National Technical University of Athens
School of Mechanical Engineering
Fluids Section
Laboratory of Thermal Turbomachines
Parallel CFD & Optimization Unit

Low-Cost Metamodel-Assisted Evolutionary Algorithms with Application in Shape Optimization in Fluid Dynamics

PhD Thesis

Dimitrios H. Kapsoulis

Supervisor: Kyriakos C. Giannakoglou
Professor NTUA

Athens, 2019

National Technical University of Athens
School of Mechanical Engineering
Fluids Section
Laboratory of Thermal Turbomachines
Parallel CFD & Optimization Unit

Low-Cost Metamodel-Assisted Evolutionary Algorithms with Application in Shape Optimization in Fluid Dynamics

PhD Thesis

Dimitrios H. Kapsoulis

Examination Committee:

1. Kyriakos C. Giannakoglou* (Supervisor), Professor, NTUA,
   School of Mechanical Engineering
2. Nikolaos Lagaros*, Associate Professor, Dean, NTUA,
   School of Civil Engineering
3. Andreas Boudouvis*, Professor, NTUA,
   School of Chemical Engineering
4. Konstantinos Mathioudakis, Professor, NTUA,
   School of Mechanical Engineering
5. Dimitrios Bouris, Associate Professor, NTUA,
   School of Mechanical Engineering
6. Athanasios Tolis, Associate Professor, NTUA
   School of Mechanical Engineering
7. Ioannis Nikolos, Professor, Technical University of Crete
   School of Production Engineering & Management

**Member of the Advisory Committee.*

Athens, 2019

Abstract

The scope of this PhD is to propose, develop and assess several upgrades to existing shape optimization methods based on Evolutionary Algorithms (EAs). The efficiency of the proposed improvements is demonstrated in a number of real-world applications in the field of fluid mechanics (aerodynamic, hydrodynamics and turbomachinery) which are associated with computationally expensive evaluation software. They noticeably reduce the computational cost of optimization compared to the available (background) methods, which are still based on EAs enhanced by metamodels (Metamodel-Assisted EAs or MAEAs) and distributed search. Metamodels, mainly Radial Basis Function networks, are on-line trained personalized surrogate evaluation models, meaning that a local metamodel is trained for the pre-evaluation of each new individual generated during the evolution. This is in contrast to the common use of off-line trained metamodels widely used by other relevant methods. Parallelization, in the form of concurrent evaluations of the candidate solutions on the multi-processor platform of the Parallel CFD & Optimization Unit (PCOpt) of the Lab of Thermal Turbomachines of the NTUA is an indispensable feature of the proposed method variants. All developments have been made in the generic optimization platform EASY (Evolutionary Algorithm SYstem) developed by the PCOpt/NTUA. In all but one optimization problems, the problem-specific model to evaluate the candidate solutions is the GPU-enabled CFD solver PUMA developed by the same group. Only in the case of the optimization of the valveless diaphragm micropump, a different in-house CFD tool based on the cut-cell method is used instead.

The most important contributions of this thesis are listed below:

a) The use of Principal Component Analysis (PCA) to assist the EAs during the evolution. In this thesis, the Kernel PCA is used and is shown to provide better results compared to the Linear PCA used so far. In each generation of the EA, the PCA performs an eigendecomposion of the offspring population. The resulting eigenvectors define a new feature space, which the population members are transformed into; the evolution operators are applied in the feature space in which they perform optimally. Moreover, the PCA assists the MAEAs. Metamodels are only trained on the most important variables (directions in the feature space) indicated by the PCA, while the rest are safely truncated, as these generate noise at the predictions. The metamodels are trained with transformed by the PCA patterns, with truncated design variables, leading to reduced training cost and more dependable predictions. The two-fold usage of PCA drives the EA-based search in a much better way.

b) A PCA-based Hybrid Algorithm aiming at maximum efficiency in Multi-Objective Optimization (MOO). This hybrid method combines the advantages of EA and Gradient-Based (GB) optimization. The EA explores the design space while the GB method regularly upgrades the most promising solutions. The required gradients of the objective functions with respect to the design variables are efficiently

computed with the continuous adjoint method developed and programmed in the PCOpt/NTUA, at a cost which is independent of the number of design variables. In MOO, the direction along which the GB method updates the selected individuals is of outmost importance. Herein, the Linear PCA computes the principal components of the objective space by processing the objective function values of individuals forming the current front of non-dominated solutions. The principal component (direction) corresponding to the minimum variance is perpendicular to the current front and points towards the direction of the simultaneous improvement of all objective functions, so this is used for the GB refinement. The proposed hybrid method performs better than the non-hybridized EA-based search.

c) Multi-Criteria Decision Making (MCDM) within EAs to account for the Decision Maker's (DM) preferences during the evolution. In contrast to standard multi-objective EAs which may insufficiently populate the preferred area(s) of the objective space, more non-dominated solutions are now driven towards them. This is achieved by using the MCDM Technique for Order of Preference by Similarity to Ideal Solution (TOPSIS), which affects the parent selection and the non-dominated front trimming operators.

d) Flow prediction with Deep Neural Networks (DNNs) to assist the design/optimization of aerodynamic shapes. Trained on databases of CFD simulations, DNNs learn to predict the flow field around/inside these bodies, such as airfoils, wings and turbomachinery cascades. In this thesis, inputs and outputs are processed as images, in 2D cases, or raw data, in 3D cases. The DNNs are validated on new shapes and their ability to replicate the CFD results with high precision and low computational cost is demonstrated. The DNNs are employed as off-line trained metamodels during the EA-based search, in contrast to the on-line trained metamodels used in the aforementioned MAEAs.

The background and the aforementioned methods can work synergistically or separately to improve the performance of EA-based optimization methods as it is demonstrated in two groups of CFD applications. The first group consists of some "standard" CFD-based optimization problems, the so-called benchmark cases. Each time a new variant is presented, these are revisited. By doing so, the reader should clearly assess the improvement offered by the proposed method. In a separate chapter of this thesis, a number of industrial cases are presented and optimized with the most efficient methods presented. These include the shape optimization of : (a) an Aircraft Wing-Body Configuration, (b) the DrivAer Car, (c) an Ultra-light Aircraft, (d) a Francis Runner and (e) a Valveless Diaphragm Micropump.

Keywords: Evolutionary Algorithms, Multi-Objective Optimization, Metamodels, Kernel Principal Component Analysis, Hybrid Optimization, Multi-Criteria Decision Making, Gradient-Based Optimization, Deep Neural Networks, Computational Fluid Dynamics.

Acknowledgements

I would like to express my gratitude towards all those who supported me during my PhD thesis, especially my supervisor Professor K. Giannakoglou, School of Mechanical Engineering, National Technical University of Athens (NTUA), for giving me the opportunity to carry out my diploma and PhD thesis under his supervision. I feel honored to have been involved in the research activities of the Parallel CFD & Optimization Unit (PCOpt) of NTUA and to have contributed to the enrichment of its computational tools and methods. I wish to express my warm and sincere gratitude towards him, for his guidance and support over the last 6 years, for the patience and time he devoted throughout our cooperation, including his thorough corrections on this text and all papers published within those years.

Next, I would like to thank the other two members of the three-member Advisory Committee, Professor at NTUA, A. Boudouvis and Assistant Professor at NTUA, N. Lagaros, for the confidence they have bestowed upon me in trusting me with this dissertation, their apt observations on this work and their suggestions regarding its presentation.

I am also grateful towards the members of the research group, Dr. Varvara Asouti, Dr. Konstantinos Tsiakas, Dr. Xenofon Trompoukis, Konstantinos Samouchos, Dr. Evangelos Papoutsis-Kechagias, and to all the others I have been fortunate to collaborate with. They have contributed to the pleasant and friendly atmosphere and cooperation that prevailed daily. I would in particullar like to especially thank Dr. Varvara Asouti with whom I have worked closely throughout my work at PCOpt/NTUA. She has always been by my side, ready to provide me with all the knowledge needed for the completion of this thesis. I want to thank her warmly for all the time and support she has offered me. Furthermore, I would like to thank Dr. Konstantinos Tsiakas and Dr. Xenofon Trompoukis for their continuous assistance on any problem I encountered during my thesis.

At this point, I also have to mention the funding sources that supported parts of my thesis (a) the project "Design-Optimization of Diaphragm Pumps under Operational and Manufacturing Uncertainties based on the Cut-Cell and the Polynomial Chaos Methods" (MIS 5004541), implemented under the Action "Supporting Researchers with an Emphasis on Young Researchers", in the context of the call EDBM34, funded by the Operational Programme "Human Resource Development, Education and Lifelong Learning" (NSRF 20142020), (b) the European project RBF4AERO, "Innovative benchmark technology for aircraft design and efficient design phase optimization", Grant Agreement 605396, FP7 and (c) other projects running in the PCOpt/NTUA, such as a project funded by TITAN Cement Company and a recent one funded by Toyota Motor on Deep Neural Netwrorks.

Last but not least, I would like to thank my family and all the friends who encouraged and supported me during my PhD thesis.

Contents

Chapter 1

Optimization in Engineering Applications - EAs & Other Optimization Methods

Practically, in all branches of engineering, any design of devices and operations is susceptible to optimization. Civil engineering requires the design of more robust buildings, bridges, etc. with less materials. Electrical engineers wish to design computers and integrated circuits consuming less energy while performing faster. Engineers wish to design faster cars and airplanes consuming less fuel. For so many years, engineers were optimizing designs by trial-and-error or experience-based methods. However, recent advances in computing have made numerical optimization the dominant (most efficient and most effective) way to design products. Even though the optimization methods developed and assessed in this thesis may fit to any engineering sector, shape optimization problems in aerodynamic/fluid mechanics are exclusively exposed. These problems are associated with a Computational Fluid Dynamics (CFD) software which simulates the flow by solving the governing flow equations. The post-processing of the CFD simulations computes the objective and constraint functions values for the problem in hand. The evaluation of a candidate solution is costly even on modern multi-processor systems equipped with many CPUs and/or GPUs.

During the last two decades, Evolutionary Algorithms (EAs) successfully penetrated into the industries and have widely been used to solve a variety of optimization problems. In general, they can reach global optima and can easily accommodate any black-box problem specific model (PSM) without extra pain. They can handle both Single-Objective (SOO) and Multi-Objective (MOO) Optimization problem, being ideal for computing Pareto fronts of non-dominated solutions. They can also handle problems with any number of constraints. Over and above, they can easily be parallelized by performing simultaneous evaluations on multiprocessor systems. Their main drawback is that they likely need an excessive amount of calls to the PSM (i.e. the CFD software) resulting in high overall optimization cost. The cost becomes higher in problems with many design

variables, which is the case in a great number of modern engineering problems. Moreover, some characteristics of the problem in hand, such as the fact that the objective functions are rarely separable (the optimization of a separable problem is much easier) and the possible existence of many local optima within the search domain, further increase the optimization cost. This thesis focuses on methods which assist EAs to reduce their overall cost. Before, however, presenting material developed in this thesis, an overview of similar developments proposed by the Parallel Computational Fluid Dynamics & Optimization Unit of NTUA (PCOpt/NTUA), in the past, is necessary.

1.1 Developement of EAs in the PCOpt/NTUA

An overview of previous PhD theses on EAs which have been carried out in the PCOpt Unit of NTUA follows.

Giotis' thesis (2003) [57] developed a unified and generalized formulation of an EA for SOO and MOO problems. This EA keeps and constantly updates three population sets (offspring, parent and elite populations) and combines evolution operators and variables encoding techniques borrowed by Genetic Algorithms and Evolutionary Strategies. In [59, 52, 93], for the first time in the history of EAs, the dynamic cooperation of the PSM with a metamodel or surrogate model, (Artificial Neural Networks, ANNs, in specific) for the pre-evaluation of the candidate solutions was proposed to reduce the optimization cost. The new feature of this search algorithm, to be referred to as Metamodel-Assisted EA (MAEA), was the implementation of on-line trained metamodels, instead of off-line trained ones, as other researchers used to do by that time. At the beginning of each generation, the Inexact Pre-Evaluation, which will hereafter be referred to as the Low-Cost Pre-Evaluation (LCPE), of all individuals belonging to the current offspring population takes place and, subsequently, only the few most promising of them are evaluated on the PSM. Metamodels are trained on the fly before each pre-evaluation to keep training cost negligible and increase predictive capabilities. Further improvement of the metamodel prediction capabilities was achieved by introducing importance factors for every design variable in order to identify noisy or less important ones and downgrade their role during the training. The elapsed or optimization wall-clock time was also reduced using parallel processing with open message passing protocols (primarily PVM, being the standard protocol, by that time). Parallelization was based on the concurrent evaluations of offspring within the same generation. The benefits of the combination of metamodels, parallel processing and EAs was demonstrated in 2D airfoils and axial compressor blade airfoil shape optimization cases, using the ancestor of the CFD code used in the present thesis as evaluation tool. Giotis' PhD thesis concluded by the first release of the EASY optimization platform [49] which was the basis of (also, upgraded during) most of the next PhD theses.

Karakasis' thesis (2006) [91, 93, 92, 94] was mainly focused on how EAs or MAEAs can efficiently solve MOO problems. One of the most important outcome of this thesis was that the updated MAEA performed equally well for both SOO and MOO problems. The use of metamodels with high generalization ability, the careful selection of training patterns and the detection and special treatment of outliers, for which the accuracy of the performance prediction becomes questionable, improve the gain from the LCPE phase. The Radial Basis Function (RBF) networks, used as metamodels, were enhanced by techniques for the self-organizing selection of centres. The LCPE phase was extended to Distributed EAs (DEAs) in [93], according to a many-island model, with the aim of both improved design space exploration and efficient use of parallel processing systems. The availability of more than one evaluation tools, i.e. CFD solvers with different accuracy and computational cost, gave rise to Distributed Hierarchical EAs (DHEAs). In a two-level configuration, the low-level islands, by performing evaluations on the low-cost CFD tool, undertake the exploration of the design space and forward promising individuals to the high-level islands, utilizing the high-cost CFD tool, in order for them to be further evolved. The proposed methods offered a reduction in the computational cost by roughly one order of magnitude in comparison with standard EAs, without compromising the exploration effectiveness.

The third EA-oriented PhD thesis (Kampolis, 2009) [77, 81, 82, 55, 78, 117] upgraded the multilevel algorithm of the DHEA. Each level was associated with (a) an evaluation tool, (b) a search technique and/or (c) a parameterization scheme. On each level, the multilevel evaluation scheme resorted to different evaluation tools, the multilevel parameterization handles problem variants with different numbers of degrees of freedom (coarse and fine) and the multilevel search employed different optimization methods (gradient-based and bio-inspired heuristics). Support Vectors Machines (SVMs) [65] were used to classify the candidate solution as feasible or non-feasible, in constrained optimization problems. The metamodels, RBF and Multi-Layer Perceptrons were trained on individuals that exclusively reside within the feasible region of the design space. A new class of metamodels, able to take both objective functions and their gradients into account was proposed; these possess superior generalization abilities, compared to conventional metamodels. Lastly, three types of multiprocessing for the aforementioned algorithms were tried: (a) the concurrent evolution of each level (and all demes within each level) using threads, (b) the concurrent evaluation of more than one candidate solutions during the evolution of the multilevel algorithm and (c) the parallel execution of the evaluation software of each candidate solution.

Later on, Kyriacou's thesis (2014) [112], funded by the EU project *"HYDROACTION-Development and laboratory testing of improved action and Matrix hydro turbines designed by advanced analysis and optimization tools"* was implemented partially in NTUA and Andritz Hydro Linz, and contributed to the even greater penetration of EASY into a turbomachinery industry. This thesis' main contribution was the

introduction of the Principal Component Analysis (PCA) within an EA or MAEA. The linear variant of the PCA assisted the EAs or MAEAs, during the application of evolution operators and the training phase of metamodels. Kyriacou's thesis prepared the ground for the present thesis, in this respect, a central part of which is the use of the Kernel variant of PCA, for similar purposes. The PCA is also used in different ways to improve the EAs performance, as shown in the following chapters.

Lastly, three other relevant theses from the PCOpt/NTUA should be reported; though not exclusively focused on EAs, they significantly contributed to them and the further development of the EASY platform. In Asouti's thesis (2009) [5, 6, 8], a new way for increasing the parallel efficiency of EAs or MAEAs was proposed. The proposed Asynchronous EAs (AEAs) or MAEAs (AMAEAs) [5] operates with a number of strongly interconnected demes, the structure of which determines the communication between demes and the implementation of evolution operators. The lack of generations removes synchronization problems and allows the maximum utilization of all available processors. Georgopoulou's thesis (2009) [47, 45, 43, 46, 44] proposed the so-called Metamodel-Assisted Memetic Algorithm (MAMA). Memetic algorithms (MA) are hybrid optimization algorithms that combine global and local search, aiming at improving the quality of promising candidate solutions. The proposed MAMA combined MAs with metamodels which undertake a dual role: pre-evaluation of the MA population during the global search and approximation of the objective functions gradients during the gradient-based refinement of promising solutions. Finally, Kontoleontos' PhD thesis (2012) [105, 106] focused, among other, on the combined use of an Asynchronous Evolutionary Algorithm (AEA) together with a gradient-based method, giving rise to a new asynchronous metamodel-assisted memetic algorithm (AMAMA). In MOO problems solved using MAs, a new scheme for the gradient computation, according to which the adjoint equations are solved only once, instead of as many times as the objectives, was proposed in order to further reduce the CPU cost. This scheme is based on the synthesis of the objectives into a scalar function, scaled by properly computed coefficients. The developed method was used to optimize energy production systems, such as geothermal power plants and a ground source heat pump system. From all the aforementioned methods improving the EA performance, only the hierarchical and asynchronous search methods are not revisited, updated or utilized, in this thesis.

1.2 Development of Gradient-based Optimization Methods in the PCOpt/NTUA

A brief review of gradient-based optimization methods follows, to complete the picture of optimization methods developed by the PCOpt/NTUA Unit and which are used, even partially, in this thesis. A part of this thesis is about the hybridiza-

tion of EAs and gradient-based methods, in aerodynamic shape optimization, by profiting of the (continuous) adjoint methods also developed by the same group. In contrast to EAs, the gradient-based methods require less computational cost but may be entrapped into local optima(um). So, a hybrid optimization algorithm combining and coordinating both methods presents great advantages.

During the last years, the PCOpt/NTUA Unit was/is developing adjoint-based methods for computing first- and higher-order derivatives of objective functions used in aerodynamic optimization. The computational cost for computing the gradient of a function is more or less equal to that of solving the flow PDEs and independent of the number of optimization/design variables. Both discrete and continuous adjoint approaches have been developed on the in-house GPU-enabled CFD solver named PUMA, the OpenFOAM flow solver and the in-house CFD solver based on the cut-cell method. In this thesis, PUMA [173, 9, 172] developed during Trompoukis' and Tsiakas' theses [174, 171], provide the required derivatives for the hybrid algorithm presented in a different chapter. The cut-cell CFD solver developed during Samouchos' thesis [150] provides the derivatives in the optimization of a valveless diaphragm micropump presented in chapter 6. In Zervogiannis' thesis [182], the discrete adjoint method was developed. Till recently, two different mathematical approaches were available for the continuous adjoint method. The first one, developed in Papadimitriou's PhD thesis [58], expresses the gradient in terms of only boundary/surface integrals (surface integral or SI adjoint). It is computationally cheap but, as shown later, depending on the grid, may compute inexact gradients. The second one expresses the gradient in terms of both surface and field integrals (Field Integral or FI adjoint); it is accurate, though computationally much more expensive. A new enhanced surface integral formulation (E-SI) assisted by the adjoint to a grid displacement model, which is both accurate and cheap as it is free of volume integrals, has been recently proposed in Kavvadias' PhD [96, 97]. Usually, in the literature, "frozen turbulence" (assuming that turbulent quantities remain unaffected by the changed optimization shape) is employed along with the continuous adjoint method. In Zymaris' thesis [187, 131], the turbulence model equations were differentiated for the first time in continuous adjoint, showing that the "frozen turbulence" assumption can lead to even wrongly signed sensitivities. In the sake of completeness, it must be reported that the PCOpt/NTUA Unit contributed to CFD-based optimizations in three subjects: (a) the efficient but flexible and automatic parameterization of arbitrary shapes, (b) the development of 'back-to-CAD' processes to deliver the optimized geometries in CAD-compatible formats and (c) the imposition of design constraints.

1.3 EAs-based Optimization and Parallel Processing

During the aerodynamic shape optimization problems this thesis is dealing with, each evaluation may cost even more than a day, on a single computing unit (typical example is the analysis of 3D turbulent flows, around or in complex geometries). Even if the evaluation tool runs in parallel on many computing units, the evaluation cost may rise up to several hours. Thus, it is important for the optimization method to be susceptible to parallelization. Fortunately, this is the case for EAs, since their population members can simultaneously be evaluated on different computing units. The parallelization is applied without modifying the evaluation tool. Practically, in computationally demanding problems with small cost of transferring files which might be necessary for the evaluation, it may induce ideal parallel performance (linear cost reduction). This occurs only if the available computing units are continuously fed with population member to be evaluated. To this end, for an aerodynamic optimization problem solved using EAs, the calls to the evaluation tool are parallelized and the evaluation (PSM or CFD) tool is also executed in parallel on different computing units, reducing as much as possible the wall-clock time of the optimization.

During this thesis, the majority of the performed optimizations are large scale ones and, thus, computationally demanding. For this reason, these have been performed on the "VELOS" platform of the PCOpt/NTUA Unit which comprises two clusters with 72 Teraflop computing power in total. The clusters are composed by a number of computing units (nodes) communicating via the TCP/IP protocol. The first cluster has 70 CPU computing nodes and the second one has 10 Graphics Processing Units (GPUs) computing nodes based on NVIDIA cards. The necessary services required for communication and data storage are handled by two dedicated servers, with the Network Information Service (NIS, holding the necessary user information) and Network File System Service (NFS, for file sharing). The computationally consuming numerical simulations employ the MPI protocol, according to the MIMD (Multiple Instruction Multiple Data) model. Parallel processing with the SIMT (Single Instruction Multiple Thread) model is performed on GPUs.

All the following optimizations take advantage of both the EA and CFD solver's parallelization. The CFD solver, used during the whole thesis and described briefly in chapter 2.4, is GPU-enabled and runs in parallel on the PCOpt/NTUA GPU cluster. Regarding the EAs, during each generation, the evaluation of population members is carried out in parallel since each of them runs on a different GPU unit and/or node. The gain in performance due to the parallelization is not included that much during the presentation of results since relevant new contributions or developments are rather limited. Nevertheless, parallelization is implicit within this thesis.

1.4 Thesis' Contributions

A short overview of this PhD thesis' contributions follows:

1) Kernel PCA driven EAs and MAEAs: The PCA is incorporated into EAs and MAEAs to reduce their computational cost, particularly in complex problems with many design variables. By extending the method proposed in Kyriacou's thesis [112], the Kernel (instead of the Linear) variant of the PCA transforms the initial design space into a new feature space, in which the unknowns are as separable as possible. The evolution operators perform optimally in this separable feature space improving the optimization convergence speed. Moreover, the metamodels efficiency remains very high even in problems with many design variables. The PCA, used as a dimensionality reduction tool, decreases the input units of the patterns the metamodels are trained on, leading to lower training cost and better prediction ability.

2) PCA-Assisted Hybrid Optimization Algorithm: A PCA-Assisted hybrid optimization algorithm which combines the EAs for the exploration of the design space and gradient-based methods for the exploitation of the obtained information is proposed for solving MOO problems. During the optimization, all the EAs population members evolve through the application of the standard evolution operators, apart from a few most promising individuals which are updated using the gradient-based method. The required gradients of the objective functions are computed by the continuous adjoint method. In MOO problem, the individuals to be updated using the gradient should improve the front of non-dominated solutions by descending along the so-called Pareto Advancement Direction (PAD); the latter is computed by appropriately concatenating the gradients of all the objective functions. It should also be "perpendicular" to the current front, to achieve the simultaneous minimization of all the objectives. The hybrid algorithm proposed in Kampolis' PhD [77] was computing the PAD based on the SPEA method. Herein, the Linear PCA method computes the PAD, making the algorithm independent from the Strength Pareto Evolutionary Algorithm (SPEA) method [185].

3) Multi-Criteria Decision Making assisting EAs: Multi-Criteria Decision Making (MCDM) techniques are introduced into the EA used in MOO problems. After the EAs optimization, the Pareto front is usually presented to the Decision Maker (DM), who, based on some not previously articulated preferences and MCDM techniques, selects the most appropriate among the computed non-dominated solutions. In this thesis, without loss of generality, the TOPSIS [72] technique is used. TOPSIS may assist EAs either 'a-posteriori' (when the DM's preferences are known after the EAs termination) or 'a-priori' (preferences known before running the optimization). In the former, TOPSIS is applied on the computed front of non-dominated solutions to select the preferred solution while, in the latter, it guides the evolution and computed fronts in the preferred areas of the objective space.

4) Flow Prediction using Deep Neural Networks: Herein, specifically modified

networks, based on the Deep Neural Networks (DNN), are utilized to predict the flow field around aerodynamic bodies. The purpose of this study is to build surrogate evaluation models able to replicate the expensive CFD tools with low evaluation cost during design/analysis procedures or optimizations. The DNNs are trained on a database of CFD solutions and, then, can predict the flow field around newly presented designs, such as airfoils and wings. Convolutional Neural Networks, a DNNs sub-category, are mainly utilized due to their ability to handle large amount of data. They are also flexible with architectures easily adjusted to each case in hand. Regarding the flow field representation, two methods are used. The first one (recommended for 2D cases) is based on images which are given as inputs and outputs and are handled optimally by the DNNs. The second one (recommended for 3D cases) expresses the inputs and outputs as raw data, making it possible to handle of 3D flow fields. Studies of flows around airfoils, wings and in turbomachinery cascades are performed.

The aforementioned methods are demonstrated in a number of benchmark and industrial aerodynamic shape optimization problems, which rely on a GPU-enabled in-house CFD solver for the flow simulations. Only a single industrial case studied is using the in-house CFD solver based on the cut-cell method.

Chapter 2

Evolutionary Algorithms (EAs)

2.1 Definition of the Optimization Problem

An optimization problem with M_o objective functions to be minimized (cast into the objective functions vector $\vec{f}$) and M_c constraints ($\vec{c}$) is defined as:

$$min \vec{f}(\vec{b}) = (f_1(\vec{b}), \ldots, f_{M_o}(\vec{b})) \in \mathbb{R}^{M_o} \quad (2.1.1)$$

$$subject\ to\ c_k(\vec{b}) \leq 0, \quad k = 1, M_c \quad (2.1.2)$$

where $\vec{b} \in \mathbb{R}^N$ (N is the number of design variables) is the vector of design variables or design vector. The design variables determine the design space restricted by the vectors of upper and lower bounds $\vec{U}, \vec{L}$. Note that, in this PhD thesis, all optimization problems are expressed as minimization problems.

In contrast to a SOO, which handles only one objective, in MOO, the EA has to handle a vector of objective functions values (objective vector $\vec{f}$). There are several techniques available in the literature [124, 19, 20] to transform the objective vector to a scalar utility value. Among them, the most commonly used techniques, such as the SPEA and NSGA, likely in their most recent forms [185, 184, 186, 25], transform the objective function values into a single value by using either weights or criteria based on distances and/or ranking. In this PhD thesis, techniques based on the Pareto dominance criteria [183] are exploited for solving MOO problems. Some basic definitions regarding Pareto dominance follow:

Weak Pareto Dominance: Individual $\vec{b}^1$ weakly dominates individual $\vec{b}^2$ ($\vec{b}^1 \preccurlyeq \vec{b}^2$) if and only if $\vec{f}(\vec{b}^1)$ is partially less than $\vec{f}(\vec{b}^2)$, i.e.

$$\vec{b}^1 \preccurlyeq \vec{b}^2 \Leftrightarrow (\forall i \in [1, M_o] : \vec{f}_i(\vec{b}^1) \leq \vec{f}_i(\vec{b}^2)) \wedge (\exists\ \vec{f}_i(\vec{b}^1) < \vec{f}_i(\vec{b}^2)), \quad \vec{b}, \vec{b}^1 \in [\vec{L}, \vec{U}] \quad (2.1.3)$$

Strong Pareto Dominance: Individual $\vec{b}^1$ strongly dominates $\vec{b}^2$ ($\vec{b}^1 \prec \vec{b}^2$) if

and only if $\vec{f}(\vec{b}^1)$ is less than $\vec{f}(\vec{b}^2)$, i.e.

$$\vec{b}^1 \prec \vec{b}^2 \Leftrightarrow (\forall i \in [1, M_o] : \vec{f}_i(\vec{b}^1) < \vec{f}_i(\vec{b}^2)), \quad \vec{b}, \vec{b}^1 \in [\vec{L}, \vec{U}] \tag{2.1.4}$$

Pareto Optimality: Individual $\vec{b}^1$ is a Pareto optimal individual if and only if there is no other $\vec{b}$ that dominates it, i.e.

$$\nexists \vec{b} : \vec{b} \prec \vec{b}^1, \quad \vec{b}, \vec{b}^1 \in [\vec{L}, \vec{U}] \tag{2.1.5}$$

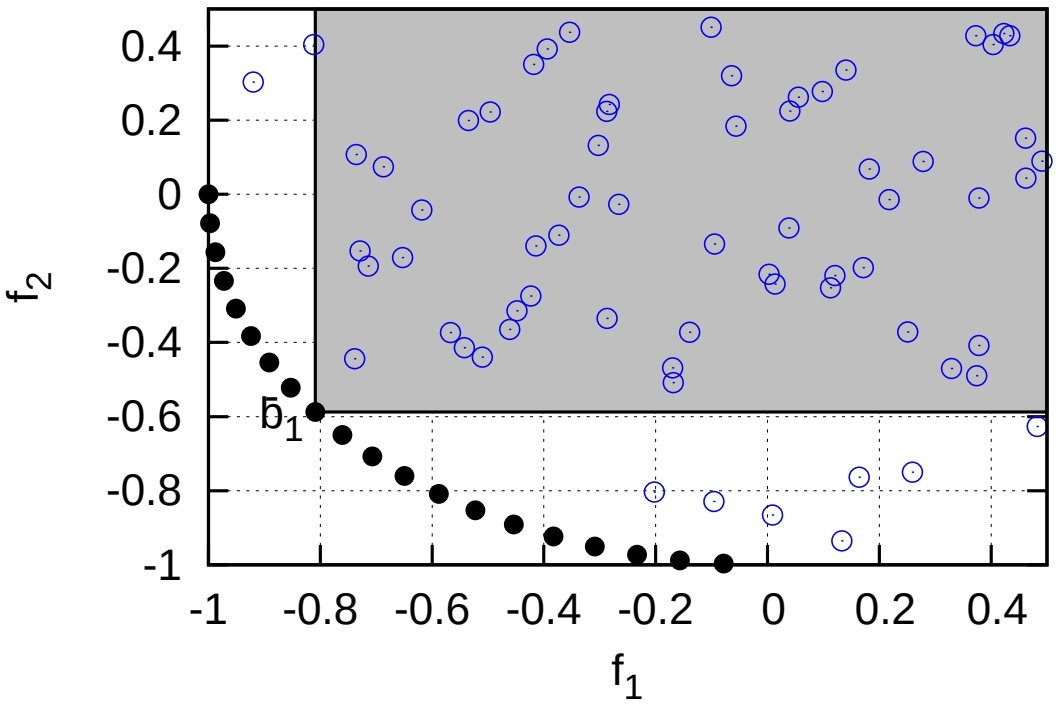


Figure 2.1: Pareto dominance in a two-objective minimization problem. Pareto optimal individuals are marked with filled circles to distinguish them from the dominated ones plotted as empty circles. A Pareto optimal individual $\vec{b}^1$ dominates all individuals inside the grey box.

Fig. 2.1 gives a schematic representation of the Pareto optimality in a two-objective minimization problem (min. f_1 and min. f_2). Engineering optimization problems must also satisfy constraints which divide the objective space into feasible and infeasible regions. Many methods have been developed for EAs to handle constraints. Some of them are adding a penalty to the objective function values whenever constraints are violated [24, 125]. Others are adjusting the surviving

possibility of individuals in the infeasible area of the objective space by adapting appropriately the evolution operators [135]. Alternatively, constraints can also be handled as extra objective functions to be minimized [166]. As described in section 2.3.4, this PhD thesis uses penalty functions in all constrained optimization problems.

2.2 Overview of EAs

Several stochastic optimization methods are based on the Darwin's theory of evolution regarding the origin of species [23]. Over the years, many variants, which herein are generally referred to as Evolutionary Algorithms (EAs) [123], have been devised and used for all kinds of optimization problems. Industrial use of EAs has been increasing due to the increasing power and availability of modern High-Performance Computers (HPCs). EAs combined with existing evaluation tools can efficiently be used as industrial design tools. In all the aerodynamic shape optimization problems presented in this PhD thesis, solved using EAs, the evaluation tool or problem-specific model (PSM) is a Computational Fluid Dynamics (CFD) solver/software. The overall optimization time/cost needed by the EAs to solve a CFD-based optimization problem is the cost of a single call to the CFD solver times the number of evaluations required to obtain the optimal solution. Note that EAs have the option of running evaluations in parallel, leading to a reduction in the optimization wall clock time with the same CPU cost, though.

During mid-50's, Friedberg, Bremermann and Box laid the ground for EAs as problem solving techniques. EAs were used by Bremermann [14] to solve numerical (linear, convex) optimization problems including systems of nonlinear equations. Box [12, 13] initiated the industrial use of EAs for optimizing both processes and productivity. After these first steps, three broad categories of EAs came up; these are known as Evolutionary Programming (EP) (an ancestor of EP, created by Friedberg [39, 40]), Evolutionary Strategies (ES) and Genetic Algorithms (GA).

During mid-60's, Fogel [35, 36, 38] developed the first EP, based on machine learning tasks by means of finite-state machines (FSM). FSM is a mathematical model for computations. It can be in exactly one of a finite number of states at any given time. External inputs (similar to the design variables) can change the FSM's state, causing a transition. The optimization method proposed by Fogel was an algorithm evolving with the objective to predict the optimal time series of the phenomenon under consideration. During a generation of EP, parents are selected among the previous generation offspring based on their fitness values and new offspring are created by randomly mutating the parents. Problems regarding predictioning models, automatic control and pattern recognition were successfully optimized using EP. Later on, EP was extended and applied in problems with continuous functions and real-valued design vectors. A self-adaptive EP including mutation variance in the evolution was proposed in [37]. Currently, EP is a wide

evolutionary computing dialect with no fixed structure or representation, thus it could hardly be distinguished from ES.

The ES were firstly developed in 1965 by Renchenberg [138], using discrete, binomially distributed mutation, a single offspring and a single parent per generation. Later, both Renchenberg [139] and Schwefel [158] upgraded ES by utilizing two evolution operators, namely mutation and recombination. The mutation operator is performed by adding a normally distributed random value to the design variables, represented with real values. The most popular ES is the covariance matrix adaptation ES (CMA-ES) [64], which introduced the recombination operator and proved useful for solving non-separable optimization problems. The addition of recombination led to the multi-membered ES with μ parents and λ offspring, referred to as a (μ, λ) ES.

In 1962, Holland proposed the GA [71, 70] as an evolution emulating tool for better understanding adaptive systems. Adaptive systems are capable of modifying their response according to the interactions with their surrounding environment. Holland ideas were based on the well known genetic operators, such as mutation, crossover/recombination and inversion. The evolving design variables were represented as binary strings, similar to the genetics genome (genes). This binary representation was the main difference between ES and GA.

The establishment of EAs in the scientific community allowed the development of Genetic Programming (GP). GP was used to encode computer programs as sets of genes and evaluate their fitness in performing a predefined task. These computer programs were evolved using GP-based logic so as to perform best on the given task. Cramer [22] introduced the "tree-based" GP, which was further improved by Koza [107]. In this "tree-based" GP, computer programs are represented as tree structures where each node is an operator function and each terminal node an operand. This representation offers easy application of the mutation and recombination operators. Replacing nodes at random practically stands for the mutation operator, whereas exchanging nodes among individuals stands for recombination one.

The standard EA is based on all the aforementioned methods by taking and combining some of their characteristics. Basically, EA is a generalized GA and ES; an EA becomes GA or ES by properly configuring it. An EA handles populations, in which each member/individual represents a candidate solution. The individuals genes are encoded using binary or real coding. Each individual is associated with a fitness value. Populations are updated via the application of evolution operators. Among other, these include parent selection, crossover, mutation and elitism. The continuing evolution of the populations leads to the optimal solution(s). Standard textbooks on EAs are the books of Goldberg [60] and Michalewitz [122].

This thesis mainly focuses on the (μ, λ) EA [57, 91, 77, 5, 112], which is a generalized scheme able to represent both GA and ES. The (μ, λ) EA is a population-based algorithm which evolves individuals based on their fitness. The latter

stands for the ability of each individual to survive in a given environment and is determined by its objective function value(s). Three populations are handled in each generation, namely the elite, parent and offspring populations. In each generation, parents are generated via the parent selection and elitism operators applied to the offspring and elite populations of the previous generation. Then, parents generate offspring via the application of crossover and mutation. In each generation, the elite population holds the fittest individuals found thus far during the evolution.

EAs have the advantage of being able to find the global optimum(a) with a finite number of calls to the PSM. The PSM can be any ready-to-use software, which takes the values of the design variables as input and returns the objective function values as output. Any software (even a commercial one) can be used as a black-box tool for the EA, without having access to its source code. On the other hand, relatively many calls to the PSM are required for the EA to converge to the global optimum(a). As a result, the optimization turn-around time/cost can be prohibitively large for industrial use. This is the case with the CFD-based optimization problems this PhD thesis is dealing with. Several methods have been proposed as a remedy to this problem giving rise to different variants of EAs [48, 32, 73].

2.3 The Evolutionary Algorithm SYstem (EASY)

The EASY platform [91, 77, 57] is the basic software for the development and application of all the methods presented in this PhD thesis. EASY is a generic optimization platform with many different optimization algorithms. EASY supports MAEAs with different metamodel types depending on the way they are trained. Other methods to enhance EAs, such as distributed, asynchronous and hierarchical EAs (possibly combined together), are available in the platform; they are analysed in the following sections. EASY is also Grid/Cluster-computing enabled and can perform parallel evaluations in different High Performance Computers [91]. A detailed analysis of the (μ, λ) EA applied in EASY, which consists the standard EA within this thesis, is described below.

2.3.1 The (μ, λ) EA

The implemented (μ, λ) EA handles three different populations in each generation, namely the offspring P_λ^g population with λ individuals, the parent P_μ^g population with μ individuals and the elite P_e^g population with ε individuals. Superscript g stands for the generation counter. Design variables can be encoded in binary, binary Gray or real. Each coding is associated with its own evolution operators. The basic nomenclature of the standard EA is presented in table 2.1.

The standard (μ, λ) EA performs the following steps:

Number of design variables	N
Number of objective functions	M_o
Number of constraints	M_c
Design vector	$\vec{b}$
Objective vector	$\vec{f}$
Utility function	ϕ
Constraint vector	$\vec{c}$
Generation counter	g
Number of offspring	λ
Number of parents	μ
Number of elites	ε
Offspring population	P_{λ}^{g}
Parent population	P_{μ}^{g}
Elite population	P_{e}^{g}

Table 2.1: Standard EA parameters and terminology.

Initialization: First generation ($g = 0$). The offspring population P_{λ}^{0} is initialized using a random number generator (RNG). This generator creates design vectors with values within the user-defined lower ($\vec{L}$) and upper ($\vec{U}$) bound for each design variables. An RNG seed is used to "warm up" the generator, different seeds can lead to different initializations and, thus, conclude to different convergence histories of the optimization runs. Moreover, the user may inject a number of design vectors (the current design to be optimized or other well performing individuals he/she might have access to, for any reason) to initialize (part or whole) population with predefined design vectors.

Evaluation: All offspring (P_{λ}^{g}) are evaluated on the possibly, expensive, PSM, so as to pair the design vector ($\vec{b} \in \mathbb{R}^{N}$) with the corresponding objective vector ($\vec{f} \in \mathbb{R}^{M_o}$). The evaluated individuals are stored in a database (DB) to avoid repeating the same evaluation if, in a subsequent generation, the same individual comes up. Evaluation is by far the most time consuming part of the algorithm, especially when the PSM is a CFD solver, which is the case in this PhD thesis.

Fitness Assignment: A utility function ϕ is assigned to each individual $\vec{b} \in P_{\lambda}^{g} \cup P_{\mu}^{g} \cup P_{e}^{g}$. In SOO, ϕ is equal to the objective function value (f). In MOO, ϕ is computed based on a scalar utility function defined by Pareto dominance techniques, section 2.3.3.

Elite Selection: The elite set of the current generation (P_{e}^{g}) is formed from the non-dominated individuals of $P_{\lambda}^{g} \cup P_{e}^{g-1}$. If these exceed than the predefined (user-defined) maximum number ε, then a trimming process selects and

retains ε of them based (usually) on Pareto front thinning criteria.

Elitism Operator: The few best (in terms of the objective functions) elite individuals replace some offspring in P_{λ}^{g}.

Parent Selection: The parent population P_{μ}^{g} is formed via the $P_{\lambda}^{g} \cup P_{\mu}^{g-1}$ individuals. The selection is based on their fitness value; fitter individuals have greater probability to be selected as parents. Parent selection is symbolically presented as $P_{\mu}^{g} = S(P_{\mu}^{g-1}, P_{\lambda}^{g})$.

Crossover Operator: The new offspring population P_{λ}^{g+1} is created by the crossover operator applied to the parent population P_{μ}^{g}. The operator is used λ times per generation, one for each offspring individual. $\rho(\geq 2)$ parents are selected with a certain probability (lower fitness value leads to a higher probability) and their genotypes/ design variables are combined in different ways (see section 2.3.2) to produce a new offspring.

Mutation Operator: New offspring undergo mutation with a user-defined small probability. The mutation operator is interfering with the design variables of the individuals by slightly changing them (see section 2.3.2).

Stopping Criteria: Stopping criteria, such as the number of total evaluations on the PSM/generations and/or the maximum allowed number of idle evaluations are checked. In case these are not yet met, the algorithm returns to the **Evaluation** step.

2.3.2 Evolution Operators

As mentioned before, the evolution operators used by EASY are the elitism, parent selection, crossover and mutation, which are discussed below.

Elitism: The application of elitism guarantees the monotonic convergence of the evolution over the generations. The elite population P_{e}^{g} maintained in EASY also guarantees that the best solutions found are retained. Elitism promotes a certain (user-defined) number of elites to the offspring population P_{λ}^{g} by removing the worst individuals found in it.

Parent Selection: The parent selection operator generates new parents P_{μ}^{g} by selecting individuals from the $P_{\mu}^{g-1} \cup P_{\lambda}^{g}$ populations. The most common parent selection techniques are:

Proportional Selection: All the available individuals of $P_{\mu}^{g-1} \cup P_{\lambda}^{g}$ are associated with a probability to become parent, which is proportional to their fitness value ϕ. Smaller ϕ corresponds to greater selection probability. Then, a roulette wheel is formed by pairing each individual with

one slot, the angular width of which is proportional to the individual's selection probability. To select μ parents, μ turns of the roulette wheel are performed.

Linear Ranking: The individuals of $P_{\mu}^{g-1} \cup P_{\lambda}^{g}$ are sorted based on their ϕ values. The probability of selecting one of them is based on its rank in a linear manner.

Probabilistic Tournament Selection: In this technique [61], ρ different individuals are randomly selected from $P_{\mu}^{g-1} \cup P_{\lambda}^{g}$. The best individual among them is selected as a parent based on a user-defined probability, otherwise another randomly selected individual becomes a parent. This process is repeated μ times. ρ is a user-defined parameter, typically equal to 2 or 3, and the tournament probability (for selecting the best individual as parent) is suggested to be greater than 80%.

Crossover: Crossover is the most basic operator since the introduction of EAs. Its purpose is to appropriately combine parents to yield offspring having an increased probability to perform better than its parents. Many crossover schemes [151, 152, 165] have been proposed over the years with their main classification depending on the design variables coding.

If binary coding is used, then crossover exchanges bits of binary strings (binary representation of the design vector) among the parents so as to produce the new offspring. The EASY platform is equipped with the following crossover operators for binary coding:

One-Point Crossover: The binary string of the parents is divided into $\rho-1$ parts with the same number of digits, when the one-point crossover is to be applied. From the P_{μ}^{g} population, ρ parents are selected and they form $\rho-1$ pairs $(1,2),\ (1,3),\ldots,(1,\rho)$, all of them containing the first parent. A random integer defines the crossover point which divides the parents into two parts. Each offspring is formed by combining the one (left) part of the first parental string and the second (right) part of the other parent, according to the pairing. A three-parent example ($\rho=3$) is shown in fig. 2.2.

Two-Point Crossover: This scheme is similar to the one-point scheme, the only difference being the use of two crossover points. A three-parent example ($\rho=3$) is shown in fig. 2.3.

One- or Two-Point Per Design Variable Crossover: The one- and two-point schemes are applied to the parts of the binary string corresponding to each design variable, separately.

The real coding crossover schemes are applied directly to the design variables (real values). The EASY platform is equipped with the following real coding crossover operators:

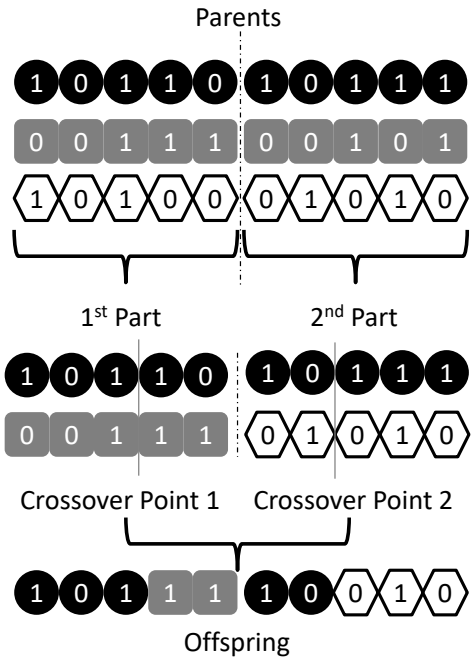


Figure 2.2: One-Point Crossover for binary coding with $\rho = 3$. Parents' binary strings are divided into two parts. The offspring is constructed by parts of the split parents.

One- or Two-Point Crossover: This scheme is inspired by the corresponding variants for binary coding. Instead of exchanging binary pieces, the crossover exchanges design variables among the selected parents. One or two randomly selected crossover points correspond to design variables. The offspring is produced by design variables from any of the parents. Variables corresponding to the crossover points are affected by both parents based on randomly defined weights. An example can be seen in fig. 2.4.

Discrete Crossover: In discrete crossover, ρ parents are selected and each design variable of the resulted offspring has 50% probability to be the same with the first parent and $50/(\rho - 1)\%$ probability to be the smae with any of the remaining $\rho - 1$ parents. An example can be seen in fig. 2.5.

Intermediate Crossover: In this scheme, ρ parents are selected, similarly with the other schemes. Each design variables ($\vec{b}$) of the resulted offspring is a linear combination of the first ($\vec{b}^1$) and another randomly

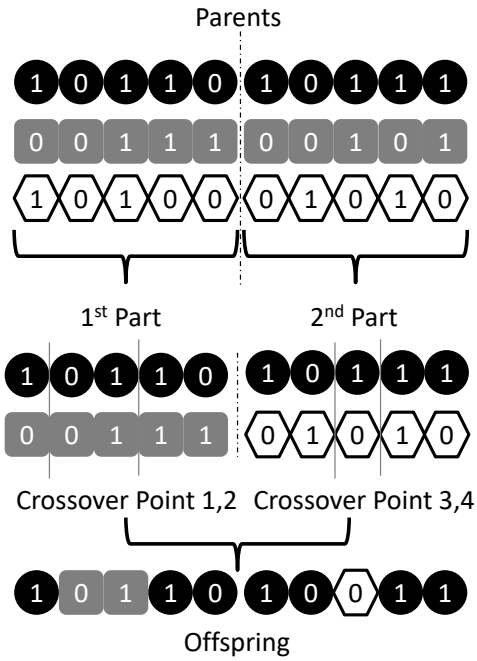


Figure 2.3: Two-Point Crossover for binary coding with $\rho=3$. Parents' binary strings are divided into two parts. These are, then, randomly split into three parts (by two crossover points). The final offspring is constructed by parts of the split parents.

chosen ($\vec{b}^{random}$) parent, mathematically expressed as

$$\vec{b} = \vec{b}^1 + r(\vec{b}^{random} - \vec{b}^1),\ r \in [0,1] \tag{2.3.2.1}$$

where r is a random number (the same exact for all design variables) uniformly distributed between 0 and 1.

Simulated Binary Crossover: This scheme [26] aims at recreating the one-point binary crossover by maintaining two basic properties, namely average and spread factor. The average property implies that decoded variables before and after crossover have the same average value. The spread factor (β) is defined as the ratio of the spread (how well the population covers the design space) of offspring to that of parents. $\beta < 1$ corresponds to contracting crossover, $\beta > 1$ to expanding and $\beta = 1$ to stationary. The spread factor property implies that the spread factor will be closer to 1 in most cases. Taking these into consideration, the

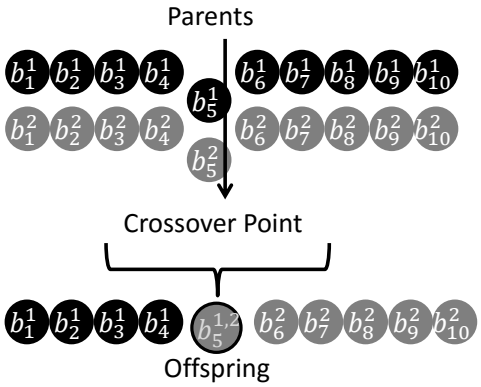


Figure 2.4: One-Point Crossover for real coding with $\rho = 2$ parents. The parents are split into two parts. The offspring is formed by the first part of the first parent and the second part of the second parent. The design variable of crossover point is computed as $\vec{b}_5^{1.2} = \vec{b}_5^1 + r(\vec{b}_5^2 - \vec{b}_5^1),\ r \in [0, 1]$.

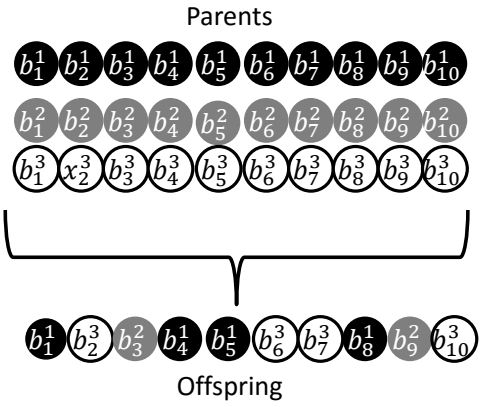


Figure 2.5: Discrete Crossover for real coding with $\rho = 3$ parents.

design vector $\vec{b}$ of the new offspring is computed as

$$\vec{b} = \begin{cases} \vec{b}^{middle} - \frac{\beta}{2}(\vec{b}^{random} - \vec{b}^1), & \text{if } r < 0.5 \\ \vec{b}^{middle} + \frac{\beta}{2}(\vec{b}^{random} - \vec{b}^1), & \text{otherwise} \end{cases} \tag{2.3.2.2}$$

where $r \in [0,1]$, $\vec{b}^1$, $\vec{b}^{random}$ and $\vec{b}^{middle}$ are the first, randomly selected and middle (between the other two) parents respectively. The spread factor β depends on the r values, if $r < 0.5$ then $\beta = (2r)^n$ else $\beta = (1/2r)^{n+2}$.

Mutation: The mutation operator is used to maintain diversity in the population and to explore the design space. It prevents the premature convergence of the population in a specific area of the design space and, thus, provides a mean to avoid stagnation of the evolution. For each offspring generated by the crossover operator, a random number is generated. If this number is smaller than a small user-defined mutation probability, usually if $P_m < 10\%$, the offspring is mutated. Similarly to the crossover operator, mutation can be performed using different schemes based on the design variables' coding.

For binary coding, each bit of the binary string representing the offspring is inverted (bit flip, from 0 to 1, or vice-versa), with a probability P_m (fig. 2.6).

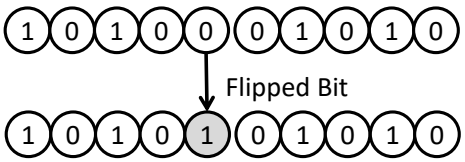


Figure 2.6: Binary mutation.

For real coding, each design variable (real value) is mutated as

$$\vec{b}^m = \begin{cases} \vec{b}, & \text{if } P_m > r \\ M(\vec{b}, D), & \text{otherwise} \end{cases} \tag{2.3.2.3}$$

where $r \in [0,1]$ is unique for all design variables and

$$M(\vec{b}, D) = \begin{cases} \vec{b} + D(g, \vec{U} - \vec{b}), & \text{if } r_1 > 0.5 \\ \vec{b} + D(g, \vec{b} - \vec{L}), & \text{otherwise} \end{cases} \tag{2.3.2.4}$$

where $r_1 \in [0,1]$ is another randomly generated number, $D(g,y) = y r_2 (1 - g/g_{max})^{0.2}$ with g_{max} the maximum number of generations and r_2 a randomly generated number. If the maximum number of generations is not a priori known, the total number of evaluations E_{max} and the current number of evaluations E can replace g_{max} and g, respectively.

2.3.3 MOO in EASY

In MOO, the objective vector ($\vec{f}$) have to be transformed into a utility function ϕ with the assistance of a scalar utility function $\phi(\vec{b}) = \phi(\vec{f}(\vec{b}))$, $\mathbb{R}^{M_o} \rightarrow \mathbb{R}^1$. EASY is equipped with the SPEA [185], SPEA2 [184] and NSGA2 [25] techniques presented below:

SPEA: SPEA (Strength Pareto Evolutionary Algorithm) was introduced in 1998 by Zitzler and Thiele [185] for assigning scalar values based on the Pareto dominance. Firstly, the strength of each individual of $P = P_{\mu}^{g} \cup P_{\lambda}^{g} \cup P_{e}^{g-1}$ is computed. The i^{th} individual's strength (S_i) is defined as the number of P members dominated by the aforementioned individual divided by the population size,

$$S_i = \frac{\sum(j : j \in P \wedge \vec{b}^i \prec \vec{b}^j)}{\sum P}, \forall i \in P \tag{2.3.3.1}$$

Then, the scalar value ϕ for each individual is calculated as the sum of the strengths of individuals dominating it, fig. 2.7,

$$\phi_i = \sum_{j \in P \wedge \vec{b}^j \prec \vec{b}^i} S_j \tag{2.3.3.2}$$

SPEA2: SPEA2 [184] was a further improvement to the initial SPEA by including density information. A more precise guidance of the exploration of the EA is achieved by using a nearest neighbor density estimation technique. The density of the individuals in the front of non-dominated solutions is improved compared to that resulting with the use of SPEA. In SPEA2, the strength functions are computed similarly to SPEA. After that, the density metric D_i for each individual is computed as its Euclidean distance from its closest neighbor in the objective space, $d_i = min(\|\vec{f}_i - \vec{f}_k\|),\ k \in P$:

$$D_i = \frac{1}{d_i + 2} \tag{2.3.3.3}$$

The raw cost R_i is computed similarly to the utility function of SPEA

$$R_i = \sum_{j \in P \wedge \vec{b}^j \prec \vec{b}^i} S_j \tag{2.3.3.4}$$

Finally, for any individual, the utility function ϕ is the sum of its raw cost

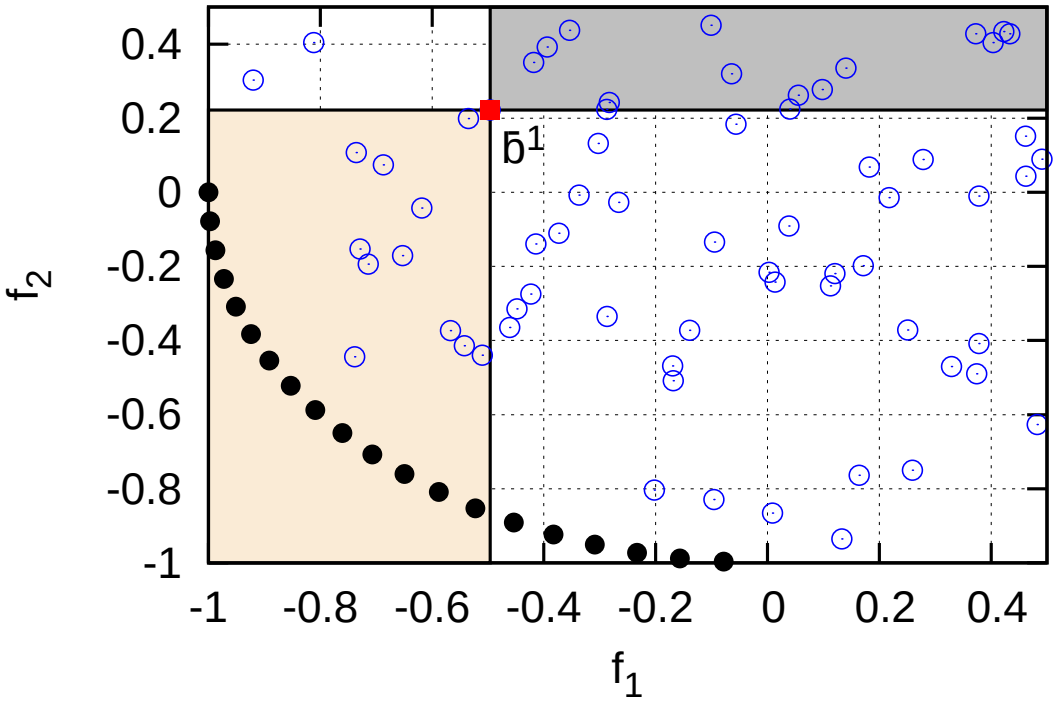


Figure 2.7: SPEA dominance applied to a population. Grey area: members dominated by $\vec{b}^1$. Yellow area: members dominating $\vec{b}^1$.

R_i and its density metric D_i,

$$\phi_i = R_i + D_i \tag{2.3.3.5}$$

NSGA2: NSGA2 [25] is an improvement to the NSGA technique [186]. It has improved the high computational complexity of non-dominated sorting and the need for specifying the sharing parameter of the initial algorithm. Moreover, the elitism operator is taken into account. NSGA2 relies upon the sorting of individuals according to Pareto dominance and density criteria. The following steps are applied:

Step 1: (Front Ranking) All individuals are classified in fronts with decreasing Pareto dominance.

Step a: (Initialization) Set front counter $i = 0$ and the individual set to $S = P_{\mu}^{g} \cup P_{\lambda}^{g} \cup P_{e}^{g-1}$.

Step b: (Find non-dominated members) Search set S for non-dominated solutions which are, then, copied to $\mathcal{F}_i$.

Step c: (Update) Remove the individuals of $\mathcal{F}_i$ from S and increase the counter $i=i+1$. If S is non-empty, repeat steps **b-c**, otherwise continue.

Step 2: (Density) For the individuals of each front $\mathcal{F}_i$, the density metric d_j is calculated. d_j is equal to the average side-length of the cuboid defined by the individual's neighbors within $\mathcal{F}_i$.

Step 3: (Scalar Value) The scalar value ϕ of each individual is based on its density metric and the front $\mathcal{F}_i$ it belongs to. ϕ is calculated according to the following steps:

Step a: (Initialize) $i = 0$ and $\phi_b = 1.0$.

Step b: (Compute ϕ) ϕ_j is computed as

$$\phi_j = \phi_b(1 + \frac{d_{max} - d_j}{d_{max}})$$

where d_{max} is the maximum d_k among the $\mathcal{F}_i$ members.

Step c: (Update) Update $\phi_b = 1.1max(\phi_j)$ and increase counter $i=i+1$. Repeat steps **b-c** for all fronts $\mathcal{F}_i$.

2.3.4 Constrained Optimization

The EASY platform is also capable of solving constrained optimization problems using penalization of the objective function values. Two thresholds, "nominal" and "relaxed", are defined by the user, separately for each constraint. The "nominal" threshold indicates the limit of each constraint function, herein, set to zero 0, without loss in generality. If the constraint of an individual is greater than zero, then a penalty term, exponentially proportional to the constraint violation, is added to its utility function. If the violation of a constraint exceeds the "relaxed" threshold ($d_k^* > 0$), a death penalty is applied (theoretically, $\phi=\infty$). If the individual's violation of constraints is less than the "relaxed" threshold, its scalar value is penalized as in [98]:

$$\phi(\vec{b}) = \phi(\vec{c}) + \prod_{k=1}^{M_c} \begin{cases} exp(a_k \frac{c_k(\vec{b})}{d_k^*}), & \text{if } d_k^* > c_k(\vec{b}) > 0 \\ 1, & \text{otherwise} \end{cases} \quad (2.3.4.1)$$

where the coefficient a_k is a user-defined parameter which defines the intensity of the penalty term.

2.3.5 Metamodel-Assisted Evolutionary Algorithms (MAEAs)

Engineering optimization problems frequently employ computationally expensive PSMs for evaluating individuals. In the problems this PhD thesis is dealing with, the PSM is an expensive CFD software which numerically solves the flow equations in either 2D or 3D domains. This, along with the increased number of calls to the PSM usually needed by EAs, results to a prohibitively large computational cost per optimization. Many methods capable of decreasing the number of evaluations needed by EAs have been devised during the last decades. The most popular among them is the use of surrogate models ("metamodels") [48, 99, 32, 17], which replicate the PSM with practically negligible computational cost. As mentioned before, this gives rise to the MAEA variant of EA which is a viable alternative for solving large-scale industrial optimization problems, in affordable wall clock time.

Artificial neural networks (ANNs), polynomial regression, Gaussian process and Kriging are only some of the metamodel types found in literature [65, 126]. Based on their training patterns, metamodels are categorized as global or local. Global metamodels are trained with individuals over the entire design space and are used to predict any given individual. On the other hand, local metamodels are optimized for a specific individual, since their training patterns are chosen in the close neighborhood of the individual to be predicted. Moreover, the metamodels are categorized as off-line or on-line [92, 7] based on whether they are trained before or during the evolution.

In the MAEA used in this PhD thesis and implemented in the EASY platform, on-line trained personalized metamodels undertake the pre-evaluation of the offspring in each generation. This Low-Cost Pre-Evaluation (LCPE) phase starts once a predefined minimum number (T^M) of evaluated individuals is archived in the DB. Up to that point, the standard EA is used and all offspring are evaluated on the PSM. Then, each individual is pre-evaluated by training a new metamodel on the closest individuals which have been evaluated on the PSM; these evaluations are often referred to as "exact evaluations" to contrast them w.r.t. evaluations on the metamodels. In MOO, different metamodels are trained to predict different objective functions and, then, the ϕ utility function is computed based on the predicted objective vector. Just a few ($\lambda_\varepsilon \ll \lambda$) most promising offspring (based on the predicted fitness value) undergo re-evaluation on the PSM (basically this determines the computational cost of each generation). Note that the elite set is populated exclusively with exactly evaluated individuals. The MAEA flowchart is shown in fig. 2.8. The basic types of metamodels used herein are described below:

Radial Basis Function (RBF) Networks

RBF networks [65] are ANNs with three neuron layers, an input, a hidden and an output layer as shown in fig. 2.9. The input layer corresponds to the design vector (input vector) with N nodes (input units). The hidden layer includes L nodes, as many as the RBF centers $\vec{c}^{\,l} \in \mathbb{R}^N$. The signal propagates forward from

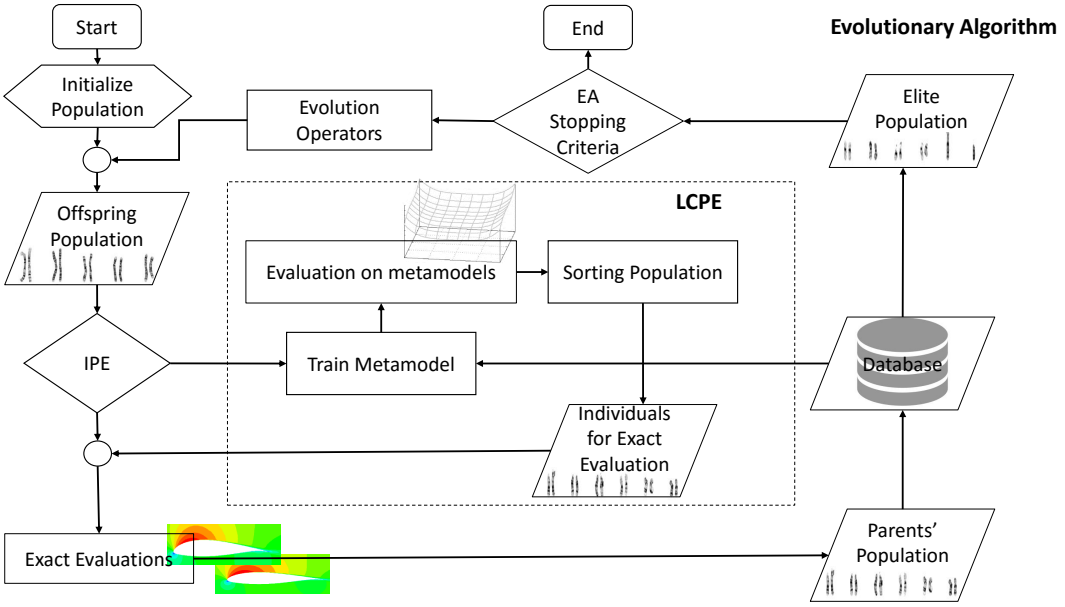


Figure 2.8: The MAEA flowchart using on-line locally trained metamodels as employed by the EASY software; developments made in this thesis (excluding application in chapter 7 dealing with Deep Neural Networks) are all based on this MAEA option.

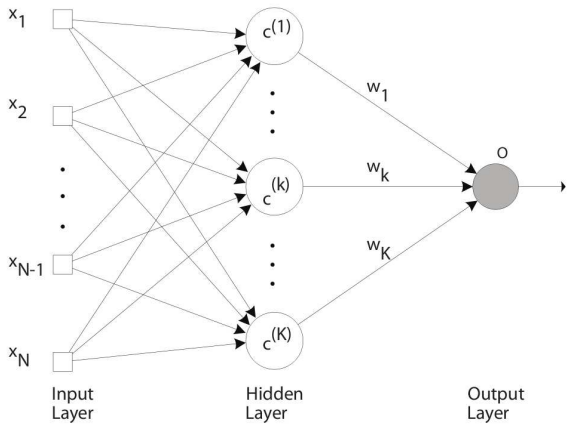


Figure 2.9: RBF Networks architecture.

the input to the hidden layer through a non-linear radial basis function which connects the input unit to each hidden node $G : \mathbb{R}^N \rightarrow \mathbb{R}$. In this thesis, the Gaussian activation function based on the distance between input $\vec{b}$ and center $\vec{c}^{\,l}$ given by

$$G(\vec{b}, r) = exp(\frac{\|\vec{b} - \vec{c}^{\,l}\|_2}{r^2}) \qquad (2.3.5.1)$$

where r is a user-defined radius (see below), is used. Then, the signal from the

hidden layer is propagated to the output layer, which has only one node/response (in our case, one objective function is to be predicted). Note that, in MOO, M_o metamodels are trained and used to predict the M_o objective functions for each individual. The response is expressed as the sum of the weighted output signals from the hidden layer, where weights w_l are computed during the training. Note that training uses a set of exactly evaluated patterns.

The RBF centers coincide with the training patterns ($\vec{c}^l = \vec{b}^l, l \in [1, T]$), if the hidden nodes are as many as the number of training patterns $T = L$. In this case, a $T \times T$ symmetric linear system of equations must be solved and the training patterns are exactly interpolated. Generally, the network can use a smaller number of hidden nodes than the training patterns ($L < T$) [134, 170]. In this case, the selection of the RBF centers is an important and complicated procedure, which the prediction ability of the network depends on. The selection scheme for the RBF centers used by EASY, which have been proposed in [92, 44, 41], is based on Self-Organized Maps (SOMs), fig. 2.10. This scheme consists of an unsupervised and a supervised learning process. At the beginning (unsupervised learning), the SOMs classify the training patterns into L clusters through standard processes: competition, cooperation and adaptation. Each cluster is represented by a single RBF center $\vec{c}^l$. The corresponding radius r needed in eq. 2.3.5.1 is computed by a heuristic method [92, 44, 95] based on the distances between the selected centers. In the second phase (supervised learning), the weights w_l are computed by minimizing the approximation error between the training patterns and the corresponding predictions provided by the network.

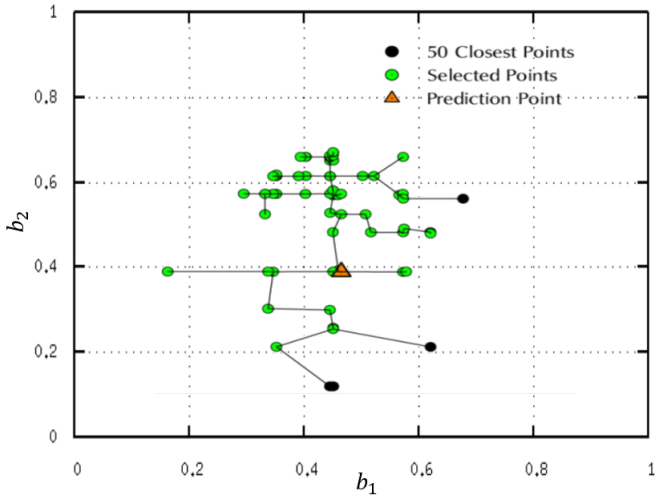


Figure 2.10: Selection of training patterns for the RBF metamodel using SOMs from Karakasis thesis [91].

Further improvement of the RBF network prediction ability can be achieved by using Importance Factors (IFs) proposed in [53]. The importance of each design

variable is quantified and exploited by including N additional coefficients (I_n, $n \in [1, N]$). High I_n values indicate high importance of the n^{th} design variable for the response (objective function) in the vicinity of the design vector into consideration. Each time an improved solution is found by the EA, the N partial derivatives $\partial\vec{f}/\partial b_n$ are computed using the closed-form expressions for the response. The importance factors (I_n) are then computed as

$$I_n = \frac{\partial\vec{f}(\vec{b})/\partial b_n}{\sum_{i=1}^{N}\partial\vec{f}(\vec{b})/\partial b_i} \tag{2.3.5.2}$$

Based on these derivatives, a weighted norm is introduced and used instead of the standard one and eq. 2.3.5.1 is re-written as

$$G(\vec{b}, r) = exp(\frac{\sqrt{\sum_{n=0}^{N} I_n(b_n - c_n^l)^2}}{r^2}) \tag{2.3.5.3}$$

Polynomial Regression Models

Polynomial regression [126] approximates the objective function (response) using polynomial functions. The polynomial coefficients, which are computed through the training phase, determine the prediction ability of the metamodel. During the training phase, a Least Squares (LSQ) method is used to best fit the polynomial function to the given training patterns. The polynomial regression model is mainly used to smooth noise found in the training patterns.

The response using a polynomial regression model is given as

$$o(\vec{b}) = a + \sum_{i=1}^{N}\sum_{j=1}^{P_i} c_{ij} b_i^j \tag{2.3.5.4}$$

where P_i is the maximum power in which the i^{th} design variable is raised to and a, c_{ij} are the unknown coefficients computed during training. Interactions among the design variables [21], expressed as $\prod_{i=0}^{N} b_i^{I_i}$ with I_i being the power which the i^{th} design variable will be raised to for the particular term, can be used to enrich eq. 2.3.5.4 . These introduce terms capable of expressing correlation among the design variables. Note that parameters P_i and the interactions are user-defined and require some knowledge of the problem (objective functions) in hand. Different polynomial setup results in different metamodel's prediction ability. The most commonly used polynomial functions are the first-order ($P_i = 1$ without interactions) and the second-order ($P_i = 2$ without interactions) function yielding satisfactory results in many problems [102].

The polynomial regression model relies on the Least Squares (LSQ) method

for computing the a, c_{ij} coefficients. Given a training set consisting of T individuals with their design vector and the corresponding response/objective function ($\vec{b}^1, \ldots, \vec{b}^T \rightarrow f_1, \ldots, f_T$), the LSQ method tries to minimize the error (ϵ) between the responses of the training patterns and the metamodel's predictions. The following analysis is based on the first-order polynomial function for simplicity reasons, given as

$$o(\vec{b}) = a + \sum_{i=1}^{N} c_i b_i \tag{2.3.5.5}$$

Based on this function, the estimated error for each pattern is expressed as

$$\epsilon_t = f_t - o(\vec{b}^t) = f_t - a - \sum_{i=1}^{N} c_i b_i^t \tag{2.3.5.6}$$

where $t \in [1, T]$ and b_i^t is the i^{th} design variable of the t^{th} training pattern. The total error is defined by

$$E_{tot} = \sum_{t=1}^{T} \epsilon^2 = \sum_{t=1}^{T} (f_t - a - \sum_{i=1}^{N} c_i b_i^t)^2 \tag{2.3.5.7}$$

and should be minimized. This means that its derivatives w.r.t the a, c_{ij} coefficients should be set to zero,

$$\frac{\partial E_{tot}}{\partial c_i} = -2 \sum_{t=1}^{T} (f_t - a - \sum_{i=1}^{N} c_i b_i^t) b_i^t = 0 \tag{2.3.5.8}$$

or

$$\frac{\partial E_{tot}}{\partial a} = -2 \sum_{t=1}^{T} (f_t - a - \sum_{i=1}^{N} c_i b_i^t) = 0 \tag{2.3.5.9}$$

Let us assume that $\mathbf{X}$ is the $T \times N$ matrix including all the design vectors and $\vec{f}$ is the vector with the T responses of the training patterns. To this end, $\vec{c}$ is a vector including all a, c_{ij} coefficients and the corresponding error is the vector $\vec{\epsilon}$. Eq. 2.3.5.7 is re-written in a compact matrix form

$$E_{tot} = \sum_{t=1}^{T} \epsilon_t^2 = (\vec{f} - \mathbf{X}\vec{c})'(\vec{f} - \mathbf{X}\vec{c}) = \vec{f}'\vec{f} - 2\vec{c}'\mathbf{X}'\vec{f} + \vec{c}'\mathbf{X}'\mathbf{X}\vec{c} \tag{2.3.5.10}$$

Then, the equations to be satisfied are

$$\frac{\partial E_{tot}}{\partial c_i} = -2\mathbf{X}'\vec{f} + 2\mathbf{X}'\mathbf{X}\vec{c} = 0$$
$$\mathbf{X}'\mathbf{X}\vec{c} = \mathbf{X}'\vec{f} \tag{2.3.5.11}$$

Eqs. 2.3.5.11 form a system of N equations ($\mathbf{X}'\mathbf{X} \in \mathbb{R}^{N\times N}$ and $\mathbf{X}'\vec{f} \in \mathbb{R}^N$), solved using the Cholesky decomposition (since the $\mathbf{X}'\mathbf{X}$ matrix is symmetric). Its solution computes the optimal coefficients ($\vec{c}^*$, * denotes the optimal vector) for the given training pattern and polynomial function. After the training phase, the objective functions value of any new individual is predicted as $o(\vec{b}) = \vec{c}^*\vec{b}$. The polynomial regression model, in its general form, has been introduced into the EASY platform, during this PhD thesis.

Multilayer Perceptrons (MLP)

The multilayer perceptron (MLP) performs a differential nonlinear mapping between the input and output space. A MLP with L layers, [66], is formed by the input layer with N sensory units, the output layer with M_o computational units and a user-defined number ($L-2$) of intermediate layers, each of which may have any number K_l of hidden neurons. Each hidden layer unit is fully connected on all units on the previous and next layers. All connections (synapses) between units are associated with synaptic weights $w_{[j,l+1],[i,l]}$, where the subscript denotes the two edge nodes along with their level. For instance $w_{[j,l+1],[i,l]}$ represents the weight of the synapsis between node i on the l-th layer (departure node) and node j on the $(l+1)$-th layer (arrival node).

The MLP training consists of the computation of synaptic weights to ensure the "best" mapping between given samples ($b^t, t \in [1,T]$) and their known responses ($\vec{\zeta}^t$). Each training pattern b^t is presented to the sensory units of the input layer ($l = 1$) and creates a signal propagating towards the network output through the intermediate layers and, finally, reaching the last (L) layer node where the network response o^t emerges. During signal propagation, the input signal $h^t_{[q,l]}$ ($[q,l]$ denotes the q-th neuron on layer l) is processed by an activation function yielding the neuron output. A widely-used activation function is the logistic one, $\mathbb{G}(h) = (1+e^{-Bh})^{-1}$ with $B > 1$. At any computational node, the output signal $v^t_{[q,l]}$ is given by

$$v^{(t)}_{[q,l]} = \mathbb{G}\left[\sum_{s=1}^{K_{l-1}} w_{[q,l],[s,l-1]} v^t_{[s,l-1]}\right] \tag{2.3.5.12}$$

where the activation function acts on the weighted sum of all signals appearing at the outputs of the K_{l-1} previous layer neurons.

The MLP networks are trained on T paired samples $(b^t, \zeta^{(t)})$ through the error back-propagation algorithm. This is an iterative algorithm which consists of suc-

cessive forward (function) and backward (error) signal passing through the MLP layers. In the forward pass, the input vector b^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 ζ^t and the prediction error is computed. In the backward pass, the synaptic weights are corrected to minimize the prediction error. This algorithm is repeated until convergence.

Kriging

Kriging, [110], belongs to the class of Gaussian random field metamodels. Kriging can predict not only the response of an individual but, also, a field of confidence (Mean Squared Error, MSE) pertinent to this prediction, [179]. For $\vec{b} \in \mathbb{R}^N$, the Kriging response $o(\vec{b})$ is expressed as

$$o(\vec{b}) = \hat{\mu} + z(\vec{b}) \tag{2.3.5.13}$$

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

$$\mathbb{C}\left(\vec{b^\kappa}, \vec{b^\lambda}, \vec{\theta}\right) = \exp\left(-\sum_{n=1}^{N} \theta_n \left(\vec{b}_n^\kappa - \vec{b}_n^\lambda\right)^2\right) \tag{2.3.5.14}$$

where the correlation parameters $\vec{\theta} = (\theta_1, \theta_2, ..., \theta_N) \in \mathbb{R}^N$ must be computed in order to find the best fit to the available data. Usually, an isotropic correlation kernel is assumed and eq. 2.3.5.14 is, then, rewritten with $\theta_n \equiv \theta$, so that a single θ value is to be computed. Training the kriging model consists in computing θ's, $\hat{\mu}$ and σ^2, which are all assumed to be invariant with respect to $\mathbb{F}$.

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

$$\max\left[\frac{1}{(2\pi)^{N/2}(\sigma^2)^{N/2}\sqrt{\det(\mathbf{C})}} \exp\left[\frac{(\boldsymbol{Z} - \mathbf{1}\hat{\mu})^T \mathbf{C}^{-1} (\boldsymbol{Z} - \mathbf{1}\hat{\mu})}{2\sigma^2}\right]\right] \tag{2.3.5.15}$$

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

$$\mathbf{C} = \begin{bmatrix} \mathcal{C}\left(\vec{b}^1, \vec{b}^1, \vec{\theta}\right) & \cdots & \mathcal{C}\left(\vec{b}^1, \vec{b}^T, \vec{\theta}\right) \\ \vdots & \ddots & \vdots \\ \mathcal{C}\left(\vec{b}^T, \vec{b}^1, \vec{\theta}\right) & \cdots & \mathcal{C}\left(\vec{b}^T, \vec{b}^T, \vec{\theta}\right) \end{bmatrix}, \; 51\boldsymbol{Z} = \begin{bmatrix} \zeta^{(1)} \\ \vdots \\ \zeta^{(T)} \end{bmatrix} \tag{2.3.5.16}$$

Eq. 2.3.5.15 can be solved by adopting generalized least-squares estimates for $\hat{\mu}$ and σ^2 , [104], as follows

$$\hat{\mu} = \frac{\mathbf{1}^T\mathbf{C}^{-1}\boldsymbol{Z}}{\mathbf{1}^T\mathbf{C}^{-1}\mathbf{1}} \; , \; \sigma^2 = \frac{1}{N}(\boldsymbol{Z} - \mathbf{1}\hat{\mu})^T\mathbf{C}^{-1}(\boldsymbol{Z} - \mathbf{1}\hat{\mu}) \tag{2.3.5.17}$$

Substituting eqs. 2.3.5.17 into eq. 2.3.5.15 , leads to the solution of an equivalent minimization problem,

$$\min\left[N\ln\left(\sigma^2(\vec{\theta})\right) + \ln\left(\det\mathbf{C}(\vec{\theta})\right)\right] \tag{2.3.5.18}$$

After computing the θ value by solving the nonlinear problem of eq. 2.3.5.18 , the corresponding response and MSE predictions for any new $\vec{b}$ in the design space are, [32],

$$o(\vec{b}) = \hat{\mu} + (\boldsymbol{Z} - \mathbf{1}\hat{\mu})^T\mathbf{C}^{-1}\mathbf{c}(\vec{b}) \tag{2.3.5.19}$$

$$MSE(\vec{b}) = \sigma^2 \cdot (1 - \mathbf{c}(\vec{b})^T\mathbf{C}^{-1}\mathbf{c}(\vec{b})) \tag{2.3.5.20}$$

where

$$\mathbf{c} = \left[\mathcal{C}\left(\vec{b}, \vec{b}^{(1)}, \vec{\theta}\right), \ldots, \mathcal{C}\left(\vec{b}, \vec{b}^{(T)}, \vec{\theta}\right)\right]^T \tag{2.3.5.21}$$

During this thesis, the Kriging model is programmed and incorporated into the EASY platform as another type of metamodel to be used.

2.3.6 Distributed EAs

Another way to reduce the wall clock time of an optimization is by employing a distributed search, which gives rise to Distributed EAs (DEAs) and, combined with metamodels, Distributed MAEAs (DMAEAs). In DEAs, a number of sub-populations (demes/islands) are evolving concurrently in semi-isolation, fig. 2.11. The demes may follow different evolution policies (crossover scheme, mutation probabilities, elitism operators etc.). The use of different or differently tuned evolution operators (for instance, different mutation or crossover probabilities) per deme is, generally, very helpful. It allows some demes (those with high mutation

probability, for instance) to explore so-far unexplored areas of the design space and the rest to undertake the exploitation of the already accumulated information by further improving current best solutions.

Demes communicate regularly by exchanging a few individuals (usually the most promising ones) during the migration cycles. The inter-deme migration operator exchanges the best performing (and/or some other randomly selected) individuals. Migration topology and/or frequency and the selection-replacement policies are user-defined parameters leading to different convergence speed.

Using a DEA, rather than a single-population EA, is more efficient in terms of CPU cost. By additionally using metamodels within each deme (DMAEA, with a single DB of already evaluated individuals shared by all its demes), the gain in the optimization turnaround time is even higher. In addition, distributed schemes are, by definition, amenable to parallelization.

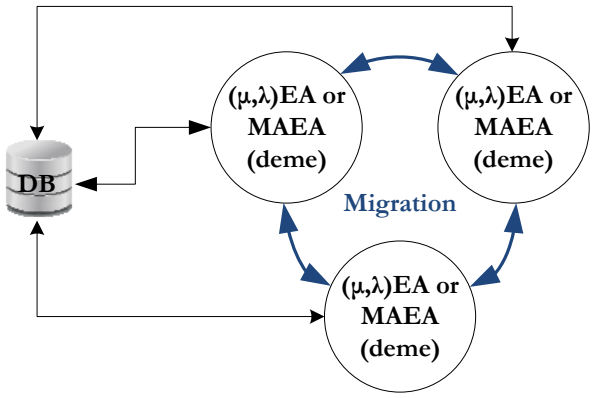


Figure 2.11: Schematic representation of a DEA or DMAEA equipped with a ring migration topology.

2.3.7 Hierarchical EAs

The hierarchical EA (HEA) consists of a number of levels, each of which can be associated with a different search technique, evaluation tool and/or parameterization/number of design variables, [159, 30, 78, 79, 80, 54]. Adjacent levels may exchange their best (and/or some other solutions apart from the current best, depending on the scheme) individuals using one- or two-way inter-level communications. In the majority of cases, HEAs are restricted to two levels only.

The incorporation of metamodels into hierarchical schemes is straightforward, by simply using MAEAs instead of EAs. This gives rise to the so-called hierarchical MAEAs (HMAEAs). Note that, on each level, different DBs of previously evaluated solutions are kept for the purpose of training the corresponding metamodels.

2.4 Benchmark Cases

All the aforementioned EA variants are demonstrated in the same five benchmark cases. In the following chapters, where new EA or MAEA etc. variants are proposed, their performance is assessed on the same cases. Without loss in generality, real encoding of the design variables is used in all cases. Each optimization, for each variant of EAs, has been performed three times with different RNG seeds, so as to reduce any randomness caused by the selected RNG seed. The presented results are the average of the three optimization runs performed in each case. The hypervolume indicator I_H [27] is used to measure the quality of the computed fronts of non-dominated solutions, fig. 2.12. Given a set A of objective vectors, the indicator is formulated via the function $I_H(A) = \int_{nadir}^{zenith} a_A(z)dz$ where $a_A(z)=1$ if A dominates z and $a_A(z)=0$ otherwise. This quantifies the percentage of the area dominated by the front of non-dominated individuals within a "box" defined by user-defined nadir and zenith points; higher values of the hypervolume indicator denote better front quality. The nadir point should be selected by con-

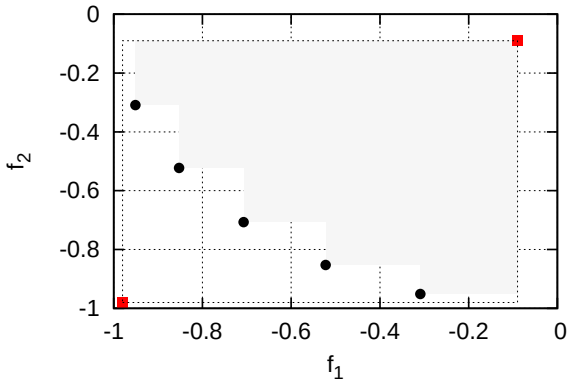


Figure 2.12: Definition of the Hypervolume Indicator in a two-objective minimization problem. Grey area dominated by the front.

sidering the worst individual, whereas the zenith point corresponds to the optimal solution. Practically, the nadir point is set to a solution which is worse than the worst attainable solution and the zenith one is set to a optimal solution which is not attainable. In this way, the front will always be contained by these points. The hypervolume indicator varies from 0 to 1, with 0 being the worst and 1 the best possible value.

A GPU-enabled Reynolds-Averaged Navier-Stokes equations' solver developed by PCOpt/NTUA, called PUMA, [174, 171], acts as the PSM in all the cases excluding one (section 6) that is solved using the in-house CFD code based on the cut-cell method. The first solver may handle unstructured grids, on which the discretization of the governing equations is carried out using the finite volume technique

with vertex-centered storage of the flow variables. It solves compressible and incompressible flows (both variants are used in this thesis). For the discretization of the convection fluxes, the Roe's flux [140] difference splitting scheme is used. For second order spatial accuracy, appropriate limiting functions are used. The software is programmed by fully exploiting the NVIDIA's CUDA programming environment and can run on a cluster of GPUs. In order to minimize the overall communication overhead, GPUs belonging to the same computational node exchange data using the on-node shared CPU memory, while inter-node communication protocols, such as the MPI, are used for passing data among GPUs on different nodes. In addition, elimination of synchronization barriers and maximization of memory bandwidth are achieved by programming techniques for discretizing and solving the PDEs which are specifically programmed for GPUs. All these features make the GPU variant of the flow solver (running on a single NVIDIA K20 GPU) about 40 times faster than its CPU variant (running on a single Intel Xeon E5-2620 CPU core). The second solver is based on the cut-cell variant of the general class of Immersed Boundary Methods and the finite volume approach. It is parallelize with the MPI protocol to run on a cluster of CPUs and shares the same numerical features with PUMA.

2.4.1 Benchmark Case 1: Shape Optimization of an Isolated Airfoil for max. Lift Coefficient

The first case is concerned with the design of an isolated airfoil for maximum lift coefficient (C_L); this is a SOO problem, constrained only by the lower and upper bounds of the control point coordinates. The 2D flow is inviscid with $M_\infty = 0.4$ and $a_\infty = 5^o$. As a starting point, an existing airfoil geometry was parameterized using Bézier polynomial curves with 8 control points per airfoil side, fig. 2.13. Using the starting/reference airfoil, the design variables and their bounds were defined; practically, all control points (apart from the first and last ones on each side, i.e. those corresponding to the airfoil leading and trailing edge) are allowed to vary along the y-axis by $\pm 20\%$ of the reference position, summing up to $N = 12$ design variables in total. An unstructured grid with about 20K nodes is generated around each airfoil; a single run of the PUMA code, including grid generation, takes ~ 10 secs on an NVIDIA K20 GPU.

The optimization is carried out using a $(20, 40)$EAand MAEA, a $(10, 20)$DEA and DMAEA. In both DEA and DMAEA, the accompanying (μ, λ) values refer to the populations of each deme. In this case, there are just two demes, which results to a total of 20 parents and 40 offspring per generation. This allows a "fair" comparison with the single-population EA and MAEA. Demes communicate every second generation by exchanging their 3 current best individuals. In the MAEA and DMAEA, on-line trained RBF networks are used as metamodels. The LCPE

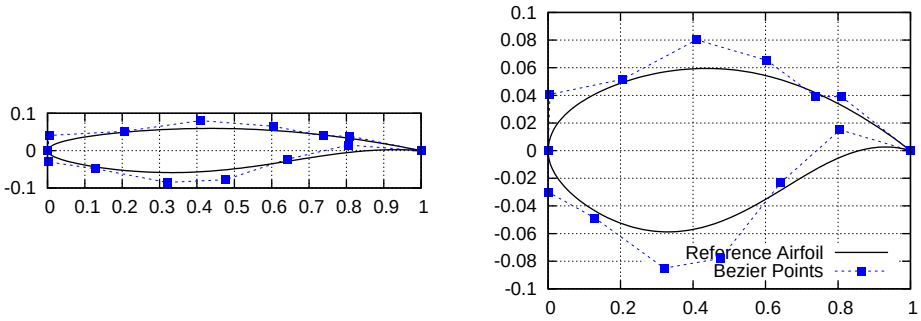


Figure 2.13: Benchmark Case 1: Parameterization/reproduction of the reference airfoil using two Bézier curves. Left: Axes in scale. Right: Axes not in scale.

phase is activated after the first $T^{MM} = 40$ evaluations on the PSM which are, of course, stored in the DB. After that, only the top $\lambda_e = 3$ individuals within any generation, based on the metamodels' predictions, are re-evaluated on the PSM. The same stopping criterion of 1000 evaluations on the CFD solver is imposed; this also allows a fair comparison to be made, since all of them have practically the same computational cost.

Comparison of the convergence histories of the aforementioned variants is presented in fig. 2.14. One may notice that a solution characterized by the same objective function value as the optimal solution of the EA, computed with the allowed computational budget, can be found by the MAEA with half of this budget. Moreover, the use of a distributed search can further improve the optimization with the DMAEA variant outperforming any other variant. This case shows, that, should the user of the optimization software be satisfied with the optimal solution reached by the EA after 1000 PSM calls, then the DMAEA could provide a solution of same quality with about $1/6$ of the overall budget. Fig. 2.17 shows the evolution of the best solution performance within each deme; it can be seen that both demes contribute to the faster convergence of the overall search method. Different initialization of the population does not affect greatly the drawn conclusions, as it can be seen in fig. 2.16. Even though, the MAEA presented in fig. 2.14 uses RBF metamodels, for comparison reasons, the same MAEA is also performed with the RSM and Kriging local metamodels. Their convergence histories are presented in fig. 2.15, which shows that the RSM and Kriging provide more or less the same results (depending on the setup and the problem in hand). Fig. 2.18 illustrates the Mach number field around the reference and the optimal airfoil and clearly shows why lift is higher in the optimized geometry: a much greater percentage of the suction side of the optimal airfoil is covered by supersonic flow, contributing to higher lift.

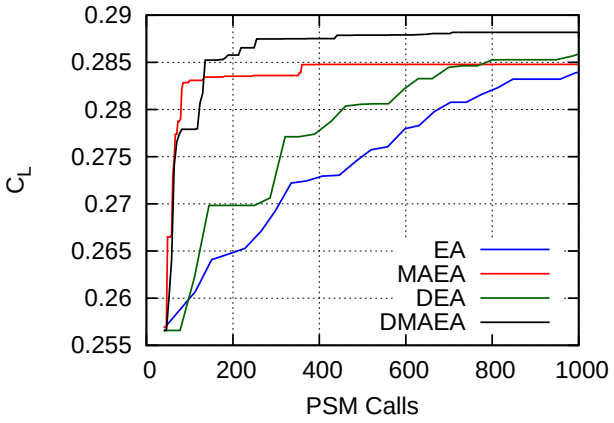


Figure 2.14: Benchmark Case 1: Comparison of the averaged convergence histories of EA, MAEA, DEA and DMAEA (average of three runs per method, with different RNG seeds) in terms of the number of CFD evaluations (or equivalently, PSM calls), being proportional to the CPU cost.

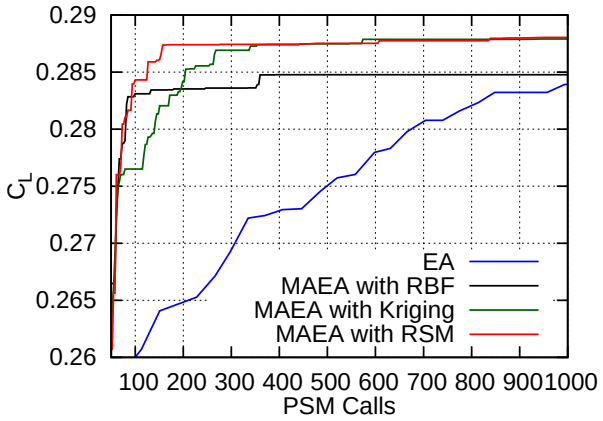


Figure 2.15: Benchmark Case 1: Comparison of the averaged convergence histories of EA and MAEA with RBF, RSM or Kriging metamodels in terms of the number of CFD evaluations.

2.4.2 Benchmark Case 2: Shape Optimization of a Transonic Wing for max. Lift and min. Drag Coefficient

The second benchmark problem is dealing with the MOO (two-objective) shape optimization of an isolated wing, [154], for max. lift coefficient (C_L) and min. drag coefficient (C_D). The flow is inviscid with $M_\infty = 0.8395$, $a_{\infty,pitch} = 3.06^o$ and $a_{\infty,yaw} = 0^o$.

The wing shape is parameterized using a $6 \times 3 \times 3$ volumetric NURBS control grid, fig. 2.19. 12, out of the 54 total control points, are allowed to move in the

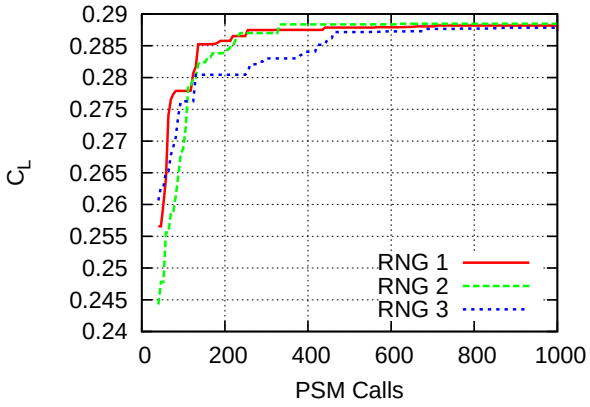


Figure 2.16: Benchmark Case 1: Convergence histories of the DMAEA with different RNG seeds.

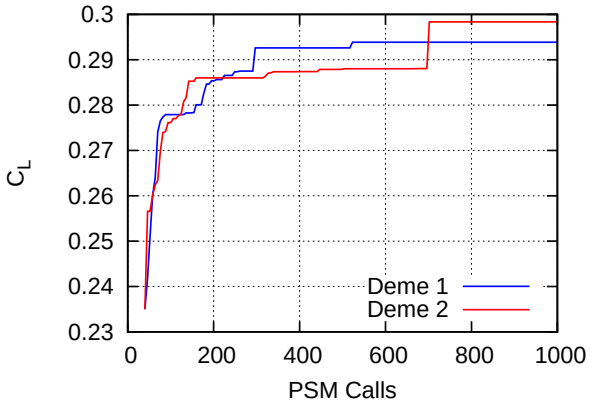


Figure 2.17: Benchmark Case 1: Comparison of the convergence histories of the two demes of the DMAEA run in terms of the number of CFD evaluations (PSM calls). The plotted curves correspond to a single run with the same RNG seed.

chordwise and the normal-to-the-planform direction, resulting to $N = 24$ design variables in total. In this case, this control grid is responsible only for morphing the wing shape (and the corresponding surface grid) and not the 3D CFD grid. The spring analogy technique is applied to deform the 3D CFD grid, according to the wing's surface grid deformation. For the CFD runs, the initial unstructured grid generated around the reference geometry is comprised by tetrahedra, prisms, pyramids and hexahedra, with ~ 1.33 million nodes in total. A single run of the PUMA software (by solving Euler equations) requires ~ 2 minutes on an NVIDIA K20 GPU.

Similarly to the previous case, a $(10, 20)$EA and MAEA, a $(5, 10)$DEA and DMAEA are used. In the distributed variants, the two demes exchange their 3 best individuals every second generation. In the MAEA and DMAEA, $T^{MM} = 30$ and $\lambda_e = 4$.

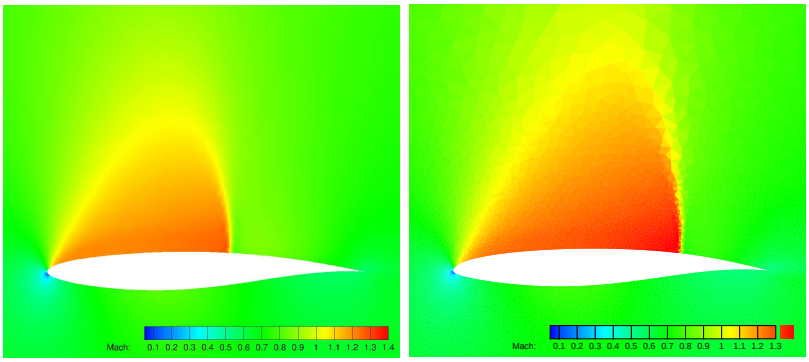


Figure 2.18: Benchmark Case 1: Mach number fields around the reference (left, $C_L = 0.222$) and the "optimal" (right, $C_L = 0.288$) airfoils. The latter is the outcome of the DMAEA run using one RNG seed as in fig. 2.16.

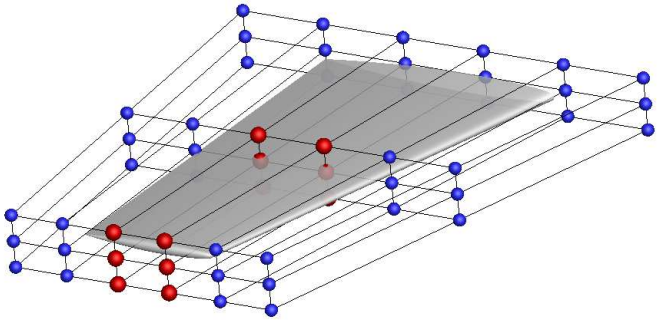


Figure 2.19: Benchmark Case 2: Control points of the volumetric NURBS, used to parameterize the wing. Blue points are kept fixed, whereas red ones are allowed to vary in the chordwise and normal-to-the-planform directions, during the optimization.

A termination criterion of 500 evaluations on the CFD solver is imposed. As mentioned before, the hypervolume indicator quantifies the quality of the fronts and the convergence of the optimization procedure. The averaged evolution histories for three different RNG seeds of the hypervolume indicator are compared in fig. 2.21. Moreover, the computed fronts of non-dominated solutions, for all the runs with different RNG seeds are presented in fig. 2.20 (left). In fig. 2.20 (right), the contribution of each front (different variant of EA) to the front of the overall non-dominated solutions is depicted with the DMAEA having most of the individuals. It is obvious that the use of on-line trained metamodels improves the convergence speed of the EA. Fig. 2.22 shows that the MAEAs using the RBF, Kriging or RSM metamodels have similar convergence speeds. Over and above, the introduction of a distributed search scheme further boosts search, reaching improved fronts of non-dominated solutions computed with the same budget with the EAs.

The Mach number fields on the wing surface for the reference, the max. lift and

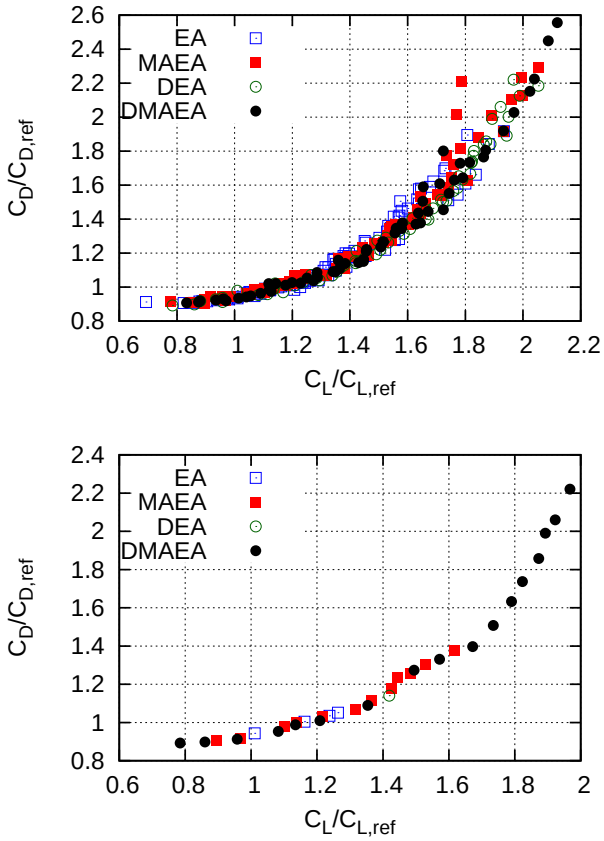


Figure 2.20: Benchmark Case 2: Comparison of the fronts of non-dominated solutions resulted from EA, MAEA, DEA and DMAEA, with the same computational budget. Both objective functions have been normalized using the corresponding values of the reference wing geometry. Especially, in the part of the front with the highest lift values, the DMAEA clearly outperforms any other method. Top: Fronts from three runs per optimization method with different RNG seeds. Bottom: Individuals from each front outperforming all the others.

the min. drag coefficient wings are shown in fig. 2.23. The optimized geometries result in different shock positions compared to the reference one. On the geometry with the min. C_D, the shock strength is reduced, the pressure difference between the pressure and suction sides is decreased and so does the drag. Regarding the other extreme point on the front of non-dominated solutions, the higher pressure difference on the max. C_L geometry is produced by the more cambered geometry. This difference increases the shock strength and, simultaneously, the lift of the wing.

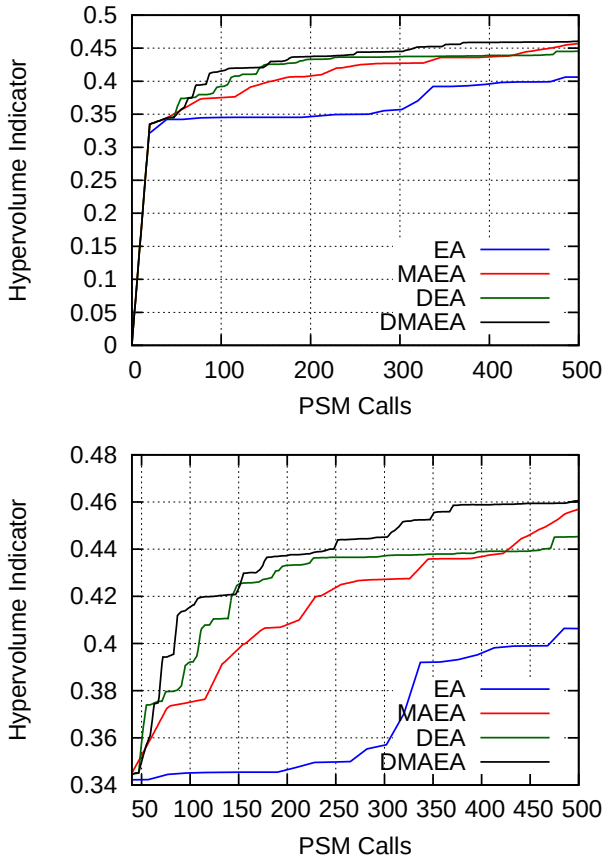


Figure 2.21: Benchmark Case 2: Top: Comparison of the averaged convergence histories of EA, MAEA, DEA and DMAEA in terms of the number of CFD evaluations. Bottom: Close-up view after the first $T^{MM} = 30$ evaluations on the PSM (CFD tool), during which phase metamodels are in use.

2.4.3 Benchmark Case 3: Optimization of a Three-Element Airfoil for max. Lift Coefficient and min. Moment Coefficient

The third benchmark case is dealing with the two-objective optimization of a three-element airfoil for max. lift coefficient(C_L) and min. pitching moment coefficient(C_M). This case was firstly presented in [58]. Herein, it is revisited with two objectives instead of one. The airfoil consists of a main body, a slotted flap and a leading edge slat. The flow conditions are: freestream Mach number $M_\infty = 0.12$ and flow angle $a_\infty = 17.18^o$. The flow is inviscid. This problem, although 2D, is representative of the design of high-lift devices and is studied at a high angle of attack to mimic take-off conditions. During the optimization, only the positioning of the flap and

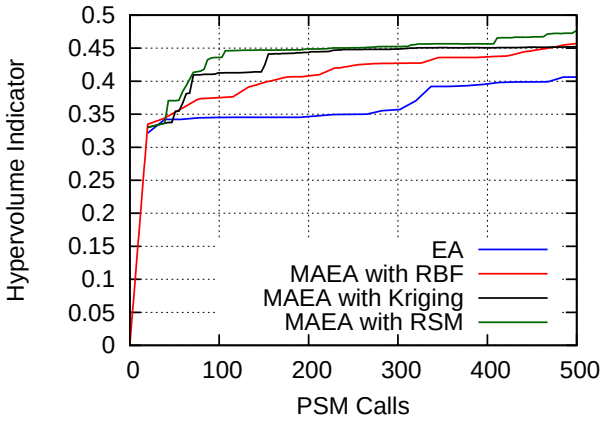


Figure 2.22: Benchmark Case 2: Comparison of the averaged convergence histories of the EA and MAEA with RBF, Kriging and RSM, in terms of the number of CFD evaluations.

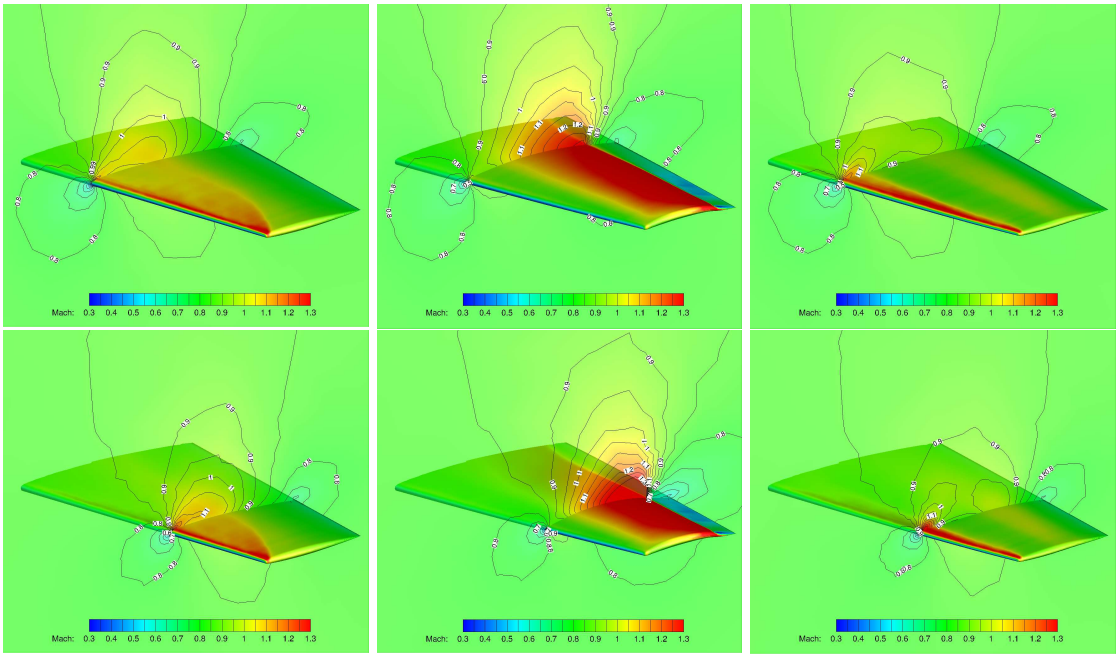


Figure 2.23: Benchmark Case 2: Mach number fields on the surface of the reference (left), the max. C_L (center) and min. C_D (right) wings. Top: Spanwise cuts at 30% for the wing root. Bottom: Spanwise cuts at 70% for the wing root. The presented optimized geometries resulted from the one of the DMAEA runs.

slat w.r.t. the main body is allowed to vary, while their shapes remain intact. This leads to 3 design variables (position vector and rotation angle in the 2D space) per element or 6 design variables in total. The CFD grid comprises triangles and quadrangles with ~ 20K nodes in total. Each individual to be evaluated on the

PSM needs to be re-meshed but this is done at almost negligible cost by means of a fast in-house grid generator. The overall cost per evaluation is $\sim 2min$ on one NVIDIA K20 GPU.

A $(10, 20)$EA and MAEA are used to optimize the case, with $\lambda_e = 1$ and $\varepsilon = 10$. The computational budget is restricted to 200 PSM calls. The LCPE phase is activated after $T^{MM} = 20$ evaluated individuals have been recorded in the DB. Comparison of the convergence histories of the optimization runs based on the hypervolume indicator is presented in fig. 2.24. All three MAEAs assisted by the RBF, Kriging and RSM metamodels perform better than the EA. Fig. 2.25 presents the front of non-dominated solutions resulted using the different variants (for the same RNG seed), both EA and MAEA have concluded to similar fronts but the MAEA with a much faster convergence.

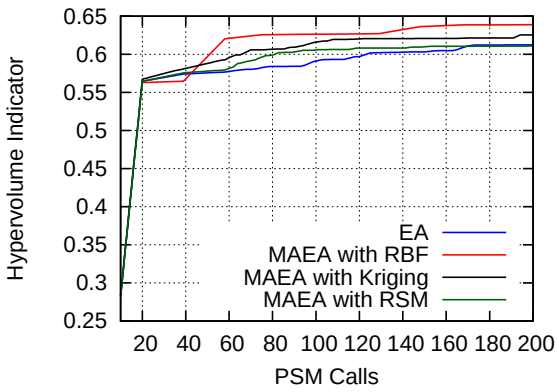


Figure 2.24: Benchmark Case 3: Comparison of the averaged convergence histories of the EA and MAEA with RBF, Kriging or RSM metamodels in terms of the number of CFD evaluations.

The different positioning of the flap and slat for the max. lift, min. moment and the reference configurations can be seen in fig. 2.26. A comparison of the Mach number field around these configurations is shown in fig. 2.27. For the max. lift configuration, both the flap and slat are deployed as much as possible in order to accelerate the flow along the main body suction side, thus generating more lift. For the computed min. moment configuration, the flap deployment is smaller so as to decrease the lift of the aft section as well as the moment.

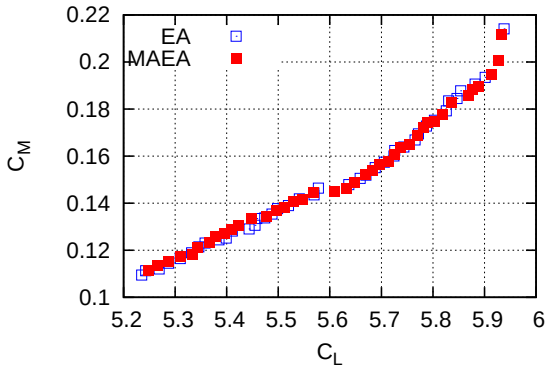


Figure 2.25: Benchmark Case 3: Fronts of non-dominated solutions computed by the EA and MAEA with RBF (for the same RNG seed), all of them with the same computational cost.

2.4.4 Benchmark Case 4: Optimization of a 2D compressor for max. Flow Turning and min. Losses

The fourth benchmark case is dealing with the shape optimization of a 2D section of the TurboLab compressor stator. The TurboLab compressor stator model is a stator blade row tested in the measurement rig of the Chair for Aero Engines at TU Berlin (TUB), [176]. The stator is an aerodynamic model representative of modern jet engine compressor bladings. A two-objective optimization is carried out aiming at maximum difference of the exit flow from the axial direction (max. $\Delta\alpha$) and minimum total pressure losses (min. Δp_t). The flow is incompressible with inlet velocity $48.13m/s$, flow inlet angle $a_1 = -42^o$ and stagger angle $a_{st} = 12^o$. The Reynolds number, based on the chord, is equal to $Re = 3.58 \times 10^6$ and the Spalart-Allmaras turbulence model [164] is used. The shape is parameterized using two Bezier curves, with 8 control points each, see fig. 2.28. All 16 control points but the four at the edge of the stator are allowed to move along the y axis, giving rise to 12 design variables in total. The CFD grid is unstructured with ~ 30K nodes. Each CFD evaluation on the PUMA software takes about $\sim 30s$ on an NVIDIA K20 GPU, including re-meshing.

A $(15, 30)$EA and MAEA, with 600 PSM calls as the allowed computational budget, $\lambda_e = 1$ and $\varepsilon = 20$, are used. The on-line trained metamodels are used after the first generation of the MAEA, i.e. by setting $T^{MM} = 30$. Fig. 2.29 shows the convergence of the EA and MAEA with all different metamodels based on the hypervolume indicator. Fig. 2.30 presents the front of non-dominated solutions (for the same RNG seed). It can be seen that the MAEA outperforms the EA, irrespective of the type of the metamodel that is used.

The different shapes for max. $\Delta\alpha$ and min. Δp_t compared with the reference

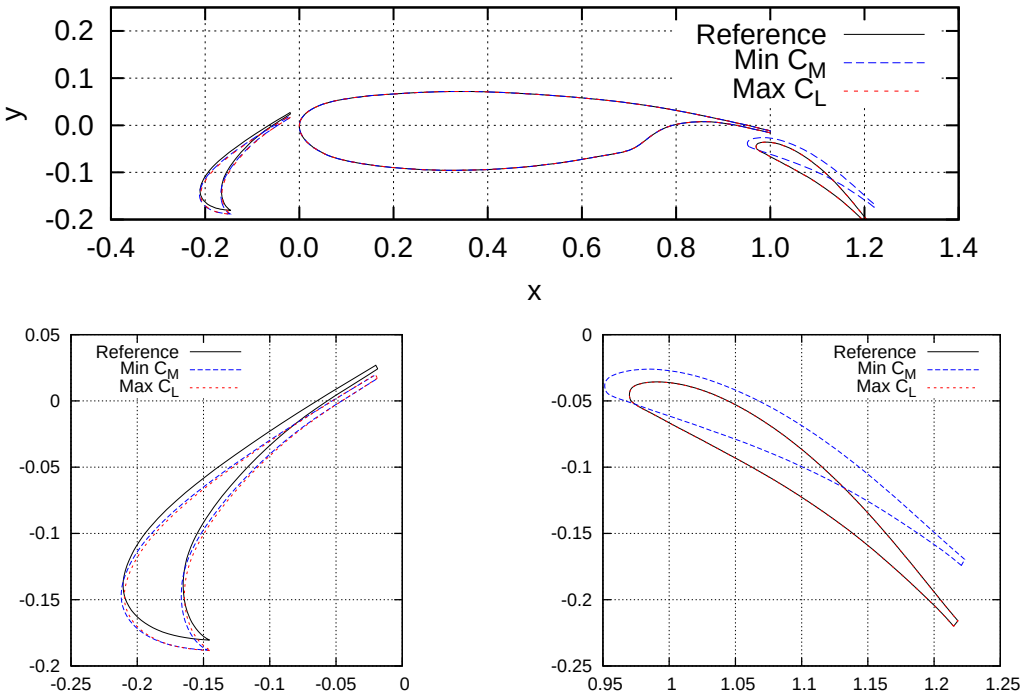


Figure 2.26: Benchmark Case 3: Comparison of the reference airfoil (black, solid), the one with max. C_L (red, dotted) and the one with min. C_M (blue, dashed) on the MAEA's front of non-dominated solutions. Top: Main body, flap and slat. Bottom: Details of the slat (left) and flap (right).

one are presented in fig. 2.31. Note that, in the stator blade airfoil corresponding to max. $\Delta\alpha$, the curvature of the pressure side and the thickness of the stator blade airfoil are increased, causing more intensive flow separation. In the stator blade airfoil achieving min. Δp_t, small changes occur compared to the reference one. Both flow separation and flow turning are reduced.

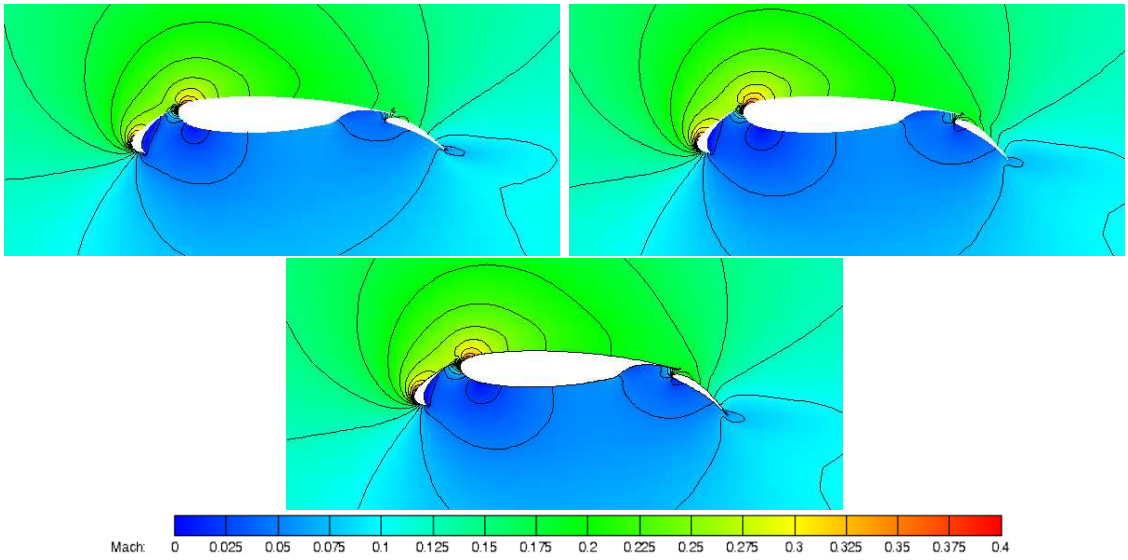


Figure 2.27: Benchmark Case 3: Mach number fields around the min. C_M (top, left), max. C_L (top, right) and reference (bottom) airfoils.

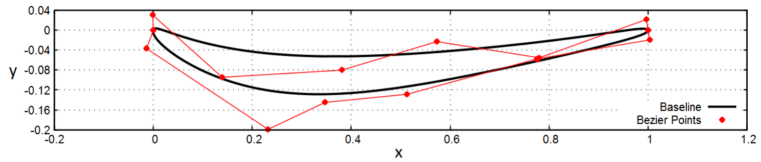


Figure 2.28: Benchmark Case 4: Stator's blade airfoil along with 2 Bezier curves used to parameterize its contour.

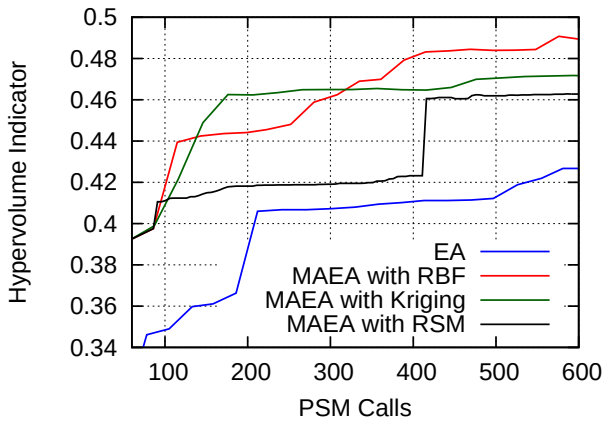


Figure 2.29: Benchmark Case 4: Comparison of the averaged convergence histories of EA and MAEA with RBF, Kriging and RSM metamodels in terms of the number of CFD evaluations.

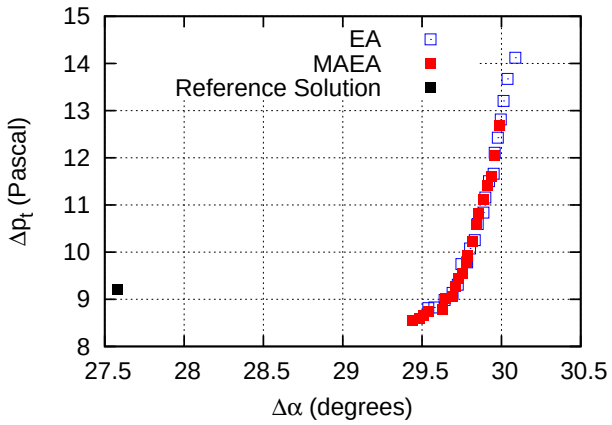


Figure 2.30: Benchmark Case 4: Fronts of non-dominated solutions computed by the EA and MAEA (for the same RNG seed), all of them with the same computational cost. The reference blade airfoil gives an exit flow angle equal to $a_2 = -14.3^o$.

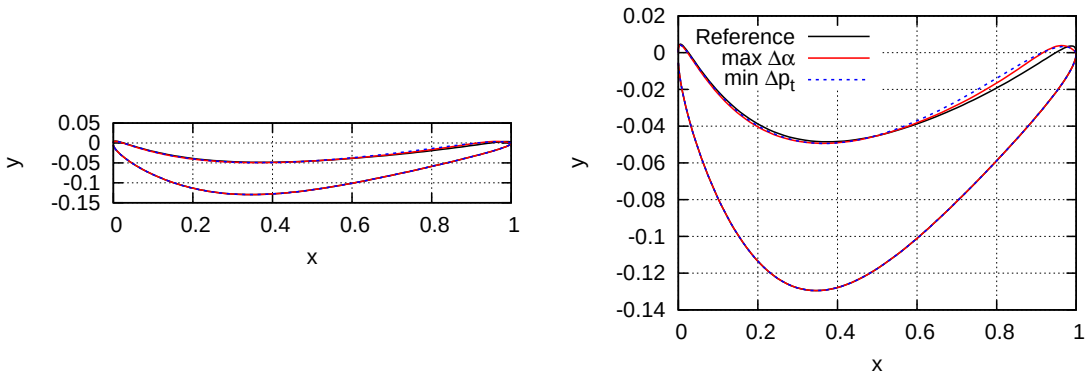
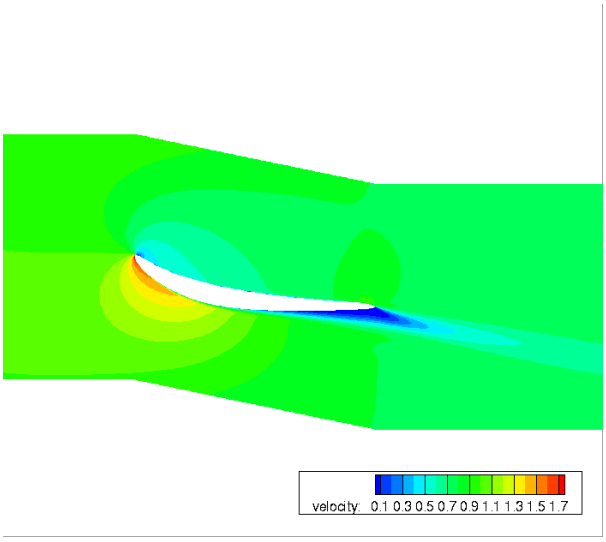
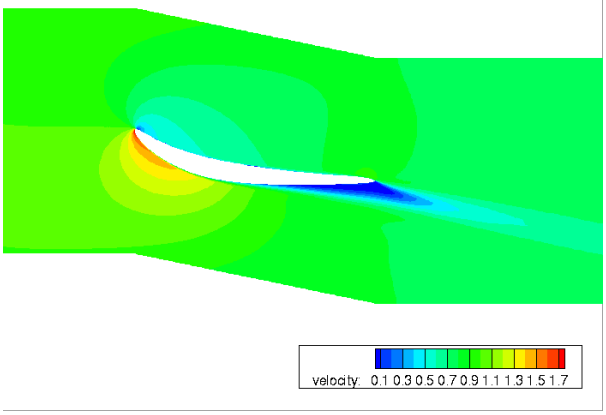


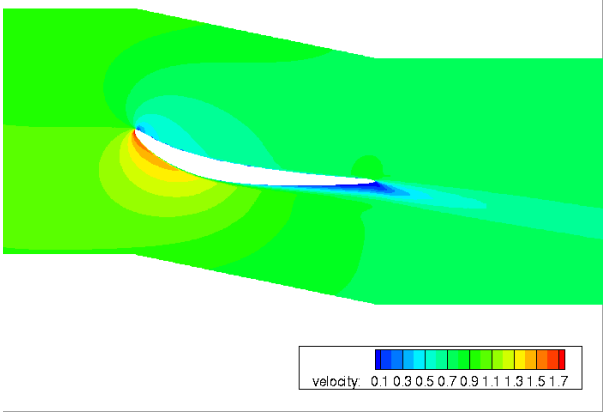
Figure 2.31: Benchmark Case 4: Comparison of the reference stator (black, solid), that with max. $\Delta\alpha$ (red, solid) and that with min. Δp_t (blue, dashed). These solutions are selected from the DMAEA's front of non-dominated solutions obtained with one of the RNG seeds. Left: Axes in scale. Right: Axes not in scale.



(a) Max. $\Delta\alpha$ ($\Delta p_t = 13.8Pa$ & $\Delta\alpha = 29.9^o$)



(b) Min. Δp_t ($\Delta p_t = 8.6Pa$ & $\Delta\alpha = 29.4^o$)



(c) Reference ($\Delta p_t = 9.1Pa$ & $\Delta\alpha = 27.6^o$)

Figure 2.32: Benchmark Case 4: Velocity magnitude fields for the min. Δp_t, the max. $\Delta\alpha$ and the reference stators.

Chapter 3

Principal Component Analysis

Industrial optimization problems deal with many objective functions introducing constraints along with complex parameterizations and a great number of design variables. Working with an EA that does not perform wrll reflects upon the total number of evaluations on the PSM needed to find the global optimum or optima, leading to an increased optimization turnaround time. The cost of the industrial use of EAs can thus become prohibitive and this hinders their extensive use in industrial optimization workflows. A possible way to alleviate this problem is by developing methods that may handle cases with a great number of design variables, as efficiently as possible. This is where this chapter focuses on. The methods proposed in this section can also be used with not so many design variables in which cases the gain is also non-negligible.

In general, any (direct) change in the parameterization and the so-defined design variables might have negative effects on the outcome of the optimization or could even be impossible. During the establishment of an optimization problem, the experienced designer sets up the design variables based on existing knowledge on this or similar problems and the goal(s) to be achieved. The designer may not be able to modify the parameterization so as to achieve the given goal. Even if the above were possible, there is no guarantee that the new parameterization results in a well-posed problem with uncorrelated design variables. Note that correlated variables are variables related with each other. Finally, very often, the most appropriate parameterization cannot be 'a priori' known, since the optimal solution(s) might be needed to extract the required information about this.

In a survey on GAs presented in [149], it was clearly explained that the efficiency of GAs is reduced when the objective functions are not separable. Though, [149] was based on GAs, the conclusions drawn there can be extended to EAs as well. In [143, 144], it was stated that, in most industrial optimization problems, the design variables are possibly correlated ("variable dependency") and the objectives are non-separable ("inseparable function interaction"), introducing several issues during optimization. As stated in [144], multi-modality, deception and dis-

continuity may also occur based on the variables correlation, causing additional loss in the optimization algorithm's efficiency. In optimization, multi-modality is the existence of many optimal (local optimal, at least) solutions of the problem in hand, as opposed to a single best solution. Deceptive optimization problems exhibit one or more deceitful optima located far away from the global optimum and the attraction of the local optima are much bigger than the one of the optimal solutions misleading the optimization algorithms. The methods proposed in this chapter aim at overcoming some of the aforementioned problems.

3.1 Curses of Engineering Optimization Problems

Engineering optimization problems usually suffer from three different, but closely connected, problems. They are often ill-posed and/or non-separable and they may also deal with a high number of design variables ($N \gg$). These factors degrade the performance of EAs and/or MAEAs and that is why they are often referred to as "curses". An "ill-posed" problem [75, 63] is one which does not meet the three Hadamard criteria for being well-posed. These criteria are: (a) having a solution, (b) having a unique solution or (c) having a solution that depends continuously on the parameters or input data. Another criterion is that the solution procedure is unstable, i.e. arbitrarily small changes in the design variable values may lead to large differences in the objective function. Usually, the aerodynamic optimization problems lack in the latter criterion. Most difficulties in solving ill-posed problems are caused by the objective(s) instability, meaning the change of the objective(s) values with a slight change in the design variables values. Moreover, "ill-posed" problems use to have anisotropic objective function(s). An objective function $f(\vec{b})$ with $\vec{b} \in \mathbb{R}^N$ is anisotropic if it is differently affected by the same changes in different design variables. This means that the same change in value for two different design variables may cause extremely different changes in the objective function value ($\frac{\partial f}{\partial b_i} \gg \frac{\partial f}{\partial b_j}$). Anisotropy leads to more or less important directions in the design space. These negatively affect the EA convergence, which perform optimally in "well-posed" problems with more or less the same importance in each design variable. The metamodel prediction ability also suffers due to the high complexity of the objective function.

The high number of design variables ($N \gg$), which many engineering optimization problems are dealing with, gives rise to the so called "curse of dimensionality". When dimensionality (N) increases, the volume of the design space increases. The EAs require a great number of evaluations to search this space which is drastically increased with N. Furthermore, individuals forming the population appear to be dissimilar in many ways, which prevents their proper organization into populations with common characteristics. Regarding the metamodels used in MAEAs, they are negatively affected by the increasing value of N. More training patterns and computational cost are required, while the predictions may not be accurate

enough. The higher prediction error decreases the efficiency of the pre-evaluation phase of MAEAs and reduces further the overall performance of the optimization.

Lastly, the engineering optimization problems usually deal with non-separable objective function(s). An objective function $f(\vec{b})$ is separable if it can be minimized or maximized separately with respect to each design variable $b_i,\ i=[1,N]$. If the optimal value of b_i is independent of the values other design variables take on at the optimal solution, then the objective is separable with respect to b_i. Mathematically expressed, a function f is separable if there exists functions $f_1, f_2, \ldots, f_N$ such that: $f(b_1, b_2, \ldots, b_N) = f_1(b_1)f_2(b_2)\ldots f_N(b_N)$ (see fig. 3.1). In optimization problems with separable objective function(s), the maximum optimization time for finding the global optimum/optima is increased with N, because, in the worst case scenario, the optimization solves N distinct optimization problems, one for each design variable. In other words, it suffices to minimize $f_i(b_i)$ in terms of $b_i,\ \forall\ i \leqslant N$. However, things are not as simple in non-separable optimization problems and, as one might easily guess, this is practically never the case in aerodynamic optimization. Since complexity of search is increased exponentially with the number of design variables, the overall cost is also greatly increased [129].

In conclusion, the EAs or MAEAs do not perform well in non-separable, "ill-posed" problems dealing with large N values, which is usually the case in engineering. All these "curses" are connected with each other leading to problems which suffer from all of them simultaneously. It would be ideal if there was a method capable of transforming the initial problem into a new one which is separable, "well-posed" and artificially deals with less design variables. So, the goal is to find an appropriate transformation mapping a problem that is difficult to be solved onto another that can more easily be solved (by EAs and MAEAs). Only the design variables the EA or MAEA sees are allowed to be modified, because, in general, the PSM computing the objective function(s) is a black-box tool. This transformation should "change" the initial design variables so as for the objective function to be expressed as separable as possible with respect to these new variables. Moreover, the significance of each new variable should be computed, so as to emphasize the most important ones during the EAs evolution, which assist at solving the "ill-posed" problem. Metamodels can also be trained only with the new significant variables, so as to alleviate the "curse of dimensionality" which they suffer from; by doing so, their prediction ability is expected to improve by reducing their training cost.

In a previous PhD thesis [112] carried out also in the PCOpt/NTUA Unit, the Principal Component Analysis (PCA) had efficiently been introduced into EAs. The PCA transforms the initial design space into a "more separable" one (referred to as the "feature space"). In the feature space, the problem is "well-posed" and as separable as possible, with new optimization variables the significance of which becomes known from the PCA. Thus, the evolution operators are applied in the

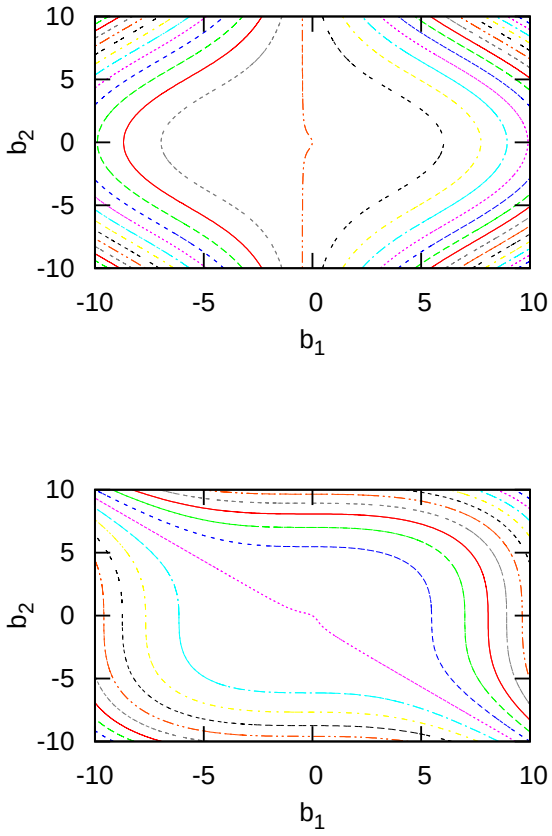


Figure 3.1: Iso-lines of a separable (top) and a non-separable (bottom) function with two design variables. The separable one is $f = b_1^3 + b_1^2 + b_1 + b_2^3 + b_2^2 + b_2$ and the non-separable one is the same but with design variables $b_1^* = b_1 sin(\pi/4) + b_2 cos(\pi/4)$ and $b_2^* = b_1 cos(\pi/4) - b_2 sin(\pi/4)$. Both have minimum F for $b_1, b_2 = 0, 0$.

new feature space in which they perform optimally. Moreover, during the LPCE phase of the MAEAs, the PCA in its capacity as a dimensionality reduction tool reduces the number of design variables the metamodels are trained with. Both usages of PCA reduce the overall optimization cost of EAs, as convincingly shown in [112]. In the latter, the Linear PCA was used. In the present thesis, the Kernel variant of PCA is introduced [85], which seems to further improve the convergence speed of EAs, as it is demonstrated in the cases examined in this chapter. One should keep in mind that the coupling of EAs and PCA is different compared to [112], as described below in detail.

3.2 Basics of the Principal Component Analysis

The PCA [65, 74] method is an unsupervised learning technique capable of converting a data-set ($\mathbf{B}$) of observations $\vec{b} \in \mathbb{R}^N$ of possibly correlated variables into a set of uncorrelated ones called principal components (PCs). These define a new design space, to be referred to as the feature space, in which the transformed design variables become as separable as possible. By doing so, non-separable "ill-posed" problems are transformed into separable "well-posed" ones, thus speeding-up the EA-based search. Ideally, the PCA should produce a transformation that makes the objective function(s) as separable as possible based on a data-set representative of the problem in hand. Unfortunately, this is not possible prior to the optimization, because such a data-set has not been formed or evaluated yet. Thus, one should use the PCA in each generation anew, updating and introducing newly collected information about the problem. Thus, the method is gradually improved as the optimization loop evolves, consequently producing better transformations.

The selection of the data-set ($\mathbf{B}$), which the PCA is trained on in each generation, greatly affects the resulting feature space. The data-set should contain all the important information about the problem in hand, the progress of the optimization as well as the fittest individuals, so as to further promote them and guide the search in the important directions of the design space. Each one of the population (elite, offspring and parent) used by the EA contains different information about the optimization procedure. The elite set, containing the most fit individuals, after some generations can be populated by individuals which are "extremely" close to each other (based on their Euclidean distance on the design space), especially in SOO. This will eventually reduce diversity, which is an important feature of PCs. Moreover, the elite set does not contain vital information about the current generation of the optimization procedure, since it might have not been changed or is stagnated during the last generations. The offspring or parent population are more diverse, while containing important fit individuals. Note that some of the best elites are inherited from generation to generation by means of the elitism operator. The offspring clearly depict the evolution state in each generation, containing some unique characteristics. Hereafter, the PCA is trained on the offspring population, due to its diversity and the unique information it contains.

Linear PCA The PCA can be either Linear (LPCA) or Kernel (KPCA). EAs and MAEAs assisted by LPCA were firstly presented in [113] and [114]. The covariance matrix ($\mathbf{P}$) for the LPCA is computed as $\mathbf{P}_{N\times N} = \frac{1}{M}\mathbf{B}\mathbf{B}^T$, where M is the number of observations (herein, the number of offspring, $M = \lambda$) and N the number of design variables. This matrix represents correlations among observations. Then, the spectral decomposition [34] of the matrix is written as $\mathbf{P}_{N\times N} = \mathbf{V}\mathbf{\Lambda}\mathbf{V}^T$ where $\mathbf{\Lambda}$ is a diagonal matrix with the eigenvalues of $\mathbf{P}$ and $\mathbf{V}$ a $N\times N$ matrix formed by the eigenvectors of $\mathbf{P}$ as rows. The Singular Value Decomposition method solves this

eigenproblem. The produced eigenvectors are the PCs, which construct the feature space, and the corresponding eigenvalues are the variances connected to the PCs. An observation $\vec{b}$ (or, practically, any possible new design vector/individual) can be transformed into the corresponding vector $\vec{c}$ in the feature space, meaning its projection to this space, through the equation

$$\vec{c} = \mathbf{V}(\vec{b} - \vec{\mu}_b) \tag{3.2.1}$$

where $\vec{\mu}_b$ is the vector formed by the mean values of the data-set's design variables. The inverse transformation (from an observation into the feature space to the design space) is given by the equation

$$\vec{b} = \mathbf{V}^{-1}\vec{c} + \vec{\mu}_b \tag{3.2.2}$$

It is important to note that the CPU cost to compute $\mathbf{V}^{-1}$ is negligible since $\mathbf{V}$ is an orthogonal matrix and, thus, $\mathbf{V}^{-1} = \mathbf{V}^T$.

Kernel PCA On the other hand, the KPCA transforms the design space into the feature one through a mapping function, $\phi : \mathbb{R}^N \to \mathbb{R}^L$, where L can be arbitrarily large. For the same data-set $\mathbf{B}$ used for the LPCA, the new covariance matrix $\mathbf{P}_{L\times L}$ (or just $\mathbf{P}$, hereafter) is expressed as

$$\mathbf{P}_{\tau\sigma} = \frac{1}{M}\sum_{i=1}^{M} \phi_\tau(\vec{b}^i)\phi_\sigma(\vec{b}^i),\ \tau, \sigma = 1, \ldots, L \tag{3.2.3}$$

or, in matrix form as, $\mathbf{P} = \frac{1}{M}\mathbf{\Phi}\mathbf{\Phi}^\mathbf{T}$ where $\mathbf{\Phi} \in \mathbb{R}^{\mathbf{L}\times\mathbf{M}}$ is formed by the $\vec{\phi}(\vec{b}^i) \in \mathbb{R}^L$ vectors as columns. The corresponding eigenproblem is expressed by the following system of L equations

$$\mathbf{P}\vec{V}^r = \lambda^r \vec{V}^r,\ r = 1, \ldots, L \tag{3.2.4}$$

where $\vec{V}^r \in \mathbb{R}^L$ is the r^{th} eigenvector of $\mathbf{P}$, λ^r is the r^{th} eigenvalue of the $L \times L$ diagonal matrix $\mathbf{\Lambda}$. Depending on the value of L, the above eigenproblem can become large and, thus, expensive to solve. The so-called kernel trick helps alleviating this problem by avoiding explicitly computing the function ϕ. The kernel matrix $\mathbf{K} \in \mathbb{R}^{M\times M}$ is introduced expressed as

$$\mathbf{K}_{ij} = k(\vec{b}^i, \vec{b}^j) = \vec{\phi}(\vec{b}^i)\vec{\phi}^T(\vec{b}^j) = \sum_{p=1}^{L} \phi_p(\vec{b}^i)\phi_p(\vec{b}^j) \tag{3.2.5}$$

A widely used kernel [155, 177] is the polynomial function $k(\vec{b}^i, \vec{b}^j) = (a(\vec{b}^i)^T\vec{b}^j +$

$c)^d$, where adjustable parameters are the slope a, the constant term c and the polynomial degree d. Another one is the sigmoid function $k(\vec{b}^i, \vec{b}^j) = tanh(a(\vec{b}^i)^T\vec{b}^j + c)$, where the slope a and the intercept constant c are parameters. However, herein, without loss of generality, another function, the RBF kernel function [89]

$$k(\vec{b}^i, \vec{b}^j) = exp\left(- \frac{||\vec{b}^i - \vec{b}^j||_2^2}{2\sigma^2} \right) \tag{3.2.6}$$

is used, where σ is the width constant. This function is equivalent to the that used in RBF networks, see section 2.3.5, with different width constant σ. σ is automatically calculated as the variance of the data-set the PCA is trained on. Each eigenvector can alternatively be written as $\vec{V}^r = \sum_{m=1}^{M} a_{rm}\vec{\phi}(\vec{b}^m),\ r = 1, \ldots, L$ or in a matrix form $\mathbf{V} = \mathbf{\Phi A}$, where matrix $\mathbf{A} \in \mathbb{R}^{M \times L}$ includes the unknown a_{im} coefficients. By substituting the eigenvectors, eqs. 3.2.4 are re-written as

$$\sum_{j=1}^{L} \mathbf{P}_{ij} \sum_{\tau=1}^{M} a_\tau^r \phi_j(\vec{b}^\tau) = \lambda^r \sum_{\tau=1}^{M} a_\tau^r \phi_i(\vec{b}^\tau), \quad i = 1, \ldots, L \tag{3.2.7}$$

The inner products of eqs. 3.2.7 and $\vec{\phi}(\vec{b}^q),\ q = 1, \ldots, M$ yield a system of M equations,

$$\sum_{i=1}^{L}\sum_{j=1}^{L}\sum_{\tau=1}^{M} a_\tau^r \mathbf{P}_{ij} \phi_i(\vec{b}^q)\phi_j(\vec{b}^\tau) = \lambda^r \sum_{i=1}^{L}\sum_{\tau=1}^{M} a_\tau^r \phi_i(\vec{b}^\tau)\phi_i(\vec{b}^q),\ q = 1, \ldots, M \tag{3.2.8}$$

By means of eqs. 3.2.3 , eqs. 3.2.8 become

$$\begin{aligned} &\sum_{i=1}^{L}\sum_{j=1}^{L}\sum_{\tau=1}^{M}\sum_{z=1}^{M} a_\tau^r \phi_j(\vec{b}^z)\phi_i(\vec{b}^z)\phi_i(\vec{b}^q)\phi_j(\vec{b}^\tau) = \\ &M\lambda^r \sum_{i=1}^{L}\sum_{\tau=1}^{M} a_\tau^r \phi_i(\vec{b}^\tau)\phi_i(\vec{b}^q), \quad q = 1, \ldots, M \end{aligned} \tag{3.2.9}$$

Finally, the kernel matrix is introduced into eqs. 3.2.9 . This reduces the number of equations from L to M, which are solved with reasonable computational cost. The final system of equations to be solved is

$$\mathbf{K}\vec{a}^q = M\lambda^q\vec{a}^q, \quad q = 1, \ldots, M \quad or \quad \mathbf{KA} = M\mathbf{A\Lambda} \tag{3.2.10}$$

The solution of eqs. 3.2.10 can provide all the necessary information for the definition of the feature space, without solving eqs. 3.2.4 . The $\mathbf{A}, \mathbf{\Lambda}$ matrices have insignificant elements with low values after the M^{th} element and can be replaced by truncated matrices $\hat{\mathbf{A}}, \hat{\mathbf{\Lambda}} \in \mathbb{R}^{M\times M}$. The resulted equivalent eigenproblem of smaller size ($M \ll L$) can be solved, where the eigenvector matrix $\hat{\mathbf{V}} \in \mathbb{R}^{L\times M}$ is given by $\hat{\mathbf{V}} = \mathbf{\Phi}\hat{\mathbf{A}}$, similarly with the non-truncated matrices. These eigenvalues and eigenvectors produce the feature space in which the given data-set is expressed as uncorrelated as possible, based on the nonlinear kernel function used.

Each individual $\vec{b} \in \mathbb{R}^N$ can be projected onto the feature space to yield $\vec{c} = \vec{c}(\vec{b}) \in \mathbb{R}^M$ by applying the transformation

$$\vec{c}(\vec{b}) = \hat{\mathbf{V}}^T\vec{\phi}(\vec{b}) = \hat{\mathbf{A}}^T\mathbf{K}(\vec{b}) \Leftrightarrow c_r(\vec{b}) = \sum_{i=1}^{M} a_i^r k(\vec{b}^i, \vec{b}), \quad r = 1, \ldots, M \tag{3.2.11}$$

During each generation, parents are transformed into the feature space using eq. 3.2.11 , crossover and mutation operators are applied on them to produce offspring and, finally, the new offspring (expressed into the feature space) are transformed back to the design space. Thus, the inverse transformation $\vec{c}(\vec{b}) \in \mathbb{R}^M \rightarrow \vec{b} \in \mathbb{R}^N$ is absolutely necessary. This is done by mapping both $\vec{c}$ and $\vec{b}$ into the $\mathbb{R}^L$ space and minimizing the Euclidean distance between their projections. The projection of $\vec{c}$ is denoted by $\vec{p} \in \mathbb{R}^L$ and that of $\vec{b}$ by $\vec{\phi}(\vec{b}) \in \mathbb{R}^L$. The following problem practically expresses the minimization of their distances,

$$\vec{b} = argmin||\vec{p} - \vec{\phi}|| = argmin\{\vec{p}^T\vec{p} + \vec{\phi}^T\vec{\phi} - 2\vec{p}^T\vec{\phi}\} \tag{3.2.12}$$

Since $\vec{c} = \hat{\mathbf{V}}^T\vec{c}$ and $\hat{\mathbf{V}}^T\hat{\mathbf{V}} = \mathbf{I}$, $\vec{p}$ can be expressed as $\vec{p} = \hat{\mathbf{V}}\vec{c}$ leading to $\vec{p}^T\vec{p} = \vec{c}^T\hat{\mathbf{V}}^T\hat{\mathbf{V}}\vec{c} = \vec{c}^T\vec{c}$ which is a known constant. Since, also, $\vec{\phi}^T\vec{\phi} = 1$, the minimization problem of Eq. (3.2.12) leads to $\frac{\partial}{\partial\vec{b}}\Big(- 2\vec{p}^T\vec{\phi}\Big) = 0$. Replacing $\vec{p}$ and $\vec{\phi}$, yields

$$\frac{\partial}{\partial\vec{b}}\Big(- 2\vec{\phi}(\vec{z})\sum_{j=1}^{M} C_j(\vec{z})\vec{\hat{V}}^j\Big) = 0$$

or, equivalently,

$$\frac{\partial}{\partial\vec{b}}\Big(- 2\vec{\phi}(\vec{z})\sum_{j=1}^{M} C_i(\vec{z})\sum_{i=1}^{M} a_{ji}\vec{\phi}(\vec{b}^j)\Big) = 0$$

After some re-arrangements, the problem becomes

$$\frac{\partial}{\partial \vec{b}}\Big(-2\sum_{j=1}^{M}\sum_{i=1}^{M} C_i(\vec{z})a_{ji}\vec{\phi}(\vec{b}^j)\vec{\phi}(\vec{z})\Big) = 0 \Leftrightarrow$$
$$\frac{\partial}{\partial \vec{b}}\Big(-2\sum_{j=1}^{M}\sum_{i=1}^{M} C_i(\vec{z})a_{ji}k(\vec{b}^j,\vec{z})\Big) = 0 \tag{3.2.13}$$

If the RBF kernel, eq. 3.2.6 is used in eq. 3.2.13 , resulting in:

$$\frac{\partial}{\partial \vec{b}}\Big(-2\sum_{j=1}^{M}\sum_{i=1}^{M} C_i(\vec{z})a_{ji}exp\Big(-\frac{||\vec{b}^j-\vec{z}||_2^2}{2\sigma^2}\Big)\Big) = 0$$

or

$$-2\sum_{j=1}^{M}\sum_{i=1}^{M} C_i(\vec{z})a_{ji}\frac{\partial}{\partial \vec{b}}\Big(exp\Big(-\frac{||\vec{b}^j-\vec{z}||_2^2}{2\sigma^2}\Big)\Big) = 0$$

and, finally,

$$-2\sum_{j=1}^{M}\sum_{i=1}^{M} C_i(\vec{z})a_{ji}exp\Big(-\frac{||\vec{b}^j-\vec{z}||_2^2}{2\sigma^2}\Big)\frac{(\vec{z}-\vec{b}^j)}{\sigma^2} = 0 \tag{3.2.14}$$

Eq. 3.2.14 is solved with the fixed point iterative algorithm, [177]. Firstly, it is separated into two terms

$$\sum_{j=1}^{M}\sum_{i=1}^{M} C_i(\vec{z})a_{ji}exp\Big(-\frac{||\vec{b}^j-\vec{z}||_2^2}{2\sigma^2}\Big)\vec{z} =$$

or

$$\sum_{j=1}^{M}\sum_{i=1}^{M} C_i(\vec{z})a_{ji}exp\Big(-\frac{||\vec{b}^j-\vec{z}||_2^2}{2\sigma^2}\Big)\vec{b}^j$$

or

$$\vec{z} = \frac{\sum_{j=1}^{M}\sum_{i=1}^{M} C_i(\vec{z})a_{ji}exp\Big(-\frac{||\vec{b}^j-\vec{z}||_2^2}{2\sigma^2}\Big)\vec{b}^j}{\sum_{j=1}^{M}\sum_{i=1}^{M} C_i(\vec{z})a_{ji}exp\Big(-\frac{||\vec{b}^j-\vec{z}||_2^2}{2\sigma^2}\Big)} \tag{3.2.15}$$

Having divided the equation into two terms, the solution $\vec{z}$ can be found iteratively

according to the following scheme

$$\vec{z}^{new} = \frac{\sum_{i=1}^{M}\sum_{j=1}^{M} C_i(\vec{z}^{old})a_{ij}exp(-\frac{||\vec{b}^j-\vec{z}^{old}||_2^2}{2\sigma^2})\vec{b}^j}{\sum_{i=1}^{M}\sum_{j=1}^{M} C_i(\vec{z}^{old})a_{ij}exp(-\frac{||\vec{b}^j-\vec{z}^{old}||_2^2}{2\sigma^2})} \tag{3.2.16}$$

In this thesis, both variants of the PCA method are used in a similar manner, and the text below does not practically distinguishes between Linear and Kernel PCA. Small changes that might occur are clarified.

3.3 EA with PCA-driven Evolution Operators

The PCA is used to transform the design space into a new feature space, which is as separable as possible and where the evolution operators are expected to perform better. Initially, the EA evolves without PCA and, when a number of user-defined evaluations and/or generations are reached, the PCA takes over. At the beginning of each generation, the PCA, trained with the current offspring, constructs a feature space (in $\mathbb{R}^{M_o}$ for the LPCA and $\mathbb{R}^{\lambda}$ for the KPCA). The current parent population is transformed into this space using eq. 3.2.1 for the LPCA or eq. 3.2.11 for the KPCA. The evolution operators are applied there to produce the new offspring population. Then, the new offspring, just created into the feature space, are transformed back to the design space using eq. 3.2.2 for the LPCA and eq. 3.2.16 for the KPCA. This procedure is limited to the evolution operators only, thus, the other processes are not affected by the transformation. The EA or MAEA assisted by the PCA during the evolution is referred to as EA(L) or MAEA(L), if the LPCA is used, and EA(K) or MAEA(K), if the KPCA is used instead.

3.3.1 PCA-Driven Crossover

Before explaining the application of crossover in the LPCA or KPCA feature space, a simple mathematical optimization example with two optimization variables ($b_i \in [-32.768, 32.768], i = 1, 2$) is presented, in brief, for the purpose of demonstration. It suffices to work with a single-objective function, namely the Ackley function [3]. This is a non-convex function used as a performance test problem for optimization algorithms. This is defined as $f(\vec{b}) = -20exp[-0.2\sqrt{0.5(b_1^2+b_2^2)}] - exp[0.5(cos(2\pi b_1)+cos(2\pi b_2))] + e + 20$ for two optimization variables. It has lot of local optima as it can be seen in fig. 3.2, and a single global optimum located at $(0, 0)$, with $f(0, 0) = 0$. Starting with the KPCA, this creates a $\mathbb{R}^{\lambda}$ feature space, where the individuals are clustered according to their importance (based on their variance). The crossover operator, applied in the feature space, promotes the first few important components, by neglecting all the others. Thus, the most signif-

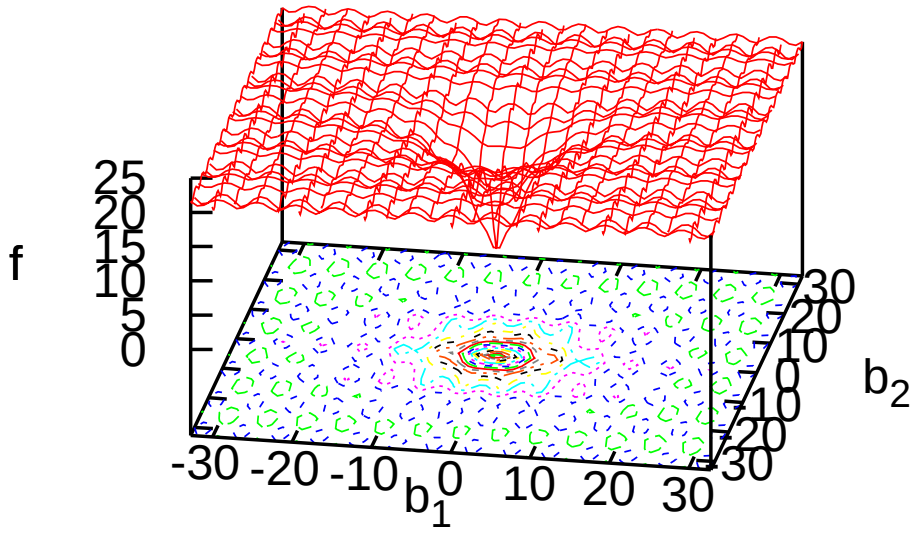
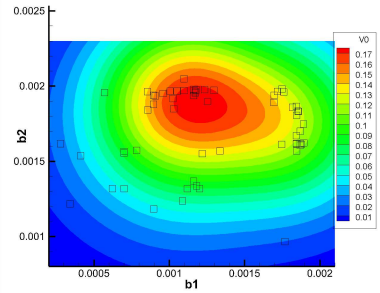
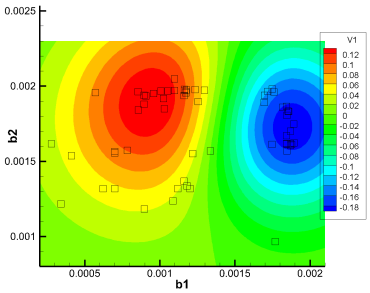


Figure 3.2: Ackley function for two optimization variables.

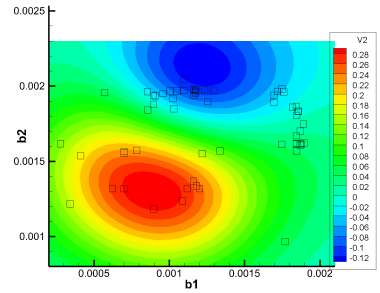
icant characteristics are evolved into the new offspring population and the next generation. A $(30, 60)$ EA with KPCA is used to optimize the Ackley case. During its evolution, the offspring population of the 40^{th} generation is extracted. Note that the population size is selected based on experience and tests and, in general, does not affect the conclusions drawn. Then, some individuals in the optimization space are generated at random in the vicinity of this population to be transformed by the PCA. Then, the KPCA is trained on the isolated offspring and both the randomly selected individuals and offspring are transformed into the feature space (R^{60}). After processing them, their principal component iso-lines projected onto the original optimization space are showcased in fig. 3.3.



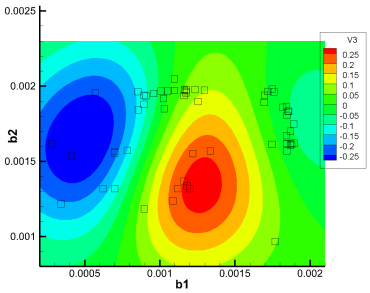
(a) 1^{st} principal component



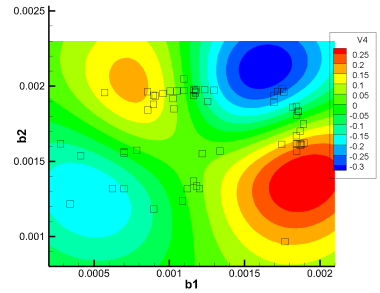
(b) 2^{nd} principal component



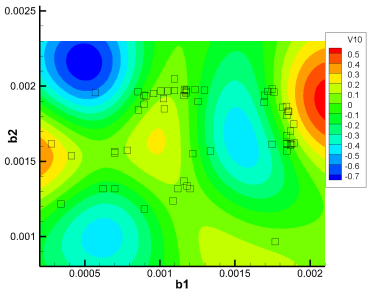
(c) 3^{nd} principal component



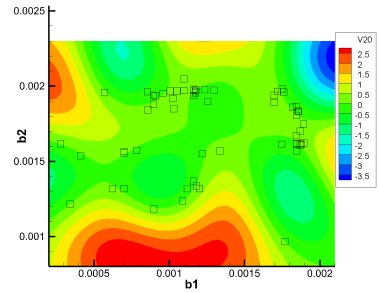
(d) 4^{th} principal component



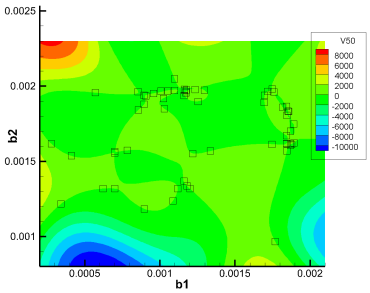
(e) 5^{th} principal component



(f) 10^{th} principal component



(g) 20^{th} principal component



(h) 50^{th} principal component

Figure 3.3: Ackley optimization problem: The 40^{th} generation offspring population (60 empty squares) and the iso-lines of the 8 first, out of the 60, principal components in the optimization space.

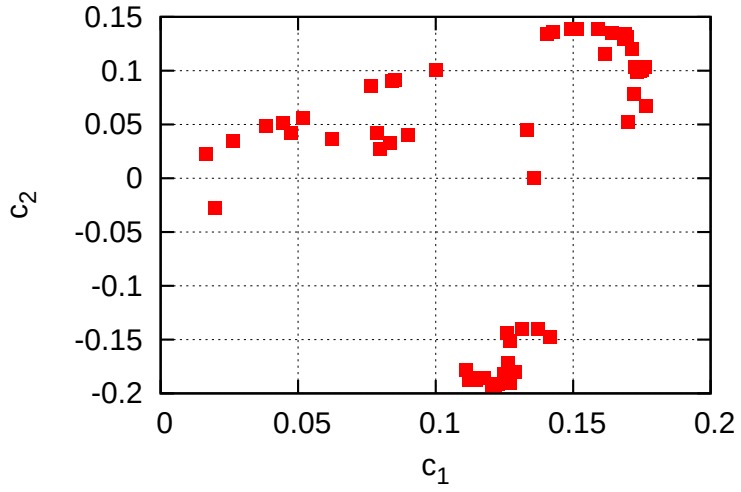


Figure 3.4: Ackley optimization problem: The first two principal components of the 40^{th} generation offspring population.

By examining fig. 3.3, three clusters, which practically classify the whole population, are readily identified. The first cluster (fig. 3.3a) is characterized by the highest values of the first PC (c_1) and consists of about 20 individuals located in the middle of the design space. The second cluster (fig. 3.3b) is characterized by the highest values of the second PC (c_2) and consists of about 20 individuals clustered in the right corner of the design space. The last clearly seen cluster (fig. 3.3c) is characterized by the highest values of the third PC (c_3) and consists of 10 individuals scattered to the bottom of the design space. The classification of the first two principal components can also be seen in fig. 3.4. Note that the most populated cluster corresponds to the first PC, whereas all the others correspond to the second PC. For the remaining PCs $c_i, i \geq 4$ (fig. 3.3), the offspring have more or less the same PC value, their variances become increasingly smaller, which shows that their importance is reduced. In such a case, the crossover operator is used mainly in the first two PCs, while neglecting (or making small changes to) the rest PCs. As a result, the important characteristics (as identified by the KPCA) are promoted and evolved into the next generation.

The KPCA creates a $\mathbb{R}^M$ feature space. Crossover, applied on the feature space, evolves the offspring towards the important directions (the ones with larger variances), thus, promoting them towards the optimal solutions. The demonstration case is revisited, but now the EA is assisted by the LPCA. During the evolution, the offspring population of the 10^{th} generation is extracted and used to train the LPCA. Its PCs along with the offspring are plotted into the design space in fig. 3.5. The directions of the LPCA are pointing towards the optimal solution ($\vec{b} = \vec{0}$), thus, promoting there new individuals. Moreover, in fig. 3.6, the probability of the offspring appearance in the design space using the SBX crossover, with and with-

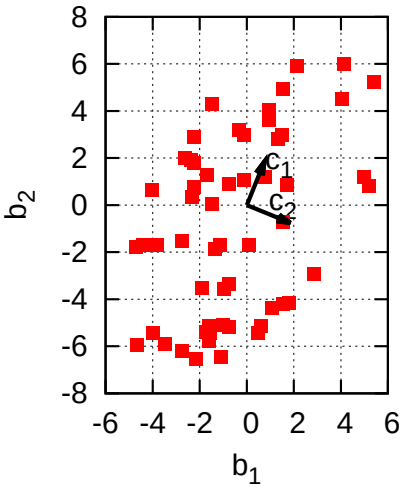


Figure 3.5: Ackley optimization problem: The offspring population of the 10^{th} generation along with the principal directions computed by the LPCA. Directions are scaled so as to be plotted along with the offspring population.

out LPCA is shown. The probability of SBX with LPCA seems to be rotated with respect to the one without PCA and, particularly, pointing towards the optimal solution. The use of the LPCA assigns higher offspring appearance probability in the less-populated direction and, thus, offspring generated after the application of crossover have higher probability to have better objective function(s).

3.3.2 PCA-Driven Mutation

The mutation operator is also applied into the feature space, as created by either the Kernel or Linear PCA. In the standard EA, the mutation probability is a user-defined value, which is uniformly applied to all design variables. When the PCA is used, this probability changes according to the variance of the corresponding variable. However, the overall mutation probability of an individual is kept constant and equal to the user-defined one. The PCA computes the feature space along with a variance for each direction of this space. This variance expresses the spread of the training patterns (herein, the current offspring population) in the corresponding direction. Mutation is responsible for the exploration of the design space. Thus, it is beneficial to give higher mutation probability to the less explored areas of the design space, which correspond to the directions of the feature space with low variances. Mutation probability is thus increased in these directions. In practice, the mutation probability (p^i_{mut}) for each principal component

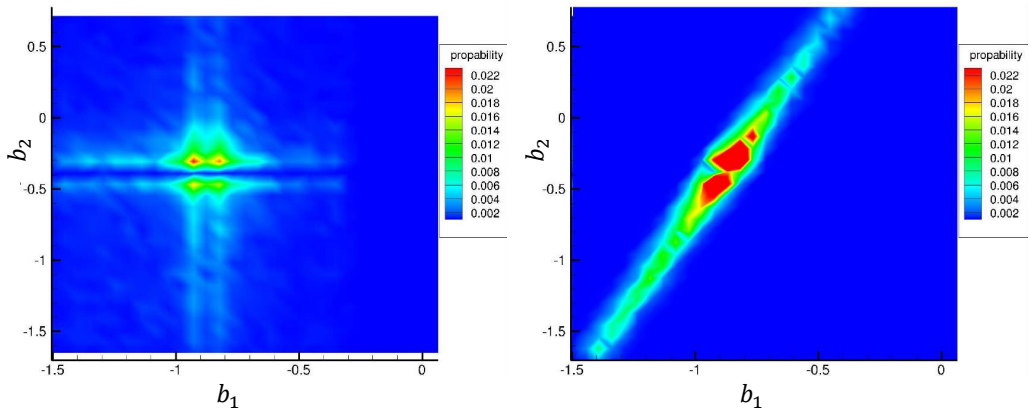


Figure 3.6: Ackley optimization problem: Probability of an offspring to appear in the design space after crossover with (right) and without (left) KPCA.

(superscript $i \in [1, M]$) is given by, [113],

$$p_{mut}^{i} = \alpha p_{mut} + M(1-\alpha)p_{mut}\frac{y_i}{\sum_{i=1}^{M} y_i} \tag{3.3.2.1}$$

where p_{mut} is a constant, user-defined global mutation probability, $\alpha \in [0, 1]$ and

$$y_i = \frac{\lambda_{max} - \lambda_i}{\lambda_{max} - \lambda_{min}} \tag{3.3.2.2}$$

where λ_i are the variances/eigenvalues of the current offspring population after being mapped into the feature space and $\lambda_{max} = \max\{\lambda_1, ..., \lambda_M\}$, $\lambda_{min} = \min\{\lambda_1, ..., \lambda_M\}$. The use of probabilities as defined in eq. 3.3.2.1 enhance the exploration capabilities of the mutation operator and direct search towards the not yet explored areas of the design space.

3.4 EAs with PCA-Truncated Metamodels

According to section 2.3.5, metamodels employed within MAEAs must be capable of predicting the objective and constraint (if any) functions with small computational cost (compared to the PSM). The metamodels prediction ability is greatly affected by the selected training patterns and the number (N) and correlation of input units (design variables). Metamodel's accuracy deteriorates for higher values

of N. Moreover, more training patterns are needed for accurate training, leading to an increased training time, too. This is the so-called "curse of dimensionality", the main challenge regarding metamodels, which arise when coping with complex objective functions having many design variables. In the past, different techniques have been used to reduce the number of input units the metamodel "sees" [155, 177]. A dimensionality reduction tool can cut-off the unwanted/insignificant input units and, thus, reduce the metamodel's training time and patterns, while increasing its prediction accuracy.

The PCA is also used as a dimensionality reduction tool. As mentioned, the eigenvectors computed by the PCA, define the feature space and each one of them is associated with a variance. These variances indicate how much scattered the data-set (current offspring) is along the feature space directions. High variance corresponds to highly scattered data along the corresponding direction. Directions with low variances have individuals clustered around the same value. These directions do not contribute important information to the metamodel's training, metamodel's predictions should not be greatly affected by the small changes in values occurred in these directions and, thus, they can be omitted. On the other hand, the high variance directions hold significant information for the training and should be retained. By cutting the N_{dr} insignificant (based on the variances) input units (directions) off, $N - N_{dr}$ input units remain, resulting in better predictions and faster training for the metamodels.

In the proposed MAEA variant, the PCA truncates the input variables. The PCA computes the feature space once at the beginning of each generation. After that, it is available to be used during the application of evolution operators (as mentioned in subsection 3.3) or/and during the pre-evaluation phase within the same generation. As in section 2.3.5, a "personalized" metamodel is built for each individual to be pre-evaluated. Firstly, the training patterns are selected from the individual's neighborhood. These together with the individual are transformed into the feature space computed by the PCA. N_{dr} PCs associated with the lowest eigenvalues (smaller variances) are truncated/cut-off. Thus, each metamodel is trained on patterns with less input units, which are also as uncorrelated as possible due to the PCA-based transformation, leading to more accurate predictions at lower cost. Regarding the LPCA, each truncation with $N_{dr} > 0$ is advantageous for the metamodel. When KPCA is used, its feature space has $M = \lambda$ directions. Thus, the truncated input units should be more than the difference between the design variables and the offspring population ($N_{dr} > M - N$), for the metamodels to benefit from the truncation.

Having established how the PCA is used to truncate the unnecessary input units from the metamodels, it is useful to check if the prediction ability of the metamodels becomes better by integrating the PCA. During the evolution of the demonstration case discussed above (subsection 3.3.1), offspring population of the 10^{th} generation was extracted. Both variants of PCA are using the offspring

population as the required data-set. A grid of uniformly distributed individuals is created in the vicinity of the offspring population (fig. 3.7). Firstly, all of them are evaluated on the PSM. Then, they are divided into two groups: 25 of them are used for training and the rest 100 for testing (fig. 3.7). A standard metamodel, a metamodel with the LPCA and one with the KPCA are trained and their prediction accuracies are finally tested on the testing patterns. The iso-lines of the objective function produced by the three different metamodels and the PSM are shown in fig. 3.8. Judging from the iso-line contours, the accuracy of the metamodel assisted by the PCA (fig. 3.8d) seems to be closer to the PSM in comparison with the standard metamodel. Differences between the metamodels assisted by the LPCA or KPCA are not that clear, but the one with the KPCA performs better as it can be seen by comparing their mean error (expressed as the mean difference between the predicted values and the real values over the testing patterns).

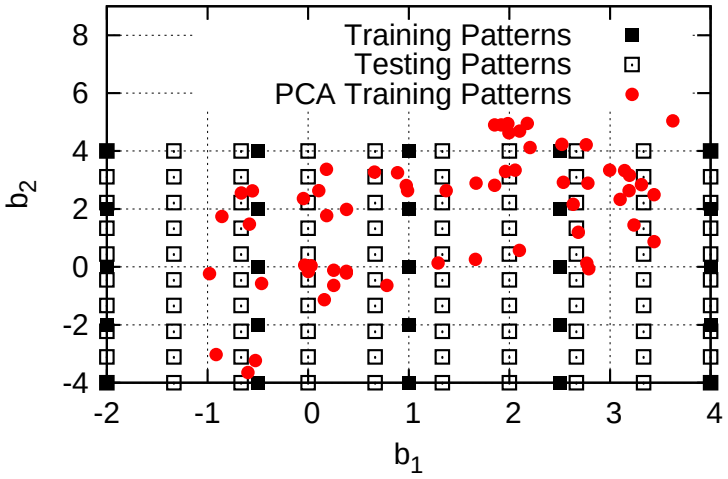
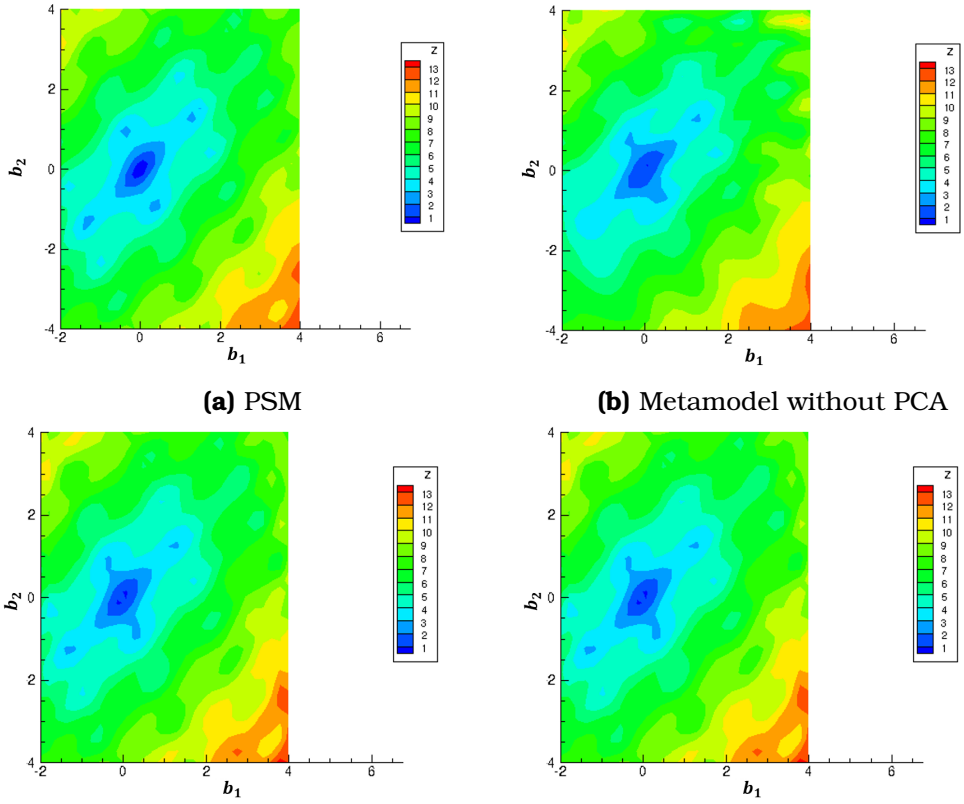


Figure 3.7: Ackley optimization problem: Training (filled squares) and testing patterns (empty squares) used for the metamodel. The 20^{th} generation offspring (filled circles) were used as training patterns for the KPCA.

A MAEA in which the PCA is employed during the training of metamodels is referred to as M(K)AEA for the KPCA and M(L)AEA for the LPCA. Therefore, M(K)AEA(K) denotes a MAEA with a dual use of the KPCA and M(L)AEA(L) with a dual use of the LPCA. Practically, the PCA is carried out once per EA generation. The resulting eigenvalues and eigenvectors can be used twice, i.e. during the evolution operators and the metamodel training,



(a) PSM **(b)** Metamodel without PCA

(c) Metamodel with LPCA **(d)** Metamodel with KPCA

Figure 3.8: Ackley optimization problem: Iso-lines of the objective function values evaluated by four different models. The use of the PCA improves the prediction accuracy of metamodels.

3.5 Mathematical Optimization Problems

In order to demonstrate the effects the presented methods have on the EA efficiency, four mathematical cases are presented for demonstration purposes, before switching to the benchmark cases.

The first mathematical optimization problem is a multi-dimensional ellipsoid test case, described by the separable function

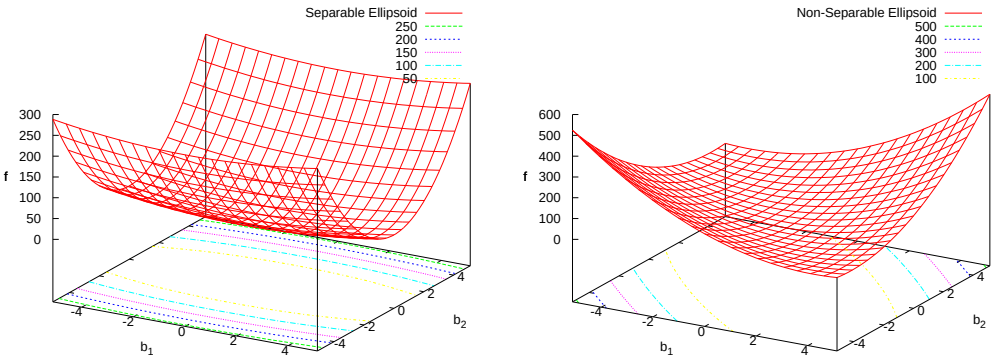
$$f(\vec{b}) = \sum_{i=1}^{N} a^{\frac{i-1}{N-1}} b_i^2 \tag{3.5.1}$$

where the value a determines the anisotropy of the objective function. Herein, a was set to 10, defining a fairly anisotropic function. Eq. 3.5.1 can easily be turned into a non-separable function by applying a rotation to the vector of optimization

variables expressed as

$$f(\vec{b}) = \sum_{i=1}^{N} a^{\frac{i-1}{N-1}} \sum_{j=1}^{N} R_{ij} b_j^2 \tag{3.5.2}$$

where $\mathbf{R}$ is a $N \times N$ rotation matrix. A 45^o rotation is applied in all optimization variables. Both functions with two optimization variables can be seen in fig. 3.9. The number of optimization variables is set to $N = 30$ to examine the effects the "curse of dimensionality" has upon EAs. Their bounds are set to $-100 < b_n < 100,\ n = 1, 30$. Based on the literature, the ellipsoid problem with $N = 30$ requires above 10^5 PSM calls to reach an objective function value close to zero. Both versions (separable or not) of this problem are investigated and are referred to as the ellipsoid separable and non-separable problems, respectively.



(a) Separable **(b)** Non-Separable

Figure 3.9: Ellipsoid Problem: Both problems are illustrated herein for two design variables.

The second mathematical optimization problem is a multi-dimensional shifted Rastrigin test case [137], expressed by

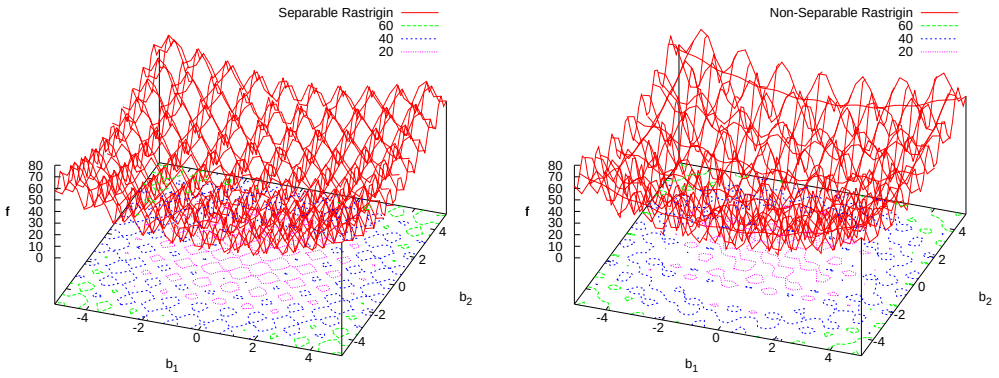
$$f(\vec{b}) = 10N + \sum_{i=1}^{N} \left(z_i^2 - 10cos(2\pi z_i) \right) \tag{3.5.3}$$

where $\vec{z} = \vec{b} - \vec{b}_{opt}$ with $\vec{z}, \vec{b}_{opt} \in \mathbb{R}^N$. $\vec{b}_{opt}$ is the optimal design vector given by $b_{opt,i} = i/N^2$ and is used to shift the optimal point from the start of the axes, which was the optimal design in the initial case. Even though this problem is not separable, it is a complex one due to the existence of many local minima, as shown in fig. 3.10. Moreover, the shift of the optimal design introduces extra difficulty.

Eq. 3.5.3 can easily be turned into a non-separable function by applying a rotation to the optimization variables vector and redefining $f(\vec{b})$ as

$$f(\vec{b}) = 10N + \sum_{i=1}^{N} \left(y_i^2 - 10cos(2\pi y_i) \right) \tag{3.5.4}$$

where $\vec{y} = \mathbf{R}\vec{z}$ with $\mathbf{R}$ being a $N \times N$ rotation matrix. A 30^o rotation per direction is applied in this case. Both functions with two optimization variables are plotted on the optimization space in fig. 3.10. Once again, the number of optimization variables is set to $N = 30$, Their limits are set to $-5.12 < b_n < 5.12,\ n = 1, 30$. Both versions (separable or not) of this problem are investigated and are referred to as the Rastrigin or the non-separable Rastrigin problem.



(a) Separable **(b)** Non-Separable

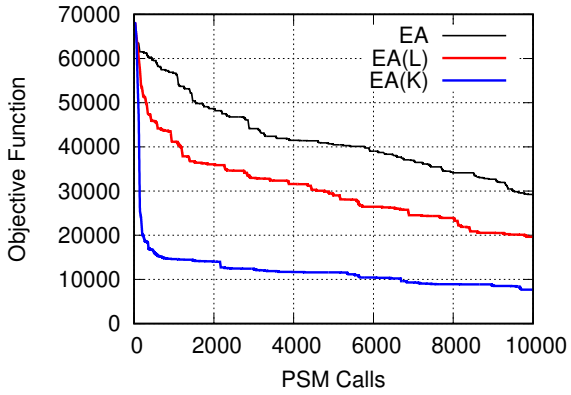
Figure 3.10: Rastrigin Problem: Both problems are illustrated herein for two design variables.

Having defined four mathematical optimization problems with different characteristics each, we can now demonstrate the gains of using all the presented methods.

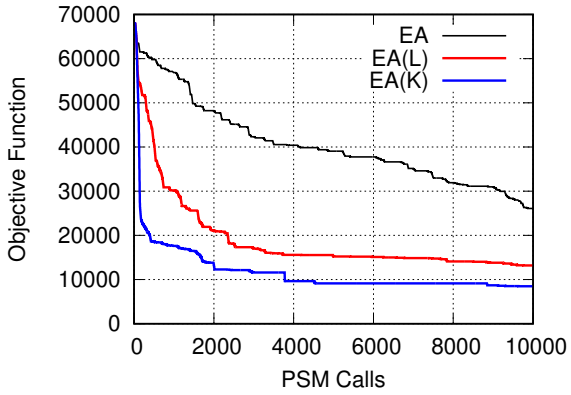
3.5.1 Demonstration of EA(L) and EA(K) Performance

The problems presented in section 3.5 are investigated to demonstrate the effect the PCA has on the EAs performance when its role is to exclusively assist the evolution operators. Each of them using three EA variants, the standard one, the EA(L) and the EA(K). For each one of these low-cost mathematical case, 10 runs per variant were performed with different RNG seeds, so as to reduce randomness.

The EA for the ellipsoid problems has $\mu = 15$, $\lambda = 30$ and $\epsilon = 15$. The PCA starts after 30 evaluations i.e. by the end of the first generation. A comparison of the convergence histories of the optimization runs is presented in fig. 3.11a for the separable problem and fig. 3.11b for the non-separable one. Note that the plotted convergences are the mean objective functions values of 10 runs performed with different RNG seeds. The EA(K) outperforms all other variants. It achieves an optimal solutions which has three times less objective function values in comparison with the one found by standard EA and two times less than the one found by the EA(L). The EA(L) outperforms the EA in either case. As expected, it performs optimally in the non-separable problem in which the optimal solution found (within the pre-defined computational budget) is approximately 2.5 times better than the one computed by EA.



(a) Separable

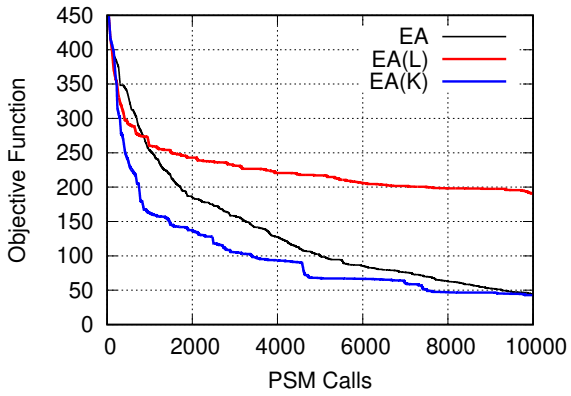


(b) Non-Separable

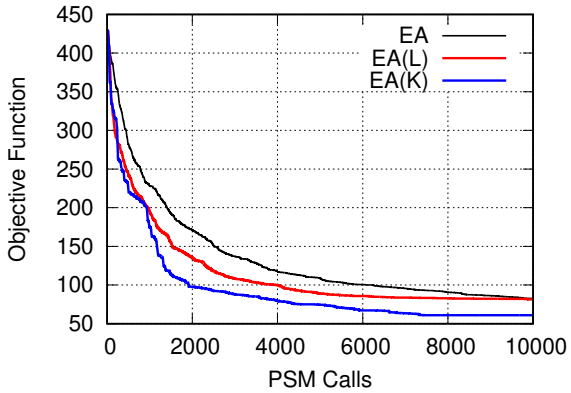
Figure 3.11: Ellipsoid Problem: Convergence histories of EA, EA(L) and EA(K).

For the Rastrigin problems (separable and non-separable), a $(10, 20)$ EA is used. The PCA starts after either 20 evaluations or the first generation. Fig. 3.12a corresponds to the convergence histories for the separable problem, while fig. 3.12b for the non-separable one. The EA(L) may not perform very well in a sep-

arable complex case, such as the separable Rastrigin problem, but performance improves significantly when the problem turns to a non-separable one. On the other hand, the EA(K) performs well in the separable problem; however, in the non-separable problem, the use of KPCA boosts the EA performance by making it $\sim 65\%$ faster.



(a) Separable



(b) Non-Separable

Figure 3.12: Rastrigin Problem: Convergence histories of EA, EA(L) and EA(K).

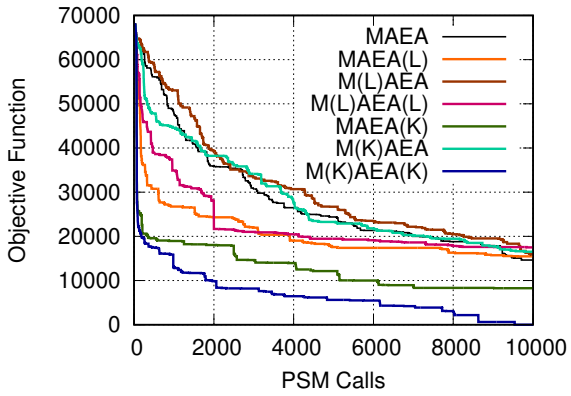
3.5.2 Demonstration of M(L)AEA(L) and M(K)AEA(K) Performance

To demonstrate the gain in performance from the use of the proposed PCA-assisted metamodels, the mathematical problems of section 3.5.1 are revisited. The optimization of these problems is performed using the MAEA, MAEA(L), MAEA(K), M(L)AEA, M(K)AEA, M(L)AEA(L) and M(K)AEA(K). Each one of them is repeated 10 times with different RNG seeds and the convergence histories presented below re-

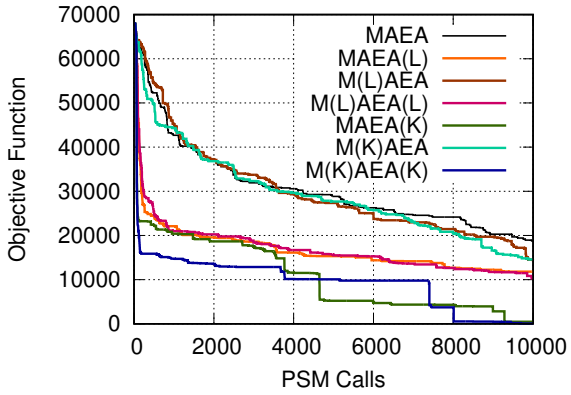
sult from averaging all these runs. Since in a previous chapter, is has been shown that the MAEA outperforms the EA, this comparison is not repeated herein.

Regarding the ellipsoid problems, the EA has $\mu = 15$,$\lambda = 30$ and $\epsilon = 15$. The pre-evaluation phase and the PCA usage(s) start after the first 20 individuals are evaluated on the PSM and stored in the DB. Metamodels are trained on 30 to 60 training patterns and only the 2 or 3 most promising individuals are re-evaluated on the PSM in each generation. When the PCA is used as a dimensionality reduction tool, $N_{dr} = 15$ input units are cut-off from the initial $N = 30$ the metamodels "see". The convergence histories, figs. 3.13a and 3.13b, correspond to the separable and non-separable problems, respectively. The MAEAs variants using the PCA only to assist the metamodels, namely M(L)AEA, M(K)AEA, seem to provide negligible (in the non-separable or separable problem) gains. This can be attributed to the fact that the evolution operators are applied to design spaces other than the one metamodels are trained on. The metamodels are trained with patterns transformed into the feature space produced by the PCA. The MAEA(L) has the same performance as the MAEA for the separable problem, but for the non-separable one it improves the optimized solution by $\sim 60\%$. The MAEA(K) greatly affects the performance over the MAEA. $\sim 100\%$ improvement is recorded on the separable problem and $\sim 300\%$ on the non-separable one. The two-fold usage of LPCA (M(L)AEA(L)) does not show any significal performance improvement in comparison with the LPCA to assist only the evolution operators (MAEA(L)). Finally, the M(K)AEA(K) outperforms all the other variants leading to $\sim 300\%$ overall improvement of the optimal solution in either case.

Moving on to the Rastrigin problems, a $(10, 20)$ MAEA is used. Having evaluated 20 individuals on the PSM, the use of metamodels and PCA starts. 30 to 60 individuals are selected to train each metamodel. From the most "promising" (according to the metamodels predictions) individuals of each generation only 2 or 3 of them are re-evaluated on the PSM. The 15 most insignificant based on the PCA variables are cut-off for the M(L)AEA, M(K)AEA, M(L)AEA(L) and M(K)AEA(K) runs, reducing the input units to half of the initial ones. In fig. 3.14a, the convergence histories of the optimization performed for the separable problem are presented, whereas fig. 3.14b repeats the same for the non-separable problem. In the separable problem, the use of LPCA in the evolution operators sees to deteriorate the EA performance, probably because the problem is neither non-separable nor anisotropic, thus any possible transformation on the design space does not offer any advantage over the original one. The M(L)AEA improves the optimized solution for $\sim 20\%$ basically due to the reduction in the number of input units the metamodels "see". On the other hand, the use of KPCA offers great performance improvement. The M(K)AEA's solution is $\sim 43\%$ better than the one found by the MAEA. The M(K)AEA(K) and MAEA(K) converge around the same value, which is by $\sim 90\%$ better than the MAEA solution. For the non-separable problem, variants using LPCA perform better, with the MAEA(L) providing $\sim 6\%$ improvement over



(a) Separable



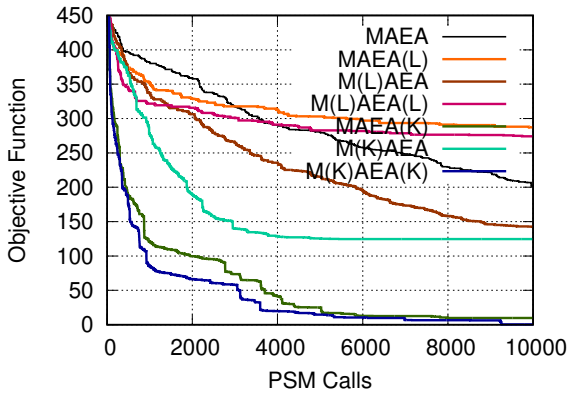
(b) Non-Separable

Figure 3.13: Ellipsoid Problem: Convergence histories of MAEA, MAEA(L), MAEA(K), M(L)AEA, M(K)AEA, M(L)AEA(L) and M(K)AEA(K).

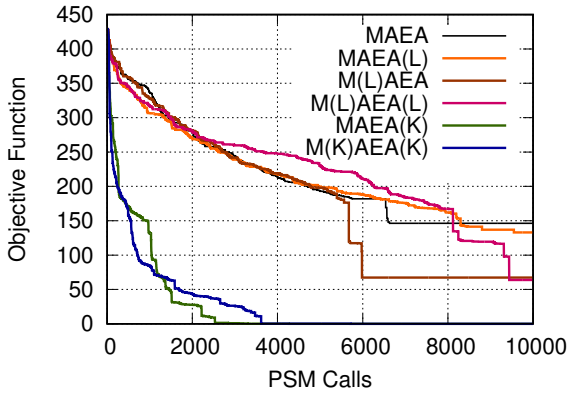
the MAEA. The M(L)AEA and M(L)AEA(L)provides $\sim 50\%$ improvement over the MAEA. In this case, the LPCA seems to identify the rotation of the design space and, thus, assist the optimization to converge better. Moreover, the use of KPCA speeds-up the convergence of the MAEA. Both MAEA(K) and M(K)AEA(K) reach the global optimal solution after ~ 3000 evaluations, while the MAEA is unable to do so.

3.6 Benchmark Cases Revisited

The aforementioned variants of EAs including PCA, either the Linear or Kernel one, are demonstrated in the four benchmark cases presented in section 2.4. The performance of the new variants of EAs are compared with the ones used in section 2.4.



(a) Separable



(b) Non-Separable

Figure 3.14: Rastrigin Problem: Convergence histories of MAEA, MAEA(L), MAEA(K), M(L)AEA, M(K)AEA, M(L)AEA(L) and M(K)AEA(K).

3.6.1 Benchmark Case 1

In addition to the optimization of section 2.4, a $(20, 40)$ MAEA(L), MAEA(K), M(L)AEA(L) and M(K)AEA(K) are used. In all cases, the use of PCA for the evolution operators started after the 2nd generation. Dimensionality reduction during the training of the metamodel driven by the PCA, begins at $T^{MM} = 40$, and reduces the metamodel sensory units by half. This means that RBF networks with 6, rather than 12, input units are trained.

Convergence histories of the MAEA, MAEA(L), M(L)AEA(L), MAEA(K) and M(K)AEA(K) runs are computed in fig. 3.15. The MAEA curve, fig. 2.14, is also repeated for the sake of comparison. It can be seen that the use of PCA for either only the evolution operators or both the evolution operators and metamodel training, leads to a faster convergence. From this case, at least, it is difficult to conclude whether the Linear or Kernel PCA performs better.

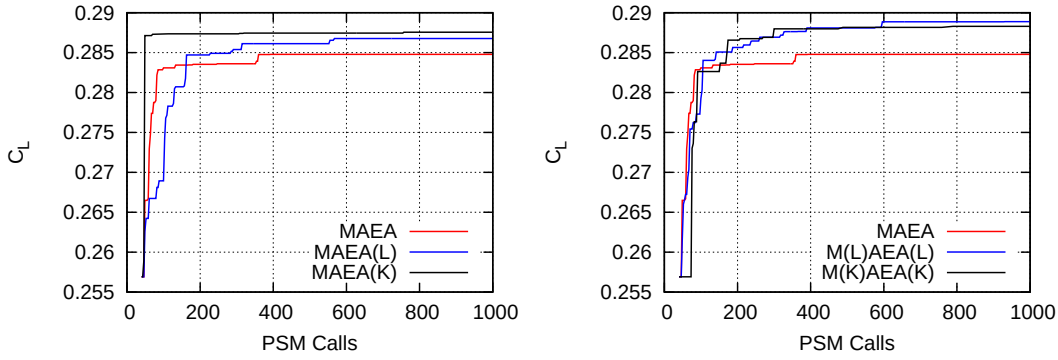


Figure 3.15: Benchmark Case 1: Comparison of the convergence histories of MAEA, MAEA(L) and MAEA(K) (left) and MAEA, M(L)AEA(L) and M(K)AEA(K) (right) in terms of the number of CFD evaluations (PSM Calls).

3.6.2 Benchmark Case 2

The shape optimization of an isolated wing with respect to max. lift coefficient (C_L) and min. drag coefficient (C_D), (second benchmark) is examined anew using the PCA. A $(10, 20)$MAEA(L), MAEA(K), M(L)AEA(L) and M(K)AEA(K) are used. Regarding the metamodels, the setting is exactly the same. The PCA starts after the first generation and, in the LCPE phase, the PCA serves as a dimensionality reduction tool, cutting 12 out of the 24 sensory units of the RBF networks. Comparisons of the convergence histories of the aforementioned runs and the MAEA run (shown in section 2.4) are presented in fig. 3.16. The fronts of non-dominated solutions for a single RNG seed are presented in fig. 3.17. Even though the final fronts of MAEA(L) and MAEA(K) do not dominate the front computed by the MAEA, the convergence speed has increased when PCA is used. It is possible that the initial design space is ("almost") separable and the PCA cannot provide further improvement. However, if the KPCA is used for both metamodels and evolution operators, the M(K)AEA(K) produces sensibly better front and convergence than any other presented EA variant. Its great advantage comes from the well-spread front (better than any other front thus far). Herein, the KPCA variants of EA seems to outperform the corresponding LPCA ones.

3.6.3 Benchmark Case 3

The third benchmark case is revisited and optimized for two objective functions for max. lift coefficient (C_L) and min. moment coefficient (C_M). A $(10, 20)$MAEA(L), MAEA(K), M(L)AEA(L) and M(K)AEA(K) are used to perform the optimization and their results are compared with the corresponding MAEA. Both usages of PCA start after the 2nd generation. Comparison of the convergence histories of the

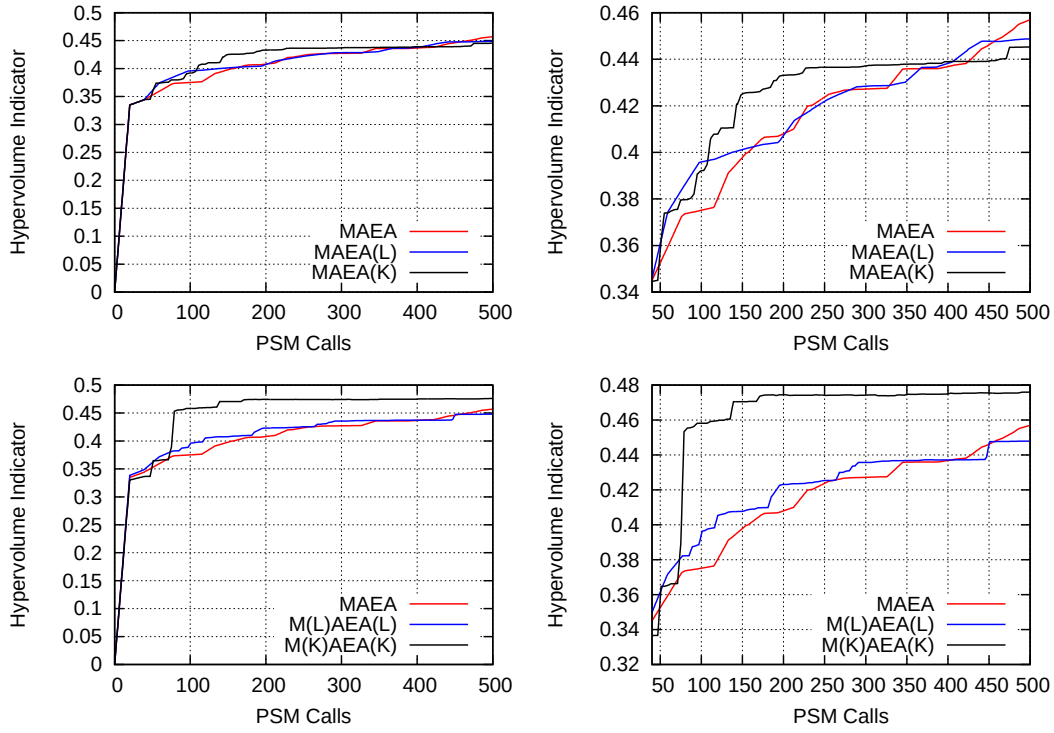


Figure 3.16: Benchmark Case 2: Left: Comparison of the convergence histories of MAEA, MAEA(L), MAEA(K) (top) and MAEA, M(L)AEA(L), M(K)AEA(K) (bottom). Right: Close-up views after the activation of metamodels and PCA.

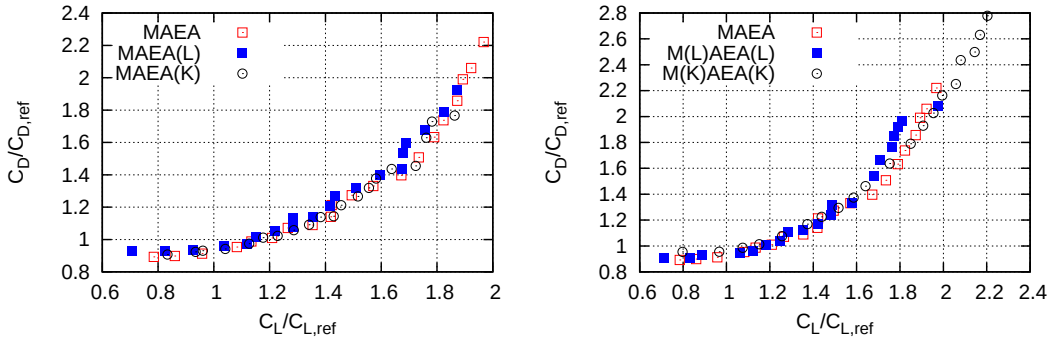


Figure 3.17: Benchmark Case 2: Comparison of the fronts of non-dominated solutions resulted from MAEA, MAEA(L) and MAEA(K) (left) and MAEA, M(L)AEA(L) and M(K)AEA(K) (right). Both objective functions are normalized using the corresponding values of the reference wing geometry.

new runs and the MAEA is presented in fig. 3.18 and the corresponding front of non-dominated solutions in fig. 3.19. Herein, the two-fold usage of PCA seems to perform poorly, whereas the assistance of the KPCA on the evolution operators

increases the performance of the optimization. This indicates that all the design variables are important. When the PCA truncates half of the variables, metamodels perform poorly, deteriorating the convergence speed, as it can be seen from the M(L)AEA(L) and M(K)AEA(K) convergence plots. In this case, the use of LPCA is significantly inferior to the use of the KPCA.

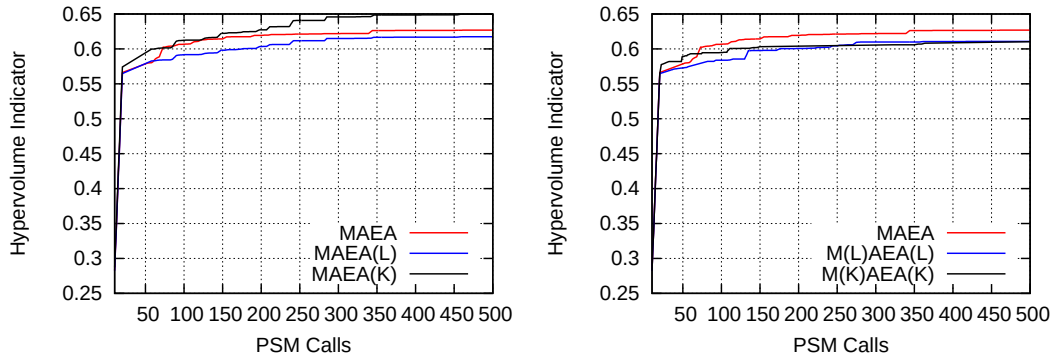


Figure 3.18: Benchmark Case 3: Comparison of the convergence histories of MAEA, MAEA(L) and MAEA(K) (left) and MAEA, M(L)AEA(L) and M(K)AEA(K) (right).

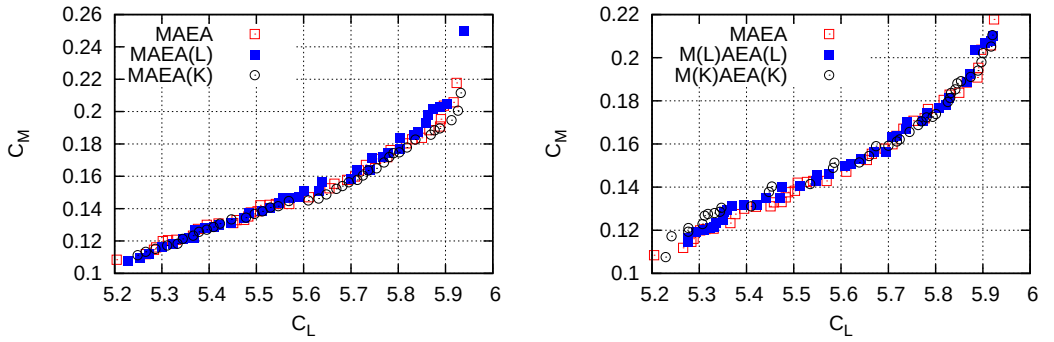


Figure 3.19: Benchmark Case 3: Comparison of the fronts of non-dominated solutions resulted from MAEA, MAEA(L) and MAEA(K) (left) and MAEA, M(L)AEA(L) and M(K)AEA(K) (right).

3.6.4 Benchmark Case 4

The shape of a 2D section of the TurboLab compressor stator blade is optimized for max. deviation of the exit flow from the axial direction (max. $\Delta\alpha$) and minimum total pressure losses (min. Δp_t). Herein, this case is revisited and optimized using a $(15, 30)$MAEA(L), MAEA(K), M(L)AEA(L) and M(K)AEA(K), with the PCA starting after the first generation. Fig. 3.20 and 3.21 shows the convergence of all the PCA

variants of EA and MAEA based on the hypervolume indicator and the front of non-dominated solutions produced by a single RNG seed. In this case, both MAEA(L) and M(L)AEA(L) performs worse than the standard MAEA, which indicates the non-linear correlation between the design variables. Indeed, the MAEA(K) and M(K)AEA(K), which are able to uncouple non-linearly correlated design variables, perform better than other EA variants.

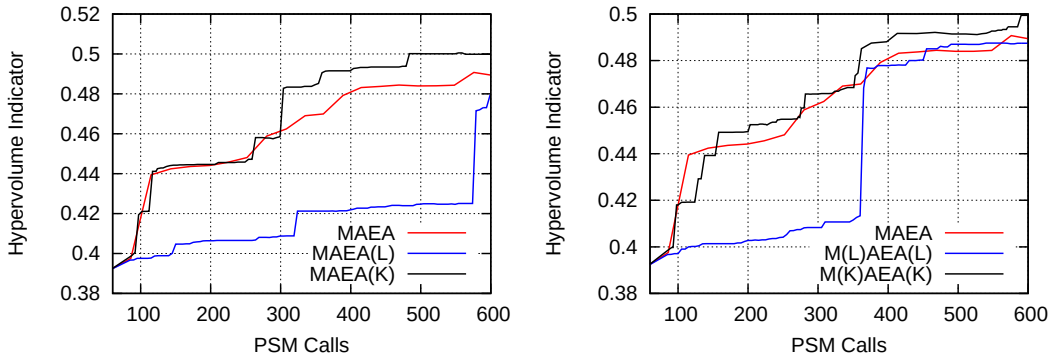


Figure 3.20: Benchmark Case 4: Comparison of the convergence histories of MAEA, MAEA(L), MAEA(K) (left) and MAEA, M(L)AEA(L), M(K)AEA(K) (right).

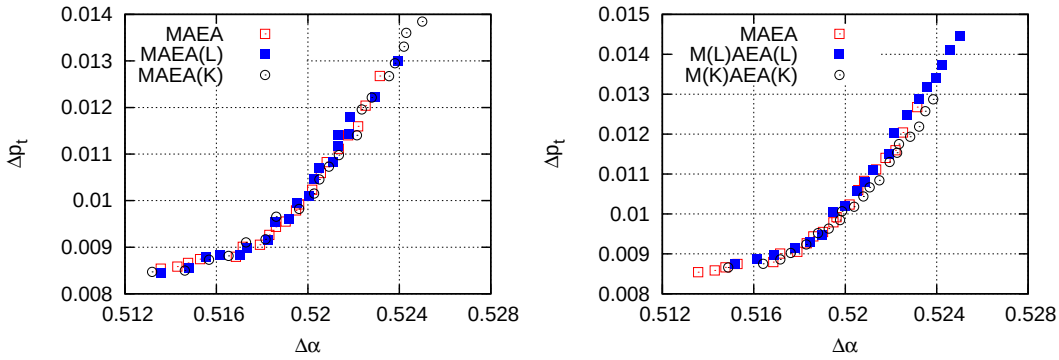


Figure 3.21: Benchmark Case 4: Comparison of the fronts of non-dominated solutions computed by MAEA, MAEA(L), MAEA(K) (left) and MAEA, M(L)AEA(L), M(K)AEA(K) (right).

Chapter 4

PCA-Assisted Hybrid Algorithm Combining EAs and Adjoint Methods

As mentioned before, solving SOO or MOO problems using EAs becomes prohibitively expensive if the PSM used to evaluate candidate solutions is computationally demanding, as it is the case for the CFD solvers used in this thesis. Even if the aforementioned methods/techniques assist the EAs reduce the number of calls to the PSM, the optimization can still be expensive in some cases. In contrast, Gradient-Based (GB) optimization methods require a considerably smaller number of calls to the PSM, but they can easily be "trapped" into local minima. For the GB methods to work, the computation or approximation of the gradient(s) of the objective function(s) with respect to the design variables are required. For problems having a CFD solver as the PSM, the gradients can be computed with the adjoint method at a cost which is more or less equal to that of solving the flow problem. A reasonable extension is to combine the advantages of both methods (EAs and GB methods) in a hybrid optimization algorithm, which performs better than its individual components.

In general, hybrid optimization algorithms combine two or more optimization methods, for solving the same problem by periodically switching between them, [118]. The desired features of each optimization method are combined. Hybrid optimization algorithms usually combine a variant of EA and a GB method [16, 162], since the former is best suited for the exploration of the design space (randomized search) and the latter for the refinement of promising solutions (deterministic search). Memetic Algorithms (MA) [127, 160, 4] are a sub-category of hybrid methods, which couple EAs with local search algorithms. In a MA [106, 78, 128], the local search method (usually a GB one) selectively updates the design vector and/or the objective vector of some offspring. MA can further be divided into two main categories, namely the Baldwinian [10] and the Lamarckian search algorithm [178]. In the former, each individual of the population has the ability to learn new behaviors, so as to adapt to the changing environment. These behav-

iors have an effect on the genetic makeup of the next generation through natural selection. In the Lamarckian search algorithm, parents can pass on genetic material acquired through adaptation during their lifetime, directly to their offspring. Some of the parents characteristics are inherited to the offspring while bypassing the evolution. Multi-meme [108, 147], hyper-heuristic [100] and self-generating MA [109, 161] are some variants of MAs developed thus far. The first two select individuals to perform local refinement in a smart way. They use a pool of individuals, each of which is rewarded based on its ability to generate local improvements so far. Their rewards determine whether they will be further refined or not. In self-generated MA, the way communication takes place between the EA and the local search is not known 'a-priori', as was the case in all aforementioned variants. A rule-based local search is used to introduce individuals in the evolution of the EA, thus recognizing regularly repeated features or patterns in the objective space, such as local optima, to speed-up the convergence, [121].

This thesis presents a new hybrid algorithm which is suitable for MOO problems and combines a variant of EA with a GB method. It belongs to the Lamarckian MA as it will become clear below. Aiming at maximum efficiency, the M(K)AEA(K), proposed in the section 3, is used instead of a standard EA, because it has been demonstrated to perform better than any other EA variant. Regarding the GB method, this is assisted by the continuous adjoint method [56, 130], which computes gradients by first differentiating the flow equations to produce the adjoint Partial Differential Equations (PDEs) and, then, discretizing and numerically solving them. With the adjoint method, the number of design variables does not reflect on the cost for computing the gradients. During the last years, the PCOpt/NTUA Unit develops both discrete and continuous adjoint-based methods for computing the objective function derivatives, details can be found in section 1.2. Adjoint to both the PUMA solver and the cut-cell in-house CFD solver have been developed. Hereafter, the PUMA solver computes the required gradients using continuous adjoint, for all the cases, instead of one industrial case, in which the cut-cell CFD solver is used.

If Memetic Algorithms are applied in SOO problems, the direction in which the GB method updates the individuals is straightforward. The individuals move along the computed gradient or its negative direction, depending on whether the objective is minimized or maximized. In MOO problems, as many gradients as the objective functions are computed. The descending or ascending direction for each individual to be refined should be a combination of these gradients [31]. It should always point towards the improvement of all objective functions, for the new individual to be an improvement to the previous one and (hopefully) the previous front of non-dominated solutions. Hereafter, this direction will be referred to as the Pareto Advancement Direction (PAD), fig. 4.1, because it is one direction that gradually improves/advances the current front. A technique defining the PAD, based on an existing front of non-dominated solutions, has

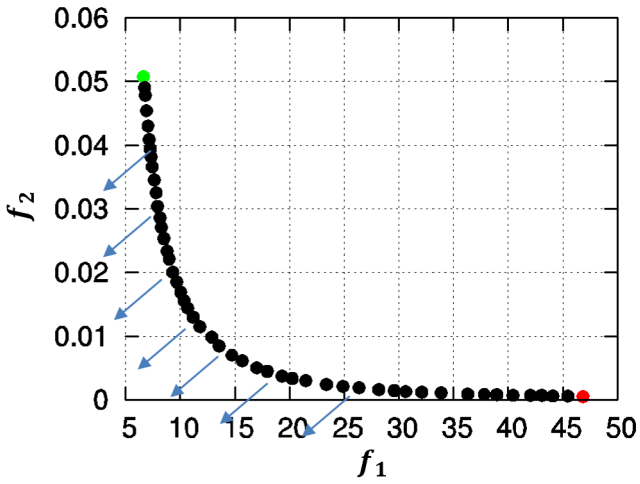


Figure 4.1: PAD for each individual of a given front in a problem with two objectives.

been proposed in [106, 78] from the PCOpt/NTUA Unit, see section 4.1. In these papers dealing with MOO problems, the objective functions are combined into a scalar utility value according to the SPEA2 technique [184]. Thus, the derivatives of a differentiable approximation to the SPEA2 utility function together with the gradients of the objective functions were used for the GB refinement.

Herein, the LPCA [2, 74] recomputes the PAD, in each generation. The objective function values of the elite members in each generation are used as training patterns for the LPCA. The LPCA gives the principal components of the objective space. The principal component with the smallest variance is aligned with the PAD. This direction is "perpendicular" to the current fronts hyper-surface and points towards the simultaneous minimization of all objective functions. Having computed the PAD, the GB method updates the design variables of the selected individual(s) by moving along its direction so as to improve the current front of non-dominated solutions. In the following section, the new hybrid algorithm, called PCA-assisted hybrid method, is discussed in detail.

4.1 SPEA2-based Hybrid Optimization Algorithm.

In the hybrid method proposed in [106, 78], weights $w_j = \frac{\delta \phi}{\delta f_i}^{SPEA2}$ are computed as the derivatives of the SPEA2 function. A major difference from the PCA-Assisted Hybrid Algorithm proposed in the following section, is that the method of [106, 78] uses a different PAD for each individual undergoing steepest descent within the same generation. In the objective space, directions "perpendicular" to the SPEA2 utility function isoline or isosurface for the non-dominated solutions imply that such a displacement improves all objective functions, fig. 4.2. As mentioned

before in details (2.3.3), in SPEA2,

$$\phi_j^{SPEA2} = \frac{1}{2+d_j} + \sum_{k \in K_j} s_k \mathcal{H}\left(\sum_{m=1}^{M_o} \mathcal{H}(f_{j,m} - f_{k,m}) + 0.5 - M_o \right)$$

where M_o is the number of objectives and $f_{j,m}$ is the m^{th} objective function value of the j^{th} individual. It is basically the sum of the so-called strength (s) and density (d) function values of all individuals dominating the j^{th} individual. Here, K_j is the population size excluding the j^{th} individual and $\mathcal{H}$ is the Heaviside function. In its standard form, the SPEA2 function cannot be differentiated, since the $\mathcal{H}$ is not differentiable. In [78], a sigmoid function replaces the non-differentiable part yielding a closed-form expression for $\frac{\delta\phi}{\delta f_j}$. Note that, in [78], the f_j included in the gradient computation is approximated by metamodels (unless this individual has been evaluated on the PSM in a previous generation), which yields a less accurate PAD. A restriction of the previous method is that, for the purpose of compatibility, the EA should exclusively rely upon the SPEA2 technique. This restriction is abolished by implementing the PCA-Assisted Hybrid Algorithm.

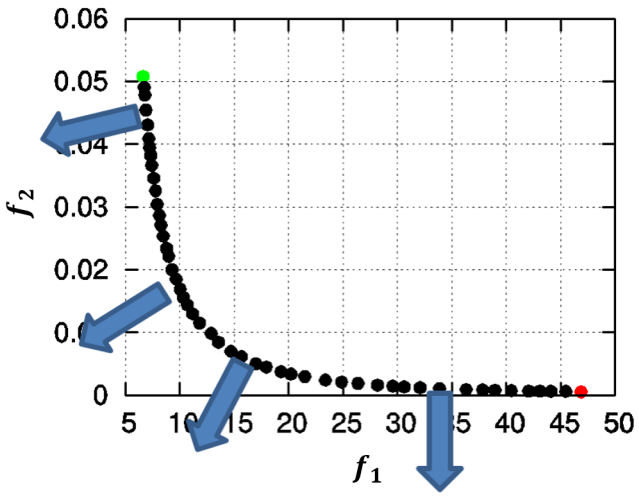


Figure 4.2: Direction PAD for each individual computed by the SPEA2-based Hybrid Algorithm [106, 78].

4.2 The PCA-Assisted Hybrid Algorithm, in detail

This section dives into the details of the PCA-Assisted Hybrid Algorithm. As in the MAEA described in section 2.3.5, once there are enough archived results from previous evaluations on the PSM to rely upon, all S_λ^g members are pre-evaluated using local personalized metamodels. Then, the top λ_e of them are re-evaluated

on the PSM. In the PCA-Assisted Hybrid Algorithm, the top λ_{GB} out of them ($\lambda_{GB} \leqslant \lambda_e$) are selected to undergo a single descent step by the GB method. This step, which involves the computation of gradients, generates the first λ_{GB} members of the next generation offspring S_λ^{g+1}. The remaining $\lambda - \lambda_{GB}$ members of S_λ^{g+1} result from the evolution operators as applied in the standard EA. As mentioned before, the PCA-Assisted Hybrid Algorithm is considered to be a Lamarckian MA, because it promotes the refined individuals directly to the next generation (inheritance). Before performing any refinement or call to the adjoint solver (gradient computation), the error among the objective functions values ($\vec{f}$) computed by the PSM and the metamodel(s) are checked. If they are close enough, which is a clear indication that the metamodel accuracy is acceptable, the algorithm proceeds with the PAD computation for the top λ_{GB} individuals (and the subsequent GB refinement). This ensures that gradient computations are carried out only for really promising individuals.

As mentioned in section 2.3.3, ϕ is the scalar utility function which is computed using the objective vector ($\vec{f}$). Once, the PAD, equal to $\nabla\phi = (\frac{\delta\phi}{\delta b_1}, \ldots, \frac{\delta\phi}{\delta b_N})$, at the λ_i offspring of generation has been computed by the adjoint method, the offspring's design vector is updated by performing a single descent step, as follows

$$\vec{b}_{\lambda_i}^{g+1} = \vec{b}_{\lambda_i}^{g} - \eta \nabla\phi(\vec{b}_{\lambda_i}^{g}) \tag{4.2.1}$$

where η is a user-defined step. The outcome of eq. 4.2.1 , $\vec{b}_{\lambda_i}^{g+1}$, is directly injected into the new offspring population (S_λ^{g+1}), without being affected by the evolution operations.

In MOO, the simplest way to compute the scalar utility function ϕ (having in mind that this is used by the GB method) is by concatenating the M_o objective functions multiplied with weights w_j as

$$\phi(\vec{b}) = \sum_{j=1}^{M_o} w_j f_j(\vec{b}) \tag{4.2.2}$$

By differentiating eq. 4.2.2 , the gradient of ϕ becomes equal to the weighted sum of the gradients of all objective functions,

$$\nabla\phi(\vec{b}) = \sum_{j=1}^{M_o} w_j \frac{\delta f_j}{\delta \vec{b}} \tag{4.2.3}$$

The automatic selection of weights w_j is crucial since these determine the PAD. In each generation, the PCA-Assisted Hybrid Algorithm computes a single set of weights, which are used by all the individuals to undergo refinement. These

weights are computed at the beginning of each generation and are known before any call to the CFD or adjoint solver. For each individual, if all f_j are able to be computed by a single call to the CFD solver, then the gradients of f will also result from a single call to the adjoint solver developed by differentiating eq. 4.2.2 . If the weights were not known before the gradient computation, the adjoint solver would have been called as many times as the number of objective functions.

In the PCA-Assisted Hybrid Algorithm, LPCA computes the weights used in eq. 4.2.2 . This usage should not be confused with the previous usages of PCA inside the EA presented in chapter 3. In each generation, the data set $\mathbf{D}$ used to train the LPCA comprises the M_o objective function values of the ϵ current elite individuals, with one objective vector per column of $\mathbf{D}$. It is important that the elite set is exclusively populated by individuals exactly evaluated on the PSM. The elite individuals define the front to be improved, thus the "almost perpendicular" direction to it should be the PAD. This is characterized as "almost perpendicular" because there is not a single perpendicular direction at all points of the front in the M_o objective space.

The $M_o \times M_o$ covariance matrix $\mathbf{P}$ of $\mathbf{D}$ is computed as $P_{ij} = \frac{1}{M} \sum_{k=1}^{M} D_{ki} D_{kj}$ where D_{ki} is the f_i value of the k^{th} individual. The eigenvalues $\mathbf{\Lambda}$ and eigenvectors $\mathbf{U}$ defining the new feature space are computed by solving the eigenproblem $\mathbf{P} = \mathbf{U}\mathbf{\Lambda}\mathbf{U}^T$ through the Singular Value Decomposition method. The equation which maps any objective vector $\vec{f}$ into the feature space, is given as

$$\mathbf{c} = \mathbf{c}(\vec{f}) = \mathbf{U}(\vec{f} - \vec{\mu}), \ \mathbf{c} \in \mathbb{R}^{M_o} \tag{4.2.4}$$

where $\vec{\mu}$ is the vector of mean values of the objective functions of $\mathbf{D}$. The elements of $\vec{c}$ correspond to directions with variances in descending order. The direction with the smallest variance is the one along which the members are comparatively less scattered and is almost perpendicular to the current front of non-dominated solutions. Thus, the GB method should update the elite members by making them move along this direction.

Having computed the transformation from the design to the feature space, the gradients $\frac{\delta\vec{\phi}}{\delta\vec{b}}$ are transformed there with the equation $\frac{\delta\vec{c}}{\delta\vec{b}} = \mathbf{U}\frac{\delta\vec{\phi}}{\delta\vec{b}}$ which results by differentiating eq. 4.2.4 ($\mathbf{U}$ is constant w.r.t. the design variables). Then, the M_o^{th} row of the $\frac{\delta\vec{c}}{\delta\vec{b}}$ matrix replaces $\nabla\phi(\vec{b})$, in eq. 4.2.1 , and forming the PAD in this case. Instead of computing each gradient of $\vec{\phi}$ separately, a single adjoint run computes directly the M_o^{th} row of $\frac{\delta\vec{c}}{\delta\vec{b}}$ and each GB refinement costs as much as a single call to the PSM. This is possible only in problems in which objectives can be computed with a single call to the PSM.

The PCA-Assisted Hybrid Algorithm is applicable with any scalar utility function overcoming the restriction of the previous hybrid method (the method in [106, 78] can be used, theoretically at least, only with SPEA2). The scalar utility

function solely affects the EA component of the hybrid algorithm. In the method of [106, 78], each individual had its unique descent direction; in contrast, in the PCA-Assisted Hybrid Algorithm, the PAD is the same for all individuals, varying only from generation to generation or, practically, whenever the training set (elite) changes. Lastly, the PCA is trained with the objective function values of the elite set evaluated on the PSM, in contrast to the method of [106, 78] which basically uses metamodel-based approximations. Thus, the computed PAD is thought to be more precise and reliable than the one computed by the method of [106, 78]. The pseudo-code and the corresponding flowchart for the PCA-Assisted Hybrid Algorithm are presented in algorithm 1 and fig. 4.4.

For better understanding of the PAD along which the GB method updates the individuals, a simple mathematical example with two design variables ($b_i \in [-10, 10]$, $i = 1, 2$; $N = 2$) and two objectives to be minimized is presented

$$f_1 = b_1^2 + b_2^2, \;\; f_2 = (b_1 - 2)^2 + (b_2 - 2)^2 \tag{4.2.5}$$

The EA used in the hybrid algorithm has $\mu = 10$, $\lambda = 20$ and $\varepsilon = 20$. Fig. 4.3 shows the PAD of the elites in different generations. The magnitude of the plotted vectors is just for visualization purposes and does not quantify the computed gradients. Directions computed by the method of [106, 78] and the PCA-Assisted Hybrid Algorithm are compared. Both of these methods provide very similar descent directions pointing towards the minimization of both objectives. Both of them provide satisfactory results, if individuals are updated along these directions. For the method of [106, 78], the arrows are perpendicular to the SPEA2 utility function isoline defined by the front and they are slightly curved according to the front. For the PCA-Assisted Hybrid Algorithm, the arrows are parallel to the direction of the smallest variance produced by the PCA and have all the same angle and magnitude. Note that the remaining direction with the largest variance is almost parallel to the front. The PCA-Assisted Hybrid Algorithm takes the difference in scale between the given objective function values into account and this is why the PAD is more inclined towards the second objective than the one produced by the method of [106, 78]. Both objectives are expected to be improved equal for each individual updated towards this direction.

4.3 Benchmark Cases Revisited

The PCA-Assisted Hybrid Algorithm is demonstrated in the benchmark cases (see section 2.4). In all cases, the gradients of the objective function with respect to the design variables are computed with a continuous adjoint solver, which is embedded to the PUMA software used to solve the flow equations. During the optimization with the PCA-Assisted Hybrid Algorithm, the weights which concatenate the gradients are known before the adjoint runs. They are included in the

Algorithm 1 The PCA-Assisted Hybrid Algorithm

$g \leftarrow 0$; {g: generation counter}
$n_{evals} \leftarrow 0$; {n_{evals}: evaluation counter}
$S_\lambda^g \leftarrow Initialization()$
while $n_{evals} \leq n_{evals,max}$ **do**
$PCA\ Training(\vec{f}(\vec{b}_1^g), \cdots, \vec{f}(\vec{b}_\lambda^g))$
if $n_{evals} < T^{MM}$ **then**
$\vec{f}(\vec{b}_1^g), \cdots, \vec{f}(\vec{b}_\lambda^g) \leftarrow Evaluate()$; {on the PSM}
$n_{evals} \leftarrow n_{evals} + \lambda$
else
$\hat{\vec{f}} \leftarrow Low{-}Cost\ Pre{-}Evaluation(S_\lambda^g)$
$\lambda_e \leftarrow Sorting(\hat{\vec{f}})$
for $i \in [1,\lambda_e]$ **do**
$\vec{f}(\vec{b}_i^g) \leftarrow Evaluate()$
if $i \leq \lambda_{GB}$ **then**
$\nabla f(\vec{b}_i) \leftarrow ComputeGradients()$; {using PCA-transformed adjoint }
end if
end for
$n_{evals} \leftarrow n_{evals} + \lambda_e + \lambda_{GB}$
end if
$S_\epsilon^g \leftarrow Dominance(S_\epsilon^g, \{\vec{f}(\vec{b}_1^g), \cdots, \vec{f}(\vec{b}_{\lambda_e}^g)\})$; {update elite set}
$S_\mu^g \leftarrow ParentSelection(S_\epsilon^g, S_\lambda^g)$
for $i \in [1,\lambda]$ **do**
if $n_{evals} \geq T^{MM}$ & $i \leq \lambda_{GB}$ **then**
$\vec{b}_i^{g+1} \leftarrow \vec{b}_i^g + \eta \nabla f(\vec{b}_i^g)$; {GB refinement}
else
$\vec{b}_i^{g+1} \leftarrow Evolution\ Operators(S_\mu^g, S_\epsilon^g)$
end if
end for
$S_\lambda^{g+1} \leftarrow \{\vec{b}_1^{g+1}, \cdots, \vec{b}_\lambda^{g+1}\}$
$g \leftarrow g + 1$
end while
$Export\ Front\ of\ Non{-}Dominated\ Solutions\ S_\epsilon^g$

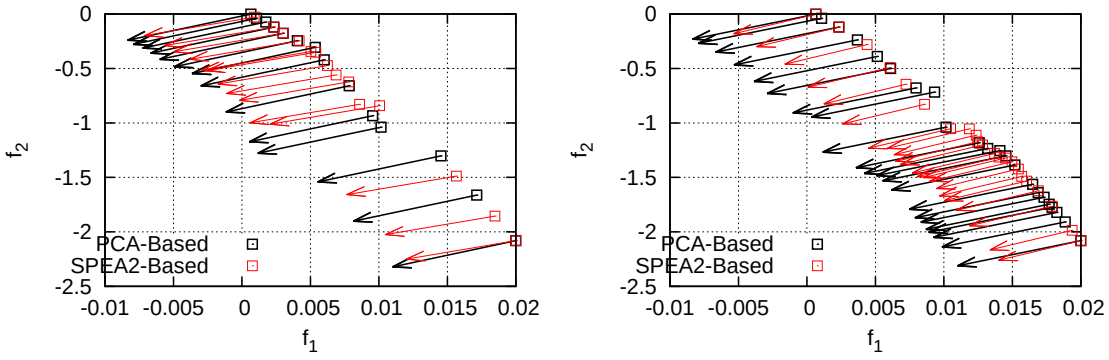


Figure 4.3: Two-objective mathematical minimization problem, eq. 4.2.5 : Elite sets at two generations (left: 6^{th}, right: 20^{th}). Directions computed by the method of [106, 78] (red vectors) and the present one (black).

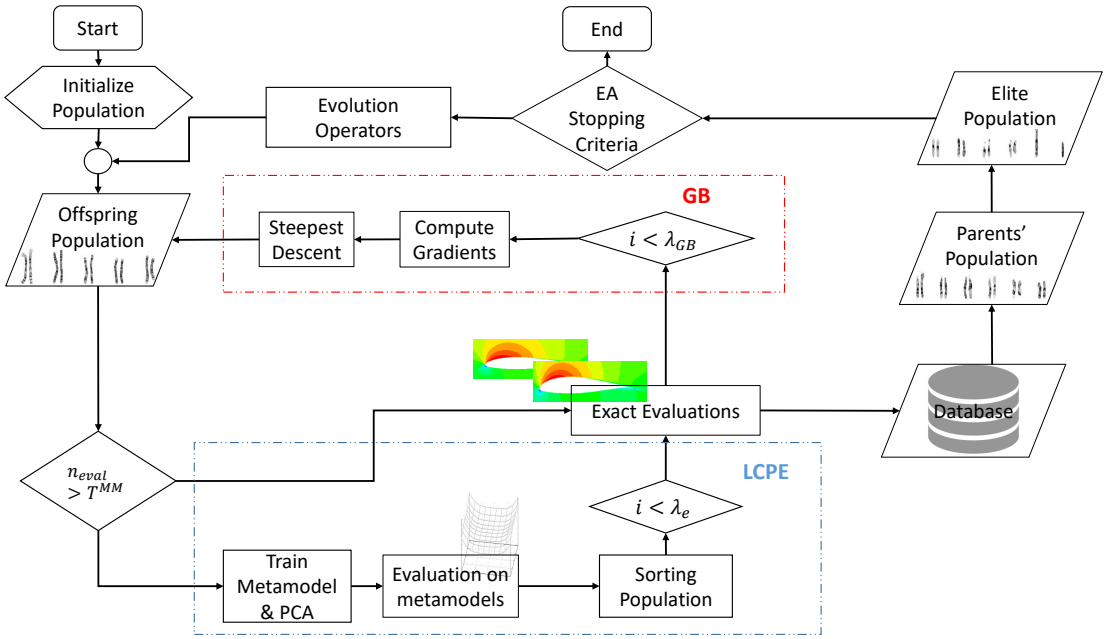


Figure 4.4: The PCA-Assisted Hybrid Algorithm workflow.

adjoint software, a single call of which computes all the gradients needed by the GB method to update the individuals to be refined. The performance of the PCA-Assisted Hybrid Algorithm is compared with the M(K)AEA(K) described in section 5.2, which was shown to outperform all the other EA variants presented in other sections.

4.3.1 Benchmark Case 1 as a Three-Objective Problem

The first benchmark problem, the shape optimization of an airfoil, is modified to become a MOO, so as to use the PCA-Assisted Hybrid Algorithm. Two more objective functions are introduced and the case is optimized for max. lift (L), min. drag (D) and min. moment (M). Flow conditions, CFD grid and design variables remain the same.

The optimization is carried out using an M(K)AEA(K) and the PCA-Assisted Hybrid Algorithm. In both variants, $\mu = 10$, $\lambda = 20$ and $\varepsilon = 20$ and the LCPE phase begins after the first $T^{MM} = 20$ evaluations on the PSM. In the PCA-Assisted Hybrid Algorithm, $\lambda_{GB} = \lambda_e = 2$ members undergo refinement based on the GB method. The computational budget is restricted to 500 calls to the PSM which include calls to the flow and the adjoint solver (considered as having the same cost). Fig. 4.5 compares the convergence histories of the hypervolume indicator of all the optimization runs. The PCA-Assisted Hybrid Algorithm outperforms the M(K)AEA(K) and this shows that the local advancement can accelerate an already well performing EA. During the optimization runs of the PCA-Assisted Hybrid Algorithm, 275 evaluations are spent for the PSM calls and the rest 225 evaluation for the adjoint calls. Fig. 4.6 compares the reference airfoil shape with the optimal solutions for min. drag, max. lift and min. moment resulted from a single run (one RNG seed) of the PCA-Assisted Hybrid Algorithm. For these four airfoils, a comparison of the Mach number field can be seen in fig. 4.7. For the min. drag airfoil, the suction side becomes practically “flat" and the shock wave is reduced leading to drag reduction. For the max. lift airfoil, the airfoil is more cambered, the pressure is even lower on the suction side and increased along the pressure side so as to produce more lift. For the min. moment airfoil, the shape is modified so as to make the shock move upstream (fig. 4.7c) and, thus, minimize the effect of the shock induced compression on the pitching moment.

4.3.2 Benchmark Case 2

The Benchmark Case 2, the shape optimization of an isolated wing, is revisited with the PCA-Assisted Hybrid Algorithm, with settings $(\mu, \lambda) = (10, 20)$, $\varepsilon = 20$ and $\lambda_{GB} = 2$. 263 evaluations are spent for CFD evaluations whereas the rest 237 for the adjoint solver. Comparison of the convergence histories based on the hypervolume indicator is shown in fig. 4.8 and the corresponding fronts of non-dominated solutions in fig. 4.9. The PCA-Assisted Hybrid Algorithm outperforms the M(K)AEA(K) during almost the entire evolution and the final front is slightly better in all regions of the objective space. With the PCA-Assisted Hybrid Algorithm, the GB refinement delivered 7 out of the 20 elite individuals of fig. 4.9. Fig. 4.10 shows how the GB method upgrades the front of non-dominated solutions, from generation to generation.

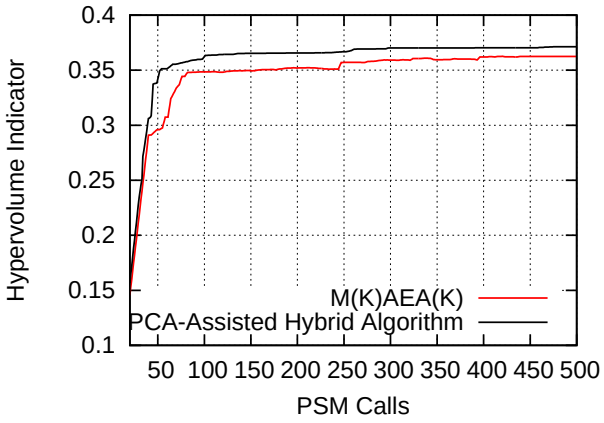


Figure 4.5: Benchmark Case 1 with three objectives: Comparison of the convergence histories of the M(K)AEA(K) and the PCA-Assisted Hybrid Algorithm in terms of the number of PSM calls. Adjoint runs are included and counted as an extra CFD evaluation/ PSM calls.

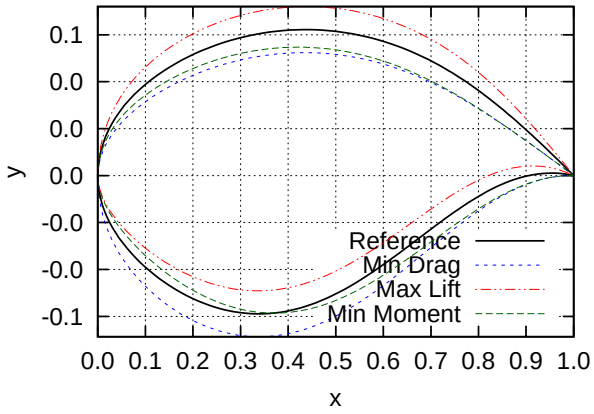
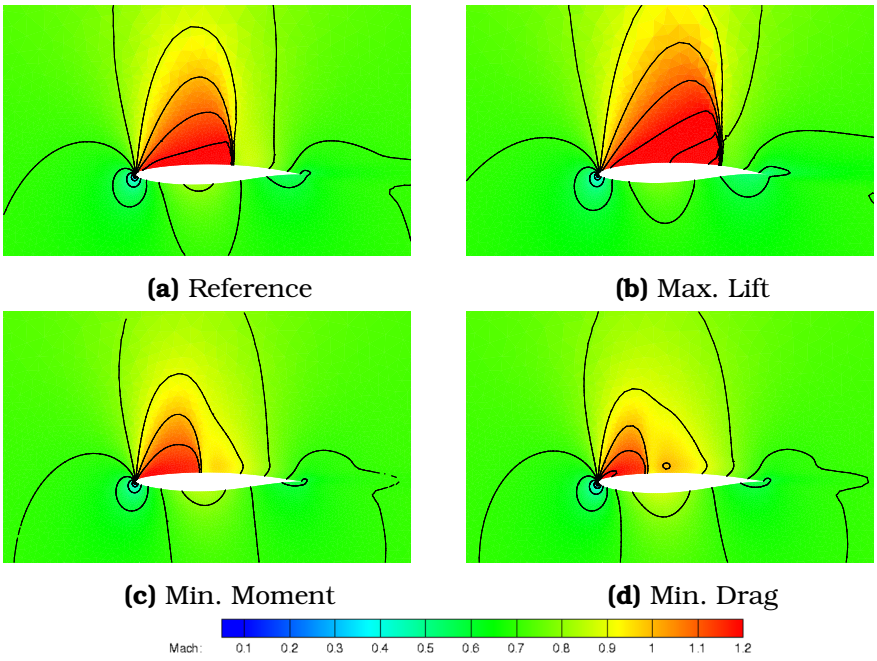


Figure 4.6: Benchmark Case 1 with three objectives: Comparison of the reference and optimized airfoil shapes corresponding to the three extreme points of the Pareto front. Axes not in scale.

4.3.3 Benchmark Case 3

The Benchmark Case 3 problem is revisited using the method of [106, 78] and the PCA-Assisted Hybrid Algorithm with $\lambda_e = \lambda_{GB} = 1$. This is performed only once, in this case, in order to make an interesting comparison between the two (old/new) Hybrid Algorithms. Fig. 4.11 shows that the PCA-Assisted Hybrid Algorithm performs better than both the M(K)AEA(K) and the SPEA2-Based Hybrid Algorithm of [106, 78]. The superiority of the proposed method over the one of [106, 78] is due to the way PAD is computed, since the two algorithms practically share all other features. Fig. 4.12 shows the improvement of the front from generation to



(a) Reference **(b)** Max. Lift

(c) Min. Moment **(d)** Min. Drag

Figure 4.7: Benchmark Case 1 with three objectives: Mach number iso-areas around the reference, the max. lift, the min. moment and the min. drag airfoils.

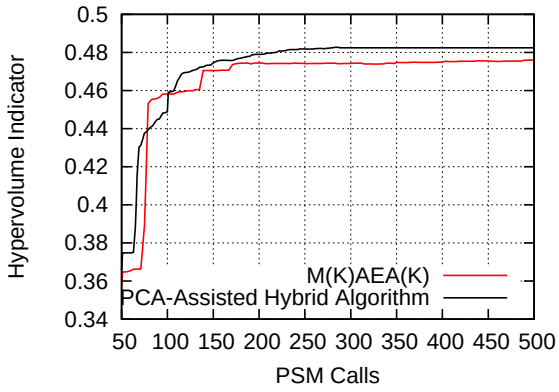


Figure 4.8: Benchmark Case 2: Comparison of the convergence histories of the M(K)AEA(K) and the PCA-Assisted Hybrid Algorithm in terms of the number of the CFD evaluations.

generation, each time a GB-refinement takes place. Fig. 4.13 presents the front of non-dominated solutions produced by the optimization runs using the three methods (for the same RNG seed). During the optimization with the PCA-Assisted Hybrid Algorithm, 263 out of the 500 evaluations are spent to evaluate individuals, i.e. to run the PSM, whereas the rest 237 to compute the gradients (adjoint runs).

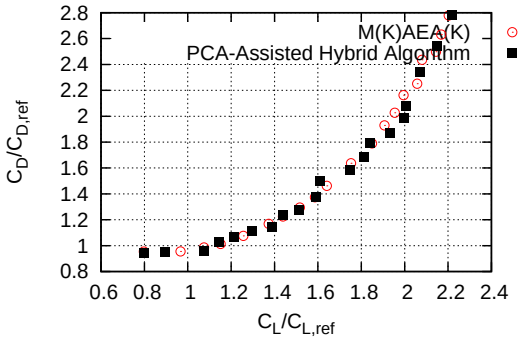


Figure 4.9: Benchmark Case 2: Comparison of the fronts of non-dominated solutions resulted from the M(K)AEA(K) and the PCA-Assisted Hybrid Algorithm.

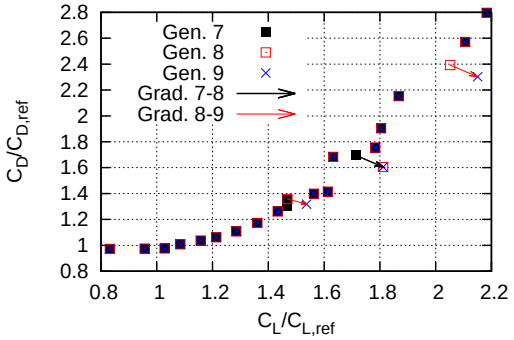


Figure 4.10: Benchmark Case 2: Fronts of non-dominated solutions at three successive generations computed by the PCA-Assisted Hybrid Algorithm. Vectors represent the improvements made by the GB method.

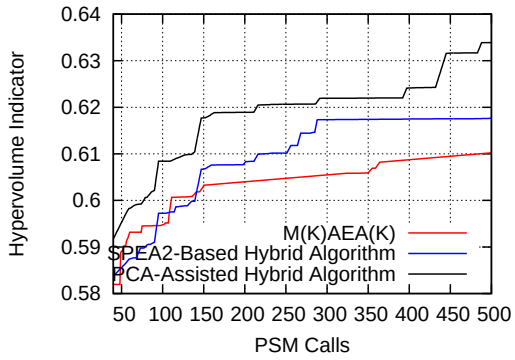


Figure 4.11: Benchmark Case 3: Comparison of the convergence histories of M(K)AEA(K), PCA-Assisted Hybrid Algorithm and the method of [106, 78] (SPEA2-Based Hybrid Algorithm) in terms of the number of CFD evaluations (including flow and adjoint runs, both considered at the same cost).

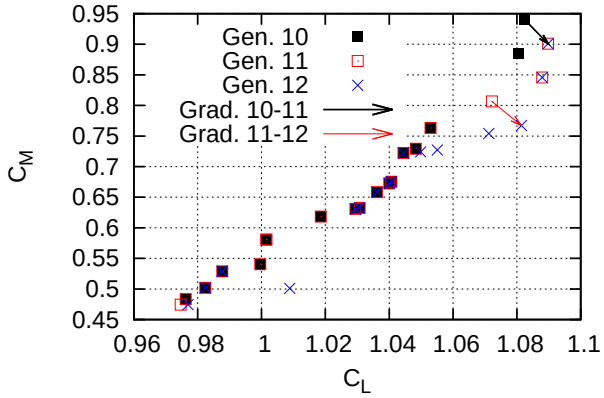


Figure 4.12: Benchmark Case 3: Fronts of non-dominated solutions for three subsequent generations ($Gen. = 10$, 11 and 12) computed by the PCA-Assisted Hybrid Algorithm. Vectors represent improvements made by the GB method.

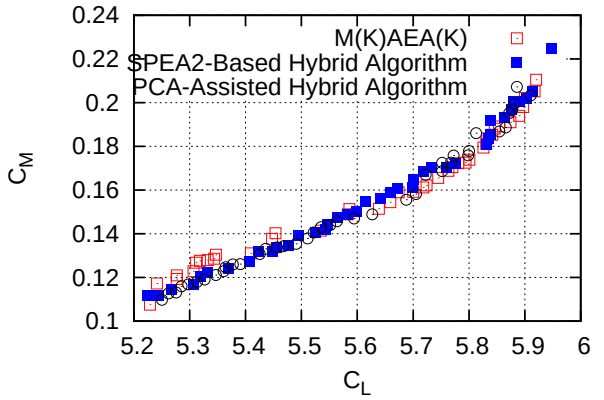


Figure 4.13: Benchmark Case 3: Comparison of the fronts of non-dominated solutions resulted from M(K)AEA(K), the PCA-Assisted Hybrid Algorithm and the method of [106, 78].

Chapter 5

Multi-Criteria Decision Making within EAs

A Multi-Objective Optimization (MOO) problem consists of more than one conflicting objective functions to be minimized, $min.\vec{f}(\vec{b}) = min.f_1(\vec{b}), \ldots, min.\ f_{M_o}(\vec{b})$. As already explained, the optimization ends up with a front of non-dominated solutions see chapter 2. Decision Maker (DM) usually needs to isolate one solution from this front, according to not articulated 'a priori' additional preferences. They may have more interest in some objectives over the others and MCDM techniques can help the DM eliminate solutions which are not of interest to them and select one according to their preferences.

For the MCDM techniques to be applied, the DM preferences must be quantified, which can be done in various ways. During this thesis, weights express and quantify the DM preferences with respect to each objective function. In general, MCDM techniques can be used in three different ways to enhance the EAs, [181]. The first way is 'a posteriori', in which the DM preferences are not known during the EA-based search. The EA (or any of its variants) is performed without any change, the final front of non-dominated solutions is produced and, then, processed by the MCDM technique which selects the final 'optimal' solution(s) based on the DM preferences. In contrast, when preferences are known 'a priori', the EAs take advantage of these preferences and, to this end, they undergo appropriate modifications. During the evolution, the MCDM technique replaces the method computing the scalar utility function ϕ for each individual in the EA populations. Consequently, the value of ϕ depends on the DM preferences (weights for each objective function). The more preferred individuals are assigned smaller ϕ, thus, get higher chances to create offspring during the evolution. The scalar utility function ϕ and, through it the MCDM technique, affect the parent selection and trimming processes, evolving the front of non-dominated solutions towards areas preferred by the DM. The final front is different than the one that would result from an EA without having access to the DM preferences. Lastly, the MCDM techniques can be used 'interactively' with the EAs, [180]. In this case, DM preferences may change during the evolution. The EA is affected as in the 'a

priori' usage of MCDM. The algorithm stops periodically and presents the front of non-dominated solutions computed thus far to the DM. The latter, after judging the front, may change the preferences given to the EA. Thus, the EA interactively changes the focus of the evolution converging to fronts much closer to the updated DM preferences.

5.1 MCDM Techniques

Many different MCDM techniques [72, 168] have been devised and applied in different kind of problems. For a better understanding, it is important to start with the most important relevant definitions.

A decision Matrix **P** is formed by elements which quantify the performance of the i solution of the MOO problem over the j objective function. **P** is, thus, a $M_e \times M_o$ matrix, where M_e is the number of processed solutions (herein, the elite set) and M_o the number of objective functions.

For the MCDM techniques presented below, the decision matrix should be normalized. The normalized $M_e \times M_o$ matrix **r** is computed as $r_{i,j} = \frac{p_{i,j}}{\sqrt{\sum_{k=1}^{M_e} p_{k,j}^2}}$ or $r_{i,j} = \frac{p_{i,j}}{min_{k=1}^{M_e} p_{k,j}^2}$. The DM preferences are introduced into matrix **r** by multiplying each element with the weight corresponding to its objective function; the resulting matrix **V** has elements computed as $v_{i,j} = r_{i,j} w_j$ where w_j is the j^{th} objective function's weight. Without loss of generality, the sum of weights is equal to $\sum_{j=1}^{M_o} w_j = 1$.

The MCDM techniques rank the given solutions, from best to worst, based on the weights/DM preferences. This rank essentially affects the selection of the best solutions of a front of non-dominated solution ('a posteriori'). The population members of each generation can be ranked based on the MCDM technique to continuously promote solutions preferred by the DM. Some MCDM techniques are presented epigrammatically:

1) TOPSIS (Technique for the Order of Preference by Similarity to the Ideal Solution, [72]): This technique is the one programmed and used in this thesis; it is thus described below in detail.

2) VIKOR (VIseKriterijuska Optimizacija I Komoromisno Resenje, [133]): This technique ranks the solutions based on their distance from the ideal solution, which is formed by combining the minimum value of all objective functions as in the **V** matrix. Three different metrics based on the distance are computed and solutions with smallest metric values are prioritized.

3) WS or WP (Weighted Sum or Weighted Product, [33]): The summation or product of the **V** values for each solution is computed correspondingly for the Weighted Sum or Product. Better solutions correspond to smaller overall values.

4) AHP (Analytic Hierarchy Process, [146]): The DM preference for each objective function is expressed with a new scale corresponding to the priority one objective has over the others. The objectives weights are computed based on the new scale and multiplied with the **r** elements to compute matrix **V**. For each solution, the **V** elements are added and the solutions are ranked based on the resulted values.

5) PROMETHEE (Preference Ranking Organization METHod for Enrichment Evaluations, [111]): Rather than pointing out a "right" decision, this method helps the DM find the alternative that best suits DM's goal and understanding of the problem. At first, pairwise comparisons will be made between all the solutions for each objective. The notion of preference function is introduced to translate the difference into a uni-criterion preference degree for each couple. A multi-criteria preference degree is then computed, based on the DM weights, to globally compare every couple of solutions. In order to position every solution with respect to all other, two scores are computed: (a) the positive preference flow which quantifies how a given solution is globally preferred with respect to all the other solutions and (b) the negative preference flow quantifies how a given solution is being globally preferred by all the other solutions. The difference between these scores determines the solutions' ranking, where smaller values correspond to better solutions.

This thesis focuses only on the TOPSIS technique, but any other method can be applied instead, with just some minor modifications to the algorithm.

5.1.1 The TOPSIS Technique

TOPSIS was developed by Hwang and Youn [72] for MOO problems. It ranks the presented solutions based on the DM preferences, which must be expressed in the form of weights associated with each objective function. Two reference/ideal points are used, the zenith corresponding to the point with all objectives at their minimum values and the nadir corresponding to the point with all objectives at their maximum values. This technique can be used without any change for both 'a priori' and 'a posteriori' articulated DM preferences. For the former, TOPSIS ranks the EA populations in each generation while, for the latter, it ranks only the final front of non-dominated solutions presented to the DM. Either way, a data-set is presented to TOPSIS, which ranks its members.

TOPSIS steps are:

Step 1: Construction of the decision matrix **P**.

Step 2: Construction of the normalized decision matrix **r**.

Step 3: Construction of matrix **V** by processing matrix **r** with the weights articulated by the DM.

Step 4: Computation of two ideal solutions, the positive (I^+) and the negative (I^-) one. I^+ corresponds to the minimum objective functions found in the data-set, whereas I^- corresponds to the maximum objectives, fig. 5.1. Note that, for the 'a priori' use of TOPSIS within an EA, the data-set and, consequently, the ideal solutions are changed in each generation,

$$I^+ = (max_i v_{i1}, max_i v_{i2}, \ldots, max_i v_{iM_o}) = (v_1^+, v_2^+, \ldots, v_{M_o}^+) \quad (5.1.1.1)$$

$$I^- = (min_i v_{i1}, min_i v_{i2}, \ldots, min_i v_{iM_o}) = (v_1^-, v_2^-, \ldots, v_{M_o}^-) \quad (5.1.1.2)$$

Step 5: Computation of the Euclidean in the objective space distances of each solution from the positive and negative ideal solutions, fig. 5.2,

$$d_i^+ = \sqrt{\sum_{j=1}^{M_o} (v_{ij} - v_j^+)^2} \quad (5.1.1.3)$$

$$d_i^- = \sqrt{\sum_{j=1}^{M_o} (v_{ij} - v_j^-)^2} \quad (5.1.1.4)$$

Step 6: Computation of the relative distance of each solution from the ideal solutions

$$D_i = \frac{d_i^-}{d_i^- + d_i^+} \quad (5.1.1.5)$$

D_i is the scalar utility value ϕ assigned in the i^{th} solution and used for the application of the evolution operators.

5.1.2 TOPSIS-driven EAs

For the 'a-priori' articulated DM preferences, TOPSIS is introduced into EAs for the solution of MOO problems. Its purpose is to drive the optimization towards areas of the objective space preferred by the DM. Given that the M(K)AEA(K) variant outperforms any aforementioned EA 's variant, as shown in previous sections, TOPSIS is incorporated into this variant to further improve its performance, producing a new variant called TOPSIS-driven M(K)AEA(K) or TO-M(K)AEA(K). Needless to say that TOPSIS can be used with any other variant of EA or MAEA. It

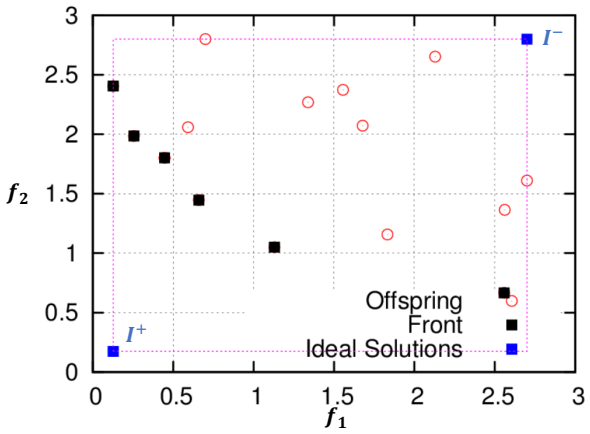


Figure 5.1: Front of non-dominated solutions along with the two ideal solutions defined in TOPSIS.

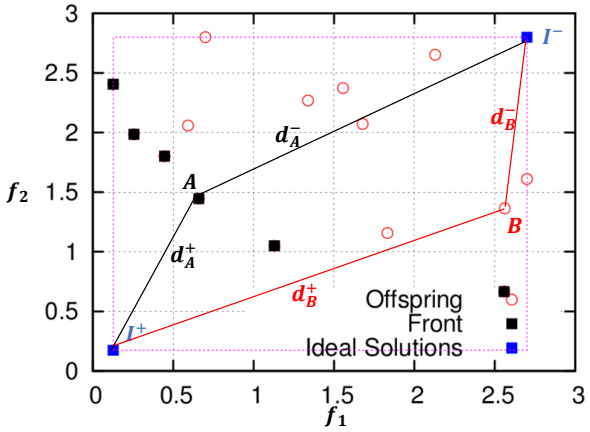


Figure 5.2: Euclidean distances between candidate solutions and the ideal solutions in TOPSIS.

replaces the scalar utility function, assigns to each individuals values ϕ based on the DM preferences and affects the parents and elites selection. The resulting algorithm gradually moves the front of non-dominated solutions towards the preferred objective space areas. Finally, it generates better fronts in these areas in comparison with the ones generated by the M(K)AEA(K) or any other EA variant. TO-M(K)AEA(K)'s prerequisite is the DM preferences to be available in the form of weights for each objective function before the optimization.

TO-M(K)AEA(K) includes the following steps (excluding steps for the PCA procedures), which are also shown, in flowchart form, in fig. 5.3:

Initialization: First generation ($g = 0$). The DM preferences are known in the form of weights for each objective function. The offspring population P_λ^0 is initialized using a random number generator (RNG).

Evaluation: All offspring (P_λ^g) are evaluated on either the PSM or the metamodels (in case of a MAEA).

Fitness Assignment: TOPSIS assigns a scalar utility value ϕ to each individual $\vec{b} \in P_\lambda^g \cup P_\mu^g \cup P_e^g$. This values are based on the DM preferences and distances in the objective space as computed by TOPSIS.

Elite Selection: The elite set of the current generation (P_e^g) is formed based on the aforementioned ϕ values computed by the TOPSIS algorithm.

Elitism Operator: A small number of elite individuals replace the offspring members of P_λ^g.

Parent Selection: The parent population P_μ^g is formed by processing the $P_\lambda^g \cup P_\mu^{g-1}$ individuals. The selection is based on their fitness value ϕ computed by TOPSIS.

Crossover and Mutation Operators: The new offspring population P_λ^{g+1} is created by applying the crossover and mutation operators (see section 2.3.2).

Stopping Criteria: If the stopping criteria are met, terminate the algorithm; otherwise, return to the **Evaluation** step.

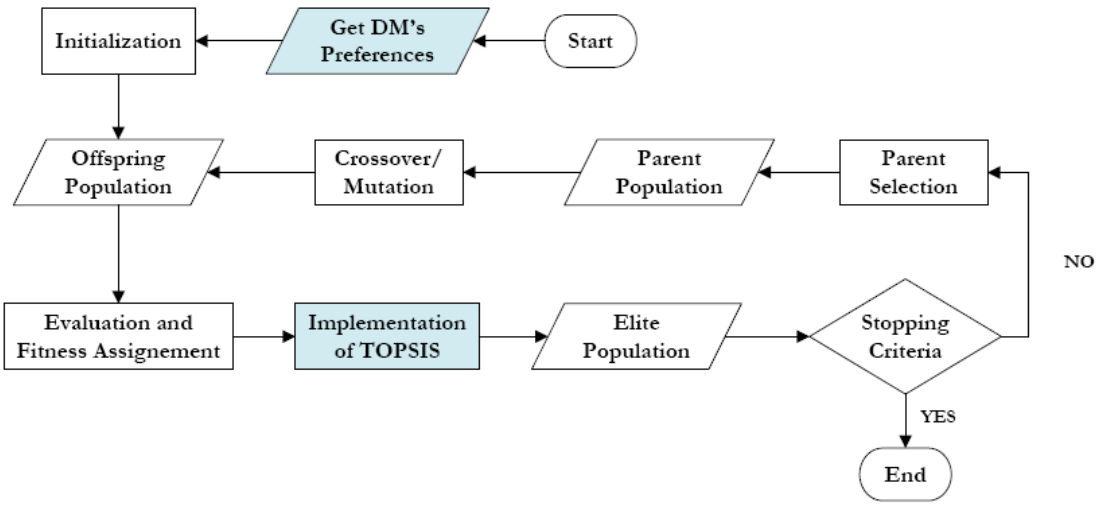


Figure 5.3: TO-M(K)AEA(K) flowchart.

5.2 Applications

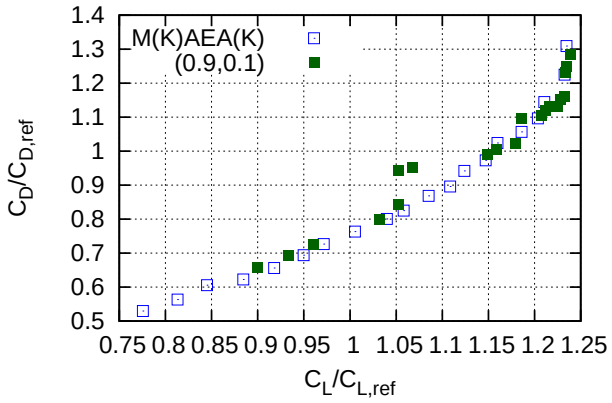
The previously presented TO-M(K)AEA(K) is demonstrated in four benchmark cases. These optimization problems have already been presented in section 2.4. The new EA variant performance is compared with the one of M(K)AEA(K), which has been shown to outperform all the other variants used in section 5.2.

5.2.1 Benchmark Case 1 with two objectives

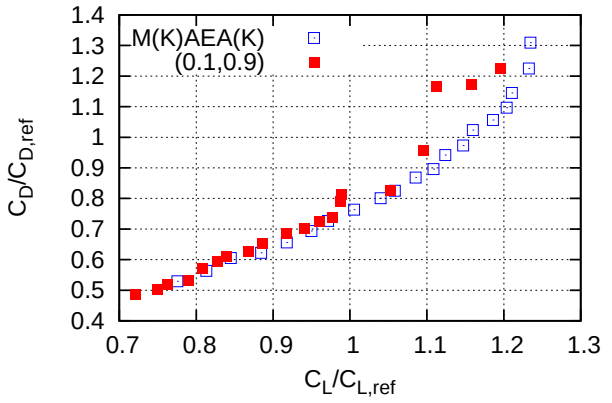
Herein, the shape optimization of an isolated airfoil, as in section 2.4, is modified to perform a MOO problem with two objective functions: max. lift coefficient (C_L) and min. drag coefficient (C_D). All the other optimization parameters are kept the same. Three case studies are performed with the DM giving three different sets of weights which corresponds to his/her preferences for each objective function. The first set is $(w_1, w_2) = (0.9, 0.1)$ which gives emphasis on the first objective function, while the second one $(w_1, w_2) = (0.1, 0.9)$ giving empashis on the second objective function. For the third set $(w_1, w_2) = (0.5, 0.5)$, both objectives appear to be equally important. In this case, the convergence speed should not be measured with the hypervolume indicator, since the latter does not include the DM preferences (unless defined in a different manner). On the other hand, the comparison of the final fronts of non-dominated solution for the TO-M(K)AEA(K) with different weights and the M(K)AEA(K) runs, which is presented in fig. 5.4, shows that the optimization is driven towards preferred areas of the objective space. For the first set of weights, the final front is gathered in the area of high C_L values, while, for the second set, in the area of low C_D values. Weights set to $(0.5, 0.5)$ drive the front towards the middle of the Pareto front.

5.2.2 Benchmark Case 2

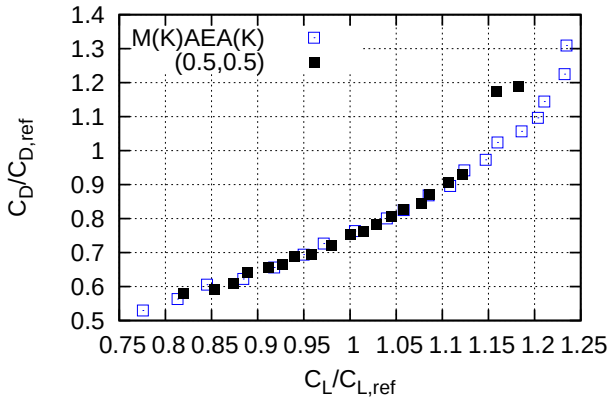
The shape optimization of an isolated wing, [154], for max. lift coefficient (C_L) and min. drag coefficient (C_D) is examined anew using a $(10, 20)$ TO-M(K)AEA(K). The usage of KPCA and metamodels is made according to the algorithm presented in section 5.2. The final front is compared in fig. 5.5 with the one produced by the M(K)AEA(K) run. Once again, the DM preferences are set to: (a) $(w_1, w_2) = (0.9, 0.1)$ with emphasis on the max. lift area of the objective space, (b) $(w_1, w_2) = (0.1, 0.9)$ with emphasis on the min. drag area, (c) $(w_1, w_2) = (0.5, 0.5)$ with emphasis on the middle of the Pareto front. The corresponding fronts are shown in fig. 5.5. In all cases, the TO-M(K)AEA(K)'s fronts outperform the M(K)AEA(K)one in the corresponding areas.



(a) $(w_1, w_2) = (0.9, 0.1)$

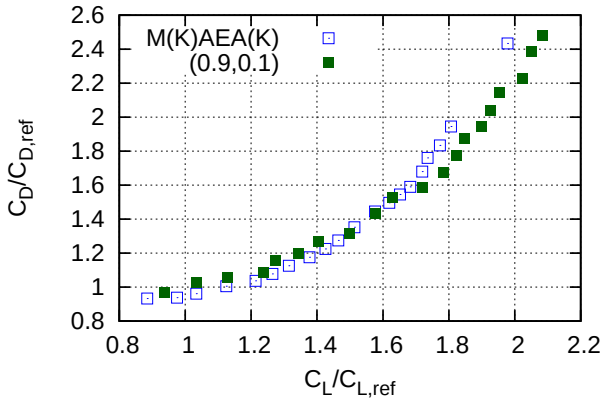


(b) $(w_1, w_2) = (0.1, 0.9)$

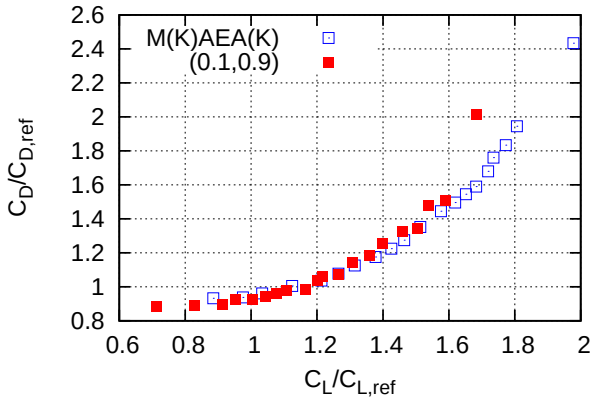


(c) $(w_1, w_2) = (0.5, 0.5)$

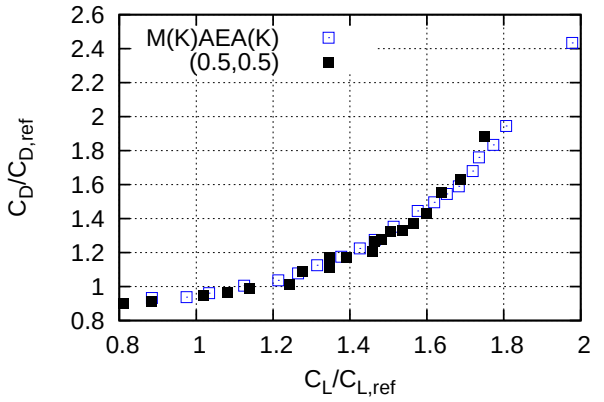
Figure 5.4: Benchmark Case 1: Front of non-dominated solutions computed by TO-M(K)AEA(K) and M(K)AEA(K).



(a) $(w_1, w_2) = (0.9, 0.1)$



(b) $(w_1, w_2) = (0.1, 0.9)$



(c) $(w_1, w_2) = (0.5, 0.5)$

Figure 5.5: Benchmark Case 2: Front of non-dominated solutions computed by TO-M(K)AEA(K) and M(K)AEA(K).

Chapter 6

Industrial Optimization Problems

This section presents a number of industrial optimization problems, taken from real-world applications. Once again, the GPU-enabled PUMA solver developed by the PCOpt/NTUA Unit acts as the PSM in all but one case (the optimization of a valveless diaphragm micropump) in which the cut-cell in-house CFD solver is used instead. Each CFD evaluation requires much more computational time than the Benchmark Cases, that is why only a few of the EA variants (mainly the best of them according to the findings thus far) are utilized for their optimization. The aircraft and Francis runner optimizations are both carried out in the context of research funded by European industries. The shape optimization cases of the ultra-light aircraft and the DrivAer car were provided by the partners of a European project in which the PCOPT/NTUA Unit participated, too. Lastly, a 3D valveless diaphragm micropump, for use in medical and biochemical processes, is designed. Its diaphragm motion, which induces the fluids flow inside the pump, is parameterized and the incompressible cut-cell CFD solver simulates the unsteady flow. This micropump is optimized so as to minimize the back-flow at the exit and maximize the net volume flux [84, 83]. This research was funded by the project "Design-Optimization of Diaphragm Pumps under Operational and Manufacturing Uncertainties based on the Cut-Cell and the Polynomial Chaos Methods" (MIS 5004541), implemented under the Action "Supporting Researchers with an Emphasis on Young Researchers", in the context of the call EDBM34, funded by the Operational Programme "Human Resource Development, Education and Lifelong Learning" (NSRF 2014-2020) and co-financed by Greece and the European Union (European Regional Development Fund).

6.1 Industrial Case 1: Shape Optimization of an Aircraft Wing-Body Configuration

The first industrial problem is concerned with the re-design of an aircraft wing-body configuration, [15], by only changing the shape of the wing, for max. lift coefficient (C_L) and min. drag coefficient (C_D). The flow conditions are: Reynolds number $Re_c = 10^6$ based on the mean aerodynamic chord, freestream Mach number $M_\infty = 0.75$ and flow angles at the farfield are 0^o. The flow is turbulent, so the Navier-Stokes equations with the Spalart-Allmaras [164] turbulence model are solved. Only half of the aircraft is simulated due to symmetry. A customized

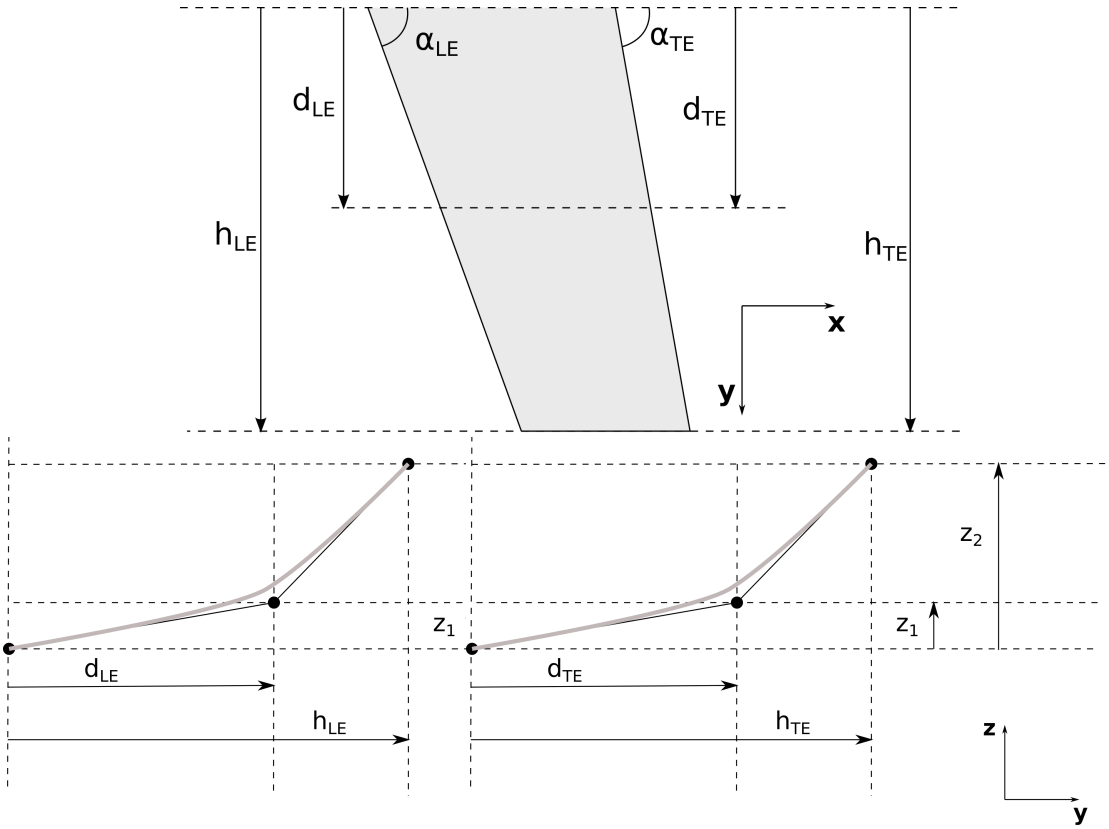


Figure 6.1: Industrial Problem 1: Definition of the 8 design variables ($\alpha_{LE}, \alpha_{TE}, d_{LE}, d_{TE}, h_{LE}, h_{TE}, z_1, z_2$). Top: wing planform. Middle: wing dihedral angle at the leading edge. Bottom: wing dihedral angle at trailing edges.

parameterization model with 8 design variables is used to control the wing shape (fig. 6.1). In specific, the first two of them (α_{LE}, α_{TE}) control the sweep angle. Distances d_{LE} and d_{TE} affect the leading and trailing edge curvature close to mid-span. The wing span is affected by h_{LE} and h_{TE} which change the wing tip, whereas the variables z_1 and z_2 can lift and/or twist the wing. The aforementioned design variables affect, in turn, the coordinates of some of the internal nodes of

the $3\times5\times4$ NURBS control grid (fig. 6.2). Out of a total of 60 control points, some are kept fixed to ensure continuity with the surrounding undeformed CFD grid while others are adapted to the values of the aforementioned design variables. The morphing box (fig. 6.2) leaves the wing-pylon and pylon-nacelle junctions intact and acts complementarily to the wing parameterization of fig. 6.1, affecting exclusively the grid (displacement) in the interior of the flow domain. The CFD grid consists of tetrahedra and prisms, with ~0.77M nodes. Each PSM call costs ~16min on an NVIDIA K20 GPU, including the morphing process.

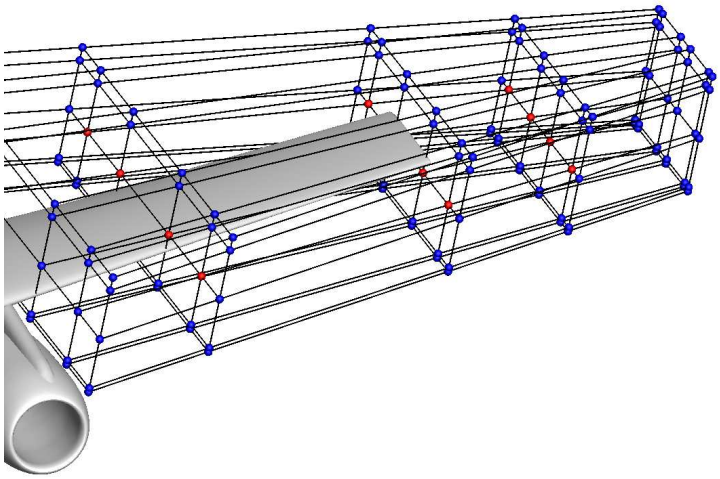


Figure 6.2: Industrial Problem 1: Control points of the volumetric NURBS grid, used to deform the wing and the CFD grid around it (not shown in this figure). Blue points are kept fixed so as to ensure CFD grid continuity. Red ones are allowed to vary. The volumetric NURBS grid is used for displacing the internal CFD nodes (the wing shpae is affected by the parameterization of fig. 6.1).

The shape optimization problem is solved using a $(5,10)$EA, MAEA, MAEA(K) and M(K)AEA(K). The LCPE phase starts after $T^{MM}=20$ calls to the PSM and the $\lambda_e=4$ most "promising" individuals are re-evaluated in each generation. The stopping criterion was set to 200 CFD evaluations. Fig. 6.3 presents the mean convergence of the hypervolume indicator obtained by averaging the results of three runs (with three different RNG seeds) of the EA and MAEA. During the first generations, the metamodels seem to perform poorly. By enriching the DB with new individuals evaluated on the PSM, the on-line trained metamodels improve

their performance, resulting to better solutions for the optimization and faster convergence than the standard EA. Fig. 6.4 compares the convergence of the KPCA variants with the MAEA and fig. 6.5 the corresponding fronts of non-dominated solutions from a single RNG seed. While the MAEA(K) converge faster than MAEA, the final fronts are more or less the same. In contrast, the M(K)AEA(K) computed an improved front, due to the wide spreading of the optimal solutions.

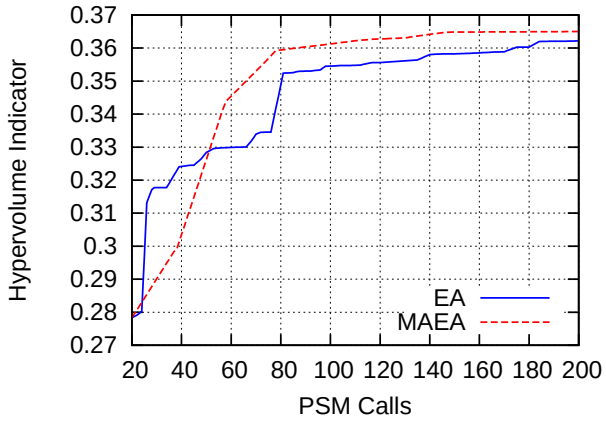


Figure 6.3: Industrial Problem 1: Comparison of the averaged convergence histories for three RNG seeds of EA and MAEA, in terms of the number of CFD evaluations.

A comparison of the pressure coefficient distributions at three span-wise positions and the pressure field around the whole aircraft over the reference shape and the ones optimized for max. C_L and min. C_D, obtained from the best run of the M(K)AEA(K), are shown in fig. 6.6 and fig. 6.7, respectively. Figs. 6.8 and 6.9 highlight differences among the aforementioned wings. The corresponding airfoil geometries at the same span-wise positions are also shown in fig. 6.10. In the optimal solution for max. C_L, due to the change in the shape, the angle of attack has increased. For the min. C_D design, the pressure side close to the trailing edge is straightened, minimizing the effect of high pressure on the drag. Also, the wing surface has decreased, reducing the effect of viscous stresses on drag.

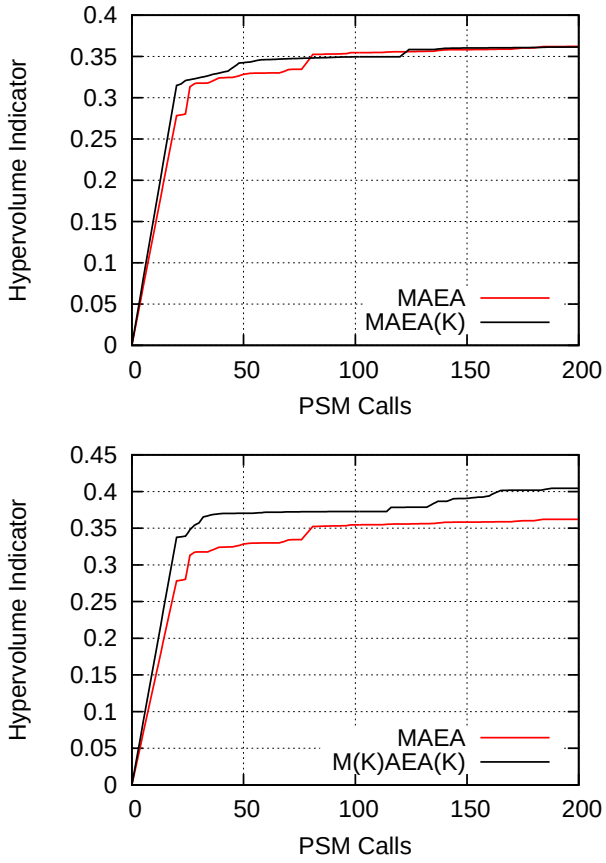


Figure 6.4: Industrial Problem 1: Comparison of the convergence histories of MAEA and MAEA(K) (top) and MAEA and M(K)AEA(K) (bottom), in terms of the number of CFD evaluations.

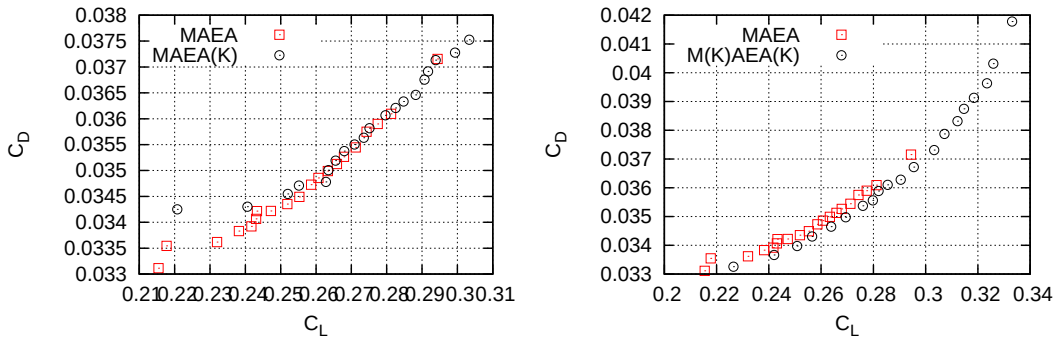
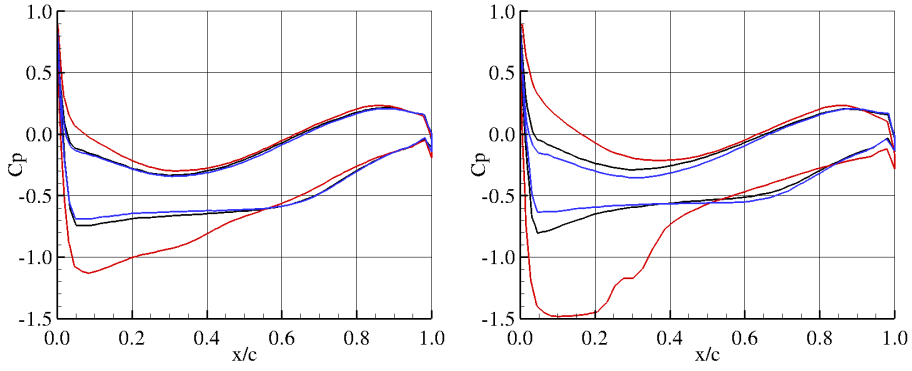
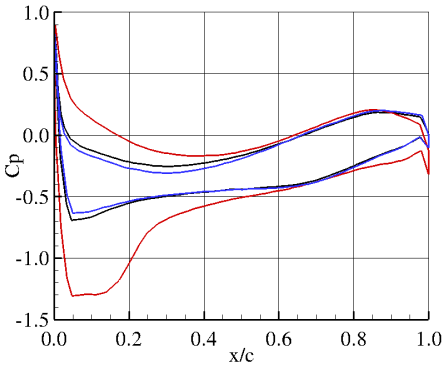


Figure 6.5: Industrial Problem 1: Comparison of the fronts of non-dominated solutions resulted from MAEA and MAEA(K) (left) and MAEA and M(K)AEA(K) (right).



(a) 50% span **(b)** 68% span

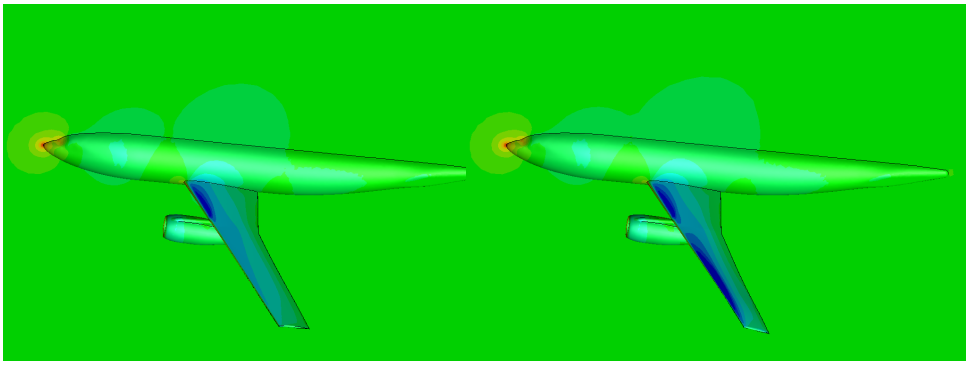


(c) 85% span

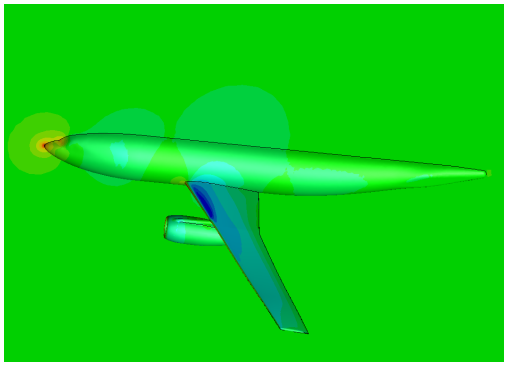
Figure 6.6: Industrial Problem 1: Comparison of the pressure coefficient (C_p) distributions at different spanwise positions for the wings with max. C_L (red), min. C_D (blue) and the reference one (black).

6.2 Industrial Case 2: Shape Optimization of the DrivAer Car

The second industrial problem is concerned with the optimization of the shape of the fastback configuration of the DrivAer car model [68]. The air flows in the axial direction with $11m/s$, while road and car wheels remain static. The aim is to redesign the car for min. drag coefficient (C_D). The car shape parameterization (together with the displacement of a part of the 3D grid around it) was based on the volumetric NURBS method, [132]. In particular, a $7 \times 7 \times 7$ NURBS control box is used to parameterize the volume around the car boat tail and rear under-body, fig. 6.11. The three Cartesian coordinates of the internal rows of control points in each direction are allowed to vary. All other control points are kept fixed, in order to ensure grid continuity and smooth transition between deformable and non-deformable areas. This setup results in 81 design variables in total. Due to symmetry, only half of the car is modeled. The volumetric NURBS method



(a) Reference. **(b)** Max. C_L.



(c) Min. C_D.

Figure 6.7: Industrial Problem 1: Comparison of the pressure fields for different wing shapes.

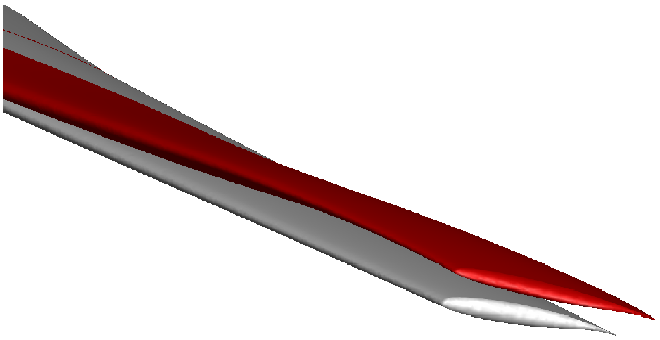


Figure 6.8: Industrial Problem 1: Comparison between the reference wing (grey) and the one optimized for max. C_L (red).

additionally undertakes the deformation of the CFD grid according to the changing geometry of the car.

Each candidate solution is evaluated on the incompressible fluid flow solver of the PUMA software. Even though mild flow unsteadiness might appear at the

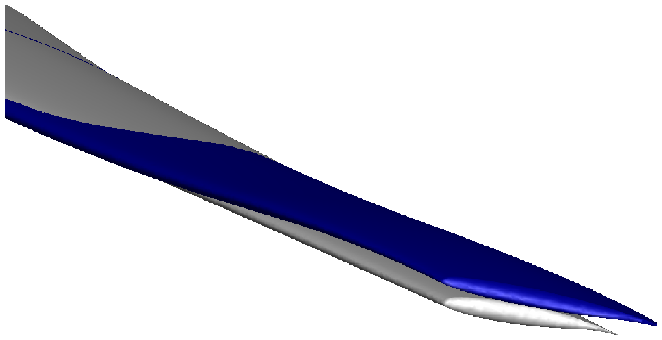


Figure 6.9: Industrial Problem 1: Comparison between the reference wing (grey) and the optimized for min. C_D (blue).

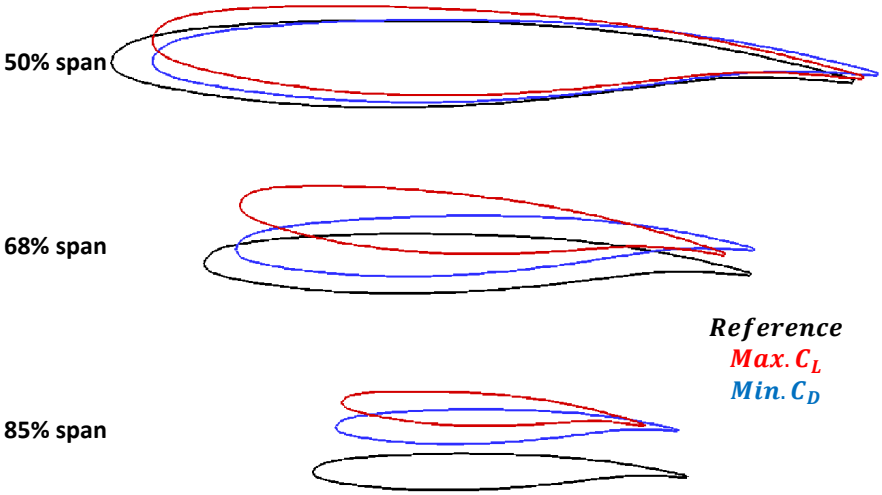


Figure 6.10: Industrial Problem 1: Airfoil shapes at three span-wise positions of the wings optimized for min. C_D (blue) and max. C_L as well as the reference one.

rear part of the car, a steady flow solver is used. To extract valid results for the objective function, the objective function is the time-averaged drag of the last iterations. The Spalart-Allmaras [164] turbulence model is used. The CFD grid consists of $\sim 1.4M$ nodes and each evaluation takes about ~ 40min on an NVIDIA K20 GPU, including morphing.

The optimization is carried out using a $(10, 20)$ EA, MAEA, MAEA(K) and M(K)AEA(K). The stopping criterion is 200 evaluations on the CFD model. In the variants using metamodels, $T^{MM} = 20$ and $\lambda_e = 3$. The convergence histories of the EA and MAEA are presented in fig. 6.12, while those of the MAEA(K) and M(K)AEA(K) in fig. 6.13. It is clear that, the use of metamodels accelerates the optimization algorithm and make it find better solutions with the same computational budget. Only the population of the first generation evolves without the assistance of the KPCA. Judging

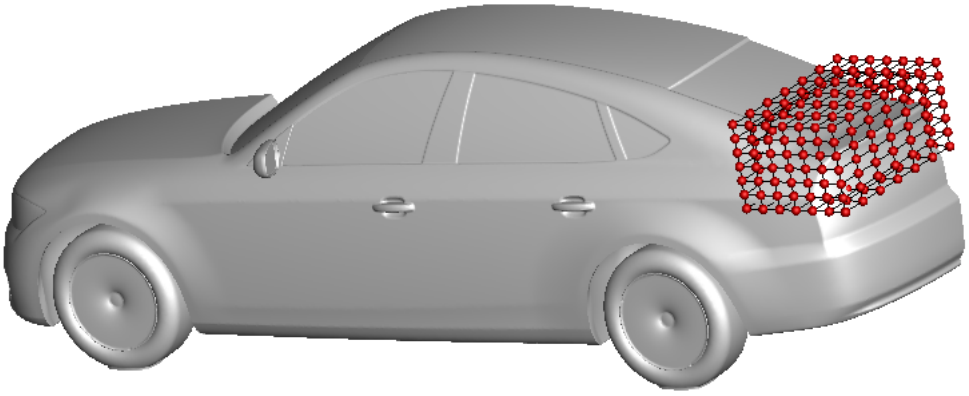


Figure 6.11: Industrial Problem 2: The NURBS morphing box encapsulating the rear part of the car is shown.

from the convergence results, more generation should have passed before starting the KPCA, because, at the beginning, the KPCA under-performs. This shows that each optimization should be carefully setted up and tuned. Once enough calls to the PSM have been made and the population used for the KPCA training is representative of the problem in hand, the optimization take advantage of the usage of KPCA. Finally, both variants outperform the standard MAEA. A comparison between the pressure fields on the optimal and reference car shapes is shown in fig. 6.14. The rear part of the optimized geometry exhibits higher pressure and this reduces the drag coefficient (C_D).

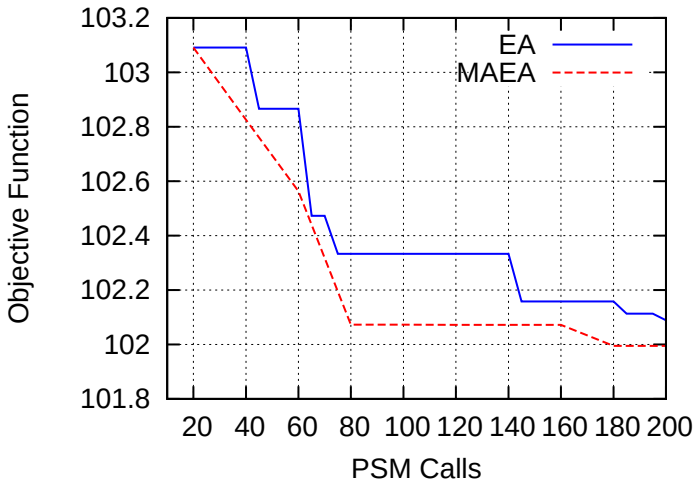


Figure 6.12: Industrial Problem 2: Comparison of the averaged convergence histories for three different RNG seeds of EA and MAEA, in terms of the number of CFD evaluations.

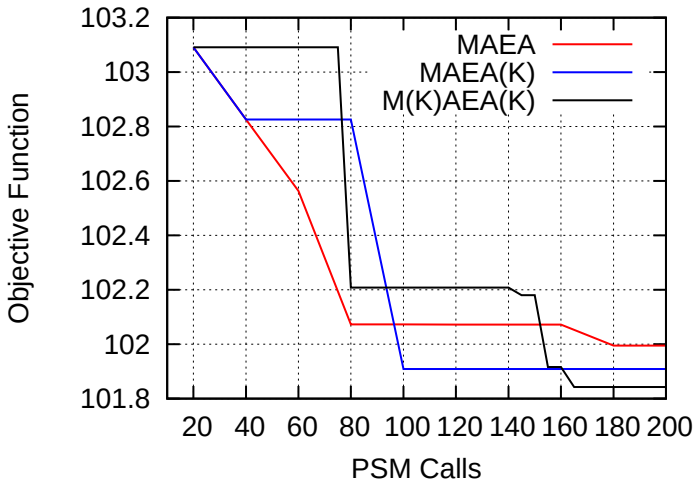


Figure 6.13: Industrial Problem 2: Comparison of the averaged convergence histories of MAEA, MAEA(K) and M(K)AEA(K), in terms of the number of CFD evaluations.

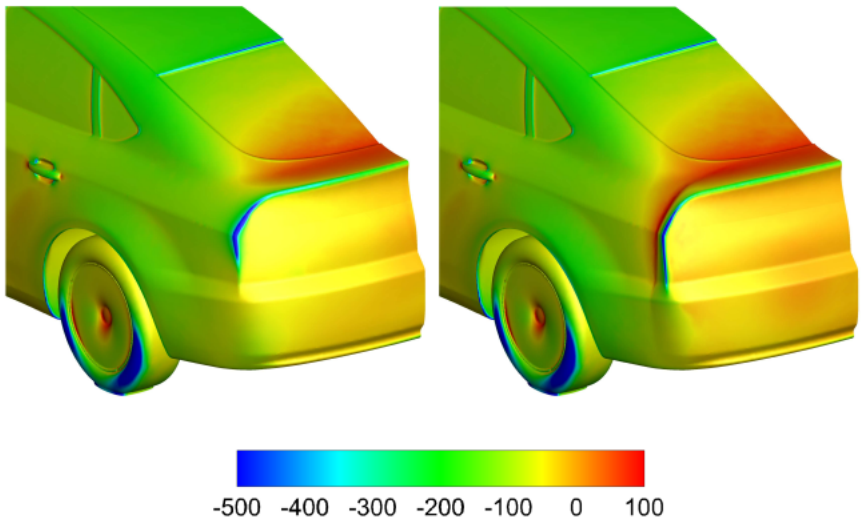


Figure 6.14: Industrial Problem 2: Comparison of the pressure field on the optimal (right) and the reference (left) car shapes.

6.3 Industrial Case 3: Shape Optimization of an Ultra-light Aircraft

The third industrial case is concerned with the re-design/optimization of an ultra-light aircraft [11] aiming at the minimization of its drag force (D). The aircraft geometry was kindly provided by Pipistrel, a light aircraft manufacturer, partner in the RBF4AERO project (*Innovative benchmark technology for aircraft engineering*

design and efficient design phase optimization, Grant Agreement 605396, FP7). The re-design focuses on the wing root-body junction, which is control by 343 NURBS control points shown in fig. 6.15. These control points form a parallelepiped control box with 7 control points in each direction. The two outside layers (black points in fig. 6.15) are not allowed to move so as to limit the morphing action and ensure the CFD grid continuity. Thus, the remaining control points are 27 and are allowed to move along the y and z axis, resulting in a total of 54 design variables.

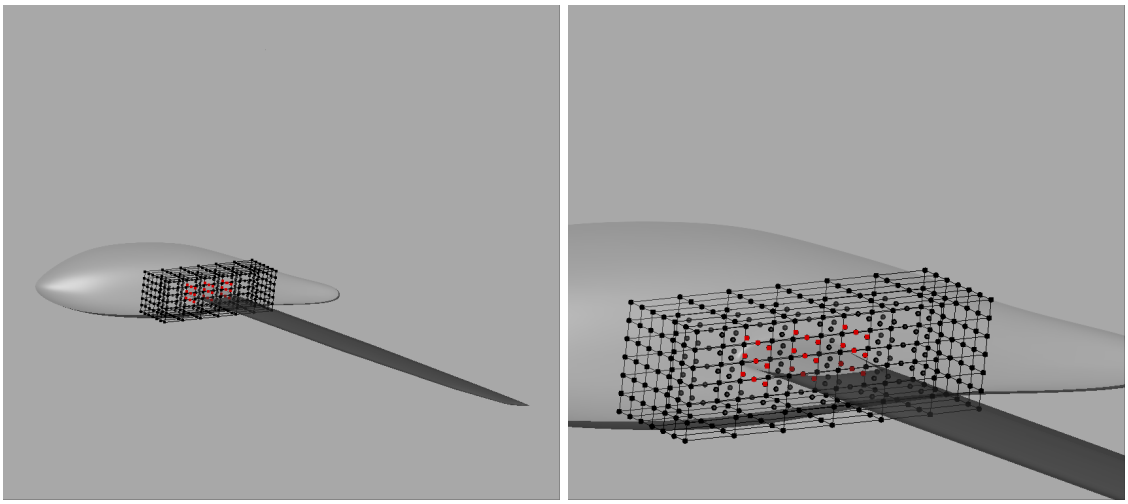


Figure 6.15: Industrial Problem 3: Left: Control box along with the ultra-light aircraft. Right: Close-up view of the wing root-body junction and control points.

The flow conditions are $M_\infty = 0.08$, flow angle 10^o and $Re = 10^6$ (based on the wing chord). Each candidate solution is evaluated on the aforementioned PUMA incompressible flow solver coupled with the Spalart-Allmaras turbulence model [164] with wall functions and it takes ~ 70min on two NVIDIA K20 GPUs, including grid morphing. The CFD grid around the aircraft is unstructured and consists of ~ 1.4M nodes.

A $(10, 20)$ MAEA, MAEA(K) and M(K)AEA(K) are used to optimize the aircraft's shape. The computational budget is limited to 500 calls to the PSM. The variant utilizing metamodels, started using them after $T^{MM} = 40$ individual evaluations. From the $\varepsilon = 10$ elites of each generation, $\lambda_e = 1$ are re-evaluated with the exact evaluation tool. Fig. 6.16 shows convergence plots of the drag normalized with the reference one (D/D_{ref}) for all EA variants. At the beginning, MAEA(K) converges slowly probably due to poor population quality. KPCA may not be able to identify the "correct" feature space (the one that makes the objective function separable), due to the poor characteristics of populations in the first generations. After ~ 240 evaluations, the KPCA starts improving the convergence speed. This shows that the KPCA cannot properly reduce the design variables for the metamodels, thus

their prediction accuracy worsens. There is always a small possibility for this to happen, because the EAs are stochastic algorithms.

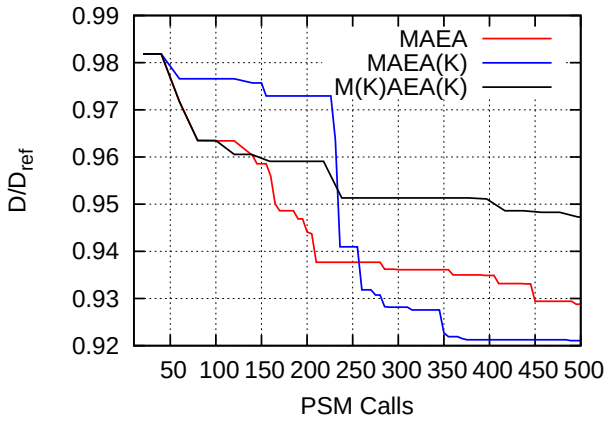


Figure 6.16: Industrial Problem 3: Comparison of the averaged convergence histories of MAEA, MAEA(K) and M(K)AEA(K), in terms of the number of CFD evaluations.

The optimized geometry yields a drag which is lower by $\sim 8\%$ compared to the reference one. The displacement of the junction towards the rear and bottom part of the fuselage (fig. 6.17) is responsible for the observed drag reduction. A by-product of this optimization is that the lift of the optimized geometry becomes higher, though this is not included in the objective function. Figs. 6.17, 6.18 and 6.19 compare the pressure field on the aircraft's surface for the reference and optimized geometries.

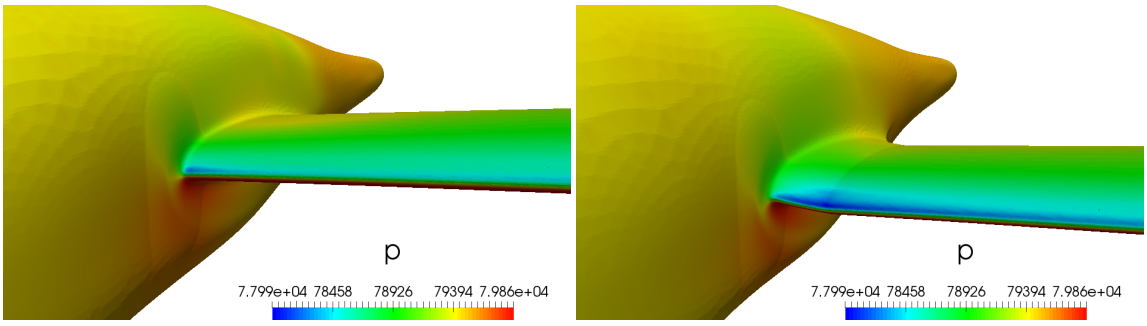


Figure 6.17: Industrial Problem 3: Comparison of the reference (left) and optimized (right) aircraft geometries. Close-up front view of the wing root-body junction.

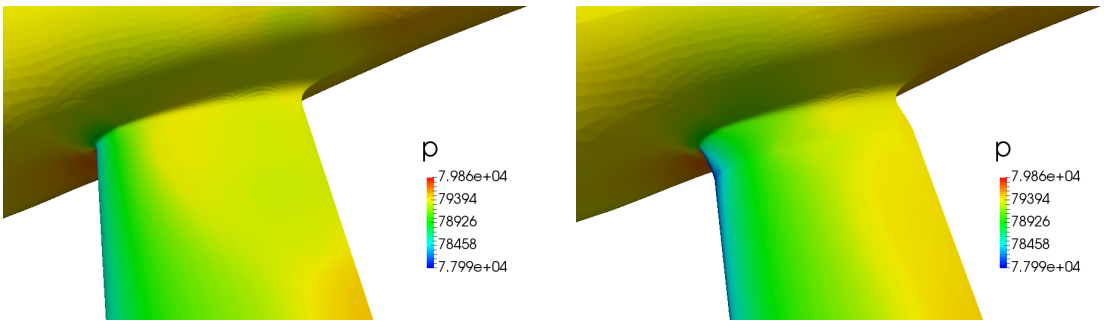


Figure 6.18: Industrial Problem 3: Comparison of the reference (left) and optimized (right) aircraft geometries. Top view.

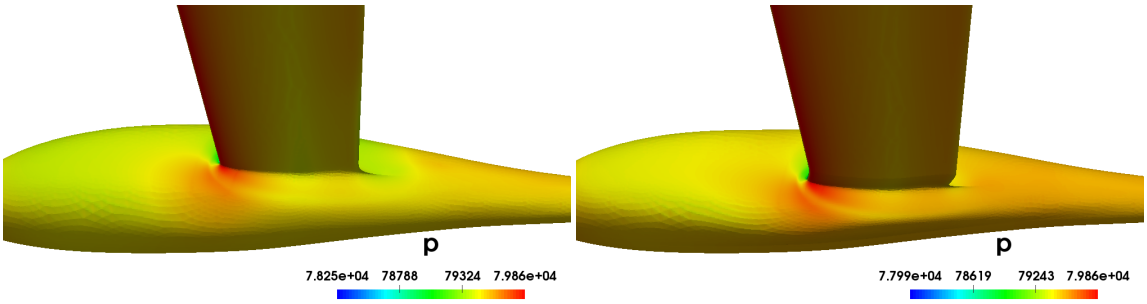


Figure 6.19: Industrial Problem 3: Comparison of the reference (left) and optimized (right) aircraft geometries. Bottom view.

6.4 Industrial Case 4: Shape Optimization of a Francis Runner

This optimization problem is concerned with the shape optimization of a Francis runner with two objectives: max. efficiency and min. cavitation (practiaclly, maximization of the minimum pressure on the blade surface). The flow conditions at the inlet of the runner are $V_{inlet} = 8.198 m/s$, $a_{swirl} = 22.36^o$ and $a_{axial} = 0^o$, for the outlet the static pressure is $39900 Pa$ and the rotation speed is $117.8 rad/s$. This Francis runner is a mixed flow type turbine, the geometry of which can be seen in fig. 6.20. The GMTurbo software, developed by the PCOpt/NTUA [175, 174], is used to parameterize the geometry and provide the design variables for the optimization. It may handle multiple rows and is CAD compatible. A single blade row geometry is defined by meridional projections of the geometry, mean camber lines and thickness profiles. In this case, the GMTurbo tool defines 75 design variables corresponding to the span-wise distributions of quantities parameterizing the camber surface. An unstructured CFD grid of $\sim 300K$ nodes is generated on the CAD model (exported from the GMTurbo software) of the reference geometry. A CFD grid is generated for the reference geometry and morphed accordingly to adapt it to each newly generated blade surface, by means of the spring analogy technique, [171]. The CFD evaluation of each candidate solution takes about

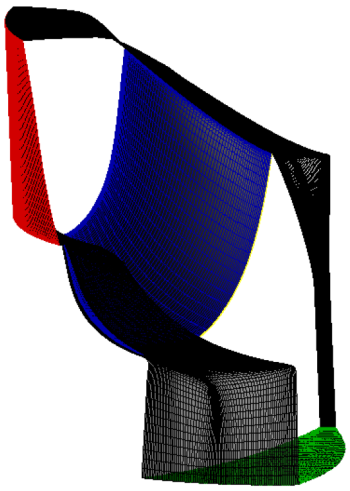


Figure 6.20: Shape Optimization of a Francis Runner: Solid walls along with the inlet (red) and outlet (green) domain.

$1hour$ on one NVIDIA K40 GPU.

A $(10, 20)$MAEA, MAEA(K) and M(K)AEA(K) are used. In the MAEA, the metamodels start being used after the first generation ($T^{MM} = 20$) with $\lambda_e = 2$. A stopping criterion of 300 evaluations on the CFD solver is imposed. The average evolution histories of the hypervolume indicator for three different RNG seeds are compared in fig. 6.21. The performance of the EA driven by the KPCA is poor during the initial generation, but is quickly improved through the evolution. In general, the MAEA(K) does not provide significant improvement over the MAEA, because, probably, the design variables are not highly correlated and the PCA can hardly un-correlate them. On the other hand, truncating of the 55 design variables improves the metamodel prediction accuracy and assists the MAEA to converge faster to even better solutions. It can be seen from fig. 6.22 that the fronts of non-dominated solutions produced by the variants with PCA dominate the MAEA although they are less scattered. Fig. 6.23 shows the reference geometry (suction side) in comparison with the optimized ones for maximum efficiency and minimum cavitation. The pressure distribution on the suction side of each of the aforementioned runners is presented in fig. 6.24.

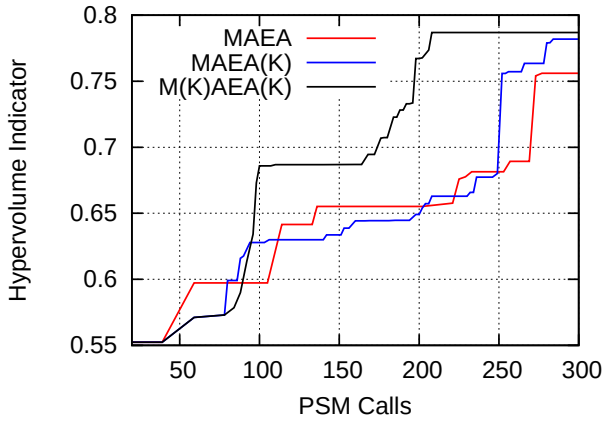


Figure 6.21: Industrial Problem 4: Comparison of the averaged convergence histories of MAEA, MAEA(K) and M(K)AEA(K), in terms of the number of CFD evaluations.

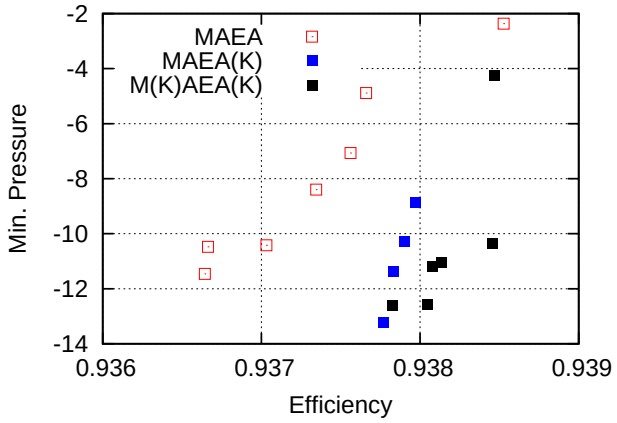


Figure 6.22: Industrial Problem 4: Comparison of the fronts of non-dominated solutions resulted from MAEA, MAEA(K) and M(K)AEA(K).

6.5 Industrial Case 5: Optimization of a Valveless Diaphragm Micropump

This case aims at the optimization of a 3D valveless diaphragm micropump [163]. Diaphragm pumps mainly consist of a main chamber, a moving diaphragm and an inlet and outlet channel. Their key characteristic is that they do not have rotating parts, which is why they are preferred over conventional pumps with blades (especially in medical applications). Instead of blades, the periodic motion of the diaphragm induces the fluids flow inside the chamber. They can also handle various types of fluids with high efficiency and their operation is relatively noiseless. They are manufactured in large or small scales. The large scale pumps are used for cleaning tank bottoms or sewage, while the small scale ones (micropumps)

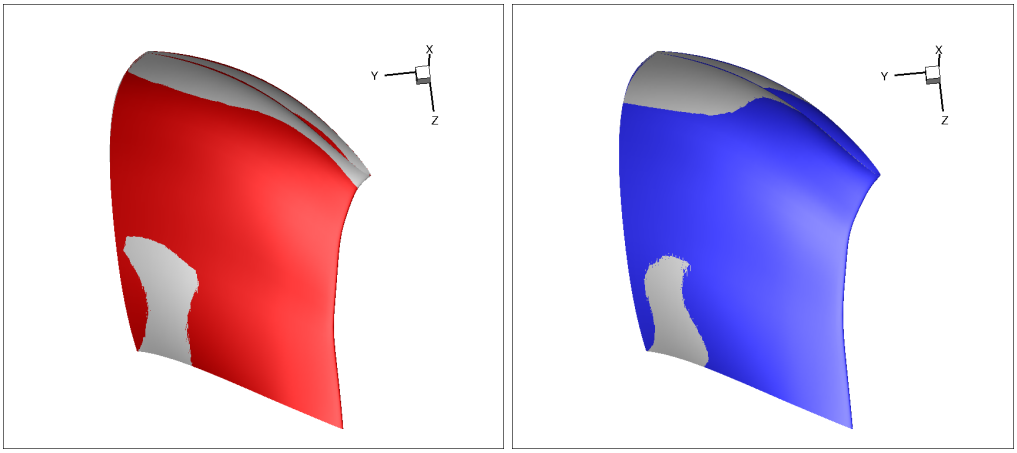


Figure 6.23: Industrial Problem 4: Comparison between the reference runner (grey) and the one with max. efficiency (red, left) and min. cavitation (blue, right).

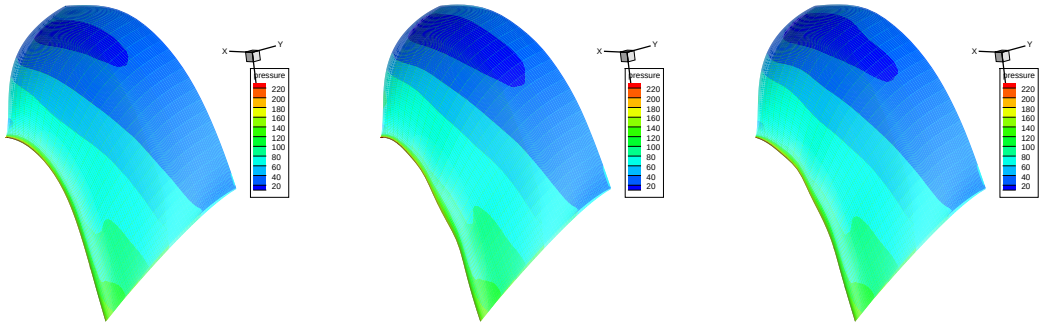


Figure 6.24: Industrial Problem 4: Pressure distribution on the surface of the reference (left), the one with max. efficiency (middle) and that with min. cavitation (right) runners. The optimized geometries resulted from the MAEA run.

are used as medical analysis devices, in biochemical-processing applications or to deliver drugs to patients [18]. Usually, the large scale pumps are equipped with valves do not allow the back-flow, while micropumps are valveless and have diffusers as inlet and outlet channels. Unfortunately, during some time instants of the valveless micropump's operation, some back-flow may appear at the outlet/exit. This is not ideal for medical applications or drug injections and should be minimized during the optimization.

The 3D valveless diaphragm micropump to be optimized is based on an existing design found in the literature [163], fig. 6.25. The main chamber of this pump, fig. 6.26, is modeled as a parallelepiped box, covered by the moving diaphragm. The inlet and outlet diffusers are attached to the two opposite sides of the chamber. The micropump length is $\sim 10mm$. The chamber volume is $\sim 40mm^3$, the inlet cross-sectional area is $0.03mm^2$ and the outlet area is $0.2mm^2$. Both straight diffusers are the same for cost and manufacturing purposes. The mi-

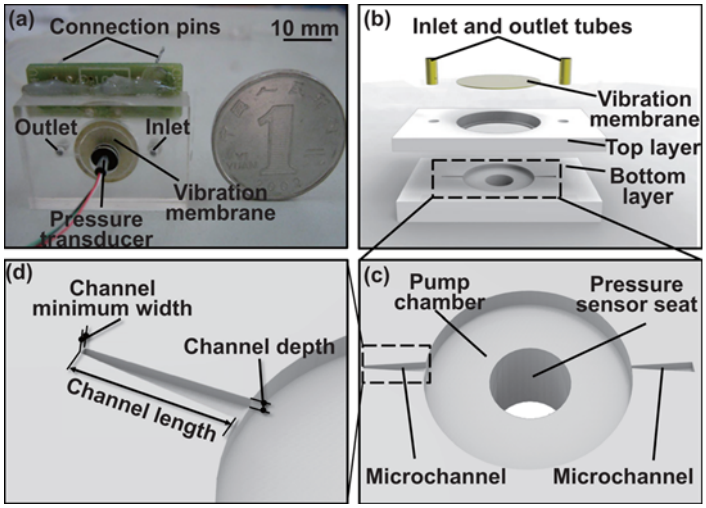


Figure 6.25: Optimization of a Valveless Diaphragm Micropump: The existing micropump found in [163], which was the basis for this design.

cropump operation relies on the periodic motion of the diaphragm, which changes the chamber's volume and the pressure of the contained fluid. Technically, the diaphragm moves thanks to a piezoelectric device. This generates a periodic motion with a predetermined frequency. The needed volume flow rate at the outlet is achieved by the motion's adjustment, which is the ultimate goal of the undertaken optimization.

The diaphragm motion over time is parameterized using 8 design variables in total, which are also the optimization variables. The diaphragm position of rest is at $y = 0$, the x axis points in the longitudinal direction and the z axis in the span-wise direction. The first two design variables b_1 and b_2 is a percentage of the length L_x^m and the width L_z^m of the chamber ceiling; they vary from 75% to 90%. The defined rectangular area contains the diaphragm motion. This area is a bit smaller than the whole chamber's area; the border remains fixed for the diaphragm to be able to mount on the chamber. The rest 6 design variables define the motion over time. The maximum displacement $y_{max}(t)$ over time is computed by the exponential function $y_{max} = b_3 exp(-b_4(t - T/2)^2)(1 - |1 - \frac{2t}{T}|)$, where $b_3 \in [0.1mm, 1.5mm]$ is the maximum displacement over all time-steps given in millimeters, achieved at the half period, b_4 controls the function's abruptness and $T = 0.02s.$ is the period. Thus, y_{max} is increased till the half period and, then, is decreased. The position x in which the maximum displacement occurs at a certain time-step (t) can be computed as $x = L_x^m t/T$; y_{max} is linearly correlated with the longitudinal direction. The longitudinal instantaneous diaphragm motion is a bell-shape deformation, fig. 6.27. Its (right) half curve is defined through the Bezier curve with 5 control points. The first two control points at on $y = 0$ and the remaining at $y = y_{max}$. The final Bezier equation is $y(x,t) = y_{max}(t)(6\tau_x^2 - 8\tau_x^3 + 3\tau_x^4)$, $\tau_x = \frac{x+\delta x}{Dx}$, where $\delta x = b_5 min(x + L_x^m/2, L_x^m/2 - x)$ and $Dx = (1 - b_6)\delta x$.

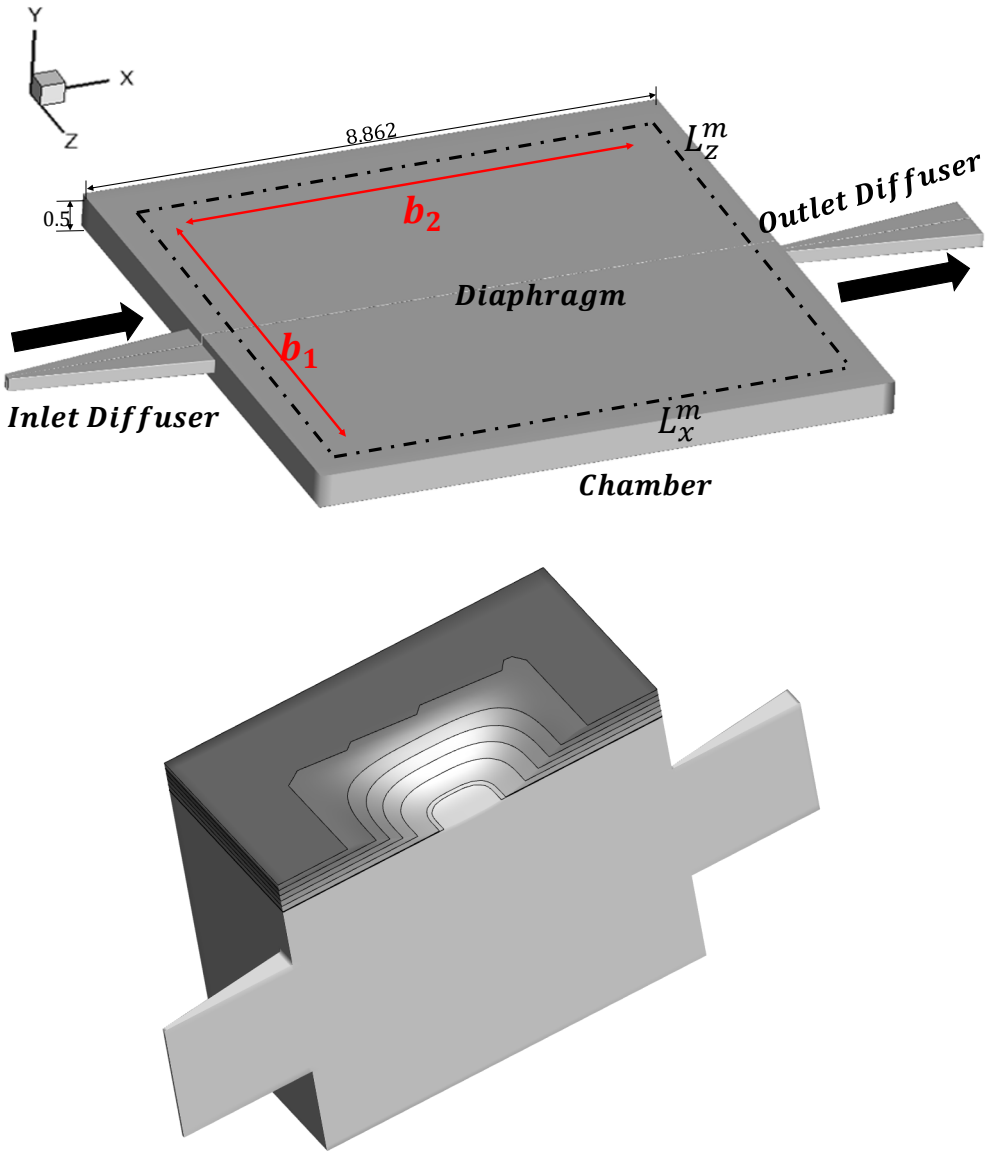


Figure 6.26: Optimization of a Valveless Diaphragm Micropump: Top: The pump with its moving diaphragm. Bottom: Instantaneous diaphragm shape, not in scale.

Thus, two design variables $b_5 \in [0.8, 1.0]$ and $b_6 \in [0.0, 0.2]$ control the Bezier curve. The other (left) half curve of the deformation results from mirroring the right one. The span-wise (along z axis) deformation of the diaphragm is defined through

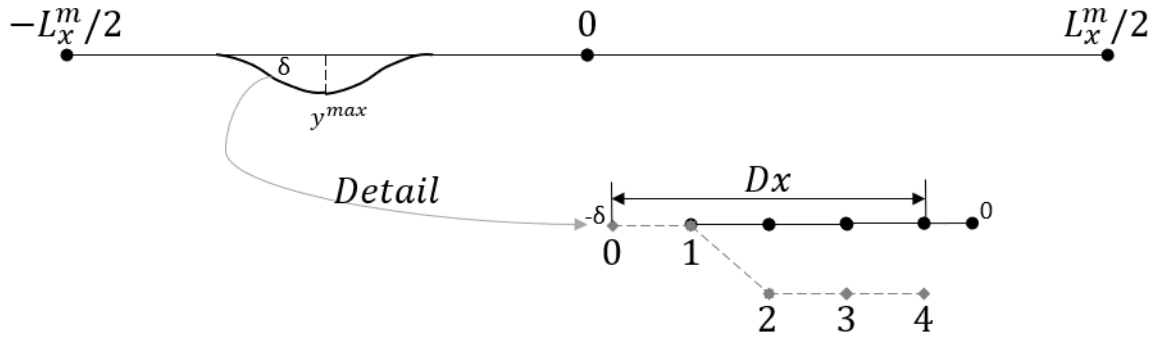


Figure 6.27: Optimization of a Valveless Diaphragm Micropump: Instantaneous deformation of the diaphragm at the symmetry plane. Detailed view of the Bezier curve's control points.

another Bezier curve with 5 control points, similarly with the previous one, fig. 6.27. The design variables $b_7 \in [0.8, 1.0]$ and $b_8 \in [0.0, 0.2]$ define the $\delta z = b_7 min(z + L_z^m/2, L_z^m/2 - z)$ and $Dz = (1 - b_8)\delta z$, which are required to compute the Bezier curve with equation $y(z,t) = y_{max}(t)(6\tau_z^2 - 8\tau_z^3 + 3\tau_z^4)$, $\tau_z = \frac{z+\delta z}{Dz}$. Finally, the overall diaphragm motion is given by combining the longitudinal and span-wise deformation as

$$y(x,z,t) = y_{max}(t)(6\tau_x^2 - 8\tau_x^3 + 3\tau_x^4)(6\tau_z^2 - 8\tau_z^3 + 3\tau_z^4) \tag{6.5.1}$$

with y_{max}, τ_x and τ_z as defined above. Fig. 6.28 shows the final deformation of the diaphragm at four different time instants.

Having defined the shape/geometry of the micropump and the motion of the diaphragm, the unsteady CFD simulation of the flow within a period T follows. The CFD grid is formed by hexahedral elements and $\sim 260K$ nodes for the half micropump, fig. 6.29. Only the half pump is simulated due to its symmetry along the xy plane. The kinematic viscosity of the working fluid is $10^{-6}m^2/s$, the inlet total pressure is $101325.383Pa$, the outlet static pressure $101325Pa$, which (for an arbitrarily selected diaphragm motion; the reference one) induces a small flow from the inlet to the outlet. The CFD software simulates the laminar flow for 20 time instants per period. Due to the unsteadiness of the phenomenon, 3 periods are required to be solved to suppress any transient phenomena and establish periodicity, fig. 6.30. Each simulation takes $\sim 1hours$ in one NVIDIA K40 GPU. Moreover, the cut-cell software is capable of solving the unsteady adjoint PDEs to compute the gradient of the objective function(s) with respect to the design variables. The adjoint solution has more or less the same computational cost as

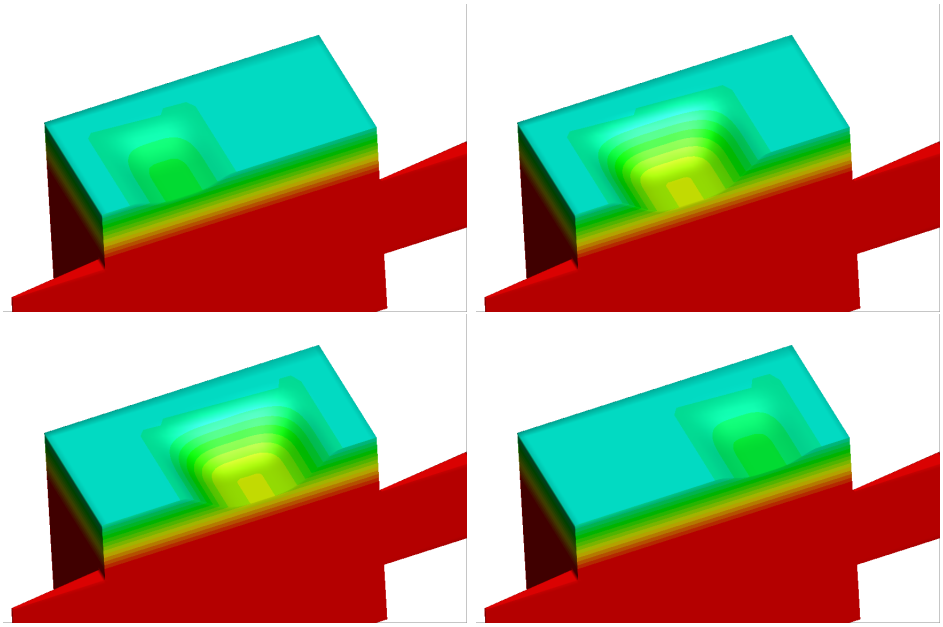


Figure 6.28: Optimization of a Valveless Diaphragm Micropump: Reference diaphragm motion. Diaphragm shapes/strokes at $t = 0.25T$,$0.4T$, $0.6T$ and $0.75T$, from top-left to bottom-right, respectively. Axes not in scale.

a single flow simulation.

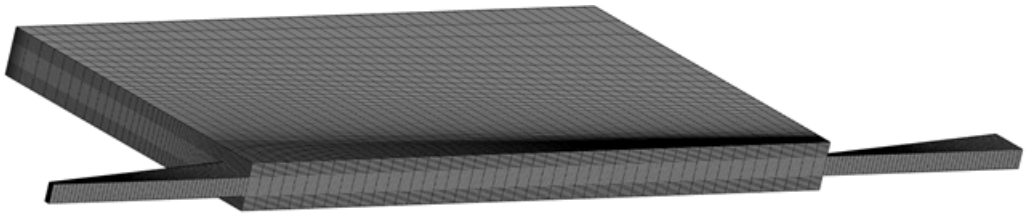


Figure 6.29: Optimization of a Valveless Diaphragm Micropump: CFD grid in half of the pump.

During medical applications, the back-flow at the outlet is unwanted and, thus, this is minimized during the optimization, while the net volume flux should be maximized. The net volume flux at the exit within a period T is computed $Q_{net} = \frac{6 \cdot 10^{10}}{T} \int_T \int_{S_{outlet}} \vec{V}(t) \cdot \vec{n} dS dt (\mu l/min)$, where $\vec{V}(t)$ is the velocity vector and $\vec{n}$ the outward unit normal vector. The volume flow entering from the outlet (back-flow, index bf) is computed by integrating the negative velocity, $Q_{bf} = -\frac{6 \cdot 10^{10}}{T} \int_T \int_{S_{outlet}} min(0, \vec{V}(t) \cdot \vec{n}) dS dt (\mu l/min)$. The instantaneous velocity at the outlet is shown in fig. 6.31. A $(5, 10)$ PCA-Assisted Hybrid Algorithm is used for the two objective optimization problem (min. Q_{bf} and max. Q_{net}), since this usually gives the best convergence speed compared to other EA variants. Both meta-

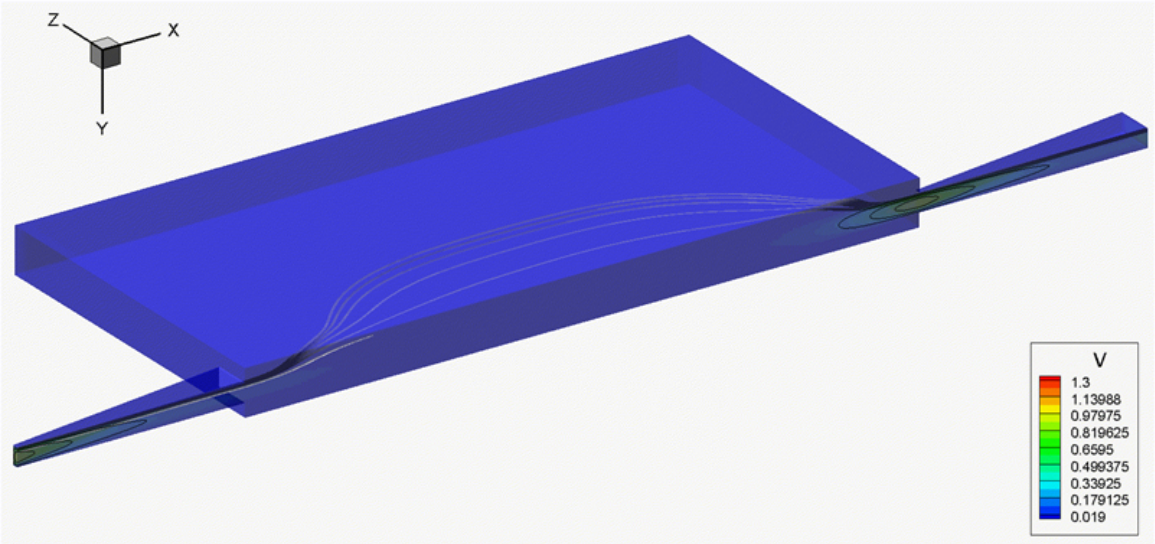


Figure 6.30: Optimization of a Valveless Diaphragm Micropump: Velocity iso-lines and stream-lines.

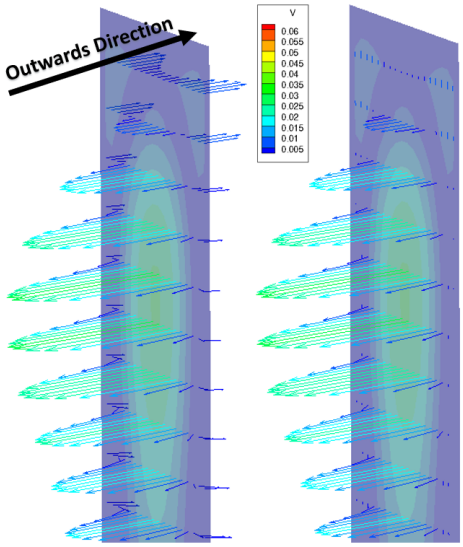


Figure 6.31: Optimization of a Valveless Diaphragm Micropump: Left: Velocity vectors at the outlet used to compute Q_{net}. Right: Negative velocity vector for the outlet used to compute Q_{bf}. Both correspond to the same (arbitrarily selected) time instant.

models and PCA start being used after the first generation. The cut-cell based unsteady adjoint solver provides the required gradients and the GB method refines one individual per generation. The computational budget is restricted to 200 calls to the PSM (or its adjoint), due to their high computational cost ($\sim 1hour$ on an NVIDIA K40 GPU). From these, 63 calls are spent on the adjoint solver. The final front of non-dominated solutions is presented in fig. 6.32.

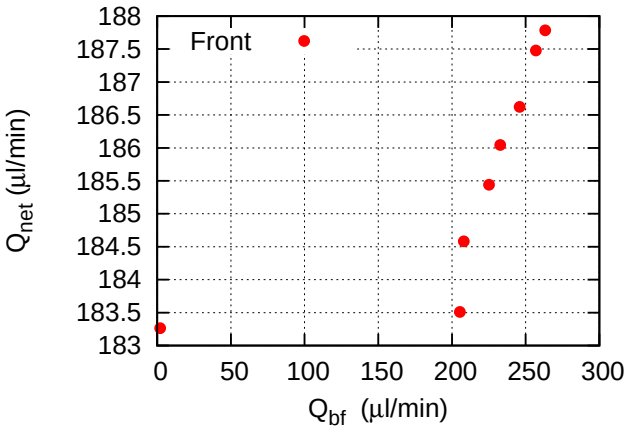


Figure 6.32: Optimization of a Valveless Diaphragm Micropump: Computed front of non-dominated solutions computed by the PCA-Assisted Hybrid Algorithm. The point ($Q_{bf} = 0.516\mu l/min$, $Q_{net} = 5\mu l/min$) which corresponds to the pump operation with the reference diaphragm motion cannot be represented within the selected graph limits.

Fig. 6.33 shows how the back-flow and net volume flux evolves over time, within a period, for the two edges of the front and the reference solution. The solution corresponding to max. Q_{net}, has negative net volume flux for a bit less than half of the period, so there is a noticeable back-flow. However, the positive overweighs the negative part resulting in max. Q_{net}. Regarding the min. Q_{bf} solution, back-flow becomes negative during only one step given a time resolution of 20 steps within a period. During this time step, the instantaneous net volume flux remains positive. In contrast, the reference solution has much greater back-flow and less net volume flux compared with the other solutions. The net volume flux is positive during half of the period yielding a small Q_{net}. From this point of view, any of the Pareto solutions by far outperforms the reference pump.

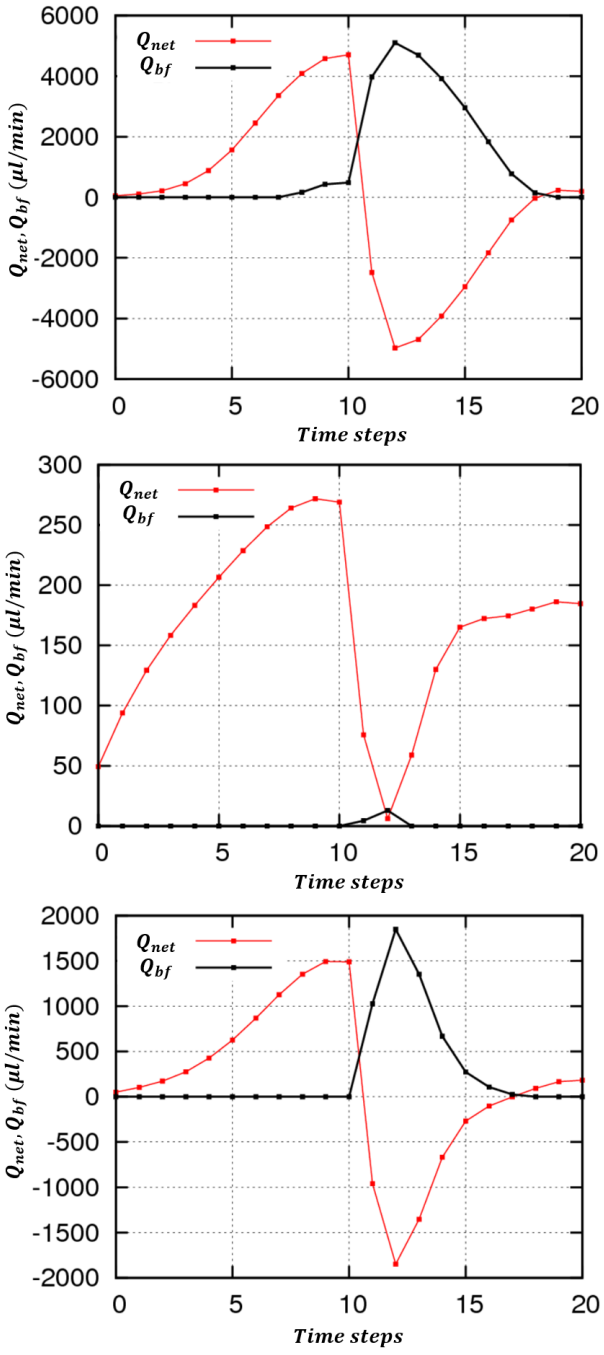


Figure 6.33: Optimization of a Valveless Diaphragm Micropump: Q_{bf} and Q_{net} time-series for the reference (top), min. Q_{bf} (middle) and max. Q_{net} motion (bottom).

Chapter 7

Flow Prediction using Deep Neural Networks

In design/optimization practices, a number of different designs, usually evaluated by CFD runs, are processed to finally derive the optimal solution(s). As it has already been mentioned, CFD methods may generate highly accurate results but are computationally expensive. While industrial requirements are increased, CFD codes are used to solve more and more complex flow problems and, thus, the computational cost increases a lot. Significant efforts have been made to reduce, as much as possible, the cost of a CFD analysis. State-of-the-art solvers, such as the PUMA used herein, take advantage of the GPU processing power and speed, which dramatically reduce the simulation cost but this is still quite high. On the other hand, for big data [119], Artificial Neural Networks (ANNs) and, in specific, Deep Neural Networks (DNNs) [29], which is a special class of ANNs, have been repeatedly proven to learn complex and non-linear features that are informative for the task in hand; these features make them appealing for modeling complex aerodynamic problems. In addition to this, industries have been making similar designs of their products, for years, building up massive databases with the results of the corresponding CFD analyses. These databases can be used to train DNNs to replicate the expensive CFD solvers with negligible cost (this cost does not include the cost of training the DNNs). That is why dedicated ANNs or DNNs are developed to approximate as accurately as possible the outcomes of CFD simulations and provide a viable alternative to the way expensive CFD solvers are used within an optimization loop.

ANNs are in widespread use in aerodynamic optimization in the last two, at least, decades. In the literature, two major categories of ANN applications can be found based on the end goal of the problem in hand. The first one deals with the prediction of the integral aerodynamic quantities, otherwise computed by post-processing the outcome of the CFD analysis. The second one deals with the use of ANNs for flow field predictions around or inside aerodynamic bodies. The first

category is more popular and more widely used than the second one. A brief description of both methods follows.

The first category is used to predict a small number (sometimes, a single one only) of integral responses (such as the lift, drag or moment) of CFD simulations in optimization loops involving a relatively low number of design variables (compared to the second category, see below). These predictions replace the objective function values. This is exactly the way ANNs are implemented within the MAEAs, or any of their variants, presented in this thesis (as in the previous chapters). A distinguishing feature of the implementation of ANNs in this thesis, so far at least, is that there is no single ANN covering the whole design space; instead, personalized ANNs are trained for each new individual to be (pre-)evaluated. Based on the standard terminology of this thesis, this usage of ANNs is referred to as "on-line trained", to distinguish them from "off-line trained" ANNs. Regarding ANN types, the feedforward ANNs and Recurrent Neural Networks (RNNs) are often utilized in these applications, where only integral responses are predicted. RNNs are used in time/history-dependent problems; they incorporate memory effects in their systems and can predict responses affected by previous states, such as unsteady flows. Despite the complexity of aerodynamic problems, the ANNs architecture is usually compact with a low number of layers, neurons and weights/parameters (such as the RBF networks used in previous chapters), due to the relatively small set of design/input variables. Usually, more input variables require more layers, neurons and weights. Even with compact architectures, ANNs are able to accurately predict 3D unsteady aerodynamic metrics/objective functions.

On the other hand, in the second category of ANN applications, the flow field around/inside aerodynamic bodies is predicted. In [157, 62, 69], ANNs and more specifically DNNs, are used to predict entire flow fields. Networks of this category are bound to deploy larger, more complex ANN architectures to cope with spatial dependencies in the flow domains. The Convolutional Neural Networks (CNNs), a subcategory of DNNs, have been proved powerful at solving similar problems with spatial dependencies. CNNs are specialized in extracting patterns and features from topological data and are widely used in image processing. These CNNs display great potential to replicate the CFD solvers with significantly lower computational cost. In [62], a general and flexible approximation model based on CNNs for real-time predictions of steady laminar flow in 2D or 3D domains is proposed. Trained CNNs are shown to accurately estimate the velocity field faster than any GPU- or CPU-enabled CFD solver. This approach can provide immediate feedback for real-time design at early design stages. In [69], a method based on DNNs to perform automated design on complex real-world engineering tasks is proposed. A DNN is trained to compute the pressure field which is, then, integrated to calculate the objective function(s) used in an optimization. This method is demonstrated in shape optimization cases of airfoils and wings aiming at maximum lift to drag ratio. In [156], unsteady surface pressures on a wing pitching

beyond static stall are firstly measured. Then, both linear and non-linear ANNs trained on these data learn to predict unsteady surface pressures and unsteady aerodynamic loads. ANNs predictions are compared directly to surface pressures and aerodynamic loads computed by CFD codes and proved to accurately predict both. In [157], an efficient surrogate model is developed for the prediction of motion-induced unsteady surface pressure fluctuations, integral forces and moment coefficients. A recurrent linear neuro-fuzzy approach is deployed to capture the characteristics of the dynamic system. Once the reduced-order model (ROM) is trained, it can replace the CFD solver in unsteady aerodynamic or aeroelastic simulations for a fixed aerodynamic shape but different flow conditions. For demonstration purposes, the ROM approach is applied on the LANN wing in high subsonic and transonic flow yielding a close match between the ROM's predictions and the CFD solutions. This surrogate approach significantly speeds-up the unsteady aerodynamic calculations, which is beneficial for multidisciplinary computations. Even though, in some problems only integral flow quantities are required for the aerodynamic bodies analyses, DNN-based predictions of whole flow fields show great potential and capabilities, which industries may appreciate and use in several applications.

Herein, complex DNNs, which combine different state-of-the-art layers and architectures, are used to predict the flow fields in 2D or 3D aerodynamic cases, such as flows around airfoils and wings. The set of training patterns (database) consists of flow fields of aerodynamic shapes simulated with the PUMA solver, using unstructured grids. Their predictions, as it will be demonstrated later in this chapter, have acceptably low error compared to the CFD simulations and their cost is negligible compared to that of the PUMA solver (it is lower by several orders of magnitude, excluding the training cost of DNNs). This makes these DNNs a viable tool for quick CFD design/optimization and other applications requiring CFD simulations. Nevertheless, the use of CFD simulations is a prerequisite for the preparation of the database used to train the DNNs as well as for verifying the quality of the optimized (using exclusively the DNN) solution.

7.1 Basics of DNNs

During the last decade, DNNs have been successfully applied in fields including, but not limited to, computer vision, speech/audio/image recognition, natural language processing. They were introduced to the machine learning community by Rina Dechter in [28, 153].

For a better understanding of DNNs, an introduction to ANNs is first required. ANNs are a set of algorithms designed after the human brain and mammal nervous system and inspired by information processing in biological systems. Their main computational units are the artificial neurons which are organized in layers, fig. 7.1, and modeled based on the structure and synapses of biological neurons,

fig. 7.2. Each biological neuron receives signals from other neurons via their dendrites. The cell (neuron) body processes the signals and it sends an output down its one axon, only if this exceeds a given threshold. The axon splits into multiple branches, connecting the axon via synapses to the dendrites of numerous other neurons. Detailed discussion about biological neurons is beyond the scope of this thesis.

Artificial neurons are built similarly, fig. 7.2. Their synapses correspond to weighted signals forwarded from other neurons. Their dendrites carry those signals to the cell body where the main processing is performed. The main operation is the summation of the weighted signals, which is passed down through the dendrites. The threshold for sending the output down the axon is modeled using activation functions. These are mathematical functions applied to each neuron and are used to filter the neuron's output by introducing non-linearities that help the DNN to handle complex functions/tasks. The most popular activation functions are the sigmoid ($g(z) = \frac{1}{1+e^{-z}}$), the tangent hyperbolic ($g(z) = tanh(z)$), the rectifier linear unit (ReLU) ($g(z) = max(0, z)$) and the softmax [42] along with its variations, expressed by system of equations not presented here.

Artificial neurons perform a transformation of their input followed by an element-wise non-linearity (i.e. one that is applied on each neuron, separately), mathematically shown in fig. 7.2 and the corresponding operation is

$$o_j = g(\sum_{i}^{r} w_{ij}x_i + b_j) \tag{7.1.1}$$

where w_{ij} are the weights connecting the i^{th} with the j^{th} neuron, x_i are the signals passed from the i^{th} neuron, b_j is the j^{th} neuron's bias, g is the activation function and r is the number of the layer's neurons. ANN consisting of multiple neurons and layers, can be expressed mathematically via a complex function which combines each individual neuron's expression. The parameters v are computed during the training phase and determine the properties and prediction ability of the ANN. These are essentially the weights and bias of all available neurons.

The procedure of computing these parameters is the training phase of the ANN, which is computationally expensive. During the training, some training patterns of the task to be approximated are fed into the ANN. which are essentially the combination of the inputs and the corresponding outputs found in the database. The ANN learns to predict the outputs from the given set of inputs. In "swallow" ANNs (networks with one intermediate layer, such as an RBF network), the training process is the solution of a system of equations computing the ANN parameters. In DNNs, the training process is an optimization which minimizes an appropriate cost function by adjusting accordingly the values of the parameters. This cost function measures the DNN's prediction ability by checking the difference between the real outputs paired with the training patterns and their

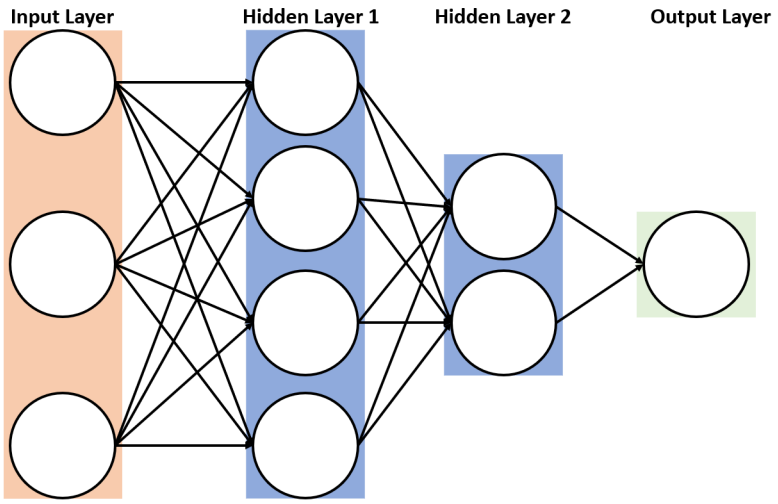


Figure 7.1: Multi-layer ANN with a single output unit.

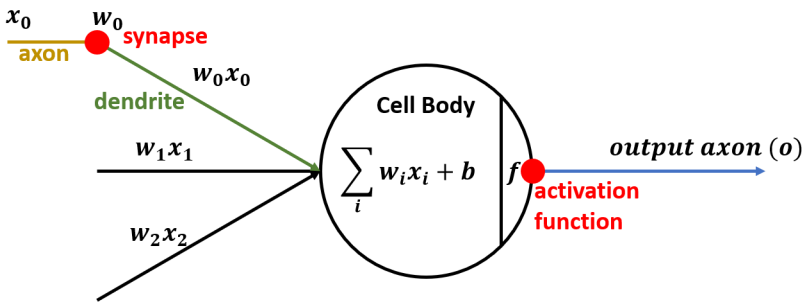


Figure 7.2: Graphical representation of an artificial neuron.

predictions. Mathematically, if the input units are $\vec{x}$ and the outputs are $\vec{y}$, then the cost function to be minimized is expressed as $C = \frac{1}{N}\sum_{i=0}^{N} C_i(w, b, \vec{x}^{(i)}, \vec{y}^{(i)})$ where C_i is the cost function for a single training pattern/example and N is the total number of training patterns. C_i can be many different functions such as the absolute difference or the mean square value of the i^{th} pattern compared to its prediction. In general, the DNNs are expressed with analytical functions, thus, the derivatives $\frac{\partial C}{\partial v}$ of the cost function with respect to (w.r.t.) the parameters are easily calculated analytically. Note that parameters are the weights and biases of the DNN neurons. This is why the optimization methods utilized for training DNNs are mainly gradient-based ones. These methods, firstly, compute the gradients of the cost function w.r.t. the parameters. The latter are updated via the equation $v^{new} = v^{old} - \eta\frac{\partial C}{\partial v}$ where η is the learning rate. This procedure is performed repetively till convergence. The result of the optimization/training phase is the optimal values of the parameters with which DNNs predict, as accurately as possible, the training patterns.

w_{ij}^k	weight for the j^{th} neuron of the k^{th} layer coming from the i^{th} neuron of the previous layer
b_i^k	bias for the i^{th} neuron of the k^{th} layer
a_i^k	product sum plus bias for the i^{th} neuron of the k^{th} layer
o_i^k	output for the i^{th} neuron of the k^{th} layer
r^k	number of neurons in the k^{th} layer
g	activation function

Table 7.1: Basic DNN symbols.

7.1.1 Basic DNN Mathematics and the Back-Propagation Algorithm

Even though the gradients required for training DNNs can be computed analytically, this analytical method has been shown to be extremely slow for the DNNs. An efficient method for updating the parameters is the back-propagation algorithm proposed by Rumelhart in [145], which is a key method in machine learning. Back-propagation allows the information from the cost function to propagate backwards through the DNN architecture, in order to compute the gradients of each neuron through each layer. This is implemented with the assistance of the chain rule for the gradients. According to LeCun [115], the premise of back-propagation is that "the derivative of the output w.r.t. the input of a layer can be computed by working backwards from the gradient with respect to the output of that layer (or the input of the subsequent layer)". This procedure is applied repetitively from the output layer to the input layer, so as to compute all required gradients.

Before diving into the mathematics of DNNs and back-propagation algorithm, some essential symbols are defined in Table 7.1.

The outputs (o_i^k is the output after and a_i^k before the activation function) of each neuron in a multi-layer structure are computed as:

$$a_i^k = \sum_l^{r^k} w_{li}^k o_l^{k-1} + b_i^k$$

$$o_i^k = g(\sum_l^{r^k} w_{li}^k o_l^{k-1} + b_i^k) = g(a_i^k) \tag{7.1.1.1}$$

For simplicity, a single-output DNN is assumed with the output of the last m layer (output layer) denoted by $\hat{f} = g(a_1^m)$. During the training phase, the cost function C should be minimized. The derivatives of the cost function w.r.t. w and b are

computed with the back-propagation algorithm. Firstly, the partial derivative of C w.r.t. w_{ij}^k is computed as:

$$\frac{\partial C}{\partial w_{ij}^k} = \frac{\partial C}{\partial a_j^k}\frac{\partial a_j^k}{\partial w_{ij}^k} \tag{7.1.1.2}$$

The decomposition of this partial derivative shows that the change in the cost function from the weight is the product of the change due to its activation and the activation's change due to the weight. $\frac{\partial a_j^k}{\partial w_{ij}^k}$ is computed as $\frac{\partial a_j^k}{\partial w_{ij}^k} = \frac{\partial}{\partial w_{ij}^k}(\sum_{l=0}^{r_{k-1}} w_{lj}^k o_l^{k-1}) = o_i^{k-1}$ and $\frac{\partial C}{\partial a_j^k} = \delta_j^k$ is essentially the neuron's error, δ is Kronecker delta. Eq. 7.1.1.2 is re-written as:

$$\frac{\partial C}{\partial w_{ij}^k} = \delta_j^k o_i^{k-1} \tag{7.1.1.3}$$

Now, the neuron's error δ_j^k can be computed. For the output layer, this error is based on the cost function. For the remaining layers, it can be written as:

$$\delta_j^k = \frac{\partial C}{\partial a_j^k} = \sum_{l=1}^{r^{k+1}} \frac{\partial C}{\partial a_l^{k+1}}\frac{\partial a_l^{k+1}}{\partial a_j^k}$$

or,

$$\delta_j^k = \sum_{l=1}^{r^{k+1}} \delta_l^{k+1}\frac{\partial a_l^{k+1}}{\partial a_j^k}$$

Based on the definition of a_l^{k+1}, eq. 7.1.1.1 , re-written as

$$a_l^{k+1} = \sum_{j=1}^{r^k} w_{jl}^{k+1} g(a_j^k)$$

its derivative is

$$\frac{\partial a_l^{k+1}}{\partial a_j^k} = w_{jl}^{k+1} g'(a_j^k)$$

Plugging all the equations together, δ_j^k is expressed as

$$\delta_j^k = g'(a_j^k) \sum_{l=1}^{r^{k+1}} \delta_l^{k+1} w_{jl}^{k+1}$$

and the derivatives of the cost function w.r.t. the weights are

$$\frac{\partial C}{\partial w_{ij}^k} = o_i^{k-1} g'(a_j^k) \sum_{l=1}^{r^{k+1}} \delta_l^{k+1} w_{jl}^{k+1} \tag{7.1.1.4}$$

The derivatives of the cost function w.r.t. the biases b_j^k, are

$$\frac{\partial C}{\partial b_j^k} = \frac{\partial C}{\partial a_j^k} \frac{\partial a_j^k}{\partial b_j^k} = \delta_j^k \frac{\partial a_j^k}{\partial b_j^k} \tag{7.1.1.5}$$

b_j^k is the biases and $\frac{\partial a_j^k}{\partial b_j^k}$ is equal to unit, thus, the eq. 7.1.1.5 is written as

$$\frac{\partial C}{\partial b_j^k} = g'(a_j^k) \sum_{l=1}^{r^{k+1}} \delta_l^{k+1} w_{jl}^{k+1} \tag{7.1.1.6}$$

Eqs. 7.1.1.6 and 7.1.1.4 show that the information flows backwards and the algorithm is named after that. First, the error terms based on the cost function are computed. Then, the error terms for the previous layer are computed by performing a weighted product sum of the neurons error for the next layer (the errors are multiplied with weights and, then, summed) and scaling it by the activation function's derivatives and this is repeated until the input layer is reached. The DNN output is updated in the forward phase, while the backward phase computes neurons errors' and derivatives. Furthermore, the neurons outputs after the application of the activation function must be computed before the backward phase, because the derivatives and errors are dependent on them. Thus, in each cycle of the gradient-based optimization, the forward phase computes the activation functions and outputs, the backward phase computes the required partial derivatives and only then the weights and biases can be updated. This process is repeated till the minimization of the cost function.

7.2 Gradient-Based Optimization for DNN training

As already mentioned, DNNs are trained using of Gradient-Based (GB) methods which minimize the cost function by adjusting the network parameters, so as for the DNN to accurately predict the output(s) of the training patterns. Recall that GB methods can quickly reach optimal solutions, but they may get "trapped" into local optima. For this reason, an efficient GB method capable of overcoming local optimum has been devised. In previous chapter(s), the simplest GB, steepest

descent, was used. The gradients w.r.t. the parameters are computed via the back-propagation algorithm. Except from these gradients, the method requires the definition of the learning rate η. Its selection is a compromise between speed and accuracy of the learning process. Big learning rate means big jumps in the objective space, which may hinder the optimization's convergence to the global minimum, without though being "trapped" into a local minimum. On the other hand, small learning rate means small steps resulting on a higher number of iterations and higher computational cost. Several methods have been proposed for optimally calculating η and avoiding local minima. Usually, these methods are based on the steepest descent algorithm. Some of the most popular methods are briefly explained:

Gradient descent with momentum [145]: This method takes advantage of exponentially weighted averages (EWA) to avoid problems with gradients close to zero. The algorithm gains momentum from the previously computed gradients, so as to move forward even if some local gradients are zero. The weights and biases are updated with the used of EWA focusing the optimization on the most important parameters. Parameters (aforementioned trainable weights and biases) helping the minimization are amplified and those being responsible for the oscillations of the algorithm's convergence are slowly eliminated. This leads to faster convergence and reduced oscillations, but, while the minimum is approached, the momentum may become so large that the algorithm will not be able to stop at the global minima.

RMSProp [169]: The Root Mean Squared Propagation is an adaptive method using EWA, which means that it allows for the adjustment of the learning rate separately for each parameter. The current parameters are based on gradients computed during the previous iterations. In each iteration, EWA is used to average the element-wise squares of derivatives of the cost function w.r.t. the parameters. This averaged square root is summed for each element and then divides the parameter's update. This algorithm reduces oscillations and prevents noise. The drawback of this method is that as the denominator of the updates increases in each iteration, the learning rate is getting smaller.

Adam [103]: The Adaptive Moment Estimation which takes advantage of the biggest advantages of RMSProp and combine them with ideas known from momentum optimization. The result is a method that allows for quick and effective optimization, but has a complex and more expensive algorithm.

Apart from these methods, another way to improve the convergence speed of the optimization/training is to break the training data into subsets or batches [116]. If large sets are to be implemented, there is no need to compute the exact gradients using each sample. In fact, the training process can be run faster, if

rapid estimates of the gradients can be used instead of the computationally heavy exact gradients. This reduces the overall computational cost per optimization cycle and requires less memory. The methods using batches are named batch gradient descent methods and are essential in large DNNs with millions of training patterns. Further analysis of the training methods used for the networks training is beyond the scope of this thesis. As it will be mentioned below, the DNNs in this thesis are trained with the Adam method and the training patterns are broken into subsets.

7.3 Network Architecture

The first type of DNN is the feedforward neural network. It consists of multiple neurons arranged into layers and connected with each other. There are three general types of layers found in any DNN:

Input Layer: The input layer contains all the input neurons, the input information provided to the network to learn the given task. This layer does not perform any computation.

Hidden Layers: These layers perform all the computations and transformation of information from the input to the output layer. They do not interact with the outside environment of the DNN, hence the name "hidden".

Output Layer: This layer contains the results of the DNNs processes. It is responsible for transferring information from the network to the outside environment.

It is reminded that "Swallow" ANNs are the simplest types of ANN with only one hidden layer. Every ANN containing more hidden layers is called DNN. Each hidden layer can perform different operations and computations, contains different number of neurons and uses different activation functions. In this thesis, the types of layers used are mainly convolution, pooling and fully-connected. According to the layer types, DNNs many have different names and applications. Some of the most popular DNN types are the Convolutional, Fully Connected, Recurrent and Long Short-Term Memory Neural Networks. In this thesis, only the convolutional neural networks are used for the applications and that is why they are analyzed in more detail in the next section.

7.3.1 Convolutional Neural Networks

Nowadays, Convolutional Neural Networks (CNNs) are one of the most popular DNNs, since they are efficient in many applications. They are named after the type of its hidden layers which are mainly convolution layers based on local connections in contrast with the fully-connected ones. However, they may contain

all other types of layers, such as pooling and fully-connected ones. The fully-connected networks may need a prohibitively large number of parameters for tasks with large number of input data. CNNs are ideal for processing big data due to the reduced number of parameters involved, resulting from the locally connected neurons. Thus, they are preferred over the fully-connected ones. They are also able to successfully capture the spatial and temporal dependencies in a data-set through the application of relevant filters/kernels. These filters learn to identify different features of the processed input (feature extraction). They exploit spatially local correlation by enforcing a sparse local connectivity pattern between neurons of adjacent layers: each neuron is connected to only a small part of the previous layer. A more detailed analysis follows.

7.3.1.1 Convolution Layer

Term convolution is borrowed from the field of Computer Vision. Convolution is the operation which takes two signals, the first one called "input" and the other "kernel" or "filter", and produces an output signal. It takes an input signal and applies a filter to it. The convolution is mathematically expressed as the dot product of the "input" array (f) and the "filter" array (g) as $fg(i) = \sum_{j=1}^{m} g(j) \cdot f(i - j + m/2)$ where i, j are the arrays' indices and m is the stride (explained below). The convolution is easily extended into 2D and 3D matrices usually manipulated by DNNs and expressed in matrix form as $\mathbf{O} = \mathbf{W} \cdot \mathbf{X}$ with $\mathbf{X}$, $\mathbf{O}$ and $\mathbf{W}$ being the input, output and weight matrices with dimensions $w_i \times h_i \times d$, $w_o \times h_o \times d$ and $F \times F \times d$ respectively. Images are represented as 3D matrices with dimensions $h \times w \times d$ which corresponds to the two spatial directions and the color of each pixel. In the case of image processing, the convolution is visualized as a filter/kernel sliding over the spatial directions of the entire image and changing the value of each pixel in the process. Fig. 7.3 shows this process in a 2D matrix. At the beginning, the filter is convoluted with the top left part of the input matrix computing the top left element of the output matrix. Then, it slides by two elements (defined by the stride, see below) to the right and is convoluted again so as to compute the top right element. This process continues through all the rows of the input matrix resulting to the output matrix. If the input processed is a 3D matrix, the filter is also extended through the third direction (depth). Through this process, the filter identifies different features presented in the image helping the DNN predict the task in hand.

During the design of a DNN architecture, the size of each hidden layer is of great interest. Using convolutional layers, this size can be computed and is affected by three variables, the depth of the filter, the stride and the zero-padding applied during the convolution. The depth d controls the number of neurons in a layer that connect to the same region of the previous layer. Stride S controls how the filter slides around the width and height of the input, meaning the elements that it "jumps" after each computation. When the stride is 1, the filter "jumps"

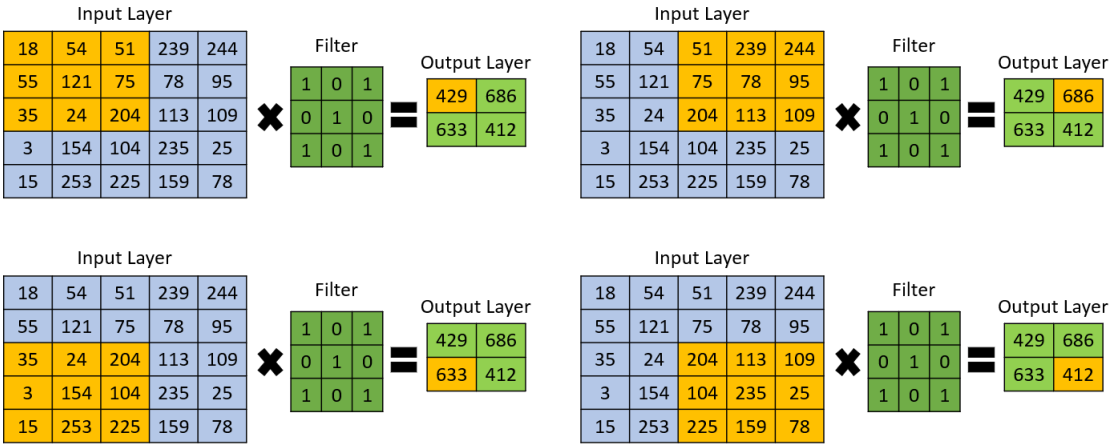


Figure 7.3: 2D representation of the operations/computations performed by a convolutional layer.

one element at a time, leading to overlapping receptive fields and large output dimensions. Similarly, for any stride $S > 0$, the filter moves S elements at a time. When S is increased, the receptive fields overlap less and the output dimensions is decreased. The padding P is a common practice, when the layer's width and height should be preserved as in previous layer's. Before the application of the convolution, the previous layer is padded with zeros on its borders. All these result in a layer with width and height computed as a function of the previous layer's size w_i and h_i, the filters size F, the padding size P and the stride S:

$$w_o = \frac{h_i - F + 2P}{S} + 1 \quad h_o = \frac{h_i - F + 2P}{S} + 1 \tag{7.3.1.1}$$

This function shows that convolution reduces the previous layer's size and compacts the information into the most important one.

In some cases, the inverse of the convolution layer (Transpose Convolution Layer, [136]) is desired in application where the DNN output is of the same dimension as the input. The transpose/inverse operation increases the layer's size and expands/decompresses the information. If the convolution layer changes the layers dimensions from $w_i \times h_i \times d$ to $w_o \times h_o \times d$, then the transpose convolution take a layer with size $w_o \times h_o \times d$ and expands it to $w_i \times h_i \times d$, where $w_o < w_i$ and $h_o < h_i$. This is done by applying the inverse operation of the convolution operator. Transpose layers increase information; an element of the previous layer is expanded into a square area for the next one. The transpose convolutional matrix form is computed as $\mathbf{O} = \mathbf{W}^T \cdot \mathbf{X}$. The dimensions w_o and h_o are computed based on the previous layer's dimensions via the reverse (of the convolution layers)

formula

$$w_o = s(w_i - 1) + F - 2Ph_o = s(h_i - 1) + F - 2P \tag{7.3.1.2}$$

7.3.1.2 Pooling Layer

Pooling is a sample-based discretization process which down-samples the input signal/image, reduces its dimensionality and allows for assumptions to be made about contained features. Intuitively, it expresses the notion that the exact location of a feature is less important than its rough location relative to other features. This is practically done by taking one after another all the regions of the previous layer and applying a specific function which compresses the information. There are three main types of pooling commonly known as max. , min. and average pooling. As the name suggests, max. pooling is based on picking up the maximum value from the selected region from the previous layer, fig. 7.4, the min pooling is based on picking up the minimum value and the average pooling is averaging the values. It is common to periodically insert a pooling layer between successive convolution layers in a CNN architecture. The combination of convolution and pooling layer reduces input's dimensions and they are often referred to as down-sampling layers.

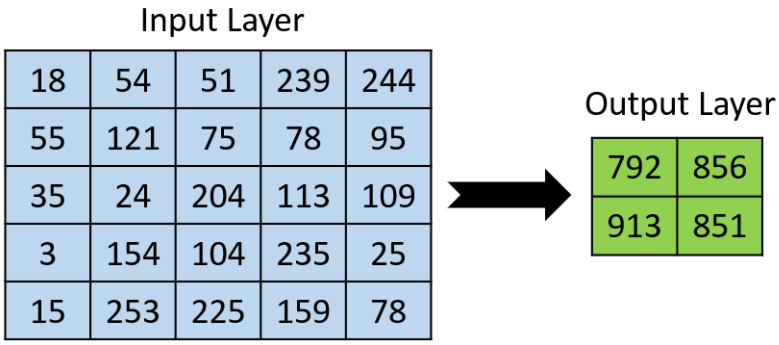


Figure 7.4: 2D representation of the calculations performed by a averaging pooling layer. Each element of the output layer is the sum of all the values of a 3×3 square in the input layer.

7.3.2 Encoding-Decoding CNNs

CNNs are often used for classification purposes, such as for classifying objects shown in images. In these cases, the inputs are labeled with a limited number of classes; there are only a few possible outcomes/outputs. The DNNs copying with classification problems, first, encode the inputs through the down-sampling

layers, which subtract features from the images and compress the important information. Then, fully connected layers follow to connect the subtracted features with the outputs and predict the predefined labels. Even though this method can process dense and spatially correlated inputs, the outputs are restricted to only few labels or classes. For the flow prediction, the CNN architecture should be modified to be able to predict large numbers of outputs, which is not possible with the aforementioned architectures. There are many cases in which input dimensions must be preserved to the output. Numerous image processing tasks from the field of semantic segmentation [120] require the label and feature recognition of the whole input image. These led to the development of a CNN architecture capable of preserving the resolution of the input images to the output image, called herein encoding-decoding architecture or ED-CNN. The architecture is based on the simultaneous usage of encoders and decoders. Encoder is a stack of convolution and pooling layers used to down-sample the input layer, compress its information and recognize its features. An encoder is basically the down-sampling part of a CNN architecture used for classification. Contrary to the encoder, the decoder consists of transpose convolution layers to up-sample the features extracted from the encoder, enhance the compact information and connect it with the output layer. The architecture of ED-CNN is shown in fig. 7.5, where the down-sampling and up-sampling is presented.

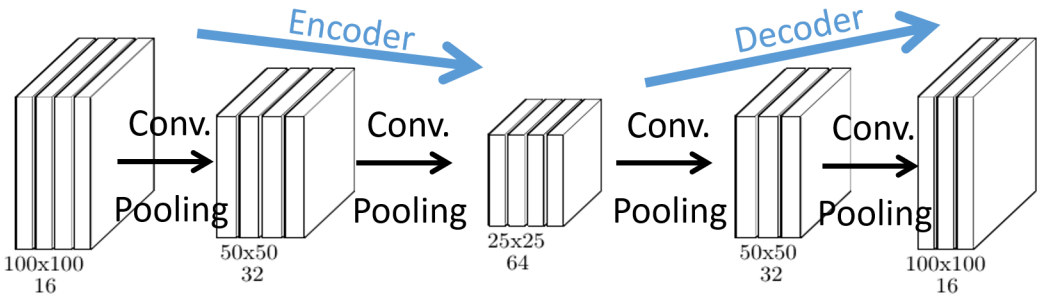


Figure 7.5: Architecture of the ED-CNN. Each box represents a convolution layer.

Recent studies have shown that the ED-CNN is capable of predicting steady state aerodynamic flow fields. In [62], ED-CNNs predict steady velocity fields around 2D and 3D objects. The network inputs are the object geometries and the outputs are the corresponding flow fields. The CFD simulations used to train the DNN has been performed on a CFD solver based on the Lattice Bolzmann Method (LBM), the CFD grid is given and fixed and the ED-CNN predicts the velocity at all the CFD nodes simultaneously. A great number (100000) of training patterns are required to properly train the ED-CNN. Since each pattern corresponds to one CFD run, the required computational cost to collect the database outweights the advantage of the proposed ED-CNN.

A solution to the big database problem required in [62] is proposed in [69]. In [69], the main ED-CNN architecture logic is similar with the one of [62], but with

one key difference. The U-net framework [141] is built and added to the previous ED-CNN. The simple convolutional layers are upgraded into convolutional blocks, gated residual blocks [148]. The resulting new network requires only 5000 training patterns to predict the flow fields around the objects with the same accuracy. Moreover, the study is extended to the prediction of the pressure fields around airfoil geometries. The network prediction ability made it possible to employ it in optimization for searching the optimal airfoil geometry.

In this thesis, this U-net architecture is utilized to predict the flow fields. This architecture is similar with the one presented in the literature and shown in this paragraph. However, variations occur at the overall ED-CNN architecture at it is shown specifically in each application. It starts with an encoder which reduces the inputs size and extracts its features and continues with a decoder which up-samples the extracted features and restores similar dimensionality with the input. The network employs large depth dimension which is a method to allow information to transfer fluently inside the layers. Each encoder's layer halves the width and height of the layer while doubling its depth. The reverse applies to the encoder, each layer doubles the width and height, while the depth is halved. Moreover, skip connections between the layers help transfer the features through the networks architecture. The simple convolution layers are substituted with gated residual blocks [148], for further reduction of the training cost and increment of the predictions accuracy. The gated residual blocks, as proposed in [69], are schematically shown in fig. 7.6. It consists of three convolution layers. The first two pass through a non-linear activation and keep both the input size (w, h, d) unchanged. The third convolution doubles the depth d while it keeps the height and width the same. Then, the output of the convolution is split in half along its depth, yielding two layers of equivalent sizes. The first half remains unchanged, while the second half is fed through a sigmoid gate (activation function). After that, both halves are multiplied together, in an element-wise manner. Finally, the residual connection, which is the main branch, is added to the shortcut connection. The advantage of using such gated residual blocks is that the gating mechanism (the split process) has been shown to improve network performance and convergence speed [167]. The gated residual blocks are also used in [69] to down-sample data, by applying two differences in the block structure. The first convolution layer of the block applies a stride larger than one, i.e. when the kernel slides through the input it moves over more than one neuron. This results in a layer at the first branch's end with smaller size along all three dimensions. The shortcut connection should match this size, thus, the input data in the corresponding branch passes through an average pooling layer with a stride equal with the one used before (in the other branch).

Lastly, in [69], the proposed gated residual blocks are inspired and based on the ResNet architecture [67], fig. 7.7, which are introduced as a solution to the vanishing gradient problems and are widely used for faster training of the net-

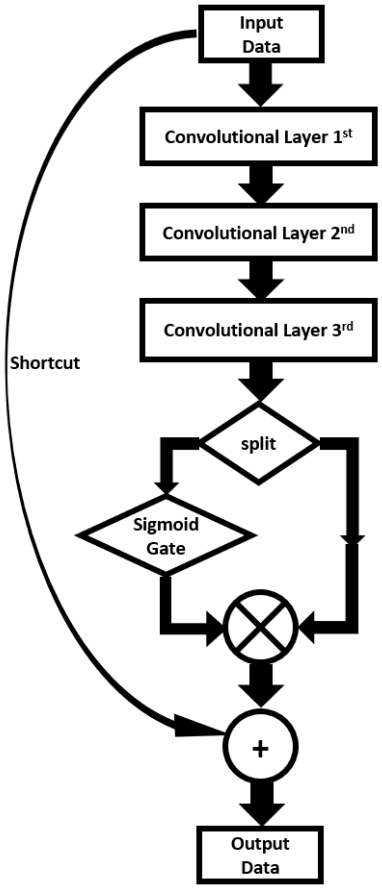


Figure 7.6: Gated residual block.

works. Their key characteristic is the so-called identity shortcut connection. The convectional convolution layers propagate the data along a single route, passing through similar operations. The identity connection bypasses these operations and is added directly to the outputs of the block. These connections are similar to the skip connections of the U-net.

7.4 Applications in Aerodynamic Cases

Industrial designers perform hundreds or thousands of CFD-based analyses during the process of optimizing (according to several criteria) aerodynamic or hydrodynamic shapes. These CFD simulations are usually stored in large databases, for future reference or usage. In general, it would be of great importance for industries to be able to quickly and accurately analyze the flow around new shapes (often similar with the ones already stored in the database). DNNs can be trained on these large databases so as to replicate the CFD code and produce results around new shapes. To this end, in this chapter, DNNs learn how to predict the

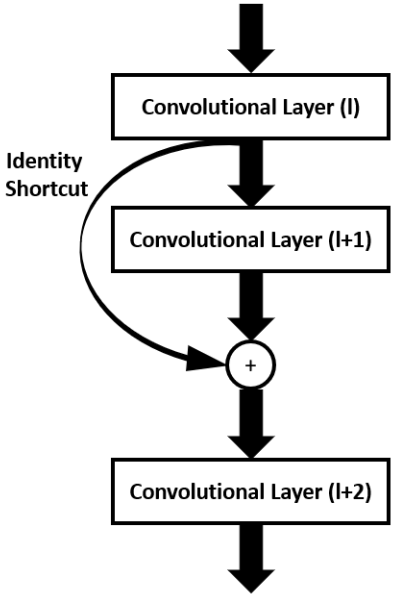


Figure 7.7: ResNet block.

flow field around airfoils and wings. Their training databases are built by CFD results produced by the PUMA software used also in the majority of Benchmark Cases in previous chapter 2.4.

Regarding the ANNs, their architecture is adapted for each application; this is explained on a case-by-case basis. However, some general settings are globally applied to all of them. They are based on the aforementioned theory on DNN and consist mainly of convolution layers. Even though, any other optimizer may be chosen, the Adam optimizer is used and this is proved to be quite fast. Subsets of training patterns are fed into the DNNs during the training, which further increase the convergence speed of the training. For maximum accuracy in the results, the cost function C is the absolute difference between the prediction and the exact output. The RELU activation functions are applied in each convolution layer, but other activation functions do not greatly affect the resulting predictions. The DNN architectures along with the pre- and post-processing of the data in each application has been implemented on Python 2.7 [142] linked with the TensorFlow library [1]. TensorFlow is a symbolic math open-source library used widely for machine learning applications and DNN. It is easily intrinsically parallel and runs on both GPUs and CPUs by taking full advantage of their parallel processing power.

Two different methods are devised to train the DNNs with the available CFD data. The first one is based on image processing and is applied in 2D aerodynamic shapes (such as airfoils). This is done because, as it has been already shown before and in [69], CNNs are powerful at manipulating images and capable of performing similar tasks with flow field predictions (equally complex tasks). The

second method is based on raw data processing and is preferably used in 3D flows. This is done because it is prohibitively expensive to extract and process images from 3D flow fields and essential information may be lost in the process. In the examined cases, input data include shape parameterization data and spatial coordinates of the points at which predictions should be made which are fed into the DNN to predict flow variables at discrete points in the computational domain. This greatly decreases the computational cost and memory requirements. Moreover, it is independent of the CFD grid coordinates, it can predict the flow variables in new grid not previously seen during the training phase.

7.5 Flow Prediction around an Isolated Airfoil

This application is concerned with the prediction of the Mach number field around an isolated airfoil. The flow is inviscid with $M_\infty = 0.15$ and $a_\infty = 0^o$. The airfoil is parameterized using two Bézier curves with 12 control points per airfoil side (separately for the pressure and the suction side). All control points (apart from the first and last ones on each side) are allowed to vary along the y-axis by $\pm 20\%$ of their reference position. Different airfoils are created by randomly changing the control points within the predetermined bounds. The database for training is formed by 50 different airfoils evaluated on the PUMA s/w. An unstructured CFD grid with about $10K$ nodes is generated from scratch for each airfoil; all these grids have different sizes and, of course, different connectivities. The PUMA code takes $\sim 10 secs$ on an NVIDIA K20 GPU to perform one run. Regarding the DNN, the image processing method is used to handle the data. For each airfoil in the database, both its shape and Mach flow field are converted into colored images comprising the input and output set for the training pattern, respectively. Each input image (airfoil shape) has a width of 800 pixels and a height of 800 pixels. Its depth is 3 as many as a colored image (RGB) channels. Overall, each input data has dimensions $800 \times 800 \times 3$. Each output image (Mach flow field) is of dimensions $1024 \times 1024 \times 3$. The role of the DNN is to predict the flow field image (output) given the shape image (input). The DNN used for this application is a CNN based on the gated residual blocks, as proposed in [148, 69]. 33 gated residual blocks are used for down-sampling the input and 22 for up-sampling and connecting with the output, fig. 7.8. Note that each block is a cluster of convolution and pooling layers with a non-linear activation function. This architecture was carefully selected after trying different architectures and checking their prediction abilities and convergence characteristics. Via this trial and error process, the optimal architecture for this application is constructed. For the training phase, a total of 50 training patterns are used. Training costs around $20 hours$ running in parallel on 20 CPUs. The network prediction ability is tested on two new airfoil shapes. These have not been presented to the network during its training. Moreover, they are also evaluated on the PUMA s/w and the

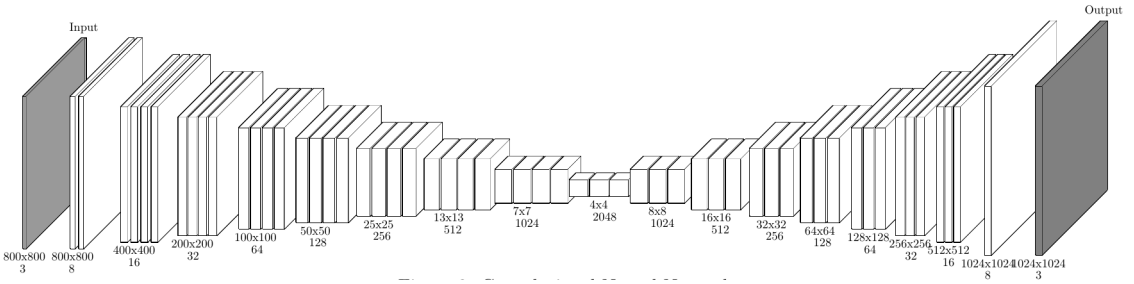


Figure 7.8: Isolated Airfoil: DNN architecture. Each box represents a convolution layer. Each cluster of boxes with the same size represent a gated residual block.

CFD results are compared with the DNN predictions in fig. 7.9. Differences are not visible in the figures due to the high network accuracy. In this application, the relative error is computed as $\epsilon = \frac{1}{N}\sum_{i=0}^{N}\frac{(\hat{p}_i - p_i)^2}{p_i}$ where N is the number of pixels of each image ($N = 800 \times 800 = 1600$), $\hat{p}_i$ is the predicted value of the i^{th} pixel and p_i is the value produced by the CFD analysis. The relative error between predictions and CFD runs is 0.12% for the first airfoil and 0.17% for the second. It is clear that the trained DNN has excellent accuracy and could have been used instead of the CFD analysis, during design/optimization processes.

7.6 Transonic Flow Prediction around an Isolated Wing

The isolated wing of the Benchmark Case 2 in section 2.4 is revisited herein for predicting the surface pressure field. Parameterization, CFD grid and all other setting are the same. The database used for training is comprised of the first offspring population generated while running the EA-based optimization in section 2.4. Basically, they are wings produced by changing the position of the pre-defined parameterization control points picked at random by the EA. From each CFD simulation, only the $100K$ surface nodes are of interest. Regarding the DNN, the raw data method is used to handle the data. The input data are the 24 design/optimization variables and the 3 coordinates of the node to be evaluated; 27 variables in total. In contrast to the previous application, this network predicts the pressure value of each node separately. This means that the trained DNN must be used $100K$ times so as to predict the pressure oat all the surface nodes of a new wing. Note that the coordinates of the node to be predicted must be inside the maximum and minimum coordinated bounds used for the network training. The input is an array with 27 values and the output is a single value. In this case, the input and output data cannot easily be represented by images and this representation would not have physical meaning compared to the previous case. A new DNN architecture, fig. 7.10, is proposed to efficiently handle the small input size. To extract as much information as possible from the input, the DNN, firstly, expands in size with 10 up-sampling convolution layers. Then, it compresses the

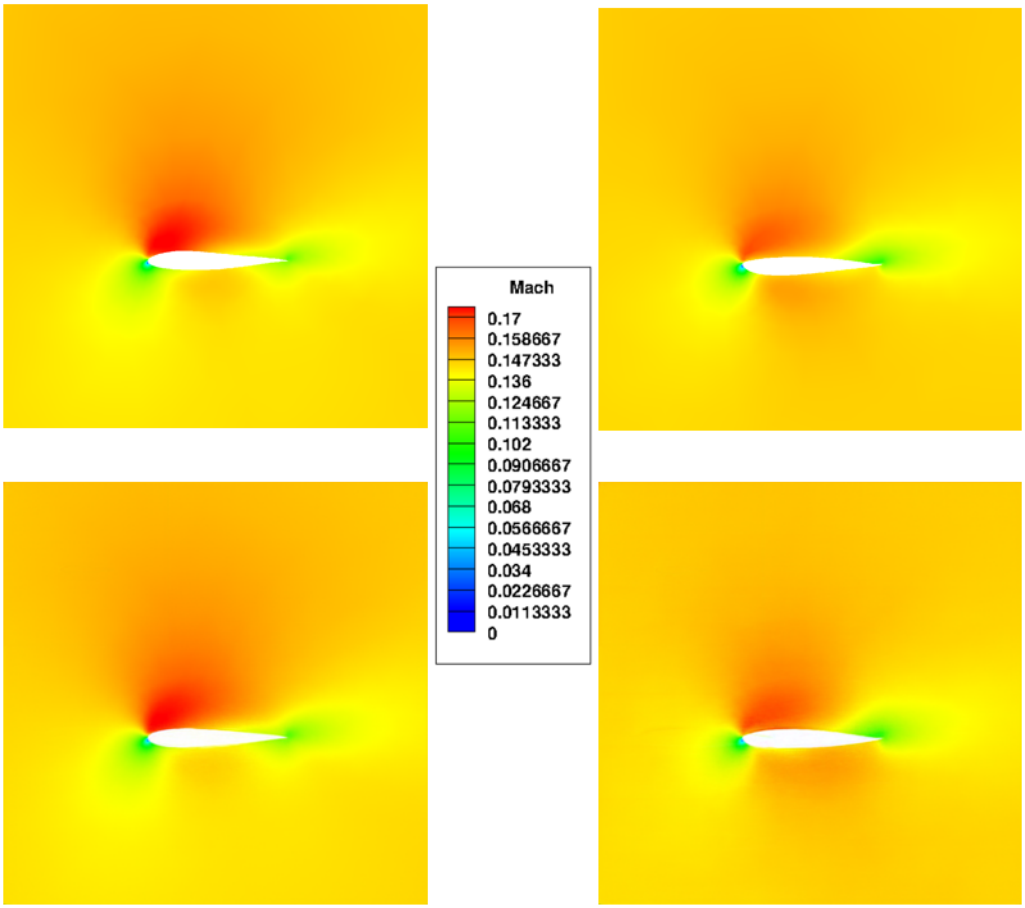


Figure 7.9: Isolated Airfoil: Two new airfoil shapes not included in the training set (the first one on the right column and the second one on the left). Top: CFD analyses. Bottom: DNN predictions.

information with 6 down-sampling transpose convolution layers leading to the single output. This architecture allows to optimally extract all information from the input. In contrast with this DNN, the DNN used in the previous application has reverse architecture (first the down-sampling and then the up-sampling layers), which is the usual practice, and its input/output can easily be represented with images. Herein, the architecture is made so as to handle relatively small number of input/output data with no physical image representation.

For the training phase, only 10 wings are used, each of them with $\sim 100K$ surface nodes resulting in $10\ wings \times 100K\ nodes = 1M$ training patterns in total. In this application, the training is performed on a single NVIDIA K20 GPU and takes $\sim 10hours$. The trained DNN is tested on two new wing shapes, which are also simulated by means of the PUMA software. Figs. 7.11 shows comparisons between the predictions and the CFD simulations for these two wings. Herein, the relative error is computed as $\epsilon = \frac{1}{N}\sum_{i=0}^{N}\frac{(\hat{f}_i - f_i)^2}{f_i}$ where $N = \simeq 100K$ is the number

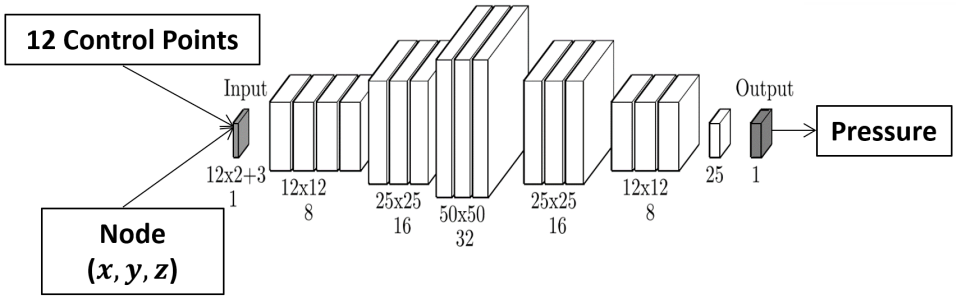


Figure 7.10: Isolated Wing: DNN architecture.

of the surface nodes, $\hat{f}_i$ is the predicted and f_i the CFD computed pressure value on the i^{th} node. Even though, the relative error has been increased compared to the first application (2.3% and 3.57%, in the two studied 3D cases), it is still low enough for the DNN to be reliably used as a metamodel in a EA-based optimization, as shown in the section below.

7.6.1 Optimization Assisted by a DNN

This optimization demonstrates how DNNs can be used within the EA-based optimization to reduce its computational cost. It is concerned with the aforementioned isolated wing and aims at maximizing the lift (L). Two different EA variants using metamodels are compared. The first one is the MAEA as described in section 2.3.5 and the second one is an MAEA with a single off-line metamodel (i.e. the DNN). In the former, personalized on-line trained RBF networks are used and, in the latter, the previously off-line trained DNN is used as metamodel. The DNN predicts the pressure field on the surface of each new wing/individual presented to it and, then, a post-processing tool integrates the pressure to compute the lift force. This procedure requires around $\sim 1sec$ in a personal computer, whereas a CFD evaluation requires $\sim 2min$ on a NVIDIA K20 GPU. Before running the EA with the off-line metamodel (DNN), a database of 50 individuals evaluated on the expensive CFD tool is gathered, the DNN is trained based on a subset of this database and its prediction accuracy is tested, as shown in the previous section. Then, a $(10, 20)$ EA performs the optimization using only the DNN as evaluation tool. The optimal solution is re-evaluated using the exact CFD tool and the error between the predicted and "exact" lift is computed. If the error is smaller than a user-defined threshold or other termination criteria are met, the optimization terminates. Otherwise, the database is enriched with one CFD simulation of the optimal solution found, the DNN is trained again incrementally with newly evaluated individual and, then, a new EA-based optimization starts. The procedure continues till convergence.

The convergence histories of the MAEA with an off-line trained DNN as meta-

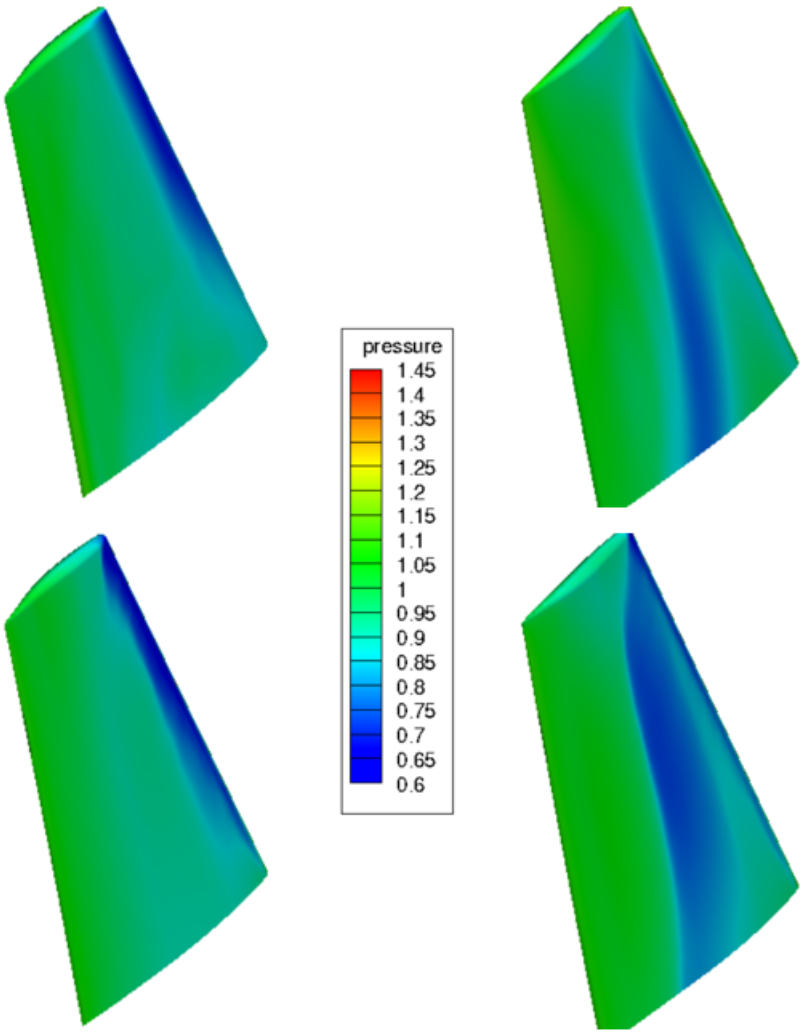


Figure 7.11: Isolated Wing: Two new wing shapes not included in the training set (the first one on the right column and the second one om the left). Top: CFD analyses. Bottom: DNN Predictions.

model and MAEA are compared in fig. 7.12. Notice that the first 50 evaluations for the EA with off-line metamodels are used to build the initial database. The off-line usage of metamodels improves the performance of EAs in this case. This happens due to the high prediction accuracy achieved by the DNN. A comparison between the predicted and "exact" pressure fields on the surface of the optimal wing is shown in fig. 7.13. It is noted that the DNN was adequately trained during the optimization cycles and the average error on the optimal wing has dropped to 0.7%.

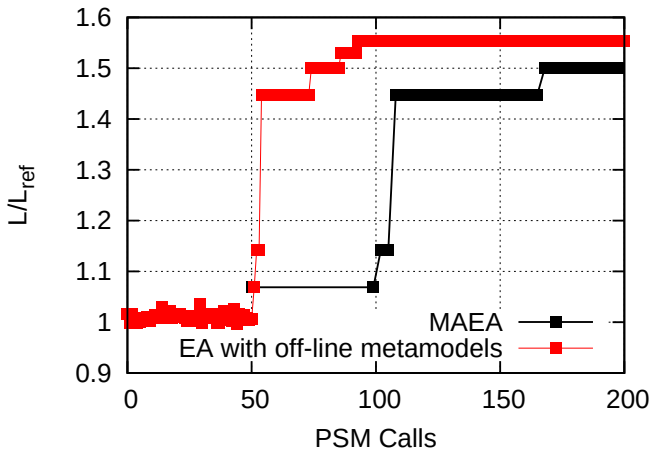


Figure 7.12: Isolated Wing: Convergence histories of the MAEA (based on on-line RBF networks, as in all previous chapters of this thesis) and the MAEA with off-line DNN.

7.7 Transonic Flow Prediction around an Aircraft Wing-Body Configuration

The first industrial problem presented in section 6 concerned with an aircraft wing-body configuration is revisited herein to predict the pressure field on its surface. The same setup is used here with 8 design variables controlling the wing shape. Training patterns are selected from the previously formed EA database with individuals evaluated on the CFD code; 80 different wings are used in total. The aforementioned raw data method is used to handle the data and the DNN predicts the pressure at each wing surface node separately. The input data are the 8 design variables plus the 3 coordinates of the node position to be evaluated; 11 variables in total. Each wing gives $34K$ training patterns, as much as the nodes on its surface. In contrast with previous applications, the DNN architecture is based on the ResNet architecture, fig. 7.14, which provides a more compact architecture with only 3 fully-connected layers greatly reducing the computational cost of the training phase. The same architecture performs poorly in the previous application. On the other hand, many training patterns are required to accurately train this minimalistic DNN. Once again, only the surface nodes are predicted due to limitations in the computational budget. The training required $\sim 5hours$ on one NVIDIA K20 GPU. A new aircraft wing-body configuration is used to validate the DNN prediction accuracy. Figs. 7.11 shows a comparison between the prediction and the CFD simulation on this configuration. The relative error computed similarly with the previous application remains relatively small (3.18%), even though the network's architecture has been significantly decreased. Note that by training the DNN with only 10 different wings instead of 80, the prediction accuracy of the

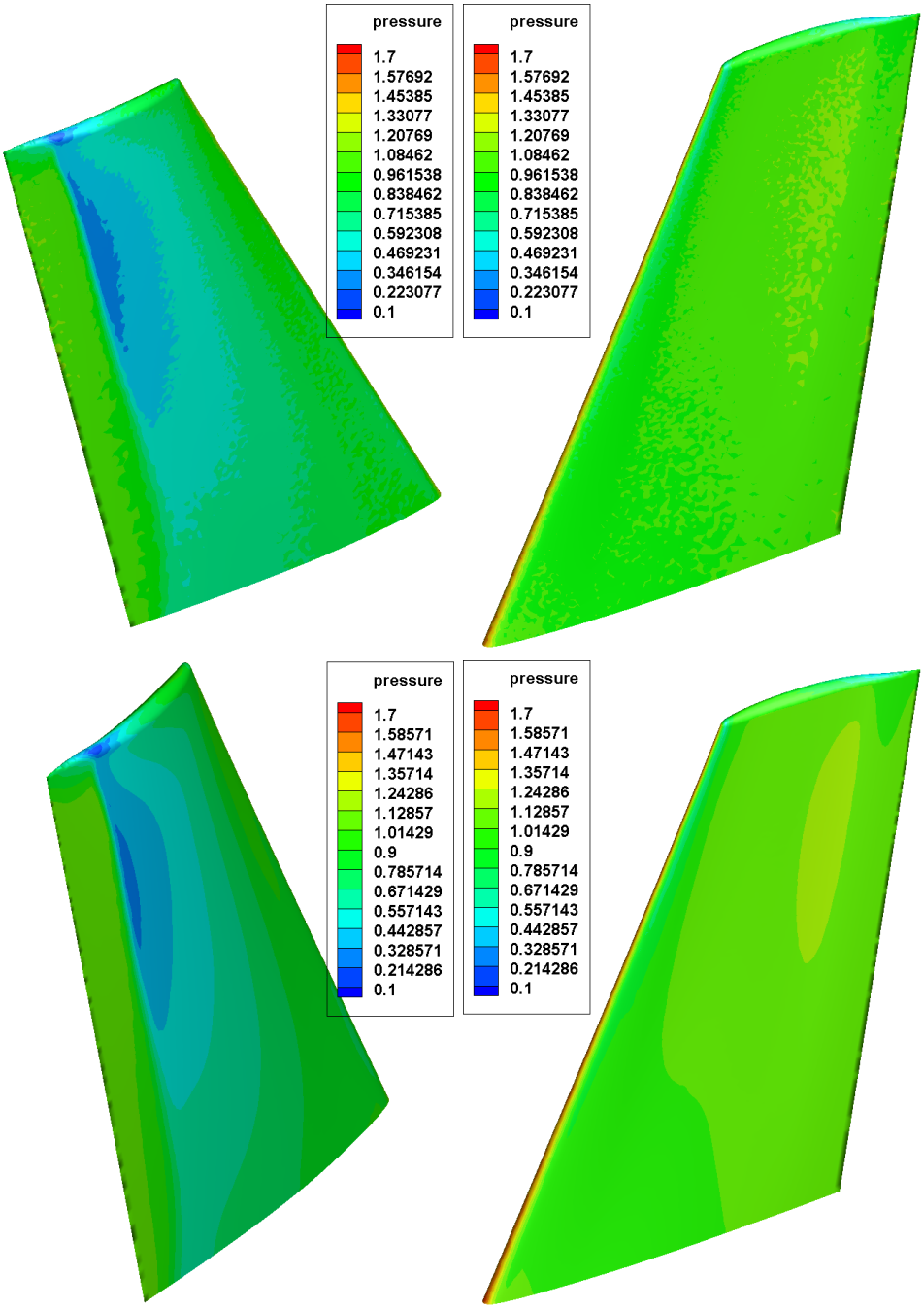


Figure 7.13: Isolated Wing: Predicted and exact pressure field on the surface of the optimal wing

DNN would be worse. DNN prediction of a new wing-body configuration, fig. 7.16, in the case of less training patterns, has an error of $\sim 10\%$ in comparison with the CFD simulation.

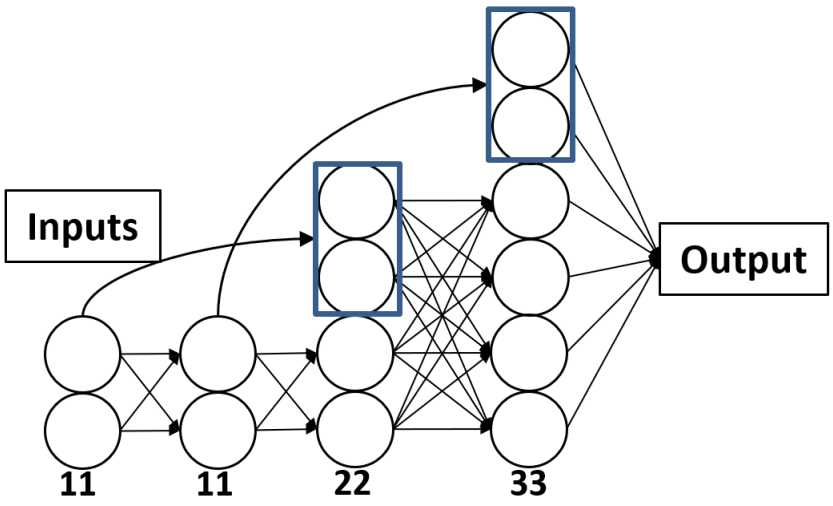


Figure 7.14: Aircraft Wing-Body Configuration: DNN architecture. Each circle represents a neuron and arrows represent intermediate connections. The actual number of neurons is written below each layer.

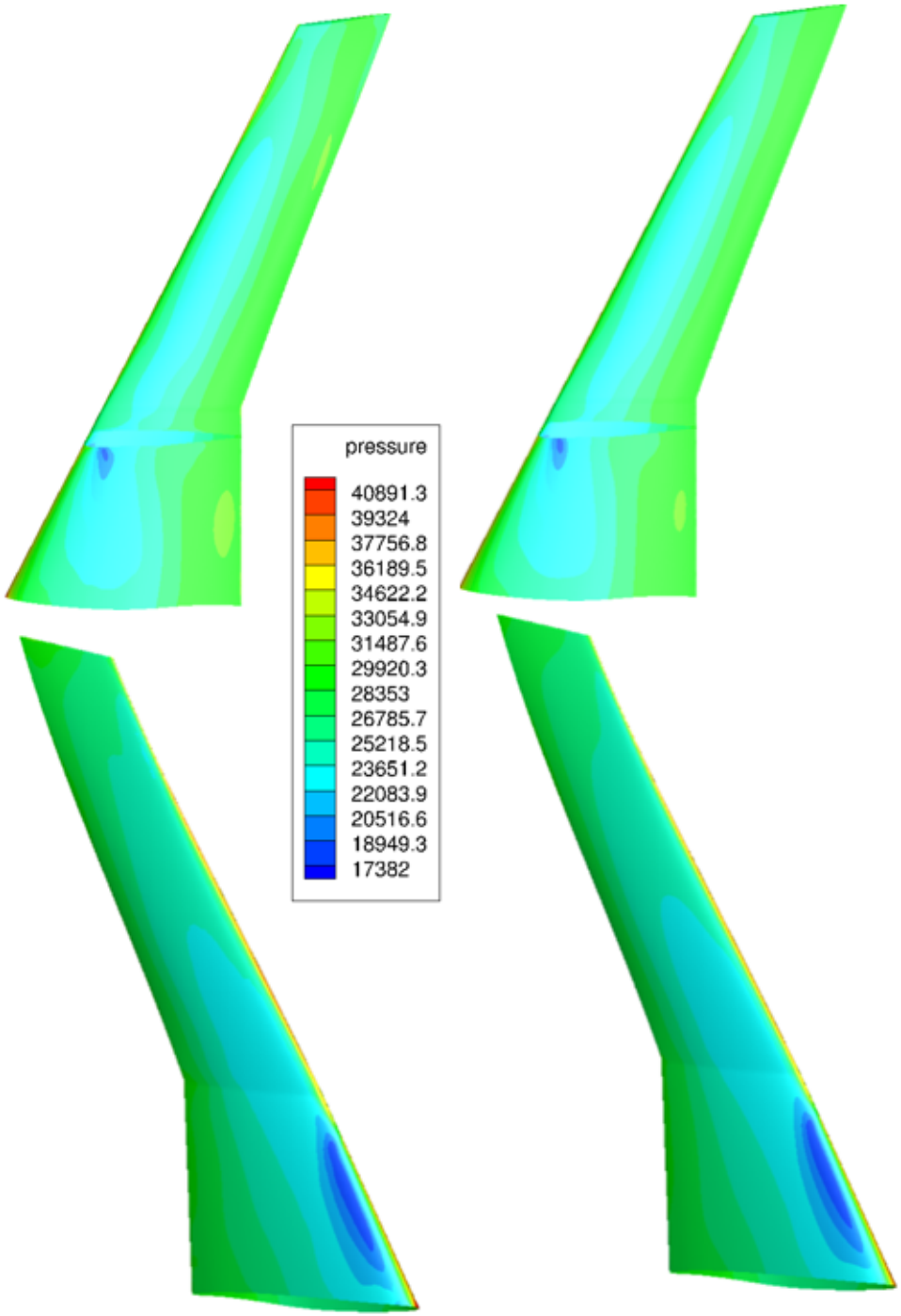


Figure 7.15: Aircraft Wing-Body Configuration: A new wing-body configuration not included in the 80 training patterns. Left: CFD analyses. Right: DNN predictions.

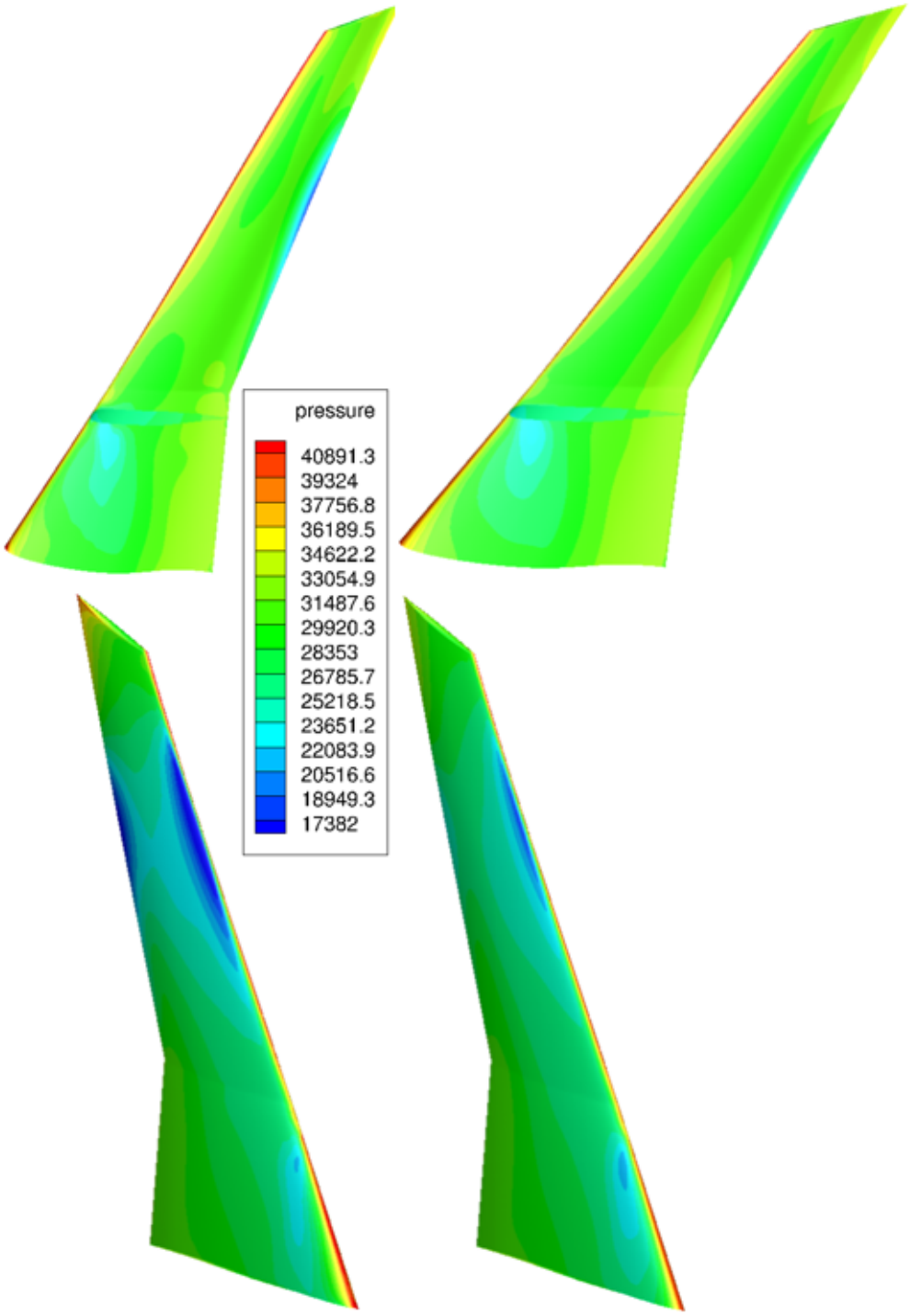


Figure 7.16: Aircraft Wing-Body Configuration: A new wing-body configuration not included in the 10 training patterns. Left: CFD analyses. Right: DNN predictions.

Chapter 8

Conclusions

This PhD thesis aimed at developing, applying and demonstrating methods capable of making Evolutionary Algorithms as efficient as possible. The result is an enhanced EA with reasonable computing cost, which cab be used to solve any optimization problem, even large-scale ones in the field of CFD. Throughout this PhD thesis, all the proposed methods have been embedded into the EASY platform used to perform all the optimizations. The advantages of the methods assisting the EAs have been demonstrated in a wide range of problems, from low-cost mathematical minimization problems up to 2D or 3D aerodynamic shape optimizations. In this thesis these are the shape optimization of an Aircraft Wing-Body Configuration, the DrivAer Car, an Ultra-light Aircraft, a Francis Runner and a Valveless Diaphragm Micropump. Of course, the fact that all applications presented in this thesis are in the field of aerodynamics/fluid mechanics is not restrictive at all; the CFD evaluation software could easily be replaced by any other software in computational mechanics, making thus the proposed methods applicable to any other application domain. A great part of this thesis is based on the use of low-cost metamodels (on-line trained personalized surrogate models, such as RBF networks or other regression models), assisted by Principal Component Analysis technique. On the other hand, off-line trained metamodels, in the form of Deep Neural Networks are also used to predict flow fields around/inside aerodynamic bodies. After their training procedure, they can be used for design-optimization of aerodynamic bodies instead of the expensive CFD software.

The contribution of the PhD thesis can be summarized in the three innovative methods presented below:

(a) The Principal Component Analysis (PCA) assists EAs so as to increase their performance in problems with many design variables. In this thesis, the Kernel variant of PCA, applied to the offspring population in each generation, transforms each individual from the design space to a new (feature) space with directions (the so-called Principal Directions) in decreasing order of importance. Through this transformation, the design variables are redefined as these directions in the

feature space. Then, the evolution operators are applied to the new design variables/space, in which they perform "optimally" and, thus, the EA has a more efficient search mechanism able to deal with initially non-separable, "ill-posed" and high dimensional problems. In addition, PCA can alleviate the "curse of dimensionality" of metamodels. This problem is related to the fact that, while the number of design variables/ input units to the metamodels is increased, their prediction accuracy is decreased and their training requires more patterns, which is additional cost for the EAs (since the start of the Low-Cost Pre-Evaluation phase is delayed), and more computational cost. With the help of the Kernel PCA, the metamodels are trained with input units that correspond to the most important variables (the principal directions with the highest variance) of the feature space, by appropriately truncating the less significant ones. The number of input units a metamodel "sees" becomes lower, which is always beneficiary to improve the metamodel prediction ability and limits their training requirements. This research ended with a conference paper presentation [89] and a journal [85] paper publication.

(b) A hybrid optimization method for Multi-Objective Optimization problems is assessed. It combines the already efficient PCA-driven MAEA for the exploration of the design space and a Gradient-Based (GB) optimization method for the exploitation of the space and combines the advantages of both method. For the GB method, the gradients of the objective functions w.r.t. the design variables are computed in an efficient way using the (continuous) adjoint method, which computational cost is independent from the number of design variables. The key feature of the proposed method is the definition of the descending direction or the so-called Pareto Advancement Direction (PAD) along which the GB method updates a few selected individuals in each generation. The Linear PCA method, applied to the elite set of the previous generation, identifies the important directions of the objective space. From these, the one with smallest variance (most insignificant for the PCA) is always "perpendicular" to the current front of non-dominated solutions and points in the direction all objectives are reduced. This is the direction aligned with the PAD; in each generation, the GB method refines a few selected individuals along the PAD. In the presented applications, where all objectives are computed by a single CFD run, one call to the adjoint solver suffices for the computation of the gradients of each individual to be updated by the GB. Judging from the results, this hybrid method can efficiently approximate the Pareto front faster than a MAEA or the hybrid method presented by the same group of authors in [106, 78]. Note that this method has been validated on CFD-based problems but it is not restricted only to these problems. It can also be used in any optimization problem governed by different set of PDEs, provided that the numerical solver and its adjoint are available. The hybrid method applied in aerodynamic optimization problems ended in a journal [90] paper publication.

(c) Multi Criteria Decision Making (MCDM) techniques are introduced into EAs

or MAEAs solving MOO problems. The "a priori" articulated Decision Maker (DM) preferences (known before the beginning of the optimization) help the optimization to focus on specific parts of the Pareto front, thus, driving the EA towards preferred areas in the objective space. Herein, the DM's preferences are processed by the known MCDM technique, namely TOPSIS; however, this could be replaced by any other similar method. TOPSIS relies upon weights expressing the DM preferences for each objective function. Based on these, in each generation, distances in the objective space are used to rank the given solutions (offspring population) and the method assigns them a scalar utility value corresponding to this rank. These values affect the parent selection, elitism and non-dominated front trimming operators and the front is driven towards areas preferred by the DM. Finally, the front of non-dominated solutions provides better results in the corresponding areas than the EA or MAEAs computed without implementing MCDM techniques . Note that TOPSIS can work synergistically with all the other methods used to improve the performance of EAs or MAEAs. This EA variant applied in industrial aerodynamic optimization problems has been demonstrated in a journal [85] paper publication.

(d) Deep Neural Networks predict flow fields around aerodynamic bodies and can be used as a quick evaluation tool for shape design or optimization. DNNs are trained on databases of CFD analysis on different aerodynamic shapes, such as airfoils and wings, and learn to predict the pressure or Mach number distribution around/inside/on new shapes, not previously seen by the network. The networks inputs contain information about the shape and the output is the sought flow field. For the data handling and representation of the problem, two scenarios are adopted. The first one, applied in 2D cases, is based and processes images, because the Convolutional Neural Networks (CNNs), used herein, perform optimally in image processing tasks. The input image is the shape of the airfoil to be predicted, while the output is the flow field (Mach number around the airfoil) as a colored image. The second scenario is based on raw data, because it is extremely computationally expensive to represent 3D flow fields with 2D images. A node-based approach is used for the 3D cases, where the DNN predicts the pressure at each CFD node, separately. The networks inputs are the parameterization or some description of the shape along with the (x,y,z) coordinates of the node to be predicted and the output is its pressure. Both scenarios are implemented with different network architectures. The architecture is a crucial part of the network and is selected carefully to achieve the satisfactorily predictions. It is based on all the state-of-the-art networks, such as CNN and ResNet. Finally, DNNs provide acceptable predictions on both 2D and 3D cases, proving their ability to assist CFD analysis.

Industrial and other applications/demonstrations of the material presented in this PhD thesis can be found in the conference paper presentations [87, 86], a journal [88] paper publication and VKI Lectures on Optimization [50, 51].

8.1 Future Work

Future research topics that could be considered as a follow up of the work presented in this PhD thesis are:

- The investigation of other non-linear activation functions for the Kernel PCA and their effect on the convergence of EAs.
- The integration of the proposed methods into the Asynchronous EA for better parallel efficiency.
- The migration of the proposed ideas/methods in other stochastic population methods, such as the PSO [101] and the pity beetle algorithm [76].
- The investigation of methods for constraint treatment involving metamodels and classification techniques.
- The use of EAs with adaptive populations size would be of interest. The population could be varied in order to keep quantities such as population diversity constant.
- The investigation of other MCDM techniques for driving the EAs evolution or other implementation methods, such as interactive communication of EAs and DM.
- The extension and/or generalization of the DNNs so as to be able to predict accurately a wide range of shapes and flow conditions.

Bibliography

[1] M. Abadi, A. Agarwal, and P. Barham. TensorFlow: Large-scale machine learning on heterogeneous systems, 2015. Software available from tensorflow.org.

[2] H. Abdi and L. Williams. Principal component analysis. *Wiley Interdisciplinary Reviews: Computational Statistics*, 2(4):433–459, 2010.

[3] D. Ackley. *A connectionist machine for genetic hillclimbing*. Kluwer Academic Publishers, Boston MA, 1987.

[4] J. Amaya, C. Cotta, and A. Fernandez-Leiva. Memetic and hybrid evolutionary algorithms. pages 1047–1060, 01 2015.

[5] V.G. Asouti. *Aerodynamic analysis and design methods at high and low speed flows, on multiprocessor platforms*. PhD thesis, National Technical University of Athens, 2009.

[6] 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.

[7] V.G. Asouti and K.C. Giannakoglou. A low-cost evolutionary algorithm for the unit commitment problem considering probabilistic outages. *International Journal of Systems Science (SI: Computational Intelligence in the Presence of Uncertainties)*, 43(7):1322–1335, 2012.

[8] 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*, 10(3):373–389, 2009.

[9] 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.

[10] M. Baldwin. A new factor in evolution. *The American Naturalist*, 30(354):441–451, 1896.

[11] M. Biancolini, E. Costa, U. Cella, C. Groth, G. Veble, and M. Andrejasic. Glider fuselage-wing junction optimization using CFD and RBF mesh morphing. *Aircraft engineering and aerospace technology*, 88:740–752, 01 2016.

[12] G. Box. Evolutionary Operation: A method for increasing industrial productivity. *Applied Statistics*, 6(2):81–101, 1957.

[13] G. Box and N. Draper. *Evolutionary Operation: A Statistical Method for Process Improvement*. Wiley, New York, 1969.

[14] H. Bremermann. Optimization through evolution and recombination. *Self-Organizing Systems*, pages 93–106, 1962.

[15] O. Brodersen. Drag prediction of engine–airframe interference effects using unstructured Navier–Stokes calculations. *Journal of Aircraft*, 39(6):927–935, 2002.

[16] S. Chakradeo, A. Hendre, and S. Deshpande. Generalized theory for hybridization of evolutionary algorithms. In *IEEE International Conference on Computational Intelligence and Computing Research, Coimbatore, India*, December 2014.

[17] R. Cheng, H. Cheng, Y. Jin, and X. Yao. Model-based evolutionary algorithms: a short survey. *Complex & Intelligent Systems*, 08 2018.

[18] H. Chun-Wei, H. Song-Bin, and L. Gwo-Bin. A microfluidic device for precise pipetting. *Journal of Micromechanics and Microengineering*, 18(3):35–39, 2008.

[19] C.A. Coello Coello. An updated survey of evolutionary multiobjective optimization techniques: State of the art and future trends. *Progress on Evolutionary Computation*, 1999.

[20] C.A. Coello Coello, D.A. Van Veldhuizen, and G.B. Lamont. *Evolutionary algorithms for solving multi-objective problems*. Kluwer Academic Publisher, 2002.

[21] D.R. Cox. *Interaction. International Statistical Review*. Wiley, 1984.

[22] N. Cramer. A representation for the adaptive generation of simple sequential programs. In *International Conference on Genetic Algorithms and the Applications*, pages 183–187, 1987.

[23] C. Darwin. *On the origin of species by means of natural selection*. 1859.

[24] K. Deb. An efficient constraint handling method for genetic algorithms. *Computer Methods in Applied Mechanics and Engineering*, 186(2):311–338, 2000.

[25] K. Deb, S. Agarwal, A. Pratap, and T. Meyarivan. A fast elitist non-dominated sorting genetic algorithm for multi-objective optimization: NSGA-II. *Technical Report 200001, Indian Institute of Technology Kanpur*, 2000.

[26] K. Deb and R.B. Agrawal. Simulated binary crossover for continuous search space. *Technical report, IITK/ME/SMD-94027*, 1994.

[27] K. Deb, A. Pratap, S. Agrawal, and T. Meyarivan. A fast and elitist multi–objective genetic algorithm NSGA–II. *IEEE Transactions on Evolutionary Computation*, 6(2):182–197, 2002.

[28] R. Dechter. Learning while searching in constraint-satisfaction-problems. In *AAAI'86 Proceedings of the Fifth AAAI National Conference on Artificial Intelligence*, pages 178–185, 1 1986.

[29] L. Deng and D. Yu. Deep learning: Methods and applications. *Foundations and Trends in Signal Processing*, 7(3-4):1–199, 2014.

[30] J. 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*, Barcelonn, Spain, 2003.

[31] J. Desideri, J. Periaux, and Z. Tang. Multi-objective design strategies using deterministic optimization with different parameterizations in aerodynamics. In *ECCOMAS 2004, Jyvaskyla, Finland*, July 2004.

[32] 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.

[33] P. Fishburn. Additive utilities with incomplete product set: Applications to priorities and assignments. *Operations Research Society of America (ORSA), Baltimore, USA*, 1967.

[34] I. Fodor. A survey of dimension reduction techniques. *Technical Report, UCRL ID-148494*, 9:1–18, 2002.

[35] L. Fogel. Autonomous automata. *Industrial Research Magazine*, 4(2):14–19, 1962.

[36] L. Fogel. *On The Organization of Intellect*. PhD thesis, University of California, 1964.

[37] L. Fogel, P. Angeline, and D. Fogel. An evolutionary programming approach to self-adaptation on finite state machines. In *Proceedings of the fourth annual conference on evolutionary programming*, pages 355–365, 1995.

[38] L. Fogel, A. Owens, and M. Walsh. Artificial Intelligence through Simulated Evolution. In *John Wiley & Sons*, New York, 1966.

[39] R. Friedberg. A learning machine: Part i. *IBM J. Res. Dev.*, 2(1):2–13, 1958.

[40] R. Friedberg, B. Dunham, , and J. North. A learning machine: Part II. *IBM J. Res. Dev.*, 3(3):282–287, 1959.

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

[42] B. Gao and L. Pavel. On the properties of the softmax function with application in game theory and reinforcement learning. *Arxiv Preprint:1704.00805*, 04 2017.

[43] 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.

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

[45] C.A. Georgopoulou and K.C. Giannakoglou. Two-level, two-objective evolutionary algorithms for solving unit commitment problems. *Applied Energy*, 86(7-8):1229–1293, 2009.

[46] C.A Georgopoulou and K.C. Giannakoglou. Metamodel-assisted evolutionary algorithms for the unit commitment problem with probabilistic outages. *Applied Energy*, 87(5):1782–1792, 2010.

[47] H.A. Georgopoulou. *Optimization techniques for committing combined cycle power plants and designing their components*. PhD thesis, National Technical University of Athens, 2009.

[48] K.C. Giannakoglou. Design of optimal aerodynamic shapes using stochastic optimization methods and computational intelligence. *Progress in Aerospace Sciences*, 38(1):43–76, 2002.

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

[50] K.C. Giannakoglou, V.G. Asouti, D.H. Kapsoulis, and K.T. Tsiakas. Low cost evolutionary algorithms for engineering applications. In *Introduction to Optimization and Multidisciplinary Design*, Lecture Series. von Karman Institute, Rhode–St–Genése, Belgium, May 2016.

[51] K.C. Giannakoglou, V.G. Asouti, D.H. Kapsoulis, K.T. Tsiakas, X.S. Trompoukis, and F. Gagliardi. Low cost evolutionary algorithms for engineering applications. In *Introduction to Optimization and Multidisciplinary Design*, Lecture Series. von Karman Institute, Rhode–St–Genése, Belgium, September 2018.

[52] 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.

[53] 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*, 9:389–412, 2001.

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

[55] K.C. Giannakoglou, D.I. Papadimitriou, and I.C. Kampolis. Aerodynamic shape design using evolutionary algorithms and new gradient-assisted metamodels. *Computer Methods in Applied Mechanics and Engineering*, 195(44-47):6312–6329, 2006.

[56] M. Giles and N. Pierce. Improved lift and drag estimates using adjoint Euler equations. In *AIAA Paper 1999-3293, 14th Computational Fluid Dynamics Conference, Norfolk, U.S.A*, 1999.

[57] A.P. Giotis. *Application of evolutionary algorithms, computational intelligence and advanced computational fluid dynamics techniques to the optimization-inverse design of turbomachinery cascades, using parallel processing*. PhD thesis, National Technical University of Athens, 2003.

[58] A.P. Giotis. *Application of evolutionary algorithms, computational intelligence and advanced computational fluid dynamics techniques to the optimization-inverse design of turbomachinery cascades, using parallel processing*. PhD thesis, National Technical University of Athens, 2003.

[59] A.P. Giotis and K.C. Giannakoglou. Low-cost genetic optimization based on inexact pre-evaluations and the sensitivity analysis of design parameters. *Advances in Engineering Software*, 29(2):129–138, 1998.

[60] D.E. Goldberg. *Genetic Algorithm in Search, Optimization and Machine Learning*. Massachusett, Addison-Wesley, 1989.

[61] D.E. Goldberg and K. Deb. A comparative analysis of selection schemes used in genetic algorithms. *FGA1*, pages 69–93, 1991.

[62] X. Guo, W. Li, and F. Iorio. Convolutional neural networks for steady flow approximation. In *22Nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 481–490, New York, USA, 2016.

[63] J. Hadamard. Sur les problemes aux derivees partielles et leur signification physique. *Princeton University Bulletin*, pages 49–52, 1902.

[64] N. Hansen, S. Muller, and P. Koumoutsakos. Reducing the time complexity of the derandomized evolution strategy with covariance matrix adaptation (CMA-ES). *Evolutionary Computation*, 11(1):1–18, 2003.

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

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

[67] K. He, X. Zhang, S. Ren, and J. Sun. Deep residual learning for image recognition. In *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 770–778, June 2016.

[68] A. Heft, T. Indinger, and N. Adams. Experimental and numerical investigation of the DrivAer model. In *ASME 2012, Symposium on Issues and Perspectives in Automotive Flows*, pages 41–51, Puerto Rico, USA, July 8-12 2012.

[69] O. Hennigh. Automated design using neural networks and gradient descent. *ArXiv e-prints, arXiv:1710.10352*, 10 2017.

[70] J. Holland. Outline for a logical theory of adaptive systems. *Journal ACM*, 9(3):297–314, 1962.

[71] J. Holland. *Adaptation In Natural and Artificial Systems*. The University of Michigan Press, 1975.

[72] C. Hwang and K. Yoon. *Multiple Attribute Decision Making: Methods and Applications*. Springer-Verlag, New York, 1981.

[73] Y. Jin, M. Olhofer, and B. Sendhoff. A framework for evolutionary optimization with approximate fitness functions. *IEEE Transactions on Evolutionary Computation*, 6(5):481–494, 2002.

[74] I. Jolliffe. *Principal Component Analysis, 2nd Edition*. Springer Series in Statistics, 2002.

[75] S. Kabanikhin, N. Tikhonov, V. Ivanov, and M. Lavrentiev. Definitions and examples of inverse and ill-posed problems. *Journal of Inverse and Ill-posed Problems*, 16:317–357, 01 2008.

[76] N. Kallioras, N. Lagaros, and D. Avtzis. Pity beetle algorithm - a new metaheuristic inspired by the behavior of bark beetles. *Advances in Engineering Software*, 121:147–166, 2018.

[77] I.C. Kampolis. *Parallel, multilevel algorithms for the aerodynamic optimization in turbomachines*. PhD thesis, National Technical University of Athens, 2009.

[78] 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.

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

[80] 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.

[81] I.C. Kampolis, E.I. Karangelos, and K.C. Giannakoglou. Gradient-assisted radial basis function networks: theory and applications. *Applied Mathematical Modelling*, 28(13):197–209, 2004.

[82] I.C. Kampolis, D.I. Papadimitriou, and K.C. Giannakoglou. Evolutionary optimization using a new radial basis function network and the adjoint formulation. *Inverse Problems in Science and Engineering*, 14(4):397–410, 2006.

[83] D. Kapsoulis, K. Samouchos, X. Trompoukis, and K. Giannakoglou. Design-optimization of a valveless diaphragm micropump under uncertainties using evolutionary algorithms. In *ADMOS 2019*, Alicante, Spain, May 2019.

[84] D. Kapsoulis, K. Samouchos, X. Trompoukis, and K. Giannakoglou. Hybrid optimization of a valveless diaphragm micropump using the cut-cell method. *Journal of Mechanics Engineering and Automation*, to be published, 2019.

[85] D. Kapsoulis, K. Tsiakas, X. Trompoukis, V. Asouti, and K. Giannakoglou. Evolutionary multi-objective optimization assisted by metamodels, Kernel PCA and multi-criteria decision making techniques with applications in aerodynamics. *Applied Soft Computing*, 64(2):182–197, 2017.

[86] D.H. Kapsoulis, V.G. Asouti, K.C. Giannakoglou, S. Porziani, E. Costa, C. Groth, U. Cella, and M.E. Biancolini. Evolutionary aerodynamic shape optimization through the RBF4AERO platform. In *ECCOMAS Congress 2016, VII European Congress on Computational Methods in Applied Sciences and Engineering*, Crete island, Greece, June 5-10 2016.

[87] D.H. Kapsoulis, V.G. Asouti, E.M. Papoutsis-Kiachagias, K.C. Giannakoglou, S. Porziani, E. Costa, C. Groth, U. Cella, M.E. Biancolini, M. Andrejavsivc, D. Ervzen, M. Bernaschi, A. Sabellico, and G. Urso. Aircraft & car shape optimization on the RBF4AERO platform. In *11th HSTAM International Congress on Mechanics*, Athens, Greece, May 27-30 2016.

[88] D.H. Kapsoulis, K. Samouchos, X.S. Trompoukis, and K.C. Giannakoglou. Optimization under uncertainties of a valveless diaphragm pump using the cut-cell method. *The International Journal of Engineering and Science*, 8(8):7–14, 2019.

[89] D.H. Kapsoulis, K.T. Tsiakas, V.G. Asouti, and K.C. Giannakoglou. The use of kernel pca in evolutionary optimization for computationally demanding engineering applications. In *2016 IEEE Symposium Series on Computational Intelligence (IEEE SSCI 2016)*, Athens, Greece, December 2016.

[90] D.H. Kapsoulis, K.T. Tsiakas, X.S. Trompoukis, V.G. Asouti, and K.C. Giannakoglou. PCA-assisted hybrid algorithm combining EAs and adjoint methods for CFD-based optimization. *Applied Soft Computing*, 73:520–529, 2018.

[91] M.K. Karakasis. *Hierarchical, distributed evolutionary algorithms and computational intelligence in aerodynamic shape optimization, on multiprocessing systems*. PhD thesis, National Technical University of Athens, 2006.

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

[93] 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.

[94] 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.

[95] 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.

[96] I.S. Kavvadias. *Aerodynamic optimization for turbulent flows using adjoint methods and GPUs (in progress)*. PhD thesis, National Technical University of Athens.

[97] I.S Kavvadias, E.M. Papoutsis-Kiachagias, and K.C. Giannakoglou. On the proper treatment of grid sensitivities in continuous adjoint methods for shape optimization. *Journal of Computational Physics*, 301:1–18, 2015.

[98] S. Kazarlis and V. Petridis. Varying fitness functions in genetic algorithms: Studying the rate of increase of the dynamic penalty terms. In *Parallel Problem Solving from Nature*, pages 211–220, Berlin, Heidelberg, 1998. Springer Berlin Heidelberg.

[99] A. Keane and B. Nair. Computational approaches to aerospace design. In *John Wiley & Sons*, Chichester UK, 2nd edition, 205.

[100] G. Kendall, E. Soubeiga, and P. Cowling. Choice function and random hyperheuristics. *Proceedings of the Fourth Asia-Pacific Conference on Simulated Evolution and Learning*, pages 667–671, 2002.

[101] J. Kennedy and R. Eberhart. Particle swarm optimization. In *Proceedings of IEEE International Conference on Neural Networks IV*, pages 1942–1948, 1995.

[102] J. F. Kenney and E. S. Keeping. *Linear Regression and Correlation, Ch. 15 in Mathematics of Statistics, Pt. 1*. 3 edition, 1962.

[103] D. Kingma and J. Ba. Adam: A method for stochastic optimization. In *3rd International Conference for Learning Representations*, San Diego, 2015.

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

[105] E.A. Kontoleontos. *Designing Thermo-Fluid Systems using Gradient-based Optimization Methods and Evolutionary Algorithms*. PhD thesis, National Technical University of Athens, 2012.

[106] 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.

[107] J. Koza. Genetic programming ii: Automatic discovery of reusable programms. *MIT Press*, 1994.

[108] N. Krasnogor. Coevolution of genes and memes in memetic algorithms. *Graduate Student Workshop*, page 371, 2000.

[109] N. Krasnogor and S. Gustafson. A study on the use of "self-generation" in memetic algorithms. *Natural Computing*, 3(1):53–76, 2004.

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

[111] A. Krstic, J. Figueira, S. Greco, and M. Ehrgott. Multicriteria Decision Analysis: State of the art surveys. *Springer-Verlag, Ekonomski horizonti*, 20:189–191, 2018.

[112] S. Kyriacou. *Evolutionary Algorithm-based Design-Optimization Methods in Turbomachinery*. PhD thesis, National Technical University of Athens, 2013.

[113] S.A Kyriacou, V.G Asouti, and K.C. Giannakoglou. Efficient PCA-driven EAs and metamodel-assisted EAs, with applications in turbomachinery. *Engineering Optimization*, 46(7):895–911, 2014.

[114] 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*, 3(2):45–63, 2012.

[115] Y. LeCun, Y. Bengio, and G. Hinton. Learning representations by back-propagating errors. *Nature*, 521(7553):436–444, 2015.

[116] M. Li, T. Zhang, Y. Chen, and A. Smola. Efficient mini-batch training for stochastic optimization. In *Proceedings of the 20th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 661–670, New York, USA, 2014.

[117] 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.

[118] L. Lin and G. Mitsuo. Hybrid evolutionary optimisation with learning for production scheduling: state-of-the-art survey on algorithms and applications. *International Journal of Production Research*, 56(1-2):193–223, 2018.

[119] S. Lohr. The origins of 'big data': An etymological detective story. *The New York Times, Retrieved 28 September*, 2016.

[120] J. Long, E. Shelhamer, and T. Darrell. Fully convolutional networks for semantic segmentation. In *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2015.

[121] A. Maesani, G. Iacca, and D. Floreano. Memetic viability evolution for constrained optimization. *CoRR*, abs/1810.02702, 2018.

[122] Z. Michalewicz. *Genetic Algorithms + Data Structures = Evolution Programs*. Springer-Verlang, 3rd edition, 1996.

[123] Z. Michalewicz and D. Fogel. How to solve it: Modern heuristics. In *Springer-Verlag*, 2000.

[124] K. Miettinen. *Nonlinear multiobjective optimization*. Kluwer Academic Publishers, Boston, 1999.

[125] A.K. Morales and C.V. Quezada. A universal eclectic genetic algorithm for constrained optimization. In *6th European Congress on Intelligent Techniques & Soft Computing, Verlag Mainz*, pages 518–522, 1998.

[126] R. Myers and D. Montgomery. *Response Surface Methodology Process and Product Optimization Using Designed Experiments*. 2nd edition, 2002.

[127] F. Neri and C. Cotta. Memetic algorithms and memetic computing optimization: A literature review. *Swarm and Evolutionary Computation*, 2:1–14, 2012.

[128] Ferrante Neri and Carlos Cotta. Memetic algorithms and memetic computing optimization: A literature review. *Swarm and Evolutionary Computation*, 2:1–14, 02 2018.

[129] M. Omidvar, X. Li, and X. Yao. Cooperative co-evolution with delta grouping for large scale non-separable function optimization. In *IEEE Congress on Evolutionary Computation*, pages 1–8, July 2010.

[130] D.I. Papadimitriou and K.C. Giannakoglou. A continuous adjoint method with objective function derivatives based on boundary integrals for inviscid and viscous flows. *Computers & Fluids*, 36(2):325–341, 2007.

[131] E.M. Papoutsis-Kiachagias. *Adjoint Methods for Turbulent Flows, Applied to Shape or Topology Optimization and Robust Design*. PhD thesis, National Technical University of Athens, 2013.

[132] L. Piegl and W. Tiller. *The NURBS book*. Springer, 1997.

[133] Y. Po Lung. A class of solutions for group decision problems. *Management Science*, 19(8):936–946, 1973.

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

[135] D. Powell and M.M. Skolnick. Using genetic algorithms in engineering design optimization with non-linear constraints. In *5th International Conference on Genetic Algorithms, Morgan Kaufmann Publishers Inc.*, pages 424–431, 1993.

[136] A. Radford, L. Metz, and S. Chintala. Unsupervised representation learning with deep convolutional generative adversarial networks. *CoRR*, abs/1511.06434, 2015.

[137] L. A. Rastrigin. *Systems of extremal control*. Mir, Moscow, 1974.

[138] I. Rechenberg. Cybernetic solution path of an experimental problem. In *Library Translation 1122, Royal Aircraft Establishment*, Farnborough, UK, 1965.

[139] I. Rechenberg. *Optimierung technischer Systeme nach Prinzipien der biologischen Evolution*. PhD thesis, Technical University of Berlin, 1971.

[140] P. Roe. Approximate Riemann solvers, parameter vectors, and difference schemes. *Journal of Computational Physics*, 43:357–372, 1981.

[141] O. Ronneberger, P. Fischer, and T. Brox. U-net: Convolutional networks for biomedical image segmentation. In *Medical Image Computing and Computer-Assisted Intervention*, pages 234–241. Springer International Publishing, 2015.

[142] G. Rossum. Python tutorial. *Technical Report CS-R9526*, 1995.

[143] R. Roy and A. Tiwari. Generalised regression GA for handling inseparable function integration: Algorithm and application. *Parallel Problem Solving from Nature - PPSN VII, Lecture Notes in Computer Science*, 2439:452–461, 2002.

[144] R. Roy and A. Tiwari. Variable dependence interaction and multi-objective optimizations. *Genetic and Evolutionary Computation Conference*, 2002.

[145] D. Rumelhart, G. Hinton, and R. Williams. Learning representations by back-propagating errors. *Nature*, 323(6088):533–536, October 1986.

[146] T. Saaty. The analytic hierarchy process. *McGraw-Hill, New York*, 1980.

[147] M. Sakharov and A. Karpenko. *Multi-memetic Mind Evolutionary Computation Algorithm Based on the Landscape Analysis*, pages 238–249. Dublin, Ireland, 01 2018.

[148] T. Salimans, A. Karpathy, X. Chen, and D. Kingma. PixelCNN++: Improving the PixelCNN with discretized logistic mixture likelihood and other modifications. *CoRR*, abs/1701.05517, 2017.

[149] R. Salomon. Re-evaluating genetic algorithm performance under coordinate rotation of benchmark functions. a survey of some theoretical and practical aspects of genetic algorithms. *BioSystems*, 39:263–278, 1996.

[150] K. Samouchos. *The continuous adjoint method to immersed boundary methods for turbomachinery applications*. PhD thesis, National Technical University of Athens, (in progress).

[151] J.D. Schaffer and L.J. Eshelman. On crossover as an evolutionary viable strategy. In *Forth International Conference on Genetic Algorithms*, pages 61–68, 1991.

[152] J.D. Schaffer and A. Morishima. An adaptive crossover distribution mechanism for genetic algorithms. In *Second International Conference on Genetic Algorithms*, pages 36–40, 1992.

[153] J. Schmidhuber. Deep Learning. *Scholarpedia*, 10(11):32–38, 2015.

[154] V. Schmitt and F. Charpin. Pressure distributions on the ONERA M6 wing at transonic mach numbers, experimental data base for computer program assessment. Technical report, AGARD 138, 1979.

[155] B. Schölkopf, A. Smola, and K. Müller. Nonlinear component analysis as a kernel eigenvalue problem. *Journal of Neuron Computation*, 10(5):1299–1319, 1998.

[156] S. Schreck, W. Faller, and M. Luttges. Neural network prediction of three-dimensional unsteady separated flowfields. *Journal of Aircraft*, 32(178), 1995.

[157] S. Schreck, W. Faller, and M. Luttges. Efficient unsteady aerodynamic loads prediction based on nonlinear system identification and proper orthogonal decomposition. *Journal of Fluids and Structures*, 67(1), 2016.

[158] H. Schwefel. *Evolutionsstrategie und numerische Optimierung*. PhD thesis, Technical University of Berlin, 1975.

[159] M. Sefrioui and J. Périaux. A hierarchical genetic algorithm using multiple models for optimization. In *6th international conference on parallel problem solving from nature (PPSN VI). Lecture Notes in Computer Science*, volume 1917, pages 879–888. Paris, France, 2000.

[160] R. Shang, J. Wang, L. Jiao, and Y. Wang. An improved decomposition-based memetic algorithm for multi-objective capacitated arc routing problem. *Applied Soft Computing*, 19:343–361, 2014.

[161] V. Shim, K. Tan, and H. Tang. Adaptive memetic computing for evolutionary multiobjective optimization. *IEEE Transactions on Cybernetics*, 45(4):610–621, 2015.

[162] K. Sindhya, K. Miettinen, and K. Deb. A hybrid framework for evolutionary multi-objective optimization. *IEEE Transactions on Evolutionary Computation*, 17:495–511, 2013.

[163] L. Songjing, J. Liu, and D. Jiang. Dynamic characterization of a valveless micropump considering entrapped gas bubbles. *Journal of Heat Transfer*, 135, 2013.

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

[165] W.M. Spears. Crossover or mutation? In *Foundations of Genetic Algorithms 2, Morgan Kaufmann*, pages 221–237, 1992.

[166] P.D. Surry and N.J. Radcliffe. The COMOGA method: Constrained optimisation by multiobjective genetic algorithms. *Control and Cybernetics*, 26:391–412, 1997.

[167] K. Tan, J. Chen, and D. Wang. Gated residual networks with dilated convolutions for monaural speech enhancement. *IEEE/ACM Transactions on Audio, Speech, and Language Processing*, 27(1):189–198, 2019.

[168] L. Thiele, K. Miettinen, P. Korhonen, and J. Molina. A preference–based evolutionary algorithm for multi–objective optimization. *Evolutionary Computation*, 17(3):411–436, 2009.

[169] T. Tieleman and G. Hinton. Lecture 6.5-rmsprop: Divide the gradient by a running average of its recent magnitude. *COURSERA: Neural Networks for Machine Learning*, 2012.

[170] 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.

[171] X.S. Trompoukis. *Solving aerodynamic-aeroelastic problems on Graphics Processing Units*. PhD thesis, National Technical University of Athens, 2012.

[172] X.S. Trompoukis, V.G. Asouti, I.C. Kampolis, and K.C. Giannakoglou. *CUDA implementation of Vertex-Centered, Finite Volume CFD methods on Unstructured Grids with Flow Control Applications*, chapter 17. Morgan Kaufmann, 2011.

[173] X.S. Trompoukis, K.T. Tsiakas, M. Ghavami Nejad, V.G. Asouti, and K.C. Giannakoglou. The continuous adjoint method on graphics processing units for compressible flows. In *OPT-i, International Conference on Engineering and Applied Sciences Optimization*, Kos Island, Greece, June 4-6 2014.

[174] K. Tsiakas. *Development of optimization methods for use on Graphics Processing Units, with turbomachinery applications*. PhD thesis, National Technical University of Athens, 2019.

[175] K.T. Tsiakas, F. Gagliardi, X.S. Trompoukis, and K.C. Giannakoglou. Shape optimization of turbomachinery rows using a parametric blade modeller and the continuous adjoint method running on GPUS. In *ECCOMAS Congress 2016, VII European Congress on Computational Methods in Applied Sciences and Engineering*, Crete island, Greece, June 5-10 2016.

[176] TUBStator. TurboLab Stator Description, 2018. http://aboutflow.sems.qmul.ac.uk/events/munich2016/benchmark/testcase3/.

[177] Q. Wang. Kernel principal component analysis and its applications in face recognition and active shape models. *Technical Report*, arXiv:1207.3538, 2012.

[178] C. Wellock and B. Ross. An Examination of Lamarckian Genetic Algorithms. In *2001 Genetic and Evolutionary Computation Conference Late Breaking Papers*, pages 474–481, 2001.

[179] 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.

[180] B. Xin, L. Chen, J. Chen, H. Ishibuchi, K. Hirota, and B. Liu. Interactive multiobjective optimization: A review of the state-of-the-art. *IEEE Access*, 6:41256–41279, 2018.

[181] X. Yu and X. Lu, Y. Yu. Evaluating multiobjective evolutionary algorithms using mcdm methods. *Mathematical Problems in Engineering*, 2018:1–13, 03 2018.

[182] T. Zervogiannis. *Optimization methods in aerodynamics and turbomachinery based on the adjoint technique, hybrid grids and the exact Hessian matrix*. PhD thesis, National Technical University of Athens, 2011.

[183] E. Zitzler, K. Deb, and L. Thiele. Comparison of multiobjective evolutionary algorithms: Empirical results. *Evolutionary Computation*, 8:173{195, 2000.

[184] 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, Barcelona, Spain*, pages 19–26, 2002.

[185] E. Zitzler and L. Thiele. An evolutionary algorithm for multiobjective optimization: The strength Pareto approach. *TIK-Report No. 43, Computer Engineering and Communication Networks Lab, ETH Zurich*, 1998.

[186] E. Zitzler and L. Thiele. An evolutionary algorithm for multiobjective optimization: The strength Pareto approach. *Computer Engineering and Communication Networks Lab*, 1998.

[187] A.S. Zymaris. *Adjoint methods for the design of shapes with optimal aerodynamic performance in laminar and turbulent flows*. PhD thesis, National Technical University of Athens, 2010.