

Comparison and Analysis of Mutation-based Evolutionary Algorithms for ANN Parameters Optimization

Kristina Davoian, Alexander Reichel, Wolfram - M. Lippe

Abstract—Mutation-based Evolutionary Algorithms, also known as Evolutionary Programming (EP) are commonly applied to Artificial Neural Networks (ANN) parameters optimization. This paper presents a comparative analysis of mutation approaches, based on the different distributions for the purpose to examine their performance for ANNs with the predefined architectures, evaluate the average improvement of chromosomes and investigate their ability to find solutions with the high precision. The experiments have been provided for training feed-forward ANNs using the XOR as a case problem. Besides established EP techniques, i.e. the classical EP (CEP), based on a standard normal distribution, the Fast EP (FEP), based on the Cauchy distribution, the improved FEP (IFEP), based on both Cauchy and Gaussian distributions, a novel self-adaptive dynamic mutation strategy, based on a uniform distribution is considered. The proposed approach consists of two components; the first component describes the ANN “internal” architecture and depends on a total number of hidden layers and an average number of neurons on hidden layers. This relationship is determined by the Fermi-Dirac-like function. The second component changes its value depending on the fitness of a chromosome, exposed to mutation. Including phenotype and genotype information about the problem the proposed strategy adjusts the mutation strength to a given ANN architecture and a mutated chromosome simultaneously.

Keywords: Artificial Neural Networks, Evolutionary Algorithms, Evolutionary Programming, Mutation

I. INTRODUCTION

Evolutionary Programming (EP) represents a methodology of Evolutionary Algorithms (EA) in which mutation is considered as a main reproduction operator [1-5]. The mutation-based approaches have been successfully applied not only to combinatorial optimization problems but also to other areas of Artificial Intelligence (AI), where they are used as subsidiary evolutionary approaches for some parameters optimization, e.g. for ANNs connection weights training [6-13].

The classical strategy used in EP is Gaussian mutation operator, denoted as the classical EP (CEP), based on a standard normal distribution [1, 5-9]. The main disadvantage of this approach is its slow convergence to a near-optimal

solution. To overcome this drawback research has been devoted towards the development of new mutation strategies. Yao and Liu [15, 16] have proposed an alternative mutation technique, called the Fast Evolutionary Programming (FEP), which is based on the Cauchy distribution function. Their comparative study of both mutation-based algorithms showed that solving high-dimensional optimization problems, the FEP converges quicker to minima than the CEP. Furthermore, Yao *et al.* [17] have developed an improved strategy of FEP, called the improved Fast Evolutionary Programming (IFEP).

This paper addresses feed-forward ANNs connection weights training with the EP approaches. We present a comparative study of three known mutation approaches (CEP, FEP and IFEP), based on Gaussian and Cauchy distributions and a new self-adaptive dynamic mutation strategy (denoted as the network-weight (NW)-based mutation strategy), which is based on a uniform distribution. In comparison to established EP mutation operators, which include the genotype information (number of connections between neurons) a proposed approach contains not only genotype, but also phenotype information. Phenotype information is included in the value, called the *network weight*, which depends on the number of hidden layers and the average number of neurons on hidden layers, and adapts the mutation strength to a given ANN architecture. The network weight quantity is defined by the Fermi-Dirac-like function. Genotype information is encapsulated in the mutation’s dynamic part and is represented by the fitness value of every individual; using fitness as a dynamic component we achieve a particular change of the mutation strength, proportional to the chromosome’s mean square error.

We provided a comparative study of all considered mutation-based algorithms in order to determine their performance for ANNs with varied number of hidden layers and neurons on them, i.e. to indicate the speed of their convergence to optima, investigate their ability to find solutions with the high precision and quantify the average rate of successful mutations per generation (the rate of chromosomes, improved after mutation). All experiments were carried out using the same initial conditions for all mutation strategies, e.g. size of a population, ANN topology, and absence of other genetic operators using the XOR simple function to evaluate the performance of the mutation strategies.

K. Davoian and W.-M. Lippe are with the Department of Computer Science, University of Münster, Einsteinstr. 62, 48149 Münster, Germany (phone: +49-2518333796, fax: +49-2518333755, email: {kristina.davoian, lippe}@uni-muenster.de).

A. Reichel is with the Institute of Theoretical Physics, University of Jena, Max-Wien-Platz 1, 07743 Jena, Germany (phone: +49-1746906492, email: a.reichel@uni-jena.de).

This paper is organized as follows: section II describes the mutation approaches, including the network-weight based strategy (Section II-D). Section III presents and analyzes the experimental results. Section IV concludes this paper.

II. MUTATION STRATEGIES

A. The Classical Evolutionary Programming (CEP)

The evolution of connection weights with the self-adaptive Gaussian mutation [5-9] is implemented as follows:

1. Generate an initial population of M chromosomes randomly. Each chromosome is represented by a pair of real-valued vectors (U_i, S_i) , $\forall i \in \{1, \dots, M\}$, where $U_i = (u_1, u_2, \dots, u_n)$ is the set of n connection weights of the network and $S_i = (s_1, s_2, \dots, s_n)$ is the variance vector, also known as a self-adaptive parameter.
2. Create offspring (U'_i, S'_i) by applying a Gaussian random variable:

$$s'_j(i) = s_j(i) \exp(\tau' N(0, 1) + \tau N_i(0, 1)) \quad (1)$$

$$u'_j(i) = u_j(i) + s_j(i) N_i(0, 1) \quad (2)$$

where $N_i(0, 1)$ is a normally distributed random value with mean 0 and variance 1 and is generated anew for each value of i . $N(0, 1)$ is a similar random number, but is generated once for every population member. The parameters τ and τ' are commonly set to:

$$\tau = \left(\sqrt{2\sqrt{n}} \right)^{-1}$$

$$\tau' = \left(\sqrt{2n} \right)^{-1}$$

3. Calculate the fitness of each chromosome of parental and offspring population, based on the training error.
4. Compare pairwise all individuals of parental (U_i, S_i) and offspring (U'_i, S'_i) populations and generate the new population from the best ones.
5. Repeat the process until some halting criteria are satisfied.

B. The Fast Evolutionary Programming (FEP)

The algorithms based on the Gaussian mutation have one significant disadvantage, i.e. they slow convergence to a near-optimal solution. To overcome this drawback the mutation approach based on Cauchy distribution has been proposed [15, 16].

The one-dimensional Cauchy probability density function is defined by the following equation:

$$f(x) = \frac{\gamma}{\pi(\gamma^2 + x^2)}, \quad -\infty < x < \infty \quad (3)$$

where $\gamma > 0$ is a scale parameter. The corresponding Cauchy distribution function is

$$F(x) = \frac{1}{\pi} \arctan\left(\frac{x}{\gamma}\right) + \frac{1}{2} \quad (4)$$

The implementation of mutation strategy based on Cauchy distribution function is the same as the self-adaptive Gaussian mutation approach (described in section II-A) except for the equation (2) which is replaced by the following:

$$u'_j(i) = u_j(i) + s_j(i) \delta_i, \quad (5)$$

where δ_i is a Cauchy random variable, which is generated anew for each value of i . The relationship for changing s_i is the same as in the case of Gaussian mutation (given by equation (1)), but with $N_i(0, 1)$ replaced by the Cauchy, instead of Gaussian random variables. The scale parameter γ is generally taken to be 1.

C. The Improved Fast Evolutionary Programming (IFEP)

The IFEP strategy combines the described above CEP and FEP mutation approaches [17]. The main idea of IFEP is to form two offspring chromosomes from the same parent, applying CEP and FEP mutation operators to the first and second offspring's creation, respectively. After comparing their fitness an individual with the smaller error is chosen for the next generation.

D. The network-weight based mutation strategy

The described above mutation approaches based on Gaussian and Cauchy distribution functions contained genotype information of considered problem, i.e. the total number of connection weights. In comparison to these approaches we present a self-adaptive dynamic mutation strategy which contains not only genotype, but also phenotype information. The phenotype information is incorporated in a value, called the *network weight*, which adapts the mutation operator to a given ANN architecture. The network weight value depends on the number of hidden layers and the average number of neurons on hidden layers and thus describes the ANN's "internal" structure. The second part of proposed mutation strategy is determined by the fitness function, which is defined by the individual's mean square error. This involves a dynamic component in the mutation operator, which changes its value for a particular chromosome exposed to mutation, and thus adjusts the efficiency of reproduction to a particular chromosome. The described mutation approach is based on a uniform distribution.

The ANN's training with the network-weight based mutation strategy is implemented as follows:

1. Create an initial population consists of M randomly generated chromosomes. Each chromosome $S_i = (s_1, s_2, \dots, s_m)$, $\forall i \in \{1, \dots, M\}$, represents one possible set of connection weights, where m is a total number of connections between neurons and $s_j \in [-1.0, 1.0]$, $j \in \{1, \dots, m\}$.

2. Select a chromosome for reproduction using the tournament selection method [18], which selects two parental chromosomes and compares their fitness; the individual with the smaller error reaches next step of evolution.
3. Create new individuals by the following equation:

$$s'_j = s_j \left(1.0 + N_w(l, n) N_{dyn} N_{rand} \right) \quad (6)$$

where s_j is a gene randomly chosen out of a chromosome S_i exposed to mutation; N_{dyn} – the fitness of S_i , which is represented by the mean square error between the expected and the actual outputs over all examples of a considered task, and N_{rand} is a uniformly distributed random value in the interval $[-1.0; 1.0]$. Network weight $N_w(l, n)$ is defined by the function, which depends on the number of hidden layers l and the average number of neurons on hidden layers n . For each ANN the quantity N_w is calculated only once and does not change its value during the evolution, as we consider the evolution of connection weights in the environment determined by an ANN architecture. Extensive empirical studies provided in order to establish their dependency on the ANN architecture showed that the relationship of hidden layers and the average number of neurons on hidden layers is defined by the Fermi-Dirac-like function and is calculated according to the following formula:

$$N_w(l, n) = A_1 + \frac{l}{2} + \frac{B_1 - \frac{l}{2}}{1 + \exp\left(\frac{n - \mu}{T_1}\right)} \quad (7)$$

The value μ has the same physical sense as a chemical potential in thermodynamics [19] and depends on the number of hidden layers:

$$\mu = A_2 + \frac{B_2}{1 + \exp\left(\frac{l - B_2}{T_2}\right)} \quad (8)$$

The constants $A_1 = 3.0$, $B_1 = 2.0$, $T_1 = 0.4$, $A_2 = 1.2$, $B_2 = 3.2$, $T_2 = 0.6$ were obtained in the “trial and error” way.

As it is obvious from the equation (7), the behavior of its fractional part is changed with the increment of hidden layers: up to 4 layers it becomes positive values, when the number of hidden layers is 4 it becomes zero (the exceptional case), i.e. the network weight is independent of the average number of neurons and has optimal value 5.0, and with more than 4 layers the fractional part of the expression (7) becomes negative. Fig. 1 presents the network weight values for considered ANN architectures with different numbers of hidden layers and neurons on them. It shows that the network weight has always positive values, in spite of the described above features of the equation (7), which are conditioned by the adjustment of the optimal mutation step size to a particular ANN architecture.

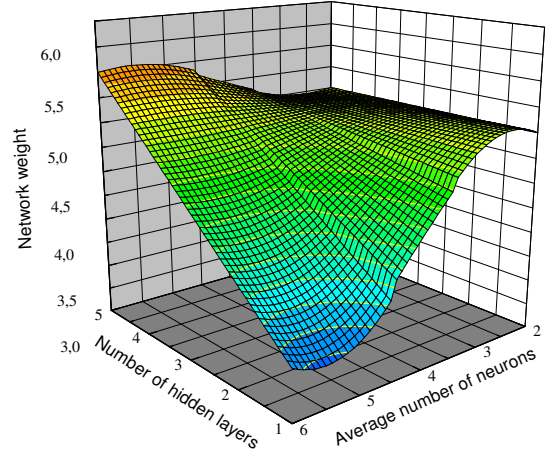


Fig. 1 The network weight value

The dynamic component (N_{dyn}) is based on genotype information. Use of a mean square error as a dynamic component, which is different for every mutated chromosome, enables to control a randomly generated value and to adjust the mutation strength to an individual, i.e. the higher the error of chromosome is the higher will be the strength of mutation. A dynamic component achieves a particular improvement of the chromosomes, depending on their fitness: after mutation individuals with relatively good fitness will slightly be changed (as a results of mutation the fitness of a chromosome can be improved or declined), while genes in the chromosomes with high error will be changed considerably.

4. Compare the fitness of parental and offspring chromosomes. The individual with the higher fitness reaches the offspring population.
5. Repeat the process until some halting criteria are satisfied.

The described mutation strategy adjusts the mutation strength according to the characteristics of a predefined ANN topology (determined at the start of simulation) and to a mean square error of particular chromosome simultaneously and ensures the significant improvement of the population, thus enables a rapid convergence of the algorithm to an optimal solution. Another advantage of the NW-based mutation approach is that it is independent of the ANN’s “external” architecture – the number of input and output neurons, determined by a considered task, since the network weight value is related with the “internal” structure of ANN.

III. EXPERIMENTS AND ANALYSIS

For the experiments we have considered the different ANN architectures using the XOR logical function as a case application (we considered a simple task in order to investigate the features of the NW-based strategy). To make a fair comparison, all mutation-based algorithms started their

evolution with the same initial conditions (size of a population, ANN structure, absence of other genetic operators). The tests were carried out for ANNs with 1-5 hidden layers and 2-6 numbers of neurons on each hidden layer, i.e. the simplest ANN had 1 hidden layer with 2 neurons and the most complex ANN had 5 hidden layers with 6 neurons on each layer. The initial population of chromosomes was generated randomly and consisted of 50 individuals. For each considered ANN 1000 runs of the algorithms have been made (for the NW-based mutation approach the network weight values were calculated, according to the equation (7)). The stopping criterion was the precision of the best individual's mean square error equal to $1.0e-3.0$.

The first set of experiments was provided to determine the speed of algorithms' convergence to optimal solutions. The goal of these experiments was to quantify the average number of iterations and time, necessary to find an optimal set of connection weights. In Fig. 2 the comparative average results of four algorithms are presented. Table 1 presents the average time in *ms*, needed to find the optimal solutions.

The obtained results show that the proposed self-adaptive dynamic mutation enables rapid convergence to optimal solution. At the same time the advantage of phenotype information presence is obvious: the function, describes the behavior of NW-based mutation for different ANN architectures is monotonic, which means that for varied number of hidden layers with the same number of neurons on them we have practically an invariable number of iterations, needed for the algorithm's convergence. This specific feature is achieved by the network weight value. The CEP, FEP and IFEP mutation strategies have small differences in the iteration numbers at which they found the optimal set of connection weights; this is conditioned by the relationship between Gaussian and Cauchy distributions.

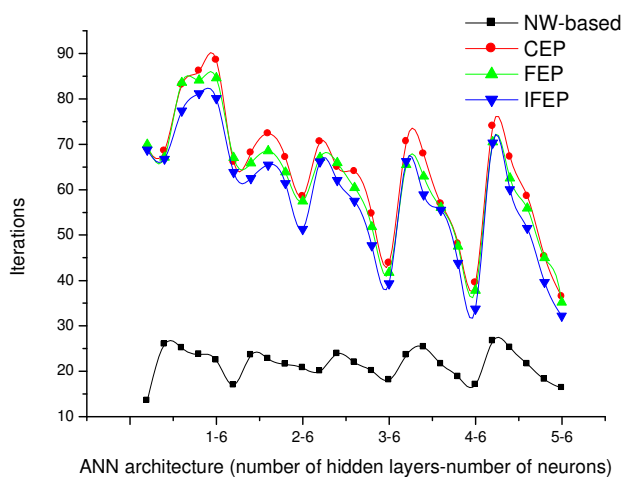


Fig. 2 Average number of iterations needed to find an optimal set of connection weights for four mutation-based algorithms

| ANN architecture | | Average time, ms | | | |
|------------------|-------------------|------------------|------|------|-------------|
| Hidden layers | Neurons (average) | CEP | FEP | IFEP | NW-mutation |
| 1 | 2 | 8.8 | 9.0 | 8.8 | 1.7 |
| 1 | 3 | 12.6 | 12.4 | 12.3 | 4.8 |
| 1 | 4 | 19.4 | 19.5 | 18.1 | 5.9 |
| 1 | 5 | 29.1 | 28.4 | 27.4 | 8.0 |
| 1 | 6 | 40.9 | 39.1 | 37.0 | 10.4 |
| 2 | 2 | 10.3 | 10.4 | 10.0 | 2.6 |
| 2 | 3 | 17.8 | 17.3 | 16.4 | 6.2 |
| 2 | 4 | 24.7 | 23.4 | 22.4 | 7.8 |
| 2 | 5 | 34.3 | 32.6 | 31.4 | 11.0 |
| 2 | 6 | 39.1 | 38.4 | 34.3 | 13.8 |
| 3 | 2 | 13.5 | 12.8 | 12.7 | 3.8 |
| 3 | 3 | 22.0 | 22.3 | 21.0 | 8.1 |
| 3 | 4 | 28.2 | 26.6 | 25.4 | 9.7 |
| 3 | 5 | 36.0 | 34.1 | 31.4 | 13.2 |
| 3 | 6 | 40.6 | 38.7 | 36.5 | 16.7 |
| 4 | 2 | 15.0 | 13.9 | 14.1 | 5.0 |
| 4 | 3 | 27.2 | 25.2 | 23.6 | 10.1 |
| 4 | 4 | 30.8 | 30.3 | 30.0 | 11.7 |
| 4 | 5 | 40.6 | 40.1 | 37.1 | 15.8 |
| 4 | 6 | 45.7 | 43.7 | 39.0 | 19.7 |
| 5 | 2 | 18.5 | 17.6 | 17.5 | 6.6 |
| 5 | 3 | 33.1 | 30.8 | 29.6 | 12.4 |
| 5 | 4 | 36.8 | 35.1 | 32.4 | 13.5 |
| 5 | 5 | 45.7 | 45.4 | 40.0 | 18.3 |
| 5 | 6 | 50.9 | 49.1 | 45.0 | 22.8 |

Table 1 Average time, needed to algorithms convergence

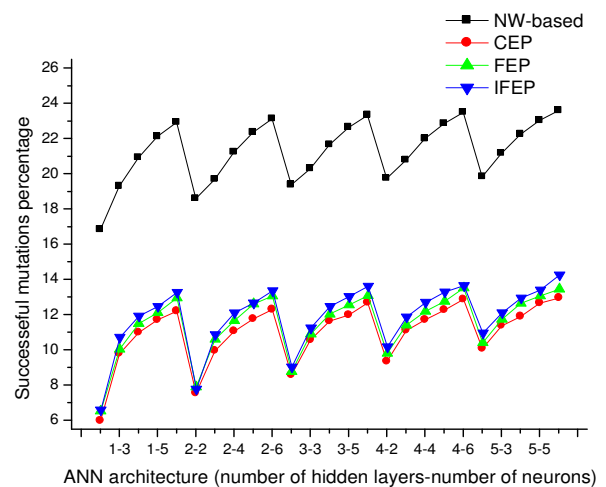


Fig. 3 Average rate of successful mutated chromosomes

The second set of experiments was carried out to quantify the average percentage of successful mutations (percent of mutated chromosomes, which reached the offspring population) for each of compared algorithms. Fig. 3 presents the average results, obtained after 1000 runs of algorithms for predefined ANN architecture. The results show, that CEP, FEP and IFEP approaches have approximately the same rate of successful mutations (6-14%); with the NW-based strategy 17-24% of all chromosomes, undergo mutation improve their values at each stage of evolution. This efficiency of NW-based algorithm is achieved by the incorporated dynamic component, which is equal to the mean

square error and changes its value during the run-time. In this case the change of every mutated gene does not depend strongly on a randomly generated number; the dynamic component adjusts it to a chromosome, exposed to mutation, in other words adapts the mutation strength to a particular chromosome.

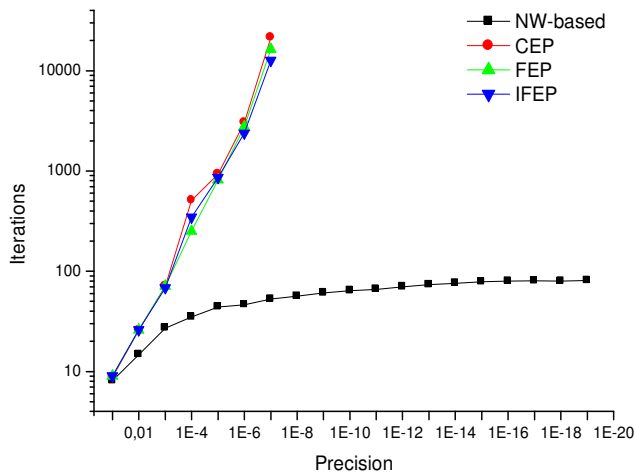


Fig. 4 Algorithms ability to find solutions with the high precision

In the described experiments above the algorithms stopped, when the mean square error of the best chromosome in the population achieved the predefined precision equal to $1.0e-3.0$ (the algorithm found a set of connection weights could solve the problem with the precision $1.0e-3.0$). To study the algorithms' ability to find solutions with the high precision (and thus to investigate their resistance to convergence to a local instead of a global optimum) we provided a set of runs with the stopping condition equal to machine precision. Fig. 4 presents an example of four algorithms' convergence for the ANN with 1 hidden layer and 3 neurons on it. It shows that the NW-based algorithm is likely to be resistant to a premature convergence to local minima, while the CEP, FEP and IFEP mutations are unable to find a solution with a machine precision

IV. CONCLUSIONS

In this paper we have presented a comparative analysis of different mutation-based algorithms for ANN connection weights training. Besides well-known CEP, FEP and IFEP approaches, based on Gaussian and Cauchy distributions, a proposed novel self-adaptive mutation strategy, based on a uniform distribution, has been considered. The goal of the provided experimental study was to investigate the behavior of mutation-based algorithms for feed-forward ANNs with varied numbers of hidden layers and neurons on them: determine the average improvement on each stage of evolution (establish the rate of successful mutations), evaluate the average numbers of iterations and time needed to find optimal sets of connection weights, and examine the ability of compared strategies to find solutions with the high

precision. The results showed that CEP, FEP and IFEP approaches have insignificant distinctions, which is conditioned by the similarity of Gaussian and Cauchy distributions, while proposed network-weight based mutation strategy (see Section II-D), which includes both phenotype and genotype information increases the rate of successful mutations, which leads to rapid convergence of algorithm and enables to find solutions with the high precision (Fig. 3, 4, Table 1). This significant acceleration is achieved by two components, incorporated in described mutation approach: the self-adaptive control parameter network weight (Fig. 1), which depends on a number of hidden layers and an average number of neurons on hidden layers (this relationship is determined by the Fermi-Dirac-like function – see equation (7)), and thus includes the information about ANN's "internal" structure, and dynamic, which changes its value depending on the mean square error of a chromosome, exposed to mutation. The proposed strategy encapsulates all necessary information about the problem and adjusts the mutation strength to a given ANN architecture and to a particular chromosome simultaneously, which leads to a rapid convergence to optima. The results indicated also the benefit of phenotype information inclusion: the speed of the NW-based algorithm's convergence is independent of the hidden layers number, i.e. ANN complexity (Fig. 2); this feature is reached by means of an adaptation to the environment, achieved by the network weight value.

In our future work we will utilize the proposed NW-based strategy for evolving the connection weights and architectures simultaneously, as well as apply it to the complex problems in order to establish its generalization ability.

ACKNOWLEDGMENT

The authors would like to thank Angelos Molfetas from University of Western Sydney for his helpful comments.

REFERENCES

- [1] X. Yao, "Evolutionary artificial neural networks", in *Encyclopedia of Computer Science and Technology*, Vol. 33, New York: Marcel Dekker, pp. 137–170, 1995
- [2] X. Yao, "Evolving Artificial Neural Networks", in *Proc. of the IEEE*, 87 (9), pp. 1423-1447, 1999
- [3] D. B. Fogel, "Evolving Neural Networks: Selected Medical Applications and the Effects of Variation Operators", *Modeling and Simulation: Theory and Practice – A Memorial Volume for Professor Walter J. Karplus*, Kluwer Academic Press, Boston, MA, pp. 217-248, 2003
- [4] D. G. Landavazo and G. B. Fogel, "Evolved Neural Networks for Quantitative Structure-Activity Relationships of Anti-HIV Compounds", in *Proc. of the IEEE Congress on Evolutionary Computation*, Vol. 1, Honolulu, HI, USA, pp. 199-204, 2002
- [5] A. Abraham, "Meta-Learning Evolutionary Artificial Neural Networks", *Neurocomputing Journal*, Elsevier Science, Netherlands, Vol. 56c, pp. 1-38, 2004
- [6] A. E. Eiben, R. Hinterding, Z. Michalewicz, "Parameter Control in Evolutionary Algorithms", *IEEE Trans. on Evolutionary Computation*, Vol. 3, pp. 124-141, 2000
- [7] R. Hinterding, "Gaussian mutation and self-adaption for numeric genetic algorithms", in *Proc. of the Second IEEE Conference on Evolutionary Computation*, pp. 384-389, 1995

- [8] A. Jain, D. Fogel. "Case studies in applying fitness distributions in evolutionary algorithms: I. Simple neural networks and Gaussian mutation", *Applications and Science of Computational Intelligence III, Proc. SPIE*, Vol. 4055, pp. 168-175, 2000
- [9] P. A. Castillo, J. J. Merelo, V. Rivas, G. Romero, and A. Prieto, "Evolving Multilayer Perceptrons", *Neural Processing Letters* 12(2), pp.115-127, 2000
- [10] J. Branke, "Evolutionary approaches to dynamic optimization problems – a survey", *GECCO Workshop on Evolutionary Algorithms for Dynamic Optimization Problems*, pp. 134-137, 1999
- [11] D. B. Fogel and K. Chellapilla, "Revisiting evolutionary programming", in *SPIE AeroSense'98, Applications and Science of Computational Intelligence*, Orlando, FL, pp. 2-11, 1998
- [12] W.-M. Lippe, "Soft-Computing mit Neuronalen Netzen, Fuzzy-Logic und Evolutionären Algorithmen", *Springer-Verlag*, Berlin Heidelberg, 2006
- [13] H. Abbass and R. Sarker, "Simultaneous evolution of architectures and connection weights in anns", in *Artificial Neural Networks and Expert Systems Conference*, Dunedin, New Zealand, pp. 16-21, 2001
- [14] Lock and C. Giraud-Carrier. "Evolutionary Programming of Near-Optimal Neural Networks", in *Proc. of the Fourth International Conference on Artificial Neural Networks and Genetic Algorithms (ICANNGA99)*, Springer-Verlag, pp. 302-306, 1999
- [15] X. Yao and Y. Liu, "Fast Evolutionary Programming", in *Proc. of the Fifth Annual Conference on Evolutionary Programming (EP'96)*, the MIT Press, San Diego, CA, USA, 29/2-2/3/96. pp. 451-460, 1996
- [16] X. Yao and Y. Liu, "Fast evolution strategies," *Control and Cybernetics*, 26(3), pp. 467-496, 1997
- [17] X. Yao, Y. Liu, and G. Lin, "Evolutionary programming made faster", *IEEE Transactions on Evolutionary Computation*, pp. 82-102, 1999
- [18] D. B. Fogel, "A Comparison of Evolutionary Programming and Genetic Algorithms on Selected Constrained Optimization Problems", *Simulation*, pp. 397-404, 1995
- [19] W. Greiner, L. Neise, H. Stöcker, "Thermodynamics and statistical mechanics", *Springer-Verlag*, New York [u.a.] 2000