Hierarchical, Metamodel–Assisted Evolutionary Algorithms, with Industrial Applications

Kyriakos C. GIANNAKOGLOU *
Varvara G. ASOUTI †
Stelios A. KYRIACOU ‡
Xenofon S. TROMPOUKIS §

NATIONAL TECHNICAL UNIVERSITY OF ATHENS,
School of Mechanical Engineering,
Lab. Of Thermal Turbomachines,
Parallel CFD & Optimization Unit,
e-mail: kgianna@central.ntua.gr

May 2012

*Professor, NTUA
†Dr. Research Engineer, NTUA
‡Research Engineer, NTUA & Andritz HYDRO, RD, Linz, Austria
§Research Engineer, NTUA

Contents

1 Introduction

Nowadays, **Evolutionary Algorithms** (EAs) have become very attractive across a wide range of scientific fields/disciplines, including engineering optimization. They can efficiently handle complex, constrained, multi–objective optimization problems by accommodating any ready–to–use analysis software without necessarily having access to its source code. The main disadvantage of EAs is the great number of evaluations required to reach the optimal solution(s) which, depending on the computational cost of the evaluation software (CFD, CSM, etc), may increase a lot the optimization turnaround time. The most commonly used techniques to shorten the CPU cost of an EA–based optimization and, consequently, allow the use of EAs for solving computationally demanding industrial problems include the use of: (a) surrogate evaluation models (or metamodels) (b) distributed search algorithms (c) hierarchical algorithms and/or (d) parallel and/or asynchronous variants of EAs. In this lecture, an overview of all these techniques is presented; their implementation within a generalized EA will be explained in detail. In what follows, ths CFD or CSM etc. software will be referred to as the **exact evaluation model** or the **problem–specific model**, so as to make a clear distinction from any surrogate model in use.

The first part is concerned with the use of metamodels within EAs, which gives rise to the so–called **Metamodel–Assisted EAs** (MAEAs). Metamodels are low–cost approximations to the costly problem–specific evaluation model and can be used to evaluate the population members of an EA with less accuracy and much lower CPU cost. Off– and the on–line trained metamodels can be used. They differ by the timing of the metamodels' training, i.e. whether this takes place before (off–line) or during (on–line) the evolution. Another classification of MAEAs can be made based on whether a single metamodel valid over the entire search space or local metamodels, separately trained for each and every new individual, are used [46, 6, 21, 33, 12, 48].

MAEAs with off–line trained metamodels, almost always rely on a single global metamodel [13, 25]. The selection of the appropriate set of patterns for training a global metamodel is a complex task which, in the case of off–line trained metamodels, is usually carried out by sampling the search space via a design of experiments (DoE) technique, [45]. The most costly approach is that of full factorial designs. Given that the sample size grows exponentially with the number of design variables, fractional factorial designs (such as orthogonal arrays) can be used instead. Once the global metamodel has been trained, this is exclusively used during the EA–based search, [52, 22, 7, 54, 46, 6, 17], as shown in figure 1. The CPU cost of the EA is negligible, compared to that of evaluating the samples which are necessary for training the metamodel(s). The "optimal" solution, according to the metamodel, must undergo exact evaluation and, depending on the deviation between the outcomes of the two evaluation models (exact and surrogate), the optimization either terminates or restarts after retraining the metamodel on additional training patterns.

On the other hand, MAEAs with on–line trained metamodels, are based on the alternating use of the metamodels and the problem–specific model, during the evolution. Both tools are employed on the entire EA population, either periodically or by switching from metamodels to the exact tool [56, 25] depending on several criteria. Alternatively, only selected (i.e. promising) individuals in each generation, according to the results of a pre–evaluation phase exclusively based on the surrogate model (referred to as the **Inexact**

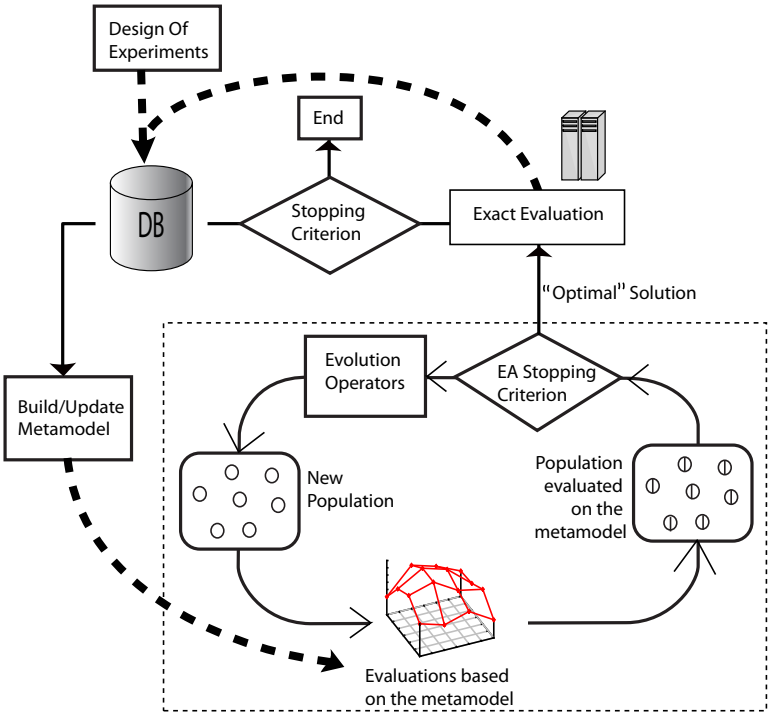


Figure 1: EAs assisted by an off-line trained global metamodel.

Pre–Evaluation or IPE phase, [34, 33]) [17, 47, 63, 5] may undergo re–evaluations on the exact model. With the exception of the few starting generations, which must necessarily be based exclusively on the exact model, all population members of the subsequent generations are approximately evaluated using local metamodels and only a few most promising among them are, then, re–evaluated on the problem–specific model, fig. 6. The implementation of the IPE technique within an EA is discussed in section 3.2. In some of the examples to be demonstrated, the IPE technique is adapted to **Distributed Evolutionary Algorithms** (DEAs), leading to the so–called distributed MAEAs (DMAEAs), [34]. DEAs handle a number of semi–autonomously evolving subpopulations and outperform single–population EAs. Similarly, DMAEAs outperform MAEAs.

The second part of this lecture is dedicated to **Hierarchical** or **Multilevel Evolutionary Algorithms** (HEAs) used in combination with MAEAs or DMAEAs (HDMAEAs), [35, 27, 28, 29, 20]. Hierarchical (or multilevel) optimization methods consist of a number (usually two or three) search levels. These levels can be associated with evaluation tools of different computational complexity (or accuracy) and CPU cost each and/or different sets of design variables and/or different search tools. Adjacent levels exchange promising solutions, according to various migration policies. The so–called **Memetic Algorithms** (MAs) can also be classified as hierarchical algorithms since they combine a global search method (usually an EA) with local search for the refinement of promising solutions, [49, 40]. Metamodel–assisted MAs (MAMAs) have also been proposed in the literature, [51, 50, 15, 16, 39].

Finally, **Asynchronous EAs** (AEAs), a class of **Parallel EAs** (PEAs) which is suitable for solving optimization probems on multiprocessor systems will be discussed. In synchronous EAs, the parallelization approach is based on the master–slave paradigm,

based on which candidate solutions are concurrently evaluated [43, 44]. This parallelization does not ensure maximum parallel efficiency due to the synchronization barrier at the end of each generation. For instance, some processors may remain idle for a certain period of time waiting for the processor performing the last pending evaluation in this generation to complete. This is why AEAs, [1], which can fully exploit all the available resources by removing the notion of "generation", have been devised.

2 Surrogate Evaluation Models

To set up an efficient MAEA, an important task is to select the appropriate metamodel. This decision should be made once the expectations from the metamodel are clearly defined. Depending on the MAEA, one may expect the metamodel to provide different pieces of information. For instance, a metamodel may approximate the fitness or cost-function or, in addition, give a measure of the confidence for this prediction or even to act as a classifier between feasible and infeasible solutions or between more and less promising ones, etc.

Before introducing the various metamodel types, some basic notations are provided. We are dealing with an optimization problem with N design variables and M objectives. $\boldsymbol{x} \in \mathcal{R}^N$ denotes a candidate solution and $\boldsymbol{F}(\boldsymbol{x}) \in \mathcal{R}^M$ the array of its objective function values. Of course, in SOO problems ($M=1$), this is a scalar quantity. Regarding metamodels, $\boldsymbol{x}^{(t)} = (x_1^{(t)}, \ldots, x_N^{(t)}) \in \mathcal{R}^N$, $t \in [1, T]$, is used to denote the T training patterns and $\boldsymbol{\zeta}^{(t)} = (\zeta_1^{(t)}, \ldots, \zeta_M^{(t)}) \in \mathcal{R}^M$ their known responses.

In section 2.1, a widely used metamodel type, namely the so–called Radial Basis Function (RBF) networks, will be described in detail. The RBF networks are used in most of the cases presented in this lecture. In section 2.2, some other metamodels will briefly be presented. For more information, the interested reader may turn to the cited books or papers. This lecture lays more emphasis to the way metamodels are used in the framework of EAs, rather than the mathematical formulation of training processes. Theoretically, the majority of the examples presented below can be reproduced by any surrogate evaluation model.

In all subsections of section 2, without loss in generality, it will be assumed that $M=1$ and $\zeta^{(t)} \equiv \zeta_1^{(t)}$.

2.1 Radial Basis Function (RBF) Networks

The structure of an RBF network, is shown in fig. 2. The network performs the mapping $\mathcal{R}^N \mapsto \mathcal{R}^M$, [55, 23], corresponding to that from the design variables to the objectives of the optimization problem. It consists of three layers of processing units, namely the input layer with N nodes where the input vectors $\boldsymbol{x}$ are applied to, the hidden layer with K processing neurons and the output layer with M nodes where the network response(s) emerge(s). Signals propagate through the network in the forward direction, by performing a nonlinear mapping from the input to the hidden layer, followed by a linear mapping to the output nodes. The K links connecting the hidden nodes to the output one are associated with the array of synaptic weights $\boldsymbol{w}$, to be computed during the training. The K hidden layer units are associated with the so–called RBF centers, $\boldsymbol{c}^{(k)} \in \mathcal{R}^N,\ k = 1, K$, which are the

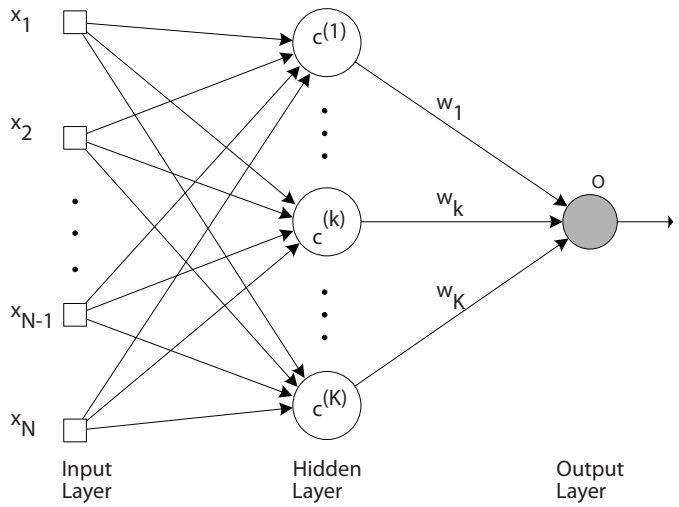


Figure 2: RBF network with N inputs, K hidden neurons and a single output ($M=1$).

centers of the neuron's nonlinear radial–basis activation function $\mathcal{G} : \mathcal{R}^N \mapsto \mathcal{R}$. Before setting–up an RBF network, the RBF centers must be selected. The activation function $\mathcal{G}(h)$ acts on the distance of the network's input $\boldsymbol{x}$ from the corresponding center $\boldsymbol{c}^{(k)}$,

$$h = \|\boldsymbol{x} - \boldsymbol{c}^{(k)}\|_2 \tag{1}$$

Several choices for $\mathcal{G}(h)$ are possible, such as the multiquadric function

$$\mathcal{G}(h) = \left(h^2+r^2\right)^{1/2} \tag{2}$$

the inverse multiquadric function

$$\mathcal{G}(h) = \left(h^2+r^2\right)^{-1/2} \tag{3}$$

or the Gaussian function

$$\mathcal{G}(h) = exp\left(-\frac{h^2}{r^2}\right) \tag{4}$$

etc., where r is the RBF *radius* or *width*. Depending on the training process, either a single r value for all centers or a different r value for each center can be used.

The network response $o(\boldsymbol{x})$ is computed as the weighted sum of the output signals from the hidden neurons,

$$o(\boldsymbol{x}) = \sum_{k=1}^{K} w_k \mathcal{G}(\|\boldsymbol{x} - \boldsymbol{c}^{(k)}\|_2) \tag{5}$$

For $K = T$, the choice of the RBF centers is straightforward, i.e. $\boldsymbol{c}^{(t)} \equiv \boldsymbol{x}^{(t)}$, $t \in [1, T]$, ensuring that the T samples are exactly interpolated. In this case, the network training requires the solution of a $T \times T$ symmetric linear system of equations, as follows,

$$\sum_{t=1}^{T} w_t \mathcal{G}(\|\boldsymbol{x}^{(t)} - \boldsymbol{c}^{(t)}\|_2) = \zeta^{(t)}$$

A interesting way to increase the network's generalization [55, 61, 62] is by using less, though appropriately selected, hidden nodes than the training patterns, i.e. selecting $K < T$. In such a case, the selection of the RBF centers is important, since it affects strongly the prediction ability of the network. In [33], a selection scheme for the RBF centers based on self–organizing maps (SOMs, [14, 23]) was proposed. This iterative scheme comprises two levels of learning, namely the unsupervised and the supervised ones. During the unsupervised learning, SOMs (through their standard processes: competition, cooperation and adaptation) classify the training patterns into K clusters. Each cluster gives a single RBF center $\boldsymbol{c}^{(k)}$, considered to be representative of this cluster (as schematically shown in fig. 3) and, based on heuristics based on distances between the centers, [36, 33, 23, 4], the corresponding radius r_k is computed. During the supervised learning, the synaptic weights are computed by minimizing the approximation error of the RBF network over the training set, while considering smoothness requirements.

To improve the prediction ability of the RBF network, the idea of enhancing it with the so–called *Importance Factors* (IFs) was proposed in [19]. Through extra coefficients $(I_n, n=1, N)$, the effect/importance of each design parameter on the network response is quantified and used. A high I_n value indicates high sensitivity of the objective function, close to the design point under consideration, with respect to the n–th input variable. The IFs are computed by the RBF network as a by–product of the training process and are used to improve the quality of the networks' output. In particular, each time an improved solution is computed by the EA (index b, which stands for the current best), the corresponding local RBF network is built and N partial derivatives $\partial o^{(b)}/\partial x_n$ are computed using the network's closed–form expressions; by doing so, $\partial o^{(b)}/\partial x_n$ are the exact derivatives of an approximate function. Based on these derivatives, a weighted norm is introduced and used instead of the standard one (i.e. instead of eq. 1), as follows

$$\left\|\boldsymbol{x}^{(t)} - \boldsymbol{c}^{(k)}\right\|_{wei} = \sqrt{\sum_{n=1}^{N} I_n \left(x_n^{(t)} - c_n^{(k)}\right)^2}, \text{ where } \quad I_n = \frac{\left|\frac{\partial o^{(b)}}{\partial x_n}\right|}{\sum_{i=1}^{N}\left|\frac{\partial o^{(b)}}{\partial x_i}\right|} \tag{6}$$

2.2 Other Surrogate Evaluation Models

Multilayer Perceptron Networks (MLP)

The multilayer perceptron performs a differential nonlinear mapping between the input and output space. In our case, this expresses the relation between the N design variables and M responses, namely $\mathcal{R}^N \rightarrow \mathcal{R}^M$. A conventional MLP, [23], is formed by the input layer with N sensory units, the output layer with M computational units and a user–defined number $(L-2)$ of intermediate layers, each of which may have any number K_l of hidden neurons or computational nodes, fig. 4. Each hidden layer unit is fully connected to all units lying in the previous and next layer. All connections (synapses) between units are associated with synaptic weights $w_{[j,l+1],[i,l]}$ where the subscripts denote their edge nodes: the arrival node j in the $(l+1)$–th layer and the departure node i in the l–th layer. MLP training is the process during which the synaptic weights take on appropriate values that guarantee the "best" mapping between the given samples $(\boldsymbol{x}^{(t)} \doteq (x_1^{(t)}, \ldots, x_N^{(t)}) \in \mathcal{R}^N,\ 1 \leq t \leq T)$ and their known responses $(\boldsymbol{\zeta}^{(t)} = (\zeta_1^{(t)}, \ldots, \zeta_M^{(t)}) \in \mathcal{R}^M)$.

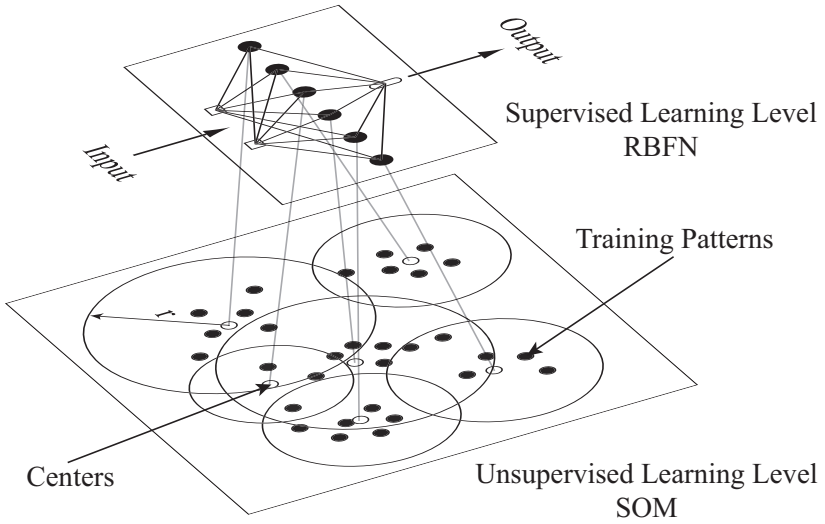


Figure 3: The two phases of an RBF network training based on SOMs: (a) self-organized positioning of the RBF centers (unsupervised learning) and (b) computation of the synaptic weights (supervised learning.

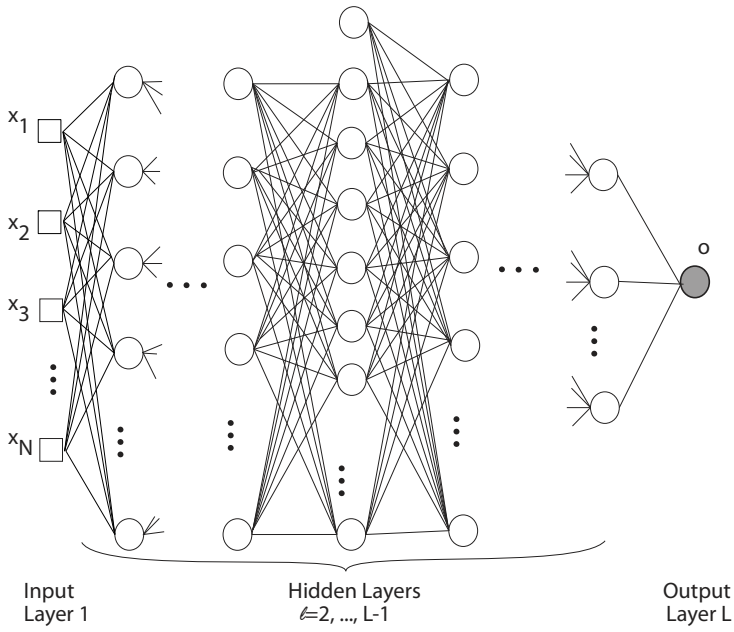


Figure 4: Multilayer Perceptron with L layers and a single node ($M = 1$) on the output layer.

Each training pattern $\boldsymbol{x}^{(t)}$ is presented to the sensory units of the input layer ($l=1$) and creates a signal that propagates towards the network output through the intermediate layers and, finally, reaches the last (L) layer node where the network response $o^{(t)} \doteq o(\boldsymbol{x}^{(t)})$ emerges. During the signal propagation, the input signal $h^{(t)}_{[q,l]}$ to the $q-th$ neuron in layer l is processed by the nonlinear neuron model $\mathcal{G}(h)$ and yields the neuron output $v^{(t)}_{[q,l]}=\mathcal{G}(h^{(t)}_{[q,l]})$. A widely–used activation function is the logistic one, defined by

$$\mathcal{G}(h) = \frac{1}{1+exp(-bh)} \ , \quad -\infty<h<\infty \ , \quad 0<\mathcal{G}(h)<1 \ , \quad B>1 \tag{7}$$

At any computational node (neuron q lying in layer l; marked as $[q,l]$), the output signal $v^{(t)}_{[q,l]}$ is given by

$$v^{(t)}_{[q,l]} = \mathcal{G}\left[\sum_{s=1}^{K_{l-1}} w_{[q,l],[s,l-1]} v^{(t)}_{[s,l-1]}\right] \tag{8}$$

expressing the output of the activation function on the weighted sum of all signals appearing at the outputs of the K_{l-1} previous layer neuron outputs.

The MLP networks are trained in a supervised manner through the error back-propagation algorithm (EBP) which is based on the error–correction learning rule. For the training, the T paired samples $(\boldsymbol{x}^{(t)}, \zeta^{(t)})$ are used. EBP learning is an iterative algorithm which consists of forward (function) and backward (error) signal passing through the MLP layers. In the forward pass, the input vector $\boldsymbol{x}^{(t)}$ of each training pattern is presented to the input layer and a signal propagates forward through the network. In the last layer, the network's prediction $o^{(t)}$ is compared to the known response $\zeta^{(t)}$ and the prediction error is computed. In the backward pass, the synaptic weights are corrected to minimize the prediction error.

Kriging

Kriging belongs to the class of Gaussian random field metamodels. Kriging, [41], may predict not only the response to a given input signal (such as a newly generated individual in an EA) but, also, a measure of confidence (*mean squared error*, MSE) for this prediction, fig. 5, [64];

As with the MLP or RBF networks, we assume that T training patterns $\boldsymbol{x}^{(t)} \in \mathcal{R}^N$, along with their responses $\boldsymbol{\zeta}^{(t)} \in \mathcal{R}^M$ are available ant that $M=1$. In kriging, for any $\boldsymbol{x} \in \mathcal{R}^N$, the corresponding response $o(\boldsymbol{x})$ is expressed by

$$o(\boldsymbol{x}) = \hat{\mu} + z\left(\boldsymbol{x}\right) \tag{9}$$

where $\hat{\mu}$ is the expected value and the covariance function $z\left(\boldsymbol{x}\right)$ is supposed to have zero mean and a user–defined correlation function $\mathcal{C}$ with any other point in $\mathcal{R}^N$. So, the model output function becomes a realization of a random field $\mathcal{F}$ that assigns a $1D$-Gaussian distributed random variable $\mathcal{F}(\boldsymbol{x})$ with constant mean $\hat{\mu}$ (the expected value of $\mathcal{F}$) and variance σ^2 to each point in the search space. For two points $\boldsymbol{x}^{(\kappa)}$ and $\boldsymbol{x}^{(\lambda)}$, a typical choice for the correlation function is

$$\mathcal{C}\left(\boldsymbol{x}^{(\kappa)}, \boldsymbol{x}^{(\lambda)}, \boldsymbol{\theta}\right) = \exp\left(-\sum_{n=1}^{N} \theta_n \left(x_n^{(\kappa)} - x_n^{(\lambda)}\right)^2\right) \tag{10}$$

where the correlation parameters $\boldsymbol{\theta} = (\theta_1, \theta_2, ..., \theta_{_N}) \in \mathcal{R}^N$ must be computed in order to fit the model to the available data. Quite often, an isotropic correlation kernel is assumed and eq. 10 is, then, rewritten with $\theta_n \equiv \theta$; in this case, a single θ value must be computed. Training the kriging model is the process of computing θ's, $\hat{\mu}$ and σ^2, which are all assumed to be invariant with respect to $\mathcal{F}$.

According to the maximum likelihood hypothesis, the probability density function of $\mathcal{F}(\boldsymbol{x}^{(t)})$ for each and every training pattern to be equal to its known response $\zeta^{(t)}$ must be maximized. This is mathematically expressed as

$$\max \left[\frac{1}{(2\pi)^{N/2}(\sigma^2)^{N/2}\sqrt{\det(\mathbf{C})}} \exp \left[\frac{(\boldsymbol{Z} - \mathbf{1}\hat{\mu})^T \mathbf{C}^{-1} (\boldsymbol{Z} - \mathbf{1}\hat{\mu})}{2\sigma^2} \right] \right] \tag{11}$$

where $\mathbf{1}$ is a column vector of length N with unit entries and

$$\mathbf{C} = \begin{bmatrix} \mathcal{C}\left(\boldsymbol{x}^{(1)}, \boldsymbol{x}^{(1)}, \boldsymbol{\theta}\right) & \cdots & \mathcal{C}\left(\boldsymbol{x}^{(1)}, \boldsymbol{x}^{(T)}, \boldsymbol{\theta}\right) \\ \vdots & \ddots & \vdots \\ \mathcal{C}\left(\boldsymbol{x}^{(T)}, \boldsymbol{x}^{(1)}, \boldsymbol{\theta}\right) & \cdots & \mathcal{C}\left(\boldsymbol{x}^{(T)}, \boldsymbol{x}^{(T)}, \boldsymbol{\theta}\right) \end{bmatrix}, \ \mathbf{1} = \begin{bmatrix} 1 \\ \vdots \\ 1 \end{bmatrix}, \ \boldsymbol{Z} = \begin{bmatrix} \zeta^{(1)} \\ \vdots \\ \zeta^{(T)} \end{bmatrix} \tag{12}$$

The problem of eq. 11 can be solved by adopting generalized least squares estimates for $\hat{\mu}$ and σ^2 , [38], as follows

$$\hat{\mu} = \frac{\mathbf{1}^T \mathbf{C}^{-1} \boldsymbol{Z}}{\mathbf{1}^T \mathbf{C}^{-1} \mathbf{1}} \tag{13}$$

$$\sigma^2 = \frac{1}{N} (\boldsymbol{Z} - \mathbf{1}\hat{\mu})^T \mathbf{C}^{-1} (\boldsymbol{Z} - \mathbf{1}\hat{\mu}) \tag{14}$$

The substitution of eqs. 13 and 14 into eq. 11, leads to the solution of an equivalent minimization problem,

$$\min \left[(2\pi)^{N/2} (\sigma^2)^{N/2} \sqrt{\det(\mathbf{C})} \exp(-\frac{N}{2}) \right]$$

or, finally,

$$\min \left[N \ln \left(\sigma^2(\boldsymbol{\theta}) \right) + \ln \left(\det \mathbf{C}(\boldsymbol{\theta}) \right) \right] \tag{15}$$

After computing the θ values by solving the nonlinear problem of eq. 15, the corresponding response and MSE predictions for any new $\boldsymbol{x}$ in the design space are, [11],

$$o(\boldsymbol{x}) = \hat{\mu} + (\boldsymbol{Z} - \mathbf{1}\hat{\mu})^T \mathbf{C}^{-1} \mathbf{c}(\boldsymbol{x}) \tag{16}$$

$$MSE(\boldsymbol{x}) = \sigma^2 \cdot (1 - \mathbf{c}(\boldsymbol{x})^T \mathbf{C}^{-1} \mathbf{c}(\boldsymbol{x})) \tag{17}$$

see fig. 5, where

$$\mathbf{c} = \left[\mathcal{C}\left(\boldsymbol{x}, \boldsymbol{x}^{(1)}, \boldsymbol{\theta}\right), \ldots, \mathcal{C}\left(\boldsymbol{x}, \boldsymbol{x}^{(T)}, \boldsymbol{\theta}\right) \right]^T \tag{18}$$

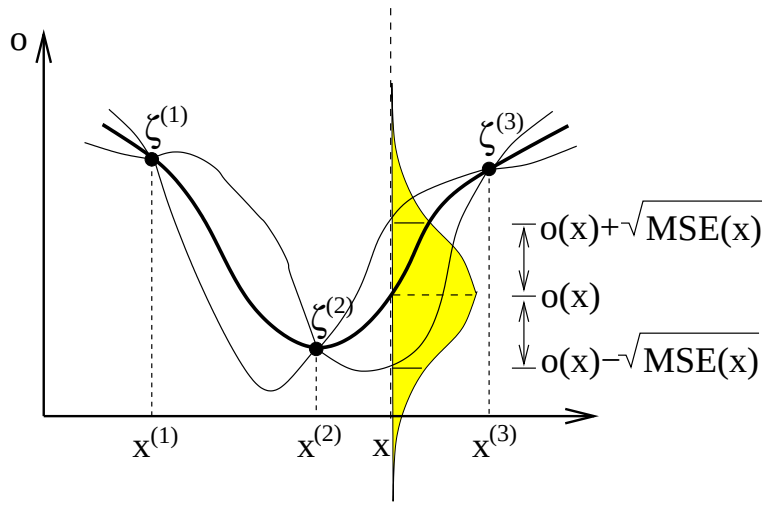


Figure 5: An 1D example of the use of kriging. Based on three data points $(x^{(1)}, \zeta^{(1)})$, $(x^{(2)}, \zeta^{(2)})$ and $(x^{(3)}, \zeta^{(3)})$, the response and the square root of the mean squared error for any other x value can be computed.

Polynomial–based regression surface methods (RSM)

A polynomial–based RSM utilizes polynomials of user–defined order to model the function landscape. Unknown coefficients are introduced and their value set that best fits the training data is sought. This set is computed by solving a linear least–squares problem, i.e. a much easier process than the non–linear regression required for all the neural networks other than the RBF ones. A polynomial–based RSM can be used to filter noise out from experimental data and their only disadvantage is that they can hardly interpolate data corresponding to complex functions.

A first–order (linear) RSM, with N independent (design) variables or regressors is

$$\boldsymbol{o}(\boldsymbol{x}) = \beta_0 + \sum_{n=1}^{N} \beta_n x_n + \epsilon \tag{19}$$

where $\beta_n,\ n = 0, \ldots, N$ are the unknown regressor coefficients. Similarly, a second–order RSM can be written as

$$\boldsymbol{o}(\boldsymbol{x}) = \beta_0 + \sum_{n=1}^{N} \beta_n x_n + \sum_{i=1}^{N} \beta_{ii} x_i^2 + \sum_{i=1}^{N} \sum_{j=i+1}^{N} \beta_{ij} x_i x_j + \epsilon \tag{20}$$

where, in both, ϵ is a random error. Either of them can be written in matrix form as

$$\boldsymbol{o}(\boldsymbol{x}) = \boldsymbol{X}\boldsymbol{\beta} + \boldsymbol{\epsilon} \tag{21}$$

The vector of least–squares estimators $\boldsymbol{\beta}$ that minimizes

$$\boldsymbol{\epsilon}^T \boldsymbol{\epsilon} = (\boldsymbol{o} - \boldsymbol{X}\boldsymbol{\beta})^T (\boldsymbol{o} - \boldsymbol{X}\boldsymbol{\beta}) \tag{22}$$

is sought; this leads to the least–squares normal equations and, though them, to

$$\boldsymbol{\beta} = (\boldsymbol{X}^T \boldsymbol{X})^{-1} \boldsymbol{X}^T \boldsymbol{o} \tag{23}$$

The prediction performance of the trained RSM can be quantified using various statistical measures. Further discussion on polynomial-based RSM is beyond the scope of this lecture.

3 MAEAs based on the IPE technique

This section describes the implementation of on–line trained metamodels, according to the IPE technique, on a generalized (μ, λ) EA, namely the EASY software, [18], which is the algorithm used for all the applications to be presented in this lecture. In particular, the basic algorithmic steps of the (μ, λ) EA are presented, along with the "changes" due to the implementation of IPE technique. Due to its importance, the selection of the appropriate training patterns for the RBF networks used as metamodels is discussed separately.

3.1 The Evolutionary Algorithm SYstem–EASY

The set of design variables is designated by $\boldsymbol{x} \in \mathcal{R}^N$ whereas $\boldsymbol{F}(\boldsymbol{x}) \in \mathcal{R}^M$ denotes the objective function. In MOO problems, for the approximation of the Pareto front of non–dominating solutions, a technique that maps $\boldsymbol{F}(\boldsymbol{x})$ into a scalar cost function value ϕ is additionally employed. For this purpose, SPEA[66], SPEA–2[65], NSGA[60], NSGA–2[8], or any variant of them may be used. At each generation g, the (μ, λ) EA maintains and updates three populations, namely the offspring $(\mathcal{P}_{\lambda,g})$ with λ individuals, the parent $(\mathcal{P}_{\mu,g})$ with μ individuals and the archival one $(\mathcal{P}_{\alpha})$ of elite individuals. Furthermore, during the evolution, all previously evaluated (on the problem–specific model) individuals are recorded in a database (DB).

Algorithm EASY (Evolutionary Algorithm SYstem or (μ, λ) EA).

EASY1. [Initialization] Set $g=0$ and randomly initialize the offspring population $\mathcal{P}_{\lambda,g}$.

EASY2. [Offspring evaluation] All $(\mathcal{P}_{\lambda,g})$ members are evaluated on the problem–specific evaluation model and $(\boldsymbol{x}, \boldsymbol{F}(\boldsymbol{x}))$ pairs are archived in the DB. In constrained optimization problems, $\boldsymbol{F}(\boldsymbol{x})$ is penalized if any of the constraints is violated. The penalty depends exponentially on the degree of the constraint violation.

EASY3. [Cost (ϕ) assignment] For $\mathcal{P}_g = \mathcal{P}_{\lambda,g} \cup \mathcal{P}_{\mu,g} \cup \mathcal{P}_{a,g}$ members, $\phi(\boldsymbol{x})$ is computed through

$$\phi(\boldsymbol{x}) = \phi(\boldsymbol{F}(\boldsymbol{x}), \{\boldsymbol{F}(\boldsymbol{z}) \mid \boldsymbol{z} \in \mathcal{P}_g \setminus \{\boldsymbol{x}\}\}) \in \mathcal{R}, \quad \boldsymbol{x} \in \mathcal{P}_g \tag{24}$$

Based on the $\phi(\boldsymbol{x})$ values the best offspring (current best for SOO and non–dominated for MOO) are singled out in the temporary set $\mathcal{P}_e$. Thinning algorithms can optionally be used to keep the $\mathcal{P}_e$ population size less than a user–defined maximum value. The most frequently used criterion is the distance between them (measured in the objective space), in order to give priority to individuals at the less populated parts of the current front of non–dominated individuals.

EASY4. [Elitism] The elite population is updated through the elitism operator E applied to the union of $\mathcal{P}_e$ and the previous elite archive $\mathcal{P}_{\alpha,g}$,

$$\mathcal{P}_{a,g+1} = E(\mathcal{P}_e \cup \mathcal{P}_{\alpha,g})$$

EASY5. [Evolution] New parents are selected using the selection operator S,

$$\mathcal{P}_{\mu,g+1} = S(\mathcal{P}_{\mu,g} \cup \mathcal{P}_{\lambda,g})$$

and new offspring are formed through the application of the crossover, C, and mutation, M, operators,

$$\mathcal{P}_{\lambda,g+1} = M(C(\mathcal{P}_{\mu,g+1}, \mathcal{P}_{\alpha,g+1}))$$

EASY6. [Termination] If the maximum number of exact evaluations is reached or the elite population does not change during the last generations, the algorithm terminates. Otherwise, $g \leftarrow g+1$ and the algorithm continues from step EASY2. ∎

In distributed EAs (DEAs), the above algorithm is valid for each one of the semi–autonomously evolving sub–populations (the so–called *demes* of *islands*). The demes communicate according to a user-defined topology and exchange their best performing and, sometimes, a few randomly–selected individuals according to an inter–deme migration operator applied with a user–defined frequency.

3.2 Use of the Inexact Pre–Evaluation within the (μ, λ) EA

IPE is a low–cost screening technique applied within each generation, [21, 33]. In specific, the entire population is pre–evaluated on locally trained metamodels and only a few top individuals among them are, then, re–evaluated on the problem–specific model (see fig. 6).

The algorithm starts as a conventional (μ, λ) EA (by, exclusively, making use of the exact evaluation model) for the few starting generations until a user–defined minimum number of exact evaluations enter the DB. Then, the IPE phase starts and, in subsequent generations, instead of evaluating the offspring population on the costly, problem–specific model (as in step EASY2), the following actions are taken:

Algorithm IPE (Inexact Pre-Evaluation).

IPE1. [Inexact evaluation] For each offspring, $\boldsymbol{x} \in \mathcal{P}_{\lambda,g}$, the objective function values $\tilde{\boldsymbol{F}}(\boldsymbol{x})$, are approximated using a local metamodel trained on a small number of data selected from the DB. The selection of the training set is described in section 3.3.

IPE2. [Screening] Based on the $\tilde{\boldsymbol{F}}(\boldsymbol{x})$ values, a provisional ϕ value (denoted by $\tilde{\phi}$) is assigned to each offspring

$$\tilde{\phi}(\boldsymbol{x}) = \tilde{\phi}(\tilde{\boldsymbol{F}}(\boldsymbol{x}), \{\tilde{\boldsymbol{F}}(\boldsymbol{z}) \mid \boldsymbol{z} \in \mathcal{P}_{\lambda,g} \setminus \{\boldsymbol{x}\}\}) \in \mathcal{R}, \quad \boldsymbol{x} \in \mathcal{P}_{\lambda,g}. \tag{25}$$

In SOO problems, $\tilde{\phi}(\boldsymbol{x}) \equiv \tilde{\boldsymbol{F}}(\boldsymbol{x})$. In MOO problems, the assignment method of step EASY3 can be used, i.e. by using $\boldsymbol{F}(\boldsymbol{x})$ instead of $\tilde{\boldsymbol{F}}(\boldsymbol{x})$. However, since $\tilde{\boldsymbol{F}}(\boldsymbol{x})$ is just an approximation to $\boldsymbol{F}(\boldsymbol{x})$ and due to the large number of comparisons required by any scalar cost assignment technique (SPEA, NSGA, etc), it is recommended to compute $\tilde{\phi}$ using the simplest possible scheme. Irrespective of the scheme used in step EASY3, a simple ranking in non–dominated fronts (without sharing) often suffices. Based on the so–computed $\tilde{\phi}$ values, the best $\lambda_e < \lambda$ ($\lambda_e = s\lambda$, where $s \ll 1$ is user–defined) offspring are singled out in the temporary set $\mathcal{P}_e$,

$$\mathcal{P}_e \triangleq \{\boldsymbol{x}_i,\ i = 1, 2, \ldots, \lambda_e : \tilde{\phi}(\boldsymbol{x}_i) < \tilde{\phi}(\boldsymbol{z}),\ \boldsymbol{z} \in \mathcal{P}_{\lambda,g} \setminus \mathcal{P}_e\}.$$

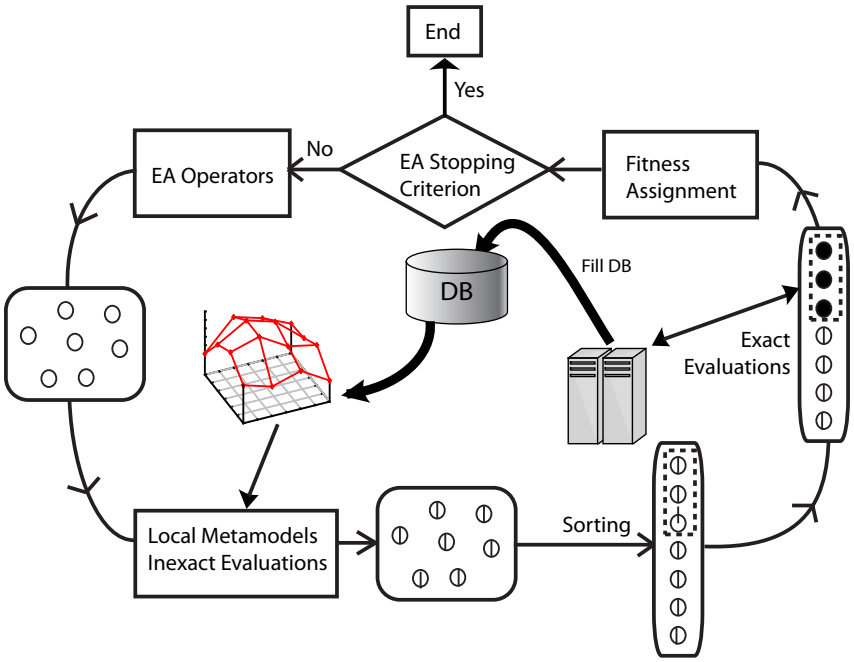


Figure 6: The IPE algorithm based on on–line trained local metamodels used to filter the offspring population at each generation, so that only a small fraction of it to be re–evaluated on the costly problem–specific model.

IPE3. [Exact evaluation] For all $\boldsymbol{x} \in \mathcal{P}_e$, the exact objective function values $\boldsymbol{F}(\boldsymbol{x})$ are computed and stored in the DB. Practically, this step, determines the CPU cost of each generation. ∎

During IPE, in step EASY3, eq. 24 is applied by using either the exact ($\boldsymbol{F}(\boldsymbol{x})$) or the approximate ($\tilde{\boldsymbol{F}}(\boldsymbol{x})$) objective function values, depending on whether each individual has been evaluated on the exact model or only the metamodel.

In the IPE phase, various metamodels, either global (periodically retrained or updated, [26]) or local (constructed on the fly for each and every individual, using the most appropriate subset of the available data, [34]) can be used. However, practice has shown [21, 19] that local metamodels generally provide better predictions, especially in cases with complex objective function landscapes.

The implementation of the IPE phase within a DEA is straightforward, since the same evaluation model is used within all sub–populations and a *common* DB of previously evaluated individuals is maintained. This means that the patterns used for training a local metamodel for any individual belonging to any deme may have been evaluated by any other deme.

3.3 Selection of training patterns for local metamodels

An important issue that affects the prediction ability of local metamodels is the selection of the most appropriate training patterns, from the DB of previously evaluated individuals. For each individual $\boldsymbol{x} \in \mathcal{R}^N$ to be pre–evaluated on the metamodel, the corresponding training pattern set $\mathcal{T}$ must "optimally" be populated. The number T of training patterns may vary between user–defined lower (T_{min}) and upper (T_{max}) bounds.

An easy way to populate $\mathcal{T}$ is by selecting T DB entries lying in the vicinity of $\boldsymbol{x}$, based on distances measured in the non–dimensional design space. In MOO problems, populating $\mathcal{T}$ is much more demanding. When seeking the Pareto front, the local metamodels should provide accurate predictions along a front of solutions and not only around a single point which is likely close to the sought optimal solution. Furthermore, in MOO problems, the identification and special treatment of *outliers*, i.e. new individuals that are far apart from most of the DB entries, fig. 7, is very important. For the outliers, since it is difficult to form an adequate training set, the problem–specific model must be used. An algorithm to identify outliers is required.

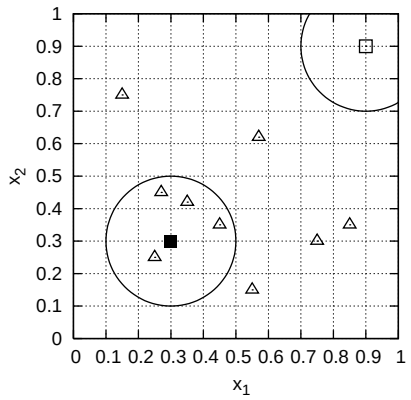


Figure 7: Example of a new individual (filled square) for the pre–evaluation of which a dependable training set (with four training patterns located within a circle of user–defined radius) can be found. This figure also shows an outlier (empty square) that must be identified and, then, evaluated on the problem–specific model. It is assumed that $\boldsymbol{x} \in \mathcal{R}^2$ and that the two design variables x_1 and x_2 are nondimensionalized.

Below, a well performing (in either SOO or MOO) algorithm, is presented, [33]. For each new individual $\boldsymbol{x}$, for which a metamodel is to be built, its closest neighbors in the design space are retrieved from the DB to form a temporary pool ($\mathcal{T}_{pool}$). For the gathered points, including $\boldsymbol{x}$, the corresponding minimum length spanning tree (MST, [37]) is generated. The MST connects all these points with a line of minimum total length. Then, each MST branch is traversed and, depending on a number of criteria, a point may be cut off. Points which finally remain on the tree form the training set. The basic steps of this algorithm are described below:

Algorithm TPS (Training pattern selection).

TPS1. [Pool formation] An initial pool $\mathcal{T}_{pool}$ of DB members which are close to $\boldsymbol{x}$ is formed. As rule of the thumb, the size of $\mathcal{T}_{pool}$ must be 3 to 4 times larger than T.

TPS2. [Pool Thinning] The MST for the $\mathcal{T}_{pool}$ members is created. MST branches with length below a user–defined value are consolidated by eliminating the ending point with the minimum distance to any other individual within the pool. When a branch is eliminated, the MST is updated and the process is repeated until all branches satisfy the minimum length criterion.

TPS3. [Treatment of outliers] From the data points left on the MST, the T_{min} patterns which are closer to the new individual are selected. If the closest pattern to $\boldsymbol{x}$ is at distance greater than a user–defined trust region radius, $\boldsymbol{x}$ is marked as *outlier*. This radius is either equal to the MST average branch length plus a tolerance expressed in terms of the corresponding standard deviation or proportional to the design space size.

TPS4. [Complementary data] Depending on the structure of the closest data points, more data points may enter $\mathcal{T}$, without however exceeding the upper bound of T_{max} patterns. ∎

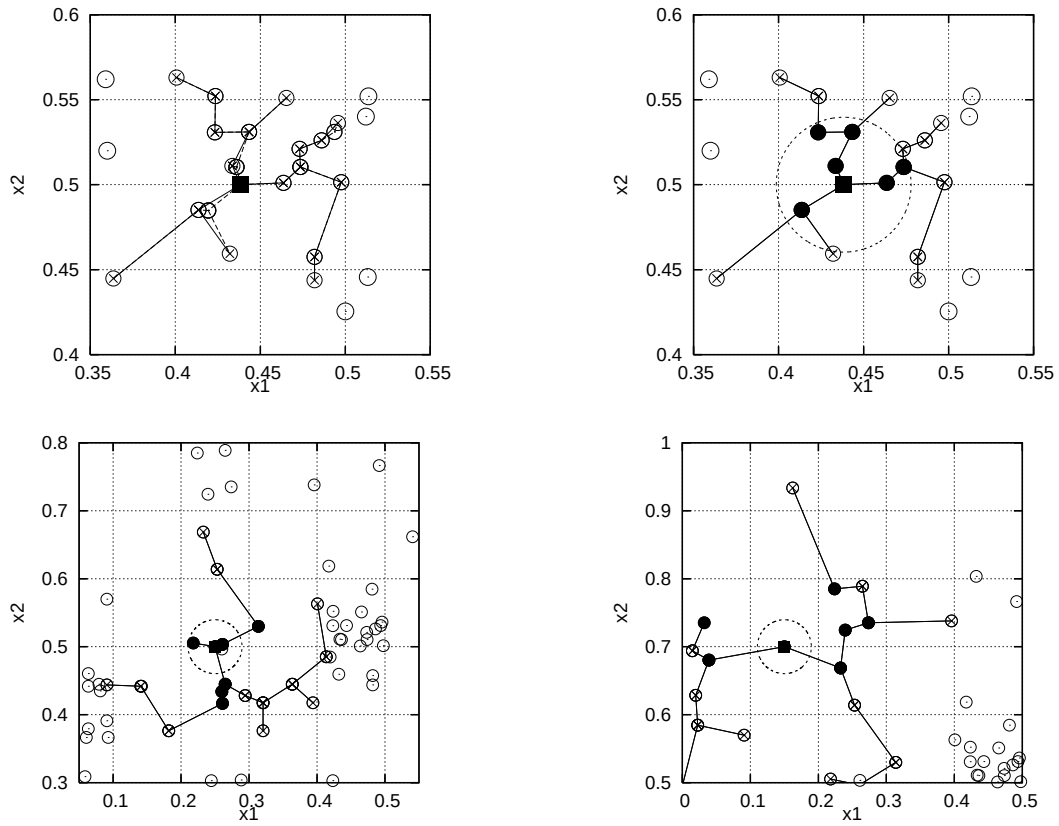


Figure 8: Indicative cases of training pattern selection in a problem with two design variables (x_1, x_2), according to the *TPS* algorithm. It is assumed that $\boldsymbol{x} \in \mathcal{R}^2$.

Fig. 8 presents snapshots of the use of the *TPS* algorithm in a problem with two design variables. The new individual $\boldsymbol{x}$ is denoted by a black square and the DB entries with empty circles. Points marked with a cross designate the closest neighborhood to the individual, whereas the final training set $\mathcal{T}$ is shown with filled circles. The MST drawn is designated by solid lines. The first case (fig. 8, top) is expected to give a dependable training set. After creating the MST (top–left), the thinning process leads to a reduced MST (top–right). A circle of user–defined radius serves to finally select the training patterns (top–right). Fig. 8 (bottom–left) shows an individual failing to form a dependable training set, since most of the available data points fall outside the

circle. Thus, any metamodel trained on these few patterns is expected to provide quite inaccurate guesses. An outlying individual is shown at fig. 8 (bottom–right).

3.4 Application of MAEAs

Below, four aero/hydrodynamic optimization problems, either SOO or MOO, are presented. Through these problems, it is convincingly shown that the use of surrogate evaluation models to support an EA, based on the IPE technique (i.e. a MAEA) constantly outperforms conventional EAs. In addition, the last two cases demonstrate that one can further improve the performance of EAs and MAEAs using the *principal component analysis* technique.

☛ The first case is dealing with the optimal deployment of a four–element airfoil for maximum lift coefficient (C_L) at $M_\infty = 0.2$, $a_\infty = 10^o$ and an inviscid fluid. The design variables stand for the relative positions of the slat and the two flaps with respect to the main body of the airfoil (fig. 9). In particular, for each but the main element, the design variables are its two displacements (horizontal and vertical) along with a rotation angle, summing up to $3 \times 3 = 9$ design variables, in total. The evaluation software is the GPU implementation of an in–house Euler equations solver [30, 3], for compressible flows that employs a time–marching, vertex–centered, finite volume scheme on unstructured grids. For this problem, the mixed precision arithmetic variant GPU_{MP} which uses single precision arithmetic for the l.h.s. coefficient matrices and double precision arithmetic for the residuals, was used; compared to its CPU counterpart this code is faster by about $90\times$. The optimization was carried out using both a $(20, 50)$ EA and a $(20, 50)$ MAEA in which $\lambda_e = 6$ out of the $\lambda = 50$ pre–evaluated offspring were re–evaluated on the Euler solver. The stopping criterion was set at 1000 evaluations on the GPU_{MP} code. An NVIDIA GPU cluster was used. In the MAEA, the IPE phase began after the first 100 candidate solutions were stored in the DB. Comparison of the convergence histories of EA and MAEA runs is presented in fig. 10. From this figure, it is concluded that the MAEA is about three times faster than the EA. The Mach field on the optimal solution computed with MAEA is shown in fig. 11.

☛ The second case is concerned with the shape optimization of a compressor cascade for minimal total pressure losses, while maintaining an acceptable performance level at off-design flow conditions. The two objectives are

$$\min F_1 = \omega_{\alpha_1}^{(OP_1)} = \frac{p_{t1} - p_{t2}}{\frac{1}{2}\rho_1 V_1^2}$$
$$\min F_2 = 10^4 \cdot \{(\omega_{\alpha_1}^{(OP_2)} - \omega_{\alpha_1}^{(OP_1)})^2 + (\omega_{\alpha_1}^{(OP_3)} - \omega_{\alpha_1}^{(OP_1)})^2\}$$

where OP_1 denotes the design operating point with $\alpha_1^{(OP_1)} = 47^o$ and OP_2, OP_3 are two off–design points with $\alpha_1^{(OP_2)} = 43^o$ and $\alpha_1^{(OP_3)} = 51^o$ respectively. In all operating points $M_1 = 0.62$ and $Re = 8.41 \cdot 10^5$. The evaluation model used for this case is an integral boundary layer software $(CFD - IBL)$[1]. The design variables are the coordinates of the NURBS control points parameterizing the airfoil. Geometrical constraints on the minimum airfoil thickness, the radius of curvature at the leading edge and the angles

[1]Drela's MISES analysis software, [10], based on an integral boundary layer method coupled with an external inviscid flow solver.

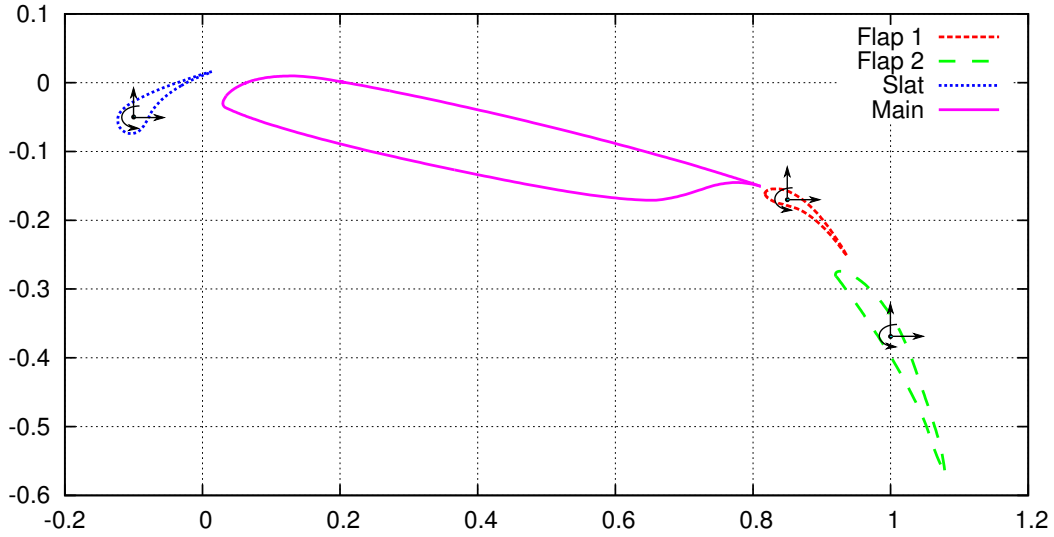


Figure 9: Optimal deployment of a four–element airfoil for maximum lift coefficient. Definition of the design variables (displacements and rotation) that define the relative positions of the slat and the two flaps with respect to the main body.

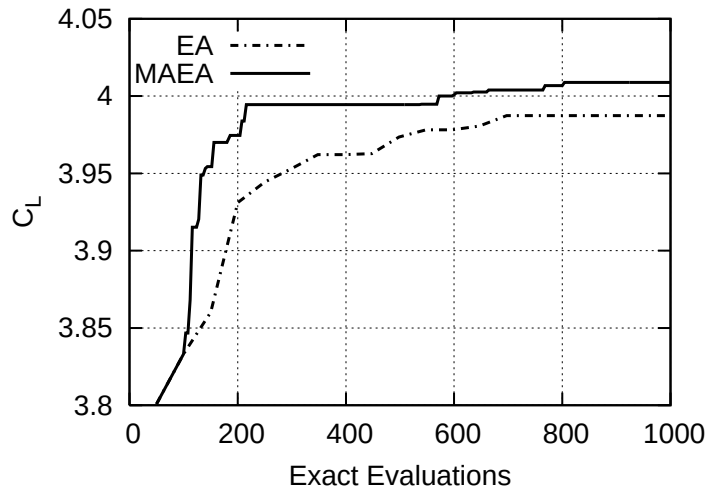


Figure 10: Optimal deployment of a four–element airfoil for maximum lift coefficient. The convergence history of the SOO problem is plotted in terms of the number of exact evaluations, i.e. the number of Euler computations which is proportional to the CPU cost.

between the first pressure/suction side control points and the leading edge were imposed. The flow exit angle must be kept between 19^o and 21^o and this was imposed as constraint.

In this case, some difficulties arose due to the imposition of severe constraints and the inability of the evaluation tool to cope with massively separated flows, which are likely to occur at off–design conditions (at least for several intermediate airfoil shapes evaluated during the evolution). These difficulties resulted in a high number of failing evaluations; indeed, approximately 350 out of the 2000 evaluations failed. Nevertheless, the IPE phase proved to be quite robust and reduced the computational effort required for generating a front of given quality, fig. 12. In fig. 13, the front obtained from the MAEA is compared

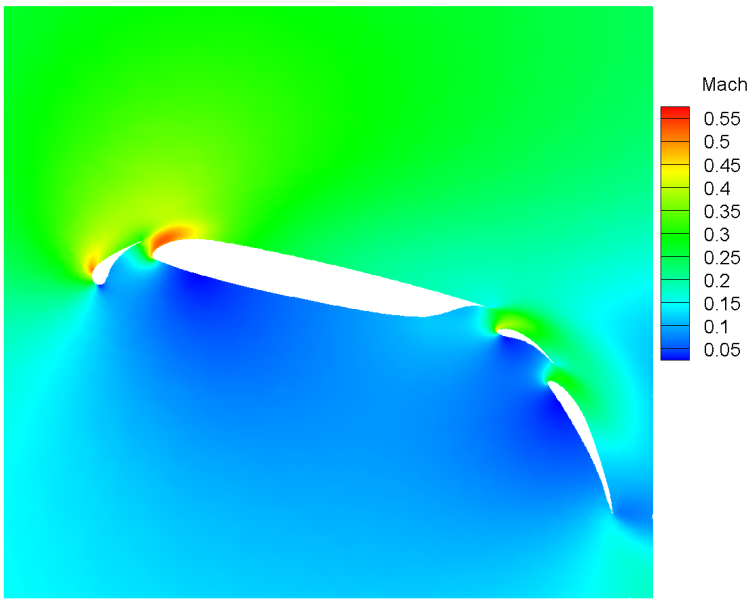


Figure 11: Optimal deployment of a four–element airfoil for maximum lift coefficient. Mach field for the optimal solution computed with MAEA.

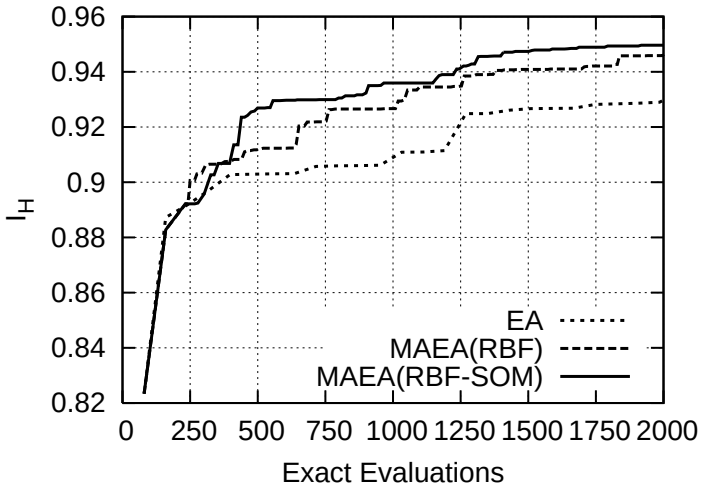


Figure 12: Shape optimization of a compressor cascade airfoil. Convergence history of the *hypervolume indicator* I_H (a metric of the quality of the computed front; greater metric value denotes a "better" front of non–dominating solutions). This case demonstrates that, in MOO problem, the use of SOMs for the selection of RBF centers during training (as discussed in section 2.1) performs better than the conventional RBF networks trained on just the immediate neighbors. From [32].

with that achieved by a conventional EA; also, three representative solutions belonging

to the front are shown.

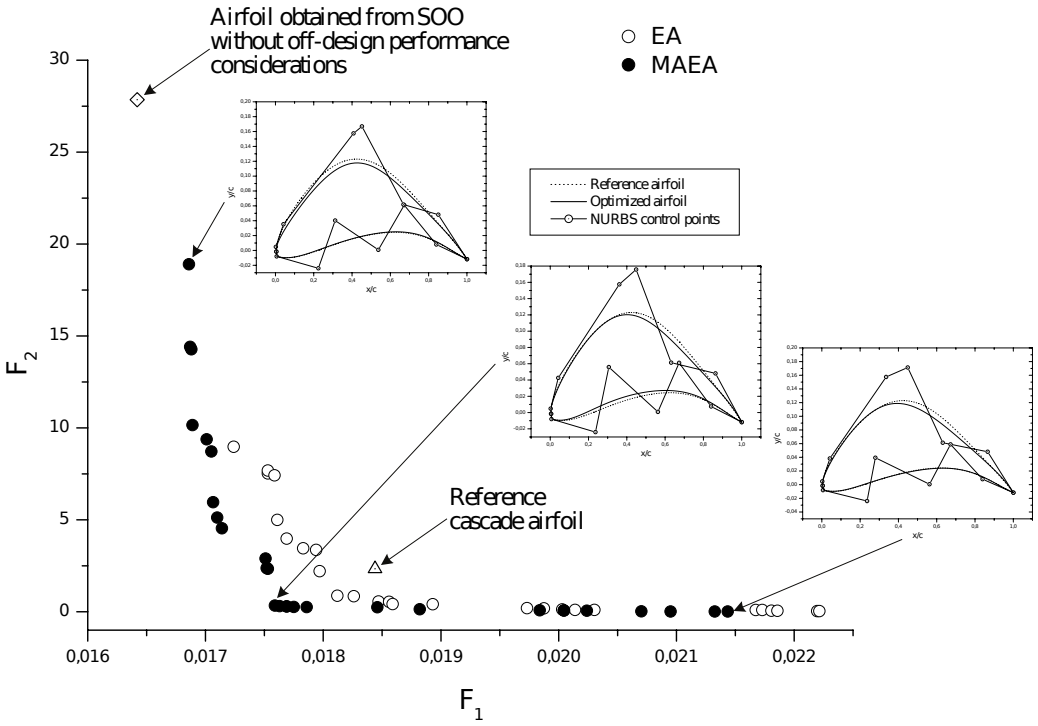


Figure 13: Shape optimization of a compressor cascade airfoil. Non–dominated fronts after 2000 exact evaluations using EA and MAEA (EA with IPE), along with a couple of selected optimal cascade airfoils obtained by the MAEA. From [32]. The front computed using the MAEA practically dominates all the front members computed by the conventional EA at the same CPU cost.

☛ The third case is concerned with the shape optimization of a (different) compressor cascade airfoil for minimum total pressure losses ω. The cascade operates at the following flow conditions: $M_1 = 0.54$, $a_1 = 44^o$, $Re = 4 \cdot 10^5$ and has an axial–velocity–density–ratio equal to 1.12173. The stagger angle is fixed and equal to 20^o. The airfoil mean curvature line was parameterized using three design variables and the thickness distribution was then parameterized using 9 design varables for the pressure and 15 for the suction side. The total number of design variables sums up to 27. Geometrical constraints on the airfoil thickness at three chord–wise positions ($t^{0.4c} \geq 0.1$, $t^{0.6c} \geq 0.08$ and $t^{0.9c} \geq 0.01$) and an additional constraint on the flow turning (to be greater than 30^o) were imposed. The evaluation tool is the same $CFD - IBL$ software.

This SOO case was studied not only for demonstrating, once more, that a MAEA outperforms the standard EA but, also, for proceeding beyond the MAEAs described thus far. In this respect, a new enhanced variant of MAEAs, which additionaly employs the *principal components analysis* (PCA) of the design variables' vector, was devised and used in this problem. Thus far, PCA has found application in pattern recognition or image compression techniques, for handling data sets of high dimension [23]. PCA is the process of transforming a data set in such a way that the transformed data set can be represented by a reduced number of "effective" features, without damaging the "useful" contents of the original data set. Practically, PCA is nothing but a "smart" coordinate

transformation of points in the data space to points in the so–called feature space. The rotated vector corresponding to a point in the feature space can optionally be truncated; it can be proved that the rotation ensures maximum rate of decrease in variance; so, this truncation is capable of maintaining the maximum of the information contained into the original data set.

One way of using the PCA during an EA– or MAEA–based optimization is for improving the performance of the evolution operators. An EA-based method performs better if applied onto functions with *separable* design variables. By definition, a function of separable variables can be written in components, each of which depends on a single design variable; thus, in a problem with N optimization variables, the minimization of the objective function is equivalent to the minimization of N $1-D$ functions. Therefore, the minimization of functions with separable variables is not subject to the curse of dimensionality. Using PCA, the design variable space is rotated, so as to make the design variables of the objective function under consideration as separable as possible. In different words, this could be understood as the process of rotating the axes of the design space so as to come up with a new set of the "most important" directions; no truncation takes place. This can be done for all the population members just before employing the evolution operators. After the application of the evolution operators, the opposite rotation is necessary so as to return the new offspring population members in the standard coordinate system (according to the selected parameterization). The new variant will be denoted by EA(PCA) or MAEA(PCA), depending on whether metamodels are also used.

Based on the above, this problem was solved twice, using MAEA and MAEA(PCA). The reason is to demonstrate that the MAEA's performance can further be improved using the PCA technique. The comparison is shown in fig. 14. MAEA(PCA) is much faster than MAEA.

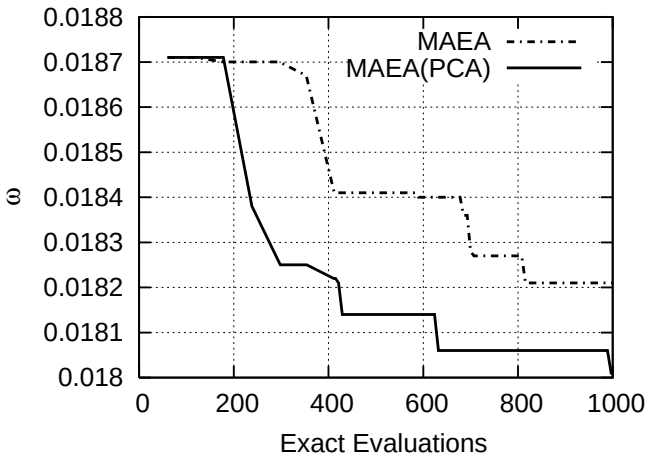


Figure 14: Shape optimization of a compressor cascade airfoil for minimum total pressure losses at the nominal operating point. Comparison of the convergence history of MAEA with and without PCA.

☛ The last case is dealing with the design–optimization of a Hydromatrix®[2] turbine runner at three operating points (part load, full load and best efficiency point). The

[2]Hydromatrix® is a new innovative concept of low–head hydraulic energy generation developed by Andritz HYDRO. It can be easily integrated into existing dam structures or weirs, combining the

design was performed by designers/engineers of Andritz HYDRO at Linz, Austria using the optimization software EASY. The runner has 4 blades and each runner blade is parameterized via superimposing a thickness distribution onto a mean camber surface, fig. 15. The blade thickness distribution is chosen based on the expected structural stresses exerted to the blade and is fixed during the optimization. Therefore, the runner blade shape is controlled by the parameters defining its mean camber surface. The design variables are the coordinates of the control points of the Bezier curves used to parameterize the spanwise distributions of (a) the mean camber surface angles at the leading (LE) and trailing (TE) edges, (b) the circumferential position of the blade LE and TE and (c) the mean camber surface curvature. Overall, there are 52 design variables and the same MAEA assisted by PCA was used.

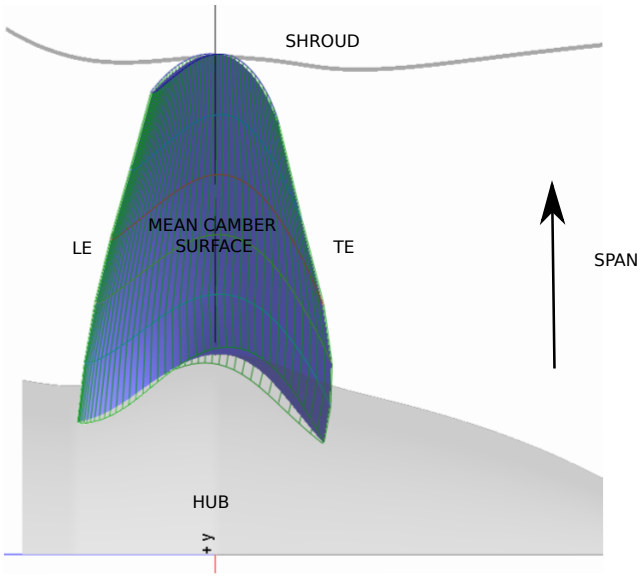


Figure 15: Parameterization of a Hydromatrix® runner blade. Eleven blade profiles in the spanwise direction generated on the mean camber surface using known thickness distributions and the surface grid of the blade are shown.

For each operating point, three metrics are used to measure the runner flow quality. The first metric (f_1) quantifies how close the velocity profile at the runner outlet is to the recommended one at the existing draft tube inlet. Given that the draft tube is fixed, known swirl and axial velocity profiles at the exit of the runner are required; from this point of view, metric f_1 becomes the objective function of an inverse design problem in which the target velocity components' distribution is defined at the runner exit. The second metric (f_2) is related to the blade loading which must be uniform (i.e. with minimum variations) over the blade surface. The third metric (f_3) quantifies the cavitation danger for the blade based on a safety margin defined by the designer. This

advantages of low cost installation and minimal environmental impact since no new civil structures are needed. The Hydromatrix® system utilizes a factory assembled "matrix" or module of small (with diameter ranging from 0.9 to 1.3 meter), axial flow, fixed blade ("unregulated") turbine generator units instead of a larger conventional "regulated" one. The discharge regulation of a Hydromatrix® power plant is achieved by taking single units in or out of operation depending on the need to either increase or decrease the per unit discharge. www.andritz.com

case was studied as a two–objective constrained optimization problem. The weighted sum (for all the three operating points) of metrics f_1, f_2 were used as objective functions

$$F_1 = \sum_{i=1}^{3} w^{OP_i} f_1^{OP_i} \quad , \quad F_2 = \sum_{i=1}^{3} w^{OP_i} f_2^{OP_i}$$

with $w^{OP_1} = 1.0$, $w^{OP_2} = 0.1$, $w^{OP_3} = 0.1$, while f_3 was imposed as a constraint.

Optimization runs were carried out using a $(30, 90)$ MAEA with $\lambda_e = 18$. The IPE phase was activated after the first 300 non–failed evaluations were stored in the DB. In the case of MAEA assisted by PCA, the PCA–assisted evolution operators were applied after the 5^{th} generation. The resulting fronts of non–dominated solutions are compared in fig. 16 (left) where it is obvious that the use of PCA is very important in such a high–dimensional problem. A selected geometry (point A) from the front resulted from the MAEA(PCA) is shown in fig. 16 (right). For point A, the whole Hydromatrix® geometry is shown, including both the runner that was optimized (with its 4 blades) and the stator which was known beforehand.

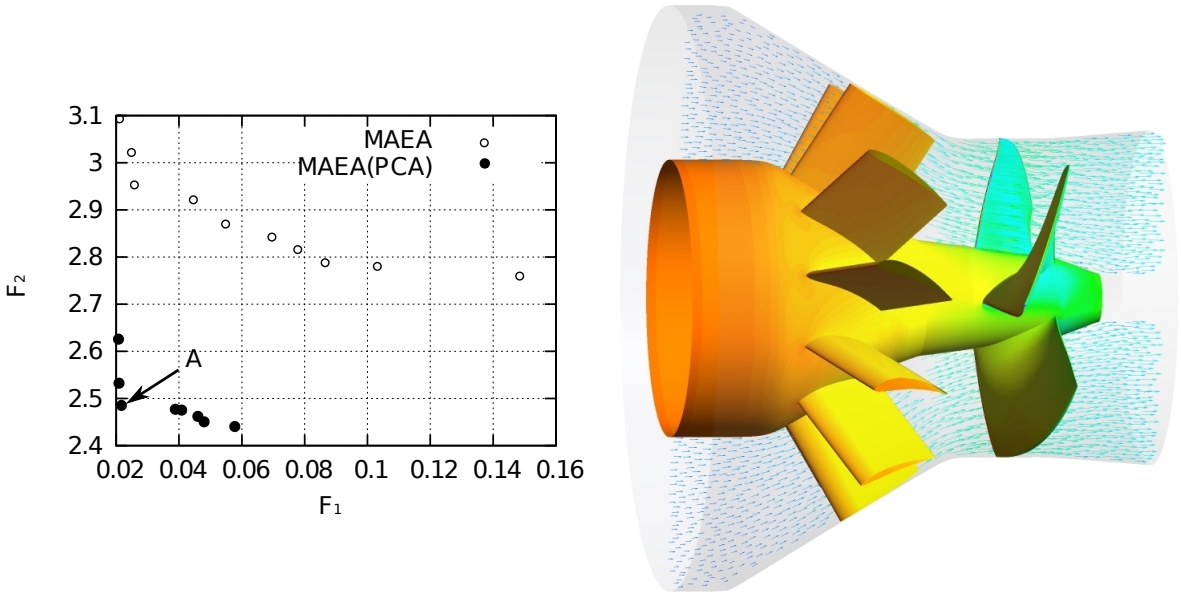


Figure 16: Two–objective optimization of a Hydromatrix® runner. Comparison of the front of non–dominated solutions obtained with MAEA and MAEA(PCA) (left). The pressure field is plotted over the geometry corresponding to point A (right). For A, the runner and the stator (which remained constant during the optimization) are shown. From [42].

4 Hierarchical EAs and MAEAs

As mentioned in the introduction, another way to reduce the wall clock time of an optimization problem is by means of hierarchical schemes. This section presents various concepts of hierarchical structures and applications of HEAs and HMAEAs.

4.1 HEAs: Basic Concepts

The hierarchical structure requires the definition of a number of semi–autonomously "evolving" levels. In this multilevel structure, each level can be associated with different evaluation tools, search techniques and/or problem parameterizations, [24, 57, 9, 35, 31, 27, 28, 67, 20]. Adjacent levels exchange their best individuals using one– or two–way inter–level communications. Depending on the type and timing of inter–level communications, the terms "hierarhical" or "multilevel" optimization might or might not mean the same. The most frequenlty used algorithms are those based on two levels only.

Hierarchical Evaluation Scheme

In the *hierarchical evaluation scheme* a different evaluation tool/software is assigned to each level. By convention, the lower level undertakes the exploration of the design space, i.e. the detection of near–optimal solutions at low CPU cost before delivering them to its higher level for further refinement and so forth. Thus, inexpensive, low–fidelity evaluation models are usually associated with the lower levels. On the higher level(s), which are exploitation oriented and the uppermost of them is responsible for delivering the optimal solution(s), evaluation models of higher fidelity and CPU cost are employed. A two–way inter–level communication can optionally be used allowing (apart from the migration of promising individuals upwards for them to undergo refinement) other individuals to move downwards to stimulate a more exhaustive search in their neighborhood, based on lower cost evaluation tools.

Hierarchical Search Scheme

In the *hierarchical search scheme*, each level is associated with a different search technique. Any combination of stochastic/heuristic and gradient–based methods can be used. Stochastic search techniques, such as EAs (DEAs, DMAEAs, etc.), are preferably used on the low level for the exploration of the design space. Promising solutions migrate to the high level, which is usually (but not necessarily) associated with a gradient–based method for their refinement. The migration of promising solutions is, preferably, bi–directional so as to accentuate the exploration capabilities of low–level EAs.

Hierarchical Parameterization Scheme

The *hierarchical parameterization scheme* associates coarse and fine problem definitions with each level. The coarser definition (reduced number of design variables and/or relaxed constraints) corresponds to the low level, where rough designs are quickly detected due to the small dimensionality of the problem. The hierarchical or multilevel parameterization scheme must be configured in a way that makes all but the highest level not affected (or, at least, slightly affected) by the curse of dimensionality. During the inter–level migration steps, a few promising solutions detected on the low level(s) are forwarded to the higher level(s) for refinement. This hierarchical scheme is suitable for shape optimization problems in which the design vector is formed by the coordinates of the control points of the parametric curves and surfaces since, by increasing or decreasing the number of

control points via knot insertion and removal algorithms [53] allows exact (upwards) or approximate (downwards) transformations of the design vectors.

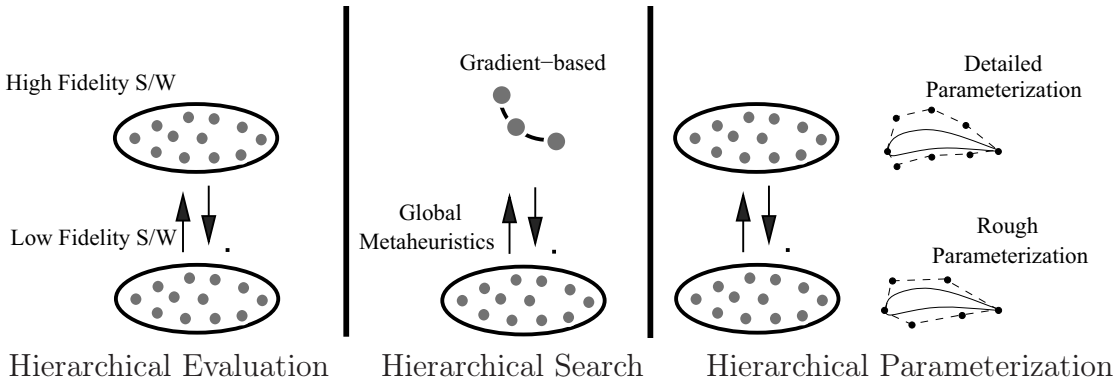


Figure 17: HEAs. Schematic representation of the three hierarchical schemes.

Metamodels are automatically incorporated into hierarchical schemes, by just using MAEAs in lieu of EAs. This gives rise to the so–called hierarchical MAEAs (HMAEAs) in which, in contrast to single level MAEAs, different DBs of previously evaluated solutions must be kept for the purpose of training the corresponding metamodels on each level. Each level may optionally apply a DMAEA leading to a Hierarhical DMAEA (HDMAEA). An alternative way is to apply the hierarchy within each deme which gives rise to a Distributed HMAEA (DHMAEA), fig. 18.

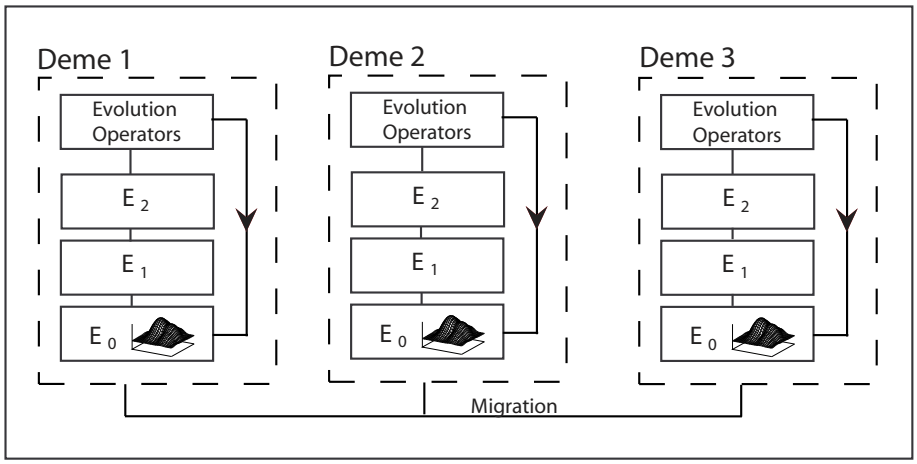


Figure 18: Distributed hierarchical structure with a metamodel (E_0), a low–fidelity (E_1) and a high–fidelity (E_2) evaluation model in each deme.

4.2 Applications of HEAs

Below, two optimization problems solved using the hierarchical evaluation schemes are presented. The first one demonstrates the use of the distributed hierarchical evaluation schemes by taking advantage of two different GPU–enabled code variants that use either single precision arithmetic with a speed–up up to 110× (low level) or the mixed precision

arithmetic with a speed–up up to 90× (high level). In the second case, a hierarhical scheme with a single population is used for the design of an annular compressor cascade.

☛ The first case deals with the optimal deployment of a four–element airfoil for maximum lift coefficient (C_L), as described in 3.4, which is herein revisited using the distributed hierarchical evaluation scheme, with and without metamodels. The low–cost, low–fidelity evaluation tool uses a single precision arithmetic (GPU_{SP}, E_1) variant of the GPU-enabled Euler software and the high fidelity one uses the mixed precision arithmetic (GPU_{MP}, E_2). The CPU cost ratio of E_1 and E_2 is $\sim$0.48 : 1 for the same computational grid. The low level solver is faster but less accurate due to its single precision arithmetic. Fig. 19 compares the convergence history of DHEA and a DHMAEA in terms of CPU cost. For the sake of completeness, the convergence history of the EA of section 3.4 is also shown in this figure. The three algorithms were configured as follows:

1. A $(20, 50)$ EA using the E_2 evaluation model.
2. A DHEA with three $(5, 15)$ EA demes. In each generation, the 15 offspring of each deme were firstly evaluated on E_1 and only the top two of them were then re–evaluated on E_2.
3. A $3 \times (5, 15)$ DHMAEA. Here in each deme, 15 offspring were evaluated on the metamodel E_0 (i.e. an RBF network), the best 9 among them were re–evaluated on E_1 and only one of them on E_2. All local metamodels were trained on individuals previously evaluated on E_1.

In both distributed schemes, migration was employed every 8 generations by exchanging two individuals within any pair of demes. The two emigrants of each deme were selected after ranking the offspring population members in terms of their fitness value, without distinguishing among the evaluation models used to get this value. The migration policy is that, in the destination deme, immigrants replace the worst performing members.

As shown in fig. 19, the distributed hierarchical schemes perfoms better than the conventional EA and the combined use of metamodels and hierarchical schemes leads to even better performance.

☛ The second case is concerned with the optimization of a $3D$ annular compressor cascade. An existing cascade with 19 straight blades (chord length equal to C=0.1 m, stagger angle equal to 51.4^o) was used as reference. The blades were mounted on the casing forming a $t/C = 2\%$ clearance with the stationary hub, [20]. The purpose of this study was to redesign the airfoil of the straight blade so as to minimize the mass-averaged pressure loss coefficient PLC_{ave}. At each radial position r, PLC, is defined as

$$PLC(r) = \frac{\overline{p}_{t1}(r) - \overline{p}_t(r)}{\overline{p}_{t1}(r) - \overline{p}_1(r)}$$

where $\overline{p}_t(r)$ and $\overline{p}(r)$ stand for the circumferentially mass–averaged total and static pressures. PLC_{ave} is the average of $PLC(r)$, for all $r \in [r_{hub}, r_{shroud}]$. The cascade airfoil is parameterized using 15 NURBS control points on each side (pressure and suction sides), 5 of which were allowed to vary both on the chordwise and the normal to the chord direction, summing up to 20 design variables. Geometrical constraints on the airfoil thickness and the mean exit flow angle $\bar{\alpha}_2$ ($\bar{\alpha}_2 \leq 53^o$) were imposed.

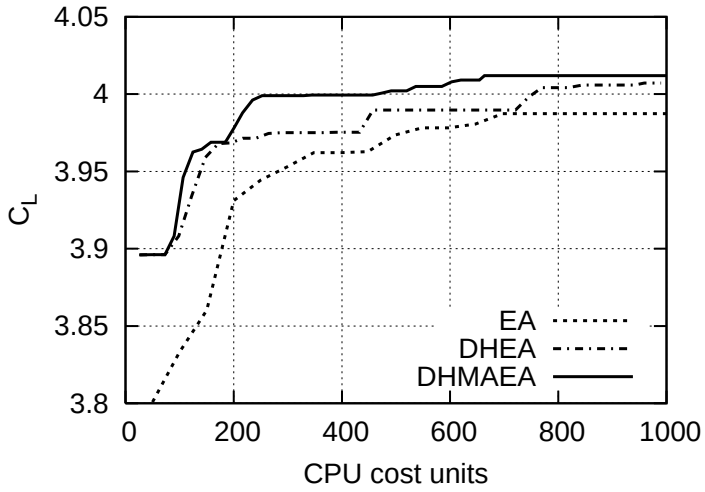


Figure 19: Optimal deployment of a four–element airfoil for maximum lift coefficient. Convergence history of the SOO problem, in terms of CPU cost.

In this case, a HMAEA with the hierarchical evaluation scheme was used. The same evaluation tool (in–house Navier–Stokes equations solver, $CFD-NS$; CPU implementation) was used on both levels, with different spatial resolution and turbulence model. In particular, the high–fidelity evaluation model (E_2) used the low–Reynolds number Spalart–Allmaras [59] turbulence model on unstructured grids of about $\sim$1.000.000 nodes. A coarser grid of $\sim$600.000 nodes and the same turbulence model with wall functions was used as the low–fidelity evaluation model (E_1). The CPU cost ratio of E_1 and E_2 $\sim 0.2:1$.

A $(20, 60)$ HMAEA, with a single population was used. Fitness inheritance[3] and RBF networks supported the IPE process. During the first generation, all individuals were evaluated on E_1 and only the 6 top of them were re–evaluated on E_2. In the second generation, the fitness inheritance technique was activated. The most promising (between 5 and 10 of them) members in the population were re–evaluated on E_1 and only the best of them on E_2. Once 100 previously evaluated individuals on E_1 were archived in the DB, RBF networks were used in place of fitness inheritance. The HMAEA terminated at 80 CPU cost units, i.e. 80 equivalent high level $CFD-NS$ runs.

Fig. 20 illustrates the convergence of the optimization algorithm. The PLC_{ave} values of the reference and optimal blade cascades were found equal to 0.1189 and 0.0843, respectively. The gain from using the HMAEA is clear since locating the optimal solution at the cost of only 80 CPU cost units would be otherwise impossible.

In fig. 21, iso–contours of the total pressure loss coefficient $C_{pt} = \frac{\overline{p}_{t1} - \overline{p}_t}{\overline{p}_{t1} - \overline{p}_1}$ are plotted on a transversal cross–section downstream of the blade leading edge, for both the reference and the optimal blades. In the redesigned blade, losses induced by the tip clearance vortex are much lower and which lead to the lower PLC_{ave} value, compared to the reference blade. The total pressure field over part of the stator row (four blades) for the optimal blade geometry is shown in fig. 22.

[3]Fitness inheritance is a method that approximates the fitness of any offspring from the fitness values of its parents.

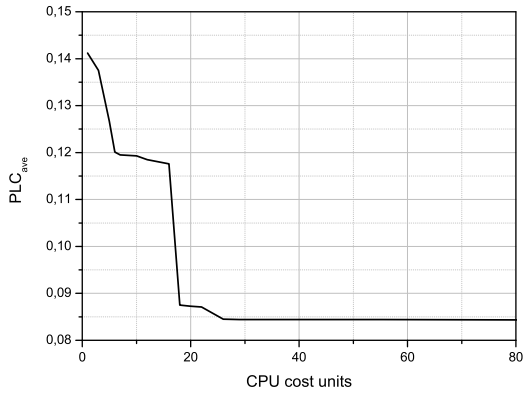


Figure 20: Optimization of an annular cascade. Convergence of the HMAEA. One CPU cost unit corresponds to a single computation using the high–fidelity CFD model (E_2). From [20].

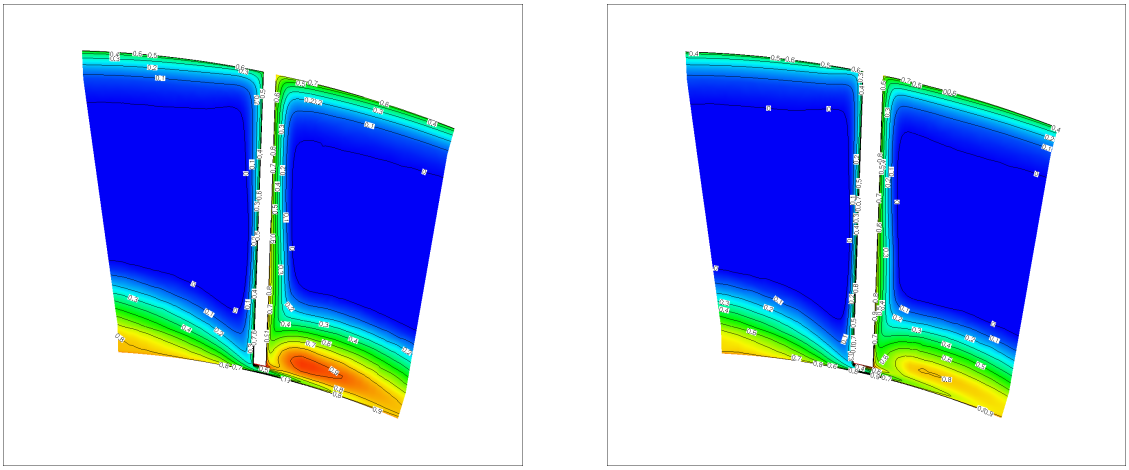


Figure 21: Optimization of an annular cascade. Total pressure loss fields plotted over a transversal cross–section at 0.782% axial chord downstream of the blade leading edge. Computed results for the reference (left) and optimal (right) blade. From [20].

5 Asynchronous EAs & MAEAs

The EAs (MAEAs, HMAEAs, etc.) discussed in the previous sections are referred to as synchronous EAs since these are characterized by the notion of "generation". The end of each generation acts as a synchronization barrier, regularly during the evolution. Upon completion of each generation, the need for synchronization may seriously harm the parallel efficiency of the (synchronous) EAs. As a remedy to this problem, asynchronous EAs (AEAs), that overcome the notion of generation and may optimally use all available computational resources, have been proposed.

Below, the AEA originally proposed by the author's group, [1], which is assisted by metamodels based on the IPE technique (asynchronous MAEA [2]) along with its application to the design of a Hydromatrix® runner are presented.

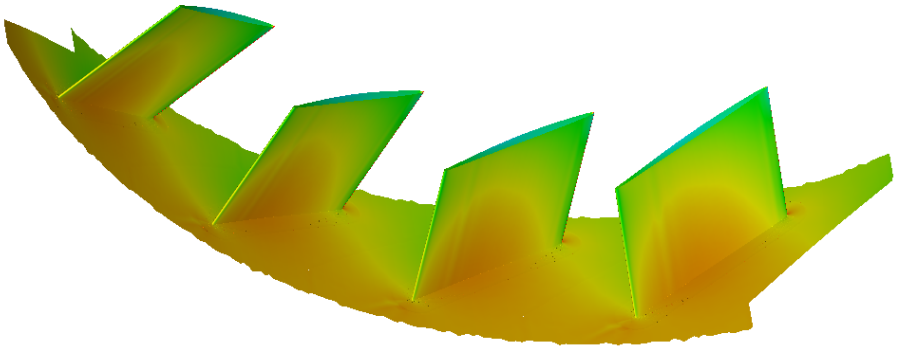


Figure 22: Optimization of an annular cascade. Total pressure field on the optimal cascade blade.

5.1 AEA: Basic Features

The AEA presented in [1] utilises a number of search agents located at the nodes of a 2D structured mesh ("supporting mesh", fig. 23) with dimensions $n_1 \times n_2$ which is considered to be periodic along its two pairs of opposite sites (n_1, n_2 are user–defined even integers).

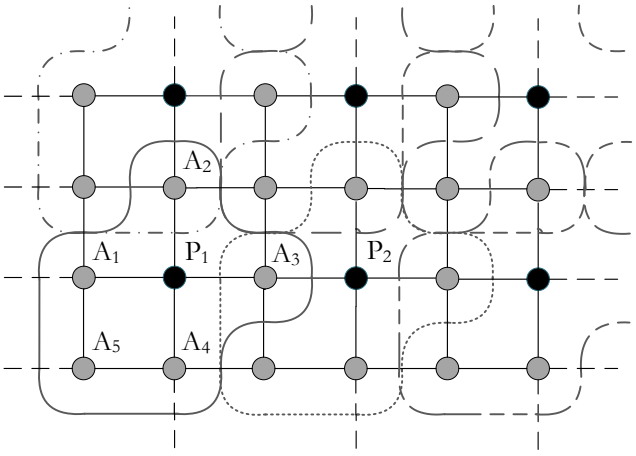


Figure 23: Asynchronous EA. A 6×4 supporting mesh. Poles correspond to circles filled in black.

The mesh is divided into a number ($n_1 n_2/4$) of overlapping demes ($\mathcal{D}_p$) with six nodes each. Each deme comprises five search agents and a single pole, leading to $N_{poles} = n_1 n_2/4$ and $N_{agents} = 3n_1 n_2/4$. The demes overlapping enables the inter–deme communication and

exchange of information among them. Each pole acts as the front–end of its deme, where the best–so–far individual (local best) computed by the search agents of this deme is stored and, through the local best, affects the formation of new candidate solutions. On the other hand, search agents undertake evaluations, generate new candidate solutions (based on evolution operators applied within each deme) and are responsible for updating the information stored in the poles (according to the outcome/result of each evaluation).

The AEA algorithm, which is suitable for either *Cluster* or *Grid Computing* with N_{CPU} available processors, is presented below:

Algorithm AEA (Asynchronous Evolutionary Algorithm).

AEA1. [Start Algorithm] N_{CPU} randomly generated individuals are associated with N_{CPU} agents and undergo evaluation on all available processors.

AEA2. [Receive] The evaluation of an individual $(\boldsymbol{x}_a, \boldsymbol{F}(\boldsymbol{x}_a))$ (the subscript a stands for agent) is completed and the corresponding CPU ($CPU_{\boldsymbol{x}_a}$) becomes instantaneously idle.

AEA3. [Elitism] The elite archival set, $\mathcal{P}_a$, is updated accordingly.

AEA4. [Pole Displacement] Through an intra–deme process, a decision on whether the just evaluated individual $\boldsymbol{x}_a$ must displace $\boldsymbol{x}_p$ which is stored at the corresponding pole(s) is made. To this end, $\boldsymbol{F}(\boldsymbol{x}_a)$ is compared with $\boldsymbol{F}(\boldsymbol{x}_p)$, based on dominance criteria (MOO problems).

AEA5. [Update Ages] Agents' and poles' ages are updated. The age A_k of each agent k with $k \in [1, N_{agents}]$ is defined as the difference between the serial number of the last evaluation carried out for this agent and the serial number of the current evaluation. Each pole takes on the average age of its agents,

$$A_p = \frac{1}{5} \sum_{k \in \mathcal{D}_p} A_k, \text{ with } p \in [1, N_{poles}]$$

AEA6. [Update Priorities] The priority of each pole Pr_p (which is equal to the priority of its deme) is the product of the age– and a cost–based priorites, as

$$Pr_p = Pr_p^{age} Pr_p^{cost}$$

with

$$Pr_p^{age} = \frac{A_p}{\max\,(A_1, \cdots, A_{N_{poles}})} \text{ and } Pr_p^{cost} = \frac{\phi_p}{\phi_{max} - \phi_{min}}$$

where $\max\,(A_1, \cdots, A_{N_{poles}})$ denotes the maximum age of all poles, $\phi_p = \phi(\boldsymbol{x}_p)$ is the scalar cost function associated with pole p, and $\phi_{max} = \max\,(\phi_1, \cdots, \phi_{N_{poles}})$, $\phi_{min} = \min\,(\phi_1, \cdots, \phi_{N_{poles}})$ are the maximum and minimum ϕ values among all poles, respectively.

AEA7. [Select New Deme & Agent] The agent with the maximum age, within the deme with the maximum priority, is selected as the one where the new individual to undergo evaluation must be formed.

AEA8. [Recombination & Mutation] The new individual is formed by superimposing the weighted difference between two agents of the deme selected in AEA7 to the individual associated with the pole $\boldsymbol{x}_p$, as

$$\boldsymbol{x}_a^{new,temp} = \boldsymbol{x}_p + \omega_r(\boldsymbol{x}_{k_1} - \boldsymbol{x}_{k_2}), \text{ with } k_1, k_2 \in \mathcal{D}_p^n \ \& \ k_1 \neq k_2 \tag{26}$$

where $\omega_r \in [0, 1]$. A non–uniform mutation scheme, with a user–defined probability is, then, applied to $\boldsymbol{x}_a^{new,temp}$, yielding $\boldsymbol{x}_a^{new}$.

AEA9. [Assign Evaluation] Assign the evaluation of $\boldsymbol{x}_a^{new}$ on the idle CPU, $CPU_{\boldsymbol{x}_a}$.

AEA10. [Termination] If the maximum number of evaluations is reached or the elite set does not further improve for a user–defined number of successive evaluations the algorithm terminates. Otherwise, it continues from step AEA2. ∎

5.2 Implementation of Metamodels in AEA

The efficiency of the AEA can be significantly improved by employing metamodels, according to the IPE technique. In the synchronous variants of EAs which have been discussed in previous sections, the IPE technique is applied for inexactly pre–evaluating all the population at each generation and screen out the non–promising individuals. The lack of generations in AEA requires a "different" way of performing the IPE of candidate solutions.

Thus, the asynchronous metamodel–assisted EA (AMAEA, [2]), starts as an AEA until a user–defined number of exact evaluations are completed and archived in DB. Then, the IPE phase begins. During IPE, instead of generating a single new individual in the agent selected to undergo evaluation (step AEA8), N_{IPE} trial individuals are generated and inexactly pre–evaluated on the metamodel. The "best" of them, according to the metamodel is selected to undergo evaluation. This algorithm, which replaces step AEA8, is presented below:

Algorithm AMAEA (Asynchronous MAEA).

AMAEA1. [Generate and approximately evaluate N_{IPE} trial individuals] For each trial individual t, $t \in [1, N_{IPE}]$, the following actions are taken:

AMAEA1a. [Recombination & Mutation] Each trial individual ($\boldsymbol{x}_a^t$) is generated according to the recombination and mutation schemes of step AEA8 and eq. 26.

AMAEA1b. [Inexact Evaluation] For $\boldsymbol{x}_a^t$, a local metamodel is trained on a small, user–defined, number of data selected from the DB. The training pattern's selection is based on the algorithm described in subsection 3.3. Then, approximate objective values, $\tilde{\boldsymbol{F}}(\boldsymbol{x}_a^t)$, are computed.

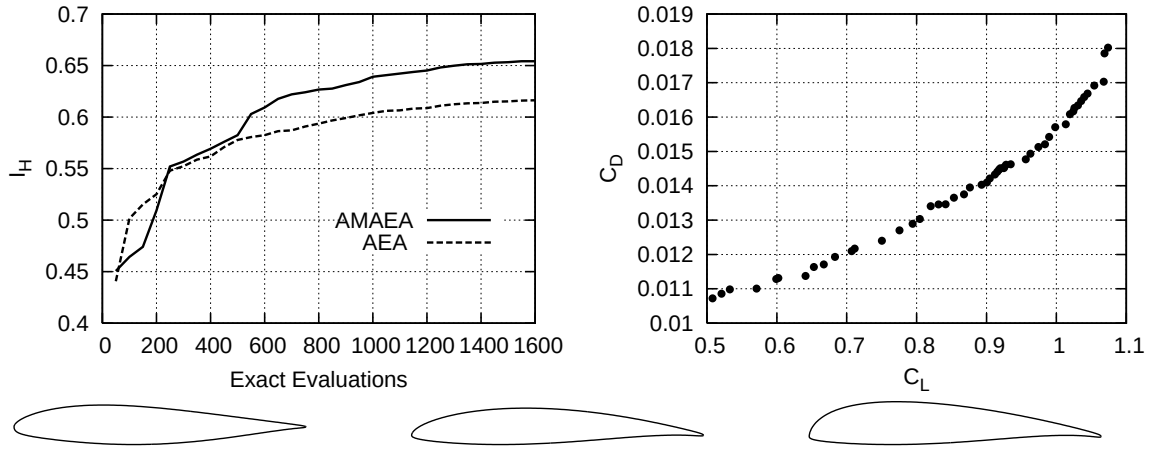
AMAEA2. [Select Trial Individual] The "best" trial individual according to the metamodel, ($\boldsymbol{x}_a^t$, $\tilde{\boldsymbol{F}}(\boldsymbol{x}_a^t)$), $t \in [1, N_{IPE}]$ is selected to undergo evaluation on the problem–specific model. In MOO problems, this selection is based on dominance and strength criteria. ∎

5.3 Applications of AEAs

This section presents two optimization problems solved using AEAs and AMAEAs. The first case demonstrates the gain from the use of metamodels on AEA on the design of an isolated airfoil whereas, in the second case, the AMAEA is used for the design–optimization of a Hydromatrix® runner blade.

☛ The first case is dealing with the two–objective shape optimization of an isolated airfoil for minimum C_D and maximum C_L. The flow conditions are $M_\infty = 0.3$, $\alpha_\infty = 4^o$ and $Re_c = 5 \cdot 10^6$. The $CFD-NS$ solver was used. The airfoil is parameterized using 24 design variables. Geometrical constraints were imposed on the airfoil thickness.

This case was studied using AEA and AMAEA, both of them on a 12×10 supporting mesh. AMAEA employed the IPE technique by generating $N_{IPE} = 15$ trial members for each individual to be evaluated. The comparison of the evolution of the hypervolume indicator I_H for the AEA and the AMAEA along with the front of the non–dominated solutions obtained using the AMAEA are shown fig. 24. From this figure, it is clear that AMAEA outperforms AEA.



$C_L = 0.508, C_D = 0.0107$ $C_L = 0.9047, C_D = 0.0142$ $C_L = 1.0739, C_D = 0.018$

Figure 24: Shape optimization of a 2D isolated airfoil. Comparison of the convergence of the hypervolume indicator for the AEA and the AMAEA and the computed front of non–dominated solutions. Three airfoil shapes corresponding to three points of the computed front of non–dominated solutions are also shown. From [2].

☛ The second case presents the design–optimization of a Hydromatrix® runner blade using AMAEA. Compared to the case presented in section 3.4, the new design is carried out at different operating points and is handled as a three objective problem (min F_1, F_2 and F_3). The three objective functions are the weighted sum of metrics f_1, f_2 and f_3 (for the three operating points), as

$$F_j = \sum_{i=1}^{3} w^{OP_i} f_j^{OP_i}, \quad j = 1, 2, 3$$

with $w^{OP_1} = 1.0$, $w^{OP_2} = w^{OP_3} = 0$ for F_1 and $w^{OP_1} = 1.0$, $w^{OP_2} = w^{OP_3} = 0.1$ for both F_2 and F_3. The f_1, f_2, f_3 metrics have been defined in section 3.4.

The design was performed using an AMAEA with a 10×10 supporting mesh, i.e. with 75 agents and 25 poles on 16 interconnected CPUs and a stopping criterion of 2000 CFD–based evaluations. The IPE technique was applied after the first 300 non–failed evaluations were stored in the DB, by generating and approximately evaluating $N_{IPE}=8$ trial members.

Fig. 25 presents the front of non–dominated solutions computed by the AMAEA at the cost of 2000 CFD evaluations. The runner's blades along with the pressure field at the best efficiency point (OP_1) for an indicative solution from this front (point marked with the cube) are shown in fig. 26. Fig. 27 presents statistical results on the evaluations performed by each agent (left part) and deme (right part), expressed as percentage (%) of the total number of evaluations. From this figure, one may notice that the evaluations are almost equally shared among the 25 demes. Also, as expected, unlike agents shared by two demes (see fig. 23), those belonging exclusively to a single deme usually perform less evaluations.

In this case, the important thing is that the available CPUs were used without idle periods of time, since no synchronization occurs in AMAEA; so, this optimization problem practically had 100% parallel efficiency.

Acknowledgement

The optimization of the Hydromatrix® design cases was supported by Andritz HYDRO (Linz, Austria and Vevey, Switzerland), in the framework of direct collaborations between NTUA and the aforementioned company and/or the EU funded project "HYDROACTION".

References

[1] V.G. Asouti and K.C. Giannakoglou. Aerodynamic optimization using a parallel asynchronous evolutionary algorithm controlled by strongly interacting demes. *Engineering Optimization*, 41(3):241–257, 2009.

[2] V.G. Asouti, I.C. Kampolis, and K.C. Giannakoglou. A grid-enabled asynchronous metamodel-assisted evolutionary algorithm for aerodynamic optimization. *Genetic Programming and Evolvable Machines (SI:Parallel and Distributed Evolutionary Algorithms, Part One)*, 10(3):373–389, 2009.

[3] V.G. Asouti, X.S. Trompoukis, I.C. Kampolis, and K.C. Giannakoglou. Unsteady CFD computations using vertex-centered finite volumes for unstructured grids on Graphics Processing Units. *International Journal for Numerical Methods in Fluids*, 67(2):232–246, 2011.

[4] N. Benoudjit, C. Archambeau, A. Lendasse, J. Lee, and M. Verleysen. Width optimization of the Gaussian kernels in radial basis function networks. In *European Symposium on Artificial Neural Networks – ESANN 2002*, pages 425–432, Bruges, Belgium, 2002.

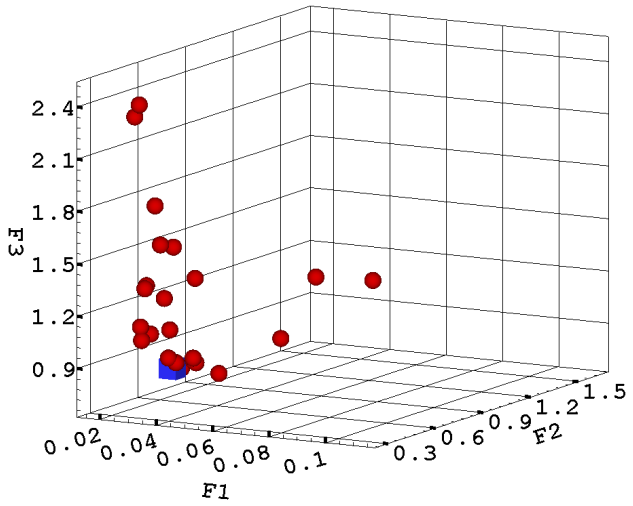


Figure 25: Three–objective optimization of a Hydromatrix® runner. Front of the computed non–dominated solutions, at the end of 2000 CFD–based evaluations. From [58].

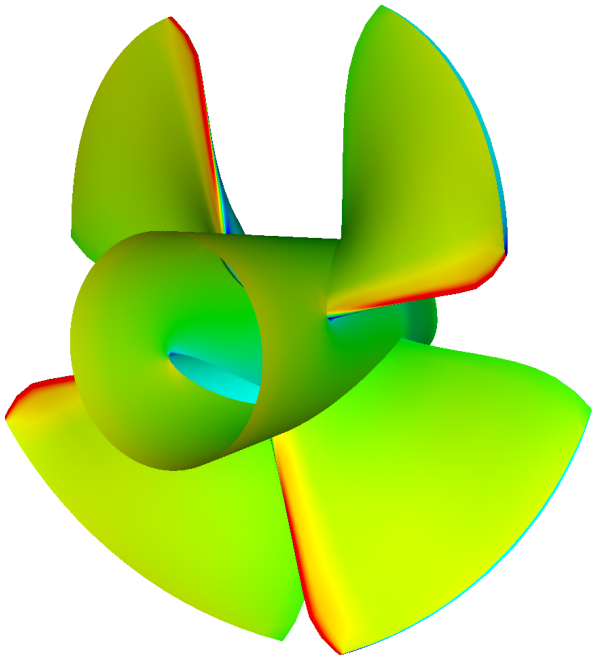


Figure 26: Three–objective optimization of a Hydromatrix® runner. View of the blades of the runner for the selected solution (point marked with a cube in fig. 25) along with the pressure field at OP_1. From [58].

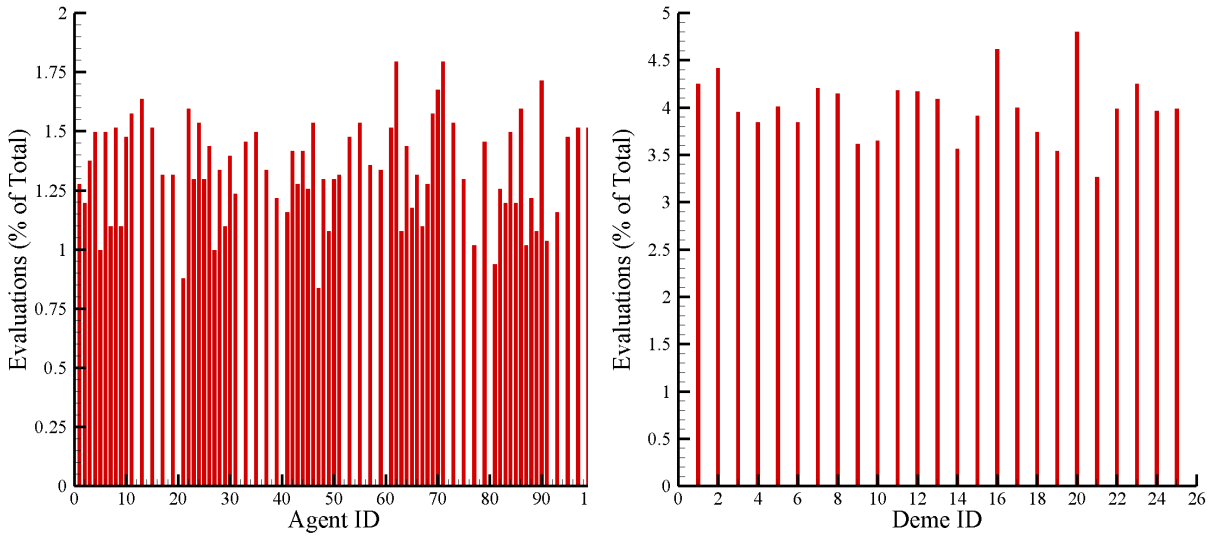


Figure 27: Three–objective optimization of a Hydromatrix® runner. Statistical results on the evaluations performed by each agent (left) and deme (right) of the supporting mesh, as a percentage of the total number of evaluations. From [58].

[5] J. Branke and C. Schmidt. Faster convergence by means of fitness estimation. *Soft Computing – A Fusion of Foundations, Methodologies and Applications*, 9(1):13–20, 2005.

[6] D. Büche, N. Schraudolph, and P. Koumoutsakos. Accelerating evolutionary algorithms with Gaussian process fitness function models. *IEEE Transactions on Systems, Man, and Cybernetics — Part C: Applications and Reviews*, 35(2):183–194, May 2005.

[7] L. Bull. On model-based evolutionary computation. *Soft Computing — A Fusion of Foundations, Methodologies and Applications*, 3(2):76–82, 1999.

[8] K. Deb, S. Agrawal, A. Pratap, and T. Meyarivan. A fast elitist non-dominated sorting genetic algorithm for multi-objective optimization: NSGA-II. In *Parallel Problem Solving from Nature – PPSN VI*, Paris, France, 2000.

[9] J.-A. Désidéri and A. Janka. Hierarchical parametrization for multilevel evolutionary shape optimization with application to aerodynamics. In *EUROGEN 2003, Evolutionary Methods for Design, Optimisation and Control with Applications to Industrial Problems*, Barcelona (Spain), 2003.

[10] M. Drela and M.B. Giles. Viscous-inviscid analysis of transonic and low Reynolds number airfoils. *AIAA Journal*, 25(10):1347–1355, 1987.

[11] M. Emmerich, K.C. Giannakoglou, and B. Naujoks. Single- and multi-objective evolutionary optimization assisted by Gaussian random field metamodels. *IEEE Transactions on Evolutionary Computation*, 10(4):421–439, 2006.

[12] M. Emmerich, A. Giotis, M. Ozdemir, Bäck T., and K.C. Giannakoglou. Metamodel-assisted evolution strategies. In *Parallel Problem Solving from Nature – PPSN VII*, pages 361–370, Granada, Spain, 2002. Springer.

[13] M. Farina. A neural network based generalized response surface multiobjective evolutionary algorithm. In *2002 Congress on Evolutionary Computation – CEC '02*, Honolulu, HI, USA, May 2002.

[14] B. Fritzke. Fast learning with incremental RBF Networks. *Neural Processing Letters*, 1(1):2–5, 1994.

[15] C.A. Georgopoulou and K.C. Giannakoglou. A multi-objective metamodel-assisted memetic algorithm with strength-based local refinement. *Engineering Optimization*, 41(10):909–923, 2009.

[16] C.A. Georgopoulou and K.C. Giannakoglou. *Multiobjective Metamodel-Assisted Memetic Algorithms*. Springer Series, 2009.

[17] K.C. Giannakoglou. Designing turbomachinery blades using evolutionary methods. ASME Paper 99-GT-181, 44th ASME Gas Turbine & Aeroengine Congress, Indianapolis, IN, USA, June 1999.

[18] K.C. Giannakoglou. The EASY (Evolutionary Algorithms SYstem) software, http://velos0.ltt.mech.ntua.gr/EASY, 2008.

[19] K.C. Giannakoglou, A.P. Giotis, and M.K. Karakasis. Low-cost genetic optimization based on inexact pre-evaluations and the sensitivity analysis of design parameters. *Inverse Problems in Engineering*, 9(4):389–412, 2001.

[20] K.C. Giannakoglou and I.C. Kampolis. *Multilevel Optimization Algorithms based on Metamodel- and Fitness Inheritance-Assisted Evolutionary Algorithms.*, chapter 3. Springer-Verlag, 2009.

[21] A.P. Giotis, K.C. Giannakoglou, and J. Périaux. A reduced-cost multi-objective optimization method based on the Pareto front technique, neural networks and PVM. In *European Congress on Computational Methods in Applied Sciences and Engineering – ECCOMAS 00*, Barcelona, Spain, September 2000.

[22] R.M. Greenman and K.R. Roth. Minimizing computational data requirements for multi-element airfoils using neural networks. In *37^{th} AIAA Aerospace Sciences Meeting and Exhibit*, Reno, NV, USA, January 1999. AIAA-1999-0258.

[23] S. Haykin. *Neural Networks: A Comprehensive Foundation*. Prentice Hall, New Jersey, USA, 2nd edition, 1999.

[24] F. Herrera, M. Lozano, and C. Moraga. Hierarchical distributed genetic algorithms. *International Journal of Intelligent Systems*, 14(9):1099–1121, 1999.

[25] 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, October 2002.

[26] Y. Jin and B. Sendhoff. Reducing fitness evaluations using clustering techniques and neural network ensembles. In *Genetic and Evolutionary Computation – GECCO 2004*, pages 688–699, Seattle, WA, USA, June 2004.

[27] I.C. Kampolis and K.C. Giannakoglou. A multilevel approach to single- and multi-objective aerodynamic optimization. *Computer Methods in Applied Mechanics and Engineering*, 197(33-40):2963–2975, 2008.

[28] I.C. Kampolis and K.C. Giannakoglou. Distributed evolutionary algorithms with hierarchical evaluation, engineering optimization. *Engineering Optimization*, 41(11):1037–1049, 2009.

[29] I.C. Kampolis and K.C. Giannakoglou. Synergetic use of different evaluation, parameterization and search tools within a multilevel optimization platform. *Applied Soft Computing*, 11(1):645–651, 2011.

[30] I.C. Kampolis, X.S. Trompoukis, V.G Asouti, and K.C. Giannakoglou. CFD-based analysis and two-level aerodynamic optimization on graphics processing units. *Computer Methods in Applied Mechanics and Engineering*, 199(9-12):712–722, 2010.

[31] I.C. Kampolis, A.S. Zymaris, V.G. Asouti, and K.C. Giannakoglou. Multilevel optimization strategies based on metamodel-assisted evolutionary algorithms, for computationally expensive problems. In *2007 Congress on Evolutionary Computation – CEC '07*, Singapore, September 2007.

[32] M.K. Karakasis and K.C. Giannakoglou. Metamodel-assisted multi-objective evolutionary optimization. EUROGEN 05, Evolutionary and Deterministic Methods for Design, Optimization and Control with Applications to Industrial and Societal Problems, Munich, September 2005.

[33] M.K. Karakasis and K.C. Giannakoglou. On the use of metamodel-assisted, multi-objective evolutionary algorithms. *Engineering Optimization*, 38(8):941–957, 2006.

[34] M.K. Karakasis, A.P. Giotis, and K.C. Giannakoglou. Inexact information aided, low-cost, distributed genetic algorithms for aerodynamic shape optimization. *International Journal for Numerical Methods in Fluids*, 43(10-11):1149–1166, 2003.

[35] M.K. Karakasis, D.G. Koubogiannis, and K.C. Giannakoglou. Hierarchical distributed evolutionary algorithms in shape optimization. *International Journal for Numerical Methods in Fluids*, 53(3):455–469, 2007.

[36] N.B. Karayiannis and G.W. Mi. Growing radial basis neural networks: Merging supervised and unsupervised learning with network growth techniques. *IEEE Transactions on Neural Networks*, 8(6):1492–1506, 1997.

[37] D. Knuth. *The Art of Computer Programming, Volume 1: Fundamental Algorithms*. Addison-Wesley, 3rd edition, 1997.

[38] J.R. Koehler and A.B. Owen. *Handbook on Statistics*, volume 13, chapter Computer Experiments, pages 239–245. Elsevier-Science, 1996.

[39] E.A Kontoleontos, V.G. Asouti, and K.C. Giannakoglou. An asynchronous metamodel-assisted memetic algorithm for cfd-based shape optimization. *Engineering Optimization*, 44(2):157–173, 2012.

[40] N. Krasnogor and J. Smith. A tutorial for competent memetic algorithms: model, taxonomy, and design issues. *IEEE Transactions on Evolutionary Computation*, 9(5):474–488, 2005.

[41] D.G. Krige. A study of gold and uranium distribution patterns in the Klerksdorp gold field. *Geoexploration*, 4(1):43–53, 1966.

[42] S.A. Kyriacou, S. Weissenberger, and K.C. Giannakoglou. Design of a matrix hydraulic turbine using a metamodel-assisted evolutionary algorithm with pca-driven evolution operators. *International Journal of Mathematical Modelling and Numerical Optimization (SI:Simulation-Based Optimization Techniques for Computationally Expensive Engineering Design Problems)*, 2011.

[43] P.I.K. Liakopoulos, I.C. Kampolis, and K.C. Giannakoglou. Grid-enabled, hierarchical distributed metamodel-assisted evolutionary algorithms for aerodynamic shape optimization. *Future Generation Computer Systems*, 24(7):701–708, 2008.

[44] N. Melab, S. Cahon, and E-G. Talbi. Grid computing for parallel bioinspired algorithms. *Journal of Parallel and Distributed Computing*, 66(8):1052–1061, 2006.

[45] D.C. Montgomery. *Design and Analysis of Experiments*. John Wiley & Sons, New York, USA, 6th edition, 2005.

[46] H. Nakayama, K. Inoue, and Y. Yoshimori. Approximate optimization using computational intelligence and its application to reinforcement of cable-stayed bridges. In *ECCOMAS 2004, European Congress on Computational Methods in Applied Sciences and Engineering*, Jÿvaskÿla, Finland, July 2004.

[47] Y. S. Ong, K. Y. Lum, P. B. Nair, D. M. Shi, and Z. K. Zhang. Global convergence of unconstrained and bound constrained surrogate-assisted evolutionary search in aerodynamic shape design. In *2003 Congress on Evolutionary Computation – CEC '03*, volume 3, pages 1856–1863, Canberra, Australia, 2003.

[48] 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.

[49] Y.S. Ong and A.J. Keane. Meta-lamarckian learning in memetic algorithms. *IEEE Transactions on Evolutionary Computation*, 8(2):99–110, 2004.

[50] Y.S. Ong, M.H. Lim, N. Zhu, and K.W. Wong. Classification of adaptive memetic algorithms: a comparative study. *IEEE Transactions on Systems, Man, and Cybernetics - Part B*, 36(1):141–152, 2006.

[51] 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.

[52] M. Papadrakakis, N.D. Lagaros, and Y. Tsompanakis. Structural optimization using evolution strategies and neural networks. *Computer Methods in Applied Mechanics and Engineering*, 156(1–4):309–333, 1998.

[53] L. Piegl and W. Tiller. *The NURBS Book*. Springer-Verlag, Germany, 2nd edition, 1997.

[54] S. Pierret and R. A. Van den Braembussche. Turbomachinery blade design using a Navier-Stokes solver and artificial neural network. *Journal of Turbomachinery*, 121(2):326–332, April 1999. ASME Paper 99-GT-4.

[55] T. Poggio and F. Girosi. Networks for approximation and learning. *Proceedings of the IEEE*, 78(9):1481–1497, 1990.

[56] A. Ratle. Optimal sampling strategies for learning a fitness model. In *1999 Congress on Evolutionary Computation – CEC '99*, pages 2078–2085, Washington, DC, USA, 1999.

[57] M. Sefrioui and J. Périaux. A hierarchical genetic algorithm using multiple models for optimization. In M. Schoenauer, K. Deb, and et al., editors, *6th international conference on parallel problem solving from nature (PPSN VI). Lecture Notes in Computer Science*, volume 1917, pages 879–888. Springer-Verlag, Paris, 2000.

[58] I.A. Skouteropoulou, S.A. Kyriacou, V.G. Asouti, K.C. Giannakoglou, S. Weissenberger, and P. Grafenberger. Design of a hydromatrix turbine runner using an asynchronous algorithm on a multi-processor platform. In *7th GRACM, International Congress on Computational Mechanics*, Athens, 30 June-2 July 2011.

[59] P. Spalart and S. Allmaras. A one–equation turbulence model for aerodynamic flows. *La Recherche Aérospatiale*, 1:5–21, 1994.

[60] N. Srinivas and K. Deb. Multiobjective optimization using nondominated sorting in genetic algorithms. *Evolutionary Computation*, 2(3):221–248, 1995.

[61] A. Tikhonov and V. Arsénine. *Méthodes de Résolution de problèmes mal posés.* Editions MIR, Moscou, 1976.

[62] A.N. Tikhonov, A.V. Goncharsky, V.V. Stepanov, and A.G. Yagola. *Numerical Methods for the Solution of Ill-Posed Problems*. Kluwer Academic Publishers, 1995.

[63] H. Ulmer, F. Streichert, and A. Zell. Evolution strategies assisted by Gaussian processes with improved pre-selection criterion. In *2003 Congress on Evolutionary Computation – CEC '03*, Canberra, Australia, 2003.

[64] L. Willmes, T. Back, Y. Jin, and B. Sendhoff. Comparing neural networks and kriging for fitness approximation in evolutionary optimization. In *Proceedings of the 2003 Congress on Evolutionary Computation – CEC '03*, volume 1, pages 663–670, 2003.

[65] E. Zitzler, M. Laumans, and L. Thiele. SPEA2: Improving the strength Pareto evolutionary algorithm for multiobjective optimization. In *Eurogen 2001, Evolutionary Methods for Design, Optimisation and Control with Applications to Industrial Problems*, pages 19–26, Barcelona, 2002. CIMNE.

[66] E. Zitzler and L. Thiele. Multiobjective evolutionary algorithms: A comparative case study and the Strength Pareto approach. *IEEE Transactions on Evolutionary Computation*, 3(4):257–271, November 1999.

[67] A.S. Zymaris, D.I. Papadimitriou, K.C. Giannakoglou, and C. Othmer. Adjoint wall functions: A new concept for use in aerodynamic shape optimization. *Journal of Computational Physics*, 229(13):5228–5245, 2010.