Multiobjective Optimization by Nessy Algorithm

Mario Köppen[1], Stephan Rudlof[1]

[1]Fraunhofer-Institute for Production Systems and Design Technology
Pascalstr. 8-9, 10587 Berlin, Germany
email: {mario.koeppen|stephan.rudlof}@ipk.fhg.de

Abstract. This paper presents the extension of the Neural Evolutional Strategy System (Nessy) to the multiobjective optimization case. The neural architecture of the Nessy algorithm is extended by using more than one output neuron, one neuron for each objective. The learning law of Nessy is modified according to the presence of multiple measures of performance. Each hidden neuron of the generation layer randomly selects an objective for one cycle of the network. From this, the multiobjective ranking of the population (or neurons of the solutions layer) is stochastically approximated. The modified Nessy algorithm (Monessy) is able to search for the Pareto set of a multiobjective optimization problem. A test function from literature with well-known Pareto and trade-off set is examined. The newly proposed algorithm effectively searches for the Pareto set by switching between explorational and exploitational search phases. This was compared with random search, which did not hit the Pareto set as nearly as often as the Monessy algorithm. Also, the replacement of a weighted-sum matching measure with multiple matching measures in a framework for texture filter design is considered as a second example.

1 Introduction

In real world problems, one is often faced with the problem of multiple, possibly competing, measures of performance, which should be optimized simultaneously. Conventional optimization techniques, such as steepest descent or simulated annealing, are difficult to extend to the true multiobjective case. However, evolutionary algorithms had been pointed out to be possibly well-suited to multiobjective optimization since early in their development [1]. This is based on the ability of evolutionary algorithms to search for multiple solutions in parallel and to handle complicated tasks such as discontinuities, multimodality and noisy function evaluations.

Recently, the Nessy algorithm has been proposed [2,3]. Nessy (Neural Evolutional Strategy System) is a neat integration of an evolutionary algorithm into a neural network architecture. The neural architecture is based on a multilayer backpropagation neural network. The state of every neuron equals a chromosome. One iteration cycle of the Nessy network equals one generation of the underlying evolutionary algorithm. The three layers of Nessy are: input layer, hidden layer and output layer. Input and hidden layer and hidden and output layer are fully connected, weights are assigned to these connections. What Nessy makes different from Neuro-GA approaches are its redefined genetic operators. To ensure the neuron based evaluation mode of a neural network, genetic operators are modified in a suitable manner. The essential steps of the Nessy algorithm are: selection, computation of error, weights modification, transduction and mutation.

There is evidence for the Nessy algorithm to maintain memory in the evolutionary algorithm in a unique manner [3]. Besides of keeping search space "knowledge" in the population's schemata, the weights of Nessy can be employed as a memory, changing at a lower time scale than the population's schemata.

However, the current implementation of the Nessy algorithm uses only one neuron in the output layer. This is due to the fact, that there is only one value for the fitness function, from which the error of the selection operation is calculated. By using more than one output neuron, the Nessy algorithm is well-suited to multi-objective optimization in the context of multiple, possibly competing, fitness functions.

There is one essential difference between single objective optimization and multiobjective optimization. This difference is based on the fact, that there is no natural sort order of points of the n-dimensional Euclidian space, if $n \geq 2$. Hence, a solution of an optimization problem can not be directly compared with some other. There is a special subspace structure of optimal solutions in multiobjective optimization, referred to as the Pareto set. A solution is Pareto-optimal, if this solution is not dominated by some other solution, i.e. if no change in the optimization problems' domain variables gains increase in all fitness values at once. The set of all Pareto-optimal solutions is the Pareto set (or Pareto-front). The task of multiobjective optimization is generally considered as the search for the Pareto-front.

Few evolutionary algorithms for Pareto-optimization have been designed and proposed so far, starting with the proposal of the VEGA system [4] (see [1] for a comprehensive overview and [5] for a new approach). The essential problem here is that of multiobjectively ranking the individuals of a population. In [6], a study is given on different approaches to multiobjective ranking. The main problem is the sorting of individuals, which are on the same multiobjective ranking level. Nessy offers a surprisingly simple approach to this complicated task, bypassing the ranking problem, because it can be characterized as an individual evolutionary algorithm [7] (fitness values of individuals in the input layer are never directly competed against

each other based on their fitness values - hence, there is no need for ranking). For doing so, Pareto-optimal is considered as a probabilistic feature. Instead of saying, that the modification of one domain variable gains no increase for all fitness function simultaneously, it is considered, that there will be no fitness increase for a randomly chosen fitness function value out of the set of all fitness function values. By this means, the weights update and transduction decision are restricted to only one of the error measures of the output layer neurons. Which one, is randomly assigned to every hidden neuron for every network cycle.

This paper is organized as follows. In section 2, the Nessy algorithm is recalled. The definition of Pareto set and the extension of the Nessy algorithm to the multiobjective case (Monessy) are given in section 3. Section 4 studies a well-known multiobjective optimization, using the Monessy algorithm, section 5 demonstrates, how the proposed Monessy algorithm can be employed for minimizing multiple performance measures simultaneously within the Lucifer (Lookup-Compositional Inference) framework for the automated design of 2D-Lookup texture filters. The paper ends with the conclusions.

2 The Nessy Algorithm

The Nessy algorithm [2,3] is a neat integration of an evolutionary algorithm into a multilayer backpropagation neural network. The Nessy architecture is shown in figure 1. In this section, the Nessy algorithm will be recalled for the single objective case. Its extension to the multiobjective case will be given in the next section.

Nessy is composed of three layers: a solution (or input) layer, a generation (or hidden) layer and an output layer. The state of every neuron of the solution layer is a chromosome. Altogether, the neurons of the solution layer are the counterparts of the individuals of a population in conventional genetic algorithms (GA). The generation layer is of the size of the number of genes of a chromosome, as will be explained below. Every neuron of the generation layer points to a neuron of the solution layer, i.e. it refers to its state. The functionality of the generation layer is the counterpart to the selection operation in conventional GA. The output layer has only one neuron with state 0 (representing a lower bound of the fitness values of a single objective optimization problem).

Solution layer and generation layer are fully connected, every connection has a weight assigned to it. These weights are considered as relative selection probabilities. The symbol p_{ij} stands for the weight of the connection from the generation layer neuron i to the solution layer neuron j. Generation layer neurons and the only output neuron are also connected, but no special weights are assigned.

Once initialized with random chromosomes and weights, Nessy works autonomously. The outline of the Nessy algorithm is as follows:

1. Each generation layer neuron *i* randomly chooses exactly one solution layer neuron *j* with the probability:

$$\frac{p_{ij}}{\sum_k p_{ik}}.$$

2. The backpropagated error of every generation layer neuron is simply the normalized fitness value of its state, i.e.

$$h_i = \frac{f_i}{\sum_k f_k}.$$

The training value Zero is the assumed limit of the optimization.

3. The weights of the solution-generation layer connections are updated according to the learning law:

$$p_{ij}^{new} = p_{ij}^{old} - \alpha \frac{f_j - g_i}{O},$$

where α is the learning rate, g_i is the fitness value of the state of generation layer neuron *i* and *O* the state of the output neuron. The new weights are restricted to the range [0,1].

4. The neuron states of the solution layer are modified by the transduction operator. For transduction, the modification of every solution is performed fitness-proportionately. Each solution compares its fitness with the fitness of generation layer neuron state *i*. If the second one is lower than the solutions' fitness, the solution takes gene *i* of the generation layer neuron state. This operator was introduced in [9] as implantation operation, later [10] it was referred to as transduction operator and deduced from bacterial genetics. Transduction is a directed and asymmetric genetic operator.

5. The solutions are mutated. Mutation is necessary if the transduction operator is applied. The whole population would tend to premature convergence, because parts of successful chromosomes are copied over unsuccessful ones. The mutation is performed by adding a Gaussian distributed random number to the actual gene. Two parameters control the mutation: mutation probability p_μ and the standard deviation of the Gaussian distribution σ_μ.

Altogether, Nessy uses one structural parameter (size of solution layer) and three learning parameters (learning rate α, mutation probability p_μ and mutation standard deviation σ_μ).

Essential insights into the backgrounds of this algorithm were given in [3]. On its base, there is the interplay of weights and the so-called diagonal prototype, which is the chromosome constructed from the diagonal of the chromosomes of the hidden layer. Please note, that the number of hidden neurons is equal to the number of genes of a chromosome. The weights are used as a memory for the genetic search, which stochastically approximates the fitness ranking of the population, while the diagonal prototype acts like an independent "explorer" of the search space. Hence, the Nessy algorithm maintains the exploration/exploitation phases of a GA in a unique manner, which results into faster optimization and better avoidance of local extrema.

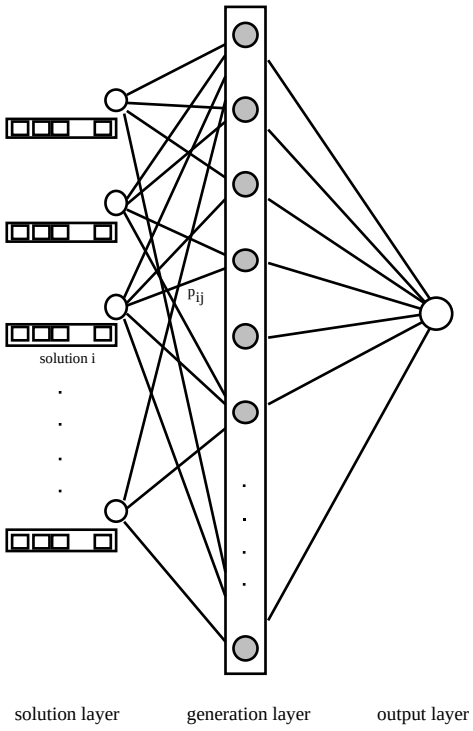


Figure 1. Nessy architecture

3 Multiobjective Nessy Algorithm (Monessy)

As pointed out in the introduction, multiobjective optimization is generally considered as search for the Pareto set. If a multiobjective optimization problem is given in terms of n fitness functions of m independent domain variables x_1, x_2, …,

x_m, or a vector $\mathbf{x}$ in the m-dimensional Euclidian space R^n, then a solution $\mathbf{x}_u$ is said to be Pareto-optimal, if it is not dominated by any other $\mathbf{x}_v$ from R^m, i.e. there is no $\mathbf{x}_v$ from R^m such that

$$\forall_{i \in \{1,\ldots,n\}} (f_i(\vec{x}_v) \leq f_i(\vec{x}_u)) \quad \wedge \quad \exists_{i \in \{1,\ldots,n\}} (f_i(\vec{x}_v) < f_i(\vec{x}_u)) .$$

Often, there is more than one Pareto-optima. The set of all such Pareto-optima comprises the Pareto-front or Pareto set. All function values of the Pareto set are referred to as trade-off set. In practical applications, this notion is only a first step towards the solution of a multiobjective optimization problem. Additional selection criteria have to be defined in order to select one solution out of all solutions of the Pareto set.

The use of multiple fitness functions in an evolutionary algorithm has been intensively discussed in [1]. The basic concept is fitness sharing. The population is divided into n subsets, each of which uses exactly one out of all n fitness functions. This is basically the famous VEGA approach, as proposed in [4]. However, this approach depends on the scaling of the objectives. Fourman [11] further examined the case and proposed a version, wherein the objectives for each comparison are chosen randomly. This approach was reported to work surprisingly well. In [1], this was explained with the stochastic approximation of the ranking of the individuals, which is performed by this modification.

Starting from these experiences, it was decided to use the ranking approximation for the extension of the Nessy algorithm to the multiobjective case, too.

Three modifications of the Nessy algorithm, which was presented in the last section, were introduced:

1. There are n output neurons, each neuron for one objective. Now, in step 2 of the algorithm, each output neuron determines the error value for its objective independently.

2. Each hidden neuron randomly selects an objective for every network cycle.

3. Weights update (step 3) and transduction operation (step 4) are performed due to the objective chosen by each hidden neuron.

From this, the Nessy algorithm is easily extended to the Monessy algorithm, which is able to process multiple fitness functions.

4 A Case Study for the Monessy Algorithm

In order to learn about the modified algorithm, a test function from literature [1] with a well-known Pareto set was examined. The simple test function is

$$f_1(x,y) = 1 - \mathbf{exp}[-(x-1)^2 - (y+1)^2]$$
$$f_2(x,y) = 1 - \mathbf{exp}[-(x+1)^2 - (y-1)^2]$$

Both functions, if considered independently, have a minimum function value of 0 at (1,-1) and (-1,1) for (x,y), respectively. For the Monessy run, both functions are scaled with an exponential γ, i.e. $\mathbf{f}_1(x,y)^\gamma$ and $\mathbf{f}_2(x,y)^\gamma$ were considered for simultaneous minimization. Figure 2 shows the trade-off sets for different values of γ.

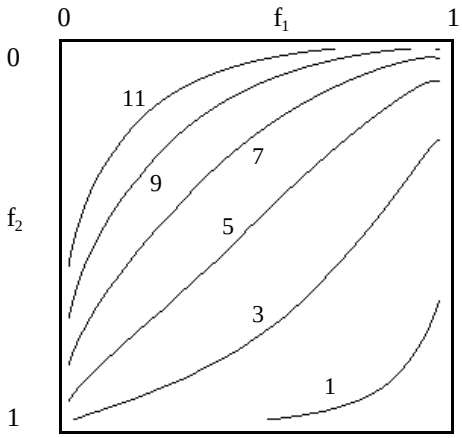


Figure 2. Trade-off sets for different γ-values

Four experiments were performed, two using the Monessy algorithm and two using "blind" random search. Reals from -4 to 4 were encoded into a bitstring of size 44 by rescaling the binary numbers represented by the bitstrings onto this intervall. The Monessy network was composed of 30 solution neurons, 44 generation neurons and two output neurons. The learning rate was 3.0, the mutation used a (0, 0.5)-normal distribution. The network run 1000 cycles. The value for γ was chosen as 9, for which the trade-off set is nearly circular.

In the first experiment, the (x,y)-positions examined by the network were plotted to see how the Monessy algorithm approximates the trade-off set. In the second experiment, the same was done for 30000 random numbers (i.e. the same number as test points of the Monessy network during 1000 cycles). In the third experiment, it was counted, how often the Pareto-front was hit during the run. As reference set, the interior part of the Pareto-set for the γ value 9 was used. Finally, in the fourth experiment, the same was done for random numbers.

The results for the first experiment can be seen in figure 3. The Monessy algorithm approximates the trade-off set. Moreover, the region below the trade-off set is nearly empty. The Monessy algorithmus allocates most of its trials nearby the two single objective optima (either f_1 is 0 or f_2 is 0), or nearby the Pareto set. This should be compared with the result of the second experiment, as shown in figure 4. For x and y, random values from [-4,4] were chosen. The figure shows the distribution of the resulting function values in the f_1,f_2-plane.

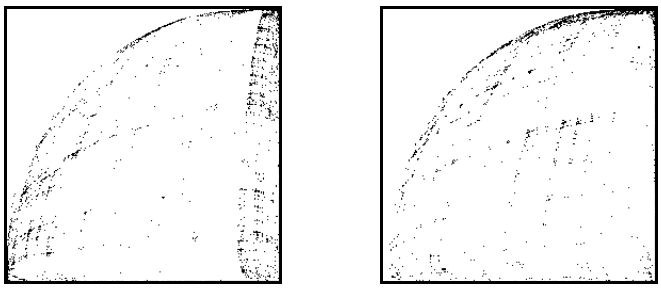


Figure 3. Trade-off set approximated by Monessy algorithm

What can not be seen from the figures 3 and 4 is, how often the Pareto front was reached by the Monessy algorithm and how this could be compared to randomly "guessing." Experiments 3 and 4 were carried out in order to reveal this. Figure 5 shows the number of solution layer neurons for cycle 1 to 1000, which were nearby the Pareto set. For reference, the plot of figure 2 was used with a γ value of 9. From figure 5, it can be seen, that the Monessy algorithms switches between explorational and exploitational phases. During exploration, the evolving population covers the fitness space in a manner similar to a standard genetic algorithm (SGA). Once a good initial region is found, the algorithm switches to the exploitational phase. For a SGA, this switch is performed due to the schemata theorem. If a SGA started exploitation, this process can't be stopped, because the number of trials allocated to exploitation exponentially grows in relation to the number of trials allocated to exploration (the number of schemata decreases to one). This also holds for the (Mo)nessy algorithm, but the algorithm may not converge at this point. Monessy

quickly reaches the Pareto-front, but, after a while, the stability of the population breaks down and the individuals "fall back" to the simple single objective optimization stage. Then, the exploration phase restarts, and, after a while, the Pareto-front is revisited. During 1000 cycles, the Pareto set was hit 1657 times by the 30 neurons of the Monessy algorithm. The number of hits on the Pareto-front for experiment 4, the random search, was 80 in 30000 in average! Hence, the Monessy algorithm is able to effectively search for the Pareto-front of this multiobjective optimization problem.

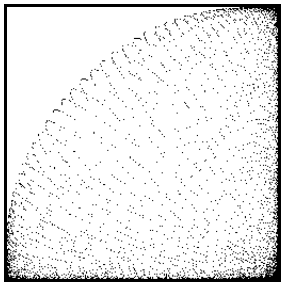


Figure 4. Trade-off set randomly approximated

The result of the experiments are surprisingly well. The Monessy algorithm is able to approach the Pareto-front. From this, it can be used as a tool for the needed refinement of the optimization goal in multiobjective optimization problems. After the Pareto-front is revealed, its subregions can be qualified according to its relevance for the (true) real optimization goal. Also, stopping criterias for a second run of the Monessy algorithm can be designed.

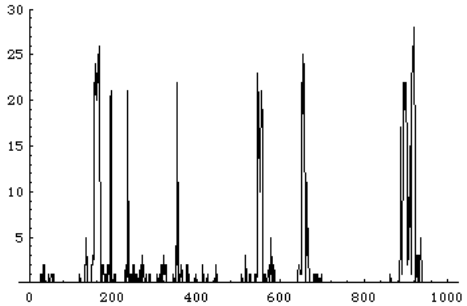


Figure 5. Number of solution layer neurons within the Pareto-set vs. cycle number

5 Texture filtering example

The Monessy algorithm was applied to the Lucifer framework [8]. This framework generates texture filters by employing the 2D-Lookup algorithm. For 2D-Lookup, a twodimensional matrix is given as lookup table for pairs of grayvalues. Two grayvalued images are scanned at every position for obtaining such a pair of grayvalues. The entry of the matrix at that position is a new grayvalue, which is the entry at that position in the result image. In the Lucifer framework, there are only entries for black (grayvalue 0) and white (grayvalue 255) in the matrix. Hence, the result of the algorithm is a binary image. The two input images of the algorithm are the results of the application of two image processing operations onto the original image. The optimization task here is a twofold one: the image processing operations have to be chosen out of a set of predefined image processing operators (e.g. convolution operators, morphological operators, rescaling operations, differences-of-gaussians, texturenumbers, statistical operators, together 256 operators), and the 2D-Lookup matrix has to be specified.

original image with low-contrats texture fault goal image

Figure 6. Images given by the user for the Lucifer framework

For using evolutionary search for this, a fitness function has to be formulated. In the Lucifer framework, this is done by means of a so-called goal image, which is given by the user. Consider figure 6, where the original image and the goal image are shown. The recognition task is to design a filter for extracting the surface faults. The binary image, which is the result of the application of the 2D-Lookup algorithm for some setting of the operators and the matrix, has to be compared with the binary goal image. This is the point where multiobjective optimization comes into the play. From the users point of view, goal and result image are similar, if most of the reference points of the goal image are also black in the result image. To obtain a measure for this, three quantities are considered: the amount of white pixels of the result image, which are also white in the reference image (**whiteok**), the amount of black pixels of the result image, which are also black in the goal image (**blackok**) and the amount of black reference pixels, which are also black in the result image (**blackrefok**). Please note, that **blackrefok** and **blackok** are not identical.

For the Lucifer system, the fitness function was a weighted sum of these three quantities. For optimization, the initial Nessy algorithm was used. For multiobjective optimization, these three measures (**whiteok**, **blackok**, **blackrefok**) can not be used directly. If so, the completely black (fitness (0,1,1)) and completely white images (fitness (1,0,0)) would belong to the Pareto-set! Instead of this, the measures were combined in two ways. In the first version, the fitness vector was given as (2-(**blackrefok**+**whiteok**), |**blackrefok**-**whiteok**|), in the second version in a similar manner as (2-(**blackok**+**whiteok**), |**blackok**-**whiteok**|). The Pareto-fronts were scanned for 300 network cycles. The results are given in figure 7. As can be seen there, the two fitness functions behaves different. In the first version, both measures can simultaneously reach the value (0,0). In the second version, the first quantity can not go beyond a value of about 0.5. For the first version, it has to be considered, that all result images with at least one black pixel in the reference and all other pixels white has a fitness vector of (0,0)! Hence, the Pareto set contains all framework settings leading to such a situation, and nothing more. In the second version, the demands for zero measures can not be fulfilled. It is impossible to generate a result image with all black pixels correctly set. Here, the trade-off set is the vertical line with (blackok+whiteok) about 0.5. This can be used to design a stopping criterion for further runs of the Monessy algorithm.

Figure 8 shows one result of the run and the two operation images. Altogether, the Monessy algorithm could be used for framework adaptation as well as the Nessy algorithm. However, the Monessy algorithm gains more insight into the way, the best possible solution can be yielded within this framework, if the plots of the trade-off sets are considered.

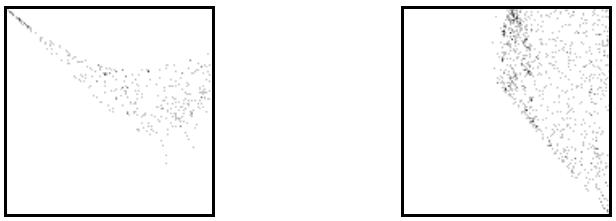
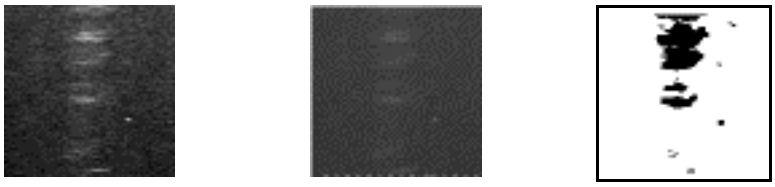


Figure 7. Trade-off sets approximated by Monessy algorithm for the problem given with figure 6, first and second version (see text for details)

Also, this example has demonstrated, that care has to be taken into account, if multiple quality functions are designed for multiobjective optimization. Often, the Pareto set contains solutions which do not comprise solutions of the real problem.

This also shows, that much more research is necessary in order to become "familiar" with the complicated task of Pareto-optimization.



operation 1 operation 2 result image

Figure 8. Operation images and result image for a run of Monessy, using version 2 for the fitness vector

6 Conclusions

In this paper, the Monessy algorithm for multiobjective optimization was proposed. It is an extension of the Neural Evolutional Strategy System (Nessy) by using more than one output neuron, one neuron for each objective. The multiobjective ranking of the individuals of the population (or the neurons of the solution layer) is stochastically approximated, if each hidden neuron of the generation layer chooses one objective in every cycle at random. All internal decisions of the algorithm (transduction decision, weights modification), which were based on one fitness measure, are now based on the randomly selected fitness measure. By this means, the Monessy algorithm is able to search for the Pareto set of a multiobjective optimization problem. The search can be divided into explorational and exploitational phases. Exploitational phases end up at the Pareto-front, but the stability of the population breaks down after a while and the explorational phase restarts. Therefrom, the Monessy algorithm often revisits the Pareto set during a long run. If the trade-off sets (for the biobjective case) are plotted, insight can be gained for the refined formulation of the optimization problem. Also, stopping conditions can be derived. The Monessy algorithm was examined on a test function from literature. The effective search for the Pareto-front of the Monessy algorithm was demonstrated and compared with the low performance of random search. Also, the replacement of weighted-sum approaches with multiple objectives was considered, employing a real-world application from the field of texture filtering. It comes out, that such replacements should be performed with caution. Often, the Pareto set contains unwanted solutions. The objectives have to be designed in a manner, which lowers the performance of such unwanted solutions.

Reference

1. Forensica C M, Fleming P J 1995 An Overview of Evolutionary Algorithms in Multiobjective Optimization. *Evolutionary Computation*, **3**(1):1-16

2. Köppen M, Teunis M, Nickolay B 1997 A Neural Network that uses Evolutionary Learning. *Proc. of 1997 IEEE International Conference on Evolutionary Computation, ICEC'97, Indianapolis, IN, April*, pp 635-639

3. Köppen M, Teunis M, Nickolay B 1997 Nessy - an Evolutionary Learning Neural Network. *Proc. of the 2nd International ICSC Symposium on Soft Computing, SOCO'97, Nîmes, France, September*, pp 243-248

4. Schaffer J D 1985 Multiple Objective Optimization with Vector Evaluated Genetic Algorithms. In: Grefenstette J J (ed) 1985 *Genetic Algorithms and Their Applications: Proc. of the First International Conference on Genetic Algorithms*, Lawrence Erlbaum, pp 93-100

5. Clergue M, Collard P, Gaspar A 1997 DGA and Pareto elitism: Improving Pareto optimization. *Proc. of the 2nd International ICSC Symposium on Soft Computing, SOCO'97, Nîmes, France, September*, pp 315-321

6. Forensica C M, Fleming P J 1993 Genetic Algorithms for Multiobjective Optimization: Formulation, Discussion and Generalization. *Proc. of the 5th International Conference on Genetic Algorithms, Illinois at Urbana Champaign, NM, July*, pp 416-423

7. Zhao Q 1996 Efficient Learning of NN-MLP based on Individual Evolutionary Algorithm. *Neurocomputing* 13 2-4:201-215

8. Köppen M, Teunis M, Nickolay B 1997 A Framework for the Evolutionary Design of 2D-Lookup Filters. Technical Report ME-MK-97-03, Department for Pattern Recognition, FhG IPK Berlin, Berlin, Germany

9. Furuhashi T, Nakaoka K, Uchikawa Y 1994 A New Approach to Genetic Based Machine Learning and an Efficient Finding of Fuzzy Rules. *Proc. WWW'94, Nagoya, Japan*, pp 114-122

10. Hashiyama T, Furuhashi T, Uchikawa Y 1995 Design of Fuzzy Controllers for Semi-Active Suspension generated through the Genetic Algorithm. *Proc. ANNES'95, Dunedin, New Zealand*, pp 166-169

11. Fourman M P 1985 Compaction of Symbolic Layout Using Genetic Algorithms. In: Grefenstette J J (ed) 1985 *Genetic Algorithms and Their Applications: Proc. of the First International Conference on Genetic Algorithms*, Lawrence Erlbaum, pp 141-153