Individual-based management of meta-models for evolutionary optimization with applications to three-dimensional blade optimization

Lars Gräning, Yaochu Jin, Bernhard Sendhoff

2007

Preprint:

This is an accepted article published in Evolutionary Computation in Dynamic and Uncertain Environments. The final authenticated version is available online at: https://doi.org/[DOI not available]

10

Individual-based Management of Meta-models for Evolutionary Optimization with Application to Three-Dimensional Blade Optimization

Lars Gräning, Yaochu Jin, and Bernhard Sendhoff

Honda Research Institute Europe
Carl-Legien-Str. 30, 63073 Offenbach am Main, Germany
{lars.graening,yaochu.jin,bernhard.sendhoff}@honda-ri.de

Summary. To reduce the number of expensive fitness function evaluations in evolutionary optimization, individual-based and generation-based strategies for meta-model management (evolution control) have been proposed. In this work, four individual-based frameworks for meta-model management are investigated. A feed-forward neural network is employed to construct an approximation model of the fitness function. Structure optimization of the neural network is used to reduce the approximation error. In an attempt to adapt the number of controlled individuals, adaptation mechanisms are suggested based on the model error, selection error, rank correlation, and fitness correlation. Preliminary results indicated that the adaptation mechanisms do not work well as expected.

Two of the frameworks are implemented in 3D blade design optimization. The results showed that individual-based meta-model management is promising, though further efforts are still needed to improve the performance of the evolutionary algorithms with meta-models for fitness estimation.

10.1 Introduction

It has been shown that evolutionary algorithms are very powerful in solving many real-world optimization tasks such as 3D turbine blade aerodynamic design optimization of a jet engine [5, 13, 14], of micro heat exchanger [2] or transonic wing design [17]. The advantage of evolutionary algorithms is that they stochastically search the fitness landscape for the optimal solution without the need of any gradient information. However, this advantage is at the cost of a large number of fitness evaluations. In 3D blade optimization, one evaluation of the fitness will take huge computational time because computational fluid dynamics (CFD) simulations have to be performed to evaluate the performance of the blade.

To reduce the number of fitness evaluations, one idea is to estimate the fitness using computationally efficient meta-models, see [8] for an overview of existing methods. One problem to deal with in real-world optimization problems is that it is difficult to acquire enough data so that the meta-model can sufficiently approximate the original fitness landscape, which could result in false convergence [11, 15]. Therefore it is not advisable to use meta-models only as a surrogate for the original fitness function. To avoid false convergence, the neural network model should be used in conjunction with the original fitness function. This is termed evolution control or model management [11, 15]. If evolution control is used, new data become available during optimization, which can then be used for on-line update of the meta-model. Meta-models can also be employed in the local search embedded in evolutionary optimization [19].

In this work, four individual-based evolution control methods are compared on three benchmark problems. The two most promising methods are adopted for 3D blade design optimization. In an attempt to improve the performance of the methods, adaptation mechanisms are suggested to adjust the impact of the meta-model on the optimization process during optimization.

10.2 Evolutionary Optimization with Neural Network Based Fitness Estimation

10.2.1 Evolutionary Optimization

The evolution strategy with covariance matrix adaptation (ES-CMA) [4] is adopted for blade optimization in this work. No recombination has been used since negative influence has been observed in blade optimization. The mutation operator adds normally distributed random values to the design parameters of the individual in order to search the design space. Adaptation of the parameters of the normal distribution in each generation plays an essential role for the performance of the search algorithm. ES-CMA uses a derandomized self-adaptation mechanism where the whole covariance matrix is adapted to adjust the parameters of the normal mutation distribution. These parameters, called strategy parameters, are also encoded in the chromosome of the individuals.

One major problem in evolutionary design optimization process is the high cost of computation resources for evaluating the quality of the designs. For example, a 3D design optimization run takes upto 3 months for 200 generations of evolution on high performance computers. In this work, we employ neural networks as the meta-model to partially substitute the computationally expensive fitness evaluations.

10.2.2 Artificial Neural Networks

Up to now, polynomials, kriging model, radial-basis-function networks, and multi-layer perceptrons (MLP) have been used as meta-models in evolutionary optimization [8, 16]. We decided to use MLPs in this work because it has been shown that MLPs are very powerful in function approximation and classification. The neural network adapts its parameters and structure to learn the functional mapping between the design parameters and the performance with the help of a number of training data obtained from previous optimization. After the network is trained, it can be used to predict the fitness of new designs, given the design parameters.

Multi-layer Perceptros (MLPs)

In [1, 6] it is shown that one hidden layer is sufficient to approximate any continuous functions, provided that a sufficient number of hidden layer neurons is used. The number of hidden neurons depends strongly on the characteristics of the target function [20]. Mostly, the characteristics of the function is unknown. In this case, it is suggested that structure optimization of the MLPs should be considered [20].

In this work, the algorithm introduced by Hüsken et al [7] is employed. The architecture of the neural network is encoded in a connection matrix and a weight matrix. The values in the connection matrix determine which nodes in the network are connected and the values in the weight matrix determine the strength of connections. Fig. 10.1 illustrates the mapping between a neural network and the connection and weight matrices.

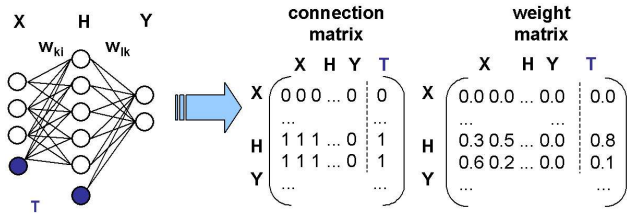

Fig. 10.1. Architecture of a neural network and the corresponding matrix representation

The output of the MLP can be calculated using the following equations:

$$y_l = \sum_{k=1}^{N_h} w_{lk} \cdot g(\sum_{i=1}^{N_x} w_{ki} x_i + t_h) + t_y, \tag{10.1}$$

where N_x is the number of input neurons and N_h is the number of hidden neurons. In the neural network the following activation function $g(z)$ is used, whose characteristic is similar to the sigmoidal function.

$$g(z) = \frac{z}{1 + |z|}. \tag{10.2}$$

The weights of the neural network are trained online during the optimization when new training samples are available. For neural network training, Rprop [18], an improved version of the back propagation algorithm is used. The main difference between Rprop and the back propagation algorithm is that the learning rate adjustments and weight changes do not depend on the magnitudes of the gradient, rather on the signs of the gradient terms.

Structure Optimization of Neural Networks

To improve the approximation quality of the neural network, one way is to optimize the architecture of the neural network during optimization. Yao [21] provided a comprehensive review of the optimization of neural networks using evolutionary algorithms. In this work, a genetic algorithm has been used for this purpose. The connections a_i and the value of the weights w_i of the neural network are encoded into the genotype of an individual. This means that each individual encodes a neural network with a different architecture and different weights. To generate offspring representing different neural networks, specific mutation methods are used. The mutation methods allow to insert or delete a single connection or neuron and the weights are mutated by adding a normally distributed random number. After mutation, the Lamarckian mechanism is used for lifetime learning of the weights. Finally, the weights are coded back into the individuals. EP-tournament-selection is used to select the individuals representing the neural networks with the lowest mean square error with respect to the training data.

10.3 Individual-Based Evolution Control Methods

It is found that if a meta-model such as a neural network is used to estimate the fitness of the individuals, the evolutionary algorithm probably will converge to a false optimum [15], which is not one of the original fitness function. In these cases, it is essential that the model be used in conjunction with the original fitness function. How often the model should be used instead of the original fitness evaluation is the task of *evolution control* or *model management* methods.

As illustrated in Fig. 10.2, evolution control methods can be divided into two basic approaches, namely individual-based and generation-based [11].

In the generation-based approach, one has to decide generation by generation for all individuals whether the fitness will be determined using the meta-model or using the original fitness function. If the individual-based approach is used, one has to decide for each individual in every generation whether the meta-model or the time-consuming fitness function should be used. In this

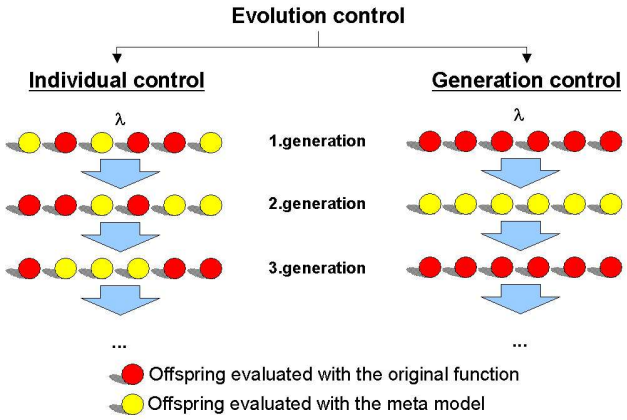

Fig. 10.2. Principle of individual-based and generation-based evolution control methods

work we concentrate on the individual-based approaches and will introduce several methods in detail.

As can be seen in Fig. 10.3, the individual-based evolution control method can be described by a common evolutionary optimization process. In each iteration, λ' offspring are generated out of the μ parents by mutation. After that, the individual-based control method decides which λ^* offspring are evaluated by the real fitness function. The results are used to train the neural network before the fitness of the remaining λ'$-\lambda^*$ offspring will be estimated by the neural network. In the end μ parents will be selected out of the λ individuals according to their fitness.

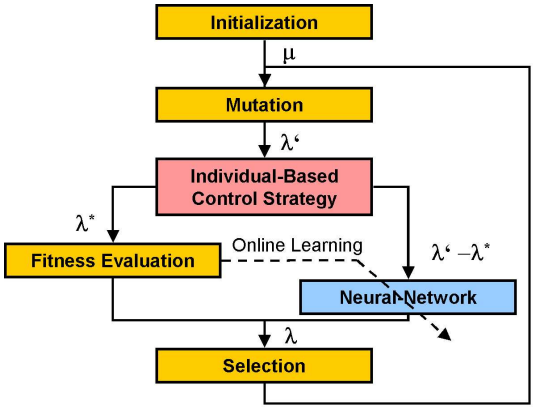

Fig. 10.3. Evolutionary optimization process including individual-based evolution control methods

10.3.1 Best Selection (BS)

In the best selection strategy [15], all λ' offspring are pre-evaluated by the neural network at first to find the most promising (best) λ^* individuals. These λ^* individuals are evaluated by the original fitness function and the results are used for training. After training the neural network, the remaining λ'$-\lambda^*$ individuals are again evaluated by the neural network to get a better estimation. At the end of each generation, the μ best individuals out of all $\lambda = \lambda$' individuals become parents for the next generation.

10.3.2 Pre-Selection (PreS)

A pre-selection has been introduced in [19], in which the Gaussian processes are used as meta-model instead of neural networks. The idea is as follows. λ'$> \lambda$ offspring are generated by mutation, and the neural network is used to estimate the fitness of these offspring. The λ^* most promising individuals are pre-selected out of the λ' offspring. Like in the best selection strategy, the λ^* individuals are evaluated using the original fitness function. The main difference to the best selection strategy is that the μ parents are selected only out of the λ^* individuals, which are all evaluated with the original fitness function.

10.3.3 Clustering Technique (CT)

In [12] a different approach is described to find out which individuals have to be evaluated with the original fitness function. Using the k-means clustering technique, all λ' individuals of a generation are grouped into λ^* clusters. Now the λ^* individuals closest to the cluster center are evaluated using the original fitness function. The results of the fitness evaluations, as in all other methods, are used to train the neural network during the optimization. The fitness of the remaining λ'$-\lambda^*$ individuals is estimated using the neural network. Last but not least the μ parents are selected out of all $\lambda = \lambda$' individuals.

10.3.4 Clustering Technique with Best Strategy (CTBS)

The idea of the clustering technique with best strategy is the same as in clustering technique. The offspring are also grouped into a number of clusters. Now the neural network is used to predict the fitness of each offspring. Instead of evaluating the individuals closest to the cluster center, the λ^* individuals with the best predicted fitness of each cluster will be evaluated by the original fitness function.

10.3.5 Simulation Results

The main reason for using individual-based evolution control is to reduce computational costs. In real-world optimization problems like the blade optimization, the calculation of the fitness needs a large amount computational time. It is impossible to test all the algorithms on the real-world design optimization problem. So all methods are tested first on three widely used benchmark functions. The test functions are the Sphere, Rosenbrock and Ackley functions. To get an impression of how the functions looks like, the equation and the two dimensional plotting of the functions are illustrated in Fig. 10.4, where n is the dimension of the test functions. In the following simulations, the dimension is set to 10. Part of the following results has been reported in [3].

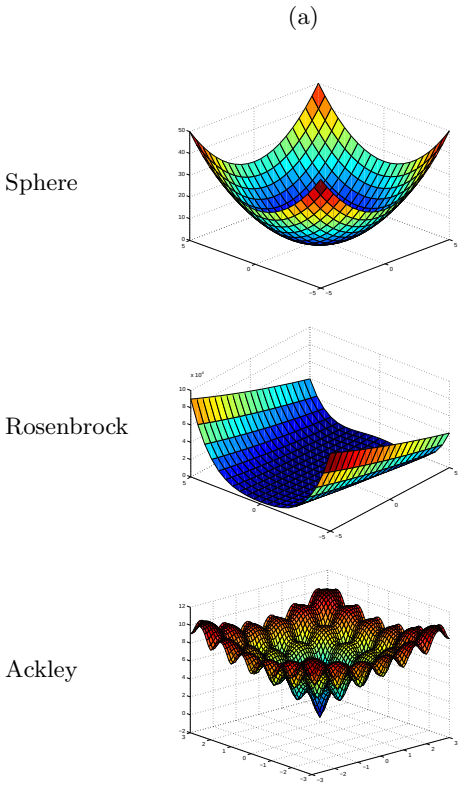

Fig. 10.4. Overview of the used test-functions with an (a) 2D illustration and (b) the equation of the functions

Simulation Setup

In all simulations, a (μ, λ) ES-CMA without recombination is adopted. The strategy parameters of the covariance matrix are randomly initialized between $\sigma_{min} = 0.05$ and $\sigma_{max} = 4$.

We compare the model management frameworks in both serial and parallel computing environments. The parameters for the evolutionary optimization process are set according to the different requirements of the used computational environment.

When the optimization is conducted in a serial environment, the performance depends only on the number of fitness evaluations needed to reach a near optimum. So in each generation μ parents are selected out of the same amount of λ offspring. The remaining parameters are adjusted according to the values of μ and λ as combined in Table 10.1. For clarity the following notation is used: $\big(\mu, [\lambda']\lambda[\lambda^*]\big)$. μ parents are always selected out of λ offspring. λ' defines the number of pre-selected and λ^* the number of controlled individuals. The ratio according to [4] is set to $\mu \leq \frac{\lambda}{3}$. The ratio of λ to λ' and λ^* are based on recommendations or findings in [12] and [19].

PlainES	PreS	BS	CT	CTBS
$(3, [12]12[12])$	$(3, [24]12[12])$	$(3, [12]12[6])$		

Table 10.1. Settings of the strategy parameters $\big(\mu, [\lambda']\lambda[\lambda^*]\big)$ to compare the performance of the individual-based control methods for optimization in a serial computational environment

If it can be assumed that in a parallel computational environment enough computers are available to evaluate all individuals in parallel, the number of fitness evaluations itself is less important. In that case the number of generations needed to reach a near-optimal solution is the main concern. We also assume that the number of used machines equals the number of fitness evaluations λ^*, which is held constant for all methods. The remaining parameters are adjusted with respect to λ^*. The entire setup is listed in Table 10.2.

PlainES	PreS	BS	CT	CTBS
$(2, [6]6[6])$	$(2, [12]6[6])$	$(2, [12]12[6])$		

Table 10.2. Settings of the parameters μ, λ' and λ^* to compare the performance of the individual-based control methods for optimization in a parallel computational environment

The neural network used in the simulations consists of 10 input nodes, one hidden layer with four hidden neurons, and one output node. If structure optimization is carried out, the number of hidden neurons is not fixed. To

achieve a good local approximation of the original fitness landscape, only data of the most recent evaluations are used for training.

Serial Optimization

In a serial computational environment, only the number of expensive fitness evaluations is of importance to reach a near-optimum. 20 independent runs are performed for each optimization to reduce the randomness. The median fitness value of the best offspring in each generation is plotted versus the number of fitness evaluations to compare the performance of the introduced methods. The results from the Sphere, Rosenbrock and Ackley functions are presented in Fig. 10.5 and Fig. 10.6. The left column presents the results with and the right column without structure optimization of the neural network.

To show the statistical significance between the evolution control methods and the plain evolution strategy, the boxplot of the results are given in Fig. 10.6. The boxplot illustrates the median and the variance of the fitness values of the best individual in the final generation over 20 runs. The notches of the boxes in the plot are the graphical equivalence to the student t-test. If the notches of two boxes do not overlap, there is a significant difference between the medians of the two strategies at a significance level of $p = 0.05$.

From Fig. 10.5 and Fig. 10.6, we can see that all evolution control methods except the best strategy improve the performance of the plain evolution strategy significantly on the 10D Sphere function. But there are no statistically significant differences between the model-assisted strategies themselves, no matter whether structure optimization of the neural networks are performed or not.

It turns out that the clustering technique with best strategy outperforms other algorithms on the 10D Rosenbrock function, when no structure optimization of the neural network is carried out. However, all algorithms fail to improve the performance of the plain evolution strategy significantly. This result may be attributed to the fact that the number of hidden neurons is not sufficiently large to approximate the Rosenbrock function. Meanwhile, the result indicates that with structure optimization, the neural networks perform locally very well on the Rosenbrock function.

From Fig. 10.5, it can be seen that the individual-based evolution control methods perform well on the Ackley function. But as we can see in the boxplots in Fig. 10.6 the variance of the strategies is very high except the pre-selection strategy. The pre-selection strategy outperforms the plain evolution strategy in almost all of the 20 runs.

In summary, it turned out that the pre-selection method shows the most stable and promising results and only fails to improve the evolution strategy on the Rosenbrock function. The reason for the stability of the pre-selection methods might be that the parents of the next generation are only selected from the individuals that evaluated with the original fitness function.

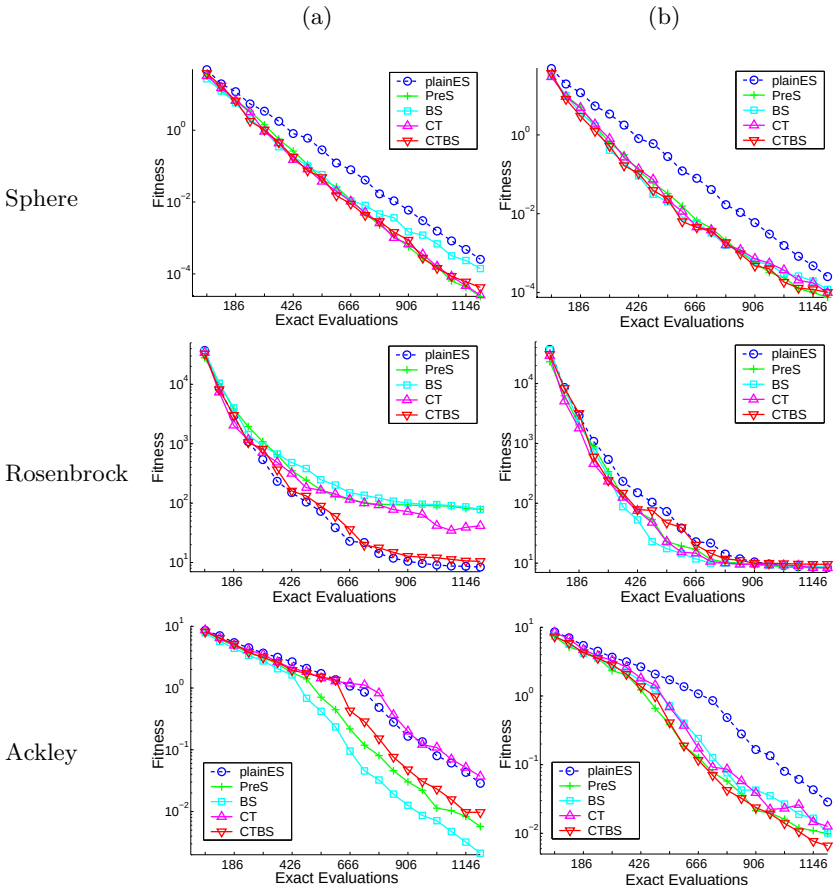

Fig. 10.5. Performance comparison of the individual-based methods in a serial computational environment: (a) without structure optimization and (b) with structure optimization of the neural network

In the following, we only show the results with structure optimization of the neural networks because the above results show that using structure optimization mostly improves the performance of the neural network and the optimization process.

Parallel Optimization

As mentioned before, to compare the performance of the individual-based evolution control methods in a parallel computational environment, the number of exact fitness evaluations is not as important as the number of generations. Therefore the fitness values are plotted versus the number of generations. To make sure that the comparison is fair, the number of real fitness evaluations

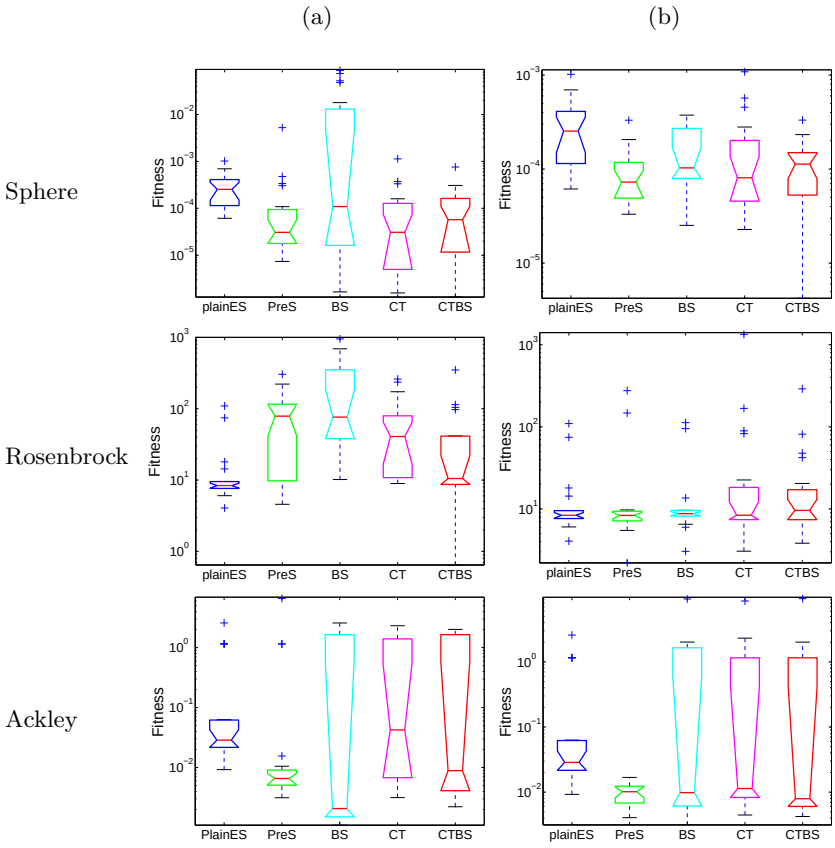

Fig. 10.6. Boxplot of the best fitness in the final generation over 20 runs after 1200 exact fitness evaluations are done: (a) without structure optimization, and (b) with structure optimization

each generation in all methods is the same. Fig. 10.7(a) shows the characteristics of the median best fitness value over the generations and in Fig. 10.7(b) the boxplot for statistical analysis is illustrated using structure optimization of the neural network.

As can be seen in Fig. 10.7, the pre-selection strategy improves the plain evolution strategy on all used test functions. So it might improve the optimization process better if the parents are only selected out of the λ^* offspring evaluated with the original fitness function. Using BS, CT or CTBS, the parents are selected out of all λ individuals, whose fitness has either been evaluated with the real fitness function or estimated by the neural network. The possible reason is that if some individuals are selected according to the estimated fitness, the optimization algorithm might be misled, which degrades the performance.

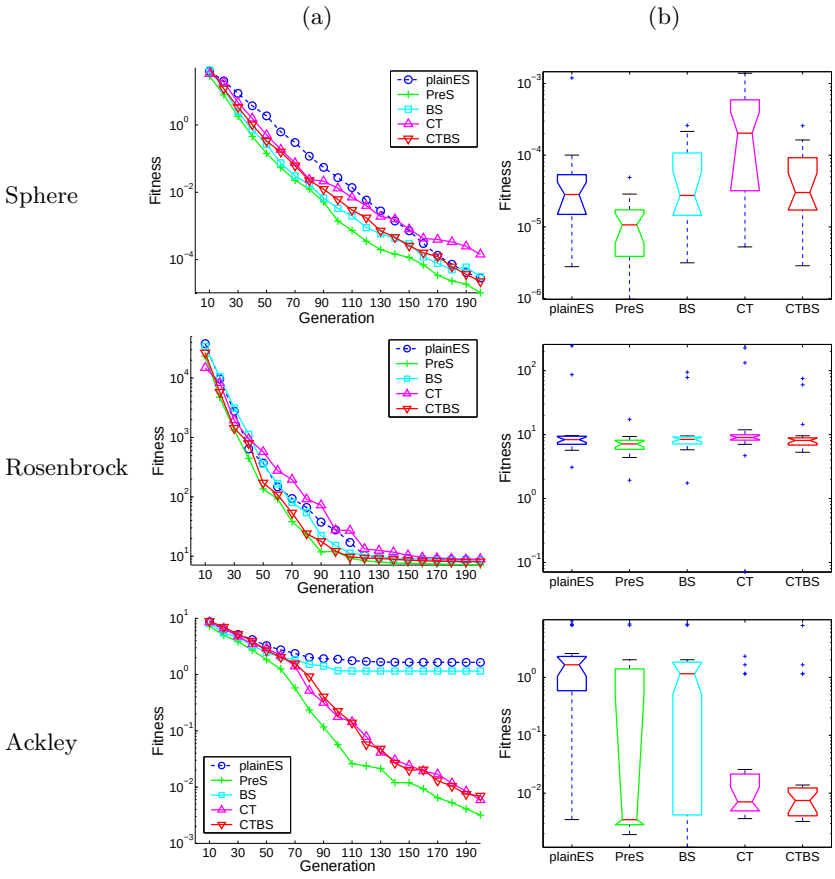

Fig. 10.7. Results of the individual-based methods in a parallel computational environment: (a) performance comparison and (b) boxplot over 20 runs after 200 generations

We can see that clustering the design space on bumpy fitness landscapes like the Ackley function gives a benefit to the algorithm. The clustering technique with best strategy performs also well on all other test functions, especially if the fitness function becomes more complex. However, it can not outperform the pre-selection strategy.

In both serial and parallel computational environments, the pre-selection and clustering technique with best strategy delivered the most promising results.

10.4 Adaptation in Individual-based Evolution Control

In the above comparisons, the number of controlled individuals is fixed during optimization. In this section, we consider adapting the number of controlled individuals. Three different adaptation frameworks are introduced. The idea of adaptation is that if the performance of the neural network increases during optimization, it should be used more often to substitute the original fitness function. There are two parameters that can be taken into account for adaptation. One is the number of fitness evaluations λ^*, and the other is the number of pre-selected individuals λ'. Adjustment of λ^* only makes sense of on-line scheduling of computational resources is possible.

10.4.1 Normalized Squared Error Driven Adaptation Mechanism (NERD)

The first quality measure we considered here to adapt the number of individuals is the approximation quality of the neural network. The approximation quality of the neural network can often be measured using the squared error between the individual's original fitness $\phi_i^{(Orig.)}$ and the estimated fitness of the neural network $\phi_i^{(MLP)}$:

$$E_i^{(SE)} = \left(\phi_i^{(MLP)} - \phi_i^{(Orig.)}\right)^2. \tag{10.3}$$

The error can be determined for each offspring i which has been evaluated with the original fitness function. The main idea of this adaptation method is that if the error of the neural network becomes smaller in the next generation $t+1$, the neural network should be used more often. But the error in the generation $t+1$ is unknown. It is only possible to compare the error in the current generation t with the error in the last generation $t-1$. There are two problems in comparing these two errors. In general, the fitness values by itself decline during optimization and so probably the value of the squared error will also decline. Therefore the squared error should be normalized by the use of the mean squared error:

$$E_i^{(NSE)} = \frac{E_i^{(SE)}}{\frac{1}{\lambda^*}\sum_{i=1}^{\lambda^*} E_i^{(SE)}}. \tag{10.4}$$

Comparing the mean of the normalized squared errors of all offspring will lead to the second problem. If there is one individual with a large error while the error of the rest of the individuals is small the comparison might be misleading. Instead of comparing the mean values, the ranks of the individuals' normalized errors are compared. An example how the rank of the error in generation $t-1$ and generation t can be calculated is illustrated in Fig. 10.8. First the values of the two sets of errors are combined and sorted. After that, each element is given its corresponding rank. If some samples carry the

same error value, the rank will be averaged. Then, the ranks of the sets from generation $t-1$ and generation t are accumulated.

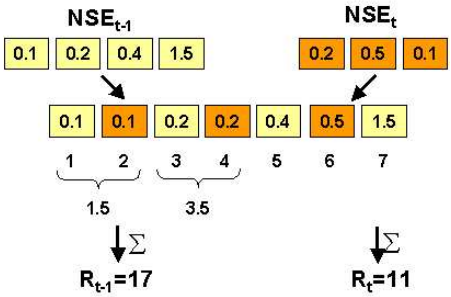

Fig. 10.8. Example of evaluating the rank with respect to the normalized square error in generation $t-1$ and generation t

Given the two ranks the quality measure $\rho^{(NERD)}$ can be determined by calculating the difference of the two normalized ranks. The ranks also have to be normalized by the number of λ^* individuals because λ^* might change if it is adapted during optimization:

$$\rho^{(NERD)} = \frac{R(t-1)}{\lambda^*_{t-1}} - \frac{R(t)}{\lambda^*_t}. \tag{10.5}$$

If R_t is smaller than R_{t-1}, the quality measure $\rho^{(NERD)}$ is less than 0 and the neural network should be used more often. On the other hand, if R_{t-1} is smaller than R_t, $\rho^{(NERD)}$ is bigger than 0 and the neural network should be used less often. λ^*_{t+1}, or rather λ'_{t+1} is adapted as follows:

$$\lambda^*_{t+1} = \lambda^*_t - \rho^{(NERD)} \cdot \Delta\lambda, \tag{10.6}$$

$$\lambda'_{t+1} = \lambda'_t + \rho^{(NERD)} \cdot \Delta\lambda. \tag{10.7}$$

The remaining difficulty is the choice of the correct free parameter $\Delta\lambda$, which might be problem-specific.

10.4.2 Selection Based Adaptation Mechanism (Sel)

From the evolutionary computation point of view, only the correct selection is of importance and not the approximation error of the model. The error of the neural network does not have direct influence on the evolutionary selection process. Therefore, as introduced in [15], a selection based quality measure can be considered to evaluate the quality of the model based selection process. For each correctly selected individual, based on the estimation of the model, it is

given a rank of $(\lambda_t^* - i)$, if the individual has the i-th best fitness based on the true fitness. To calculate the rank, as shown in Fig. 10.9, we first pick out the μ^* best individuals based on the estimation of the neural network. Note that these individuals are chosen only to calculate the error measure.

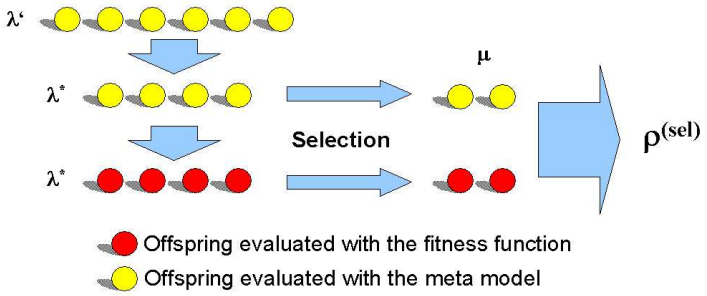

Fig. 10.9. Illustration of how to evaluate the selection-based quality measure

Afterwards, all λ^* individuals are evaluated using the original fitness function and μ individuals will be selected out of the λ_t^* (where t is the generation index) offspring and the sum of the ranks of all m correctly selected individuals measures the quality $\rho^{(sel)}$ in the current generation:

$$\rho^{(sel)} = \sum_{i=1}^{m} (\lambda_t^* - i). \tag{10.8}$$

If all individuals are selected correctly, the quality measure reaches its maximum:

$$\rho_{max}^{(sel)} = \sum_{i=1}^{\mu} (\lambda_t^* - i). \tag{10.9}$$

The idea is to compare the actual quality measure with the expected quality of a random selection process $\langle \rho^{(rand)} \rangle$:

$$\langle \rho^{(rand)} \rangle = \frac{\mu^2}{\lambda_t^*} \cdot \frac{2\lambda_t^* - \mu - 1}{2}. \tag{10.10}$$

If the quality in the current generation is better than the quality of a random selection process, then the neural network can be used to replace the original fitness function more often. Otherwise, the neural network should be less often used. The adaptation rule differs a little bit whether λ^* (Equation 10.12, 10.14) or λ' (Equation 10.11, 10.13) will be adapted.

$\rho_t^{(sel)} > \rho^{(rand)}$:

$$\lambda'_{t+1} = \lambda'_t + \frac{\rho_t^{(sel)} - \langle \rho^{(rand)} \rangle}{\rho^{(max)} - \langle \rho^{(rand)} \rangle} \cdot \Delta\lambda, \tag{10.11}$$

$$\lambda^*_{t+1} = \lambda^*_t - \frac{\rho_t^{(sel)} - \langle \rho^{(rand)} \rangle}{\rho^{(max)} - \langle \rho^{(rand)} \rangle} \cdot \Delta\lambda. \tag{10.12}$$

$\rho_t^{(sel)} < \rho^{(rand)}$:

$$\lambda'_{t+1} = \lambda'_t - \frac{\langle \rho^{(rand)} \rangle - \rho_t^{(sel)}}{\langle \rho^{(rand)} \rangle} \cdot \Delta\lambda, \tag{10.13}$$

$$\lambda^*_{t+1} = \lambda^*_t + \frac{\langle \rho^{(rand)} \rangle - \rho_t^{(sel)}}{\langle \rho^{(rand)} \rangle} \cdot \Delta\lambda. \tag{10.14}$$

It has to be considered that λ^*_{t+1} is bigger than the number of parents μ and as in all other adaptation methods λ'_{t+1} has to be equal or bigger than λ^*_{t+1}. As in the normalized squared error based adaptation framework, the free parameter $\Delta\lambda$ has also to be specified. One drawback of the selection based approach is probably the small number of μ individuals taken into account to measure the quality, which could result in strong oscillations in adaptation. Note that CMA-ES often uses a small population size.

10.4.3 Correlation Based Adaptation Mechanism (Rank, Corr)

Using the correlation based framework, all λ^*_t offspring are taken into account to evaluate the quality measure. Two different possibilities are suggested in [15] to evaluate the correlation between the λ^*_t individuals. The first correlation based quality measure is the *rank correlation (Rank)*. To evaluate the rank correlation measure, after estimation the λ^*_t individuals are sorted by their fitness and a given rank. The same is done after evaluating the fitness with the original fitness function. The rank correlation quality measure can now be calculated in the following way:

$$\rho^{(rank)} = 1 - \frac{6 \sum_{l=1}^{\lambda^*_t} (r_l - \widehat{r}_l)^2}{\lambda^*_t((\lambda^*_t)^2 - 1)}, \tag{10.15}$$

where $\widehat{r}_l$ is the rank of the l'th individual based on the estimated fitness and r_l is the rank based on the real fitness.

The second correlation based quality measure is the so called *continuous correlation (Corr)*. This quality measure calculates the correlation between the fitness values instead of the ranks. So the continuous correlation between the approximate model output and the original fitness function can be calculated by using Equation 10.16:

$$\rho^{(corr)} = \frac{\frac{1}{\lambda^*_t} \sum_{l=1}^{\lambda^*_t} \left(\phi_l^{(MLP)} - \bar{\phi}^{(MLP)}\right)\left(\phi_l^{(Orig)} - \bar{\phi}^{(Orig)}\right)}{\sigma^{(MLP)}\sigma^{(Orig)}}. \tag{10.16}$$

Here $\bar{\phi^{(m)}}$ and $\sigma^{(m)}$ are the mean value and the standard derivation of the fitness values evaluated using the neural network and $\bar{\phi^{(o)}}$ and $\sigma^{(o)}$ are the mean and the standard derivation of the real fitness values.

The rules to adapt λ^* and λ' are similar to the rules in the normalized squared error driven mechanism:

$$\lambda^*_{t+1} = \lambda^*_t - \rho \cdot \Delta\lambda, \tag{10.17}$$

$$\lambda'_{t+1} = \lambda'_t + \rho \cdot \Delta\lambda, \tag{10.18}$$

where ρ stands for $\rho^{(corr)}$ or $\rho^{(rank)}$.

10.4.4 Simulation Results

The empirical results are presented to investigate whether the adaptation mechanisms are able to improve the performance of the individual-based evolution control methods. To dynamically control the impact of the neural network on the individual-based evolutionary control strategy, two parameters can be adjusted during optimization.

Simulations have been conducted using the pre-selection strategy, where the number of pre-selected individuals is adapted using the introduced adaptation mechanisms. As mentioned, $\Delta\lambda$ has to be specified before starting the simulations. $\Delta\lambda$ was determined during some experiments and varies with the different adaptation mechanisms as listed in Table 10.3.

	Sel	NERD	Rank, Corr
$\Delta\lambda$	12	24	16

Table 10.3. Configuration of the free parameter $\Delta\lambda$ to adapt the number of pre-selected individuals

The initial value for λ' was fixed to 12 and the parameters for the evolution strategy are set to $\big(3, [\lambda'(t)]12[12]\big)$. In all simulations, the structure of the neural network is optimized.

As one can see in Fig. 10.10, using the normalized error rank based (NERD) adaptation mechanism performs very poorly on all the test functions. In Fig.10.11, the change of λ' are shown on the left and the quality measure on the right. Using the error-based adaptation mechanism (NERD), it can be seen that λ' increases continuously on all test-functions. So the network error seems to decrease during optimization. This might indicate that the error of the neural network by itself is not directly correlated with the performance of the optimization process. Another reason might be that $\Delta\lambda$ has not been chosen correctly.

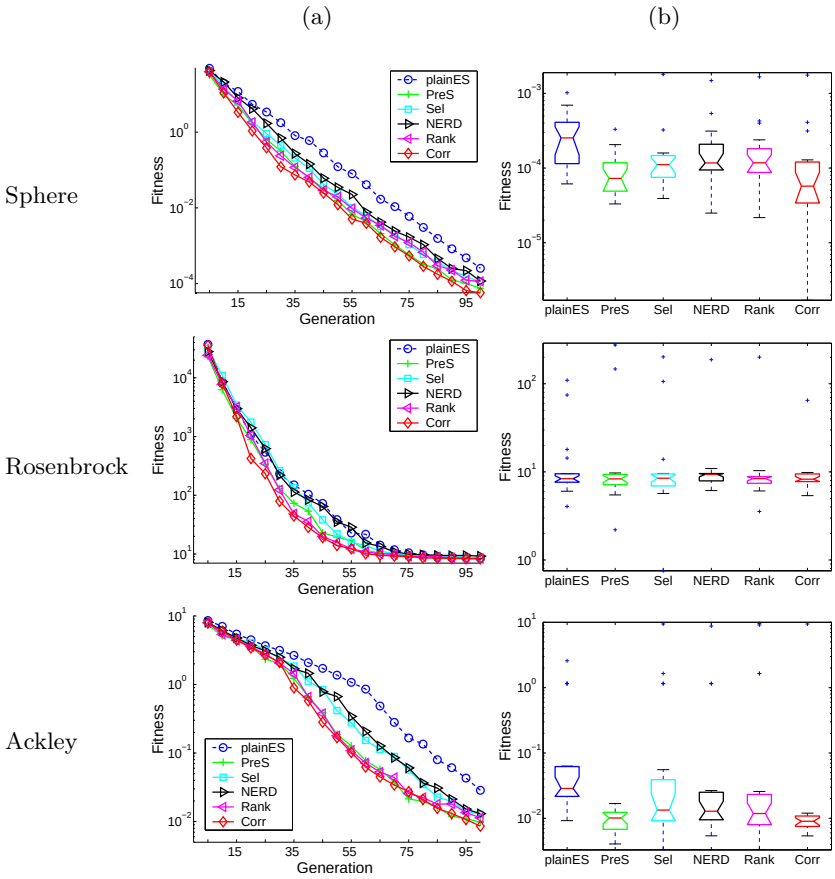

Fig. 10.10. Results using adaptation mechanisms to control the number of pre-selected individuals: (a) performance comparison and (b) boxplot over 20 runs after 100 generations

The selection based adaptation mechanism failed to perform well on all three test-functions too. The change of λ' and the quality measure oscillates dramatically, see Fig. 10.11. It is noticed that in [19], the selection-based quality measure has successfully been used to control the number of the pre-selected individuals. Note that the population size used in [19] is larger.

The best results in our simulations are obtained with the correlation-based adaptation mechanisms, especially the continuous correlation mechanism. But the correlation based adaptation mechanisms can not significant improve the pre-selection method without adaptation. Looking at the characteristic of λ' on Fig. 10.11, both adaptation mechanisms show the same trend. At the beginning the impact of the neural network to the optimization process steady increases. After some maximum is reached λ' decreases. The characteristics of

λ' appears to agree with the idea that the neural network should be less used until it is well trained. And if the optimization process comes closer to the optimum, the neural network should also be less used because the estimation is accurate enough to reach the real optimum.

From the above results, it seems that the adaptation mechanism is not successful in improving the performance of the pre-selection strategy and thus, the adaptation mechanisms are not tested on other model management frameworks.

10.5 3D Blade Design Optimization

In this section, we apply the pre-selection and the clustering with best strategy to 3D stator blade optimization.

10.5.1 Shape Representation

The 3D shape of the blade is approximated using three sections of 2D blades, as illustrated in Fig. 10.12(a). The hub section is directly connected to the hub at a radius of $R = 98.6mm$ from the engine axis. The tip section lies at a radius of $R = 130.0mm$. The 3D blade is built up by linear interpolation between these two sections. Another important section for the calculation of the design constraints is the casing section, which lies between the hub and the tip section at $R = 117.5mm$.

For each blade section, the blade length in axial direction is defined by the *axial chordlength*, refer to Fig. 10.12(b). The axial chordlength depends on the distance of the section to the engine axis.

Another parameter is the *axial solidity* s. For optimization, the solidity is measured at the casing section and the hub section. The final blade solidity is defined as the maximum value of these two measurements.

To describe the geometry of the blade, the *thickness* Θ can not be omitted. The thickness is also defined on a 2D section as the distance from the *medial axis* to a point p on the outline of the section. Depending on where the thickness is measured, the *trailing edge thickness* Θ_{TE}, near the trailing edge of the blade and the *leading edge thickness* Θ_{LE} near the leading edge are considered. Also important are the minimal Θ_{min} and maximal Θ_{max} values of the thickness.

The last parameter introduced here is the *throat area* (Fig. 10.12a). The throat area is defined as the area between two adjacent blades.

In this representation, 88 design parameters need to be optimized.

10.5.2 Performance Evaluation

Given the geometry of the 3D blade, the performance can be described by the *pressure loss*, which has to be minimized under certain constraints. This

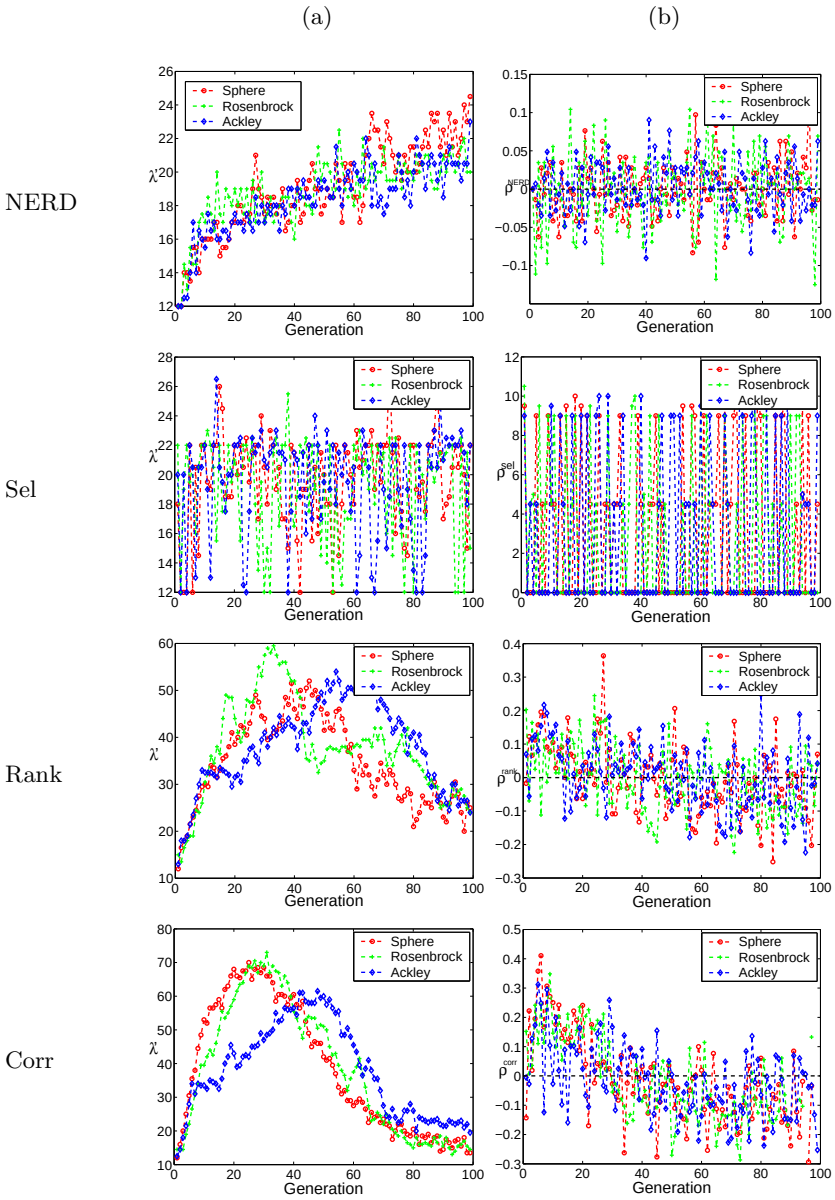

Fig. 10.11. Development of (a) λ' and (b) the quality measure during optimization

leads to the following objective or fitness function as a weighted sum of the pressure loss and the blade constraints. A penalty is applied in term of a big weight w_i so that the fitness becomes worse when a constraint is violated.

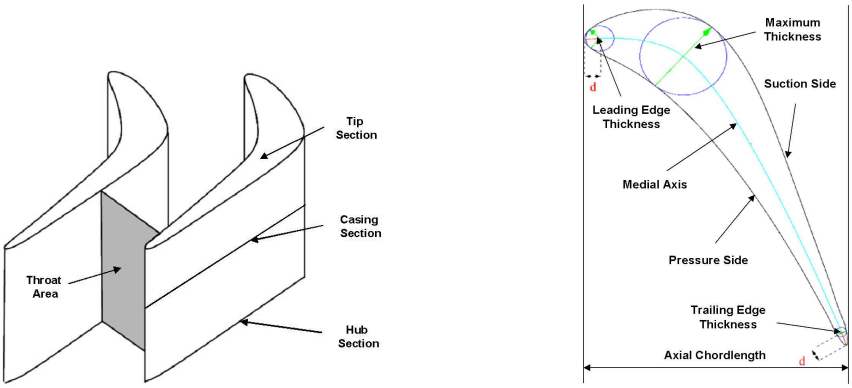

Fig. 10.12. Parameters and terminologies in 3D turbine blade design

$$f = w_0 t_0 + \sum_{i=1}^{4} w_i t_i^2 \longrightarrow min, \tag{10.19}$$

where t_0 is the pressure loss of the given blade, and t_i are the following constraints:

- $t_1 : max\big(0, |\beta_{2,design} - \beta_2| - \delta\beta_2\big)$,
- $t_2 : max\big(0, \Theta_{min,design} - \Theta_{min}\big)$,
- $t_3 : max\big(0, \Theta_{TE,min,design} - \Theta_{TE,min}\big)$,
- $t_4 : max\big(0, s_{max} - s_{max,design}\big)$,

where the following design values and tolerances are used:

- $\beta_{2,design} = 72.0deg$
- $\delta\beta_2 = 0.5deg$
- $\Theta_{min,design} = 0.72mm$
- $\Theta_{TE,min,design} = 0.9mm$
- $s_{max,design} = 0.706$.

The geometrical constraints like the minimal thickness Θ, trailing edge thickness Θ_{TE}, and the solidity s can all be determined directly from the geometry of the blade. However the *outlet angle* β_2 and the pressure loss can only be calculated from the results of the computational fluid dynamics (CFD) simulations. To simulate the fluid dynamics, the parallelized 3D Navier-Stokes flow solver HSTAR3D is used. The computational time of a CFD simulation varies between 2.5 and 6 hours on an AMD Opteron 2GHz dual processor. After flow analysis, each blade can be assigned a corresponding fitness value using Equation 10.19.

To investigate whether individual-based evolution control methods give a benefit to real world optimization problems, two methods are implemented

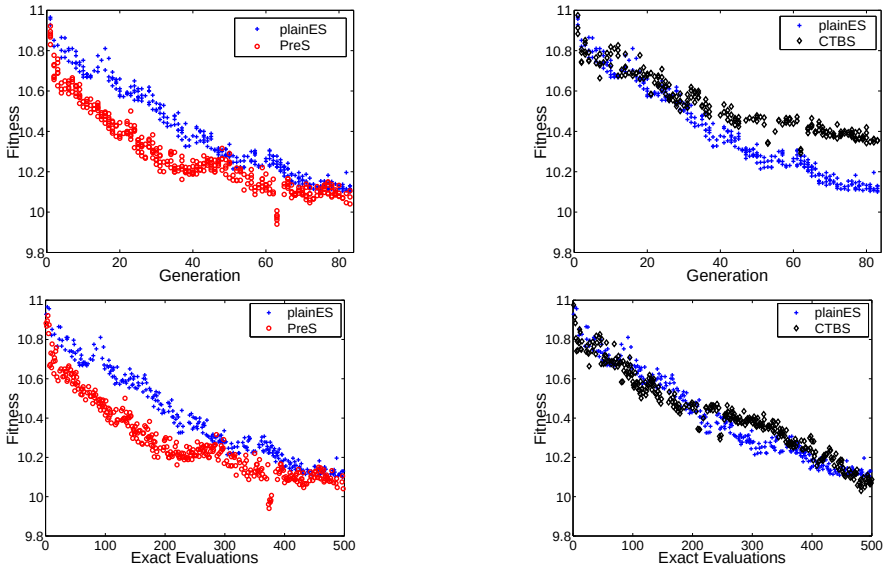

Fig. 10.13. Comparison of the performance between the plain evolution strategy and (a) using the pre-selection strategy or (b) using the clustering technique with best selection

in the 3D blade design optimization problem. The two methods are the $(1, [12]6[6])$ pre-selected strategy and the $(1, [6]6[4])$ clustering technique with best strategy. The performance of the methods are compared with the $(1, 6)$ plain evolution strategy. By use of the model assisted methods, the computationally expensive flow analysis was partially replaced by the neural network. The neural network consists of 88 input nodes, 4 hidden nodes and 2 output nodes. Using the clustering technique with best strategy, the performance of only 4 instead of 6 individuals each generation was determined by evaluating the CFD. Therefore the methods are compared by the fitness over the number of generations and also by the fitness over the number of exact fitness function evaluations, see Fig. 10.13.

As can be seen in 10.13(a), using the pre-selection strategy gives a benefit to the plain evolution strategy. It was possible to save up to about 20 generations to reach the same fitness value. The fitness of all individuals in each generation was evaluated in parallel. If the evaluation of the fitness takes about three hours, we saved about 60 hours of computational time. But the gap between the plain evolution strategy and the pre-selection strategy over the entire optimization process is nearly constant. There is no dramatic improvement in performance compared to the plain evolution strategy.

Using the clustering technique with best strategy, there is no improvement to the plain optimization process, refer to 10.13(b). Comparing the fitness over

the number of generations, it might be clear that the $(1, 6)$ plain evolution strategy can not be improved if only 4 individuals each generation are evaluated with the exact fitness function. Further it should be analyzed whether the clustering technique with best strategy might improve the $(1, 4)$ evolution strategy. But comparing the fitness against the number of exact fitness function evaluations, there is also no improvement to the plain optimization process.

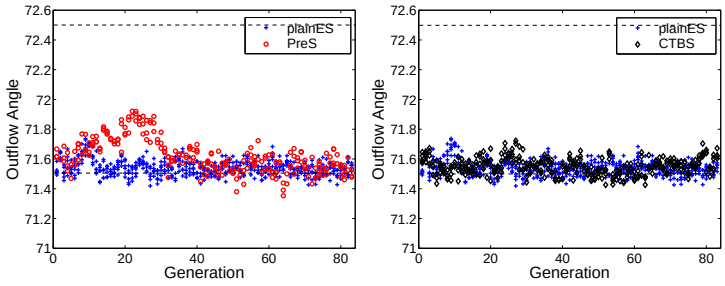

Fig. 10.14. Comparison of the outflow angle between the plain evolution strategy and (a) using the pre-selection strategy or (b) using the clustering technique with best selection

The fitness for each individual was determined by using equation 10.19. If the geometry of the blade does not violate any constraint, the pressure loss equals the fitness value because the weight for the pressure loss is set to one. Otherwise, if the geometry violates some constraints, a penalty of $10^2 2$ was given and the fitness becomes worse. Because the fitness values of invalid blade geometries are very large, these fitness values are not illustrated in Fig. 10.13.

A comparison of the outflow angle of the plain evolution strategy and the individual controlled strategies is shown in Fig. 10.14. It can be seen that the value for the outflow angle is near the lower bound. One exception occurs when the pre-selection strategy is used, where the outflow angle differs from the plain evolution strategy in the first generations.

To investigate how good the neural network substitute the time-expensive CFD-calculations, the estimation error of the two neural network outputs are illustrated in Fig. 10.15. The upper panel illustrates the estimation error of the pressure loss and the bottom panel shows the estimation error of the outflow angle. The squared error of the value evaluated with the CFD-calculation and the value estimated with the neural network has been plotted. It can be seen that the most values lie between 10^{-2} and 10^{-4}. Recall that the pressure loss is about 10 and the outflow angle is about 71.5, which indicates that the estimation of the neural network is quite good.

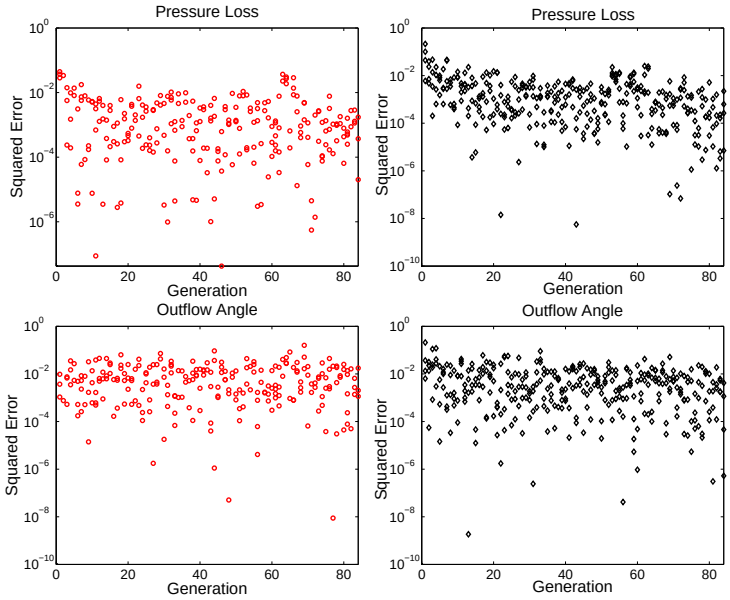

Fig. 10.15. Illustration of the neural network error for the pressure loss and the outflow angle (a) using the pre-selection strategy and (b) using the clustering technique with best selection

10.6 Conclusions

In this chapter, we have present four individual-based evolution control methods in order to reduce the number of expensive fitness evaluations. The performance of the methods is tested on three widely used benchmark functions. The pre-selection strategy shows the best results in both serial and in parallel computational environments, which improves the plain evolution strategy significant on mostly all test functions. The stability of the individual based evolution strategy might be improved if the parents for the next generation are selected out of the individuals evaluated with the original fitness function, as it is done in the pre-selection strategy.

To adaptively control the impact of the neural network on the evolutionary process, different adaptation mechanisms are investigated. The number of pre-selected individuals was controlled using the pre-selection strategy. Preliminary simulation results are not very promising. Further research should be done to see whether the free parameter $\Delta\lambda$ was chosen correctly. It should be analyzed why the results presented in [19] are much better than in our work.

To investigate whether individual-based evolution control methods are practical in real world optimization problems, the pre-selection strategy and the clustering technique with best strategy are implemented in the 3D blade

design optimization. It turned out that the pre-selection strategy gives a benefit to the performance of the optimization process. The neural network sufficient substitutes the CFD-calculation.

There are a few possible reasons that lead to the relatively poor performance of the clustering methods. First, the population size used in this work is very small, which makes the clustering less sensible. Second, we only used a single neural network contrasting the work in [12], where a neural network ensemble has been used. In the future work, it should be investigated whether a neural network ensemble instead of a single neural network can improve the estimation accuracy of the model.

References

1. G. Cybenko. Approximation by superposition of a sigmoidal function. *Math. Control Signals Systems*, 2:303–314, 1989.
2. K. Foli, M. Olhofer T. Okabe, Y. Jin, and B. Sendhoff. Optimization of micro heat exchanger: CFD, analytical approach and multi-objective evolutionary algorithms. *International Journal of Heat and Mass Transfer*, 49:1090–1099, 2006.
3. L. Gräning, Y. Jin, and B. Sendhoff. Efficient evolutionary optimization using individual-based evolution control and neural networks: A comparative study. In *European Symposium on Artificial Neural Networks*, pages 273–278, 2005.
4. N. Hansen and S. Kern. Evaluating the cma evolution strategy on multimodal test functions. In *Eight International Conference on Parallel Poblem Solving from Nature PPSN VIII*, pages 282–291. Springer, 2004.
5. M. Hasenjäger, B. Sendhoff, T. Sonoda, and T. Arima. Three dimensional aerodynamic optimization for an ultra-low aspect ratio transonic turbine stator blade. In *ASME Turbo Expo*, 2005.
6. K. Hornik, M. Stinchcombe, and H. White. Multilayer feedforward networks are universal approximators. *Neural Networks*, 2(5):359–366, 1989.
7. M. Hüsken, Y. Jin, and B. Sendhoff. Structure optimization of neural networks for evolutionary design optimization. In *GECCO Workshop on Approximation and Learning in Evolutionary Computation*, pages 13–16, 2002.
8. Y. Jin. A comprehensive survey of fitness approximation in evolutionary computation. *Soft Computing*, 9(1):3–12, 2005.
9. Y. Jin, M. Huesken, and B. Sendhoff. Quality measures for approximate models in evolutionary computation. In *Proceedings of GECCO Workshops: Workshop on Adaptation, Learning and Approximation in Evolutionary Computation*, pages 170–174, Chicago, 2003.
10. Y. Jin, M. Olhofer, and B. Sendhoff. On evolutionary optimization with approximate fitness functions. In *Genetic and Evolutionary Computation Conference*, pages 786–792. Morgan Kaufmann, 2000.
11. Y. Jin, M. Olhofer, and B. Sendhoff. A framework for evolutionary optimization with approximate fitness functions. *IEEE Transactions on Evolutionary Computation*, 6(5):481–494, 2002.
12. Y. Jin and B. Sendhoff. Reducing fitness evaluations using clustering techniques and neural networks ensembles. In *Genetic and Evolutionary Computation Conference*, volume 3102 of *LNCS*, pages 688–699. Springer, 2004.

13. M. Olhofer, T. Arima, Y. Jin, T. Sonoda, and B. Senhoff. Optimisation of transonic gas turbine blades with evolution strategies. *Honda Technical Reviews*, pages 203–216, April 2002. documents/HTR02.pdf.
14. M. Olhofer, T. Arima, T. Sonoda, and B. Sendhoff. Optimisation of a stator blade used in a transonic compressor cascade with evolution strategies. In *Adaptive Computation in Design and Manufacture*, pages 45–54. Springer, 2000.
15. Y.S. Ong, P.B. Nair, and A.J. Keane. Evolutionary optimization of computationally expensive problems via surrogate modeling. *AIAA Journal*, 41(4):687–696, 2003.
16. Y.S. Ong, P.B. Nair, A.J. Keane, and K.W. Wong. Surrogate-assisted evolutionary optimization frameworks for high-fidelity engineering design problems. In Y. Jin, editor, *Knowledge Incorporation in Evolutionary Computation*, pages 307–331. Springer, 2005.
17. A. Oyama. Multidisciplinary optimization of transonic wing design based on evolutionary algorithms coupled with cfd solver. In *European Congress on Computational Methods in Applied Science and Engineering, ECCOMAS 2000*, 2000.
18. M. Riedmiller and H. Braun. A direct adaptive method for faster backpropagation learning: The rprop algorithm. *IEEE Int. Conference on Neural Networks*, pages 586–591, 1993.
19. H. Ulmer, F. Streichert, and A. Zell. Evolution strategies with controlled model assistance. In *Congress on Evolutionary Computation*, pages 1569–1576, 2004.
20. Cheng Xiang. Geometrical interpretation and architecture selection of MLP. *IEEE Transaction on Neural Networks*, 16(1):84–96, 2005.
21. X. Yao. Evolving artificial neural networks. *Proceedings of the IEEE*, 87(9):1423–1447, 1999.