Managing Approximate Models in Evolutionary Aerodynamic Design Optimization

Yaochu Jin
yaochu_jin@de.hrdeu.com

Markus Olhofer
markus_olhofer@de.hrdeu.com

Bernhard Sendhoff
bs@el-tec.de

Future Technology Research Division
Honda R&D Europe (Deutschland) GmbH
63073 Offenbach/Main, Germany

Abstract- Approximate models have to be used in evolutionary optimization when the original fitness function is computationally very expensive. Unfortunately, the convergence property of the evolutionary algorithm is unclear when an approximate model is used for fitness evaluation because approximation errors are involved in the model. What is worse, the approximate model may introduce false optima that lead the evolutionary algorithm to a wrong solution. To address this problem, individual and generation based evolution control are introduced to ensure that the evolutionary algorithm using approximate fitness functions will converge correctly. A framework for managing approximate models in generation-based evolution control is proposed. This framework is well suited for parallel evolutionary optimization in which evaluation of the fitness function is time-consuming. Simulations on two bench-mark problems and one example of aerodynamic design optimization demonstrate that the proposed algorithm is able to achieve a correct solution as well as a significantly reduced computation time.

1 Introduction

Evolutionary algorithms have widely been applied to optimization problems that are discontinuous, multi-modal and multi-objective [1, 2]. Aerodynamic structural optimization problems such as preliminary turbine design [3], turbine blade design [4] and multi-disciplinary turbine blade design [5] are some good examples.

Despite the success in structural design optimization, there are still difficulties that impede evolutionary algorithms to be applied to more complicated problems, e.g. 3-D design optimization. One essential difficulty is the prohibiting time consumption due to high complexity of the aerodynamic analysis and large number of evaluations needed in the evolutionary optimization. Several methods have been developed for constructing approximate models to alleviate this difficulty. One widely used method in design engineering is the Response Surface Methodology, which uses low-order polynomials and the least square estimations [6]. The Kriging model, which is also called the Design and Analysis of Computer Experiments (DACE) model [7], is another very useful tool. In this method, a global polynomial approximation is combined with a local Gaussian process and the maximum likelihood method is used for parameter estimation. In the last few years, artificial neural networks, including Multi-layer Perceptrons and Radial Basis Function networks have also been employed to build approximate models for design optimization. In [8] a more comprehensive review of different approximation concepts is provided. Comparisons of the different modeling techniques can be found in [9].

One major problem arising in using approximate models is the lack of sufficient training data, which prevents the users from achieving a model with sufficient approximation accuracy. Generally, since evaluation of the original fitness function is very time-consuming, data collection is computationally very expensive. Due to this, the approximate model may be of low fidelity and may even introduce false optima. In this case, measures have to be taken to guarantee the correct convergence of the optimization algorithm when approximate models are used. In this paper, the correct convergence means that the minimum found by the evolutionary algorithm is a near-minimum or the global minimum.

Model management has been investigated in conventional optimization with approximate models. In [10], a framework for managing approximate models has been proposed based on the classical trust-region methods [11]. An extension of this work has been reported in [12]. One main feature of the framework for managing approximate models is the strong interplay between the optimization and the fidelity of the approximate model based on the trust-region method, which ensures that the search process converges to a reasonable solution of the original problem.

Managing approximate models in optimization based on evolutionary algorithms has not caught much attention until recently. In [13], a heuristic convergence criterion is used to determine when the approximate model must be updated. The basic idea is that the convergence of the search process should be stable and therefore, the change of the best solution should not be larger than a user-defined value. An assumption is that the first sets of data points are weakly correlated with the global optimum of the original problem, which is not necessarily true for high dimensional systems. An approach to coupling approximate models with evolutionary algorithms is proposed in [14] in an attempt to balance the concern of optimization with that of design of experiments. The main idea is to maintain the diversity of the individuals and to select those data points that are not redundant for model updating (online learning). In this method, when to carry out the on-line learning of the approximate model is simply based on a prescribed generation delay. Most recently, we noticed the paper [15], in which a neural network is trained with some initial samples to approximate the NK model. During evolution, the fittest individual in the current population is evaluated on the original fitness function for every 50 generations. This individual then replaces the one with the lowest fitness in the training set and the neural network is retrained. It is found that the evolutionary algorithm becomes misled by the neural network model when the complexity of the original fitness landscape is high. The common weakness in the above methods is that neither the convergence property of the evolutionary algorithm with approximate fitness functions (correct convergence is assumed) nor the issue of model management is addressed.

As we have shown in [16], an incorrect convergence occurs when the approximate fitness function has false optima. To improve the convergence of the evolutionary algorithm, the concept of evolution control is introduced. By evolution control, we mean that not only the approximate fitness function but also the original fitness function will be used in evolution. There are two possibilities to combine the true fitness function with the approximate fitness function. One approach is called individual-based evolution control, in which a certain number of individuals within a generation are evaluated with the true fitness function. Such individuals are called controlled individuals. The second approach is to introduce generation-based evolution control, which means that in every M generations, $N (N \leq M)$ generations will be controlled. In a controlled generation, all the individuals are evaluated with the true fitness function. For the sake of convenience, every M generations is called a control cycle, M the size of the control cycle, and N the evolution control frequency.

When the fitness function is computationally expensive, parallel evolutionary algorithm will often be used. In this case, generation based evolution control is more attractive because it is suitable for parallelization. The question now is to determine the evolution control frequency so that the evolutionary algorithm converges correctly with as few calls of the original fitness function as possible. To this end, a framework for model management with generation based evolution control is proposed. The main idea is that the frequency at which the original function is called and the approximate model is updated should be determined by the local fidelity of the approximate model. By local fidelity, we mean the fidelity of the model for the region where the current population is located. The lower the model fidelity is, the more frequently the original function should be called and the approximate model should be updated. Since it is hardly possible to build a globally correct approximate model for problems with a large dimensionality in the design space, a local model is of more practical importance. With this strategy, the computational cost can be reduced as much as possible while the correct convergence of the evolutionary algorithm can be guaranteed.

An evolution strategy with covariance matrix adaptation (CMA) [17] is adopted in this work. Besides its attractive convergence speed, the self-adaptation of the covariance matrix can also be taken advantage of during on-line learning. The basic idea is that the new data points that lie along the direction in which the evolutionary algorithm proceeds should be given larger weight in on-line learning. Very interestingly, rough information on the directions is contained in the covariance matrix. Results from the benchmark problems show that better results can be achieved when the covariance matrix is used to weight the new samples in on-line learning.

The remainder of the paper is organized as follows. Section 2 presents the evolution strategy used in our work along with an explanation of the covariance matrix adaptation. The parallel implementation of the evolutionary algorithm is briefly described. In Section 3, two approaches to evolution control are suggested to improve the convergence of the evolutionary algorithm with approximate fitness functions. A framework for evolutionary optimization with approximate fitness functions is proposed in Section 4, which includes the evolution control strategy, the determination of the frequency for evolution control, as well as the weighted learning algorithm on the basis of the covariance matrix. The effectiveness of this framework is demonstrated with examples on two benchmark problems and one application example in Section 5. A summary of the results concludes the paper in Section 6.

2 Evolution Strategy with Covariance Matrix Adaptation

The standard Evolution Strategy (ES) can be described as follows:

$$\sigma_i(t) = \sigma_i(t-1)\exp(\tau' z)\ \exp(\tau z_i) \qquad (1)$$

$$\vec{x}(t) = \vec{x}(t-1) + \vec{\tilde{z}} \qquad (2)$$

$$z_i, z \sim N(0,1);\ \vec{\tilde{z}} \sim N(\vec{0}, \vec{\sigma}(t)^2) \qquad (3)$$

where $\vec{x}$ is the parameter vector to be optimized and τ, τ' and σ_i are the strategy parameters. The σ_i are also called step-sizes and are subject to self-adaptation, as shown in equation (2). The z_i, z and $\vec{\tilde{z}}$ are normally distributed random numbers and a random number vector, respectively, which characterize the mutation exercised on the strategy and the objective parameters.

The derandomized Covariance Matrix Adaptation (CMA) [17] differs from the standard ES mainly in three respects:

- In order to reduce the stochastic influence on the self-adaptation, one stochastic source for both the adaptation of the objective and of the strategy parameters is used. In the derandomized approach, the *actual* step length in the objective parameter space is used to adapt the strategy parameter. Therefore, the self-adaptation of the strategy parameters depends more directly on the local topology of the search space.

- The second method is the introduction of the cumulative step size adaptation. Whereas the standard evolution strategy extracts the necessary information for the adaptation of the strategy parameters from the population (ensemble approach), the cumulative step size adaptation additionally relies on information collected during successive generations (time averaged approach)[1]. This leads to a reduction of the necessary population size.

- In the CMA algorithm the full covariance matrix of the probability density function

$$f(\vec{z}) = \frac{\sqrt{\det(\vec{C}^{-1})}}{(2\pi)^{n/2}} \exp\left(-\frac{1}{2}(\vec{z}^T\,\vec{C}^{-1}\,\vec{z})\right). \quad (4)$$

is adapted for the mutation of the objective parameter vector (B satisfies $\vec{C} = \vec{B}\vec{B}^T$ with $z_i \sim N(0,1)$, then $\vec{B}\,\vec{z} \sim N(0,\vec{C})$; $\delta(t-1)$ denotes the global step-size):

$$\vec{x}(t) = \vec{x}(t-1) + \delta(t-1)\,\vec{B}(t-1)\,\vec{z}, \quad z_i \sim N(0,1). \quad (5)$$

Since $\vec{C}^{-1}$ has to be positive definite with $\det(\vec{C}^{-1}) > 0$, the different matrix entries cannot be determined independently and the detailed adaptation algorithm is a little more involved, see [17, 18] for a detailed description.

The evolution strategy is parallelized in this work. The basic motivation to implement an evolutionary algorithm in parallel is to reduce the processing time needed to reach an acceptable solution. This is badly in need when the evaluation of the fitness takes a large amount of time. There are several approaches to parallelization and the one we adopt here is the global parallelization. In this approach, there is one single population and the evaluation of the individuals is done in parallel. For design problems, the evaluation of the individuals usually takes up the overwhelming part of the total time consumption, therefore, a sub-linear speed-up can be achieved if the global parallelization approach is used. The hardware for the implementation of parallel evolutionary algorithms can be very different. In our case, a network of computers with multi-processors is used. The implementation of the parallelization is realized with the *Parallel Virtual Machines* library.

3 Evolution Control

If the problem we are dealing with is of low dimensionality, both random sampling and regularized learning [16] are very helpful to prevent an approximate model from producing false minima. However, if the dimension of the model is very high, it will be very difficult to achieve correct convergence using these methods. To solve this problem, we introduce the concept of evolution control. By evolution control, we mean to employ the original fitness function when necessary to prevent the evolutionary algorithm from converging to a false minimum. Two methods are proposed:

Individual-based control In this approach, part of the individuals (m) in the population (P in total) are chosen and evaluated with the original fitness function. If the controlled individuals are chosen randomly, we call it a **random strategy**. If we choose the best m individuals as the controlled individuals, we call it a **best strategy**.

Generation-based control In this approach, the whole population of η generations will be evaluated with the real fitness function in every λ generations, where $\eta \leq \lambda$.

Furthermore, when controlled individuals or controlled generations are introduced, new training data becomes available. Therefore, on-line learning of the neural network will be applied to improve the approximation of the network model in the region of optimization and in turn to improve the convergence of the evolutionary algorithm.

In both evolution control schemes, it is very important to determine the minimal control frequency that is needed to guarantee a correct convergence. Empirical investigations [16] on the individual-based evolution control approach showed that more than half of the individuals need to be controlled when the random strategy is used. If the best strategy is used in individual-based evolution control, about 40% of the individuals should be controlled. It is assumed that the approximate model has false minima. Similar results have been obtained for generation-based evolution control. That is to say, if a fixed evolution control frequency is used, about half of the generations should be controlled to ensure correct convergence.

Although the effectiveness of these two evolution control methods is comparable, generation-based evolution control is more suitable when parallel evolution strategies are implemented and the number of processors equals the number of individuals in the offspring population. It is evident that in global parallelization (master-slave mode), the needed computation time for each generation is determined by the individual with the maximal time consumption. Therefore, use of computationally efficient approximate models with individual-based evolution control will not be able to reduce time consumption for parallel implementation. Due to this reason, generation-based evolution control is applied in our framework for managing approximate models.

[1] As with all analogies, the division into ensemble and time averaged approach should not be taken too literal. Firstly, also in the standard ES no explicit averaging over the inviduals in each generation takes place (besides a recombination operator), but only an indirect exploitation of the ensemble (the population) in the selection process. Secondly, the "time averaging" in the derandomized ES occurs in addition to the emsemble "average". However, still we feel that the analogy helps to gain some understanding of how both strategies operate, e.g. why the derandomized methods work well even with small population sizes.

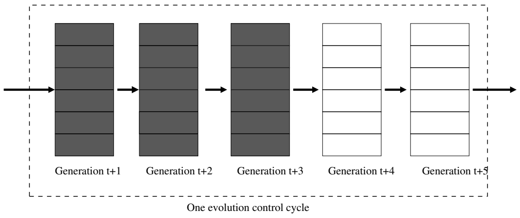

Figure 1: One evolution control cycle. The controlled generations are denoted by filled boxes.

4 Managing Approximate Models

4.1 Generation-based Evolution Control

As suggested in the last section, generation-based evolution control is employed in this work. For the sake of convenience, a more detailed description of the generation-based evolution control approach is presented in the following. Fig. 1 shows an evolution control cycle. In this example, the cycle consists of 5 generations, among which 3 generations are controlled. As we mentioned, the number of generations to be controlled within an evolution control cycle is called control frequency for short. Our task is to adjust the control frequency on-line so that the calls of the computationally expensive original fitness function can be reduced as much as possible without affecting the correct convergence of the algorithm.

4.2 Determination of Control Frequency

It is intuitive that the higher the fidelity of the approximate model is, the more often the fitness evaluation can be made using the approximate model and the smaller η can be. However, it is very difficult to estimate the global fidelity of the approximate model. Thus, a local estimation of the model fidelity has to be used. This is feasible because the evolution strategy generally proceeds with small steps, i.e., with the Gaussian distribution small mutations are most likely. Therefore, we can use the current model error to estimate the local fidelity of the approximate model and then to determine the frequency at which the original fitness function is used and the approximate model is updated. Suppose there are λ generations within an evolution control cycle, and η generations need to be controlled. To obtain a proper η, heuristic fuzzy rules can be derived as follows:

If the model error is large, then η is large.

Since the rule system has only one input variable, the input-output mapping of such a fuzzy system can approximately be expressed by:

$$\eta(k+1) = \eta_{min} + \left\lfloor \frac{E(k)}{E_{max}} \right\rfloor (\eta_{max} - \eta_{min}), \qquad (6)$$

where $\lfloor x \rfloor$ denotes the largest integer that is smaller than x, η_{max} is the maximal η, $\eta_{max} \leq \lambda$, and η_{min} usually equals 1 so that the information on the model fidelity is always available. E_{max} is the allowed maximal model error and $E(k)$ is the current model error estimation, k denotes the k-th evolution control cycle of λ generations.

The estimation of the current model error is carried out before the next cycle begins. Suppose all the new data in the last η generations are valid (in aerodynamic optimization, some designs may result in unstable fluid dynamics and therefore the data are invalid), there will be ηP data in total, where P is the population size. Thus, the model error is estimated as follows:

$$E(k) = \sqrt{\frac{1}{\eta P} \sum_{i=1}^{\eta P} (y(i) - y_{NN}(i))^2} \qquad (7)$$

where $y(i)$ is the true fitness value and $y_{NN}(i)$ is the fitness calculated using a feedforward neural network model:

$$y_{NN} = \sum_{j=1}^{H} v_j \theta(\sum_{i=1}^{n} w_{ij} x_i), \qquad (8)$$

where H is the number of hidden nodes, n is the number of inputs, w_{ij} is the weight between the input layer and the hidden layer, v_j is the weight between the hidden layer and the output layer, and $\theta(.)$ is the logic function

$$\theta(z) = \frac{1}{1+e^{-z}}. \qquad (9)$$

The framework for evolutionary optimization with approximate models can be summarized as follows:

begin
  Initilize $P(0)$, $\eta(0)$, λ, E_{max}, t_{max}, let $k = 0$
  for $t = 0$ **to** t_{max}
    if $t\%\lambda < \eta(k) - 1$
      Use the original fitness function
      Storage the data for model updating
    else
      Use the approximate model
    end if
    if $t\%\lambda = \lambda - 1$
      Estimate the local model fidelity $E(k)$
      Estimate $\eta(k+1)$
      Update the approximate model, $k = k + 1$
    end if
  end for
end

To make sure that the information on the model fidelity is always available, at least one generation should be controlled within one evolution control cycle. Since the fidelity of the model is estimated locally based on the error information from the last cycle, λ should not be too large.

4.3 Weighted On-line Learning Using Covariance Matrix

On-line learning should be efficient to improve the model fidelity without taking much time. One approach to enhancing the learning efficiency is to actively select new samples for network training. There are a number of methods for active data selection in neural network learning, which are usually called active learning [19, 20]. However, these active learning methods are mostly very time-consuming and are not suitable for on-line learning.

For on-line learning during optimization, the new data samples are 'selected' by the evolution strategy. Very interestingly, we can get a clue for which data points will most likely be visited by the evolution strategy in the next generation from the covariance matrix. Based on this information, the neural network is able to determine which data points are to be learned more rigorously. Suppose there are N new samples collected at the end of one control cycle, then the cost function for weighted learning is defined by

$$E = \frac{1}{N}\sum_{i=1}^{N} p(i)(y(i) - y_{NN}(i))^2, \quad (10)$$

where, $p(i)$ is the weight for sample i, which is calculated from the covariance matrix:

$$p(i) = \exp\left(-\frac{1}{2}(\vec{x}(i) - \vec{x}_P)^T C^{-1}(\vec{x}(i) - \vec{x}_P)\right), \quad (11)$$

where $\vec{x}(i)$ is an arbitary point in the parameter space, $\vec{x}_P$ is an individual in the current parent population to be referenced, C is the covariance matrix defined in equation (4). If the parameter vector $\vec{x}$ is close to the origin (zero), equation (11) can be approximated by

$$p(i) = \exp\left(-\frac{1}{2}\vec{x}(i)^T C^{-1}\vec{x}(i)\right). \quad (12)$$

Before applying the weights to neural network learning, they need to be normalized

$$p(i) = \frac{p(i)}{p_{max}} \quad (13)$$

where p_{max} is the maximal weight among $p(i), i = 1, 2, ..., N$. In this way, the neural network is able to learn the most important samples, and those with a weight smaller than a given threshold are discarded in learning.

5 Numerical Examples

In order to verify the feasibility of the proposed approach to managing approximate models, numerical studies are carried out on the Ackley function, the Rosenbrock function and an aerodynamic design example.

5.1 The 20-D Ackley Function

The first experimental study is conducted on the 20-dimensional Ackley function. A (2,12) evolution strategy with covariance matrix adaptation is adopted. In the simulation, λ is set to 6, the maximal η is set to 4 and the minimal η to 1. The maximal number of generations is 300.

Table 1: Generation-based evolution control

	Best Fitness	Number of CFD Calls
Average	0.98	1440
Variance	0.41	99.8

Table 2: Evolution with the original function only.

	Best Fitness	Number of CFD Calls
Average	2.33	1440
Variance	1.69	0

The results, including the best fitness and the number of calls of the real fitness function, are listed in Table 1. The average best fitness and the average η over 10 runs are 0.98 and 2.09. Compared with the results in [16], it is seen that the evolution control frequency to guarantee correct convergence has been significantly reduced through the proper estimation of the model fidelity.

To show that we can benefit from the introduction of the approximate model, we implement the optimization without using the approximate model but with the same number of evaluations of the original function. In the ten runs, the average number of calls of the original fitness function is 1440, which corresponds to 122 generations for a population size of 12. Therefore, we conduct 10 runs of the evolution with a maximum of 122 generations and the results are presented in Table 2. The average best fitness resulting from the 10 runs is 2.33. Comparing it with the average fitness using the approximate model with approximately the same computational overhead, we see that we have obtained a better result with the help of an approximate model. Note that we assume that the computational cost of the approximate model is negligible compared to that of the original fitness function.

In the above simulation, all new data are used for network training. In the following, we introduce the weighted learning based on the covariance matrix. In the simulation, it is found that the use of weighted learning should be introduced after the evolution process becomes relatively stable. This may be due to the fact that at the beginning of the evolution, the information obtained by the evolutionary algorithm is still inaccurate. The results are presented in Table 3.

It is noticed that in 10 runs, the best fitness on average is further significantly improved, while the average number of calls of the original function increases remains approximately the same (fewer than 4 generations in total).

Table 3: Evolution control with weighted learning.

	Best Fitness	Number of CFD Calls
Average	0.81	1483
Variance	0.32	77.8

5.2 The 20-D Rosenbrock Function

Simulations are also carried out on the 20-D Rosenbrock function. Similarly, simulations are done in three different cases, namely, evolution using both the approximate model and the original model, evolution using the original model only, and evolution with a combination of the approximate model and the original model with weighted learning.

When evolution control is employed, the best fitness is 35.5 and 865 calls of the original function are necessary on average within 200 generations. Since 865 calls equals approximately 73 generations with a population size of 12, we then run the evolution for 73 generations with the original function only.

The average best fitness is 207.3. Therefore, we have again achieved a much better result with the help of the approximate model. Finally, we introduce the weighted learning. The average best fitness is 25.8, again better than the results without weighted learning. We noticed that an increase of 83 calls of the original function is needed in this case, which is about 7 generations more. Therefore, we conduct another 10 runs with 100 generation using the original fitness function only and the average best fitness is 87.4. This demonstrates that we do benefit from using an approximate model by use of the proposed framework.

5.3 Blade Design Optimization

Aerodynamic design optimization is difficult partly due to the fact that complex computational fluid dynamics (CFD) simulations have to be conducted to evaluate the performance of a structure. Usually, CFD simulations are very time-consuming. For example, one 2-D CFD simulation based on Navier-Stokes equations with turbulence model takes about 30 minutes CPU time on a high-performance computer, and one 3-D CFD simulation more than 10 hours. Therefore, it is difficult to apply evolutionary algorithms to this kind of design optimization, since usually evolutionary algorithms need hundreds or thousands of performance evaluations. To address this problem, approximate models are introduced in optimization.

The objective of the optimization in our study is to maximize the efficiency of a turbine blade and to minimize the bias of the outflow angle from a prescribed value. The maximization of efficiency is realized by the minimization of the pressure loss. In addition, mechanical constraints must be satisfied to ensure that the design is stable in mechanics and feasible for manufacturing. In order to describe the two dimensional cross section of the blade, a spline encoding based on Non-Uniform Rational B-Splines [21] is used. The spline is constructed using N 4-dimensional control points that define the control polygon. For the two dimensional geometry representation, the z-coordinate is fixed to zero. The weights defined in NURBS for each control point are fixed to reduce the dimension of the search space. As pointed out in [21], in many NURBS implementations, the weights are fixed, which does not impair much the representation capacity of NURBS. In this case each control point is described only by two parameters, i.e., the x and y coordinate of the point. Figure 2 illustrates a blade (solid line) that is generated by a spline with 7 control points. The dotted line shows the corresponding control polygon.

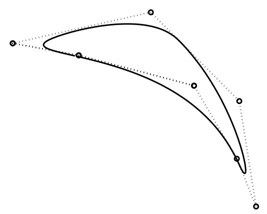

Figure 2: Representation of the blade geometry with B-splines.

A $(2, 11)$-ES is used for optimization. The population is initialized with a given blade to reduce the computation time. A neural network model is used to approximate the pressure loss and the outflow angle respectively. The size of the evolution control cycle is set to 6 and at least one generation will be controlled within one cycle.

(a)

(b)

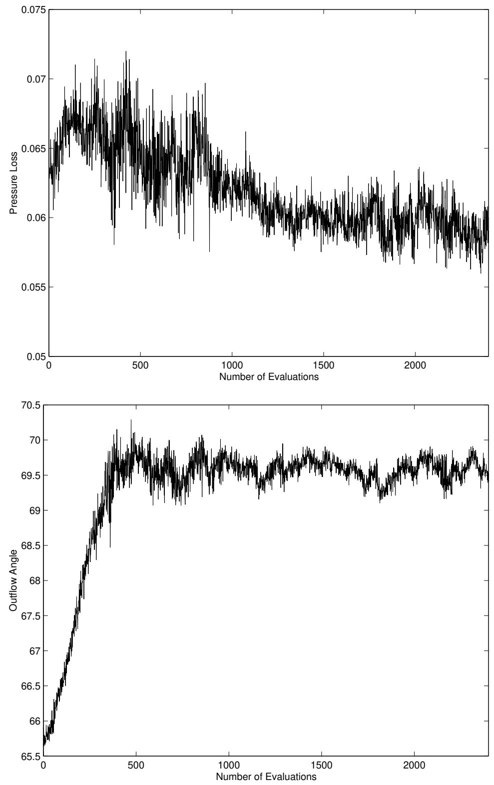

Figure 3: Blade optimization without approximate models: (a) Pressure loss, (b) Outflow angle.

The optimization was first run without any approximate models. In 218 generations, the Navier-Stokes solver was called 2398 times. The pressure loss and outflow angle are shown in Fig. 3. The minimal pressure loss is 0.058 with an outflow angle of 69.6.

In the next optimization experiment, a neural network is used for approximation of the pressure loss and a second network for the outflow angle. The networks are trained offline using data samples collected in other similar optimization tasks (but not the ones in the first optimization). In 261 generations, the Navier-Stokes solver was called for 2408 times and the network models were called 463 times. Note that the number of CFD calls is about the same in the optimization without approximate models. The change of the pressure loss and outflow angle is given in Fig. 4 and the obtained minimal pressure loss is 0.052 with an outflow angle of 69.6. This corresponds to a 10% reduction of the pressure loss. On the other hand, only about 660 Navier-Stokes calls (about 70 generation) are necessary to achieve the same loss as in the first optimization. Notice that in the figure, the number of evaluations includes both calls of the Navier-Stokes solver and that of the neural network models.

(a)

(b)

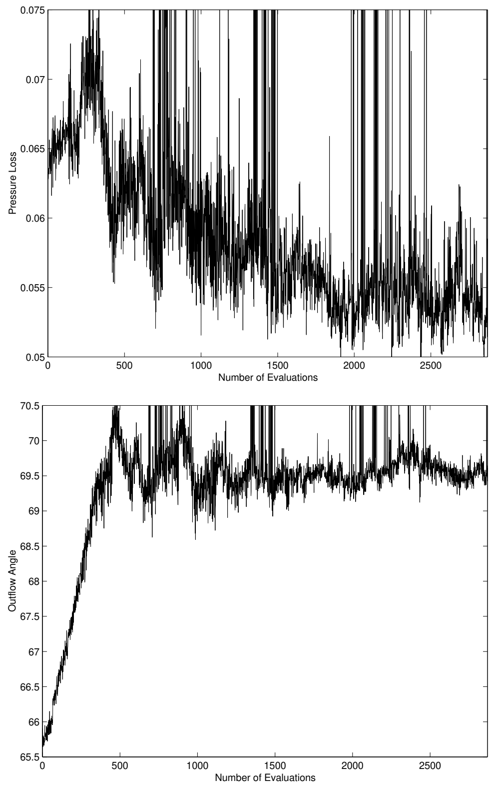

Figure 4: Blade optimization with approximate models: (a) Pressure loss, (b) Outflow angle.

Finally, the optimization with approximate models and weighted on-line learning is carried out. In this optimization, the weighted learning based on the information from the covariance matrix of the evolution strategy is employed. In 273 generations, the Navier-Stokes solver was called 2404 times. The achieved minimal pressure loss is 0.055 with an outflow angle of 69.6. Compared to the second optimization in which no weighted learning is applied, the performance is degraded from the viewpoint of the achieved minimal pressure loss. However, we still notice that the performance is about 5% better than that of the first run. In addition, we find the trajectory of the pressure loss during optimization (see Fig. 5) is smoother than that in the second run. It means that the average quality of the approximate model is better than that in the second run.

Compared with the results on the test functions, the performance of weighted learning in aerodynamic design optimization is worse. This might be ascribed to the fact that very small mutation steps are used in design optimization, since large mutation steps often result in an unstable behavior of the Navier-Stokes solver.

(a)

(b)

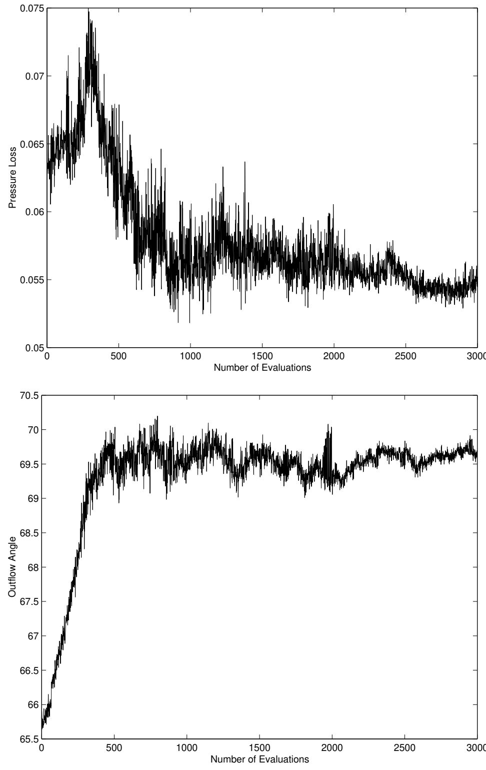

Figure 5: Blade optimization with approximate models and weighted learning: (a) Pressure loss, (b) Outflow angle.

6 Conclusions

Evolutionary optimization with approximate fitness functions is studied in this paper. Based on the empirical convergence studies on two benchmark problems, individual and generation based evolution control are suggested to ensure the correct convergence of an evolutionary algorithm using an approximate fitness function. A framework for managing the approximate model with the generation-based evolution control is proposed. The basic idea is to estimate the local model fidelity so that the frequency of evolution control can be on-line adjusted. In this way, the number of calls of the original computationally expensive function can be reduced and correct convergence can still be achieved.

Sample weighting is employed during on-line learning. The weighting information is derived from the covariance matrix of the evolution strategy. Results on both benchmark problems and an application example show that the proposed framework is very promising in that it is able to produce a good solution and reduce computation time.

ACKNOWLEDGEMENTS

The authors would like to thank T. Arima, T. Sonoda, E. Körner and W. von Seelen for their support.

Bibliography

[1] D. Goldberg D., K. Deb, and J. Clark. Genetic algorithms, noise, and the sizing of the populations. *Complex Systems*, 6, 1992.

[2] C.A. Coello Coello. An updated survey of evolutionary multiobjective optimization techniques: State of art and future trends. In *Proceedings of 1999 Congress on Evolutionary Computation*, pages 3–13, Washington D.C., 1999. IEEE Press.

[3] S. Tong and B. Gregory. Turbine preliminary design using artificial intelligence and numerical optimization techniques. *Journal of Turbomachinar*, 114(1), 1992.

[4] M. Olhofer, T. Arima, T. Sonoda, and B. Sendhoff. Optimization of a stator blade used in a transonic compressor cascade with evolution strategies. In I. C. Parmee, editor, *Evolutionary Design and Manufacture*, pages 45–54, 2000.

[5] P. Hajela and J. Lee. Genetic algorithms in multidisciplinary rotor blade design. In *Proceedings of 36th Structures, Structural Dynamics, and Material Conference*, New Orleans, 1998.

[6] R. Myers and D. Montgomery. *Response Surface Methodology*. John Wiley & Sons, Inc., New York, 1995.

[7] J. Sacks, W. J. Welch, T. J. Michell, and H. P. Wynn. Design and analysis of computer experiments. *Statistical Science*, 4:409–435, 1989.

[8] J. Bartelemy and R. T. Haftka. Approximation concepts for optimum structural design - a review. *Structural Optimization*, 5:129–144, 1993.

[9] W. Carpenter and J. Barthelemy. A comparison of polynomial approximations and artificial neural nets as response surfaces. *Structural Optimization*, 5:166–174, 1993.

[10] J. Dennis and V. Torczon. Managing approximate models in optimization. In N. Alexandrov and M. Hussani, editors, *Multidisciplinary design optimization: State-of-the-art*, pages 330–347. 1997.

[11] H. Schramm and J. Zowe. A version of the bundle idea for minimizing a nonsmooth function: Conceptual idea, convergence analysis. *SIAM Journal of Optimization*, 2, 1992.

[12] A. J. Brooker, J. Dennis, P. D. Frank, D. B. Serafini, V. Torczon, and M. Trosset. A rigorous framework for optimization of expensive functions by surrogates. *Structural Optimization*, 17:1–13, 1998.

[13] A. Ratle. Accelerating the convergence of evolutionary algorithms by fitness landscape approximation. In A. Eiben, Th. Bäck, M. Schoenauer, and H.-P. Schwefel, editors, *Parallel Problem Solving from Nature*, volume V, pages 87–96, 1998.

[14] M. El-Beltagy, P. Nair, and A. Keane. Metamodeling techniques for evolutionary optimization of expensive problems: Promises and limitations. In W. Banzhaf, J. Daida, A. Eiben, M. Gazon, V. Honavar, M. Jakiela, and R. Smith, editors, *Proceedings of Genetic and Evolutionary Conference*, pages 196–203, 1999.

[15] L. Bull. On model-based evolutionary computation. *Soft Computing*, 3:76–82, 1999.

[16] Y. Jin, M. Olhofer, and B. Sendhoff. On evolutionary optimization with approximate fitness functions. In *Proceedings of Genetic and Evolutionary Computation Conference.*, pages 786–792, Las Vegas, 2000.

[17] N. Hansen and A. Ostermeier. Completely derandomized self-adaptation in evolution strategies. *Evolutionary Computation*, 2000. To appear.

[18] M. Kreutz, B. Sendhoff, and Ch. Igel. *EALib: A C++ class library for evolutinary algorithms*. Institut für Neuroinformatik, Ruhr-Universität Bochum, 1.4 edition, March 1999. www.neuroinformatik.ruhr-uni-bochum.de/PROJECTS/SONN/Software/software.html.

[19] D. MacKay. Information-based objective functions for active data selection. *Neural Computation*, 4(4):305–318, 1992.

[20] S. Vijayakumar and H. Ogawa. Improving generalization ability through active learning. *IEICE Transactions on Information and Systems*, 82D(2):480–487, 1999.

[21] L. Piegl and W. Tiller. *The NURBS Book*. Springer, Berlin, 1997.