



**Tomas Bata University in Zlín**  
**Faculty of Applied Informatics**

**Doctoral Thesis**

**NEURAL NETWORK SYNTHESIS**

**Pavel Vařacha**

Specialization: Engineering Informatics

Supervisor: prof. Ing. Ivan Zelinka, Ph.D.

Department of Informatics and Artificial Intelligence

Zlín, 2011



# ACKNOWLEDGEMENT

My warm thanks belong especially to the following people:

- my supervisor prof. Ivan Zelinka, for his confidence, support, valuable consultations and his patience with me
- my beloved wife, parents, brother and grandmother for their unconditional love and support
- associated prof. Eva Volná who boosted my morale in times of scientific deprivation
- Tomáš Dulík and Michal Bližňák who gave me their friendship, scientific example and technical support
- Bronislav Chramcov, Viliam Dolinay, Roman Jašek, Erik Král, Monika Křištofová, Zuzana Oplatková, Martin Pospíšilík, Roman Šenkeřík, Lubomír Vašek, (in alphabetical order without academic titles) and all other Faculty of Applied Informatics members who have become my second family
- my best friend in science as well as in joy Tomáš Horák
- František Petřík and Miroslav Suchomel for their prayers and profession
- Iva Malinová for her attentive language corrections

*Motto:*

*Only love is creative.*

*Saint Maxmilian Maria Kolbe*

*Dedication:*

*To Jesus Christ for His unconditional love.*



## SUMMARY

This thesis describes a feed forward Artificial Neural Network (ANN) synthesis via an Analytic Programming (AP) by means of the ANN creation, learning and optimization. This process encompasses four different fields: Evolutionary Algorithms, Symbolic Regression, ANN and parallel computing to successfully synthesize a suitable ANN within a reasonable time.

AP performs well in many separate cases together with different evolutionary algorithms as its “engine”. Direct asynchronous parallelization of SOMA – Self-Organizing Migration Algorithm is applied here to boost AP with unusual efficiency.

Direct asynchronously parallel SOMA distribution is experimentally tested and statistically evaluated and its suitability for AP is proved. The thesis describes an ANN synthesis used for function approximation and shows that an optimized and a suitable ANN is easily found by the presented method while the innovative PRT (SOMA control parameter) adaptive strategy is employed. Statistical evaluation of this strategy impact on AP performance is evaluated as well as different AP settings.

The ANN synthesis method is applied to the real life problem of Heat Load Prediction function optimization of the heating plant in Komořany (Czech Republic) as well as on cancer classification problem and is compared with other methods.

Software for the ANN synthesis support was developed under .NET Framework 3.5 and source codes were written in C#.

ANN synthesis proved to be a useful and efficient tool for nonlinear modeling and its results were applied to intelligent system controlling an energetic framework of an urban agglomeration.

Furthermore, the ANN synthesis proved to have the ability to synthesize smaller ANN than the Genetic Programming (GP) while simultaneously almost infinitely complex ANN can be synthesized by the application of multiple evolution loops. This process can also produce ANN with feed forward branching, which is an unavailable quality for the GP.

## RESUMÉ

Tato dizertační práce popisuje metodu syntézy dopředných umělých neuronových sítí (ANN) pomocí Analytického Programování (AP). Tento proces obsahuje vytvoření, učení i optimalizaci ANN. Syntéza ANN v sobě zahrnuje poznatky ze čtyř různých odvětví: evoluční algoritmy, symbolická regrese, ANN a paralelní výpočty. Díky tomu je možno úspěšně syntetizovat vhodné ANN v přijatelném čase.

AP podává velmi dobré výsledky za použití nejrůznějších EA jako jeho „pohonu“. Přímá asynchronní paralelizace SOMA je zde použita k navýšení výkonu AP s neobyčejnou efektivitou.

Tento přístup je experimentálně testován a jeho statistické zhodnocení opravňuje jeho použití s AP. Syntéza ANN je dále úspěšně nasazena k získání optimální ANN pro aproximaci dané funkce za použití adaptivní PRT (řídící parametr SOMA) strategie. Vyhodnocení dopadu této inovativní strategie společně s různými strategiemi GFS na výkon AP dokazuje její značný přínos.

Syntéza ANN je prakticky aplikována na problémy reálného života, jako je optimalizace funkce predikující spotřebu tepla dodávaného teplárnou Komořany, nebo klasifikaci rakoviny. Dosažené výsledky jsou porovnány s konkurenčními metodami.

V rámci práce bylo vyvinuto softwarové řešení pro podporu syntézy ANN. Technologický základ tohoto software je postaven na principech .NET Framework 3.5 a jeho zdrojový kód je naprogramován v jazyce C#.

Syntéza ANN prokázala svoji užitečnost a efektivitu jako nástroj nelineárního modelování a její výsledky byly využity v rámci Inteligentního systému pro řízení energetického systému městské aglomerace.

Syntéza ANN navíc ukázala svoji schopnost syntetizovat menší sítě než algoritmus Genetického Programování (GP) a přitom současně umožňuje vytvořit téměř nekonečně komplexní ANN pomocí většího počtu evolučních kol. Tento proces může také vytvářet dopředně rozvětvené ANN, čehož GP není schopno.

# CONTENTS

<b>LIST OF FIGURES</b> .....	<b>10</b>
<b>LIST OF TABLES</b> .....	<b>13</b>
<b>LIST OF IMPORTANT TERMS AND ABBREVIATIONS</b> .....	<b>14</b>
<b>1 INTRODUCTION</b> .....	<b>16</b>
<b>2 THE AIMS OF THE DISSERTATION</b> .....	<b>18</b>
<b>THEORETICAL FRAMEWORK</b> .....	<b>19</b>
<b>3 EVOLUTIONARY ALGORITHMS</b> .....	<b>20</b>
3.1 GENETIC ALGORITHMS.....	20
3.1.1 <i>Crossover</i> .....	21
3.1.2 <i>Mutation</i> .....	22
3.2 DIFFERENTIAL EVOLUTION.....	23
3.3 PARTICLES SWARM OPTIMIZATION.....	25
3.4 SELF-ORGANIZING MIGRATION ALGORITHM .....	26
3.4.1 <i>Parameter definition</i> .....	26
3.4.2 <i>Creation of Population</i> .....	27
3.4.3 <i>Migration loop</i> .....	27
3.4.4 <i>Test for stopping condition</i> .....	28
3.4.5 <i>SOMA Recommended Settings</i> .....	30
3.4.6 <i>Null PRTVector Problem Definition</i> .....	30
<b>4 SYMBOLIC REGRESSION</b> .....	<b>34</b>
4.1 GENETIC PROGRAMMING.....	34
4.2 GRAMMATICAL EVOLUTION .....	35
4.3 ANALYTIC PROGRAMMING .....	35
<b>5 NEURAL NETWORKS OPTIMISATION</b> .....	<b>39</b>
5.1 EVOLUTIONARY DESIGNED NEURAL NETWORK.....	40
5.2 ANN GENERATING BY GENETIC PROGRAMMING .....	42
5.3 NEURAL NETWORK SYNTHESIS .....	43
5.3.1 <i>Constant Processing</i> .....	47
5.3.2 <i>Reinforced Evolution</i> .....	48
5.3.3 <i>Cost Function Specification</i> .....	48
<b>6 DISTRIBUTED COMPUTATION</b> .....	<b>50</b>
6.1 ISLAND DISTRIBUTION OF SOMA.....	50

6.1.1	<i>Synchronous Island Model</i> .....	50
6.1.2	<i>Asynchronous Island Model</i> .....	50
6.2	DIRECT ASYNCHRONOUS DISTRIBUTION OF SOMA .....	51
	<b>PRACTICAL PART</b> .....	<b>54</b>
<b>7</b>	<b>ASYNCHRONOUS SOMA PERFORMANCE</b> .....	<b>55</b>
7.1	RESULTS .....	56
<b>8</b>	<b>ANN SYNTHESIS FOR FUNCTION APROXIMATION</b> .....	<b>58</b>
8.1	RESULTS .....	60
8.2	CONCLUSION .....	62
<b>9</b>	<b>ANN SYNTHESIS STRATEGY EXPLORATION</b> .....	<b>63</b>
9.1	ADAPTIVE PRT STRATEGY .....	63
9.1.1	<i>Adaptive PRT Strategy Test on Test Functions</i> .....	63
9.1.2	<i>Adaptive PRT Strategy for AP Handling</i> .....	65
9.1.3	<i>Adaptive PRT Strategy for <math>K_n</math> Estimation</i> .....	66
9.1.4	<i>Conclusion</i> .....	66
9.2	COMPARISON OF SYNCHRONOUS AND ASYNCHRONOUS SYNTHESIS .....	66
9.3	ANN SYNTHESIS RUNNING WITH DIFFERENT EA .....	67
<b>10</b>	<b>ANN SYNTHESIS FOR PREDICTION</b> .....	<b>68</b>
10.1	HEAT LOAD PREDICTION .....	70
10.1.1	<i>Temperature dependent component</i> .....	70
10.1.2	<i>Time dependent component</i> .....	71
10.2	ANN SYNTHESIS FOR HLP .....	73
10.3	RESULTS .....	74
10.4	PREDICTION BY STANDARD FEEDFORWARD ANN.....	80
10.5	ARITHMETICAL APPROACH TO GFS STRUCTURE FORMULATION.....	82
10.6	CONCLUSION .....	83
<b>11</b>	<b>ANN SYNTHESIS FOR CLASS CLASSIFICATION</b> .....	<b>85</b>
11.1	XOR CLASSIFICATION PROBLEM.....	85
11.2	CANCER CLASSIFICATION PROBLEM.....	86
11.2.1	<i>Experiment Set Up</i> .....	87
11.2.2	<i>Results</i> .....	87
11.2.3	<i>Conclusion</i> .....	89
11.2.4	<i>Terminals Density Comparison for Different GFS</i> .....	89
<b>12</b>	<b>ANN SYNTHESIS SOFTWARE</b> .....	<b>90</b>



<b>13 FINAL CONCLUSION .....</b>	<b>97</b>
<b>14 REFERENCES.....</b>	<b>98</b>
<b>15 LIST OF AUTHOR'S PUBLICATION ACTIVITIES.....</b>	<b>107</b>
<b>16 CURRICULUM VITAE .....</b>	<b>114</b>
<b>17 APENDIX I TEST FUNCTION VISUALISATION.....</b>	<b>116</b>
<b>18 APPENDIX II ADAPTIVE PRT STRATEGY .....</b>	<b>123</b>
<b>19 APPENDIX III – ANN SYNTHESIS RESULTS .....</b>	<b>128</b>
<b>20 APPENDIX IV – SUPER MICRO SERVER .....</b>	<b>132</b>
<b>21 APPENTIX V – ASYNCHRONOUS SOMA IN C#.....</b>	<b>138</b>
<b>22 APPENDIX VI – XML RESULT FORMAT.....</b>	<b>142</b>

## LIST OF FIGURES

<i>Fig. 1: Neural network synthesis intersection with connected scientific disciplines.....</i>	<i>16</i>
<i>Fig. 2: Main principle of EA .....</i>	<i>20</i>
<i>Fig. 3: GA crossover of individuals.....</i>	<i>22</i>
<i>Fig. 4: Mutation of GA individuals .....</i>	<i>22</i>
<i>Fig. 5: DE example.....</i>	<i>24</i>
<i>Fig. 6: All-to-One SOMA migration loop.....</i>	<i>26</i>
<i>Fig. 7: PRTVector and its action on individual movement .....</i>	<i>28</i>
<i>Fig. 8: SOMA example .....</i>	<i>29</i>
<i>Fig. 9: SOMA dependence on PRT size [19].....</i>	<i>30</i>
<i>Fig. 10: Probability of null PRTVector for <math>L = 100</math> .....</i>	<i>31</i>
<i>Fig. 11: Probability of null PRTVector for <math>L = 25</math> .....</i>	<i>32</i>
<i>Fig. 12: <math>P_1</math> for <math>L = 100</math>.....</i>	<i>33</i>
<i>Fig. 14: DSH principle [31].....</i>	<i>35</i>
<i>Fig. 15: GFS subsets hierarchy.....</i>	<i>36</i>
<i>Fig. 16: Main principles of AP.....</i>	<i>37</i>
<i>Fig. 17: Example of one and two hidden layer ANN.....</i>	<i>39</i>
<i>Fig. 18: Translation of a GP chromosome into ANN.....</i>	<i>43</i>
<i>Fig. 19: Principle of the evolutionary scanning.....</i>	<i>44</i>
<i>Fig. 20: AN transfer function for various <math>w, \lambda, \phi</math> settings .....</i>	<i>45</i>
<i>Fig. 21: Graphical example of AN .....</i>	<i>45</i>
<i>Fig. 22: Graphical example of plus operator.....</i>	<i>46</i>
<i>Fig. 23: Graphical example of weighted input.....</i>	<i>46</i>
<i>Fig. 24: Translation of an individual into ANN .....</i>	<i>46</i>
<i>Fig. 25: Learning of a synthesized ANN.....</i>	<i>47</i>
<i>Fig. 26: Example of GFS reinforcement process between two ANN evolution loops .....</i>	<i>48</i>
<i>Fig. 27: Example of CF for classification .....</i>	<i>49</i>
<i>Fig. 28: Asynchronous processing of a migration loop by different threads .....</i>	<i>51</i>
<i>Fig. 29: Proportional comparison of transferred information amounts needed for one CF computation within different EA .....</i>	<i>52</i>
<i>Fig. 30: Asynchronous individual processing .....</i>	<i>53</i>

Fig. 31: Asynchronous parallel movement of SOMA individuals .....	53
Fig. 32: Approximation of (37) by synthesized ANN .....	58
Fig. 33: Time saved by asynchronous evaluation .....	60
Fig. 34: Synthesized ANN according AN usage .....	61
Fig. 35: Graphical interpretation of resulting ANN.....	62
Fig. 36: Test functions performing better for $PRT \in \langle 0.005; 0.7 \rangle$ .....	64
Fig. 37: Test functions performing better for $PRT \in \langle 0.1; 0.3 \rangle$ .....	64
Fig. 38: Relationship between a heating plant (red), an agglomeration (green), atmospheric conditions (blue) and other events (black).....	68
Fig. 39: Basic scheme of the central heating plant system Komořany – Most.....	69
Fig. 40: Predictive function $f_{ime}(t)$ .....	71
Fig. 41: Predictive function $f_{EGH1}(t)$ for $\tau=3.6$ .....	72
Fig. 42: Predictive function $f_{EGH2}(t)$ for $\tau=-3.0$ .....	72
Fig. 43: Predictive function $f_P(t, \vartheta_{ex})$ [kW] .....	73
Fig. 44: Relationship between heating plant (red), ANN (green) and atmospheric condition (blue).....	73
Fig. 45: First evolution loop of synthesized ANN (52)(in red) .....	75
Fig. 46: Second evolution loop of synthesized ANN (54)(in green).....	75
Fig. 47: Third evolution loop of synthesized ANN (56)(in blue).....	76
Fig. 48: Fourth evolution loop of synthesized ANN (58)(in yellow).....	77
Fig. 49: Fifth evolution loop of synthesized ANN (59)(in purple) .....	77
Fig. 50: Sixth evolution loop of synthesized ANN (60)(in gray).....	78
Fig. 51: Seventh evolution loop of synthesized ANN (61)(in brown).....	78
Fig. 52: Exponential downgrade of migration loop significance for precision .....	80
Fig. 53: Matlab Neural Network Toolbox.....	80
Fig. 54: BP regression .....	81
Fig. 55: HLP by standard ANN.....	81
Fig. 56: ANN resulting from GFS containing arithmetical functions .....	82
Fig. 57: Surface of HLA function provided by synthesized ANN – areas significantly corrected in comparison with formal function are depicted in red.....	83
Fig. 58: Predicted and actual curve of agglomeration heat load .....	84
Fig. 59: XOR classification problem.....	85

Fig. 60: Minimal XOR solving ANN.....	85
Fig. 61: Minimal ANN generated by GPNN that performs the XOR problem.....	86
Fig. 62: First evolution loop of the resulting ANN.....	88
Fig. 63: Resulting ANN1 structural evolution.....	88
Fig. 64: Probability of terminal occurrence for different GFS.....	89
Fig. 65: SOMA controll parameters setting.....	90
Fig. 66: ANN synthesized using (85).....	92
Fig. 67: Asynchronous SOMA results form.....	94
Fig. 68: ANN synthesis software forms.....	96
Fig. 69: Optimal computation division on eight processors.....	96
Fig. 70: Ackley (27) 3D visualization.....	116
Fig. 71: Ackley (27) 2D visualization.....	116
Fig. 72: EggHolder (28) 3D visualization.....	117
Fig. 73 EggHolder (28) 2D visualization.....	117
Fig. 74: Michalewicz (29) 3D visualization.....	117
Fig. 75: Michalewicz (29) 2D visualization.....	118
Fig. 76: Masters (30) 3D visualization.....	118
Fig. 77: Masters (30) 2D visualization.....	118
Fig. 78: Michalewicz (31) 3D visualization.....	119
Fig. 79: Michalewicz (31) 2D visualization.....	119
Fig. 80: Rana (32) 3D visualization.....	119
Fig. 81: Rana (32) 2D visualization.....	120
Fig. 82: Rastrigin (33) 3D visualization.....	120
Fig. 83: Rastrigin (33) 2D visualization.....	120
Fig. 84: Rosenbrock (34) 3D visualization.....	121
Fig. 85: Rosenbrock (34) 2D visualization.....	121
Fig. 86: Schwefel (35) 2D visualization.....	121
Fig. 87: Schwefel (35) 2D visualization.....	122
Fig. 88: SineWave (36) 3D visualization.....	122
Fig. 89: SineWave (36) 2D visualization.....	122
Fig. 90: Test functions providing the best results for $PRT \in <0.005; 0.07>$ .....	123
Fig. 91: Test functions providing the best results for $PRT \in <0.1; 0.3>$ .....	123

<i>Fig. 92: Super Micro server</i> .....	132
<i>Fig. 93: Super Micro server motherboard</i> .....	132

## LIST OF TABLES

<i>Table 1: GA individuals coded to chromosomes</i> .....	21
<i>Table 2: An example of PRTVector for 4 parameters individual with PRT = 0.3</i> .....	28
<i>Table 3: SOMA parameters and their recommended domain</i> .....	30
<i>Table 4: Probability of null PRTVector for L = 100</i> .....	31
<i>Table 5: Probability of null PRTVector for L = 25</i> .....	31
<i>Table 6: <math>P_1</math> for L = 100</i> .....	33
<i>Table 7: Example of GFS and its subsets</i> .....	36
<i>Table 8: Test functions, ML and borders</i> .....	56
<i>Table 9: SOMA average results</i> .....	57
<i>Table 10: SOMA best results</i> .....	57
<i>Table 11: Setting of SOMA used as EA for AP</i> .....	59
<i>Table 12: Setting of SOMA used to optimize <math>K_n</math></i> .....	59
<i>Table 13: Time saved by asynchronous evaluation</i> .....	60
<i>Table 14: PRT strategy for AP handling</i> .....	65
<i>Table 15: PRT strategy for <math>K_n</math> estimation</i> .....	66
<i>Table 16: Synchronous and asynchronous SOMA performance</i> .....	67
<i>Table 17: ANN performance for different EA</i> .....	67
<i>Table 18: ANN mean testing classification error</i> .....	87
<i>Table 19: Best results for different cost functions and PRT settings</i> .....	124
<i>Table 20: Normalized best results for different cost functions and PRT settings</i> .....	125
<i>Table 21: Average results for different cost functions and PRT settings</i> .....	126
<i>Table 22: Normalized average results for different cost functions and PRT settings</i> ....	127
<i>Table 23: Super Micro server technical specification</i> .....	133

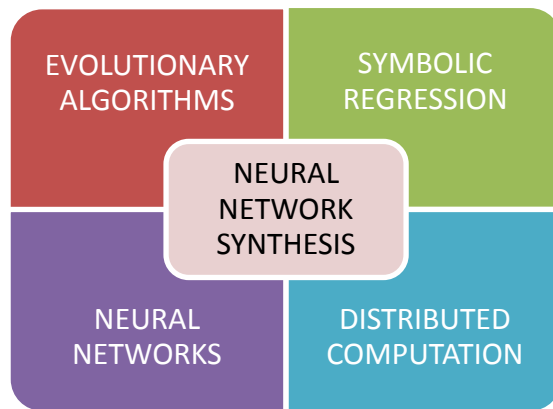
# LIST OF IMPORTANT TERMS AND ABBREVIATIONS

AN	Artificial Neuron
ANN	Artificial Neural Network
AP	Analytic Programming
BNF	Backus Naur Form
CF	Cost Function
CR	DE control parameter
DE	Differential Evolution
depth	parameter of individual in AP
EA	Evolutionary Algorithm
EGH	Gaussian Hybrid and truncated Exponential function
F	DE control parameter
GA	Genetic Algorithm
GE	Grammatical Evolution
GFS	General Function Set
$GFS_{0arg}$	Functions with 0 arguments in GFS, i.e. constants and variables
$GFS_{1arg}$	Functions with 1 argument in GFS (e.g. Sin, Cos, Tan..)
$GFS_{2arg}$	Functions with 2 arguments in GFS (e.g. +,-,/....)
$GFS_{3arg}$	Functions with 3 arguments in GFS
$GFS_{All}$	Set of all functions in general functional space

GP	Genetic Programming
GPNN	ANN design using GP
Height	border of random range (Low; Height)
HLP	Head Load Prediction
$K_n$	constants to estimate in AP
L	PRTVector's length
Low	border of random range (Low; Height)
ML	Migration Loop
NP	population size
NRMSD	Normalised Root Mean Square Deviance
PathLength	SOMA control parameter
PRT	SOMA control parameter
$P_o$	probability of null PRTVector
$P_1$	probability of PRTVector which contains 1 only ones
PSO	Particle Swarm Optimization
RMSD	Root Mean Square Deviance
SOMA	Self-Organizing Migration Algorithm
specimen	SOMA control parameter
Step	SOMA control parameter

# 1 INTRODUCTION

This thesis describes a feed forward Artificial Neural Network (ANN) synthesis (chapter 5.3) via an Analytic Programming (AP) (chapter 4.3) by means of the ANN creation, learning and optimization. This process encompasses four different fields: Evolutionary Algorithms (EA) (chapter 3), Symbolic Regression (chapter 4), ANN (chapter 5) and parallel computing (chapter 6) to successfully synthesize a suitable ANN within a reasonable time.



*Fig. 1: Neural network synthesis intersection with connected scientific disciplines*

There are well-known methods: Genetic Programming (chapter 4.1) and Grammatical Evolution (chapter 4.2), which can both symbolically regress using the evolutionary algorithm. However, this thesis is aimed at a more recent and flexible procedure called AP. (chapter 4.3)

AP performed well in many separate cases (for example [1],[2]) together with different evolutionary algorithms (EA) as its “engine”. A direct asynchronous parallelization of the SOMA – Self-Organizing Migration Algorithm [3] (chapter 6.2) is applied here to boost the AP with unusual efficiency.

SOMA (chapter 3.4) is based on the self-organizing behavior of groups of individuals in a “social environment”. It can also be classified as an evolutionary algorithm [4], despite the fact that no new generations of individuals are created during a search (due to the philosophy of this algorithm). Only the positions of individuals in the searched space



are changed during one generation called a “migration loop”. The algorithm was published in journals and books, presented at international conferences and symposiums and mentioned in numerous introductory presentations, for example [5], [6], [7].

The direct asynchronously parallel SOMA distribution is experimentally tested and statistically evaluated in chapter 7 and its suitability for the AP is proved. Chapter 8 describes an ANN synthesis usage for a function approximation and shows that the optimized and suitable ANN is easily found by the presented method while the innovative PRT (SOMA control parameter) adaptive strategy is employed. The statistical evaluation of the impact of this strategy on the AP performance is evaluated in chapter 9.

In this chapter, a total of 10 ANN synthesis abilities to successfully synthesize the ANN capable of predicting are tested on a real life problem of a heating plant. The ANN synthesis method is applied in order to optimize the Heat Load Prediction function of the heating plant in Komořany (Czech Republic).

To statistically evaluate the ANN synthesis’ ability to successfully generate an ANN performing classification, the ANN synthesis was compared with GP solving an XOR problem in chapter 11.1 while the chapter 11.2 describes the ANN synthesis usage for a real life cancer classification problem and its comparison with other methods.

Software for the ANN synthesis support was developed under .NET Framework 3.5 and source codes were written in C#. The software was used and debugged while performing experiments in chapters 6 to 11.

The ANN synthesis proves to be a useful and efficient tool for nonlinear modeling in comparison with competitive methods as described in chapter 13 which contains a final conclusion.

The following chapter 2 introduces the main aims of this thesis.

## 2 THE AIMS OF THE DISSERTATION

The main aim of the dissertation is a development of the Neural Network Synthesis method based on AP (chapter 4.3) and SOMA (chapter 3.4) algorithms; these are theoretically described in chapter 5.3 as a useful and efficient tool for nonlinear modeling.

An important part of this process is the application of the method to the specifically chosen tasks of the function approximation, prediction and classification of problems considering real life data as well as standardized benchmarks.

To support the ANN synthesis exploration, software capable of the ANN synthesis needs to be developed in order to conduct experiments and measure different approaches statistically. The obtained experimental results have to be evaluated in order to find optimal parameters for the application of the ANN synthesis to the given tasks.

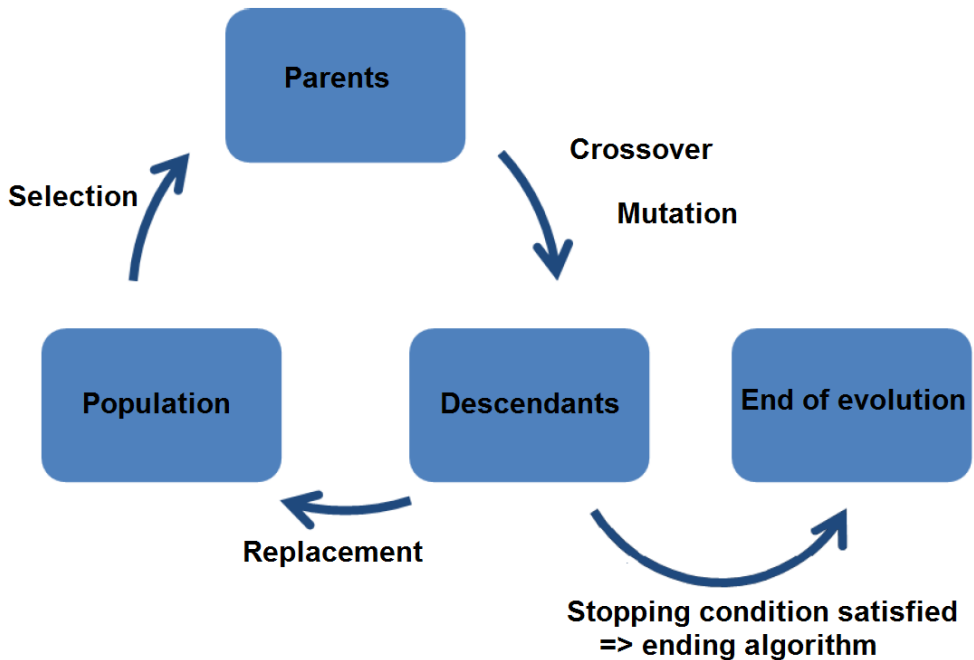
These aims are further described as follows:

- To apply the ANN synthesis for ANN creation and optimization based on the given problem of:
  - function approximation (chapter 8)
  - prediction (chapter 10)
  - classification (chapter 11)
- To statistically explore:
  - different structures of the GFS (chapter 10.5 and 11.2.4)
  - SOMA control parameters setting for AP handling (chapter 9.1.2)
  - SOMA control parameters setting for  $K_n$  estimation (chapter 9.1.3)
  - individuals' behavior invoked by the implementation of (25) (chapter 9.2)
- To develop software:
  - which automatically and efficiently distributes computation to all available processors (chapter 7)
  - which will automatically synthesize and/or optimize the ANN based on the data provided by the user within a reasonable time (chapter 12)

## **THEORETICAL FRAMEWORK**

### 3 EVOLUTIONARY ALGORITHMS

In recent years, a broad class of algorithms has been developed for stochastic optimization, i.e. for optimizing systems where the functional relationship between the independent input variables  $x$  and the output (objective function)  $y$  of a system  $S$  is not known. Using stochastic optimization algorithms such as Genetic Algorithms (GA) (chapter 3.1), Differential Evolution (DE) (chapter 3.2), Particle Swarm Optimization (PSO) (chapter 3.3) and SOMA (chapter 3.4) the system is confronted with a random input vector and its response is measured. This response is then used by the algorithm to tune the input vector in such a way that the system produces the desired output or target value in an iterative process.



*Fig. 2: Main principle of EA*

#### 3.1 Genetic Algorithms

GA belong to a group of methods, which are used to solve search and optimization problems. [8] The foundations of the GA were laid down in 1975 by John H. Holland [9].

Several different GA versions have been developed; however, the most important GA principle, coding of individuals into chromosomes, is common to all of them. [10]

The chromosome should in some way contain information about the solution, which it represents. The most used way of encoding is a binary string. The chromosome then could look like this:

*Table 1. GA individuals coded to chromosomes*

Chromosome 1	11101001000
Chromosome 2	00001010101

Simple generational genetic algorithm pseudo code [11] :

- Choose the initial population of individuals
- Evaluate the fitness of each individual in that population
- Repeat within this generation until termination: (time limit, sufficient fitness achieved, etc.)
  - Select the best-fit individuals for reproduction
  - Breed new individuals through crossover and mutation operations to give birth to an offspring
  - Evaluate the individual fitness of new individuals
  - Replace the least-fit population with new individuals

### **3.1.1 Crossover**

Crossover selects genes from parent chromosomes and creates a new offspring. The simplest way how to do this is to randomly choose some crossover point and copy everything before this point from the first parent and then copy everything after the crossover point from the second parent.

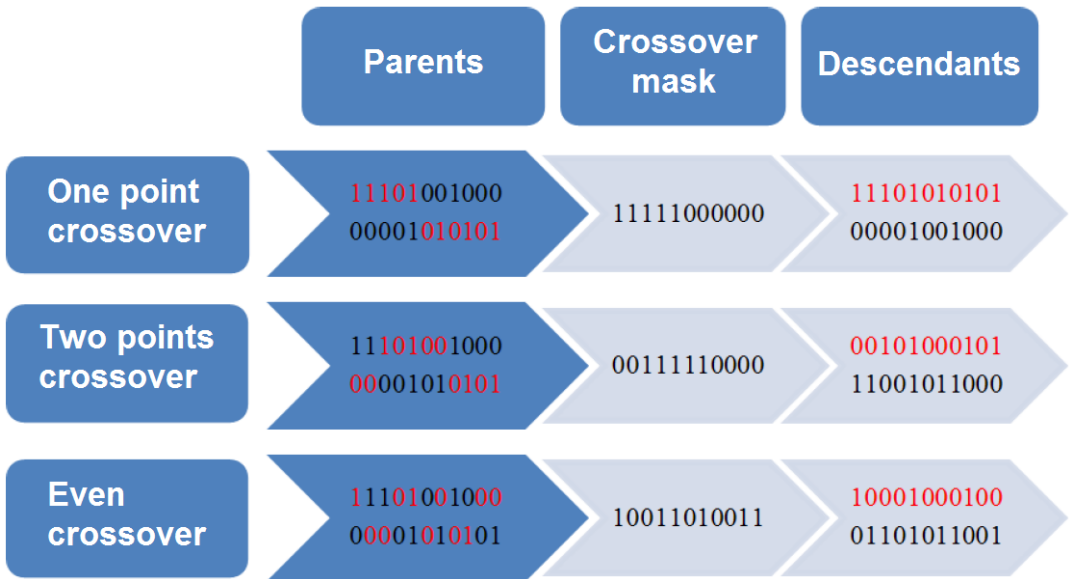


Fig. 3: GA crossover of individuals

### 3.1.2 Mutation

Mutation randomly changes the new offspring.

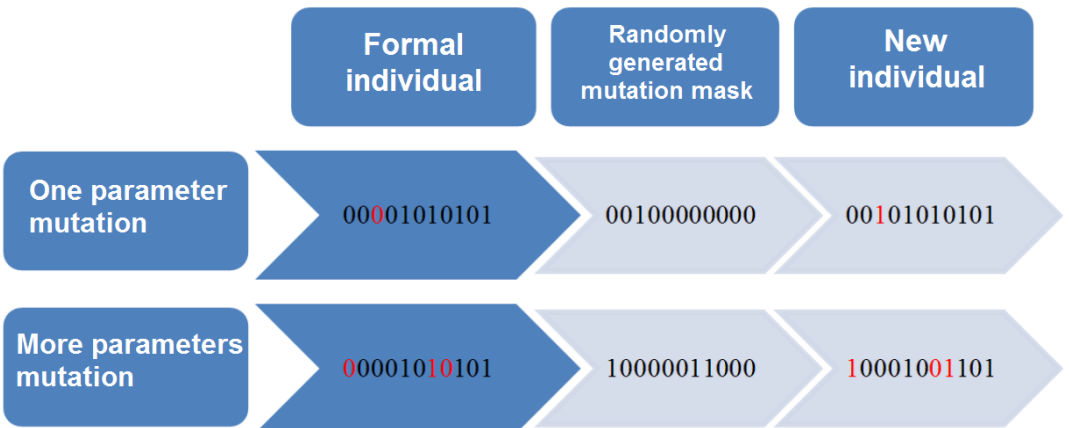


Fig. 4: Mutation of GA individuals

## 3.2 Differential Evolution

DE has been known in the scientific world since 1995. Fathers of DE are Ken Price and Rainer Storm, [13]. DE is robust, fast, and effective with a global optimization ability [14].

Let  $x \in \mathbb{R}^n$  designate a candidate solution (individual) in the population. The basic DE algorithm can then be described as follows [15]:

- Initialize all individuals  $x$  with random positions in the search-space.
- Until a termination criterion is met (e.g. number of iterations performed, or adequate fitness reached), repeat the following:
  - For each individual  $x$  in the population do:
    - Pick three individuals **a**, **b**, and **c** from the population at random, they must be distinct from each other as well as from the individual  $x$
    - Pick a random index  $R \in \{1, \dots, n\}$ , where the highest possible value  $n$  is the dimensionality of the problem to be optimized
    - Compute the individual's potentially new position  $y = [y_1, \dots, y_n]$  by iterating over each  $i \in \{1, \dots, n\}$  as follows:
      - Pick  $r_i \sim (0,1)$  uniformly from the open range  $(0,1)$
      - If  $(i=R)$  or  $(r_i < CR)$  let  $y_i = a_i + F(b_i - c_i)$ , otherwise let  $y_i = x_i$
    - If  $(f(y) < f(x))$  then replace the individual in the population with the improved candidate solution, that is, set  $x = y$  in the population.
  - Pick the individual from the population that has the lowest fitness and return it as the best found candidate solution.

Note that  $F \in \langle 0,2 \rangle$  is called the *differential weight* and  $CR \in \langle 0,1 \rangle$  is called the *crossover probability*, both these parameters are selectable by a practitioner along with the population size  $NP > 3$ , see below.

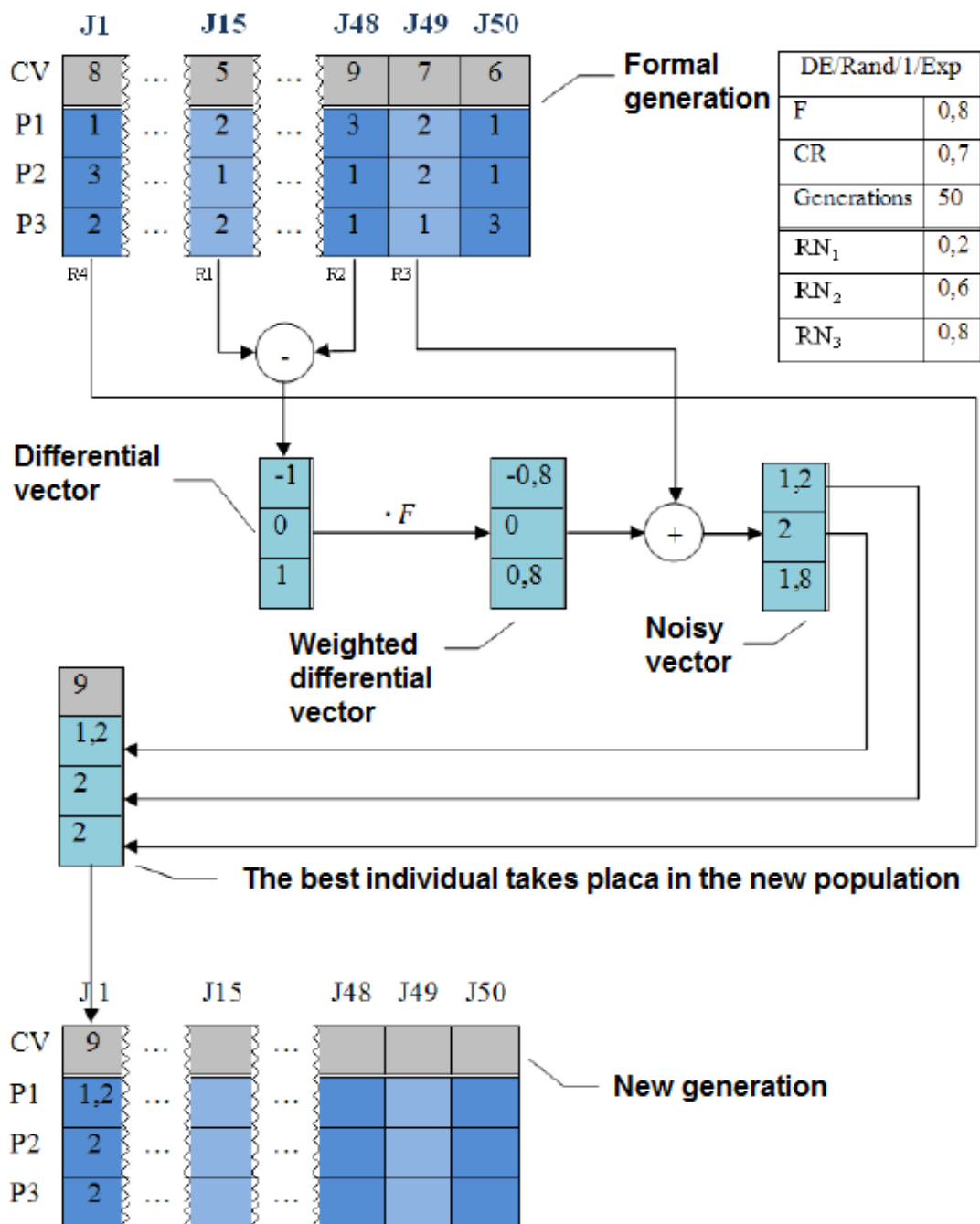


Fig. 5: DE example



### 3.3 Particles Swarm Optimization

PSO is originally attributed to Kennedy, Eberhart and Shi (1995) [4], [16] and was primarily intended for social behavior simulation.

Let  $S$  be the number of particles (individuals) in the swarm, each having a position  $x \in \mathbb{R}^n$  in the search-space and a velocity  $v_i \in \mathbb{R}^n$ . Let  $p_i$  be the best known position of particle  $i$  and let  $g$  be the best known position of the entire swarm. A basic PSO algorithm is then [17]:

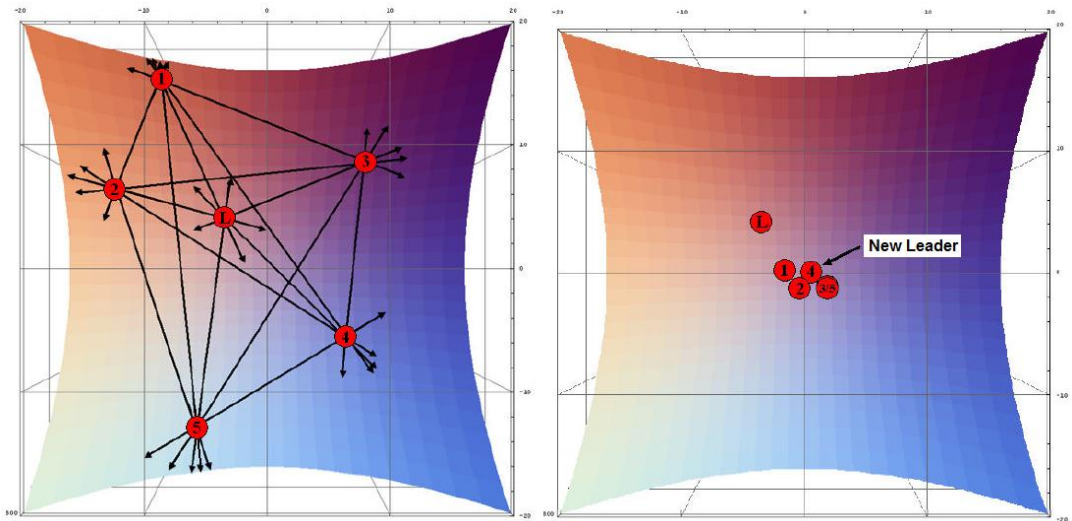
- For each particle  $i = 1, \dots, S$  do:
  - Initialize the particle's position with a uniformly distributed random vector:  $x_i \sim U(\mathbf{Low}, \mathbf{High})$ , where  $\mathbf{blo}$  and  $\mathbf{bup}$  are the lower and upper boundaries of the search-space.
  - Initialize the particle's best known position to its initial position:  $p_i \leftarrow x_i$
  - If  $(f(p_i) < f(g))$  update the swarm's best known position:  $g \leftarrow p_i$
  - Initialize the particle's velocity:  $v_i \sim (-|\mathbf{Hight} - \mathbf{Low}|; |\mathbf{Hight} - \mathbf{Low}|)$
- Until a termination criterion is met (e.g. number of iterations performed, or adequate fitness reached), repeat:
  - For each particle  $i = 1, \dots, S$  do:
    - Pick random numbers:  $rp, rg \sim (0,1)$
    - Update the particle's velocity:  $v_i \leftarrow \omega v_i + \phi p rp (p_i - x_i) + \phi g rg (g - x_i)$
    - Update the particle's position:  $x_i \leftarrow x_i + v_i$
    - If  $(f(x_i) < f(p_i))$  do:
      - Update the particle's best known position:  $p_i \leftarrow x_i$
      - If  $(f(p_i) < f(g))$  update the swarm's best known position:  $g \leftarrow p_i$
- Now  $g$  holds the best found solution.

The parameters  $\omega$ ,  $\phi p$ , and  $\phi g$  are selected by the practitioner and control the behavior and efficacy of the PSO method. [18]

### 3.4 Self-Organizing Migration Algorithm

SOMA is based on a self-organizing behavior of groups of individuals in a “social environment”. It can also be classified as an evolutionary algorithm [4], despite the fact that no new generations of individuals are created during the search (due to the philosophy of this algorithm). Only the positions of individuals in the searched space are changed during one generation called a “migration loop”. The algorithm was published in journals and books, presented at international conferences and symposiums and mentioned in numerous introductory presentations, for example [5], [6], [7].

Although several different versions of SOMA exist, this thesis is focused on the most common **All-to-One** version, which is suitable for the asynchronous parallel implementation. This chapter describes all basic All-to-One SOMA principles.



*Fig. 6: All-to-One SOMA migration loop*

#### 3.4.1 Parameter definition

Before starting the algorithm, SOMA’s parameters: Step, PathLength, PopSize, PRT and Cost Function need to be defined. The Cost Function is simply the function which returns a scalar that can directly serve as a measure of fitness. In this case, Cost Function is provided by AP.

### 3.4.2 Creation of Population

Population of individuals is randomly generated. Each parameter for each individual has to be chosen randomly from a Specimen which defines a range <Low, High> and a value type (integer, double) for each individual's dimension.

### 3.4.3 Migration loop

Each individual from a population (PopSize) is evaluated by the Cost Function and the Leader (individual with the highest fitness) is chosen for the current migration loop. Then, all other individuals begin to jump, (according to the Step definition) towards the Leader. Each individual is evaluated after each jump by using the Cost Function. Jumping continues until a new position defined by the PathLength is reached. The new position  $x_{i,j}$  after each jump is calculated by (1) as is shown graphically in Fig. 7. Later on, the individual returns to the position on its path, where it found the best fitness.

$$x_{i,j}^{MLnew} = x_{i,j,start}^{ML} + (x_{L,j}^{ML} - x_{i,j,start}^{ML})tPRTVector_j$$

where  $t \in <0, \text{by Step to, PathLegth}>$  (1)

*and ML is actual migration loop*

Before an individual begins jumping towards the Leader, a random number *rnd* is generated (for each individual's component), and then compared with PRT. If the generated random number is larger than PRT, then the associated component of the individual is set to 0 using PRTVector.

$$rnd_j < PRT \text{ then } PRTVector_j = 0 \text{ else } 1$$

where  $rnd \in <0, 1>$  (2)

*and  $j = 1, \dots, n_{param}$*

Table 2: An example of *PRTVector* for 4 parameters individual with  $PRT = 0.3$

<b>J</b>	<b>rnd<sub>j</sub></b>	<b>PRTVector</b>
1	0,234	1
2	0,545	0
3	0,865	0
4	0,012	1

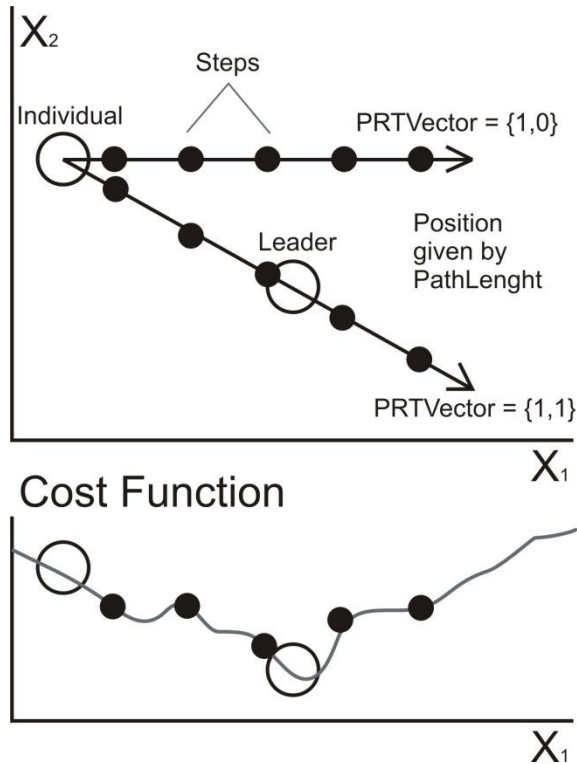


Fig. 7: *PRTVector* and its action on individual movement

Hence, the individual moves in the  $N-k$  dimensional subspace which is perpendicular to the original space. This fact establishes a higher robustness of the algorithm. Earlier experiments demonstrated that without the use of PRT, SOMA tends to determine a local optimum rather than a global one. [19]

#### 3.4.4 Test for stopping condition

If a stopping condition (time limit, sufficient fitness achieved, number of ML, etc.) is archived, stop and recall the best solution(s) found during the search.

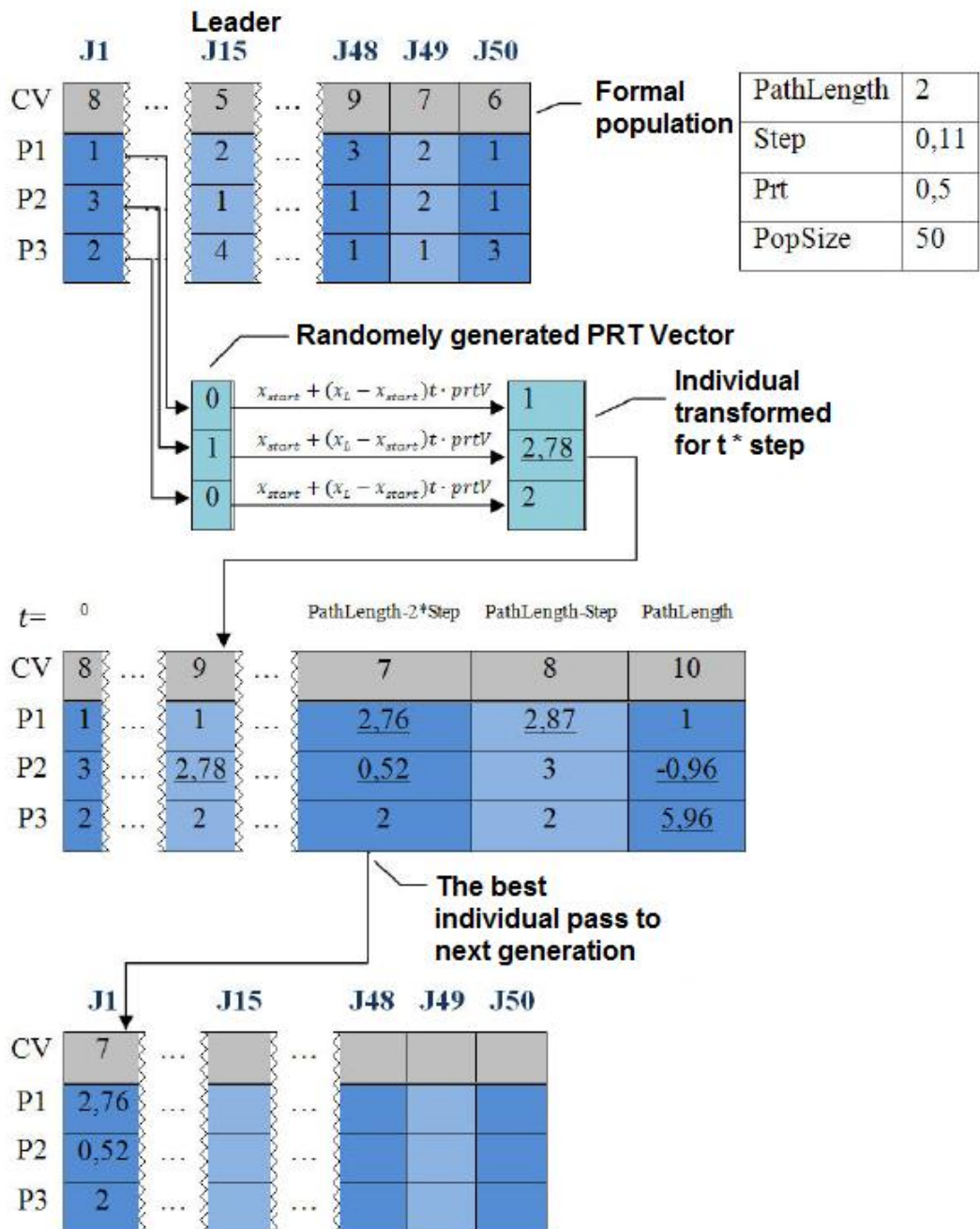


Fig. 8: SOMA example

### 3.4.5 SOMA Recommended Settings

Based on a huge number of experiments, the author of SOMA (prof. Zelinka) recommended the optimal setting for the algorithm's control parameters. [19]

Table 3: SOMA parameters and their recommended domain

Parameter name	Recommended range
PathLength	<1.1 ;3>
Step	<0.11, PathLength>
PRT	<0,1>
PopSize	<10, up to user>

As can be seen in Fig. 9 , a PRT parameter was tested within the range <0.1; 0.9> and performed best when  $PRT \in <0.1; 0.3>$ .

By contrast, this thesis explores SOMA's behavior within a much wider range  $PRT \in <0.005, 0.1>$ . The reasons why this possibility has never been explored before are described in the next chapter.

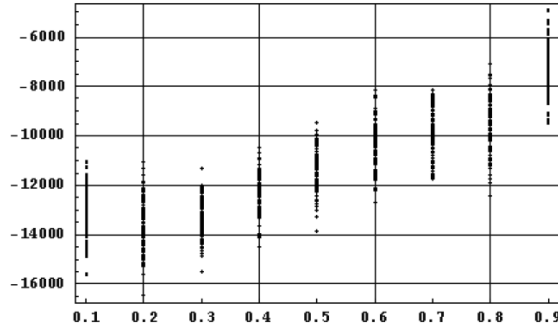


Fig. 9: SOMA dependence on PRT size [19]

### 3.4.6 Null PRTVector Problem Definition

All the experiments mentioned in [19] were performed on Cost Functions with 100 parameters. Naturally, the PRTVector's length (L) was also 100. The probability  $P_0$  that generated the PRTVector is a null vector (vector which contains nulls only, see also (1)) that is very low for  $PRT \in <0.1; 0.3>$ .

$$P_0 = (1 - PRT) \quad (3)$$

Table 4: Probability of null PRTVector for  $L = 100$

<b>PRT</b>	<b>P<sub>0</sub></b>
0,005	0,60577
0,01	0,366032
0,03	0,047553
0,05	0,005921
0,07	0,000705
0,1	2,66E-05
0,2	2,04E-10
0,3	3,23E-16

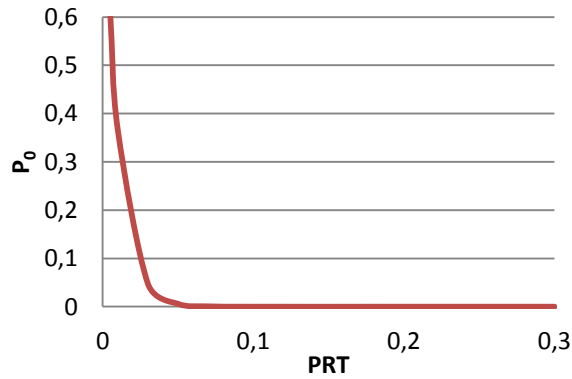
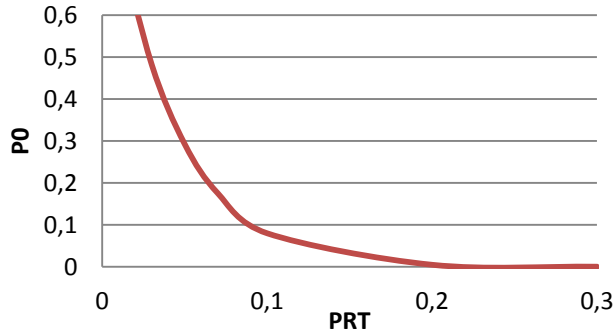


Fig. 10: Probability of null PRTVector for  $L = 100$

However,  $P_0$  increases dramatically if  $L$  or **PRT** value decreases.

Table 5: Probability of null PRTVector for  $L = 25$

<b>PRT</b>	<b>P<sub>0</sub></b>
0,005	0,886654
0,01	0,785678
0,03	0,481417
0,05	0,291989
0,07	0,175223
0,1	0,079766
0,2	0,004722
0,3	0,000192



*Fig. 11: Probability of null PRTVector for L = 25*

If the null PRTVector is generated, the individual does not move during the actual migration loop and the Cost Function is always evaluated with the very same parameters. For example, 27 evaluations are wasted if Step = 0.11 and PathLength = 3. This waste of computation time is highly improbable when L = 100 and also very low if the theoretical test functions (see chapter 7) are computed.

Let us consider a real life problem of the heating-plant parameters optimization. [20] (see also chapter 10.1 ). L = 24 means that one parameter for every hour during the day has to be optimized. If **PRT** = 0.1, P<sub>0</sub> = 0.79, almost 8% of the Cost Function evaluations are wasted. In doing so, one evaluation of the Cost Function is very time demanding (even in a range of minutes [21]) as a waste database has to be processed.

Such conditions approve an institution of a simple null PRTVector repair mechanism:

$$\begin{aligned} &\textbf{If PRTVector is the null vector,} \\ &\textbf{a new PRTVector is generated instead.} \end{aligned} \quad (4)$$

Consequently, P<sub>0</sub> is always 0. Instead of P<sub>0</sub>, probability P<sub>1</sub> of the PRTVector which contains 1 only ones can be considered.

$$P_1 = (1 - \mathbf{PRT})^L + L * \mathbf{PRT} * (1 - \mathbf{PRT})^{(L-1)} \quad (5)$$



Table 6:  $P_1$  for  $L = 100$

<b>PRT</b>	<b><math>P_1</math></b>
0,005	0,910178
0,01	0,735762
0,03	0,194622
0,05	0,037081
0,07	0,006013
0,1	0,000322
0,2	5,3E-09
0,3	1,42E-14

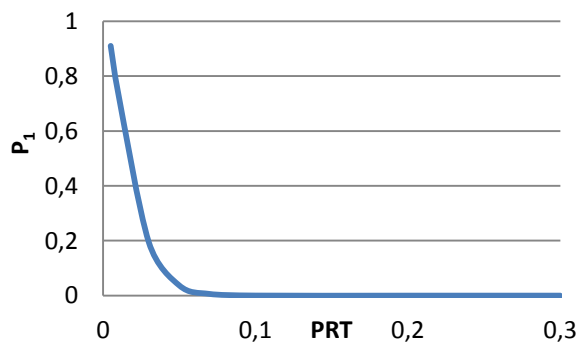


Fig. 12:  $P_1$  for  $L = 100$

The application of (4) into SOMA allows the **PRT** parameter to be set within the range  $(0; 0.1>$  which was previously unreachable due to high values of  $P_0$ .

More detailed information considering the null PRT vector problem can be found in [22].

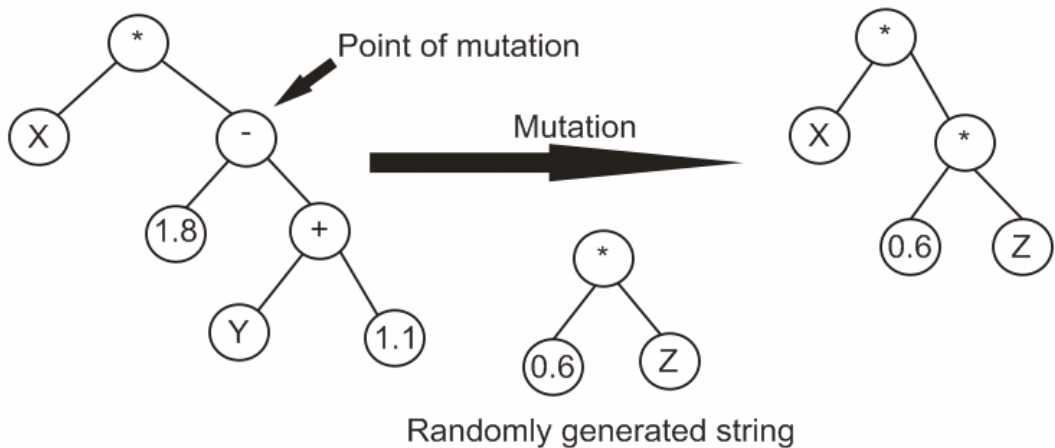
## 4 SYMBOLIC REGRESSION

The term symbolic regression represents a process in which measured data is fitted by a suitable mathematical formula such as  $x^2 + C$ ,  $\sin(x) + e^x$ , etc. This process is quite well-known and can be used when data of an unknown process is obtained.

There are two well-known methods: GP and GE, which can both symbolically regress the usage of the evolutionary algorithm; however, this thesis uses another more flexible method called Analytic Programming, which can be implemented on the arbitrary EA. A comprehensive survey of the symbolic regression methods can be found in [23].

### 4.1 Genetic Programming

GP was introduced at the end of the 1980s by John Koza [24], [25]. He suggested a modification to a genetic algorithm (see chapter 3.1) and he called it Genetic Programming. In this concept a new population is not bred in the common numerical way but in the analytical way. It means that the solution of such breeding is not values of parameters but the function itself. [26]



*Fig. 13: Mutation in Genetic Programming*

## 4.2 Grammatical Evolution

Grammatical evolution (GE) is another tool for doing symbolic regression by computers. The advantage of this tool, compared to GP, is that GE can evolve complete programs in an arbitrary programming language [27], [28] using a variable-length binary string. It uses a Backus Naur Form (BNF) grammar definition for mapping a process to a program. GE performs the whole process on variable-length binary strings. The mapping process is employed to generate programs in any language by using the binary strings to select production rules in the BNF definition. The result is the construction of a syntactically correct program from a binary string that can then be evaluated by a fitness function. [29]

## 4.3 Analytic Programming

The main principle (core) of AP is based on a discrete set handling (DSH) (Fig. 14) and is inspired by GE. DSH shows itself as a universal interface between the EA and the symbolically solved problem. This is why AP can be used almost by any EA (see chapter 3). [30]

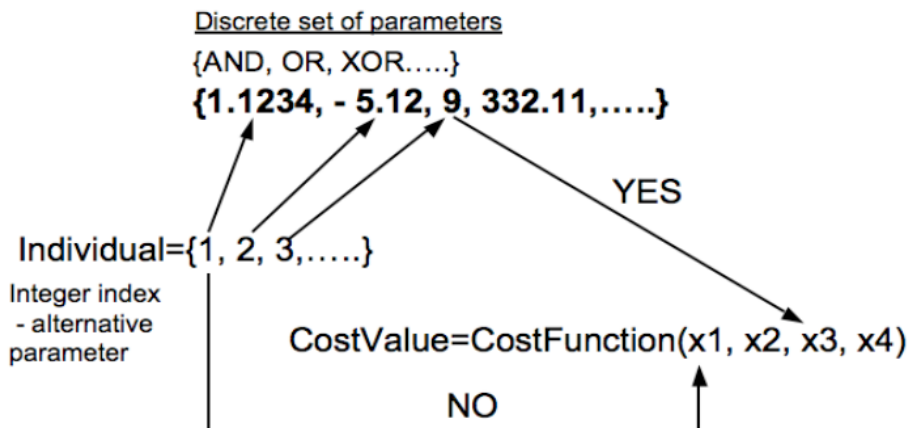


Fig. 14: DSH principle [31]

Briefly stated, in AP, individuals consist of non-numerical expressions (operators, functions,...) which are represented within the evolutionary process by their integer indexes. Each index then serves as a pointer into the set of expressions and AP uses it to synthesize the resulting function-program for the Cost Function evaluation.

All simple functions and operators are in the so called General Function Set (GFS) divided into groups according to the number of arguments which can be inserted during the evolutionary process to create subsets GFS<sub>3</sub>, GFS<sub>2</sub>...GFS<sub>0</sub>.

Table 7: Example of GFS and its subsets

GFS Degree	Contains
GFS <sub>all</sub>	f(x <sub>1</sub> , x <sub>2</sub> , x <sub>3</sub> ), +, -, *, /, Power, Abs, Round, Sin, Cos, t, K, τ, 1, 2
GFS <sub>3</sub>	f(x <sub>1</sub> , x <sub>2</sub> , x <sub>3</sub> )
GFS <sub>2</sub>	+, -, *, /, Power
GFS <sub>1</sub>	Abs, Round, Sin, Cos
GFS <sub>0</sub>	t, K, τ, 1, 2

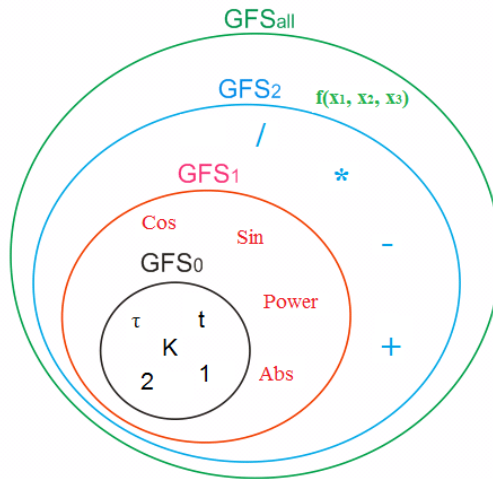


Fig. 15: GFS subsets hierarchy

The functionality of AP can be seen in the specific example in Fig. 16:

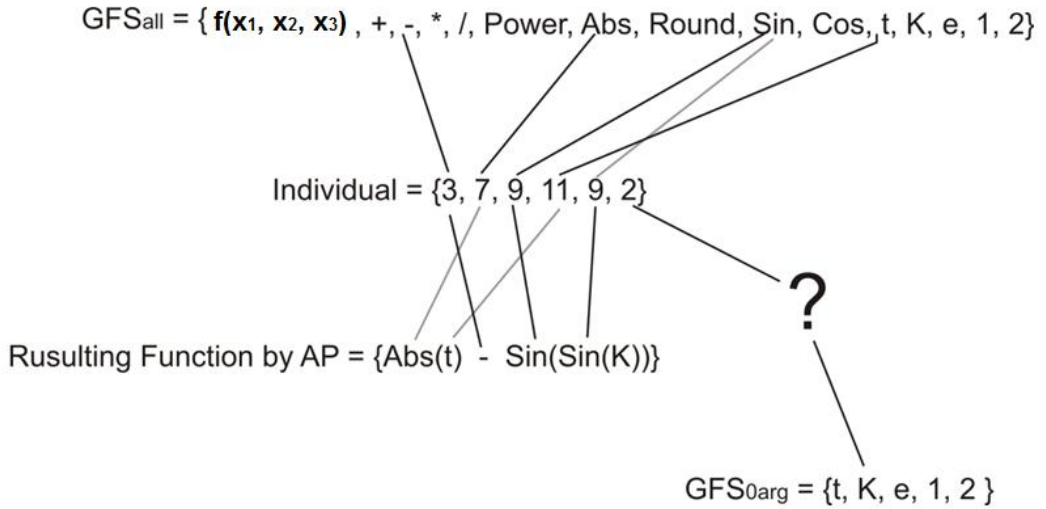


Fig. 16: Main principles of AP

The individual consists of 6 arguments (indices, pointers to GFS). The first index is 3, meaning that it is taken from the set of functions  $GFS_{all}$ . The function *minus* has two arguments; therefore indexes 7 and 9 are arguments of *minus*.

$$6 + 7 \tag{6}$$

Index 7 is then replaced by *Abs* and index 9 by *Sin*.

$$Abs + Sin \tag{7}$$

*Abs* and *Sin* are one-argument functions. Then, index 9 follows index 11, which is replaced by *t*.

$$Abs(t) + Sin \tag{8}$$

*Sin* is also a one-argument function. Then, after index 11, the individual takes index 9, which is replaced by *Sin* and this *Sin* becomes an argument of the previous *Sin*.

$$Sin(Tan) + Sin(Sin( \tag{9}$$

The last index is 2, but in this case there is the function *Plus*. *Plus* needs two arguments to work properly. AP will not allow this, as there is not any other free pointer to be used as the argument. Instead of *Plus*, AP will jump into the subspace, in this case

directly to the  $GFS_{0arg}$ . In the  $GFS_{0arg}$  it finds the second element, which is  $K$ . And by doing so, we get (10).

$$Abs(t) + Sin(Sin(K)) \tag{10}$$

The number of pointers actually used from an individual before the synthesized expression is closed is called *depth*. This example is based on the relevant and previously published work in [23].

## 5 NEURAL NETWORKS OPTIMISATION

Artificial neural networks are a widely used tool for nonlinear modeling, function approximation, prediction, classification and association [34], [35], [36]. This thesis is focused specifically on the feed forward ANN (see Fig. 17).

The network function  $f(x)$  is defined as a composition of other functions  $g_i(x)$  which can further be defined as a composition of other functions. This can be conveniently represented as a network structure, with arrows depicting the dependencies between variables. The widely used type of composition is the nonlinear weighted sum,

$$f(x) = F\left(\sum_i w_i g_i(x)\right) \quad (11)$$

where  $F$  (commonly referred to as the activation function) is some predefined function, such as the hyperbolic tangent (see (17)). [38]

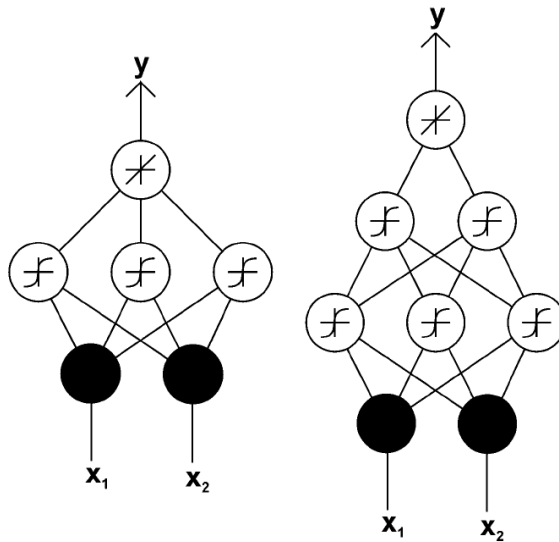


Fig. 17: Example of one and two hidden layer ANN

What has attracted the most interest in ANN is the possibility of *learning* (see chapter 5.3.1). Given a specific *task* to solve, and a *class* of functions,  $F$  (in this case GFS, see chapter 5.3), learning means using a set of observation to find  $f^* \in F$  which solves the task in some optimal sense. This entails defining a Cost Function  $C:F \rightarrow \mathbb{R}$  such that, for the

optimal solution  $f^*$ ,  $C(f^*) \leq C(f) \forall f \in F$  (i.e., no solution has a cost less than the cost of the optimal solution). [39]

## 5.1 Evolutionary Designed Neural Network

The development of evolutionary methods aiming to design the ANN structure and weight values experienced boom at the end of the millennium with the introduction of sufficiently fast computers into common scientific practice. A comprehensive survey considering history of evolutionary computation methods of designing the ANN structure can be found in [40].

According to [41] these methods can be used in the field of the ANN in several ways:

- to train the weights of the ANN
- to analyze the ANN
- to generate the architecture of the ANN
- to generate both the ANN's architecture and weights

The problem often encountered with GA is that they are quite slow in fine-tuning once they are close to a solution. Therefore, hybridization of GA and back propagation algorithm [42] (BP), where BP is used to fine-tune a near-optimal solution found GA, has proven to be successful [43]. In [44] a GA is used to evolve the ecological ANN that can adapt to their changing environment. This is achieved by letting the fitness function, which in this case is seen as individual for every gene, to co-evolve with the weights of the ANN.

De Garis [45] uses a method, which is based on the fully self-connected ANN modules. It is shown that by using this approach a network can be taught a task even though the time-dependent input changes so fast that the ANN never settles down.

In [46] and [47], GA is used in a fixed three layer feed forward ANN to find the optimal mapping from the input to a hidden layer. It is suggested that the hidden target space might have more optima than the weight space and that finding the optimum will therefore be easier.



[48], [49] and [50] used chromosomes with real-valued genes instead of binary coded chromosomes. Satisfactory results are reported using a Genitor type Steady State Genetic Algorithm with relatively small population size of 50.

An alternative approach is to use GA, where the topology and weights are encoded as variable-length binary strings [51]. In [52] a structured GA is used that simultaneously optimizes the ANN topology and the values of weights.

In [53] feed forward ANN are generated with GA, using a direct encoding scheme where every gene in a chromosome represents a connection between two neurons. This Approach is also known as *restrictive mating*. [54]

Jacob and Rehder [55] use a grammar-based genetic system, where the topology creation, neuron functionality and weight creation are split into three different modules, each using a separate GA. Similarly, Happel and Murre [56] report an approach, where modular ANN are generated using the direct encoding scheme.

Angeline et al. [57] implemented a system based on evolutionary programming where ANN evolve using both parametric mutation and structural mutation and in [58] evolutionary programming is used where the initial network is a three-layered fully connected feed forward ANN and the evolutionary programming algorithm is used to prune the connection.

In [54] a modular design approach is used, where a distinction is made between the structure, connectivity and weights optimization. Kitano [59], [60] uses a GA-based matrix grammar approach with chromosome code grammar rewriting rules that can be used to build a connectivity matrix. Gruau [61], [62] uses a graph grammar system called Cellular Encoding. The graph grammar rules work directly with neurons and their connections and include various kinds of cell divisions and connection pruning rules. Boers and Kuiper [63] use a graph grammar system based on a class of fractals called L-system. The chromosomes used in the GA code the production rules in this grammar.

In [64] and [65] a quite different approach is presented. The ANN is used to model organisms living in a two-dimensional world in which they can move in search for food and water.

## 5.2 ANN Generating by Genetic Programming

GP offers an approach to the direct encoding scheme. The approach that consists of directly encoding ANN in the genetic tree structure used by GP is described in [24].

According to [25] the ANN topology as well as the values of the weights are defined within one structure and no distinction is made between learning of the ANN topology and its weights. The terminal set is made up of the data inputs to the network (D) and a random floating point constant atom (R). This atom is the source of all the numerical constants in the ANN and these constants are used to represent the values of the weights.

$$T = \{D, R\} \quad (12)$$

[25] also proposed a function set F consisting up to six functions;  $F = \{AN, W, +, -, *, \%, \}$  however [41] proves that GP works much better for

$$F = \{AN, W\} \quad (13)$$

where the arithmetic functions are omitted.

The title given to this implementation of the ANN design using GP is the GPNN [65]. An example of a chromosome generated by the GPNN is the following ANN, which perform the XOR function (see also chapter 11.1).

$$\begin{aligned} & ( AN ( W ( AN ( W -0.65625 D1 ) ( W 1.59375 D0 ) ) 1.01562 ) ) \\ & ( W 1.45312 ( P ( W 1.70312 D1 ) ( W -0.828125 D0 ) ) ) ) \end{aligned} \quad (14)$$

The graphical representation of (14) and the corresponding ANN are shown in Fig. 18. In a similar way, the GE can be used to successfully design the ANN. Such an approach can be found for example in [66] or [67].

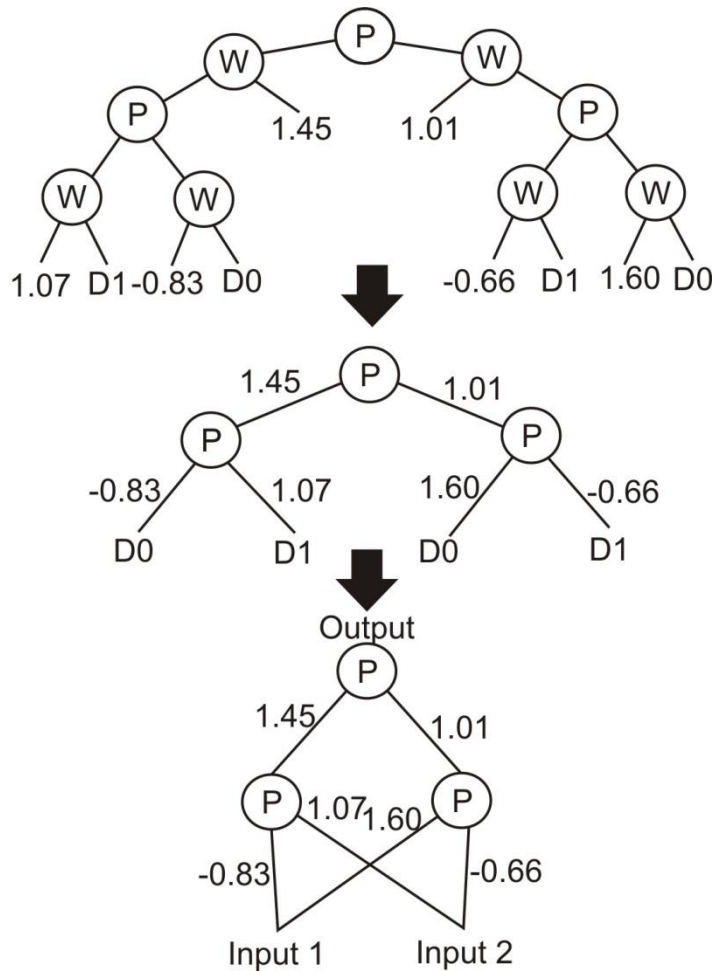


Fig. 18: Translation of a GP chromosome into ANN

### 5.3 Neural Network Synthesis

Development of the ANN synthesis as a successfully and effective method for the ANN designing is the main aim of the thesis. This chapter explains what can be understood under the term ANN synthesis and how the method works.

**Clause:** Let there be a set of all neural networks with a forward running propagation  $ANN_{all} = \{ANN_1, ANN_2, \dots, ANN_i, \dots\}$  and a set of all functions  $F_{all} = \{f_1, f_2, \dots, f_k, \dots\}$ . Then for each  $ANN_i \in ANN_{all}$  there exists a function  $f_k \in F_{all}$ , alternatively a set of functions  $F_k \subset F_{all}$  such, that holds  $ANN_i \Leftrightarrow f_k$ , alternatively  $ANN_i \Leftrightarrow F_k$ .

**The Kolmogorov theorem** further shows the validity of the inverse clause: *For every continuous function  $f_k \in F_{all}$  there exists  $ANN_i \in ANN_{all}$  such, that holds  $f_k \Leftrightarrow ANN_i$ .*

**Task:** *Design an algorithm, which will by the means of the symbolic regression methods, evolutionarily scan a set  $F_{all}$  in order to find:*

- a)  $f_k \Leftrightarrow ANN_i$
- b)  $f_k$ , whose at least some subfunctions  $\{f_1, f_2, \dots\} \Leftrightarrow \{ANN_n, ANN_m, \dots\}$

*which solves the particular problem  $P$  with a global error  $E_T < \xi$ , where  $\xi$  is the user defined biased tolerance threshold.*

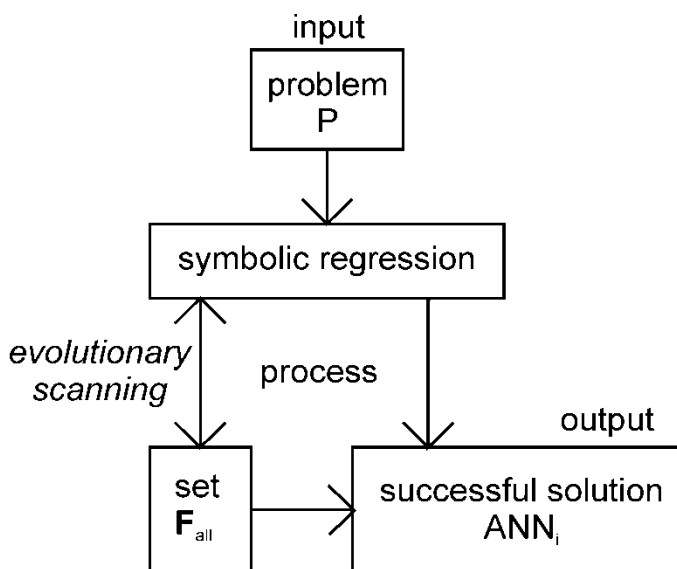


Fig. 19: Principle of the evolutionary scanning

AP can perform such evolutionary scanning above  $F_{all}$  set and provide the possibility to synthesize the ANN with an almost infinitely variable structure, complexity and scope. There is a very easy way of using AP for the ANN synthesis. [68] The most important part is to define items of which the ANN will be composed. In this case the GFS contains only three items.

$$GFS_{all} = \{+, AN, K*x\} \quad (15)$$

Most important item of (15) is an Artificial Neuron (AN) (16) with a weighted hyperbolic tangent as a transfer function (17). The weight of output, steepness and thresholds are computed as K in AP (18).

$$GFSI = \{AN\} \quad (16)$$

$$AN(S) = w \frac{e^{2\lambda(S+\phi)} - 1}{e^{2\lambda(S+\phi)} + 1} \quad (17)$$

$$AN(S) = K_1 \frac{e^{2K_2(S+K_3)} - 1}{e^{2K_2(S+K_3)} + 1} \quad (18)$$

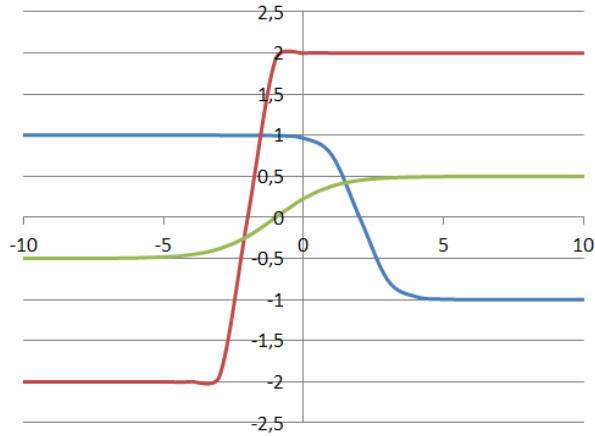


Fig. 20: AN transfer function for various  $w$ ,  $\lambda$ ,  $\phi$  settings

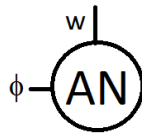


Fig. 21: Graphical example of AN

To allow more inputs into one ANN a simple plus operator (19) is used.

$$GFS_2 = \{+\} \tag{19}$$

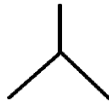


Fig. 22: Graphical example of plus operator

Finally, (20) represents the weighted input data.

$$GFS_0 = K*x \tag{20}$$

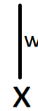


Fig. 23: Graphical example of weighted input

Under such circumstances, translation of an individual into the ANN can be easily grasped from Fig. 24.

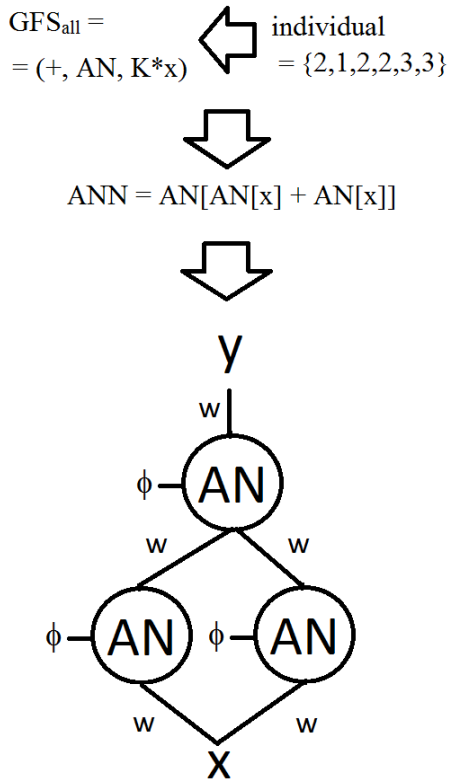


Fig. 24: Translation of an individual into ANN

The whole process is cyclical. Individuals provided by the EA are translated into ANN. ANN are evaluated in accordance with a training data set and their global errors are used to set the fitness of these individuals. Consequently, a new generation is chosen and the whole process is repeated in the next migration loop.

The introduced approach is not the only one possible. Different settings of the GFS were successfully used to synthesize the ANN performing classification. [33]

### 5.3.1 Constant Processing

The synthesized ANN, programs or formulas may also contain constants “K”, which can be defined in the  $GFS_0$  or be a part of other functions included in the  $GFS_{all}$ . When the program is synthesized, all Ks are indexed, so  $K_1, K_2, \dots, K_n$ , are obtained and then all  $K_n$  are estimated. Several versions of AP exist in accordance with  $K_n$  estimation. [32] In most cases, the Nonlinear Regression of Toolbox of Mathematica software is used. This approach provides fast results; however, the source code of this toolbox is not an open source and its inner function is not sufficiently clarified, so the resulting algorithm is not fully described and this solution cannot be used without the Mathematica software [23].

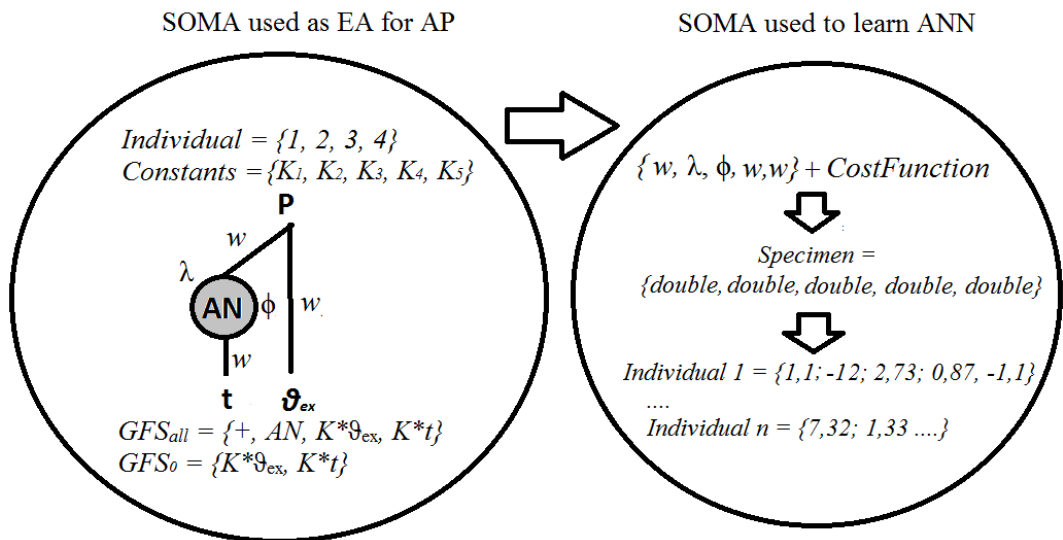


Fig. 25: Learning of a synthesized ANN

In this case, the asynchronous implementation of SOMA (inside another SOMA, which operates AP) is used to estimate  $K_n$ . This is especially convenient for the ANN synthesis.  $K_n$  can be referred to as various weights and thresholds and their optimization by SOMA as ANN learning (see Fig. 25). [33]

### 5.3.2 Reinforced Evolution

The Reinforced Evolution is a common part of AP. [32] If the ANN of adequate quality cannot be obtained during AP run, AP puts the best ANN it found as a sub ANN into the  $GFS_0$  and starts over.

This arrangement considerably improves AP ability to find the ANN with desirable parameters. For the purpose of this thesis one AP between the GFS reinforcements is called an *evolution loop*. The term evolution loop should not be mistaken for the *migration loop*. (For the migration loop see the chapter 3.4.3.)

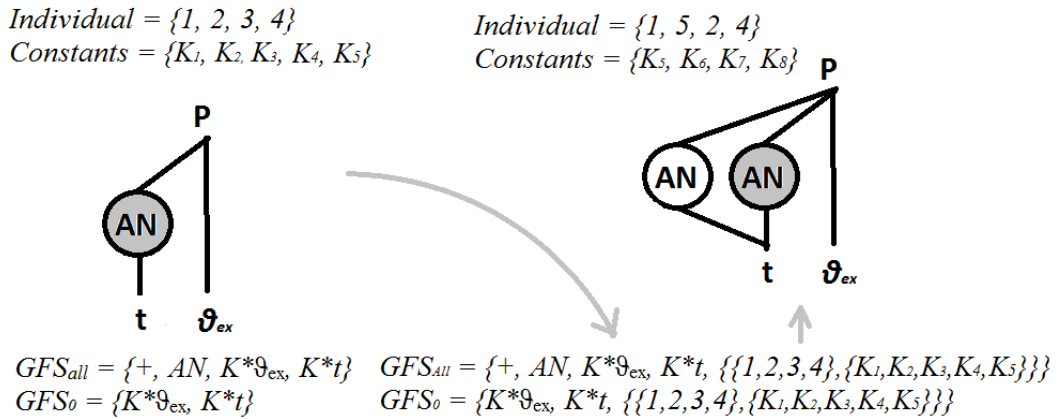


Fig. 26: Example of GFS reinforcement process between two ANN evolution loops

### 5.3.3 Cost Function Specification

As the synthesized ANN are produced to the EA, which needs to evaluate them, the Cost Function (CF) has to be specified prior to the beginning of the synthesis.

The CF specification depends on the purpose of the ANN synthesis (the solved problem) and differs for approximation (chapter 8) or prediction (chapter 10) and classification (chapter 11).



In the case of prediction or approximation, the CF can be defined as a Root Mean Square Divergence (RMSD) (21) or a Normalized RMSD (22) (NRMSD).

$$RMSD(\theta_1, \theta_2) = \sqrt{\frac{\sum_{i=1}^n (ANN(X_i) - y(X_i))^2}{n}} \quad (21)$$

$$NRMSD = \frac{RMSD}{y_{\max} - y_{\min}} \cdot 100\% \quad (22)$$

For classification tasks, the CF has to be designed as (23). To favor smaller ANN, the fraction *depth* parameter (for *depth* meaning see chapter 4.3) of an individual can be added.

$$CF = \text{number of wrongly classified examples} + \text{depth}/100; \quad (23)$$

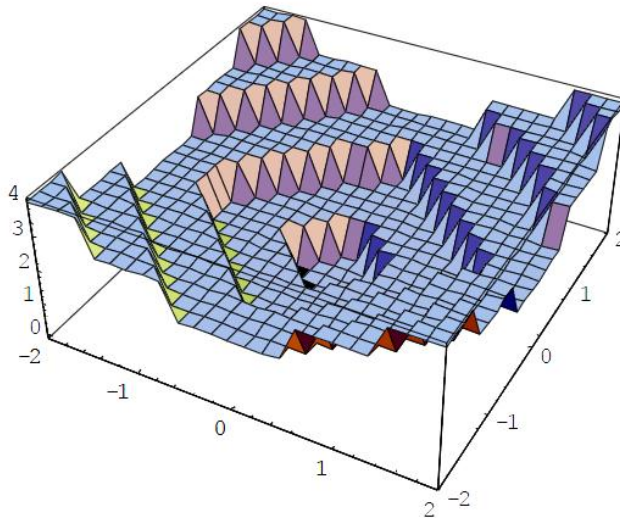


Fig. 27: Example of CF for classification

To obtain a smoother profile of the CF (23) can be developed into (24).

$$CF = \sum_{i=1}^n \begin{cases} 0 & \text{if (sample}_i \text{ is classified correctly)} \\ \rho(\text{ANN}[\text{sample}_i], \text{correct class border}) + 1 & \text{else} \end{cases} \quad (24)$$

For the practical implementation of (24) see (87) in chapter 12.

## **6 DISTRIBUTED COMPUTATION**

The basic idea of most parallel programs is to divide a task into chunks and to solve the chunks simultaneously using multiple processors. This divide-and-conquer approach can be applied to the EA in many ways and literature contains an inexhaustible number of examples of successful parallel implementations. Some parallelization methods use a single population, while others divide the population into several relatively isolated subpopulations. Some methods can massively exploit parallel computer architectures, while others are better suited for computers with fewer but more powerful CPUs. [69]

A comprehensive survey of the EA distribution can be found in [70] together with two following successful Island Model [71] SOMA distributions.

### **6.1 Island Distribution of SOMA**

This approach suits parallel SOMA running in the above described cluster platform very well. At each computation node, a randomly initialized subpopulation is created according to the configuration given by a master node. The node performs one SOMA migration and sends a local leader to the server.

#### ***6.1.1 Synchronous Island Model***

When the migration loop is done on all terminals, the server compares cost values of all received local leaders and chooses a global leader. This leader is then sent back to the terminals and replaces the worst individual in local populations. This process is repeated until the termination conditions are satisfied.

#### ***6.1.2 Asynchronous Island Model***

To avoid time delays in the former parallelization approach, the synchronism of sharing and selecting the best individual was removed. When the terminal finishes its migration loop, the local leader is sent to the server. The task of the master node is to maintain the global leader – every time it receives the leader from a subordinated node, it compares its cost value with the value of the global leader and stores the better one.

Consequently, the global leader is passed back to the terminal node, where next migration loop is started. Again, this process is repeated until stop conditions are met. This parallelization approach is also used outside the cluster platform.

## 6.2 Direct Asynchronous Distribution of SOMA

Chapter 5.3 explains the process during which a huge number of very different ANN can be synthesized. Therefore, an actual population, which needs to be evaluated, contains individuals with various numbers for  $K_n$ . This means that the algorithm is very time demanding and furthermore, computation of every individual consumes different amounts of computation time. [3]

Fortunately, in these days, standard computers are more often equipped with more than one processor. However, if the individuals are evenly divided between available processors for every migration loop, the large amount of computation time is lost due to their unevenly distributed complexity.

To overcome this set-back, a small but very important change to SOMA mechanism was made inspired by the Asynchronous Island Model (chapter 6.1.2). The individuals no longer work in the migration loops (see. chapter 3.4.3). **On the contrary:**

**Every individual is compared with the Leader just after it finishes its jumping and a new Leader is selected immediately** (25)

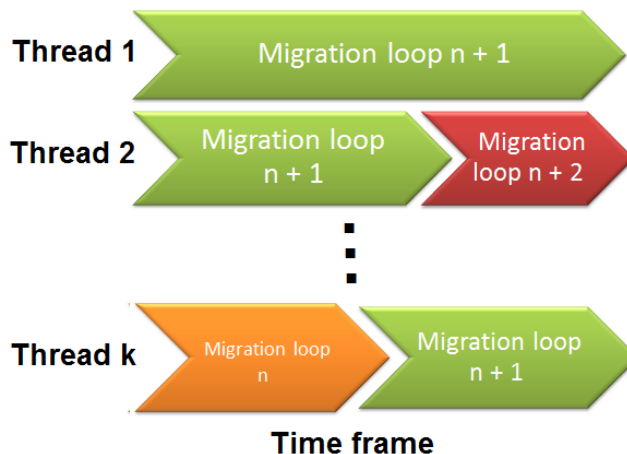


Fig. 28: Asynchronous processing of a migration loop by different threads

This makes SOMA distribution work asynchronously. All the individuals do their migrations independently and some may even move much faster than others.

The main reason why SOMA as an algorithm is especially convenient for the direct distribution approach lies in the fact that every individual needs to communicate with the leader only once per twenty-seven evaluations of the CF (depending on SOMA control parameters), so the amount of information transferred between individuals during computation is relatively low in comparison with other GA as can be seen in Fig. 29.

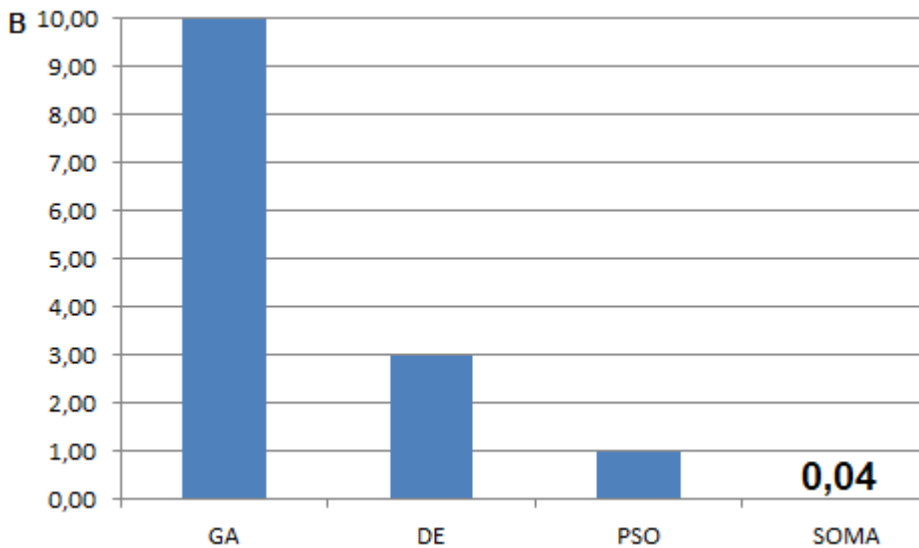


Fig. 29: Proportional comparison of transferred information amounts needed for one CF computation within different EA

As there is no synchronization point anymore to evaluate the stop condition (chapter 3.4.4), the condition is evaluated once after  $n$  evaluations of the Cost Function.

$$n = period * number\ of\ individuals * mass / step \quad (26)$$

The strategy proposed by (25) can result in interesting behavior of individuals provided each individual occupies its own thread or process as described in Fig. 30.

In such a case, a *huge* individual (the ANN with more AN) moves on a  $N-k$  hyper plane *slower* than *small agile* individuals (the ANN with less AN) as can be seen in Fig. 31. This can positively influence the ANN optimization.

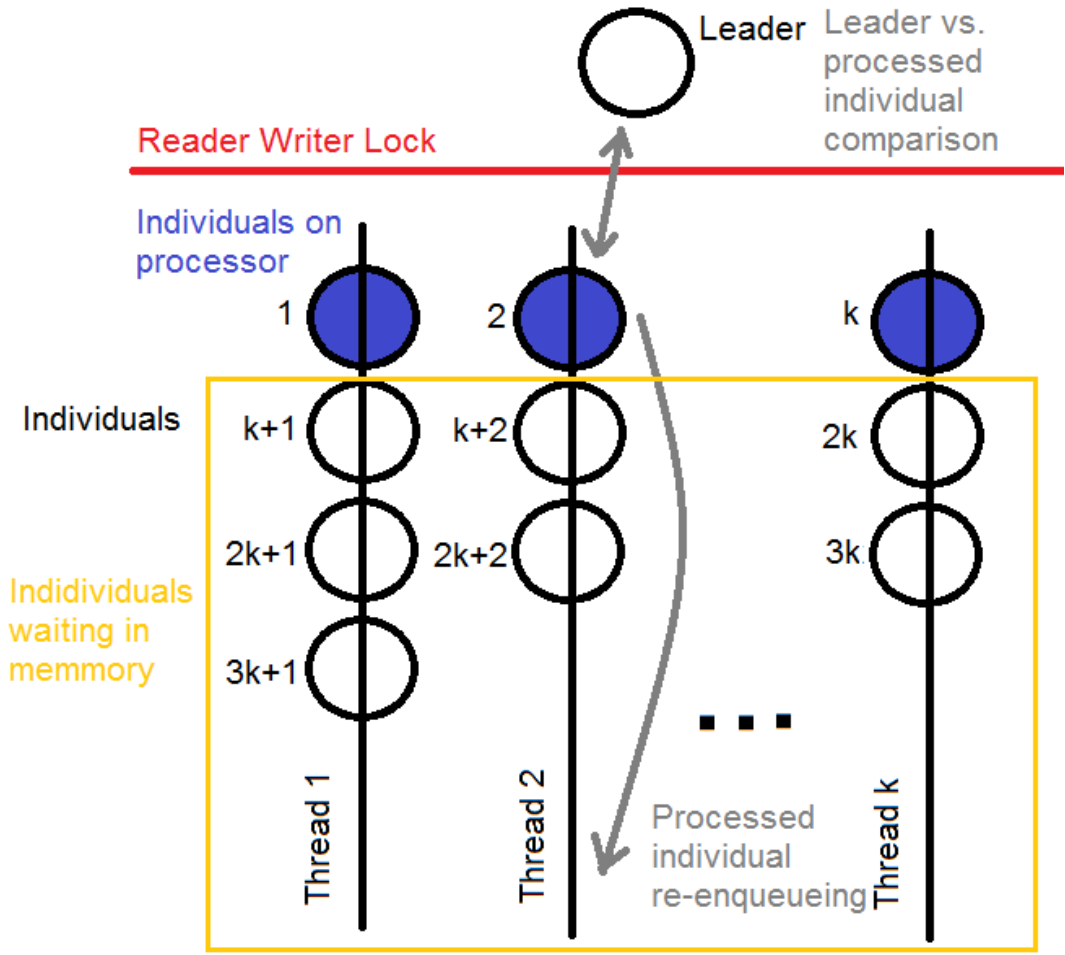


Fig. 30: Asynchronous individual processing

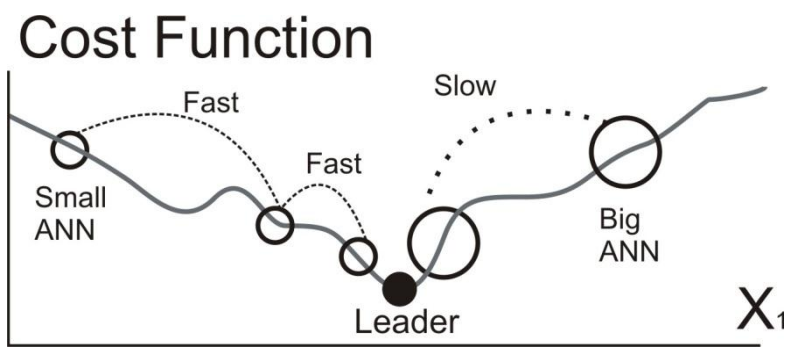


Fig. 31: Asynchronous parallel movement of SOMA individuals

## **PRACTICAL PART**

## 7 ASYNCHRONOUS SOMA PERFORMANCE

To statistically explore the efficiency of SOMA direct asynchronous distribution proposed in chapter 6.2, ten different test functions were chosen for the experiment. All these functions as well as other SOMA control parameter settings were based on [19] and used in the same way as done by prof. Zelinka when initially testing the SOMA.

$$\sum_{i=1}^{Dim-1} \left( 20 + e - 20e^{-0.2\sqrt{0.5(x_i^2+x_{i+1}^2)}} - e^{-0.5(\cos(2\pi x_i)+\cos(2\pi x_{i+1}))} \right) \quad (27)$$

$$\sum_{i=1}^{Dim-1} \left( -x_i \sin\left(\sqrt{|x_i - (x_{i+1} + 47)|}\right) - (x_{i+1} + 47) \sin\left(\sqrt{\left|x_{i+1} + 47 + \frac{x_i}{2}\right|}\right) \right) \quad (28)$$

$$1 + \sum_{i=1}^{Dim} \frac{x_i^2}{4000} - \prod_{i=1}^{Dim} \cos\left(\frac{x_i}{\sqrt{i}}\right) \quad (29)$$

$$- \sum_{i=1}^{Dim-1} \left( e^{\frac{-(x_i^2+x_{i+1}^2+0.5x_i x_{i+1})}{8}} \cos\left(4\sqrt{x_i^2 + x_{i+1}^2 + 0.5x_i x_{i+1}}\right) \right) \quad (30)$$

$$\sum_{i=1}^{Dim-1} \left( -1 \left( \sin(x_i) \sin\left(\left(\frac{x_i^2}{\pi}\right)^{20}\right) + \sin(x_{i+1}) \sin\left(\left(\frac{2x_{i+1}^2}{\pi}\right)^{20}\right) \right) \right) \quad (31)$$

$$\sum_{i=1}^{Dim-1} (x_i \sin(a) \cos(b) + (x_{i+1} + 1) \sin(a) \cos(b)) \quad (32)$$

where  $a = \sqrt{|x_{i+1} + 1 - x_i|}$  and  $b = \sqrt{|x_{i+1} + 1 + x_i|}$

$$10Dim \sum_{i=1}^{Dim} (x_i^2 - 10\cos(2\pi_i)) \quad (33)$$

$$\sum_{i=1}^{Dim-1} (100(x_i^2 - x_{i+1}^2)^2 + (1 - x_i^2)^2) \quad (34)$$

$$\sum_{i=1}^{Dim-1} -x_i \sin\left(\sqrt{|x_i|}\right) \quad (35)$$

$$-\sum_{i=1}^{Dim-1} \left(x_i^2 + x_{i+1}^2\right)^{0.25} \left(\sin\left(50\left(x_i^2 + x_{i+1}^2\right)^{0.1}\right)^2 + 1\right) \quad (36)$$

PopSize = 60, PathLength = 3, Step = 0.11, PRT = 0.1 and a number of parameters = 100 are constant for all these functions. The number of migration loops and borders of the function's parameters vary in accordance with Table 8.

2D and 3D visualizations of test functions (27) - (36) are available in Appendix I.

*Table 8: Test functions, ML and borders*

Function	ML	Low	Hight
Ackley (27)	400	-30	30
EggHolder (28)	800	-512	512
Griewangk (29)	200	-100	100
Masters (30)	400	-5	5
Michalewicz (31)	200	0	3,1415
Rana (32)	125	-500	500
Rastrigin (33)	400	-5,12	5,12
Rosenbrock (34)	125	-2,048	2,048
Schwefel (35)	400	-512	512
SineWave (36)	400	-10	10

Every test function was optimized 100 times by linear SOMA and 100 times by SOMA direct asynchronous parallelization (25) distributed among 8 independent processors of a Super Micro server (see Appendix IV). In total  $1.1 * 10^9$  Cost Function evaluations were computed during 2,000 separate SOMA runs.

## 7.1 Results

Table 9 and Table 10 show the average and best results from the previously proposed experiment for all test functions (27) to (36).



Table 9: SOMA average results

Function	Linear	Asynchronous
Ackley (27)	3368,098	3370,429
EggHolder (28)	-63855,5	-63605,3
Griewangk (29)	0,872625	0,885898
Masters (30)	-77,7542	-77,9066
Michalewicz (31)	-97,9913	-97,6051
Rana (32)	-21400,8	-21486,4
Rastrigin (33)	-958457	-950512
Rosenbrock (34)	335,933	369,7812
Schwefel (35)	-40531,1	-40241,7
SineWave (36)	-519,919	-518,877

Table 10: SOMA best results

Function	Linear	Asynchronous
Ackley (27)	3366,142	3366,184
EggHolder (28)	-68130,2	-67268,8
Griewangk (29)	0,625381	0,580481
Masters (30)	-83,9028	-84,3382
Michalewicz (31)	-98,9955	-99,0905
Rana (32)	-24067	-25117,7
Rastrigin (33)	-977084	-968021
Rosenbrock (34)	234,7636	254,7049
Schwefel (35)	-41423,7	-41305,1
SineWave (36)	-530,483	-530,24

SOMA parallelization (25) introduced in chapter 6.2 proved to be highly effective. Considering the average results (25) was in 2 cases better than linear. Furthermore, for the best results (25) was better in 4 cases and its efficiency is almost 100%.

(25) efficiency for the average results is 98.6%, which is even better than the efficiency of the asynchronous island model published in [70]. (25) excellent performance convincingly demonstrates its usage as the EA for AP.

## 8 ANN SYNTHESIS FOR FUNCTION APROXIMATION

In order to statistically evaluate the ANN synthesis' ability to successfully solve the function approximation problem (chapter 5.3), the function (37) proposed by [24] as an appropriate approximation benchmark was chosen to be approximated by the ANN.

$$y = x_i^5 - 2 x_i^3 + x_i \tag{37}$$

where  $x_i \in <-1, \text{by the step } 0.04, 1>$

Fig. 32 (automatically generated by ANN synthesis software, see chapter 12) shows an example of synthetized ANN approximating (37). The difference between the ANN and (37) is depicted as a red area which could be minimized by the process of synthesis.

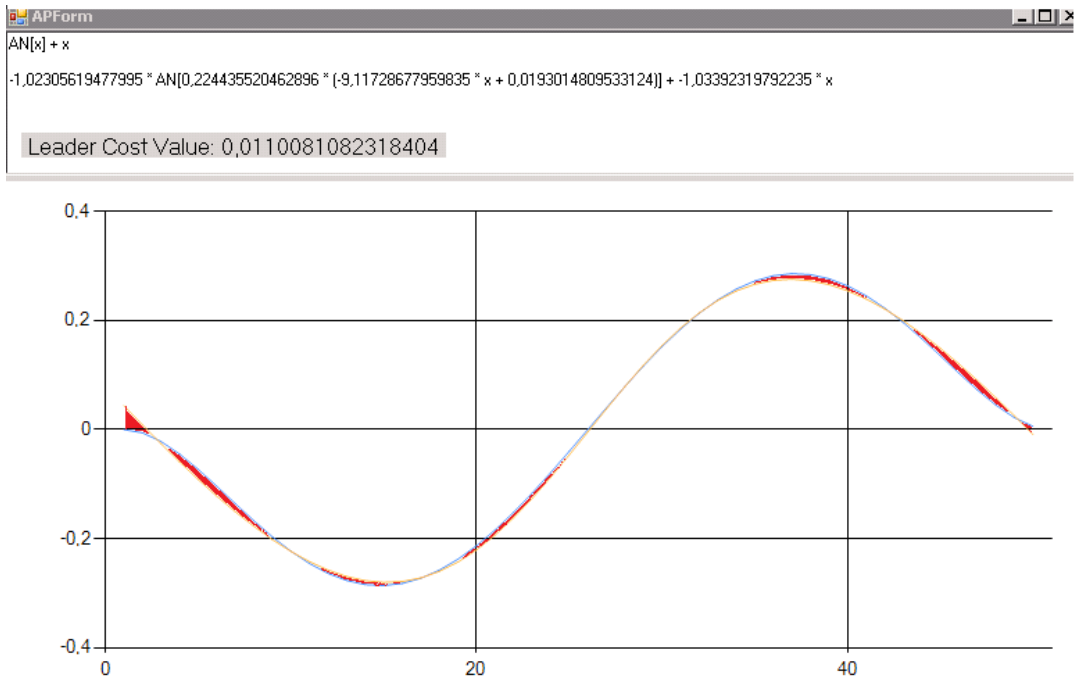


Fig. 32: Approximation of (37) by synthetized ANN

The CF of the synthetized ANN is mathematically formulated in accordance with the chapter 5.3.3 as a RMSD (21).

AP was executed 100 times (physically on 8 cores of the Super Micro Server, see Appendix IV) to produce an ANN with the RMSD < 0.005. The main intention was to

find such an ANN which met this condition and which simultaneously used as few AN as possible.

The setting of Asynchronous SOMA used as the EA for AP can be seen in Table 11 and SOMA setting used for ANN learning in Table 12.

*Table 11: Setting of SOMA used as EA for AP*

<b>Number of Individuals</b>	48
<b>Individual Parameters</b>	100
<b>Low</b>	0
<b>High</b>	3
<b>PathLength</b>	3
<b>Step</b>	0,11
<b>PRT</b>	1/ depth
<b>Divergence</b>	0.01
<b>Period</b>	1

*Table 12: Setting of SOMA used to optimize  $K_n$*

<b>Number of Individuals</b>	number of $K_n * 0.5$ (at least 10)
<b>Individual Parameters</b>	100
<b>Low</b>	-10
<b>High</b>	10
<b>PathLength</b>	3
<b>Step</b>	0,11
<b>PRT</b>	1 / number of $K_n$
<b>Divergence</b>	0.01
<b>Period</b>	6

## 8.1 Results

A total of 921,937 evaluations of AP individual fitness was done during 100 AP executions and a separate SOMA run was performed for all of them to set their  $K_n$  value. The time needed for all these evaluations was approximately 5 hours and 24 minutes.

The average time for 1 evaluation was 558 ms, however  $t_{\max} = 136,369$  ms while 98% of measured times  $t < t_{\max} / 10$ . Such results prove that the vast amount of the computation time can be saved by asynchronous distribution (26) (see chapter 6.2). The way these values increase with a growing number of processors used is described in Table 13 and Fig. 33.

Table 13: Time saved by asynchronous evaluation

Number of processors used	Percentage of saved time
2	23,7 %
4	47,3 %
8	67,2 %
16	81,2 %

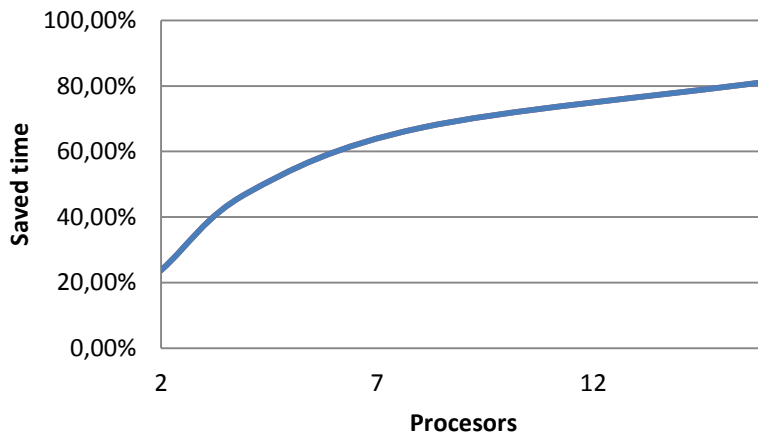


Fig. 33: Time saved by asynchronous evaluation

All 100 AP runs successfully synthesized the ANN with the RMSD < 0,005. The average number of the AN used was 9. Nevertheless, the optimization task in order to find the ANN with the lowest number of AN was the most successful in 4 cases, which employed only 2 AN. All these cases led to a similar ANN structure.

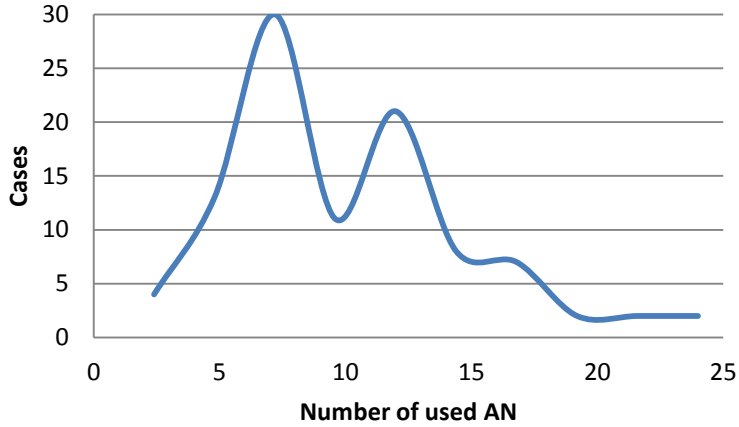


Fig. 34: Synthesized ANN according AN usage

The example of a successfully optimized ANN is shown here as (38) and its sub ANN as (39):

$$ANN0 = x + AN[x] \tag{38}$$

$$ANN1 = ANN0 + AN[ANN0] + ANN0 \tag{39}$$

After successful optimization of  $K_n$  by SOMA (38), (39) lead to (40), (41).

$$ANN0 = -0,972628914257888 * x + 0,960043432203328 * AN[0,303565531015147 * (7,00172920571721 * x + -0,00454216333835794)] \tag{40}$$

$$ANN1 = ANN0 + 0,40897485611192 * AN[-2,77100775198393 * (ANN0 + 0,000305134718869929)] + ANN0 \tag{41}$$

(42) and (43) translated (38), (39) into the mathematical formulation.

$$ANN0 = w_2 \frac{e^{2\lambda_1 (w_1x + \phi_1)} - 1}{e^{2\lambda_1 (w_1x + \phi_1)} + 1} + w_3x \quad (42)$$

$$ANN1 = w_5 \frac{e^{2\lambda_2 (w_4(w_2 \frac{e^{2\lambda_1 (w_1x + \phi_1)} - 1}{e^{2\lambda_1 (w_1x + \phi_1)} + 1} + w_3x) + \phi_2)} - 1}{e^{2\lambda_2 (w_4(w_2 \frac{e^{2\lambda_1 (w_1x + \phi_1)} - 1}{e^{2\lambda_1 (w_1x + \phi_1)} + 1} + w_3x) + \phi_2)} + 1} + w_2 \frac{e^{2\lambda_1 (w_1x + \phi_1)} - 1}{e^{2\lambda_1 (w_1x + \phi_1)} + 1} + w_3x \quad (43)$$

The ANN described as (38), (39) can be graphically interpreted (Fig. 35).

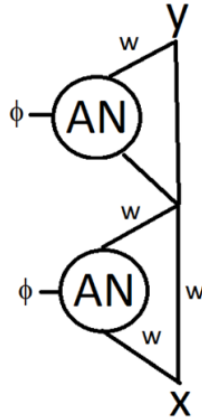


Fig. 35: Graphical interpretation of resulting ANN

## 8.2 Conclusion

Asynchronous distributions (25) proved to be crucially important for the successful AP implementation. For example, if 8 processors are used (as they were in the experiment), more than 67% of computation time (which would be wasted otherwise) can be saved. With respect to the experiment, approximately 3 hours of computation time were saved.

AP also exercised the ability to synthesize the ANN affectively and quickly with the help of asynchronous SOMA distribution (0.5 s for 1 ANN on average).

The very small ANN containing only two ANs was automatically found and solved the given problem with the satisfactory RMSD. This success reveals AP as an exceptionally useful tool for ANN synthesis and optimization. [72]

## 9 ANN SYNTHESIS STRATEGY EXPLORATION

To boost the AP performance and to obtain better and faster results of the ANN synthesis, two adaptive approaches of the PRT setting were developed (chapters 9.1.2 and 9.1.3) and applied in the experiment described in chapter 8. In order to measure their impact on the AP performance statically the very same experiment was performed without the proposed improvements and subsequently compared with the original results.

Chapters 9.2 and 9.3 explore the asynchronous SOMA dependence on the ANN synthesis efficiency and the possibility of other EA employment.

### 9.1 Adaptive PRT Strategy

The PRT adaptive approach is possible only with the application of (4) discussed in chapter 3.4.6. Experimental confirmation of such an approach is statistically processed in chapter 9.1.1.

#### 9.1.1 Adaptive PRT Strategy Test on Test Functions

The following experiment was designed to explore SOMA efficiency for  $\mathbf{PRT} \in (0; 0.1>$  and compare it with results obtained for  $\mathbf{PRT} \in <0.1; 0.3>$ . In other words, this experiment measured the dependence of  $P_1$  on SOMA behavior (see chapter 3.4.6).

The main aim of the experiment was to statistically approve the widening of the PRT parameter into  $\mathbf{PRT} \in (0; 1>$  as a necessary assumption for strategies applied in chapters 9.1.2 and 9.1.3.

Test functions (26) - (37) were chosen for the experiment.  $\text{PopSize} = 60$ ,  $\text{PathLength} = 3$ ,  $\text{Step} = 0.11$  and the number of parameters = 100 are constant for all of these functions. (see also Table 3) The number of migration loops and borders of the function's parameters vary in accordance with Table 8.

For each test function, the optimization (the search for a global minimum) via SOMA was repeated 100 times for different  $\mathbf{PRT} = \{0.005, 0.01, 0.03, 0.05, 0.07, 0.1, 0.2, 0.3\}$ . The overall 8000 repetitions were made (*test functions \* PRT variants \* 100*).

996.3 \* 10<sup>6</sup> evaluations of the Cost Function were computed in total.

$$evaluations = (Round(PathLength/Step) * ML * PopSize * 100 * test\ functions) \quad (44)$$

(4) was applied in all cases.

The final results were normalized: The best case for the given test function was set as 0 (base) and all other cases were expressed as percent divergence.

Fig. 36 and Fig. 37 graphically show the values describing SOMA behavior based on various test functions and **PRT** settings. More specific results are included in Appendix II (see Fig. 90, Fig. 91 and Table 20, Table 21, Table 22, Table 23).

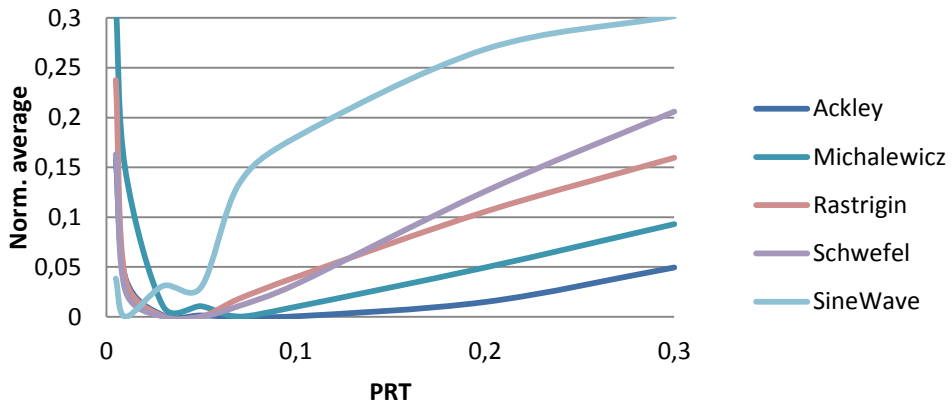


Fig. 36: Test functions performing better for  $PRT \in \langle 0.005; 0.7 \rangle$

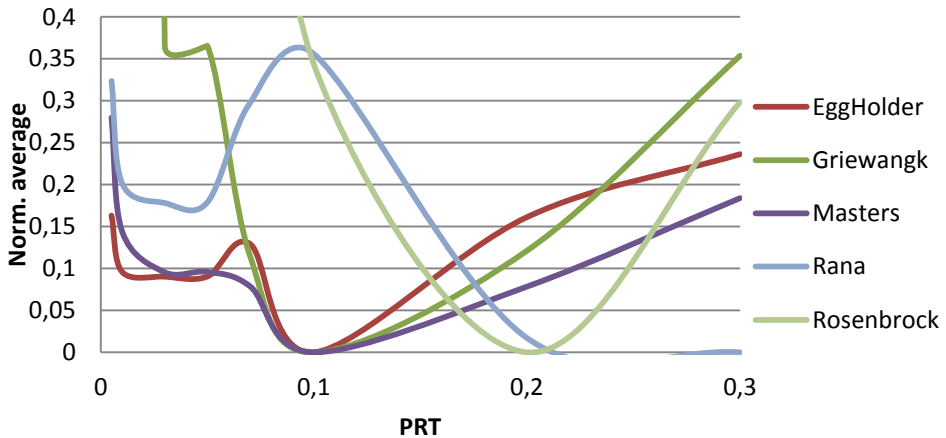


Fig. 37: Test functions performing better for  $PRT \in \langle 0.1; 0.3 \rangle$



5 out of 10 test functions employed in the experiment (see Fig. 36) achieved better results for  $\mathbf{PRT} \in \langle 0.005; 0.7 \rangle$  and the other 5 (see Fig. 37) for  $\mathbf{PRT} \in \langle 0.1; 0.3 \rangle$ . This conclusion represents a significant breakthrough in the  $\mathbf{PRT}$  setting strategy. The previously recommended range  $\mathbf{PRT} \in \langle 0.1; 0.3 \rangle$  can be extended to  $\mathbf{PRT} \in \langle 0.01; 0.3 \rangle$ ; furthermore, around 50% of the functions can be optimized by SOMA more effectively if  $\mathbf{PRT} \in \langle 0.01; 0.7 \rangle$ . An increasing value of  $P_1$  can positively influence the obtained results [73]. However, SOMA efficiency always decline if  $P_1 > 0.74$ .

Based on this conclusion, strategies described in chapters 9.1.2 and 9.1.3 using  $\mathbf{PRT} \in (0; 1)$  can be recommended for the experimental validation.

### 9.1.2 Adaptive PRT Strategy for AP Handling

The adaptive strategy for AP handling consists in the replacement of a static PRT value by a value which depends inversely on the individual's depth. This approach ensures (together with (4)) that the PRT influence is projected into an active part of the individual.

Table 14: PRT strategy for AP handling

	<b>PRT = 1/depth</b>	<b>PRT = 0.1</b>
<b>Average time needed for synthesis</b>	194 s	373 s
<b>Average number of used AN</b>	9	13

A total of 1,189,870 evaluations of AP individual fitness were completed during 100 AP executions while the PRT was set to 0.1 and the separate SOMA run was performed for all of them to set their  $K_n$  value. Without the adaptive PRT, AP was able to find an optimal ANN in only 1 case in comparison with 4 successful cases in the original experiment.

### 9.1.3 Adaptive PRT Strategy for $K_n$ Estimation

The adaptive strategy for  $K_n$  estimation consists in replacement of the static PRT value by the value which inversely depends on  $K_n$  dimension with the application of (4).

Table 15: PRT strategy for  $K_n$  estimation

	<b>PRT = 1 / number of <math>K_n</math></b>	<b>PRT = 0.1</b>
<b>Average time needed for synthesis</b>	194 s	505 s
<b>Average number of used AN</b>	9	15

A total of 672,779 evaluations of AP individual fitness were completed during 100 AP executions and the separate SOMA run was performed for all of them to set their  $K_n$  value while PRT was set to 0.1 and (4) was omitted. However, under such conditions, in 7 cases AP was not able to find a sufficient ANN at all.

### 9.1.4 Conclusion

The introduced adaptive PRT strategy proved to be highly effective as its application to AP handling works 48% faster and 16% more efficiently (computed on 8 cores of the Super Micro server, see Appendix IV). For  $K_n$  the estimation considered strategy works 61% faster and 42% more efficient. Furthermore, without it, AP was not able to successfully synthesize the ANN in 7 cases.

The adaptive PRT strategy proved to be crucial for the successful ANN synthesis.

## 9.2 Comparison of synchronous and asynchronous synthesis

In order to explore the impact of individual's asynchronous movement (discussed in chapter 6.2, see especially Fig. 31) on the ANN synthesis efficiency, the experiment considered in chapter 8 was repeated with synchronous behavior of the participated EA individuals.

*Table 16: Synchronous and asynchronous SOMA performance*

	<b>Asynchronous</b>	<b>Synchronous</b>
<b>Average time needed for synthesis</b>	194 s	212 s
<b>Average number of used AN</b>	9	11

The experiment results recorded in Table 16 show that the asynchronous movement is 8.5% faster and, interestingly, the synthesized ANN lacks 2 AN on average. Such results successfully proved the adaptation of asynchronous individual behavior (chapter 6.2) into the ANN synthesis.

### **9.3 ANN synthesis running with different EA**

A comparison of different EA performances is a complex issue as each EA needs its specific control parameters settings which often vary from task to task. Nevertheless, the purpose of the experiment recorded in Table 17 is not to compare different EA with each other, but to prove that the ANN synthesis can be successfully done with the use of the DE (chapter 3.2), PSO (chapter 3.3) or SOMA.

The experiment from chapter 8 was repeated under the same conditions with the DE or PSO as AP animator as well as ANN learning tools. The recommended control parameters settings were taken from [74] for the DE and from [75] for the PSO. As parallel versions of this EA were not available, the experiment was performed on a serial version of the EA.

*Table 17: ANN performance for different EA*

	<b>DE</b>	<b>PSO</b>
<b>Average time needed for synthesis</b>	3296 s	3876 s
<b>Average number of used AN</b>	15	14

Both EA were principally able to successfully synthesize the ANN. As can be seen in Table 17 the DE is slightly faster but the PSO provides fewer ANN. Longer computational times are understandable as both EA work serially. The DE as well as PSO provide deeper ANN on average; however, any of these EA are not tuned as suitable as SOMA in chapters 6.2, 9.1.2 and 9.1.3 .

## 10 ANN SYNTHESIS FOR PREDICTION

In this chapter the ANN synthesis ability to successfully synthesize the ANN capable of prediction is tested on a real life problem of a heating plant. The method described in Chapter 5.3 is applied in order to optimize the Heat Load Prediction (HLP) function of the heating plant in Komořany (Czech Republic). The function is later used to predict the heat load of Most agglomeration in order to provide valuable information for the heating plant control.

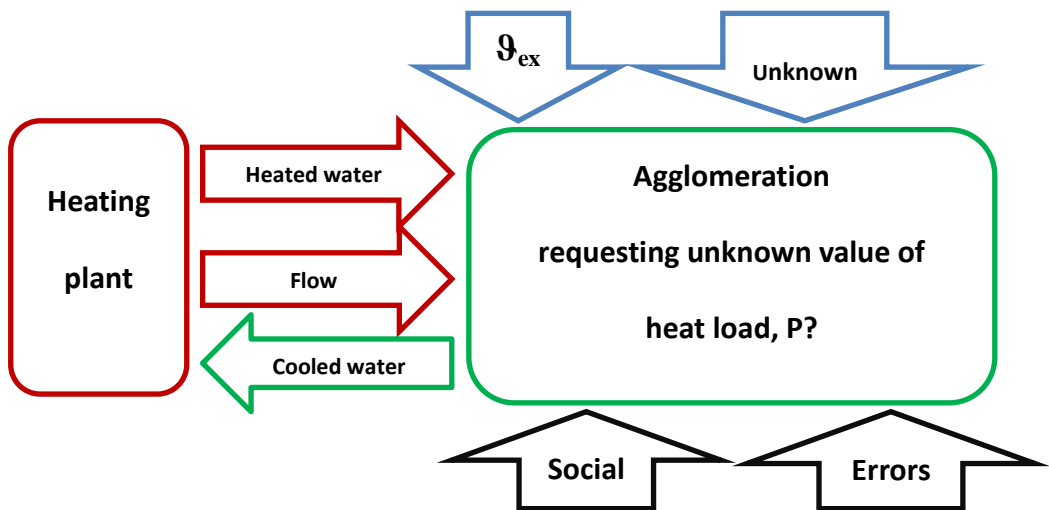
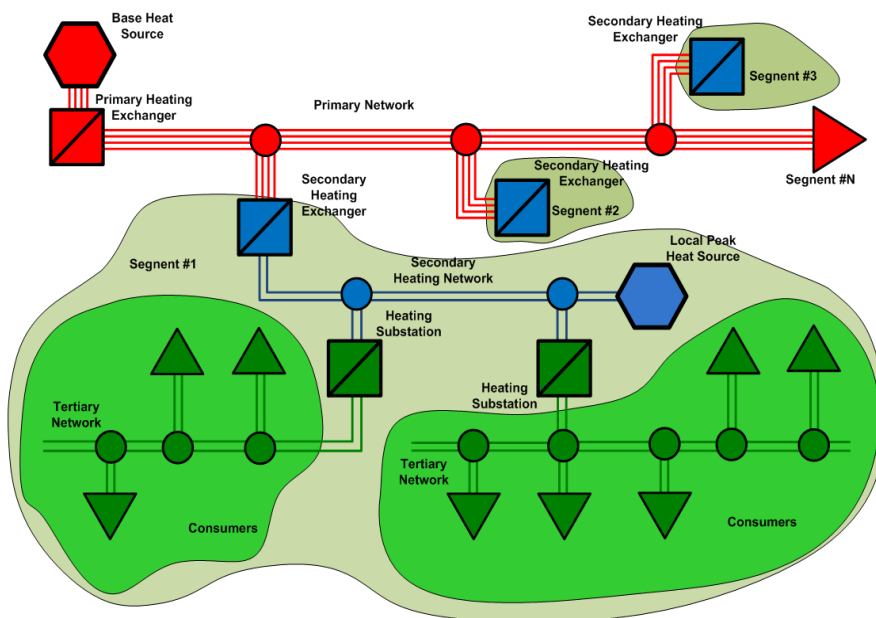


Fig. 38: Relationship between a heating plant (red), an agglomeration (green), atmospheric conditions (blue) and other events (black)

The interface between the heating plant and the agglomeration as can be seen in Fig. 28 is a highly complex system. The heating plant provides heat in the form of hot water with variable temperature and flow rates for the agglomeration while cooled water returns with a variable transfer delay. As the flow and temperature are independent variables set up by the heating plant staff or by an automatic regulator, the only unknown variable for the interface modeling is the temperature of the returned water. However, the process of prediction of the returned water temperature is not only affected by the past values of the input temperature and the flow but also by a set of various external factors. Undoubtedly, the most important value of the external factor is the past values of atmospheric temperature. Nevertheless, the weather conditions such as humidity or wind

speed should also be considered. In addition, different sociological factors, such as time when people wake-up, can also play significant roles.

In the heating plant located in Komořany, Czech Republic, owned by the United Energy a.s., the heat load is (48) and it is only predicted on the basis of the atmospheric temperature and time because the data on humidity and wind speed are not measured or are unavailable [68]. The complex situation is further complicated by the existence of the secondary and tertiary distribution networks (illustrated in Fig. 39 published in [76]) and their interactions with the primary network.



*Fig. 39: Basic scheme of the central heating plant system Komořany – Most*  
 The correct approximation of the heating power consumption, dependent on time and atmospheric temperature, is an important presumption for the heating plant’s successful control, so the HLP optimization was one of the most important tasks for the National Research Program II No. 2C06007.

As the HLP precision of standard prediction methods (described in chapter 10.1) is considered to be insufficient [77], the ANN can be designed to improve the prediction accuracy.

## 10.1 Heat Load Prediction

The heating plant uses (45) to predict the heat load by a sum of time-dependent and temperature dependent components. [4]

$$f_p(t, \mathcal{G}_{ex}) = f_{time}(t) + f_{temp}(\mathcal{G}_{ex}) \quad (45)$$

Where

- $f_{time}(t)$  is the time dependent component,
- $t_0$  is the time offset,
- $\mathcal{G}_{ex}$  is the outdoor temperature,
- $f_{temp}(\mathcal{G}_{ex})$  is the outdoor temperature

### 10.1.1 Temperature dependent component

The temperature dependent component is approximated by using the generalized logistics function.

$$f_{temp}(\mathcal{G}_{ex}) = A + \frac{K - A}{(1 + Qe^{-B(\mathcal{G}_{ex} - M)})^{\frac{1}{v}}} \quad (46)$$

Where

- $A$  is the lower asymptote,
- $K$  is the upper asymptote,
- $Q$  is the dependent on the value  $f_{temp}(0)$
- $B$  is the growth rate,
- $v$  indicates near which asymptote the maximum growth occurs,
- $M$  is the time of maximum growth if  $Q = v$ .

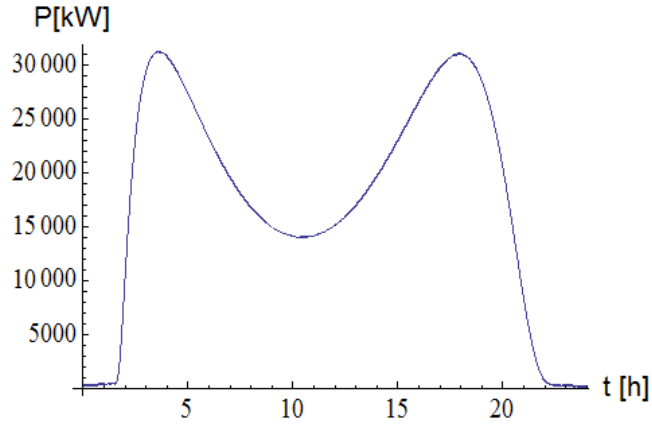


Fig. 40: Predictive function  $f_{time}(t)$

### 10.1.2 Time dependent component

The time dependent component is approximated by a sum of two peak functions. The Gaussian Hybrid and the truncated exponential function (EGH) [6] were selected as the most convenient functions. The Gaussian Hybrid and truncated exponential function are defined as follows:

$$d = 2\sigma^2 + \tau(t - t_m)$$

$$f_{EGH}(t) = H \exp\left(\frac{-(t - t_m)^2}{d}\right) \quad \text{if } d > 0 \quad \text{else } 0 \quad (47)$$

Where

- $H$  is the peak height,
- $\sigma$  is the standard deviation of the parent Gaussian peak,
- $\tau$  is the time constant of the precursor exponential decay,
- $k_L$  is the parameter of the speed of the fall of the leading trail,
- $t_m$  is the time of the peak.

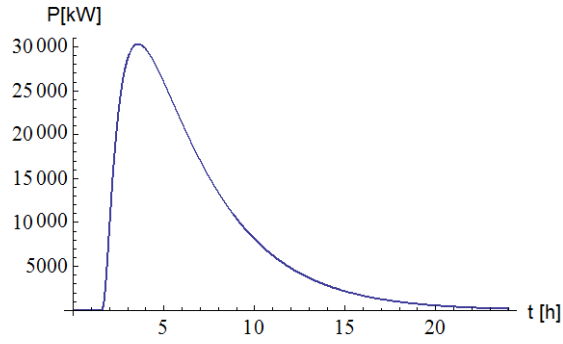


Fig. 41: Predictive function  $f_{EGH1}(t)$  for  $\tau=3.6$

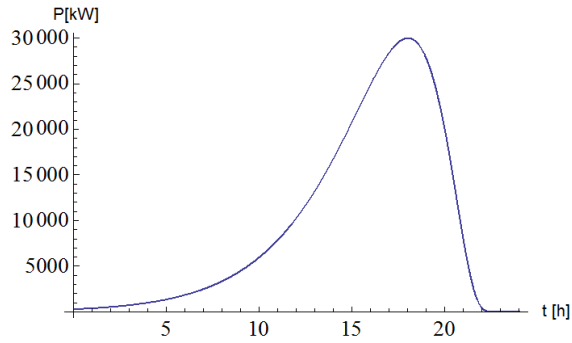


Fig. 42: Predictive function  $f_{EGH2}(t)$  for  $\tau=-3.0$

Then  $f_{time}(t)$  is a sum of the two EGH functions:

$$f_{time}(t) = f_{EGH1}(t) + f_{EGH2}(t) \quad (48)$$

$$f_p(t, \mathcal{G}_{ex}) = f_{time}(t) + A + \frac{K - A}{(1 + Qe^{-B(\mathcal{G}_{ex} - M)})^{\frac{1}{v}}} \quad (49)$$



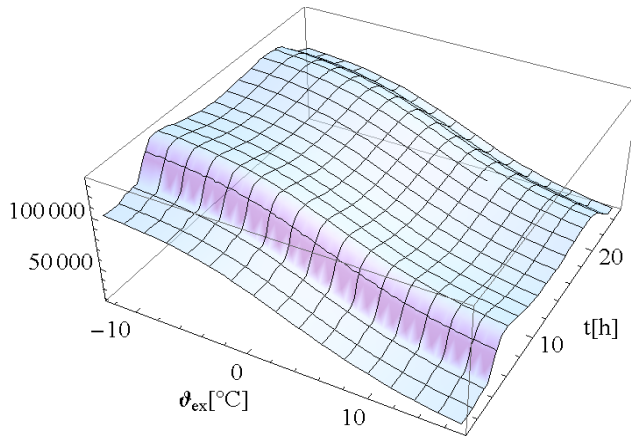


Fig. 43: Predictive function  $f_P(t, \vartheta_{ex})[kW]$

## 10.2 ANN Synthesis for HLP

The task of the ANN synthesis here is to create an ANN that provides the HLP with data containing measured heat load, time and external temperature. Data covering the period from Nov 3, 2009 to Dec 31, 2009 includes 1,416 samples taken in one-hour steps. The formal HLP function (see chapter 10.1) resulted in 4.28% NRMSD (22) within the provided data. Therefore, the ANN with the lower NRMSD is desirable.

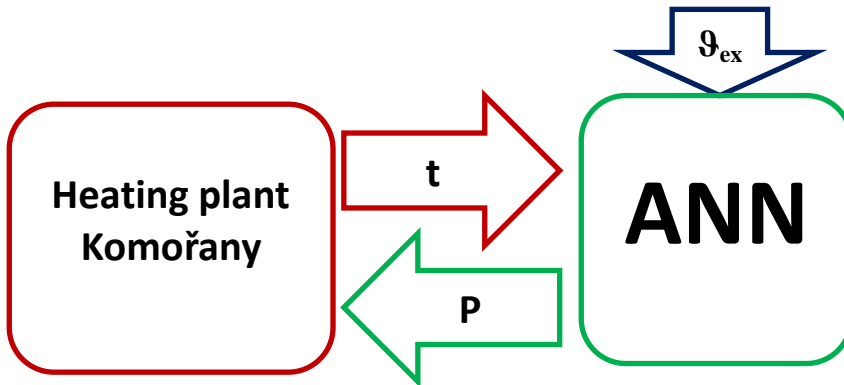


Fig. 44: Relationship between heating plant (red), ANN (green) and atmospheric condition (blue)

The used data was normalized into a  $\langle 0; 1 \rangle$  interval and divided into training, validation and test sets. The whole experiment was conducted in accordance with rules proposed in [79].

A simple but effective GFS structure was used for the ANN synthesis during this experiment:

$$GFS_{all} = \{+, AN, K * \mathcal{G}_{ex}, K * t\} \quad (50)$$

The computation of the CF was extended from (21) to (51) and then normalized again into (22).

$$\min_{ANN} \sum_t (P(t) - f_p((t - t_0) \bmod 24, \mathcal{G}_{ex}, ANN)) \quad (51)$$

Where

$ANN$  is a vector of the ANN structure,  
weight and biases,

$P$  is a measured value of head load

In case the best-synthesized ANN does not improve its CF by at least 0,001%, then the breeding is stopped.

The setting of Asynchronous SOMA used as the EA for AP can be seen in Table 11 while the Table 12 shows the setting of SOMA used for ANN learning.

### 10.3 Results

In 100 cases AP was always able to synthesize the ANN with the NRMSD = 3.46%; however, final results vary in a number of AN used. On average, the ANN with 31 AN was produced, however, the first 15 AN produced were enough to determine the HLP function more precisely (see Fig. 52).

One example of the typically obtained ANN structure is shown on the following pages with different *evolution loops* depicted in colors (for the *evolution loop* meaning see the chapter 5.3.2). Evolution loops are described functionally (52), (54), (56), (58) - (73); mathematically (53), (55), (57), (59), (91) - (94) and graphically from Fig. 45 to Fig. 51.

$$ANN0 = AN[t] + \mathcal{G}_{ex} \quad (52)$$

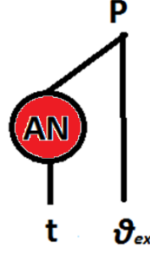


Fig. 45: First evolution loop of synthesized ANN (52)(in red)

$$ANN0 = w_2 \frac{e^{2\lambda_1(w_1t + \phi_1)} - 1}{e^{2\lambda_1(w_1t + \phi_1)} + 1} + w_3 \mathcal{G}_{ex} \quad (53)$$

$$ANN1 = ANN0 + AN[t] \quad (54)$$

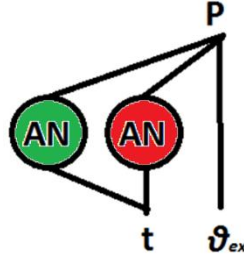


Fig. 46: Second evolution loop of synthesized ANN (54)(in green)

$$ANN1 = w_2 \frac{e^{2\lambda_1(w_1t + \phi_1)} - 1}{e^{2\lambda_1(w_1t + \phi_1)} + 1} + w_4 \frac{e^{2\lambda_2(w_5t + \phi_2)} - 1}{e^{2\lambda_2(w_5t + \phi_2)} + 1} + w_3 \mathcal{G}_{ex} \quad (55)$$

$$ANN2 = ANN1 + AN[AN[t] + t + AN[AN[t]]] \quad (56)$$

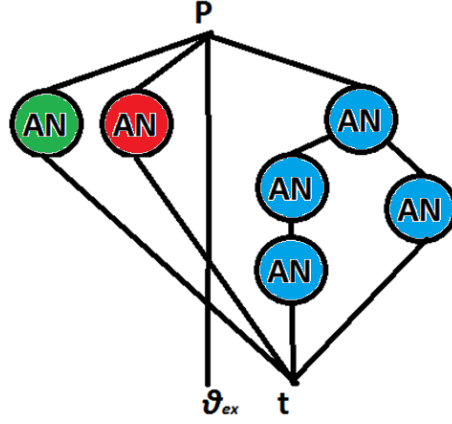


Fig. 47: Third evolution loop of synthesized ANN (56)(in blue)

$$\begin{aligned}
 & ANN2 \\
 &= w_2 \frac{e^{2\lambda_1 (w_1 t + \phi_1)} - 1}{e^{2\lambda_1 (w_1 t + \phi_1)} + 1} + w_4 \frac{e^{2\lambda_2 (w_5 t + \phi_2)} - 1}{e^{2\lambda_2 (w_5 t + \phi_2)} + 1} - \\
 &\quad \frac{2\lambda_6 (w_8 \frac{e^{2\lambda_4 (w_9 t + \phi_4)} - 1}{e^{2\lambda_4 (w_9 t + \phi_4)} + 1} + w_{10} \frac{e^{2\lambda_5 (w_6 \frac{e^{2\lambda_3 (w_7 t + \phi_3)} - 1}{e^{2\lambda_3 (w_7 t + \phi_3)} + 1} + \phi_5)} - 1 + \phi_6)}{e^{2\lambda_5 (w_6 \frac{e^{2\lambda_3 (w_7 t + \phi_3)} - 1}{e^{2\lambda_3 (w_7 t + \phi_3)} + 1} + \phi_5)} + 1} - 1}{e} \\
 &+ w_{11} \frac{2\lambda_6 (w_8 \frac{e^{2\lambda_4 (w_9 t + \phi_4)} - 1}{e^{2\lambda_4 (w_9 t + \phi_4)} + 1} + w_{10} \frac{e^{2\lambda_5 (w_6 \frac{e^{2\lambda_3 (w_7 t + \phi_3)} - 1}{e^{2\lambda_3 (w_7 t + \phi_3)} + 1} + \phi_5)} - 1 + \phi_6)}{e^{2\lambda_5 (w_6 \frac{e^{2\lambda_3 (w_7 t + \phi_3)} - 1}{e^{2\lambda_3 (w_7 t + \phi_3)} + 1} + \phi_5)} + 1} + 1}{e} \\
 &\quad + w_3 g_{ex}
 \end{aligned} \quad (57)$$

Mathematical descriptions (91) - (94) of the following four evolution loops (58) - (61) are included in Appendix III.

$$ANN3 = AN[AN[t]] + ANN2 \quad (58)$$

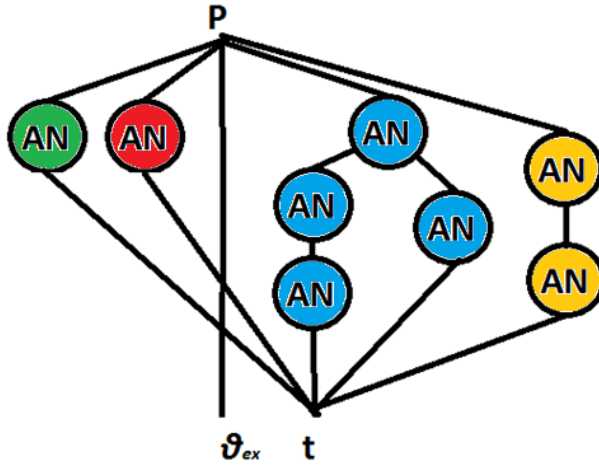


Fig. 48: Fourth evolution loop of synthesized ANN (58)(in yellow)

$$ANN4 = ANN3 + AN[AN[g_{ex}] + AN[g_{ex}] + t] \quad (59)$$

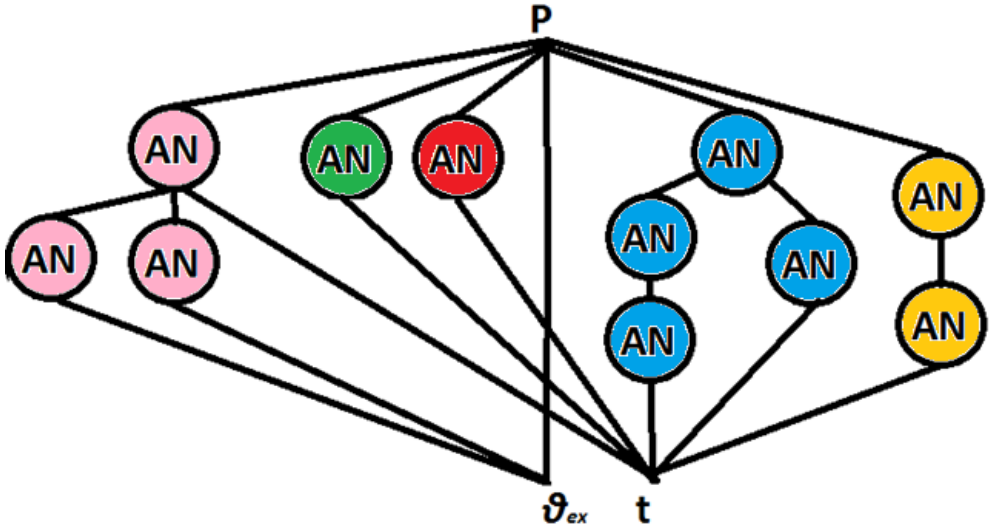


Fig. 49: Fifth evolution loop of synthesized ANN (59)(in purple)

$$ANN5 = AN[AN[AN[t + \vartheta_{ex}]] + ANN4] \quad (60)$$

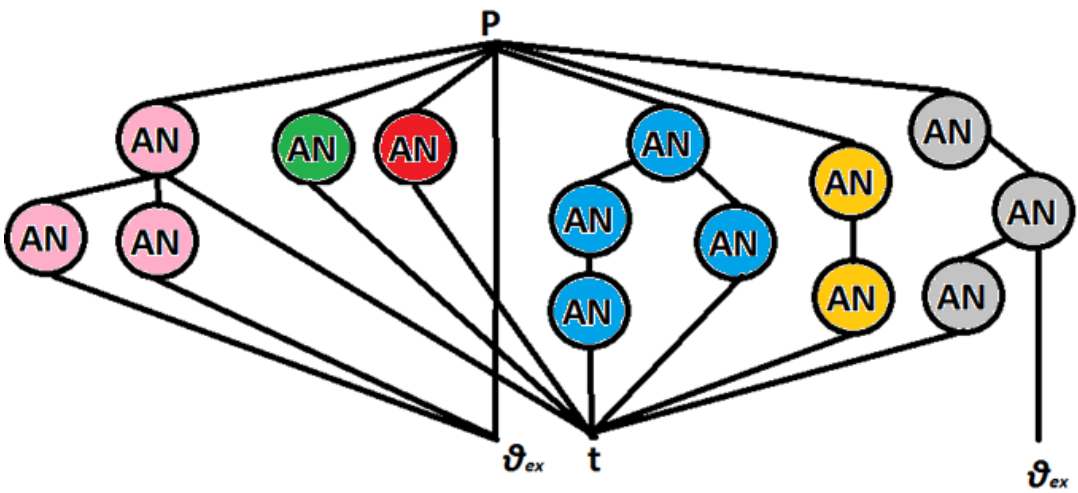


Fig. 50: Sixth evolution loop of synthesized ANN (60)(in gray)

$$ANN6 = ANN5 + AN[t + AN[t + \vartheta_{ex}]] \quad (61)$$

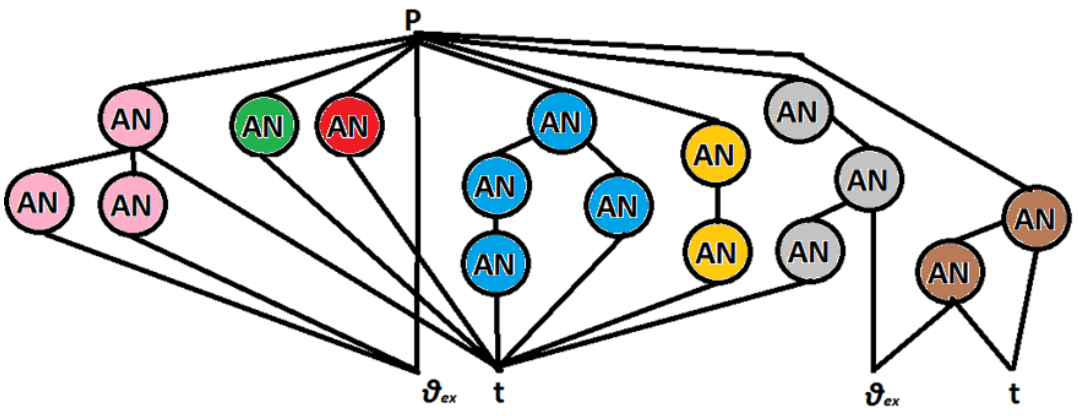


Fig. 51: Seventh evolution loop of synthesized ANN (61)(in brown)

Twelve evolution loops, which added a less significant AN follow:

$$ANN7 = AN[t + AN[AN[g_{ex}]]] + ANN6 \quad (62)$$

$$ANN8 = AN[AN[AN[g_{ex}] + h] + ANN7] \quad (63)$$

$$ANN9 = AN[AN[t]] + ANN8 \quad (64)$$

$$ANN10 = ANN9 + AN[t] \quad (65)$$

$$ANN11 = AN[g_{ex}] + ANN10 + AN[ANN10] \quad (66)$$

$$ANN12 = AN[t] + ANN11 \quad (67)$$

$$ANN13 = AN[t] + ANN12 \quad (68)$$

$$ANN14 = AN[t] + ANN13 \quad (69)$$

$$ANN15 = AN[t] + ANN14 \quad (70)$$

$$ANN16 = AN[t + g_{ex}] + ANN15 \quad (71)$$

$$ANN17 = ANN16 + AN[AN[g_{ex} + AN[ANN16 + t + ANN16] + t + AN[t] + ANN16 + AN[t] + ANN16]] \quad (72)$$

$$\underline{ANN18} = ANN17 + AN[AN[t + g_{ex}]] \quad (73)$$

In these cases, AP used 18 sub ANN to form the final ANN. The synthesized ANN have non-trivial structures, nevertheless, they can be easily simplified, if necessary, by cutting the later sub ANN with positive influence on the ANN computation speed. For example, (41) benefit for the ANN accuracy is only 0.001%. The exponential downgrade of the migration loop significance can be seen in Fig. 52.

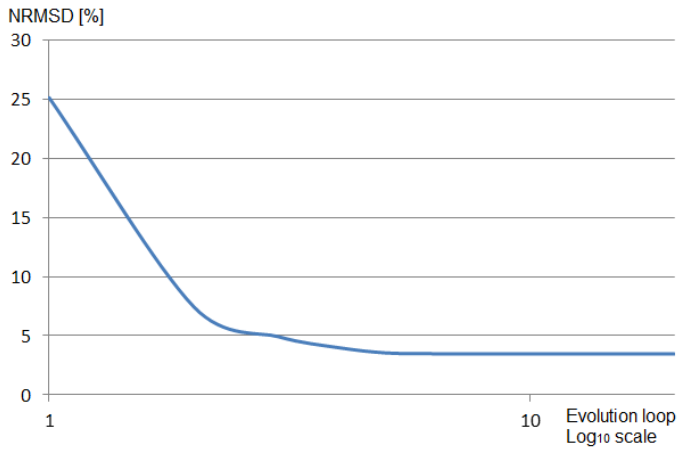


Fig. 52: Exponential downgrade of migration loop significance for precision

## 10.4 Prediction by Standard Feedforward ANN

In the previous studies [68] Matlab Neural Network Toolbox [80] (Fig. 53) was used to create the standard feedforward ANN for the HLP and train it with the help of the BP (Fig. 54).

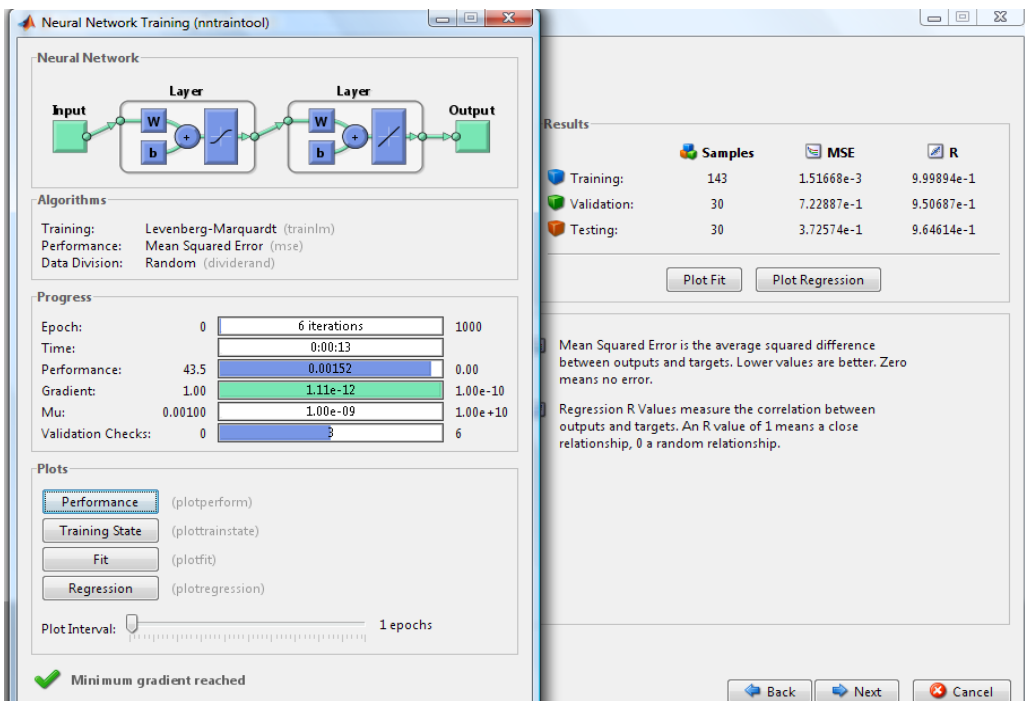


Fig. 53: Matlab Neural Network Toolbox



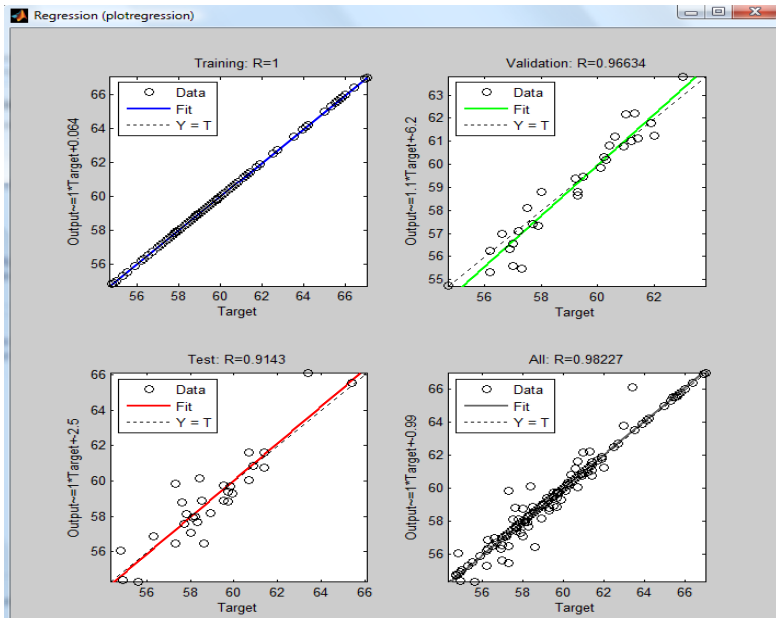


Fig. 54: BP regression

Using a comparable number of the ANN Matlab produced a 7% worse one-hidden layer ANN and a 9% worse two-hidden layer ANN (for the ANN with hidden layers see Fig. 17) than the experiment in chapter 10.2. Interestingly even ANN learning was slower as Matlab was not able to parallelize this process between more server cores.

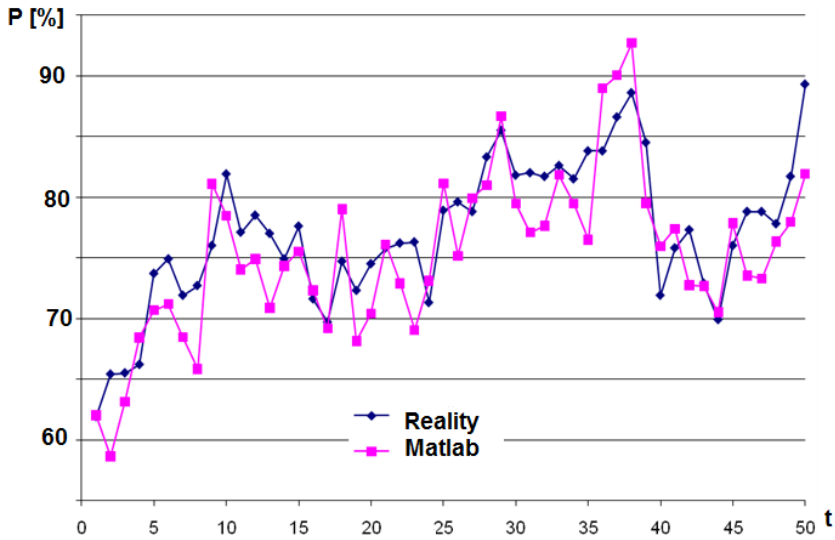


Fig. 55: HLP by standard ANN

## 10.5 Arithmetical approach to GFS structure formulation

Inspired by [78] the experiment from chapter 10.2 was repeated with a different GFS setting:

$$GFS = \{+, *, hTan, t, \mathcal{G}_{ex}, K\} \quad (74)$$

Such approach is partially similar to the F set, whose content was proposed in [25] (see chapter 5.2). The implementation of (73) produced interesting partially neural structures, as can be seen for example in Fig. 56.

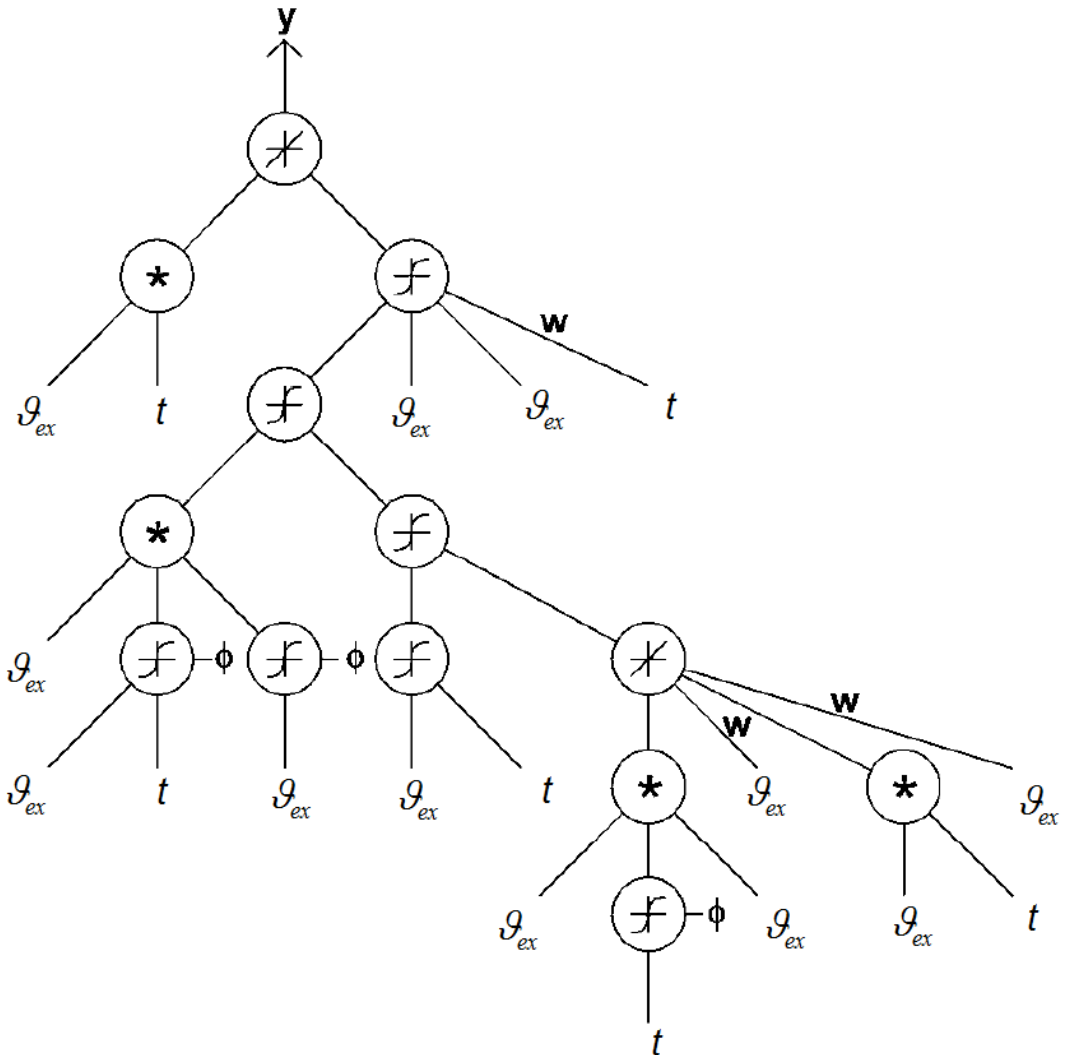
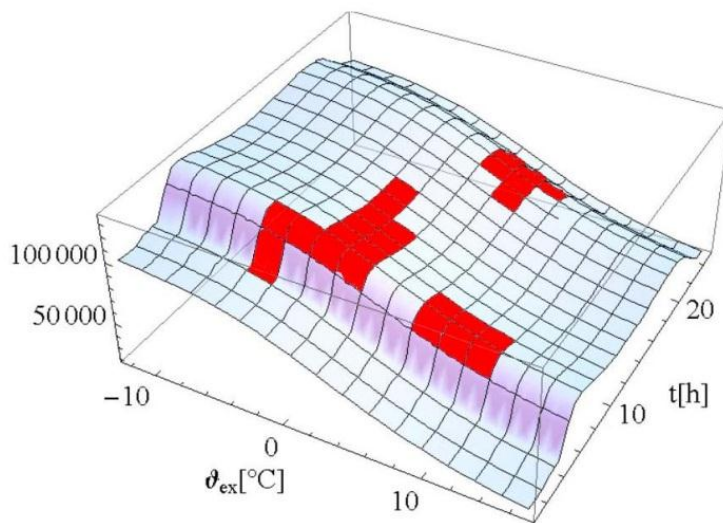


Fig. 56: ANN resulting from GFS containing arithmetical functions

Nevertheless, the ANN synthesized implementing (73) is of significantly worse quality than ANN obtained from the experiment in chapter 10.2. This determination is also in agreement with [41].

## 10.6 Conclusion

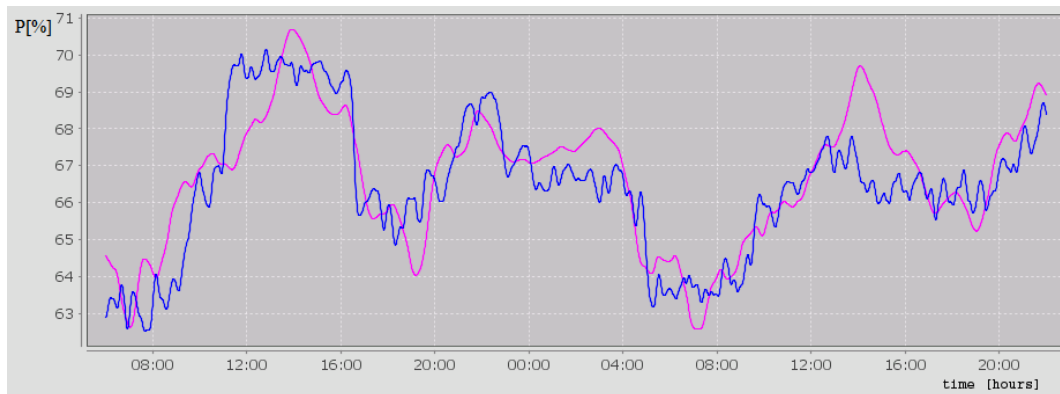
AP was able to synthesize the ANN with the NRMSD 3.46%. This success represents a 19% improvement in comparison with the commonly used HLP function (49). The synthesized ANN provides a 7% better result than the HLP function modeling with a help of the standard ANN organized into layers and taught by the BP and 5% better result than the ANN optimized via the GA [81].



*Fig. 57: Surface of HLA function provided by synthesized ANN – areas significantly corrected in comparison with formal function are depicted in red*

The application of this method to the real case of the heating plant was possible only due to a successful distribution method described in chapter 6.2. The algorithm ran on 24 of the Super Micro Server (see Appendix IV). Each core was occupied by two individuals of the algorithm. In this configuration, one algorithm's run took approximately 14 minutes, which resulted in the whole experiment lasting less than 24 hours.

The synthesized ANN was used as a part of the National Research Program II No. 2C06007; it successfully defended project the solution and can positively influence a quality control in the Komořany heating plant (see Fig. 2)



*Fig. 58: Predicted and actual curve of agglomeration heat load*

AP proved its ability to successfully synthesize the ANN in dynamic and irregular environments of real life problem situations. The implementation of the presented method was accepted for publication in [82].

# 11 ANN SYNTHESIS FOR CLASS CLASSIFICATION

To statistically evaluate the ANN synthesis' ability to successfully generate the ANN performing classification, the ANN synthesis was compared with the GPNN solving the XOR problem in chapter 11.1. Chapter 11.2 describes an example of the ANN synthesis usage for a real life cancer classification problem and its comparison with other methods.

## 11.1 XOR Classification Problem

Inspired by [25] the GPNN was used to solve an XOR classification problem which is the simplest nonlinearly separable classification problem (see Fig. 59) with a known minimal network depicted in Fig. 60.

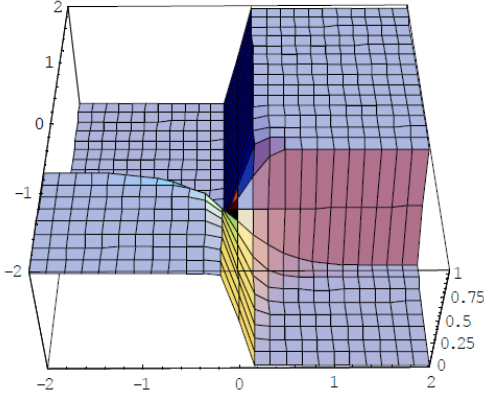


Fig. 59: XOR classification problem

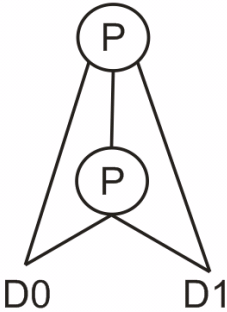


Fig. 60: Minimal XOR solving ANN

The GPNN was performed with the help of the GPC++ [25] software and minimal the ANN described in Fig. 61 was found.

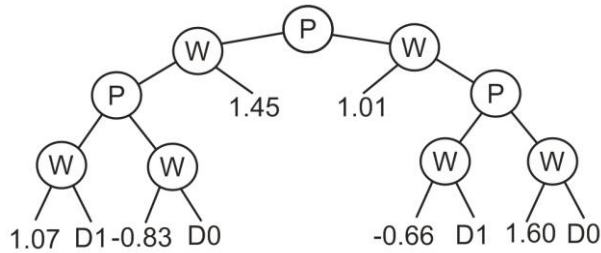


Fig. 61: Minimal ANN generated by GPNN that performs the XOR problem

According to [41] the GPNN cannot generate the ANN from Fig. 60 simply because the P function is only allowed to have two arguments, while for this particular ANN the output AN has three inputs. To overcome this problem the GPNN has to consider implementation of another function  $P_2(x_1, x_2, x_3)$ .

On the contrary, the ANN synthesis method (chapter 5.3) can synthesize the AN with almost unlimited number of inputs and with a usage of the simple GFS it was able to synthesize the minimal ANN (Fig. 60) nine times out of ten attempts.

## 11.2 Cancer Classification Problem

Breast cancer diagnosis is a classification problem introduced in [79]. The ANN tries to classify a tumor as either benign or malignant based on cell descriptions gathered by a microscopic examination.

Input attributes are, for instance, the clump thickness, the uniformity of cell size and cell shape, the amount of marginal adhesion, and the frequency of bare nuclei.

The dataset includes 699 examples with 9 inputs and 2 outputs. All inputs are continuous; 65.5% of the examples are benign. This makes for entropy of 0.93 bits per example.

This dataset was created based on the "breast cancer Wisconsin" problem from the UCI repository of machine learning databases originally provided by Dr. William H. Wolberg from the University of Wisconsin Hospitals, Madison, USA [83].

For the purpose of the executed experiment *cancer1* set was chosen. Based on [84], four ANN optimization methods provide a dissimilar mean test classification error dealing with *cancer1*:

Table 18: ANN mean testing classification error

de Falco et al. [85]	2.46%
Prechelt [79]	1.38%
Brameier and Banzhaf [86]	2.18%
The CMAC NN classifier [84]	3.94%

### 11.2.1 Experiment Set Up

To synthesize the optimal ANN, AP used the GFS with equal rates of neurons, connections and inputs:

$$\begin{aligned} \text{GFS} = \{ &+, \text{AN}, K^{*x0}, +, \text{AN}, K^{*x1}, +, \text{AN}, K^{*x2}, +, \\ &\text{AN}, K^{*x3}, +, \text{AN}, K^{*x4}, +, \text{AN}, K^{*x5}, +, \text{AN}, K^{*x6}, +, \text{AN}, \\ &K^{*x7}, +, \text{AN}, K^{*x8} \} \end{aligned} \quad (75)$$

while the CF was formulated in accordance with (23) and (24).

Such approach ensured finding the best possible ANN as well as ANN with the minimal structure.

The setting of Asynchronous SOMA is depicted in Table 11 and Table 12.

### 11.2.2 Results

During 100 runs of the algorithm ANN structurally described as (76), (77) was found to be the best solution for the given classification problem with a test classification error of 1.14%. Two wrongly classified examples within the test set were on positions 81 and 87.

$$\text{ANN0} = \text{AN}[x5] + x0 + x2 + x3 + \text{AN}[x7] \quad (76)$$

$$\underline{ANN1} = AN[ANN0 + x3] + x8 \quad (77)$$

Functions (78) and (79) described the learned ANN, which can be easily tested on *cancer1* publicly provided by [79]:

$$\begin{aligned} ANN0 = & -2,97309632219583 * AN[-1,46365223054944 * (- \\ & 5,03444335192183 * x5 + 1,76603626076413)] + -7,40609983802126 * x0 + \\ & -5,46830267210878 * x2 + -6,94991567402608 * x3 + -5,99052909574962 * \\ & AN[1,59467356605207 * (3,68066486608268 * x7 + -3,61373674292757)] \end{aligned} \quad (78)$$

$$\begin{aligned} \underline{ANN1} = & -2,83643286341635 * AN[-0,179040669733212 * (ANN0 + \\ & 0,796079062345568 * x3 + 0,670777686792787)] + -2,95757076519615 * \\ & x8 \end{aligned} \quad (79)$$

The structural evolution of the resulting ANN can be seen in Fig. 62 and Fig. 63 .

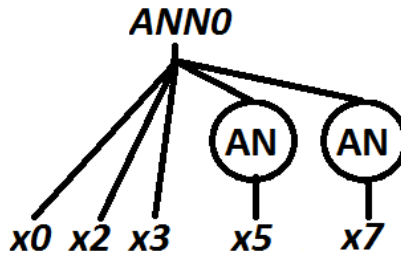


Fig. 62: First evolution loop of the resulting ANN

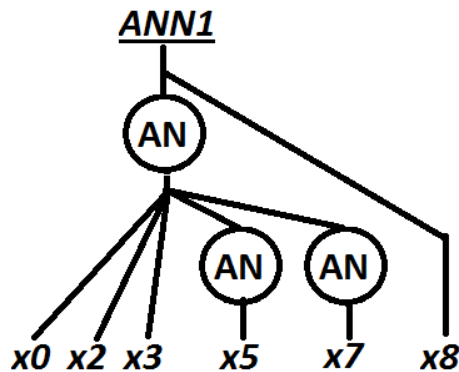


Fig. 63: Resulting ANN1 structural evolution



### 11.2.3 Conclusion

AP proves its ability to synthesize and, at the same time, optimize the ANN, which effectively classifies the given task while its structure is minimized.

The best obtained ANN had even 0.28% better test classification error than the mean test classification error of the best competing method [79]; however, the ANN (77) contains only three AN and is totally omitting inputs  $x1$  and  $x4$  causing the ANN's inability to extend a performance with respect to *cancer2* and *cancer3* sets. Nevertheless, the experiment's performance ratifies AP as an efficient tool for the ANN synthesis [87].

### 11.2.4 Terminals Density Comparison for Different GFS

Finally, the experiment from chapter 11.2.1 was repeated with the application of (80), (81) and (82). To explore the influence of terminals (for *terminal* meaning see chapter 4.3). The probability that a position within a vector of an individual is occupied by the terminal is depicted in

$$\text{GFS}_a = \{+, AN, K^*x0, K^*x1, +, AN, K^*x2, K^*x3, +, AN, K^*x4, +, AN, K^*x5, K^*x6, +, AN, K^*x7, K^*x8\} \quad (80)$$

$$\text{GFS}_b = \{+, AN, K^*x0, K^*x1, K^*x2, K^*x3, K^*x4, +, AN, K^*x5, K^*x6, K^*x7, K^*x8\} \quad (81)$$

$$\text{GFS}_c = \{+, AN, K^*x0, K^*x1, K^*x2, K^*x3, K^*x4, K^*x5, K^*x6, K^*x7, K^*x8\} \quad (82)$$

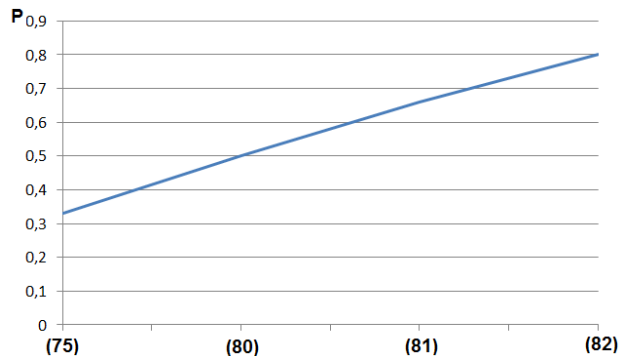


Fig. 64: Probability of terminal occurrence for different GFS

In comparison with (75) the ANN resulting from (81), (82) and (83) show a lower level of generalization as a number of the employed input was generally smaller.

## 12 ANN SYNTHESIS SOFTWARE

Software for the ANN synthesis support was developed under .NET Framework 3.5 [88] and source codes were written in C#. The software was used and debugged performing experiments in chapters 6 to 11.

Input data is supposed to be formatted as a csv file or an Excel sheet. After opening the file, a user is invited to choose between approximation, prediction or classification of the problem. The data is then automatically validated in a sense of consistency, it is normalized and divided into learning, validation and test sets in accordance with [79].

The experiment is then computed within implicit control parameters proposed and proved in the practical part of the thesis. The user can also adjust both control parameters and the GFS content as can be seen, for example, in Fig. 65.

The screenshot shows a software interface for setting SOMA control parameters. It features several input fields and a large 'Start' button. The parameters are as follows:

Parameter	Value
Cost Function	[Dropdown]
NP	60
Borders Min	-512
Borders Max	512
Step	0.11
Mass	3
Dimension	100
PRT	0.1
Evaluations	643696
Divergence	0
Control Period	1
Repetitions	1000
Save to File	[File Selection]

Fig. 65: SOMA controll parameters setting

Technically, all functions contained in the GFS set have to be inherited from abstract class *APFunction* (83).

```

public abstract class APFunction
{
    public abstract double evaluate(ref Token token);

    public abstract String toString(ref Token token);           (83)

    public abstract int countConst(ref Token token);

    public int operNumber;
}

```

The particular implementation of (16) by inheritance from (83) and redefinition of the abstract method *evaluate* as is described in (84).

```

public override double evaluate(ref Token token)
{
    token.left -= operNumber;

    double sum = AP.next(ref token);

    sum += token.constants[token.conPointer++];                (84)

    sum *= token.constants[token.conPointer++];

    return token.constants[token.conPointer++] * (Math.Pow
(Math.E, 2 * sum) - 1) / (Math.Pow(Math.E, 2 * sum) + 1);
}

```

In (85) two instances of class *Neuron* containing (84) an *evaluate* method definition are added into the *GFS* set (together with *Plus* (19) and *WeighInput* (20) instances) is defined as a generic collection *List*.

```

List<APFunction> GFS = new List<APFunction>();
GFS.Add(new Plus());
GFS.Add(new Neuron());
GFS.Add(new WeighInput(0, "x1"));
GFS.Add(new Plus());
GFS.Add(new Neuron());
GFS.Add(new WeighInput(1, "x2"));                               (85)

```

The example of the synthesized ANN using the GFS defined in (85) can be seen in Fig. 66.

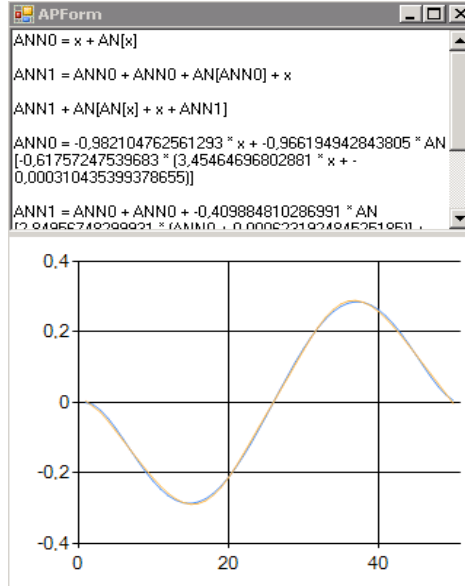


Fig. 66: ANN synthesized using (85)

(86) describes a reinforced evolution process of adding a subANN (chapter 5.3.2) defined as an instance of an *APPart* class into the GFS. All available input data located in a *dataList* is pre-counted and saved to the solved collection in order to boost the ANN synthesis performance. Then, a new evolution loop is started by a calling static method *DISOMA.start*.

```

apPart = new APPart(new Token(results.finalLeader.position,
oldGFS, null, results.finalLeader.constants), "ANN" +
pocitadlo.ToString());

for (int i = 0; i < dataList.Count; i++)
{
apPart.solved.Add(dataList[i].inputs, AP.evaluate(new
Token(results.finalLeader.position,
GFS,dataList[i].inputs,results.finalLeader.constants)).value);
}

GFS.Add(apPart);

results = DISOMA.start(new AP(GFS, dataList.ToArray(),
dataListValid.ToArray()), specimen, 50, long.MaxValue, 0.01,
3, 0.11, 3);

```

(86)

The overrated method *costFunction* (87) defines the CF described theoretically as (24) in chapter 5.3.3. The head of the *costFunction* is prescribed by an interface *CostFunction* and needs to be implemented to allow AP to be operated by the EA.

To prevent exceptions caused by *Double* type overflowing, an *apReturn.value* is tested on a *Duble.IsNaN* (not a number) condition.

```

public override Individual costFunction(double[]
position)
{
    double sum = 0;
    APReturn apReturn = evaluate(new Token(pointers, GFS,
data[0].inputs, position));

    if (data[0].output == 0)
    {
        if (apReturn.value > 0) sum += apReturn.value + 1;
    }
    else
    {
        if (apReturn.value <= 0) sum +=
Math.Abs(apReturn.value) + 1;
    }
    if (Double.IsNaN(apReturn.value)) sum += 10000;

    for (int i = 1; i < data.Length; i++)
    {
        apReturn = evaluate(new Token(pointers, GFS,
data[i].inputs, position));
        if (data[i].output == 0)

        {
            if (apReturn.value > 0) sum += apReturn.value + 1;
        }

        else
        {
            if (apReturn.value <= 0) sum +=
Math.Abs(apReturn.value) + 1;
        }

        if (Double.IsNaN(apReturn.value)) sum += 10000;
    }
    return new Individual(position, sum, apReturn.deep);
}

```

(87)

The complete code of the direct asynchronous SOMA implementation (see chapter 3.4 and 6.2 ) used within AP for the ANN synthesis can be accessed in Appendix V. Fig. 67 is a screenshot of an asynchronous SOMA result form based on the *Windows Form* technology [88].

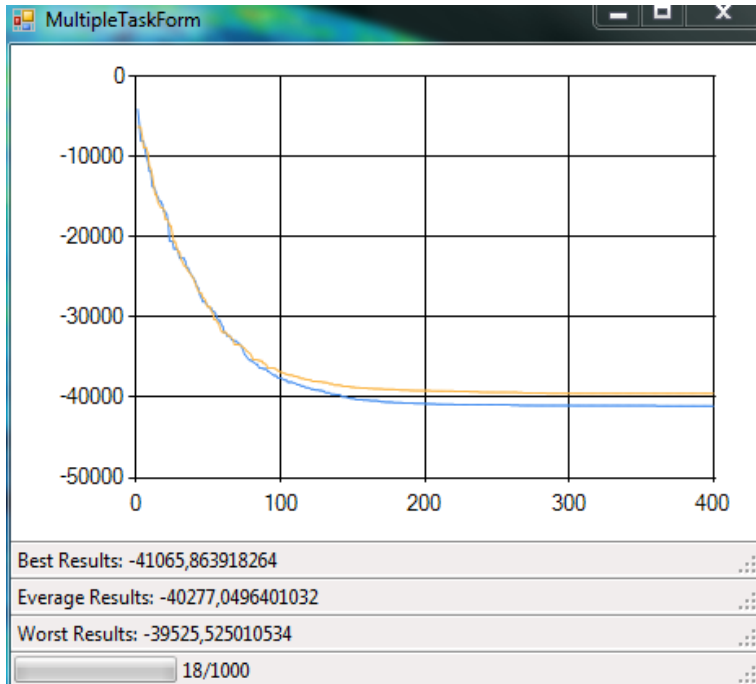


Fig. 67: Asynchronous SOMA results form

To protect SOMA leader position consistency (as can be seen in Fig. 30), *ReaderWriterLock* class is used to lock the leader position while a thread is reading this value:

```

leaderLock.AcquireReaderLock(Timeout.Infinite);

if (leader.Equals(population[i]))
{
    leaderLock.ReleaseReaderLock();
    continue;
}
    
```

(88)

A similar implementation is used in (89) while the thread is attempting to update the leader position.

```

leaderLock.AcquireReaderLock(Timeout.Infinite);

if (population[i].costValue < leader.costValue)
{
leaderLock.ReleaseReaderLock();

leaderLock.AcquireWriterLock(Timeout.Infinite);

if (population[i].costValue < leader.costValue)
leader = population[i];

leaderLock.ReleaseWriterLock();

}
else
{
leaderLock.ReleaseReaderLock();
}

```

(89)

The process of equal distribution of individuals between available processors is described in (90). This division is by default determined by a *numberOfProcessors* obtained from a

*System.Environment.GetEnvironmentVariable("NUMBER\_OF\_PROCESSORS").*

```

int sequel = 0;

while (true)
{
individuals[sequel]++;
sequel++;
if (sequel == numberOfProcessors) sequel = 0;
if (individuals.Sum() == NP) break;
}

for (int i = 0; i < numberOfProcessors; i++)
{
threads[i] = new Thread(new
ParameterizedThreadStart(DISOMAWork));
threads[i].Start(new Parameters(individuals[i],
random.Next()));
}

for (int i = 0; i < numberOfProcessors; i++)
{
threads[i].Join();
}

```

(90)

The ANN synthesis software can be run on any arbitrary platform. It only requires installation of .NET Framework 3.5. The ANN synthesis was tested on the Super Micro server (see Appendix IV).

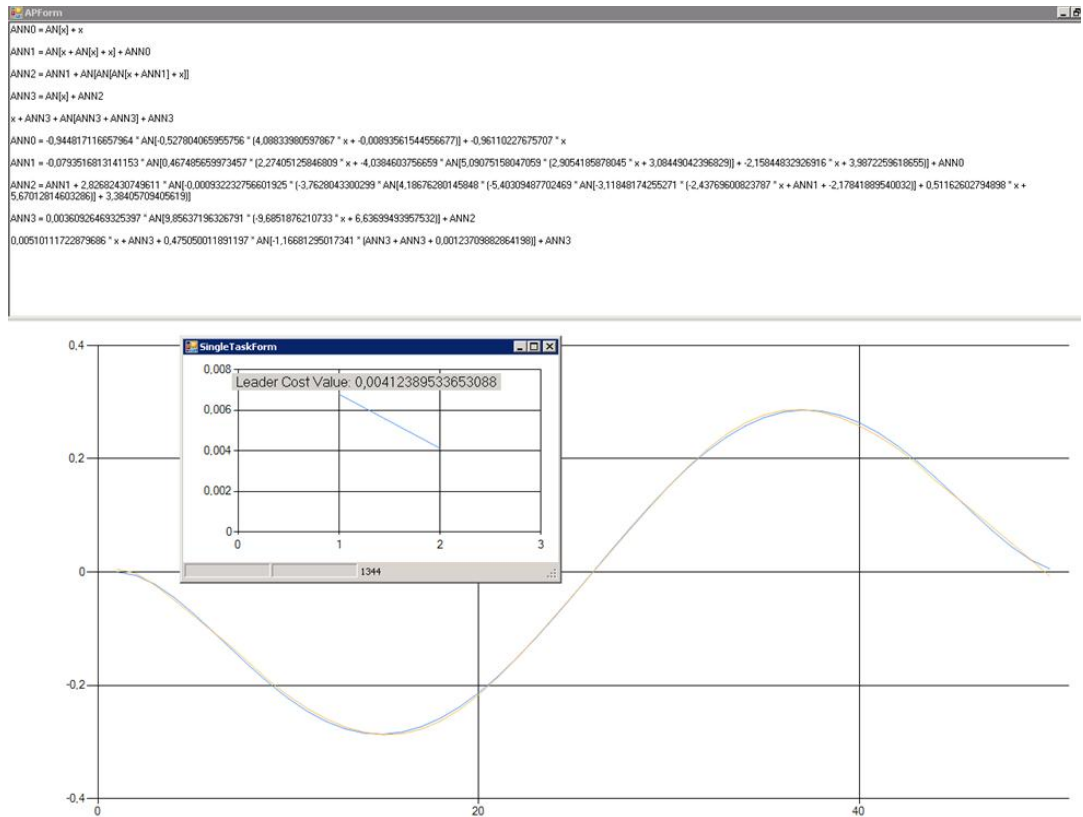


Fig. 68: ANN synthesis software forms

Individual instances of the Thread class are automatically operated by the framework to distribute computation load equally.

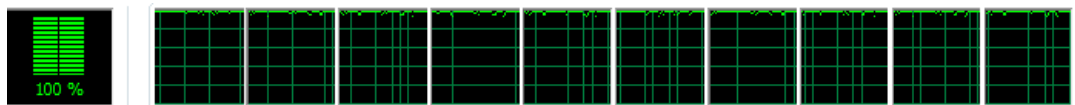


Fig. 69: Optimal computation division on eight processors

The obtained results are then saved via an interoperable XML format as can be seen in Appendix VI.



## 13 FINAL CONCLUSION

The Neural Network Synthesis was developed on the basis of AP (chapter 4.3) and SOMA (chapter 3.4) algorithms and theoretically described in chapter 5.3.

The method was successfully tested on the real life problems [67], [76] as well as on widely recognized benchmark functions [19], [24], [41] with respect to the function approximation (chapter 8), prediction (chapter 10) and (chapter 11) problems.

The ANN synthesis software was designed based on .NET Framework technology (chapter 12). The resulting software is capable of automatic synthesis and optimizing the ANN based on the user-given data within a reasonable time. Such performance has to be supported by efficiently distributed computation proposed in chapter 6.2 that was statistically proven in chapter 7.

The ANN synthesis proves to be a useful and efficient tool for nonlinear modeling in comparison with competing methods [4], [25], [79], [84], [85] and [86] while the optimal strategy of its control parameter settings (chapter 9.1.2 and 9.1.3) and the GFS composition were developed (chapter 10.5 and 11.2.4).

The ANN optimized by the ANN synthesis was practically deployed within “The intelligent system controlling an energetic framework of an urban agglomeration”, the final technical report of the National Research Program II. These results together with the theoretical background of the method were also accepted for publication by Springer [82].

Furthermore, the ANN synthesis proves its ability to synthesize smaller ANN than the GPNN as can be seen in chapter 11.1. Simultaneously, an almost infinitely complex ANN can be synthesized when using evolution loops (chapter 5.3.2). This process can also produce an ANN with feedforward branching (for example in (72)), which is a quality unavailable for the GPNN.

For particular conclusions of experimental results see chapters 7.1, 8.2, 9.1.4, 10.6 and 11.2.3.

## 14 REFERENCES

- [1] OPLATKOVÁ Z., ZELINKA I. Creating evolutionary algorithms by means of analytic programming - design of new cost function. In ECMS 2007, European Council for Modelling and Simulation, 2007, p. 271-276. ISBN/ISSN: 978-0-9553018-2-7
- [2] OPLATKOVÁ Z., ZELINKA I., Investigation on Artificial Ant using Analytic Programming. In Genetic and Evolutionary Computation Conference, 2006, p. 949-950. ISBN/ISSN: 1-59593-186-4.
- [3] VAŘACHA P., ZELINKA I., Distributed Self-Organizing Migrating Algorithm Application and Evolutionary Scanning. In Proceedings of the 22nd European Conference on Modelling and Simulation ECMS 2008 Nicosia, 2008. p. 201-206. ISBN/ISSN: 0-9553018-5-8.
- [4] KRÁL E., ET AL., Usage of PSO Algorithm for Parameters Identification of District Heating Network Simulation Model. In 14th WSEAS International Conference on Systems. Latest Trands on Systems. Volume II, Rhodes, WSEAS Press (GR) , 2010. p. 657-659. ISBN/ISSN: 978-960-474-214-1.
- [5] ČERVENKA M., ZELINKA I., Application of Evolutionary Algorithm on Aerodynamic Wing Optimisation. In Proceedings of the 2nd European Computing Conference, Venice, WSEAS Press (IT), 2008, ISBN/ISSN: 978-960-474-002-4
- [6] OPLATKOVÁ Z., ZELINKA I., Investigation on Shannon - Kotelnik Theorem Impact on SOMA Algorithm Performance. In European Simulation Multiconference, 2005, Riga, ESM , 2005. p. 66-71. ISBN/ISSN: 1-84233-112-4.
- [7] ŠENKERŤÍK R., ZELINKA I., Optimization and Evolutionary Control of Chemical Reactor. In 10th International Research/Expert Conference Trends in the Development of Machinery and Associated Technology, TMT, Zenica, Bosna and Hercegovina, 2006, p. 1171-1174. ISBN/ISSN: 9958-617-30-7.
- [8] SAXENA S., ET AL., Strategy for a generic resistance to, geminiviruses infecting tomato and papaya through in silico siRNA search, VIRUS GENES Volume: 43 Issue: 3, 2011, p. 409-434, Springer ISSN: 0920-8569

- [9] HOLLAND J. H., Genetic Algorithms, Scientific American, July 1992, p. 44 – 50
- [10] BEASLEY D., BULL D. R., MARTIN R. R., An Overview of Genetic Algorithms, Part 1, Fundamentals, University Computing, 1993, p. 58 – 69
- [11] OBITKO M., Genetic Algorithms [online]. 1998 [cit. 2011-03-02]. Operators of GA. WWW: <<http://www.obitko.com/tutorials/genetic-algorithms/operators.php>>.
- [12] SCHMITT, LOTHAR M., Theory of Genetic Algorithms, Theoretical Computer Science 259: 1–61, 2001
- [13] LAMPINEN J., ZELINKA I., New Ideas in Optimization – Mechanical Engineering Design Optimization by Differential Devolution, Volume 1. London: McGraw-hill, 1999, 20 p., ISBN 007-709506-5
- [14] ELSAYED S., SACKER R., ESSAM D., Multi-operator based evolutionary algorithms for solving constrained optimization problems, COMPUTERS & OPERATIONS RESEARCH Volume: 38 Issue: 12, 2011, p. 1877-1896 ISSN: 0305-0548
- [15] PRICE K., STORN R. M., LAMPINEN J. A., Differential Evolution : A Practical Approach to Global Optimization (Natural Computing Series), Springer; 1 edition, 2005, ISBN: 3540209506
- [16] KENNEDY J., EBERHART R., Particle Swarm Optimization. Proceedings of IEEE International Conference on Neural Networks. IV., 1995 p. 1942–1948.
- [17] SHI Y., EBERHART R.C., A modified particle swarm optimizer". Proceedings of IEEE International Conference on Evolutionary Computation. 1998, p. 69–73.
- [18] CHEN S., CHIEN C., Solving the traveling salesman problem based on the genetic simulated annealing ant colony system with particle swarm optimization techniques, EXPERT SYSTEMS WITH APPLICATIONS Volume: 38 Issue: 12, 2011, p. 14439-14450 ISSN: 0957-4174
- [19] ZELINKA I., Studies in Fuzziness and Soft Computing, New York : Springer-Verlag, 2004.
- [20] KRÁL E., ET AL. Usage of PSO Algorithm for Parameters Identification of District Heating Network Simulation Model. In 14th WSEAS International

- Conference on Systems. Latest Trends on Systems. Volume II Rhodes : WSEAS Press (GR) , 2010. p. 657-659. ISBN/ISSN: 978-960-474-214-1.
- [21] CHRAMCOV B., Heat Demand Forecasting for Concrete District Heating System. International Journal of Mathematical Models and Methods in Applied Sciences Volume 4, Issue 4, 2010, ISSN 1998-0140
- [22] VAŘACHA P., Innovative strategy of SOMA control parameter setting. In Recent Researches in Neural Networks, Fuzzy Systems, Evolutionary Computing and Automation: Proceedings of the 12th WSEAS international conference on Neural networks, fuzzy systems, evolutionary computing & automation (NNECFISIC'12). Brasov, WSEAS Press, 2011, p. 70-75. ISBN 978-960-474-292-9
- [23] OPLATKOVÁ Z., Metaevolution - Synthesis of Evolutionary Algorithms by means of Symbolic Regression. Zlín, Czech Republic, 2007. 97 p. Dissertation. University of Tomas Bata in Zlín.
- [24] KOZA J. R., Genetic Programming, MIT Press, 1998, ISBN 0-262-11189-6
- [25] KOZA J. R. ET AL., Genetic Programming III; Darwinian Invention and problem Solving, Morgan Kaufmann Publisher, 1999, ISBN 1-55860-543-6.
- [26] GALVAN-LOPEZ E., ET AL., Defining locality as a problem difficulty measure in genetic programming, GENETIC PROGRAMMING AND EVOLVABLE MACHINES Volume: 12 Issue: 4 , 2011, p. 365-401 ISSN: 1389-2576
- [27] O'NEILL M., RYAN C., Grammatical Evolution. Evolutionary Automatic Programming in an Arbitrary Language, Kluwer Academic Publishers, 2003, ISBN 1402074441
- [28] O'SULLIVAN J., RYAN C., An Investigation into the Use of Different Search Strategies with Grammatical Evolution, Proceedings of the 5th European Conference on Genetic Programming, p.268 - 277, 2002, Springer-Verlag London, UK, ISBN: 3-540-43378-3.
- [29] CHEN L ., Macro-grammatical evolution for nonlinear time series modeling-a case study of reservoir inflow forecasting, ENGINEERING WITH COMPUTERS Volume: 27 Issue: 4, 2011, p. 393-404, ISSN: 0177-0667

- [30] PELETEIRO M., EPMAS: Evolutionary Programming Multi-Agent Systems Proceedings of the 24th European Conference on Modeling and Simulation ECMS, 2010, p. 27-33, ISBN: 978-0-9564944-0-5
- [31] OPLATKOVÁ Z, ZELINKA I., Creating evolutionary algorithms by means of analytic programming - design of new cost function. In ECMS 2007, European Council for Modelling and Simulation, 2007, p. 271-276. ISBN/ISSN: 978-0-9553018-2-7
- [32] OPLATKOVÁ Z., ZELINKA I., Investigation on Artificial Ant using Analytic Programming. In Genetic and Evolutionary Computation Conference,. USA : The Association for Computing Machinery, 2006. ISBN/ISSN: 1-59593-186-4.
- [33] Wolfram Mathematica Documentation Center [online]. 2011 [cit. 2011-03-02]. Nonlinear Regression Package. WWW : <<http://reference.wolfram.com/mathematica/NonlinearRegression/guide/NonlinearRegressionPackage.html>>.
- [34] SIDHU G., ET AL., Determination of volume fraction of bainite in low carbon steels using artificial neural networks COMPUTATIONAL MATERIALS SCIENCE Volume: 50 Issue: 12, 2011, ISSN: 0927-025
- [35] CEVOLI C., ET AL., Classification of Pecorino cheeses using electronic nose combined with artificial neural network and comparison with GC-MS analysis of volatile compounds FOOD CHEMISTRY, Volume: 129 Issue: 3, 2011, p. 1315-1319, ISSN: 0308-8146
- [36] SHABANI M.O., MAZAHERY A., The ANN application in FEM modeling of mechanical properties of Al-Si alloy APPLIED MATHEMATICAL MODELLING Volume: 35, Issue: 12, 2011, ISSN: 0307-904X
- [37] VAŘACHA, P. a ZELINKA, I. Synthesis of artificial neural networks by the means of evolutionary scanning - preliminary study. In ECMS 2007, Germany : European Council for Modelling and Simulation , 2007. p. 265-270. ISBN/ISSN: 978-0-9553018-2-7.
- [38] BISHOP C.M., Neural Networks for Pattern Recognition, Oxford: Oxford University Press. 1995, ISBN 0-19-853849-9 (hardback) or ISBN 0-19-853864-2 (paperback)

- [39] GURNEY K., An Introduction to Neural Networks London: Routledge. 1997, ISBN 1-85728-673-1 (hardback) or ISBN 1-85728-503-4 (paperback)
- [40] VONK E., ET AL., Integrated Evolutionary Computation with Neural Networks, Electronic Technology Directions to the year 2000, IEEE Computer Society Press, 1995, p. 135-141
- [41] VONK E., JAIN L.C., JOHNSON R.P., Automatic Generation of Neural Network Architecture Using Evolutionary Computation, Advances in Fuzzy Systems – Applications and Theory, Volume 14, 1997, World Science, ISBN: 981-02-3106-7
- [42] GISARIO A., ET AL., Springback control in sheet metal bending by laser-assisted bending: Experimental analysis, empirical and neural network modelling, OPTICS AND LASERS IN ENGINEERING, Volume: 49 Issue: 12, 2011, ISSN: 0143-8166
- [43] LOHMANN R., Structure Evolution in Neural Systems, Dynamic, Genetic and Chaotic Programming, Chapter 15, 1992, p. 1992
- [44] LUND H.H, PARISI D., Simulations with an Evolvable Fitness Formula, Technical Report PCIA-1-94, C.N.R, 1994, Rome
- [45] GARIS H., Genetic Programming Building Nanobrain with Genetically Programmed Neural Network Modules, IEEE International Joint Conference on Neural Networks, Volume 3, 1990, New York, p. 511-516
- [46] HASSOUN M.H., Fundamentals of Artificial Neural Networks, MIT Press, 1995
- [47] MUNRO P.W., Genetic Search for Optimal Representations in Neural Networks, international Conference on Artificial Neural Nets and Genetic Algorithms (ANNGA93), 1993, p. 628-634, Innsbruck, Austria
- [48] MONTANA D.J., Automated Parameter Tuning for Interpretation of Synthetic Images, Handbook of Genetic Algorithms, 1991, p. 202-221
- [49] MONTANA D.J., DAVIS L., Training Feedforward Neural Networks Using Genetic Algorithm, Proceedings of the International Conference on Artificial Intelligence, 1989, p. 762-767

- [50] WHITLEY D., STARKWEATHER T., BOGARD C., Genetic Algorithms and Neural Networks: Optimizing Connections and Connectivity, Parallel Computing, Volume 14, 1990, p. 347-361
- [51] MANIEZZO V., Genetic Evolution of the Topology and Weight Distribution of Neural Networks, IEEE Transaction on Neural Networks, Volume 5, 1994
- [52] DASGUPTA D., MCGREGOR D., R., sGA: A Structured Genetic Algorithm, Technical Report: IKBS-11-93, 1993, Department of Computer Science, University of Strathclyde, Glasgow
- [53] BRAUN H., WEISBOROD J., Evolving Neural Feedforward Networks, International Conference on Artificial Neural Nets and Genetic Algorithms, (ANNGA93), 1993, p. 25-32, Innsbruck, Austria
- [54] NIX A.E, VOSE M.D., Modelling Genetic Algorithms with Markov Chains, Annals of Mathematics and Artificial Intelligence, Volume 5, 1992, p. 79-88
- [55] TURNER E., JACOBSON D.J., TAYLO, J.W., Genetic Architecture of a Reinforced, Postmating, Reproductive Isolation Barrier between Neurospora Species Indicates Evolution via Natural Selection, PLOS GENETICS Volume: 7 Issue: 8, 2011 ISSN: 1553-7390 32
- [56] HAPPEL B.L.M., MURRE J.M.J., Deign and Evolution of Modular Neural Network Architectures, Neural Networks, Volume 7, 1994, p. 985-1004
- [57] ANGELIA P.J., SAUNDERS G.M., POLLACK J.M., An Evolutionary Algorithm that Constructs Recurrent Neural Networks, IEEE Transactions on Neural Networks, Volume 5, 1994
- [58] McDONNELL J.R., WAAGEN D., Evolving Neural Network Connectivity, IEE International Conference on Neural Networks, 1993, San Francisco
- [59] KITANO H., Designing Neural Networks Using Genetic Algorithms with Graph Generation System, Complex Systems, Volume 4, 1990, p. 461-576
- [60] KITANO H., Neurogenetic Learning: An Integrated Method of Designing and Training Neural Networks Using Genetic Algorithms, Physica D, Volume 75, 1994, p. 225-228

- [61] GRUAU F., Genetic Synthesis of Boolean Neural Networks with a Cell Rewriting Development Process, IEEE International Workshop on Combinations of Genetic Algorithms and Neural Networks (DOGAN-92), 1992, Baltimore, p. 55-74
- [62] GRUAU F., Genetic Microprogramming of Neural Networks, Advances in Genetic Programming, 1994, MIT Press
- [63] BOERS E.J.W., KUIPER H., Biological Metaphors and Design of Modular Artificial Neural Networks, Technical report, Department of Computer Science and Experimental and Theoretical Psychology, 1992, Leiden University, The Netherlands
- [64] CANGELOSI A., PARISI D., NOLFI S., Cell Division and Migration on a 'Genotype' for Neural Networks, Networks: computation in neural systems
- [65] HU X., Applications of the general projection neural network in solving extended linear-quadratic programming problems with linear constraints, NEUROCOMPUTING, Volume: 72, 2009, p. 1131-1137 ISSN: 0925-2312
- [66] TSOULOS I., GAVRILIS D., GLAVAS E., Neural network construction and training using grammatical evolution, Neurocomputing Volume 72 Issue 1-3, December, 2008 Pages 269-277 Publisher Elsevier Science Publishers B. V. Amsterdam, The Netherlands, The Netherlands ISSN: 0925-2312
- [67] TURNER, ET AL, Grammatical Evolution of Neural Networks for Discovering Epistasis among Quantitative Trait Loci Evolutionary Computation, Machine Learning and Data Mining in Bioinformatics Book Series Title: Lecture Notes in Computer Science 2010 Publisher: Springer Berlin / Heidelberg, p: 86 – 97.
- [68] VAŘACHA P., Impact of Weather Inputs on Heating Plant - Agglomeration Modeling. In Proceedings of the 10th WSEAS Ing. Conf. on Neural Networks, Athens, WSEAS World Science and Engineering Academy and Science , 2009. p. 159-162. ISBN/ISSN: 978-960-474-065-9.
- [69] CANTÚ, PAZ, Efficient and Accurate Parallel Genetic Algorithms. Kluwer Academic Publishers, 162, 2000.
- [70] ČERVENKA M. Distributed Evolutionary Algorithms. Zlín, Czech Republic, 2006. 114 p. Dissertation. University of Tomas Bata in Zlín.



- [71] XIAO N., ARMSTRONG M., A Specialized Island Model and Its Application in Multiobjective Optimization. Proceedings of the Genetic and Evolutionary Computation Conference, pp:1530-540, Chicago, IL, 2003.
- [72] VAŘACHA, Pavel. Neural Network Synthesis via Asynchronous Analytic Programming. In Recent Researches in Neural Networks, Fuzzy Systems, Evolutionary Computing and Automation: Proceedings of the 12th WSEAS international conference on Neural networks, fuzzy systems, evolutionary computing & automation (NNECFISIC'12). Brasov: WSEAS Press, 2011, 2011. s. 70-75. ISBN 978-960-474-292-9
- [73] VAŘACHA, Pavel. Innovative strategy of SOMA control parameter setting. Proceedings of the 12th WSEAS international conference on Neural networks, fuzzy systems, evolutionary computing & automation (NNECFISIC'12). Brasov: WSEAS Press, 2011, 2011. s. 70-75. ISBN 978-960-474-292-9
- [74] MAGNUS E., HVASS P., Laboratories, Good Parameters for Differential Evolution Technical Report no. HL1002 2010
- [75] KAPP N., SABOURIN R., MAUPIN P., A Dynamic Optimization Approach for Adaptive Incremental Learning INTERNATIONAL JOURNAL OF INTELLIGENT SYSTEMS, Volume: 26, Issue: 11, 2001, p. 1101-1124 ISSN: 0884-8173
- [76] VAŠEK L., ET. AL., The intelligent system controlling an energetic framework of an urban agglomeration, final technical report of National Research Program II No. 2C06007
- [77] VAŘACHA P., JAŠEK R., ANN synthesis for an agglomeration heating power consumption approximation. Proceedings of the 13th WSEAS international conference on Automatic control, modelling & simulation (ACMOS). Lanzarote: WSEAS Press, 2011. p. 239-244. ISBN 978-1-61804-004-6
- [78] ZELINKA I., VAŘACHA P., Synthesis of artificial neural networks by of evolutionary methods. In Workshop ETID 2007 in DEXA 2007 , Germany : IEEE Computer Society, 2007, p. 153-157. ISBN/ISSN: 978-0-7695-2932-5

- [79] PRECHELT L., Proben1—A Set of Neural Network Benchmark Problems and Benchmarking Rules, Universität Karlsruhe, 1994, Germany
- [80] MathWorld Works, Neural Network Toolbox, 2011, [cit. 2011-03-02]. Operators of GA. WWW: <<http://www.mathworks.com/products/neuralnet/index.html>>.
- [81] VAŘACHA P., JAŠEK R., ANN synthesis for an agglomeration heating power consumption approximation. Proceedings of the 13th WSEAS international conference on Automatic control, modelling & simulation (ACMOS). Lanzarote: WSEAS Press, 2011. p. 239-244. ISBN 978-1-61804-004-6
- [82] VAŘACHA P., KRÁL E., HORÁK T., Asynchronous Synthesis of a Neural Network, Handbook of Optimization, Springer, accepted for publication, 2011
- [83] MANGARIANM O.L., WOLBERG W.H., Cancer diagnosis via linear programming, SIAM News, Volume 23, Number 5, 1990, p. 1-18
- [84] JUI-YU W., MIMO CMAC neural network classifier for solving classification problems, Applied Soft Computing, Volume 11, Issue 2, The Impact of Soft Computing for the Progress of Artificial Intelligence, 2011, p. 2326-2333, ISSN 1568-4946
- [85] FALCO D.I., CIOPPA E., Tarantino, Discovering interesting classification rules with genetic programming, Applied Soft Computing 1, 2002 p. 257–269.
- [86] M. Brameier, W. Banzhaf, A comparison of linear genetic programming and neural networks in medical data mining, IEEE Transactions on Evolutionary
- [87] VAŘACHA, Pavel. Neural network synthesis dealing with classification problem. In Recent Researches in Automatic Control: Proceedings of the 13th WSEAS international conference on Automatic control, modelling & simulation (ACMOS). Lanzarote: WSEAS Press, 2011. pp. 377-382. ISBN 978-1-61804-004-6.
- [88] Microsoft [online]. 2011 [cit. 2011-03-02]. .NET Framework. WWW: <<http://msdn.microsoft.com/cs-cz/netframework>
- [89] ZELINKA I., [online]. 2011 [cit. 2011-03-02]. Benchmark test function for EA. WWW: <<http://www.fai.utb.cz/people/zelinka/soma/>>

## 15 LIST OF AUTHOR'S PUBLICATION ACTIVITIES

### 2004

VAŘACHA, P. Paradoxes in mathematical logic. Zlín, 2004. 60 p. Bachelor thesis. Tomas Bata University in Zlín.

### 2006

ZELINKA I., ET AL. STRUCTURAL SYNTHESIS OF NEURAL NETWORK BY MEANS OF ANALYTIC PROGRAMMING. In 12th International Conference on Soft Computing , ČR : Fakulta strojního inženýrství, VUT Brno , 2006. p. 25-30. ISBN/ISSN: 80-214-3195-4

VAŘACHA P., Artificial neuron network synthesis using symbolic regression. Zlín, 2006. 80 p. Master thesis. Tomas Bata University in Zlín.

### 2007

VAŘACHA P., ZELINKA I., Synthesis of the neural networks by the means of the symbolic regression method. In WETDAP 2007 - Znalosti 2007 , Ostrava : Vysoká škola báňská - Technická univerzita , 2007. p. 11-22. ISBN/ISSN: 978-80-248-1332-5.

ZELINKA I., VAŘACHA P., Synthesis of artificial neural networks by of evolutionary methods. In Workshop ETID 2007 in DEXA 2007 , Germany : IEEE Computer Society , 2007. s. 153-157. ISBN/ISSN: 978-0-7695-2932-5.

VAŘACHA P., ZELINKA I. Synthesis of artificial neural networks by the means of evolutionary scanning - preliminary study. In ECMS 2007, Germany: European Council for Modelling and Simulation, 2007. p. 265-270. ISBN/ISSN: 978-0-9553018-2-7.

VAŘACHA, P., ZELINKA, I. DISTRIBUTED SELF-ORGANIZING MIGRATING ALGORITHM (DISOMA). In 8 th International Carpathian Control Conference , . vyd. Košice : Technická univerzita v Košiciach , 2007. s. 738-741. ISBN/ISSN: 978-80-8073-805-1.

## **2008**

VAŘACHA P., ZELINKA I. Analytic Programming Powered by Distributed Self-Organizing Migrating Algorithm Application. In IEEE Proceedings 7th International Conference Computer Information Systems and Industrial Management Applications Ostrava : IEEE Computer Society , 2008. p. 99-100. ISBN/ISSN: 978-0-7695-3184-7.

VAŘACHA P., ZELINKA I., Distributed Self-Organizing Migrating Algorithm Application and Evolutionary Scanning. In Proceedings of the 22nd European Conference on Modelling and Simulation ECMS 2008 Nicosia : European Council of Modelling and Simulation , 2008. p. 201-206. ISBN/ISSN: 0-9553018-5-8.

VAŘACHA P., Evaluation of mathematics textbooks on TBU in Zlín. Zlín, 2008. 57 p. Bachelor thesis. University of Tomas Bata in Zlín.

## **2009**

VAŘACHA, P. ANN Simulation of Agglomeration/Heating Plant Interface. In Internet, competitiveness and Organisational Security in Knowledge Society Zlín : Univerzita Tomáše Bati ve Zlíně , 2009. s. 6. ISBN/ISSN: 978-80-7318-828-3.

VAŘACHA, P. Impact of Weather Inputs on Heating Plant - Agglomeration Modeling. In Proceedings of the 10th WSEAS Ing. Conf. on Neural Networks Athens : WSEAS World Science and Engineering Academy and Science , 2009. p. 159-162. ISBN/ISSN: 978-960-474-065-9.

## **2010**

KRÁL E., ET AL, Using PSO Algorithm for Parameter Identification of Simulation Model of Heat Distribution and Consumption in Municipal Heating Network. In Proceedings of the 21st International DAAAM Symposium "Intelligent Manufacturing & Automation: Focus on Interdisciplinary Solutions" Vienna : DAAAM International Vienna , 2010. p. 1043 - 1044. ISBN/ISSN: 978-3-901509-73-5.

KRÁL E., ET AL. Usage of PSO Algorithm for Parameters Identification of District Heating Network Simulation Model. In 14th WSEAS International Conference on Systems. Latest Trands on Systems. Volume II Rhodes : WSEAS Press (GR) , 2010. p. 657-659. ISBN/ISSN: 978-960-474-214-1.

VAŠEK L., ET AL. Software: Heat consumption predictor. Zlín : Tomas Bata University in Zlín, Faculty of Applied Informatics, 2010.

VAŠEK L., ET AL Software: Simulation model of heat distribution.. Zlín : Tomas Bata University in Zlín, Faculty of Applied Informatics , 2010.

VAŘACHA, P. Software: Distributed Asynchronou Self-Organising Migration Algorithm (DASOMA) Provider. Tomas Bata University in Zlín, Faculty of Applied Informatics, 2010.

## **2011**

VAŘACHA P., Innovative strategy of SOMA control parameter setting. In Recent Researches in Neural Networks, Fuzzy Systems, Evolutionary Computing and Automation: Proceedings of the 12th WSEAS international conference on Neural networks, fuzzy systems, evolutionary computing & automation (NNECFISIC'12). Brasov: WSEAS Press, 2011, p. 70-75. ISBN 978-960-474-292-9

CHRAMCOV B., VAŘACHA P., Use of computer simulation with the aim of achieving more efficient production in manufacturing systems. In 12th WSEAS International Conference on Automation and information, Brasov, Romania, ISBN 978-960-474-292-9

VAŘACHA P., Neural Network Synthesis via Asynchronous Analytic Programming. In Recent Researches in Neural Networks, Fuzzy Systems, Evolutionary Computing and Automation: Proceedings of the 12th WSEAS international conference on Neural networks, fuzzy systems, evolutionary computing & automation (NNECFISIC'12). Brasov: WSEAS Press, 2011, 2011. p. 70-75. ISBN 978-960-474-292-9

VAŘACHA, P., Innovative Strategy of SOMA Control Parameter Setting – Consecutive Studies, Conference on Process Management and The Use of Modern Technologies, Tomas Bata University in Zlín, Czech Republic

VAŘACHA P., JAŠEK, R., ANN synthesis for an agglomeration heating power consumption approximation. In Recent Researches in Automatic Control: Proceedings of the 13th WSEAS international conference on Automatic control, modelling & simulation (ACMOS). Lanzarote: WSEAS Press, 2011. p. 239-244. ISBN 978-1-61804-004-6

VAŘACHA, P. Neural network synthesis dealing with classification problem. In Recent Researches in Automatic Control: Proceedings of the 13th WSEAS international conference on Automatic control, modelling & simulation (ACMOS). Lanzarote: WSEAS Press, 2011. pp. 377-382. ISBN 978-1-61804-004-6.

VAŠEK L., ET. AL., The intelligent system controlling an energetic framework of an urban agglomeration, final technical report of National Research Program II No. 2C06007

VAŘACHA, P., Strategy of SOMA PRT Control Parameter Setting, Scientific Journal Trilobit, Tomas Bata University in Zlín, Czech Republic, accepted for publication, 2011

VAŘACHA P., JAŠEK R., KRÁL E., ANN synthesis for an agglomeration heating power consumption approximation.. International Journal of Mathematical Models and Methods in Applied. ISSN 1998-0140, accepted for publication 2011

VAŘACHA P., KRÁL E., HORÁK T., Asynchronous Synthesis of a Neural Network, Handbook of Optimization, Springer, chapter in the book accepted for publication 2011

### **Citations of the previous publications in journals and important conferences**

Dolinay, V., Pálka, J., Vašek, L., Pivnièková, L.: Importance of sunny days for the determination of heat consumption, North Atlantic University Union, International Journal

of Mathematical Models and Methods in Applied Science, London, 2010, 257-264, ISSN 1998-0140

CHRAMCOV, Bronislav. Heat Demand Forecasting for Concrete District Heating System. International Journal of Mathematical Models and Methods in Applied Sciences [online]. 2010, Volume 4, Issue 4, [cit. 2010-11-30]. Dostupný z WWW: <<http://www.naun.org/journals/m3as/>>. ISSN 1998-0140.

Vašek, L., Dolinay, V.: Simulation model of heat distribution and consumption in municipal heating network, North Atlantic University Union, International Journal of Mathematical Models and Methods in Applied Science, London, 2010, 240-248, ISSN 1998-0140

DOLINAY, Viliam, VASEK, Lubomir. Municipal heating network simulation experiments based on days with similar temperature. In Recent Researches in Automatic Control: Proceedings of the 13th WSEAS international conference on Automatic control, modelling & simulation (ACMOS). Lanzarote: WSEAS Press, 2011. pp. 318-320. ISBN 978-1-61804-004-6.

VASEK, Lubomir, DOLINAY, Viliam. Simulation model of heat distribution and consumption in practical use. In Recent Researches in Automatic Control: Proceedings of the 13th WSEAS international conference on Automatic control, modelling & simulation (ACMOS). Lanzarote: WSEAS Press, 2011. pp. 321-324. ISBN 978-1-61804-004-6.

Dolinay, V.; Vasek, L.; Pivnickova, L.; Palka, J.; Dolinay, J. Sunny Days And Consumed Energy. 12th International Carpathian Control Conference (ICCC). 2011. Velke Karlovice. Czech Republic. s. 78-83. IEEE Catalog Number: CFP1142L-CDR. ISBN: 978-1-61284-359-9

KRÁL, Erik, et al. Usage of peak functions in heat load modeling of district heating system. In Recent Researches in Automatic Control: Proceedings of the 13th WSEAS international conference on Automatic control, modeling & simulation (ACMOS). Lanzarote: WSEAS Press, 2011. pp. 404-406. ISBN 978-1-61804-004-6.

SVĚTINSKÁ, Martina VASEK, Lubomir. Prediction of the intensity of direct solar irradiation. In Recent Researches in Automatic Control: Proceedings of the 13th

WSEAS international conference on Automatic control, modelling & simulation (ACMOS.). Lanzarote: WSEAS Press, 2011. pp. 417-420. ISBN 978-1-61804-004-6.

CHRAMCOV B., Utilization of Mathematica environment for designing the forecast model of heat demand, WSEAS TRANSACTIONS on HEAT and MASS TRANSFER, WSEAS Press, 2011, ISSN: 1790-504

BUCKY R., CHRAMCOV B., Modelling and Simulation of the Order Realization in the Serial Production System, INTERNATIONAL JOURNAL OF MATHEMATICAL MODELS AND METHODS IN APPLIED SCIENCES, WSEAS Press, 2011

### List of supervised and successfully defended master and bachelor thesis

No.	Authors name	Year	B/M	Title
1	Řehák Martin	2007	BT	Security of the Windows XP system
2	Kociánová Gabriela	2007	MT	Database system for the authority of ÚSP Tichá
3	Šoman Robert	2007	MT	Multimedia encyclopedia of machining technology
4	Dohnal Jiří	2008	BT	User guide of content management system
5	Halgaš Rostislav	2008	BT	Description of principle measuring surface material weight and material thickness. Design of panel display
6	Hmirák Michal	2008	BT	3D visualization of Analytic Programming
7	Molnár Zbyněk	2008	BT	3D visualization of artificial intelligence
8	Hladík Michal	2008	MT	Control of Mindstorm NXT robot via Bluetooth technology
9	Kaspříková Eva	2008	MT	Analytic Programming in C#
10	Navrátil Luděk	2008	MT	Education support of GIS



11	Novotný	Miroslav	2008	MT	The GIS as the support of decision-making at the public administration
12	Malinka	Marek	2009	BT	View of topology optimization methods and learning optimization methods of neural networks
13	Rympler	Petr	2009	BT	Communication system using Microsoft .NET
14	Špico	Jaromír	2009	BT	Data mining in energetic industry
15	Hnilica	Marek	2009	MT	The use of GPS navigation and digital communications environment PEGAS-MATRA for AVL system for the exit of the HZS ZLK
16	Hvožd'ara	Martin	2010	BT	Protection of the enterprise network against the outer threats
17	Šálek	Jiří	2010	BT	Biometric identification and RFID in operational of training polygon of The Fire Brigade Rescue Corps of the Zlín region
18	Rympler	Petr	2010	MT	Distributed evolutionary algorithm using .NET platform
19	Sládek	Jan	2010	MT	How to build secure PHP applications
20	Stavinoha	Zdeněk	2010	MT	Implementation of catholic protection system of piping into GIS of company Vodochody a kanalizace Vsetín, a.s.
21	Kolek	Jan	2011	MT	Asynchronous SOMA in Java
22	Malinka	Marek	2011	MT	Neural network synthesis
23	Semenský	Jiří	2011	MT	Data mining software tools analysis

## 16 CURRICULUM VITAE

### PERSONAL PROFILE

**Name** Pavel Vařacha

**Up to date photo**



**Nationality** Czech

**Date of birth** 12 July 1981

**Place of birth** Uherské Hradiště, Czech Republic

**Present address** Na Vyhlídce 1501  
686 05 Uh. Hradiště  
Czech Republic

**Marital status** married

**Contact** phone: +420 777 567 135  
e-mail: varacha@ fai.utb.cz

### EDUCATION

**1997 - 2001** Technical High School in Zlín, specialization: Technical lyceum GCE in Czech language, Mathematics, Physics, Programming

**2001-2004** TBU in Zlín, Faculty of technology, program Informatics Engineering, specialization Information technology, bachelor degree.

**2004-2006** TBU in Zlín, Faculty of applied informatics, program Informatics Engineering, specialization Information technology, master degree.

**2005-2008** TBU in Zlín, University institute, program Specialization in pedagogy,

specialization Teaching of technical subjects, bachelor degree

**2006-present** TBU in Zlín, Faculty of applied informatics, program Chemical and process engineering, specialization Information technology, doctoral program

## **MEMBERSHIP**

IPC member of ECMS 2007 in Prague, Czech Republic

## **EMPLOYMENT**

**2006 - present** Lecturer at TBU in Zlín, Faculty of applied informatics  
Lectures: .NET technology and C#  
Seminars and laboratories: Java, .NET technology and C#, Geographical Information Systems  
Seminars and Laboratories in past: Basic Informatics, Applied Informatics, Mathematical Informatics  
13 master and 10 bachelor thesis successfully supervised

## **PROJECT**

**2006 – 2011** National Research Program II No. 2C06007  
The intelligent system controlling an energetic framework of an urban agglomeration  
(co-investigator, successfully defended in October 2011)

## **COMPETITION**

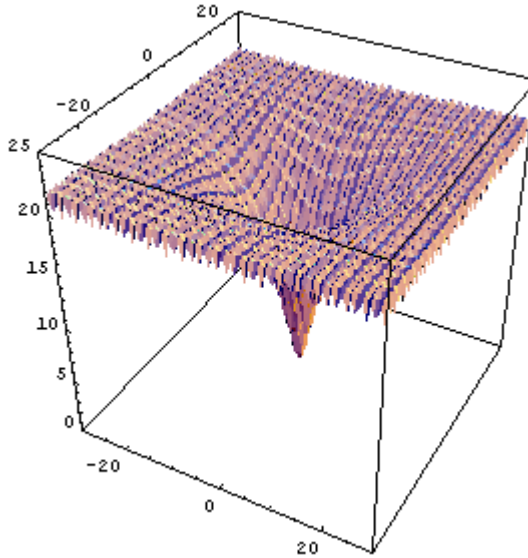
**2011** Finalist of the Joseph Fourier Price 2011 (French embassy and Bull s.r.o. contest )

## **PROFESSIONAL INTERESTS**

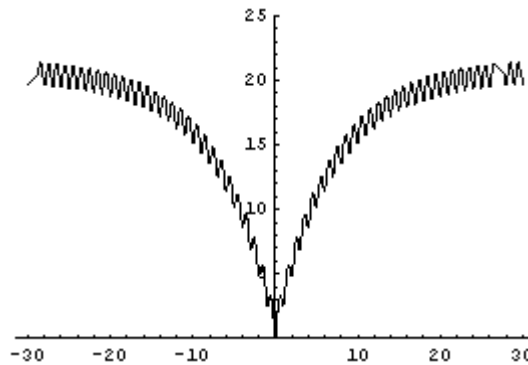
Artificial intelligence  
Nonlinear modeling  
Parallel computations

## 17 APENDIX I TEST FUNCTION VISUALISATION

Appendix I contains 2D and 3D visualizations of the benchmark functions (27) - (36) used in chapters 7 and 9.1.1. Each function is named in accordance with [89], where more detailed information and visualizations can be accessed.



*Fig. 70: Ackley (27) 3D visualization*



*Fig. 71: Ackley (27) 2D visualization*

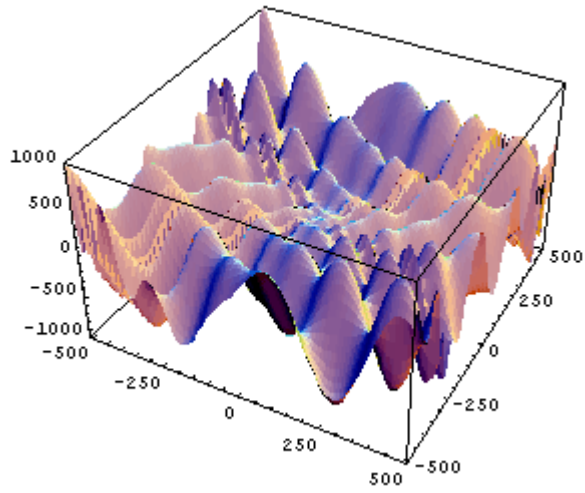


Fig. 72: EggHolder (28) 3D visualization

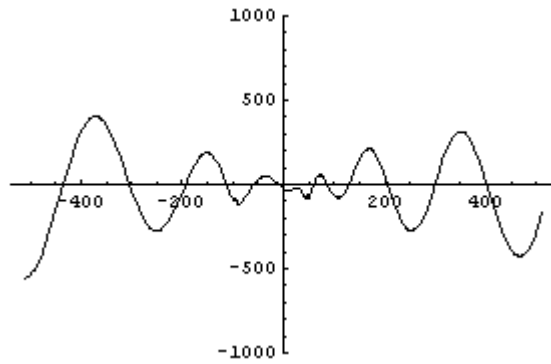


Fig. 73 EggHolder (28) 2D visualization

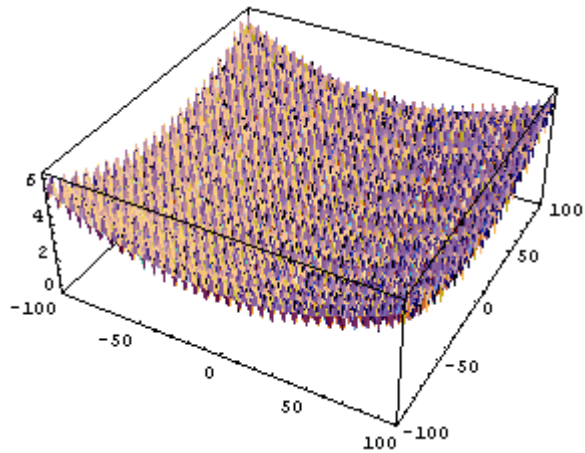


Fig. 74: Michalewicz (29) 3D visualization

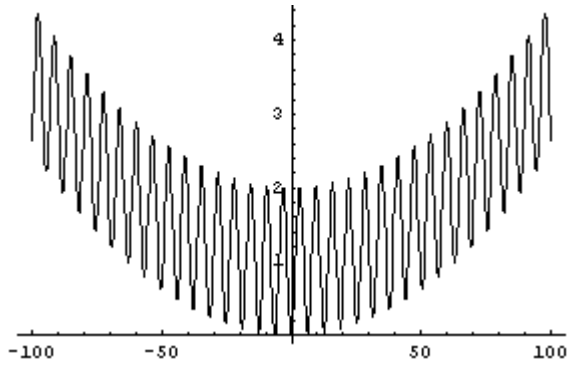


Fig. 75: Michalewicz (29) 2D visualization

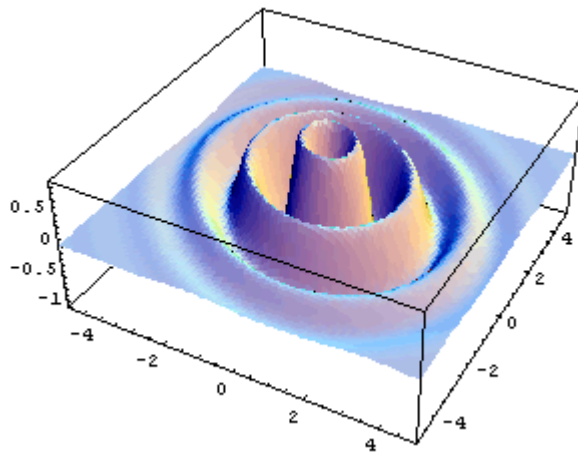


Fig. 76: Masters (30) 3D visualization

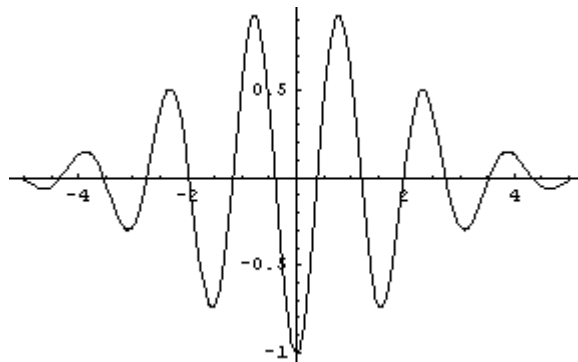
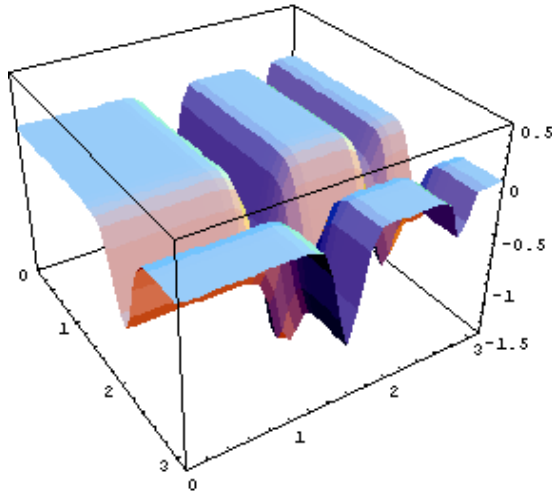
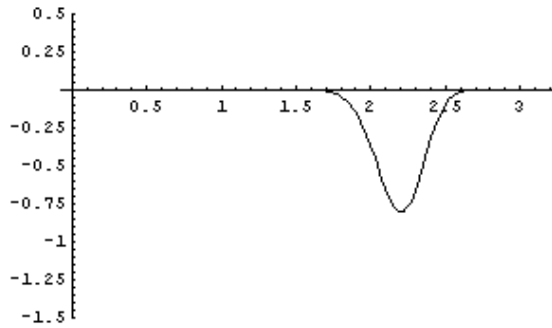


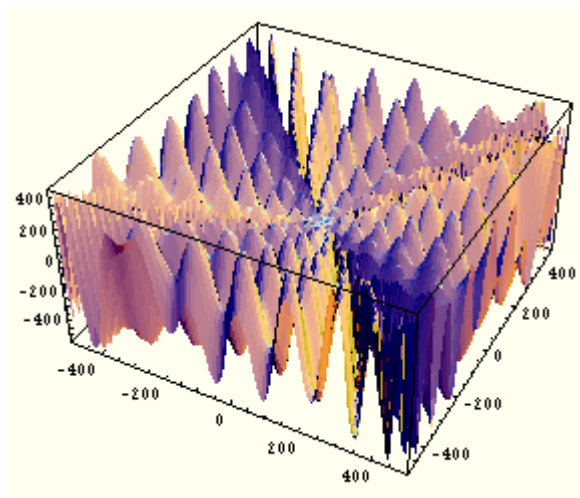
Fig. 77: Masters (30) 2D visualization



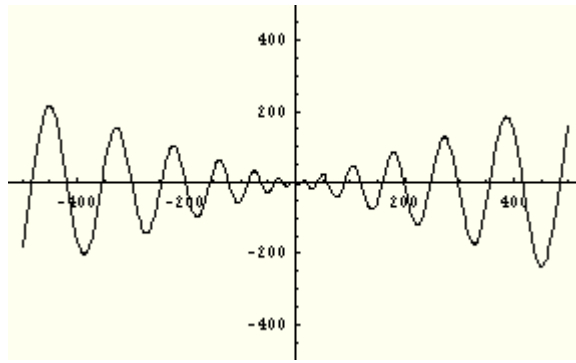
*Fig. 78: Michalewicz (31) 3D visualization*



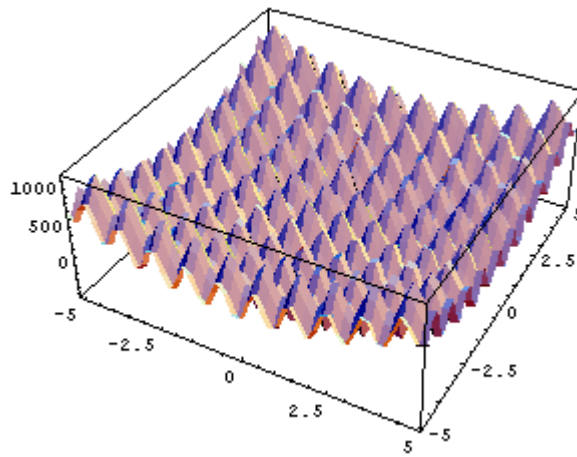
*Fig. 79: Michalewicz (31) 2D visualization*



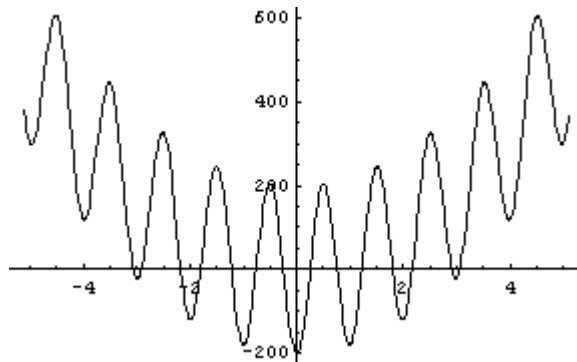
*Fig. 80: Rana (32) 3D visualization*



*Fig. 81: Rana (32) 2D visualization*

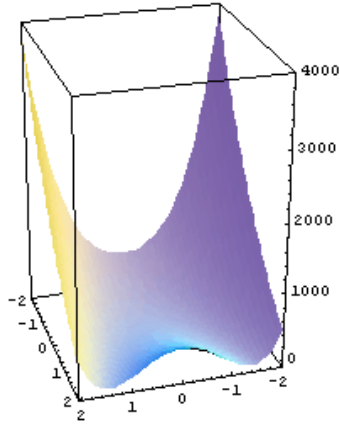


*Fig. 82: Rastrigin (33) 3D visualization*

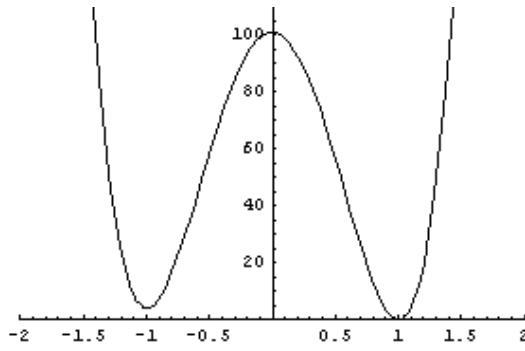


*Fig. 83: Rastrigin (33) 2D visualization*

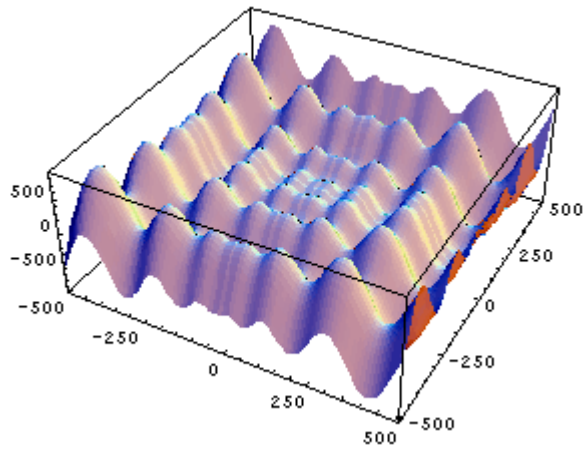




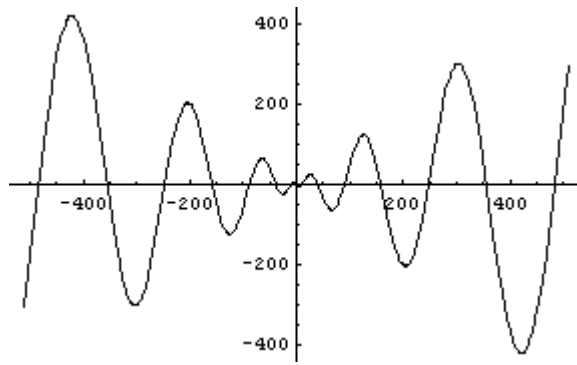
*Fig. 84: Rosenbrock (34) 3D visualization*



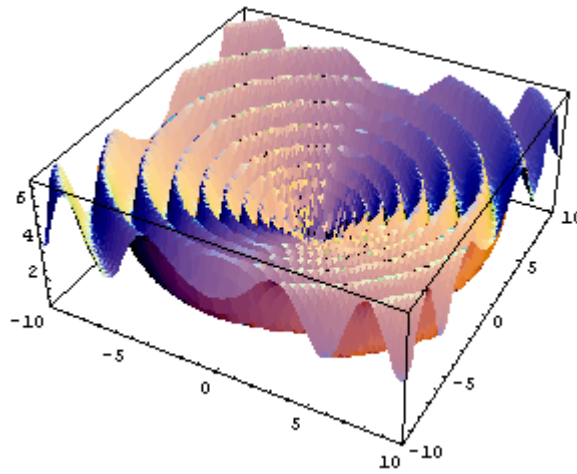
*Fig. 85: Rosenbrock (34) 2D visualization*



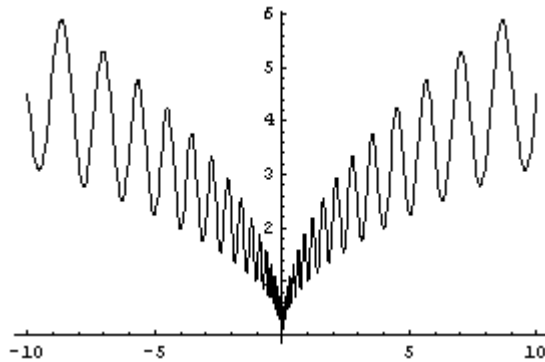
*Fig. 86: Schwefel (35) 2D visualization*



*Fig. 87: Schwefel (35) 2D visualization*



*Fig. 88: SineWave (36) 3D visualization*



*Fig. 89: SineWave (36) 2D visualization*

## 18 APPENDIX II ADAPTIVE PRT STRATEGY

Appendix II complements results connected with a PRT adaptive strategy study discussed in chapter 9.1.1. The results produced by the test functions (27) - (36) (see also Appendix I) are depicted in Fig. 90 (for functions, which prove better results for  $PRT \in \langle 0.005; 0.07 \rangle$ ) and Fig. 91 (functions, which prove better results for  $PRT \in \langle 0.1; 0.3 \rangle$ ).

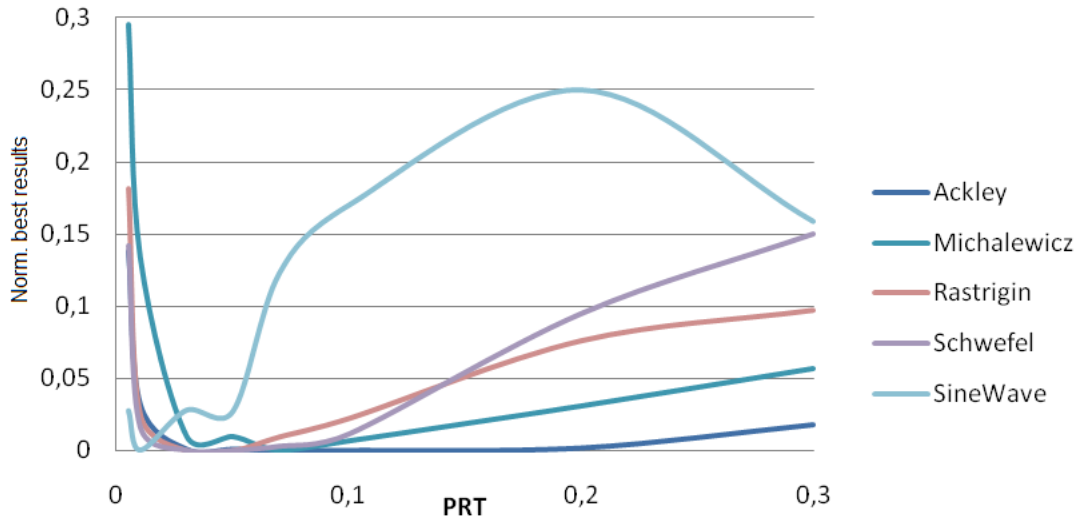


Fig. 90: Test functions providing the best results for  $PRT \in \langle 0.005; 0.07 \rangle$

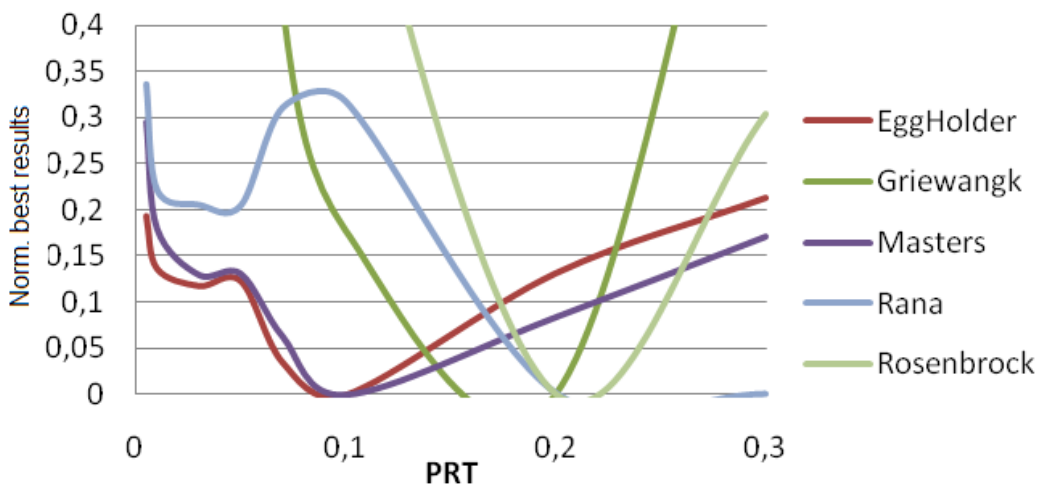


Fig. 91: Test functions providing the best results for  $PRT \in \langle 0.1; 0.3 \rangle$

Table 19: Best results for different cost functions and PRT settings

Function, PRT:	0,005	0,01	0,03	0,05	0,07	0,1	0,2	0,3
Ackley (27)	3829,415	3473,757	3369,026	3369,049	3366,263	3366,142	3372,042	3426,071
EggHolder (28)	-54910,7	-58787,2	-60078,2	-59717,5	-65743,9	-68130,2	-59140,2	-53573
Griewangk (29)	21,23287	6,371556	1,112177	1,123117	0,754679	0,625381	0,531147	0,956176
Masters (30)	-59,1238	-68,7604	-73,0045	-72,9118	-78,6305	-83,9028	-76,8502	-69,5354
Michalewicz (31)	-70,2537	-86,3075	-98,671	-98,6997	-99,6452	-98,9955	-96,5857	-94,0024
Rana (32)	-23409,9	-27426,6	-28016,2	-28044,2	-24296,1	-24067	-35200,9	-35238,2
Rastrigin (33)	-818186	-973899	-999334	-999392	-989897	-977084	-923366	-902252
Rosenbrock (34)	12362,34	5749,791	1134,754	1107,528	329,3666	234,7636	140,0399	182,5677
Schwefel (35)	-35958,3	-41214,8	-41894,4	-41894,4	-41778,7	-41423,7	-37932,6	-35622,5
SineWave (36)	-621,61	-639,085	-621,393	-621,886	-560,68	-530,483	-479,456	-537,727

Table 20: Normalized best results for different cost functions and PRT settings

Function, PRT:	0,005	0,01	0,03	0,05	0,07	0,1	0,2	0,3
Ackley (27)	0,137627	0,03197	0,000857	0,000864	3,58E-05	0	0,001753	0,017803
EggHolder (28)	0,194033	0,137134	0,118185	0,123479	0,035025	0	0,131953	0,213668
Griewangk (29)	38,97551	10,99584	1,093916	1,114512	0,420847	0,177416	0	0,80021
Masters (30)	0,29533	0,180476	0,129891	0,130997	0,062838	0	0,084057	0,171238
Michalewicz (31)	0,294961	0,133852	0,009776	0,009488	0	0,006519	0,030703	0,056629
Rana (32)	0,335666	0,221679	0,204946	0,204153	0,310518	0,317019	0,001056	0
Rastrigin (33)	0,181316	0,025508	5,74E-05	0	0,0095	0,022322	0,076072	0,097199
Rosenbrock (34)	87,27727	40,05824	7,103078	6,908659	1,351949	0,676405	0	0,303684
Schwefel (35)	0,141692	0,016222	0	1,03E-06	0,002762	0,011235	0,094565	0,149709
SineWave (36)	0,027344	0	0,027684	0,026912	0,122684	0,169934	0,249778	0,158599

Functions providing better result for  $PRT \in <0.005; 0.07>$  are marked in red and functions providing better results for  $PRT \in <0.1; 0.3>$  are marked in yellow.

Table 21: Average results for different cost functions and PRT settings

Function, PRT	0,005	0,01	0,03	0,05	0,07	0,1	0,2	0,3
Ackley (27)	3895,704	3499,265	3370,977	3370,963	3366,779	3368,098	3416,55	3533,109
EggHolder (28)	-53445,7	-57731,3	-58093,7	-58081,4	-55605,2	-63855,5	-53564,5	-48770,3
Griewangk (29)	25,14932	8,465231	1,191455	1,191591	0,972961	0,872625	0,978322	1,181165
Masters (30)	-55,9965	-66,5651	-70,3361	-70,2997	-71,631	-77,7542	-71,6636	-63,4593
Michalewicz (31)	-67,8213	-84,3369	-97,9382	-97,9319	-98,976	-97,9913	-94,088	-89,7736
Rana (32)	-22468,3	-26526,3	-27303,9	-27274	-23340,3	-21400,8	-32648,8	-33216,1
Rastrigin (33)	-760773	-959770	-997738	-998062	-980095	-958457	-892831	-838771
Rosenbrock (34)	15084,54	7097,92	1369,023	1386,179	471,804	335,933	250,1202	324,7303
Schwefel (35)	-35061,2	-40778,2	-41888,1	-41888,6	-41438,1	-40531,1	-36620,3	-33259
SineWave (36)	-609,751	-634,028	-614,379	-615,275	-549,637	-519,919	-463,773	-442,679

Table 22: Normalized average results for different cost functions and PRT settings

Functin, PRT	0,005	0,01	0,03	0,05	0,07	0,1	0,2	0,3
Ackley (27)	0,157101	0,039351	0,001247	0,001243	0	0,000392	0,014783	0,049403
EggHolder (28)	0,163021	0,095907	0,090232	0,090424	0,129203	0	0,161161	0,23624
Griewangk (29)	27,82032	8,700884	0,365369	0,365525	0,114982	0	0,121126	0,353577
Masters (30)	0,279826	0,143903	0,095404	0,095872	0,07875	0	0,078331	0,183847
Michalewicz (31)	0,31477	0,147905	0,010485	0,010549	0	0,009949	0,049385	0,092976
Rana (32)	0,323572	0,201402	0,177991	0,178892	0,29732	0,355711	0,017078	0
Rastrigin (33)	0,23775	0,038366	0,000324	0	0,018001	0,039682	0,105436	0,1596
Rosenbrock (34)	59,30918	27,37804	4,473462	4,542053	0,88631	0,343087	0	0,298297
Schwefel (35)	0,162988	0,026507	1,16E-05	0	0,010753	0,032406	0,125768	0,206013
SineWave (36)	0,038291	0	0,030992	0,029578	0,133104	0,179976	0,26853	0,301799

Functions providing better result for  $PRT \in \langle 0.005; 0.07 \rangle$  are marked in red and functions providing better results for  $PRT \in \langle 0.1; 0.3 \rangle$  are marked in yellow.

## 19 APPENDIX III – ANN SYNTHESIS RESULTS

Appendix III contains mathematical descriptions (91) - (94) of the evolution loops (58) - (61) discussed in chapter 10.3.

Mathematical descriptions of (58):

$$\begin{aligned}
 \text{ANN3} = & w \frac{e^{2\lambda (wt + \phi)} - 1}{e^{2\lambda (wt + \phi)} + 1} + w \mathcal{G}_{\text{ex}} + w \frac{e^{2\lambda (wt + \phi)} - 1}{e^{2\lambda (wt + \phi)} + 1} + \\
 & e^{2\lambda \left( \frac{e^{2\lambda (wt + \phi)} - 1}{e^{2\lambda (wt + \phi)} + 1} + \frac{2\lambda \left( w \frac{e^{2\lambda (wt + \phi)} - 1}{e^{2\lambda (wt + \phi)} + 1} + \phi \right) - 1}{e^{2\lambda \left( w \frac{e^{2\lambda (wt + \phi)} - 1}{e^{2\lambda (wt + \phi)} + 1} + \phi \right) + 1}} + \phi \right)} - 1 \\
 & + w \frac{e^{2\lambda \left( \frac{e^{2\lambda (wt + \phi)} - 1}{e^{2\lambda (wt + \phi)} + 1} + \frac{2\lambda \left( w \frac{e^{2\lambda (wt + \phi)} - 1}{e^{2\lambda (wt + \phi)} + 1} + \phi \right) - 1}{e^{2\lambda \left( w \frac{e^{2\lambda (wt + \phi)} - 1}{e^{2\lambda (wt + \phi)} + 1} + \phi \right) + 1}} + \phi \right)} - 1}{e^{2\lambda \left( \frac{e^{2\lambda (wt + \phi)} - 1}{e^{2\lambda (wt + \phi)} + 1} + \frac{2\lambda \left( w \frac{e^{2\lambda (wt + \phi)} - 1}{e^{2\lambda (wt + \phi)} + 1} + \phi \right) - 1}{e^{2\lambda \left( w \frac{e^{2\lambda (wt + \phi)} - 1}{e^{2\lambda (wt + \phi)} + 1} + \phi \right) + 1}} + \phi \right)} + 1} + \\
 & + w \frac{e^{2\lambda \left( w \frac{e^{2\lambda (wt + \phi)} - 1}{e^{2\lambda (wt + \phi)} + 1} + \phi \right)} - 1}{e^{2\lambda \left( w \frac{e^{2\lambda (wt + \phi)} - 1}{e^{2\lambda (wt + \phi)} + 1} + \phi \right)} + 1} +
 \end{aligned} \tag{91}$$



Mathematical description of (59):

$$\begin{aligned}
 \text{ANN4} = & w \frac{e^{2\lambda(wt + \phi)} - 1}{e^{2\lambda(wt + \phi)} + 1} + w\theta_{\text{ex}} + w \frac{e^{2\lambda(wt + \phi)} - 1}{e^{2\lambda(wt + \phi)} + 1} + \\
 & + w \frac{e^{2\lambda \left( w \frac{e^{2\lambda(wt + \phi)} - 1}{e^{2\lambda(wt + \phi)} + 1} + \phi \right)} - 1}{e^{2\lambda \left( w \frac{e^{2\lambda(wt + \phi)} - 1}{e^{2\lambda(wt + \phi)} + 1} + \phi \right)} + 1} + \\
 & + w \frac{e^{2\lambda \left( w \frac{e^{2\lambda(wt + \phi)} - 1}{e^{2\lambda(wt + \phi)} + 1} + \phi \right)} - 1}{e^{2\lambda \left( w \frac{e^{2\lambda(wt + \phi)} - 1}{e^{2\lambda(wt + \phi)} + 1} + \phi \right)} + 1} + \\
 & + w \frac{e^{2\lambda \left( w \frac{e^{2\lambda(wt + \phi)} - 1}{e^{2\lambda(wt + \phi)} + 1} + \phi \right)} - 1}{e^{2\lambda \left( w \frac{e^{2\lambda(wt + \phi)} - 1}{e^{2\lambda(wt + \phi)} + 1} + \phi \right)} + 1} + \\
 & + \frac{e^{2\lambda \left( w \frac{e^{2\lambda(w\theta_{\text{ex}} + \phi)} - 1}{e^{2\lambda(w\theta_{\text{ex}} + \phi)} + 1} + w \frac{e^{2\lambda(w\theta_{\text{ex}} + \phi)} - 1}{e^{2\lambda(w\theta_{\text{ex}} + \phi)} + 1} + wt + \phi \right)} - 1}{e^{2\lambda \left( w \frac{e^{2\lambda(w\theta_{\text{ex}} + \phi)} - 1}{e^{2\lambda(w\theta_{\text{ex}} + \phi)} + 1} + w \frac{e^{2\lambda(w\theta_{\text{ex}} + \phi)} - 1}{e^{2\lambda(w\theta_{\text{ex}} + \phi)} + 1} + wt + \phi \right)} + 1}
 \end{aligned} \tag{92}$$



Mathematical description of (61):

$$\begin{aligned}
 \text{ANN6} = & w \frac{e^{2\lambda(wt + \phi)} - 1}{e^{2\lambda(wt + \phi)} + 1} + w\theta_{\text{ex}} + w \frac{e^{2\lambda(wt + \phi)} - 1}{e^{2\lambda(wt + \phi)} + 1} + \\
 & e^{2\lambda \left( \frac{e^{2\lambda(wt + \phi)} - 1}{e^{2\lambda(wt + \phi)} + 1} + \frac{e^{2\lambda \left( w \frac{e^{2\lambda(wt + \phi)} - 1}{e^{2\lambda(wt + \phi)} + 1} + \phi \right) - 1}}{e^{2\lambda \left( w \frac{e^{2\lambda(wt + \phi)} - 1}{e^{2\lambda(wt + \phi)} + 1} + \phi \right) + 1}} \right) - 1} + \\
 & + w \frac{e^{2\lambda \left( \frac{e^{2\lambda(wt + \phi)} - 1}{e^{2\lambda(wt + \phi)} + 1} + \frac{e^{2\lambda \left( w \frac{e^{2\lambda(wt + \phi)} - 1}{e^{2\lambda(wt + \phi)} + 1} + \phi \right) - 1}}{e^{2\lambda \left( w \frac{e^{2\lambda(wt + \phi)} - 1}{e^{2\lambda(wt + \phi)} + 1} + \phi \right) + 1}} \right) + 1}}{e^{2\lambda \left( w \frac{e^{2\lambda(wt + \phi)} - 1}{e^{2\lambda(wt + \phi)} + 1} + \phi \right) - 1}} + \\
 & + w \frac{e^{2\lambda \left( w \frac{e^{2\lambda(wt + \phi)} - 1}{e^{2\lambda(wt + \phi)} + 1} + \phi \right) - 1}}{e^{2\lambda \left( w \frac{e^{2\lambda(wt + \phi)} - 1}{e^{2\lambda(wt + \phi)} + 1} + \phi \right) + 1}} + \\
 & + \frac{e^{2\lambda \left( w \frac{e^{2\lambda(w\theta_{\text{ex}} + \phi)} - 1}{e^{2\lambda(w\theta_{\text{ex}} + \phi)} + 1} + w \frac{e^{2\lambda(w\theta_{\text{ex}} + \phi)} - 1}{e^{2\lambda(w\theta_{\text{ex}} + \phi)} + 1} + wt + \phi \right) - 1}}{e^{2\lambda \left( w \frac{e^{2\lambda(w\theta_{\text{ex}} + \phi)} - 1}{e^{2\lambda(w\theta_{\text{ex}} + \phi)} + 1} + w \frac{e^{2\lambda(w\theta_{\text{ex}} + \phi)} - 1}{e^{2\lambda(w\theta_{\text{ex}} + \phi)} + 1} + wt + \phi \right) + 1}} + \\
 & + w \frac{e^{2\lambda \left( w \frac{e^{2\lambda(wt + \phi)} - 1}{e^{2\lambda(wt + \phi)} + 1} + \frac{e^{2\lambda \left( w\theta_{\text{ex}} + w \frac{e^{2\lambda(wt + \phi)} - 1}{e^{2\lambda(wt + \phi)} + 1} + \phi \right) - 1}}{e^{2\lambda \left( w\theta_{\text{ex}} + w \frac{e^{2\lambda(wt + \phi)} - 1}{e^{2\lambda(wt + \phi)} + 1} + \phi \right) + 1}} \right) - 1}}{e^{2\lambda \left( w \frac{e^{2\lambda(wt + \phi)} - 1}{e^{2\lambda(wt + \phi)} + 1} + \frac{e^{2\lambda \left( w\theta_{\text{ex}} + w \frac{e^{2\lambda(wt + \phi)} - 1}{e^{2\lambda(wt + \phi)} + 1} + \phi \right) - 1}}{e^{2\lambda \left( w\theta_{\text{ex}} + w \frac{e^{2\lambda(wt + \phi)} - 1}{e^{2\lambda(wt + \phi)} + 1} + \phi \right) + 1}} \right) + 1}} + \\
 & + w \frac{e^{2\lambda \left( wt + w \frac{e^{2\lambda(wt + w\theta_{\text{ex}} + \phi)} - 1}{e^{2\lambda(wt + w\theta_{\text{ex}} + \phi)} + 1} + \phi \right) - 1}}{e^{2\lambda \left( wt + w \frac{e^{2\lambda(wt + w\theta_{\text{ex}} + \phi)} - 1}{e^{2\lambda(wt + w\theta_{\text{ex}} + \phi)} + 1} + \phi \right) + 1}}
 \end{aligned} \tag{94}$$

## 20 APPENDIX IV – SUPER MICRO SERVER

Appendix IV contains Super Micro server pictures and technical specification.




*Fig. 92: Super Micro server*



*Fig. 93: Super Micro server motherboard*

Table 23: Super Micro server technical specification

<b><u>Product SKUs</u></b>	
<b>AS-1042G-TF</b>	A+ Server 1042G-TF (Black)
<b><u>Motherboard</u></b>	
<b>Product SKUs</b>	Super H8QGi+-F
<b>Form Factor</b>	SWTX
<b>Dimensions</b>	16.48" x 13" (41.9cm x 33.0cm)
<b><u>Processor/Chipset</u></b>	
<b>CPU</b>	Quad 1944-pin Socket G34
	Supports up to four <b>Twelve/Eight-Core ready AMD Opteron™ 6100 Series processors</b>  
<b>Chipset</b>	HT3.0 Link support
<b><u>System Memory</u></b>	
<b>Memory Capacity</b>	Thirty-Two DIMM sockets
	Support up to 512GB DDR3 Reg. ECC


	1333/1066/800 MHz memory or 128GB of DDR3 Unb. ECC/non-ECC memory
	Quad channel memory bus
	For Dual or Quad CPUs: Recommended that memory be populated equally in adjacent memory banks
<b>Memory Type</b>	Registered ECC or unb. ECC / non-ECC DDR3 1333/1066/800 MHz SDRAM 72-bit, 240-pin gold-plated DIMMs
<b>DIMM Sizes</b>	1GB, 2GB, 4GB, 8GB, 16GB
<b>Memory Voltage</b>	1.35V or 1.5V
<b>Error Detection</b>	Corrects single-bit errors
	Detects double-bit errors (using ECC

	memory)
<b><u>On-Board Devices</u></b>	
<b>SATA</b>	AMD SP5100 (RAID 0, 1, 10)
<b>IPMI</b>	Support for Intelligent Platform Management Interface v.2.0
	IPMI 2.0 with virtual media over LAN and KVM over LAN support
	Winbond® WPCM450 BMC
<b>Network Controllers</b>	Intel® 82576 controller, Dual-Port Gigabit Ethernet
	10/100/1000BASE-T support
<b>VGA</b>	Matrox G200 16MB DDR2 graphics
<b>Super I/O</b>	Winbond® W83527 chip
<b><u>Input / Output</u></b>	
<b>SATA</b>	6x SATA2.0 (3Gb/s) Ports
<b>LAN</b>	2 RJ45 LAN ports
	1 RJ45 Dedicated LAN supports IPMI

<b>USB</b>	7x USB 2.0 ports
	2x Rear, 4x internal header, and 1x type A
<b>Keyboard / Mouse</b>	PS/2 keyboard and mouse ports
<b>Serial Ports</b>	1x Fast UART 16550 serial port
	1x serial port header
<b><u>Expansion Slots</u></b>	
<b>PCI-Express</b>	1x PCI-e 2.0 x16
<b><u>System BIOS</u></b>	
<b>BIOS Type</b>	16Mb SPI Flash ROM with AMI® BIOS
<b>BIOS Features</b>	Plug and Play (PnP)
	DMI 2.3
	PCI 2.2
	ACPI 2.0
	USB Keyboard Support
	SMBIOS 2.3
<b><u>Chassis</u></b>	
<b>Form Factor</b>	1U Rackmount
<b>Model</b>	SC818TQ-1400LPB


<b>Dimensions</b>	
<b>Height</b>	1.7" (43mm)
<b>Width</b>	17.2" (437mm)
<b>Depth</b>	27.75" (705mm)
<b>Gross Weight</b>	43 lbs (19.5 kg)
<b>Available Colors</b>	Black
<b><u>Front Panel</u></b>	
<b>Buttons</b>	Power On/Off button
	System Reset button
<b>LEDs</b>	Power LED
	Hard drive activity LED
	2x Network activity LEDs
	System Overheat LED
<b>Ports</b>	2x Front USB Ports
	1x Serial COM Port
<b><u>Drive Bays</u></b>	
<b>Hot-swap</b>	3x 3.5" hot-swap SATA drive bays
	Enterprise SATA HDD only recommended

<b><u>Peripheral Drives</u></b>	
<b>DVD-ROM</b>	Slim DVD-ROM drive (optional)
<b><u>Backplane</u></b>	
	SAS HDD Backplane with SES2
<b><u>System Cooling</u></b>	
<b>Fans</b>	6x heavy-duty counter-rotating PWM fans with optimal fan speed control
<b><u>Power Supply</u></b>	
	1400W high-efficiency power supply with PMBus
<b>AC Input</b>	1200W: 100 - 140V, 50 - 60Hz, 10.5 - 14.7 Amp
	1400W: 180 - 240V, 50 - 60Hz, 7.2 - 9.5 Amp
<b>DC Output +5V standby</b>	4 Amp
<b>DC Output +12V</b>	100 Amp @ 100-140V
	117 Amp @ 180-

	240V
<b>Certification</b>	80 PLUS Gold Certified 
<b><u>PC Health Monitoring</u></b>	
<b>CPU</b>	Monitors CPU Core Voltages, +1.8V, +3.3V, +5V, ±12V, +3.3V Standby, -12V Standby, VBAT, HT, memory, chipset
	CPU switching voltage regulator
<b>FAN</b>	Up to 9-fan status tachometer monitoring
	Up to nine 4-pin fan headers
	Status monitor for speed control
	3-pin fan support (w/o speed control)
	Low noise fan speed control (4-pin fan only)

	Pulse Width Modulated (PWM) fan connectors
<b>Temperature</b>	Monitoring for CPU and chassis environment
	CPU Thermal Trip Support
	Thermal control for 9x fan connectors
	1°C Temperature Sensing Logic
<b>LED</b>	CPU / System Overheat LED
	+5V Standby Alert LED
<b>Other Features</b>	Chassis Intrusion Detection
	Chassis Intrusion Header
<b><u>Management</u></b>	
<b>Software</b>	PMI (Intelligent Platform Management Interface) 2.0
	Super Doctor III
<b>Power</b>	ACPI Power



<b>Configurations</b>	Management
	Wake-On-LAN (WOL) header
	Keyboard Wakeup from Soft-Off
	Power-on mode control for AC power loss recovery
<b><u>Operating Environment / Compliance</u></b>	
<b>RoHS</b>	<p>RoHS Compliant</p> <p>6/6, Pb Free</p> 
<b>Environmental Specifications</b>	<p>Operating Temperature:</p> <p>10°C to 35°C (50°F to 95°F)</p>
	Non-operating

	<p>Temperature:</p> <p>-40°C to 70°C (-40°F to 158°F)</p>
	<p>Operating Relative Humidity:</p> <p>8% to 90% (non-condensing)</p>
	<p>Non-operating Relative Humidity:</p> <p>5% to 95% (non-condensing)</p>
<b><u>Regulatory</u></b>	
<b>FCC</b>	Passed to meet FCC standard requirement

## 21 APPENTIX V – ASYNCHRONOUS SOMA IN C#

Appendix V contains a complete code of asynchronous SOMA in C# used through the practical part of the thesis including the reader/writer lock mechanisms and other thread connected arrangements.

```
static void DISOMAWork(Object _parameters)
{
    Parameters parameters = (Parameters)_parameters;
    int NP = parameters.NP;
    if (NP < 1) NP = 1;
    Random random = new Random(parameters.seed);
    Individual[] population = new Individual[NP];
    double[] randomPosition;
    double PRT;
    int sum;

    for (int i = 0; i < NP; i++)
    {
        randomPosition = new double[specimen.Length];
        for (int y = 0; y < specimen.Length; y++)
        {
            randomPosition[y] = random.NextDouble() * (specimen[y].max -
specimen[y].min) + specimen[y].min;
        }
        population[i] = model.costFunction(randomPosition);
        bestHistoryLock.AcquireReaderLock(Timeout.Infinite);
        if (population[i].costValue < bestHistory[0])
        {
            bestHistoryLock.ReleaseReaderLock();
            bestHistoryLock.AcquireWriterLock(Timeout.Infinite);
            if (population[i].costValue < bestHistory[0]) bestHistory[0] =
population[i].costValue;
            bestHistoryLock.ReleaseWriterLock();
        }
        else
        {
            bestHistoryLock.ReleaseReaderLock();
        }
        leaderLock.AcquireReaderLock(Timeout.Infinite);

        if (population[i].costValue < leader.costValue)
        {
            leaderLock.ReleaseReaderLock();
            leaderLock.AcquireWriterLock(Timeout.Infinite);
            if (population[i].costValue < leader.costValue) leader =
population[i];
        }
    }
}
```

```

    leaderString1 = AP.toString(new Token(leader.position, AP.GFS, null,
null));
    leaderString2 = AP.toString(new Token(leader.position, AP.GFS, null,
leader.constants));
    leaderValues = new List<double>();

    Double value;

    for (int k = 0; k < AP.dataValid.Length; k++)
    {
        value = AP.evaluate(new Token(leader.position, AP.GFS,
AP.dataValid[k].inputs, leader.constants)).value;
        if(value <= 0) leaderValues.Add(0);
        else leaderValues.Add(1);
    }
    leaderLock.ReleaseWriterLock();
}
else
{
    leaderLock.ReleaseReaderLock();
}
}

double[] distance = new double[specimen.Length];
int[] PRTVector = new int[specimen.Length];
double[] jump;
double coordinate;
Individual bestClone;
Individual clone;
while (true)
{
    for (int i = 0; i < NP; i++)
    {
        leaderLock.AcquireReaderLock(Timeout.Infinite);
        if (leader.Equals(population[i]))
        {
            leaderLock.ReleaseReaderLock();
            continue;
        }

        for (int y = 0; y < specimen.Length; y++)
        {
            distance[y] = leader.position[y] - population[i].position[y];
        }
        leaderLock.ReleaseReaderLock();
        PRT = 1 / (double)population[i].deep + 1;
        for (int y = 0; y < specimen.Length; y++)
        {
            if (random.NextDouble() > PRT) PRTVector[y] = 0;
            else PRTVector[y] = 1;
        }
    }
}

```

```

}
sum = 0;

for (int k = 0; k <= population[i].deep; k++)
{
    sum += PRTVector[k];
}
while (sum == 0)
{
    for (int y = 0; y < specimen.Length; y++)
    {
        if (random.NextDouble() > PRT) PRTVector[y] = 0;
        else PRTVector[y] = 1;
    }
    sum = 0;
    for (int k = 0; k <= population[i].deep; k++)
    {
        sum += PRTVector[k];
    }
}
bestClone = new Individual(null, population[i].costValue);
for (int n = 1; n < (mass / step); n++)
{
    jump = new double[specimen.Length];
    for (int y = 0; y < specimen.Length; y++)
    {
        coordinate = population[i].position[y] + (distance[y] * step *
PRTVector[y] * n);
        if ((coordinate < specimen[y].min) || (coordinate > specimen[y].max))
        {
            coordinate = random.NextDouble() * (specimen[y].max - specimen[y].min)
+ specimen[y].min;
        }
        jump[y] = coordinate;
    }

    clone = model.costFunction(jump);
    if (clone.costValue < bestClone.costValue) bestClone = clone;

    Interlocked.Increment(ref counter);
    Interlocked.Increment(ref interCounter);
    lock ("interCounter")
    {
        if (interCounter >= period * (DISOMA.NP - 1) * mass / step)
        {
            interCounter = 0;
            bestHistoryLock.AcquireWriterLock(Timeout.Infinite);
            leaderLock.AcquireReaderLock(Timeout.Infinite);
            bestHistory.Add(leader.costValue);
            leaderLock.ReleaseReaderLock();
        }
    }
}

```



## 22 APPENDIX VI – XML RESULT FORMAT

Appendix VI contains a typical example of an asynchronous SOMA result saved in the standard XML format:

```
<?xml version="1.0"?>

<Report xmlns:xsi="http://www.w3.org/2001/XMLSchema-
instance" xmlns:xsd="http://www.w3.org/2001/XMLSchema">

  <best>

    <algorithm>SOMA - All To One</algorithm>

    <finalLeader>

      <position>

        <double>0.0025934393220287959</double>

        <double>0.00012487152843137684</double>

        <double>-0.000336864255269125</double>

        <double>-0.00020993156585577457</double>

        .

        .

        .

        <double>0.00069229082615153859</double>

        <double>0.00083273363376060519</double>

      </position>

      <costValue>3368.0650139215759</costValue>

    </finalLeader>

    <finalEvaluations>648060</finalEvaluations>

    <model>costFunctions.Ackley</model>
```

```
<NP>60</NP>

<step>0.11</step>

<mass>3</mass>

<PRT>0.1</PRT>

<numberOfProcessors>1</numberOfProcessors>

</best>

<worst>

  <algorithm>SOMA - All To One</algorithm>

  <finalEvaluations>648060</finalEvaluations>

  <model>costFunctions.Ackley</model>

  <NP>60</NP>

  <step>0.11</step>

  <mass>3</mass>

  <PRT>0.1</PRT>

  <numberOfProcessors>1</numberOfProcessors>

</worst>

<everage>3368.0650139215759</everage>

<solutions>

  <double>3368.0650139215759</double>

  <double>3368.0650139215759</double>

  <double>3368.0650139215759</double>

</solutions>

</Report>
```