Soft Computing*, 7th European Congress EUFIT'99, Aachen, Germany, (published on CD-ROM).

Obuchowicz, A. (1999d) *Optimization of a neural network structure with the cascade-reduction method.* – In: *Neural Networks and Their Applications*, 4th Int. Conf., Zakopane, Poland, pp.437–442.

Obuchowicz, A. (2000a): *Optimization of neural networks architectutes.* – In: W. Duch, J. Korbicz, L. Rutkowski, and R. Tadeusiewicz, (Eds.) *Biocybernetics and Biomedical Engineering 2000. Neural Networks* – Academic Publishing House EXIT, Warsaw, Poland, pp.323–368, in Polish.

Obuchowicz, A. (2000b): *Evolutionary search with erosion of quality peaks.* – In: M. Kłopotek, M. Michalewicz, and S.T. Wierzchoń (Eds.) *Intelligent Information Systems* – Physica–Verlag, Heidelberg, pp.257–265.

Obuchowicz, A. (2001a) *The true nature of multi-dimensional Gaussian mutation.* – In: Kurkova V., Steele N.C., Neruda R., Karny M. (Eds.) *Artificial Neural Networks and Genetic Algorithms* – Springer–Verlag, Wien, pp.248–251.

Obuchowicz, A. (2001b) *Mutli–dimensional Gaussian and Cauchy mutations.* – In: M. Kłopotek, M. Michalewicz, and S.T. Wierzchoń (Eds.) *Intelligent Information Systems 2001* – Physica–Verlag, Heidelberg, pp.133–142.

Obuchowicz, A. (2002a): *Local selection in evolutionary search.* – In: J. Arabas (Ed.) *Evolutionary Computation and Global Optimization* – Warsaw University of Technology Press, Warsaw, pp.47–62.

Obuchowicz, A. (2003a): *Population in an evolutionary trap: simulation of natural exploration.* – Bulletin of the Polish Academy of Sciences, Technical Sciences, Vol.??, No.?, pp.???–???.

Obuchowicz, A. (2003b): *Multi–dimensional mutations in evolutionary algorithms based on real-valued representation.* – Int. J. System Science. (submitted)

Obuchowicz, A., and Korbicz, J. (1998): *Evolutionary search with soft selection and forced direction of mutation.* – In: M. Kłopotek, M. Michalewicz, and Z.W. Raś (Eds.) *Intelligent Information Systems*, 7th International Symposium at Malbork, Poland – Polish Academy of Sciences Press, Warsaw, pp.300–309.

Obuchowicz, A., and Korbicz, J. (1999): *Evolutionary search with soft selection in parameter optimization.* – In: R. Wyrzykowski, B. Mochnacki, H. Piech, and J. Szopa (Eds.) *Parallel Processing and Applied Mathematics PPAM'99*, 3rd Int. Conf., Kazimierz Dolny, Poland – Technical University of Częstochowa Press, Częstochowa, pp.578–586.

Obuchowicz, A., and Korbicz, J. (2002): *Evolutionary methods in diagnosis systems design.* – In: J. Korbicz, J.M. Kościelny, Z. Kowalczuk, and W. Cholewa (Eds.) *Process Diagnosis.* – WNT, Warszawa, pp. 279–309, (in Polish).

Obuchowicz, A., and Patan, K. (1997a): *About some modifications of evolutionary search with soft selection algorithm.* – In: *Evolutionary Algorithms and Global Optimization*, 2nd Conf., Rytro, Poland – Warsaw University of Technology Press, Warsaw, pp.193–200.

Obuchowicz, A., and Patan, K. (1997b) *An algorithm of evolutionary search with soft selection for training multilayer feedforward neural networks.* – In: *Neural Networks and Their Application*, 3rd Conf., Kule, Poland – Technical University of Częstochowa Press, Częstochowa, pp.123–128.

Obuchowicz, A., and Patan, K. (1998): *Network of dynamic neurons as a residual generator. The architecture optimization.* –In: *Diagnostics of Industrial Processes*, 3rd Conf., Gdańsk/Jurata, Poland – Technical University of Gdańsk Press, pp.101–106.

Obuchowicz, A., and Patan, K. (2003): *Heuristic Search for Optimal Architecture of a Locally Reccurent Neural Network.* – In: M. Kłopotek, and S.T. Wierzchoń (Eds.) *Intelligent Information Systems* – Physica–Verlag, Heidelberg, (accepted).

Obuchowicz, A., and Politowicz, K. (1997): *Evolutionary algorithms in optimization of a multilayer feedforward neural network architecture.* – In: *Methods and Models in Automation and Robotics*, 4th Int. Symp. MMAR'97, Miedzyzdroje, Poland, Vol.2, pp.739–743.

Obuchowicz, A., and Witczak, M. (2002): *Genetic programming in fault diagnosis of nonlinear dynamic systems.* In: *Automatics*, 14th National Conf., Zielona Góra, Poland – University of Zielona Góra Press, pp. 561–566, (in Polish).

Osyczka, A. (2002): *Evolutionary Algorithms for Single and Multicriteria Design Optimization.* – Physica–Verlag, Heidelberg.

Park, D.J., Yun. B.E., and Kim, J.H. (1992): *Novel fast training algorithm for multilayer feedforward neural network.* – Electronics Letters, Vol.28, No.6., pp.534–544.

Patan, K. (2000): *Artificial Dynamic Neural Networks and Their Application in Modeling of Industrial Processes.* – Ph.D. Thesis, Warsaw University of Technology, Warsaw.

Patan, K., and Jesionka, M. (1999): *Genetic algorithms approach to network of dynamic neurons training.* – In: *Evolutionary Algorithms and Global Optimization*, 3rd Conf., Potok Złoty, Poland, pp.261–268.

Patan, K., and Korbicz, J. (1996): *Application of dyamic neural network in modeling and identification.* – In: *Methods and Models in Automation and Robotics*, 3rd Int. Symp. MMAR'96, Międzyzdroje, Poland – Technical University of Szczecin Press, Vol.3, pp.1141–1146.

Patan, K., and Korbicz, J. (1997) *Dynamic neural network with filters of different orders.* – In: *Methods and Models in Automation and Robotics*, 4th Int. Symp. MMAR'97, Międzyzdroje, Poland – Technical University of Szczecin Press, Vol.2, pp.745–750.

Patan, K., and Obuchowicz, A. (1999): *A stochastic approach to training networks of dynamic neurons.* – In: *Neural Networks and Their Applications*, 4th Int. Conf., Zakopane, Poland, pp.443–448.

Patan, K., Obuchowicz, A., and Korbicz, J. (1998): *Network of dynamic neurons approach to residual generators.* – In: *Methods and Models in Automation and Robotics*, 5th Int. Symp. MMAR'98, Mieędzyzdroje, Poland, Vol.2, pp.645–650.

Patan, K., Obuchowicz, A., and Korbicz, J. (1999): *Cascade network of dynamic neurons in fault detection systems.* – In: Proc. *European Control Conference*, ECC'99, Karlsruhe, Genrmany, (published on CD-ROM).

Patton, R.J., Chen, J., and Siew, T. (1994) *Fault diagnosis in nonlinear dynamic ystems via neural networks.* – In: Proc. of CONTROL'94, Coventry, UK, Vol.2, pp.1346–1351.

Patton, R.J., Frank, P.M., and Clark, R.N. (2000): *Issues in Fault Diagnosis for Dynamic Systems.* – Springer–Verlag, London.

Pieczyński, A. (1999): *Computer Diagnosis Systems for Industrial Processes.* – Technical University of Zielona Góra Press, Zielona Góra, (in Polish).

Rechenberg, I. (1965): *Cybernetic solution path of an experimental problem.* – Roy. Aircr. Establ., libr. transl. 1122. Farnborough, Hants., UK.

Reeves, C.R., and Steele, N. (1992): *Problem–solving by simulated genetic process. A review and application to neural networks.* – In: *Applied Informatics*, 10th Int. Conf. IASTED, pp.269–272.

Reeves, C.R., and Wright, C.C. (1995): *Epistasis in genetic algorithms: an experimental design perspective.* – In: *Genetic Algorithms*, 6th Int.Conf. – Morgan Kauffman, pp.217-230.

Rich, E. (1983) *Artificial Intelligence.* – McGraw-Hill Company, New York.

Riedmiller, M., and Braun, H. (1992): *A Fast Adaptive Learning Algorithm.* – Technical Report, University Karslruhe, Germany.

Rudolph, G. (1997): *Local convergence rates of simple evolutionary algorithms with Cauchy mutations.* – IEEE Trans. Evolutionary Computation, Vol.1, No.4, pp.249–258.

Rumelhart, D.E., Hinton, G.E., and Williams, R.J. (1986): *Learning representations by back–propagating errors.* – Nature, Vol.323, pp.533–536.

Rutkowska, D., Pliński, M., and Rutkowski, L. (1997): *Neural Networks, Genetic Algorithms and Fuzzy Systems.* – PWN, Warsaw, (in Polish).

Sastry, P.S., Santharam, G., and Unnikrishnan, K.P. (1994): *Memory neuron networks for identification and control of dynamic systems.* – IEEE Trans. Neural Networks, Vol.5, pp.306–319.

Schaefer, R. (2002): *Foundations of the Genetic Global Optimization.* – Jagiellonian University Press, Kraków, (in Polish).

Schalkoff R.J. (1990): *Artificial Intelligence : an Engineering Approach.* – McGraw-Hill, New York.

Schwefel, H.-P. (1981): *Numerical optimization of computer models.* – Wiley, Chichester.

Schwefel, H.-P. (1995): *Evolution and Optimum Seeking.* – John Wiley & Sons Inc, NY.

Setiono, R., and Chi Kwong Hui, L. (1995): *Use of a quasi-newton method in feedforward neural network construction algorithm.* – IEEE Trans. Neural Networks, Vol.6, No.1, pp.273–277.

Sharkey, A.J.C. (Ed.) (1999): *Combining Artificial Neural Nets.* – Springer-Verlag, London, UK.

Silva, F.M., and Almeida, L. (1990): *Speeding up backpropagation.* – In: *Advanced Neural Computes* – Elsevier Science Publishers B.U., pp.151–158.

Skowroński, K. (1998) *Genetic rules induction based on MDL principle.* – In: M. Kłopotek, M. Michalewicz, and Z.W. Raś (Eds.) *Intelligent Information Systems*, 7th International Symposium at Malbork, Poland – Polish Academy of Sciences Press, Warsaw, pp.356–360.

Sorsa, T., and Koivo, H.N. (1992): *Application of neural networks in the detection of breaks in a paper machine.* – In: *On–line Fault Detection and Supervision in the Chemical Process Industries*, IFAC Symp., Newark, Delaware, USA, pp.162–167.

Specht, D.F. (1991): *A general regression neural network.* – IEEE Trans. Neural Networks, Vol.2, No.6, pp.568–576.

Shu, H., and Hartley, R. (1987) *Fast simulated annaeling.* – Phys. Lett. A, Vol.122, nos. 3/4, pp.605–614.

Świąć, A., and Bilski, J. (2000): *Backpropagation errors method and its modifications.* – In: W. Duch, J. Korbicz, L. Rutkowski, and R. Tadeusiewicz, (Eds.) *Biocybernetics and Biomedical Engineering 2000. Neural Networks* – Academic Publishing House EXIT, Warsaw, Poland, pp.73–110, (in Polish).

Trojanowski, K., and Michalewicz, Z. (1999a): *Extensions of evolutionary algorithms for non–stationary environments.* – In: *Evolutionary Algorithms and Global Optimisation*, 3rd National Conf. – Warsaw University of Technology Press, pp.317-328.

Trojanowski, K., and Michalewicz, Z. (1999b) *Evolutionary algorithms for non–stationary environments.* – In: M. Kłopotek, and M. Michalewicz, (Eds.) *Intelligent Information Systems*, 6th Int. Symp., Ustroń, Poland – Polish Academy of Sciences Press, Warsaw, pp.229-240.

Trojanowski, K., and Michalewicz, Z. (1999c): *Searching for optima in non–stationary environments.* – In: *Evolutionary Computation*, IEEE Congres CEC'99 – IEEE Publishing, pp.1843-1850.

Trojanowski, K., and Obuchowicz, A. (2001) *Measures for non-stationary optimization tasks.* – In: Kurkova V., Steele N.C., Neruda R., Karny M. (Eds.) *Artificial Neural Networks and Genetic Algorithms* – Springer–Verlag, Wien, pp.244–247.

Tsoi, A.C., and Back, A.D. (1994): *Locally recurrent globally feddforward networks: a critical review of architectures.* – IEEE Trans. Neural Network, Vol.5, pp.229–239.

Tu, H.-C., Lin, Y.-P., and Tang, W. (1995): *A performance study of random search techniques in neural network learning.* – In: Proc. 3rd *European Control Conference ECC'95*, Rome, Italy, pp.1179–1184.

Vavak, F., and Fogarty, T.C. (1996): *Comparison of steady state and generational genetic algorithm for use in nonstationary environments.* – In: *Evolutionary Computation*, Int.Conf. ICEC'96 - IEEE Publishing, Inc., pp.192-195.

Vavak, F., Fogarty, T.C., and Jukes, K. (1997): *Learning the local search range for genetic optimisation in nonstationary environments.* – In: *Evolutionary Computation*, Int.Conf. ICEC'97 – IEEE Publishing, Inc., pp.355-360.

Vose, M.D. (1999): *The Simple Genetic Algorithm.* – MIT Press.

Walter, E., and Pronzato, L. (1997): *Identification of Parametric Models from Experimental Data.* – Springer Verlag, Berlin.

Wang, Z., Tham, M., and Morris, A.J. (1992): *Multilayer feedforward neural networks: a cannonical form approximation of nonlinearity.* – Int. Journ. Control, Vol.56, pp.665–672.

Wang, Z., Di Massimo, Ch., Tham, M.T., and Morris, A.J. (1994) A Procedure for Determining the Topology of Multilayer Feedforward Neural Networks. Neural Networks, Vol.7, No.2, pp.291–300.

Weigend, A.S., and Rumelhart, D.E. (1991): *The effective dimension of the space of hidden units.* – In: *Neural Networks*, Int. Joint Conf. IJCNN'91, Sattle, USA.

Werbos, P.J. (1974): *Beyond Regression: New Tools for Prediction and Analysis in the Behavioral Sciences.* – Ph.D. Thesis, Harvard University.

Whitley, D. (1994): *A genetic algorithm tutorial.* – Statiscitcs and Computing, Vol.4, pp.65–85.

Williams, R.J., and Zipser, D. (1989): *A learning algorithm for continually running fully recurrent neural networks.* – Neural Computation, Vol.1, pp.270–289.

Witczak, M., and Korbicz, J. (2000): *Genetic programming based observers for nonlinear systems.* – In: *Fault Detection, Supervision and Safety of Technical Processes*, IFAC Symp. SAFEPROCESS'2000, Budapest, Hungary, pp.967–972.

Witczak, M., Obuchowicz, A., and Korbicz, J. (1999): *Design of non-linear state observers using genetic programming.* – In: *Evolutionary Algorithms and Global Optimization*, 3nd Conference, Potok Złoty, Poland – Warsaw University of Technology Press, Warsaw, pp.345–352.

Witczak, M., Obuchowicz, A., and Korbicz, J. (2002): *Genetic programming based approaches to identification and fault diagnosis of nonlinear dynamic systems.* – Int. J. Control, Vol.75, No.13, pp.1012–1031.

Yao, X. (1993): *A review of evolutionary artificial neural networks.* – Int. J. Inteligent Systems, pp.539-567.

Yao, X., and Liu, Y. (1996) *Fast evolutionary programming.* – In: L.J.Fogel, P.J. Angeline, and T.Bäck (Eds.) *Evolutionary Programming V*, Proc. 5th Annual Conference – MIT Press, Cambridge, MA, pp.419–429.

Yao, X., and Liu, Y. (1997): *Fast evolutionary strategies.* – Contr. Cybern., Vol.26, No.3, pp.467–496.

Yao, X., and Liu, Y. (1999): *Evolutionary programming made faster.* – IEEE Trans. on Evolutionary Computation, Vol.3, No.2, pp.82–102.

Zieliński R.A., and Neumann P. (1983): *Stochastic methods of the function minimum searching.* – Springer–Verlag, Berlin.

Zhang, J., and Man, K.F. (1998): *Time series prediction using RNN in multidimension embedding phase space.* – In: *Systems, Man and Cybernetics*, IEEE Int. Conf., San Diego, USA, pp.1868–1873, (published on CD-ROM).

Zhu, C., and Paul, F.W. (1995): *A fourier series neural network and its application to system identification.* – Trans. ASME J. Dynamic Syst., Measurement and Control, Vol.117, pp.253–261.