Genetic algorithms applied in Computer Fluid Dynamics for multiobjective optimizationn

The University of Vermont

Javier Lobato Perez
University of Vermont

A thesis submitted for the degree of
Bachelor of Science in Aerospace Engineering

May 2018

This thesis is dedicated to
my family
for being the greatest source of support

Acknowledgements

I am grateful to my Thesis Advisor, Professor Yves Dubief, for all the support given throughout the whole process of developing this project. The computational resources provided let me perform the required simulations for drawing conclusions and getting results. I really appreciate the help from my co-advisor at the University of Leon, Rafael Santamara, for all the advice and helpful information. I would also like to thank Professor Maggie Eppstein for the aid with the genetic algorithm topics and also other professors whose support has also been helpful for this Senior Thesis.

I will also like to thank the whole community for the indirect help: Python as the programming language used and OpenFOAM for being the whole complete simulation suite with powerful characteristics. Also, the webpage `stack-overflow` for being the reference website for solving most of the programming questions that I have while developing the code.

And finally, last but by no means least, to my parents, whose support and encouragement have helped me in the way, to my sister, for the positive feedback provided, to Ines, for being a major pillar every day, to Alberto and Veronica, for the hard-work nights in Votey, to Pablo, for some useful Python commands, and to everyone else that surely has had impact in this Senior Thesis.

Abstract

The use of optimization in computational fluid dynamics for engineering situations is a trending topic that is gaining a lot of attention. This study was focused on applying an optimization method to different computer fluid dynamics cases. Instead of using adjoint methods, genetic algorithms were used for the optimization process. The research was conducted with three different cases: vortex shedding in a cylinder wake (for minimizing the oscillations), diffuser inlet design (for maximizing the pressure ratio and Mach number) and airfoil shape optimization (different optimization configurations were tested). For these three cases, multiobjective optimization was imposed, having Pareto fronts with solutions that coexist in a trade-off situation rather than a single optimum point. The simulations produced promising results, showing that the approach is more than viable: it is a robust, adaptable and versatile way of optimizing engineering systems. Apart from the high level of parallelization and the automation of the procedure, the straightforward implementation allows that subtle changes in the code will optimize very different cases. Although the Pareto front was determined for the different cases, future work should be performed to increase the convergence of the method while reducing the number of simulations, given that it is a very time-consuming operation.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Optimization problems are everywhere, especially in engineering. Achieving the highest performance and increasing as much as possible the efficiency while reducing the costs are some of the objectives that every design tries to achieve. Finding the optimum is not always easy. In complex cases, the physics and the models used cannot be analyzed as a plain mathematical function from which extra information may be extracted to get the optimum solution.

In those cases, machine learning is one of the tools that are in constant rise for problems from all disciplines. One of any of its different subdisciplines is almost always suitable to achieve a particular solution for one specific problem.

As an aerospace engineering, the use of computer fluid dynamics is essential for the analysis of aerodynamics and heat transfer problems. In order to optimize different engineering systems, the use of computer fluid dynamics to obtain accurate solutions appears to be one of the best tools. Combining this with some optimization techniques taken from machine learning will create a robust toolbox to perform optimization problems and apply them to engineering.

Chapter 2

Literature review

Before moving into the different cases and simulations carried out, an extensive analysis of the current state-of-art must be done: beginning from the basics and going through the latest papers. The study will be structured in the three main parts of the thesis: optimization of multi-objective problems, genetic algorithms and computer fluid dynamics.

2.1 Optimization

Most parts of the real-life engineering problems face not one but more variables to be optimized. The value of those variables is tried to be minimized or maximized depending on the case. However, in both cases, the idea is to get the optimum value. Those kind of problems are called *multi-objective optimization* given that there is more than one objective that is wanted to be optimized. The main difficulty is that those objectives are usually in conflict with each other, i.e. there is not a point where the solution is optimal (in the sense that it minimizes/maximizes all the objectives).

The formal definition of multi-objective optimization is [1]:

$$\begin{array}{ll} \text{minimize} & \{f_1(\boldsymbol{x}), f_2(\boldsymbol{x}), ..., f_k(\boldsymbol{x})\} \\ \text{subject to} & \boldsymbol{x} \in S \end{array} \tag{2.1}$$

where there are k objective functions that should be minimized ($k > 2$). The decision vector $\boldsymbol{x}$ contains the n variables $\boldsymbol{x} = (x_1, x_2, ..., x_n)$ of the k functions and it must belong to a nonempty set S of the search space. S is also called the feasible set of the search space and it is usually described by constraint functions (in addition to some upper and lower bounds for each variable). Optimization problems are usually described to minimize a function, but maximize the function $f_i(\boldsymbol{x})$ is equivalent to minimize the function $-f_i(\boldsymbol{x})$.

It must be noted that there are two spaces in which the optimization works: parameter or search space (denoted with S in the literature) and optimization, objective or function space (denoted with Z). These different designations will be interchangeable during the oncoming discussion.

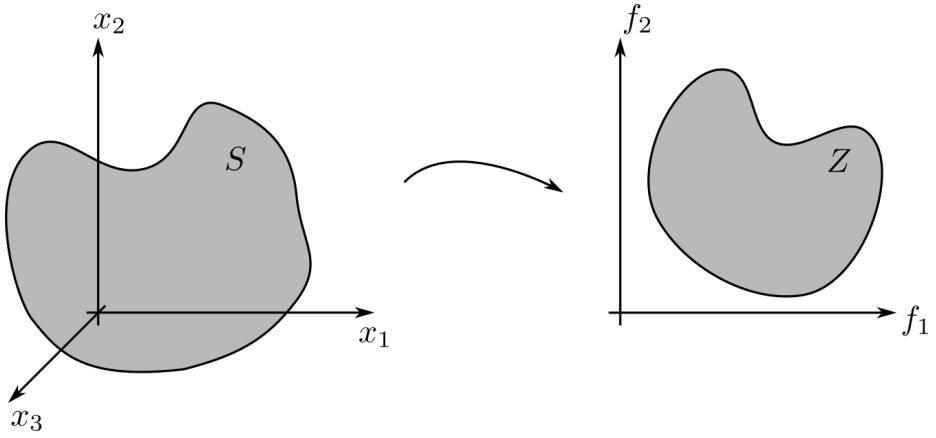

Figure 2.1: Search (S) and function (Z) space in multi-objective optimization

A visual example of this kind of problems can be seen in Figure 2.2, where the test function proposed by Binh and Korn (Test Case 2 from [2]) is plotted. The function was stated as:

$$\begin{aligned} \text{minimize} \quad & \begin{cases} f_1(x_1, x_2) = 4x_1^2 + 4x_2^2 \\ f_2(x_1, x_2) = (x_1 - 5)^2 + (x_2 - 5)^2 \end{cases} \\ \text{subject to} \quad & \begin{cases} (x_1 - 5)^2 + x_2^2 - 25 \leq 0 \\ -(x_1 - 8)^2 - (x_2 + 3)^2 + 7.7 \leq 0 \end{cases} \\ \text{bounded by} \quad & -15 \leq x_i \leq 30, \quad \forall i = 1, 2 \end{aligned} \tag{2.2}$$

As it can be seen in the figure 2.2, both objective functions are smooth and have defined maximums, but they are not located in the same $\boldsymbol{x}$: in the case of f_1 the minimum is located in $(0, 0)$ and in f_2 the minimum is in $(5, 5)$. Thus, instead of having one unique solution $\boldsymbol{x}$ a set of $\boldsymbol{x}^* \in S$ will form the 'optimum' solution. This set of possible values will not be able to optimize both functions at the same time but obtain the better parameters for both cases. From that set of possible solutions, a decision maker should choose the best combination based on previous information. The value of $f(\boldsymbol{x}^*)$ is the objective vector for the optimum decision vector $\boldsymbol{x}^*$.

Weighted averaged sum

The problem of optimizing multiple objectives at the same time may seem banal: transforming the multi-criteria problem into a weighted sum of the different objective

$$f_1(x_1, x_2) = 4x_1^2 + 4x_2^2 \qquad f_2(x_1, x_2) = (x_1 - 5)^2 + (x_2 - 5)^2$$

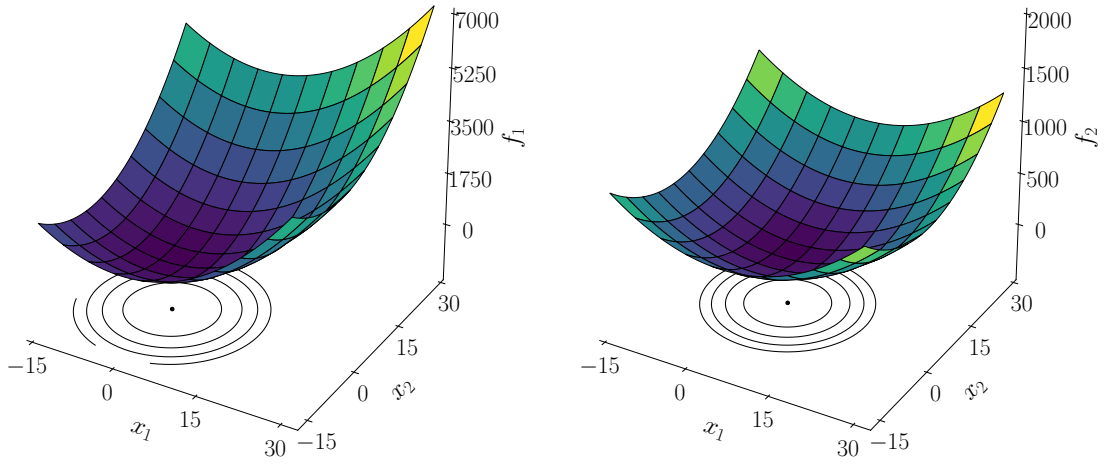

Figure 2.2: Bihn and Korn test function

functions is one approach that has been widely used and developed (see [3] and [4]). This plain aggregating approach consists of:

$$\begin{aligned} &\text{minimize} \quad F(\boldsymbol{x}) = \sum_{m=1}^{k} w_m f_m(\boldsymbol{x}) \\ &\text{subject to} \quad \boldsymbol{x} \in S \end{aligned} \tag{2.3}$$

This may seem like a good approach. Its main advantage is the simplicity, but the success of the method largely depends on the chosen weights, which value is determined with the relative importance of each objective to the additive objective. It has also problems when dealing with non-convex objective spaces, given that some solutions can't be represented with the average sum [5], [6].

2.1.1 Pareto front

When there are multiple functions that are to be minimized with a trade-off between them that doesn't allow the existence of a single optimum decision vector $\boldsymbol{x}$, Pareto dominance concept arises. Let's assume that there are two feasible solutions that belong to the search space, such that $\boldsymbol{x}^1 \in S$ and $\boldsymbol{x}^2 \in S$.

It is said that the solution $\boldsymbol{x}^1$ dominates the solution $\boldsymbol{x}^2$ when $\boldsymbol{x}^1 \prec \boldsymbol{x}^2$. The formal definition of the Pareto dominance between two decision vectors is [7]:

$$\boldsymbol{x}^1 \prec \boldsymbol{x}^2 \quad \text{if} \quad \begin{cases} f_i(\boldsymbol{x}^1) \leq f_i(\boldsymbol{x}^2) & \forall i \in \{1, 2, 3, ..., k\} \\ f_j(\boldsymbol{x}^1) < f_j(\boldsymbol{x}^2) & \text{for at least one } j \in \{1, 2, 3, ..., k\} \end{cases} \tag{2.4}$$

The concept of Pareto dominance is shown with a graphical explanation (restricted for 2 objective problems for simplicity) in the next figure:

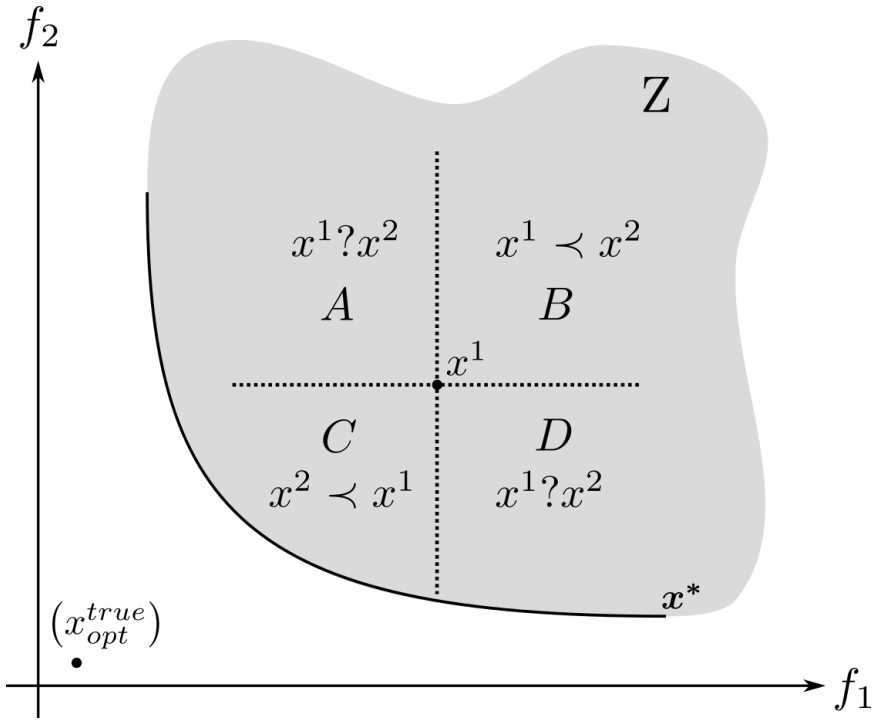

Figure 2.3: Pareto front dominance concept example

Each decision vector $\boldsymbol{x}^1$ splits the function space in 4 areas. A decision vector $\boldsymbol{x}^2$ (assuming that $\boldsymbol{x}^1 \neq \boldsymbol{x}^2$) may belong to:

- *Area A* or *Area D*: the relation between $\boldsymbol{x}^1$ and $\boldsymbol{x}^2$ cannot be determined, given that one performs better in one objective but worse in the other and viceversa.
- *Area B*: there is a Pareto dominance relation, having $\boldsymbol{x}^1 \prec \boldsymbol{x}^2$, where vector $\boldsymbol{x}^1$ dominates $\boldsymbol{x}^2$.
- *Area C*: there is also a Pareto relation between the decision vectors, having $\boldsymbol{x}^2 \prec \boldsymbol{x}^1$. Thus, $\boldsymbol{x}^1$ is dominated by $\boldsymbol{x}^2$.

If there is one decision vector $\boldsymbol{x}^*$ for which there is not any other $\boldsymbol{x}$ in *Area C* (i.e. there is not any solution that dominates $\boldsymbol{x}^*$) it is said that it is Pareto optimal or non-dominated. The set of Pareto optimal solutions $\boldsymbol{x}^*$ is usually called Pareto front or Pareto frontier, which consists of all non-dominated decision vectors (Figure 2.3). It can also be seen which is the 'true' optimum value, although given that it is outside the function space Z it is not a valid solution.

The representation of the Pareto front in optimization problems with just two objectives is quite straightforward, as seen in Figure 2.3. The Pareto front can be also represented in three dimensions with surfaces, but in high-order multi-objective optimization cases, the representation becomes harder. For those problems, there are other alternatives to the classical representation, e.g. slices of the Pareto front [8], [9]. Another approach consists of Interactive Decision Maps (IDM) that are techniques that use the concept of the Edgeworth box to show the feasible set expanded by the decision vectors dominated by it [10].

Although the optimization process has been described to minimize functions (using $-f_i$ in case it must be maximized), with the Pareto front all possible combinations of maximization and minimization may be analyzed, as seen in Figure 2.4, where depending the optimization type used, a different zone of the function space is chosen as Pareto front [11]:

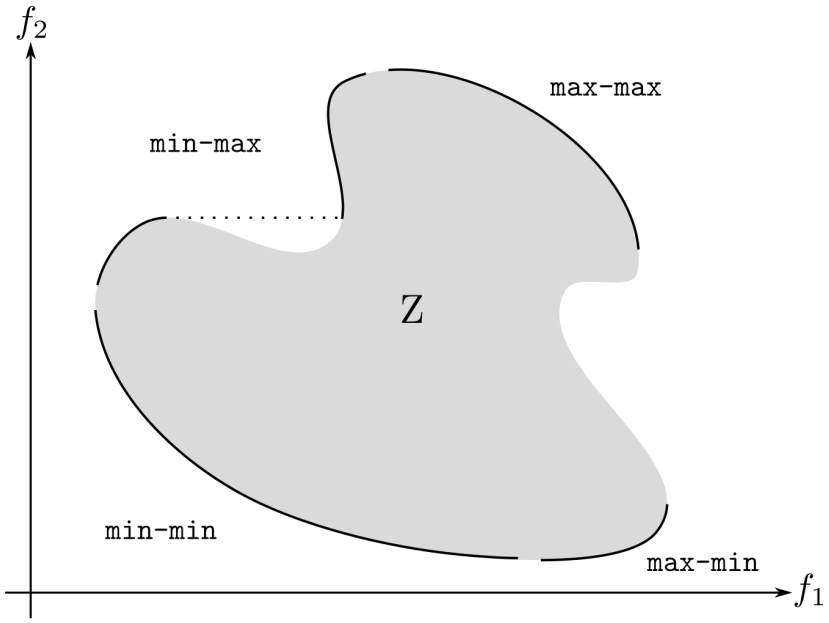

Figure 2.4: Different Pareto front depending on the type of optimization

2.1.2 Approaches for multi-objective optimization

One way to solve this kind of multi-objective problems is to create a random set of decision vectors to analyze the best possible the whole search space [12]. The most widely known stochastic techniques are the MCMC (Markov Chain Monte Carlo) methods [13]. They consist of different algorithms such as the Metropolis-Hasting algorithm [14] or Gibbs sampling [15]. MCMC methods provide a fairly good performance with low chance of the worst case performance. As stated by the name, the methods are based on the union of two different techniques [16]:

- Monte Carlo methods: they have been traditionally used in statistical physics, they are based on a random sample of the search domain, the deterministic evaluation of the inputs and the aggregation of the results.

- Markov chain processes: they provide a probabilistic model for state transitions or moves inside a domain. It is only dependant on the current position to determine the next position, i.e. it has no history retaining.

Given that error in Monte Carlo methods is reduced $\propto 1/\sqrt{N}$ [17], the number of evaluations in order to achieve a good accuracy must be high enough. Thus, the computational resources required to perform an analysis of this kind are considerable.

Other methods of solving this problem consist on physics-inspired algorithms (such as the Chaotic Optimization Algorithm), geography inspired algorithms (as the Imperialistic Competition Algorithm) and social culture inspired algorithms (such as Memetic Algorithm or the Selfish Gene Algorithm) [18]. However, the most popular methods are the biology-inspired algorithms such as evolution based algorithms and swarm-based algorithms, which will be discussed in the following section.

2.2 Evolutionary computation

Talking about artificial intelligence (AI) may seem unrelated to the previous topic at first, but given that evolutionary computation is a sub-field of it, AI must be introduced in order to know how it is structured. Artificial intelligence is a cross-disciplinary field that combines the understanding of the brain from neuroscience, the rise of computers and the new mathematics of information and control theory [16]. Therefore, AI is related to the development and investigation of systems that act in an intelligent way. Machine learning is the study of learning processes in different manifestations, i.e. machine learning is another subdivision of artificial intelligence which is focused on problem-solving [19].

Inside artificial intelligence there are different sub-disciplines, not limited to the ones shown in the next summary:

- Evolutionary computation: systems based on the Darwinian theory of evolution built on the natural selection of the better genes. There are different kinds of algorithms such as:
  - Genetic algorithms: it is a global optimization method with an adaptive strategy. It is based on the understanding of the structure and mechanisms of the genetics.
  - Genetic programming: it consists of the same principles of genetic algorithms but applied to computer software that adapts and improves its state over time. Instead of coding the lines with the instructions to perform certain tasks, the task is imposed and the evolutionary computation gets the optimum code to perform that task [20] (even improving human coded programs [21]).
  - Differential evolution: following the same principles as any other evolutionary computation, in this case, a scaled difference mutation is used instead of the typical probability distribution. It is usually used from constrained, large-scale and uncertain optimization problems, as well as multiobjective problems with multiple variables [22].
- Swarm intelligence: systems based on a large number of not very wise individuals that cooperate and interact with them, obtaining as result a collective intelligence.

- Artificial neural networks: systems that are based on a network that behaves as the neurons in the brain: managing the feedback from the environment and achieving some adaptive learning with it.

- Fuzzy intelligence: systems that consider a logic with degrees of truth instead of the constrained duality of true and false [23].

The classical AI field is divided into two approaches to the problems: *neat AI* and *scruffy AI*. The former uses symbolic representations and logic processes to analyze the given problem, achieving high fidelity in the results. The reductionist analysis is translated into scalability limits: a small increment in the size of the problem leads to an unmanageable increase in the complexity of the system (exaggerated execution time or computing resources). The latter is a descriptive method that takes advantage of the complex, emergent and self-organizing behavior of simple procedures. Using inductive approaches and some stochasticity in the system, the process is more robust when trying to approximate the solution of intractable problems with other methods. Scruffy AI is also known as *metaheuristic*: where *meta-* refers to the higher level strategy of combining different methods or procedures and *-heuristic* is the method that, in order to achieve faster computation time and less resource consuming processes, reduces the precision and quality of the solution (keeping it accurate enough to be sufficient for the case requisites) . These heuristic methods are approximate global solution cannot be ensured), normally non-deterministic and not problem-specific [16].

Conventional algorithms to approach optimization problems (not restricted to multi-objective optimization) are varied: Newton's method, Gradient Descent, Simple method, Nelder-Mead method,... Besides all these well-known techniques, there are algorithms that do not exploit information given from the problem to obtain a solution. This group is usually named as black-box optimization and it includes methods as genetic algorithms (given that only an evaluation of a decision vector is required). These techniques may be applied to a wide range of different problems with small modifications in the algorithm [24]. Associated with these black-box algorithms, the *no-free-lunch* theorem states that if one algorithm performs better than other in one particular case, that offset will be reversed for other problems where it will perform worse [25]. This proposition caused pessimism when comparing different optimization techniques, but it can be understood as if each problem is most suitable to certain algorithm that may not be valid for other cases - having to choose the better algorithm for each situation.

2.2.1 Genetic algorithms overview

A genetic algorithm (shortened as GA) is a population-based technique that tries to search for solutions to certain problems instead of searching paths for goals. Although searching one solution includes searching the path to achieving that solution, the main target of the algorithm is searching a solution in a large space efficiently instead of developing a decision tree to achieve a goal in all situations [26].

Given that genetic algorithms are based on biological evolution, some biological terminology may be used in the discussion (although mathematical terminology will take precedence). Therefore, before going into the details, these basic terms will be introduced. Cells are the basic element of all living organism and they store the DNA string structured in chromosomes. Each chromosome is divided into genes located in a particular locus of the chromosome, encoding each one gene for a protein. Most parts of the organisms have more than one chromosome per cell. All those possible chromosomes form the genome of the individuals, having that all genes contained inside a genome form the genotype of an organism.

There is a close relationship between optimization and genetic algorithms - that is why GA are such a good approach to optimization problems. As it has been mentioned (Figure 2.1), optimization used two different spaces: parameter space ($\boldsymbol{x}$) and the function space ($f(\boldsymbol{x})$). In genetic algorithms, there are also two spaces that represent the same concept: *search* space and *fitness* landscape. The search space is the set of all possible states that certain individual or chromosome may have (having each one different genes or components). Fitness landscape is the representation of all possible genotypes with their fitness. The fitness of an individual is a representation of 'how good' its chromosome is, which is translated into the possibilities of having offspring. Genetic algorithms assume that individuals with high fitness will have offspring that will have high fitness too - and it is there where the capabilities of GA relies on.

One aspect that makes genetic algorithms a very broad topic are the two possible ways of encoding the chromosome of each individual: *binary* and *real-valued* (although others as tree encoding or permutation encoding [27] have been also proposed). Binary encoding is the most common encoding, for both historical reasons and simplicity when dealing with the different operators. However, coding a state with strings of 0 and 1 when dealing with real numbers is unwieldy. In those cases, a real-valued encoding is preferred. This encoding is not limited to the use of float number but also strings of numbers, even combined with letters. The versatility of this encoding makes it widely used and the chosen one for further analysis.

2.2.2 Structure of a simple genetic algorithm

The main operation of a genetic algorithm and its structure will be included in this subsection. A genetic algorithm begins with the initialization of a population. It may be done in a random way, in an equally spaced way or in any other type of sampling of the search space. The *fitness* of the population is evaluated according to the function and objectives selected. Each individual of the population will have a fitness value associated. Then the three *genetic algorithm operators* are applied:

- Selection: this operator selects individuals from the population. There are different types of selection, but the two more common ones are [28]:
  - Roulette wheel selection: random individuals are chosen according to a probability proportional to the fitness value that each individual has.
  - Tournament method: n uniformly random chosen individuals are faced in a tournament, selecting the one with higher fitness.

  Any selection operator must ensure that the fitter the chromosome, the more times it is selected.

- Recombination: two individuals are chosen from the ones selected and are combined, giving rise to (at least) one new individual. As it has already been pointed out, GA assumes that the higher the fitness of the parents, so will be the fitness of the offspring. Recombination tries to increase convergence of the system by joining together points with high fitness. In this recombination phase there are three possibilities that may exist [29]:
  - Crossover: some of the genes are taken from one parent and the rest of the genes of the offspring chromosome are taken from the other parent. It is used to increase the diversity.
  - Average: two parents are chosen and create the chromosome of the offspring with the mean value of each of its genes. This may also be a weighted sum of the parents instead of a plain 50-50 mean.
  - Survival: it is not exactly recombination but the survival of one of the two chromosomes of the parents. This method takes the chromosome of a parent and saves it in the offspring population. Survival is also known as elitism, given that the elite of the population (sorted by the fitness value) will be preserved along generations.

- Mutation: perturbations are applied to each individual in order to ensure variability. The probability of mutation is usually chosen as input, with a value different from zero. The task of mutation is to introduce diversity back into the population. In optimization, this has an especial meaning, given that the GA must move between different optimum locations to analyze the whole domain, so it must be able to leave a local minimum in case it gets stuck in one. Mutation can be applied to a whole chromosome or only to some of its genes. It is a random process, so individuals may not be mutated and keep the chromosomes obtained after recombination.

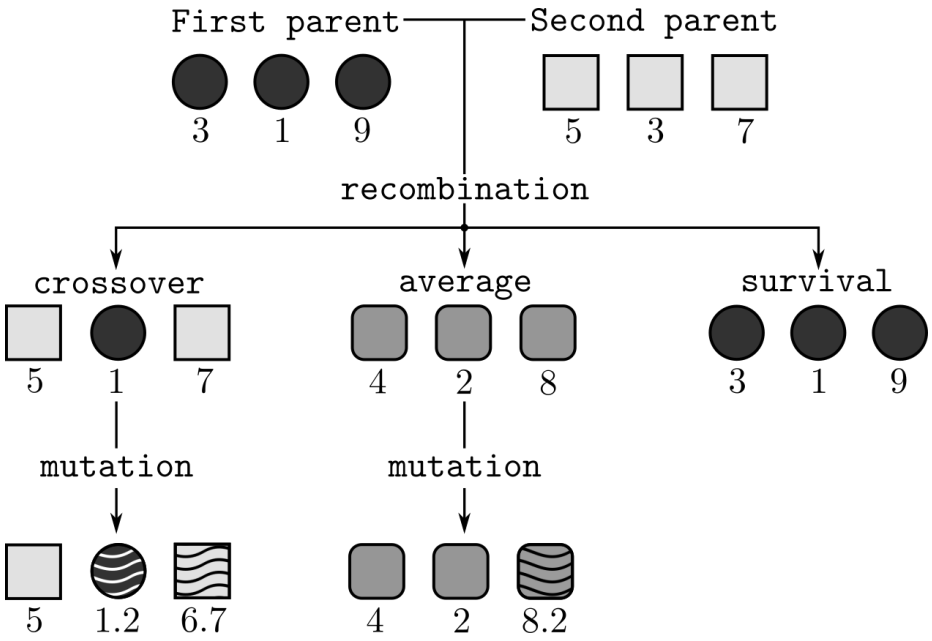

Figure 2.5: Recombination and mutation process from two parents

Once these three operators have been applied, a new population will arise - having a new generation. That population will undergo the same cycle until one stop condition is matched: it is usually the maximum number of generations or some relative error. For each generation two basic groups must exist: *parents* (individuals that have been chosen to reproduce) and *offspring* (individuals obtained from the combination of the parents). A third group, *elite*, may exist if the genetic algorithm applies the survival operator. Elitism [30] has been proved to have a considerable impact on performance, because it avoids wasting time in discovering again zones that have been already discarded as not valid. The same individual may be part of several groups at the same time, i.e. there are algorithms that compare the value of parents and offspring to keep the better for recombination and mutation.

Genetic algorithms may be mathematically formalized for simple genetic algorithms (see [31], [32], [33]), being able to analyze the dynamics of the system. GA are used for optimization problems, where once the optimum value is found, the state does not move from there. This kind of points are fixed points and in genetic algorithms are achieved once a population has completely converged to individuals with the same fitness. If a population does not have all individuals at the maximum fitness level, a small perturbation may move the solution from the fixed point. This situation causes *punctuated equilibrium* in fitness evolution: long periods of no improvement with quick rises in the fitness. Although these effects have to be rigorously quantified under different models, the implications are that a large enough number of generations must be computed to achieve a high fitness. The main assumption is that the population has an infinite size, which may lead to unexpected behaviors due to sampling errors in a genetic algorithm with a limited population size.

The basic procedure of a genetic algorithm is listed below:

Algorithm 1 Simple Genetic Algorithm

1: initialize *population*
2: **while** stopCriterion **not** reached **do**
3: calculate *fitness* of *population*
4: select *bestIndividuals* from *population*
5: *newPopulation* ← mutation ← crossover *bestIndividuals*
6: *population* ← *newPopulation*

2.2.3 Why do genetic algorithms work?

Although a genetic algorithm is procedure simple to describe and code, its behavior may be complicated and there are still many questions about how do they really work. A genetic algorithm results in complex and robust search method by implicitly sampling hyperplane partitions of a search space [34]. The genetic algorithm works by discovering and recombining good *building blocks* of solutions, given that a good building block will make up a good building block. Holland's book [35] refers to this building blocks as *schemas* that defines hyperplanes. One schema on its own does not provide enough information, that is where the population-based search concept is critical, given that many hyperplanes are evaluated at the same time. The cumulative effects of evaluating a population of points will provide enough information about any particular subset of hyperplanes.

2.2.4 Genetic algorithms for multi-objective optimization

Due to the similarities between genetic algorithms and multiobjective optimization, there are a lot of existing and well-tested methods that combine both. The ability of a GA to search in different regions of the solution space in a highly parallel fashion makes it a good tool for problems with non-convex, discontinuous and multi-modal solution spaces. Undoubtedly, genetic algorithms are one of the most popular heuristic approaches to multi-objective optimization. In the next table, the most popular genetic algorithms and some of its special features are listed [36]: Many

Table 2.1: Different genetic algorithms for multi-objective optimization

Algorithm	**Fitness assignment**	**Diversity mechanism**	**Elitism**	**External population**
VEGA	Each population is evaluated with respect to a different objective	No	No	No
MOGA	Pareto ranking	Fitness sharing by niching	No	No
PESA	No fitness assignment	Cell-based density	Yes	Yes
NSGA	Ranking based on non-domination sorting	Niching	No	No
NSGA-II	Ranking based on non-domination sorting	Crowding distance	Yes	No
SPEA	Ranking based on external archive of non-dominated	Clustering to truncate external population	Yes	Yes
SPEA-2	Strength of dominators	Density based of k nearest	Yes	Yes
DMOEA	Cell-based ranking	Cell-based density	Yes	No

other genetic algorithms exist and have been also tested, but only the most important and used have been listed above. There are some nomenclature that has not been introduced and it must be explained before analyzing the advantages and disadvantages of each method:

- **Fitness assignment**: procedure followed to assign the fitness to the individuals of each generation. It may be based on a wide range of methods of assigning.
- **Diversity mechanism**: techniques used to promote the diversity of the solutions, i.e., the spread of individuals to cover the whole search and function space.

- **Elitism**: for a single-objective genetic algorithm, elitism consists on saving the best solution for the next generation. However, in multi-objective optimization, all non-dominated solutions that form the Pareto front are considered elite solutions but elitism is not as straightforward as in single objective. Although earlier genetic algorithms did not include elitism, most recent strategies include it because it outperforms non-elitist counterparts.

- **External population**: proposes to save in a list the elitist individuals for each generation. Apart from being a computationally expensive task, the size of the list may grow too much. In order to avoid that, pruning techniques have been proposed to limit the maximum size of the elitist population.

- **Fitness sharing**: a method used to encourage the search in unexplored zones of a Pareto front by artificially reducing the fitness of the solutions in densely populated areas with some penalization method.

- **Niching**: technique consisting on segmenting the population in disjoint sets so each member may go to different zones, in order to cover more than one local optima. It is a highly used technique to widen the whole search space in fewer generations due to the splitting of the population.

- **Cell-based density**: consists on dividing the objective space into K cells, computing the density of each cell and assigning that value to each solution in the cell. Then, the most populated cells are penalized in a similar way as fitness sharing.

- **Crowding distance**: technique that eliminates the necessity of a user-defined parameter such as σ_{shared} or the k nearest neighbors. It computes the distance between the points of the Pareto front and uses that distance for the selection process.

- **Strength of dominators**: assignment of the fitness value according to the number of individuals that dominate each individual.

Once the terminology used in this kind of genetic algorithms is known, a brief discussion about each one of them will be held. Main advantages and disadvantages will be presented and shown in order to decide which will be the most optimum algorithm:

- VEGA (Vector Evaluated Genetic Algorithm): it was the first genetic algorithm designed for multi-objective optimization. Although the easy implementation, its main disadvantage is that it tends to converge to the extreme of each objective, losing an important part of the Pareto front.

- MOGA (Multi-Objective Genetic Algorithm): it is a simple extension of a genetic algorithm for single objective. However, it has problems related to the niche size parameter and the slow convergence.

- PESA (Pareto Envelope-based Selection Algorithm): it is easier to implement and (if done correctly) it is computationally efficient. The main problem is that the performance and the success of the method depend on the cell sizes and some previous knowledge about the objective space is required.

- NSGA (Non-dominated Sorting Genetic Algorithm): it has an extremely fast convergence but also major problems related to the niche size parameter.

- NSGA-II (Non-dominated Sorting Genetic Algorithm II): efficient, well tested and single parameter (N) method. The main disadvantage is that the crowding distance works only in the objective space.

- SPEA (Strength Pareto Evolutionary Algorithm): well test method without any parameter for clustering. Its inconvenient is that the clustering algorithm is quite complex.

- SPEA-2 (Strength Pareto Evolutionary Algorithm 2): it is an improved version of the SPEA that also makes sure that the extreme points of the Pareto front are preserved (in order to increase both the diversity and the convergence). The drawback of the method is that the fitness and density calculation are highly computationally expensive.

- DMOEA (Decomposition-based Multi-Objective Evolutionary Algorithm): it is a method that combines some efficient techniques to update cell densities and adaptative approaches to set the genetic algorithm parameters. The main downside is its complicated implementation.

From this list of some of the most well-known genetic algorithms for multiobjective optimization, one will be chosen for the implementation and application to the CFD cases. There are some characteristics that must exist in the genetic algorithm in order to be used successfully in these cases.

The desired characteristics that a GA must have for being applied successfully to CFD cases are:

- Elitism: keeping the best individuals along generations has shown that the convergence of the solution is faster and requires less number of generations. Thus, it is very important that the algorithm chosen has elitism, given that each individual may have an evaluation time of minutes due to the CFD simulation.
- Avoid user-defined parameters: achieving the correct solution is usually directly related to the value that those parameters have, so it is interesting to choose a genetic algorithm that does not have user-defined parameters. Some of these parameters include the k-nearest individuals or the size of the cells when performing the fitness evaluation.
- Well-tested method: an algorithm whose performance has been tested and is above certain limit when talking about convergence to a solution is preferable with respect to a newer method that has not been tested yet.
- Efficient and low resources consuming: given that the computational fluid dynamics simulations will be running for long periods of time, increasing the computational resources required for evaluating the fitness and performing the different operators is not a great way of approaching the problem.
- Straightforward implementation: given that the code will be implemented in Python, it is also interesting to have an algorithm whose coding may be done in a simple way and with a high level of parallelization.

Analyzing the algorithms in the table 2.1, the one that meets all the requirements is the NSGA-II.

2.2.5 Non-dominated Sorting Genetic Algorithm II)

The algorithm that was chosen to be implemented and used in the different computer fluid setups is the NSGA-II. This algorithm is described in [37] where the algorithm is presented as a review of the first version of the NSGA. Some characteristics that make this algorithm suitable for this study have been already mentioned. However, the potential of the algorithm lays in fitness evaluation and the way it handles the populations of parents and offspring. A combination of all these parameters is what makes this algorithm so attractive for this study.

The main features of the NSGA-II are:

- Fast non-dominated sorting: the points of an evaluation are sorted depending on the Pareto front in which they are located on. The points in the non-dominated Pareto front will have the greater fitness possible, while the points with higher values in the functions f_1 and f_2 have a smaller fitness. Selection is performed based on the number of the Pareto front in which the individual is located: the higher the fitness, the less dominated the individual will be by other individuals and the greater possibilities will it have to be selected.

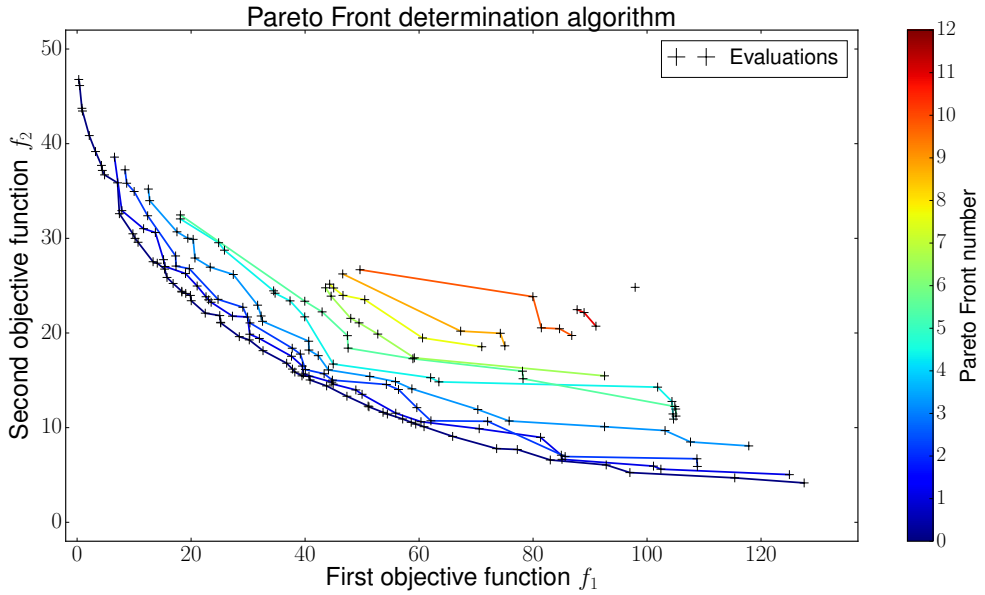

Figure 2.6: Fast non-dominated sorting

- Crowding distance: if a selection is performed with a binary tournament ($k = 2$) there are chances for two individuals of the same Pareto front to be in the tournament facing each other. Given that for the points in the same Pareto front, the value of the fitness is the same, there must exist another parameter or value used to break the tie. Crowding distance is used for that purpose, breaking possible ties in the selection process. It is assigned based on the distance to the closest point in the same Pareto front, having a higher crowding distance if the point is located in a low-density area. If the point is located in a dense area, the crowding distance will be smaller having, therefore, smaller chances of being chosen. The extrema points of the Pareto front have a crowding distance of ∞, so they are always chosen. In the Figure 2.7 it can be seen how points in less dense areas have a greater crowding distance.

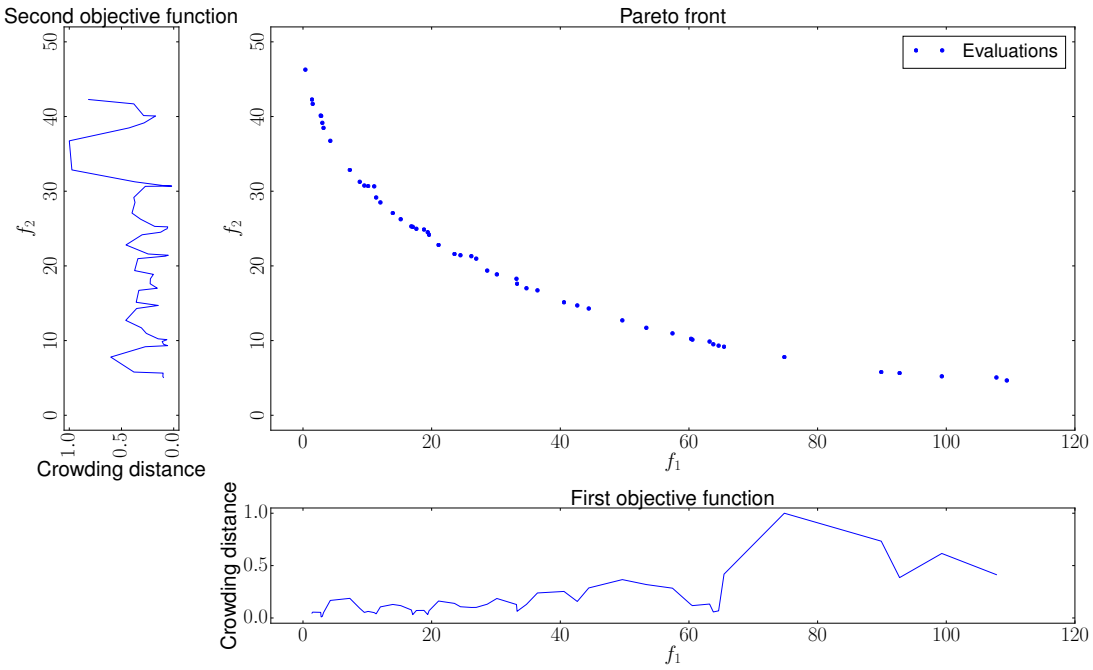

Figure 2.7: Crowding distance assignment

- Preselection process: these two ways of sorting a population allow a preselection process (prior to the classical selection, recombination and mutation processes) to increase the convergence of the method. It begins with a parent population P_t of size N which yields an offspring population Q_t of size N. These two populations are combined in the same set (having a size of $2N$), which is sorted according to the non-dominated sorting explained above (Figure 2.6). The whole set of both parents and offspring are sorted depending on the Pareto front they are located in. Provided that the size of the new population should be N, the less dominated Pareto fronts that fit completely the new population will go directly without going through any other process. Once the following Pareto front will not fit completely the new population P_{t+1}, the set is sorted with the crowding distance. Thus, from the last Pareto front, only the points with higher crowding distance will be selected for the parent population $P_t + 1$. The most rear fronts will not be selected for the new population as well as the lower crowding distance individuals from the last Pareto front. Those points are just rejected. Population P_{t+1} will undergo selection, recombination and mutation to yield population Q_{t+1}, closing this way the loop. This is an indirect way of performing elitism given that the best individuals are taken from the previous generation (P_t).

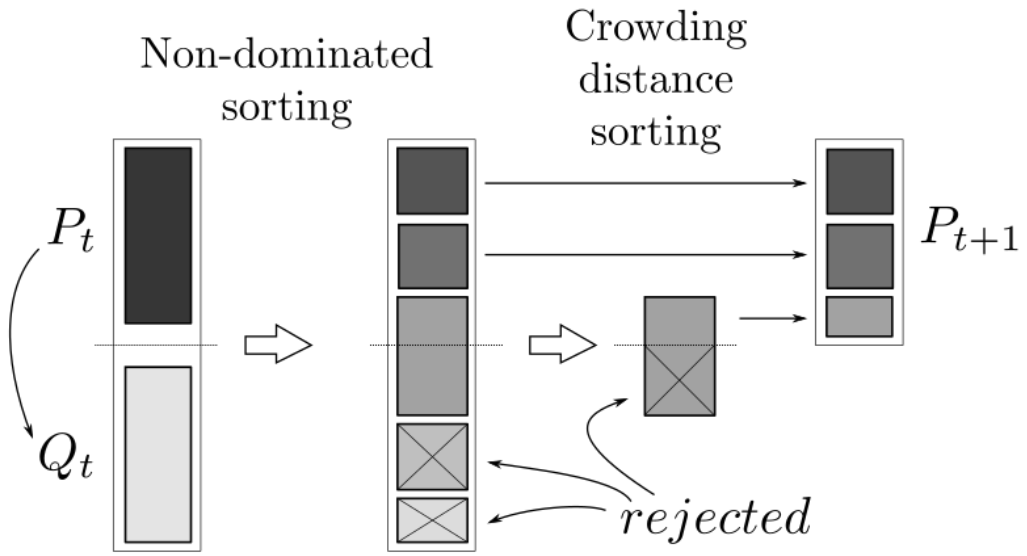

Figure 2.8: Preselection process for a parent population P_t

The main loop of the genetic algorithm combines all procedures shown before, having the next diagram:

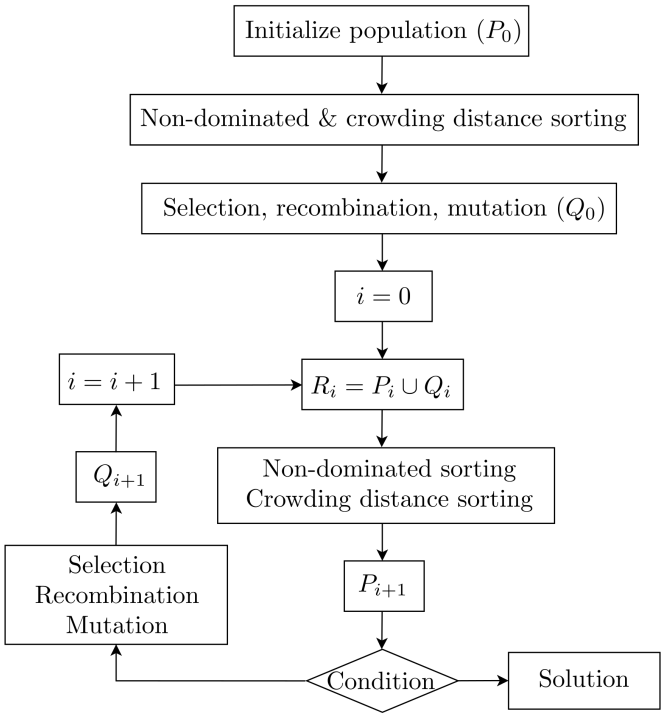

Figure 2.9: Chart flow of the NSGA-II algorithm

2.3 Computer Fluid Dynamics

Computer fluid dynamics is the analysis of systems involving especially fluid flow (but also heat transfer and chemical reactions) with computer-based simulations. The techniques are very powerful and they may be applied to a wide range of industrial applications. Although other computer-based simulation, such as stress analysis codes, have a greater capability than CFD codes (due to the incredible complexity of the underlying behavior of fluid flows), the use of CFD is essential in every aspect of the design of a new concept or product. The investment in a CFD code is smaller than the minimum cost of the required hardware for an experimental setup (having even some CFD codes that are completely free).

Almost all CFD codes work in the same fashion, having very differentiated steps [38]:

- Pre-processor: in this step, the setup of the problem is defined: the computational domain is created with a grid or mesh formed by different cells or control volumes. The set of equations to be solved and the fluid properties are selected in this step. Also, the conditions at the boundary of the domain are established.

- Solver: there are different numerical techniques available for approaching these problems: finite difference (especially the finite volume method), finite element and spectral methods. The numerical algorithm consists on a series of steps: the integration of the governing equation of fluid flow, discretization of the resulting integral equation into an algebraic system of equations and the solution of that algebraic system of equations.

- Post-processor: the output obtained in the solver phase is analyzed and data is extracted in order to draw some conclusions. With the increase of graphic capabilities of engineering workstations, a lot of versatile data visualization tools are used to present data in a more visual way.

The set of governing equations of the flow of a compressible Newtonian fluid that is solved in a computer fluid dynamics code is formed by five partial differential equations (PDEs) with two algebraic equations that supplement the five PDEs. The equations are listed in 2.2.

Table 2.2: Equations of a compressible Newtonian fluid

Continuity	$\frac{\partial \rho}{\partial t} + \text{div}(\rho \boldsymbol{u}) = 0$	(2.5)
x-momentum	$\frac{\partial (\rho u)}{\partial t} + \text{div}(\rho u \boldsymbol{u}) = -\frac{\partial p}{\partial x} + \text{div}(\mu \text{ grad } u) + S_{M_x}$	(2.6)
y-momentum	$\frac{\partial (\rho v)}{\partial t} + \text{div}(\rho v \boldsymbol{u}) = -\frac{\partial p}{\partial y} + \text{div}(\mu \text{ grad } v) + S_{M_y}$	(2.7)
z-momentum	$\frac{\partial (\rho w)}{\partial t} + \text{div}(\rho w \boldsymbol{u}) = -\frac{\partial p}{\partial z} + \text{div}(\mu \text{ grad } w) + S_{M_z}$	(2.8)
Energy	$\frac{\partial (\rho i)}{\partial t} + \text{div}(\rho i \boldsymbol{u}) = -p \text{ div } \boldsymbol{u} + \text{div}(k \text{ grad } T) + \boldsymbol{\Phi} + S_i$	(2.9)
Equations of state	$p = p(\rho, T)$ and $i = i(\rho, T)$	(2.10)
	(for a perfect gas $p = \rho R T$ and $i = C_v T$)	(2.11)

In these equations ρ represents the density of the flow, $\boldsymbol{u}$ is the velocity vector with its three components $\boldsymbol{u} = (u, v, w)$, t is time, p is the pressure, μ is the dynamic viscosity, (x, y, z) represent spatial components, i is the internal energy, k is the heat conduction coefficient and T is the temperature. Equations 2.6, 2.7 and 2.8 have a momentum source S_M, while 2.9 has an internal energy source term S_i and a dissipation function $\boldsymbol{\Phi}$.

To solve this system of equations, initial conditions and boundary conditions must be established for the case. Depending on the setup to be solved, this system may be simplified down to a more simpler system with fewer variables and equations (for example, by reducing the 3D analysis in a 2D case or doing a steady state solution where the temporal terms are neglected).

Chapter 3

Methodology

3.1 Optimization & Genetic algorithms

As explained in the previous chapter, there are different ways of approaching a multiobjective optimization problem. Three different ways will be analyzed in order to choose the most appropriate one for the coupling of a search method and computer fluid dynamics.

- Random search: this method is also known as Monte Carlo method. It consists of sampling the search space in a random fashion and hoping to get a good definition of the Pareto front. As said before this kind of methods are not very efficient and in order to achieve high accuracy values, the number of individuals must be big enough. Furthermore, this approach is only efficient in 2 and 3 variable search spaces, given that with more than 3 variables, the search space is too big to cover it with a simple sampling of points.

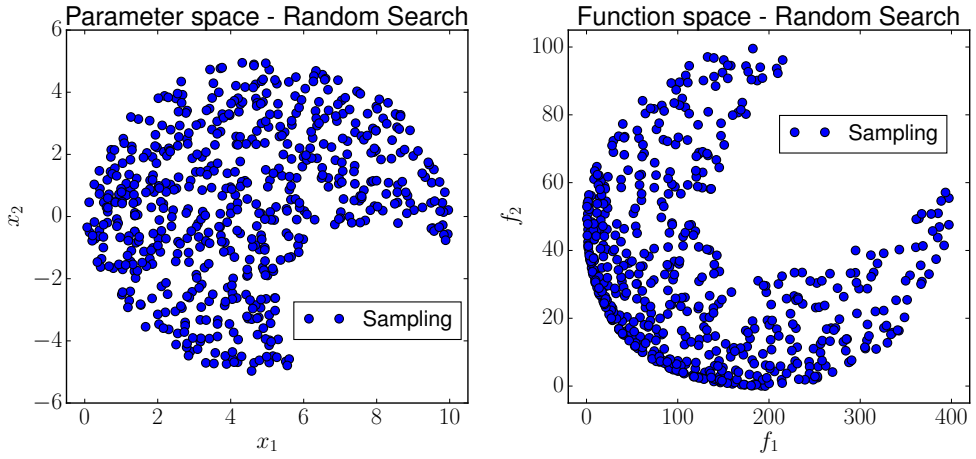

Figure 3.1: Random search method

- General genetic algorithm: a general GA was coded up in Python. This GA may be used for a great variety of tasks and it is not designed for multiobjective optimization problems. Fitness is assigned with the distance to the Pareto front of each one of the individuals. The selection method chosen is the roulette method, with linear crossover and normal distribution mutation. The results obtained with this method show the Pareto front and how the individuals converge to it along generations.

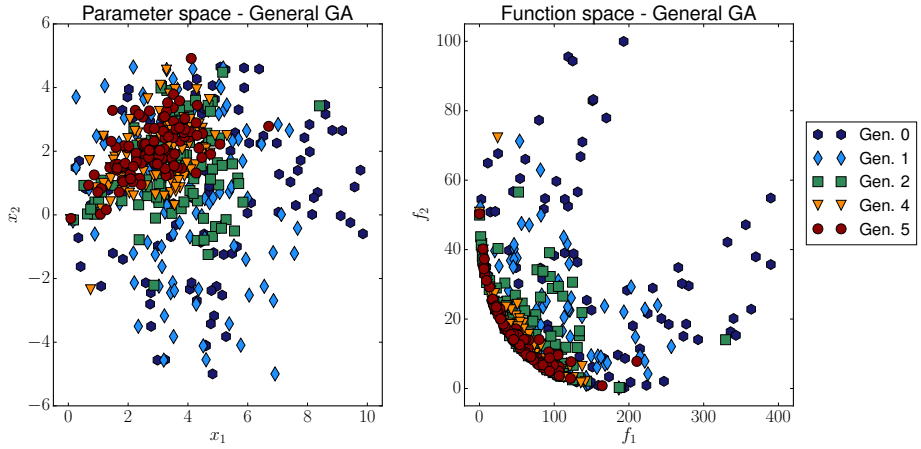

Figure 3.2: General genetic algorithm

- NSGA-II: the non-dominated sorting genetic algorithm is described in [37] and it is specially designed for multiobjective optimization problems. It is expected to have a greater convergence and more possibilities of capturing the whole Pareto front.

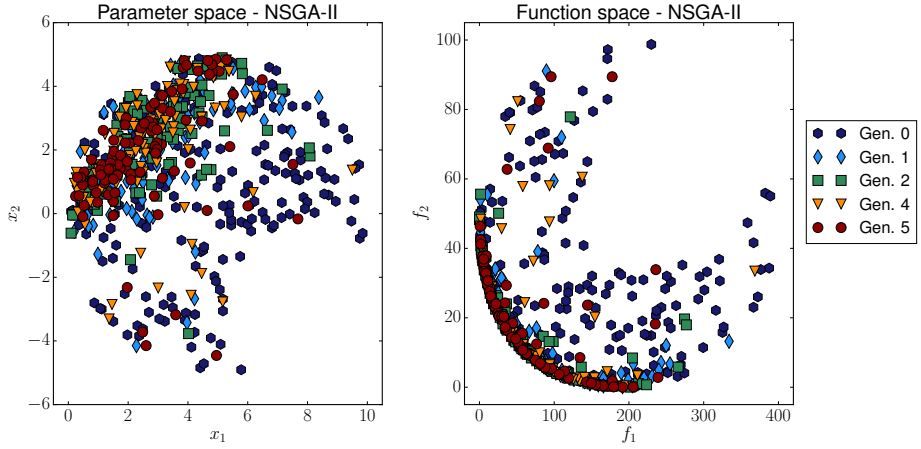

Figure 3.3: Non-dominated sorted genetic algorithm II

3.1.1 Comparison of the performance of different methods

The main question now is which one of these three methods will perform better, i.e. which one captures a more spread and precise Pareto front. The three methods searched the solution for different test multiobjective optimization problems computing a histogram to show which is the probability of having an individual in certain area of the search and fitness spaces. Sixteen runs were performed, using the values shown in Table 3.1. It may be seen that the individuals in the random search are the same as the number of evaluations. Also for the general GA the number of individuals times the number of generations yields the number of evaluations. However, for the NSGA-II case, numbers don't add up at first glance because the first population of the NSGA-II is initialized with a population twice the size ($2N$) of the rest of the populations (N).

Table 3.1: Different configurations for the different possibilities

RANDOM SEARCH	GENERAL GA		NSGA-II		NUMBER OF EVALUATIONS
Indiv.	Indiv.	Gener.	Indiv.	Gener.	
80	10	8	10	7	80
90	10	9	10	8	90
100	10	10	10	9	100
110	10	11	10	10	110
200	25	8	25	7	200
225	25	9	25	8	225
250	25	10	25	9	250
275	25	11	25	10	275
450	50	9	50	10	450
500	50	10	50	9	500
550	50	11	50	10	550
600	50	12	50	11	600
900	100	9	100	8	900
1000	100	10	100	9	1000
1100	100	11	100	10	1100
1200	100	12	100	11	1200

Genetic algorithms are usually run for 25000 iterations, with populations of 250 individuals. However, here the number of function evaluations is way lower because when applied to CFD, several minutes will take the evaluation of each individual (whereas for a function evaluation the whole process may last for seconds).

The test functions used were:

- **Bihn & Korn test function:** this function was already described before (2.2) but it is added here again for clarification. This is a function-constrained search optimization, with two objectives and two variables:

$$\begin{aligned} \text{minimize} \quad & \begin{cases} f_1(x_1, x_2) = 4x_1^2 + 4x_2^2 \\ f_2(x_1, x_2) = (x_1 - 5)^2 + (x_2 - 5)^2 \end{cases} \\ \text{subject to} \quad & \begin{cases} (x_1 - 5)^2 + x_2^2 - 25 \leq 0 \\ -(x_1 - 8)^2 - (x_2 + 3)^2 + 7.7 \leq 0 \end{cases} \\ \text{bounded by} \quad & -15 \leq x_i \leq 30, \quad \forall i = 1, 2 \end{aligned} \tag{2.2 revisited}$$

- **Poloni's test function:** it is a two objectives and two variables test function [39]:

$$\begin{aligned} \text{minimize} \quad & \begin{cases} f_1(x_1, x_2) = 1 + [A_1 - B_1(x_1, x_2)]^2 + [A_2 - B_2(x_1, x_2)]^2 \\ f_2(x_1, x_2) = (x_1 + 3)^2 + (x_2 + 1)^2 \end{cases} \\ \text{where} \quad & \begin{cases} A_1 = 0.5\sin(1) - 2\cos(1) + \sin(2) - 1.5\cos(2) \\ A_2 = 1.5\sin(1) - \cos(1) + 2\sin(2) - 0.5\cos(2) \\ B_1(x_1, x_2) = 0.5\sin(x_1) - 2\cos(x_1) + \sin(x_2) - 1.5\cos(x_2) \\ B_2(x_1, x_2) = 1.5\sin(x_1) - \cos(x_1) + 2\sin(2) - 0.5\cos(x_2) \end{cases} \\ \text{in} \quad & -\pi \leq x_i \leq \pi, \quad \forall i = 1, 2 \end{aligned} \tag{3.1}$$

- **Zitzler-Deb-Thiele's test functions:** ZDT are a classical set of test functions designed for genetic algorithms to test its performance. They are designed for 30 variables and biobjective optimization, but in this case, the ZDT2 function was adapted to just two variables [40]:

$$\begin{aligned} \text{minimize} \quad & \begin{cases} f_1(x_1, x_2) = x_1 \\ f_2(x_1, x_2) = g(x_1, x_2) \cdot h(f_1(x_1, x_2), g(x_1, x_2)) \\ g(x_1, x_2) = 1 + \dfrac{9x_2}{29} \\ h(x_1, x_2) = 1 - \sqrt{\dfrac{x_1}{x_2}} \end{cases} \\ \text{bounded by} \quad & 0 \leq x_i \leq 1, \quad \forall i = 1, 2 \end{aligned} \tag{3.2}$$

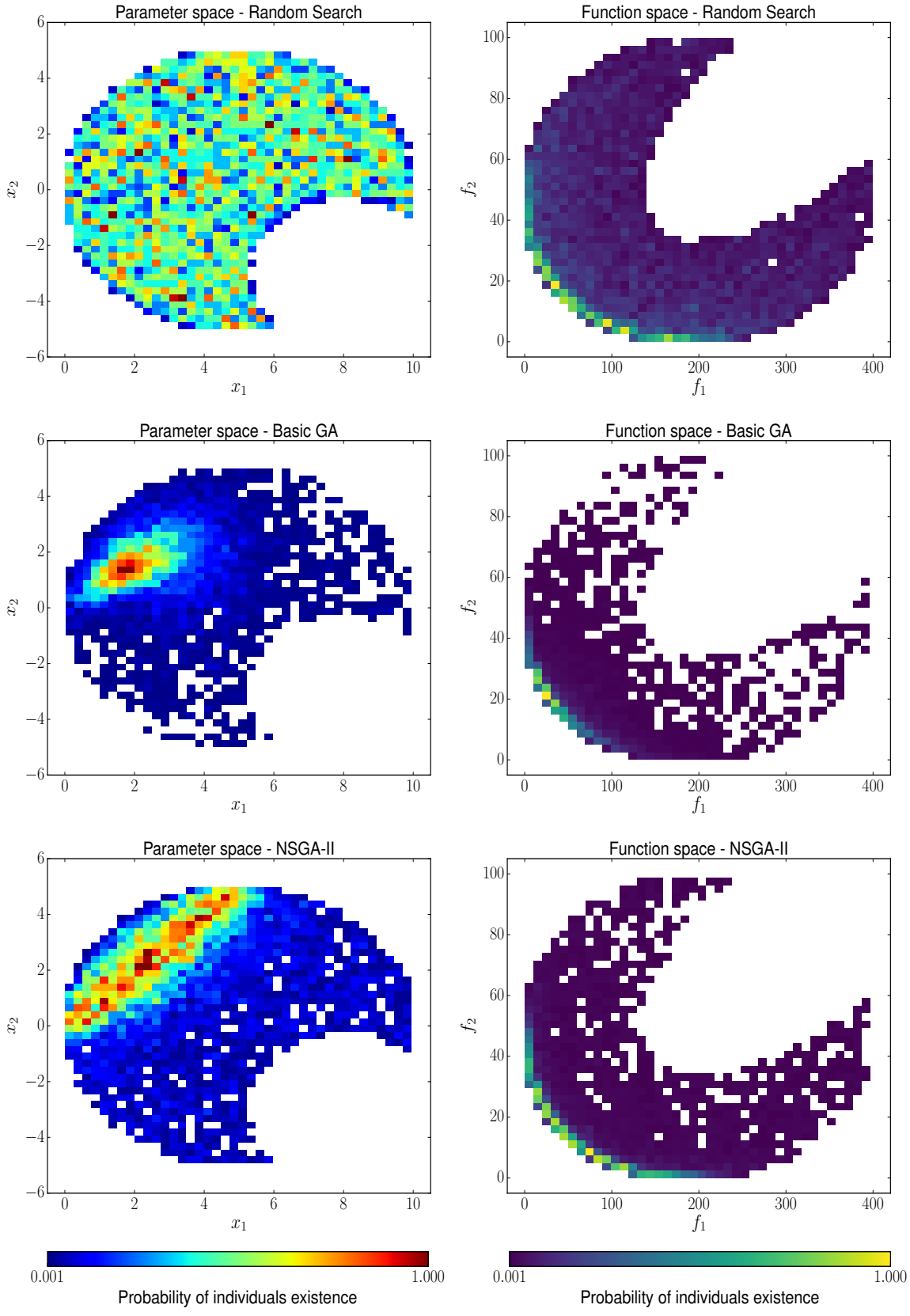

Figure 3.4: Method comparison for the Bihn & Korn function

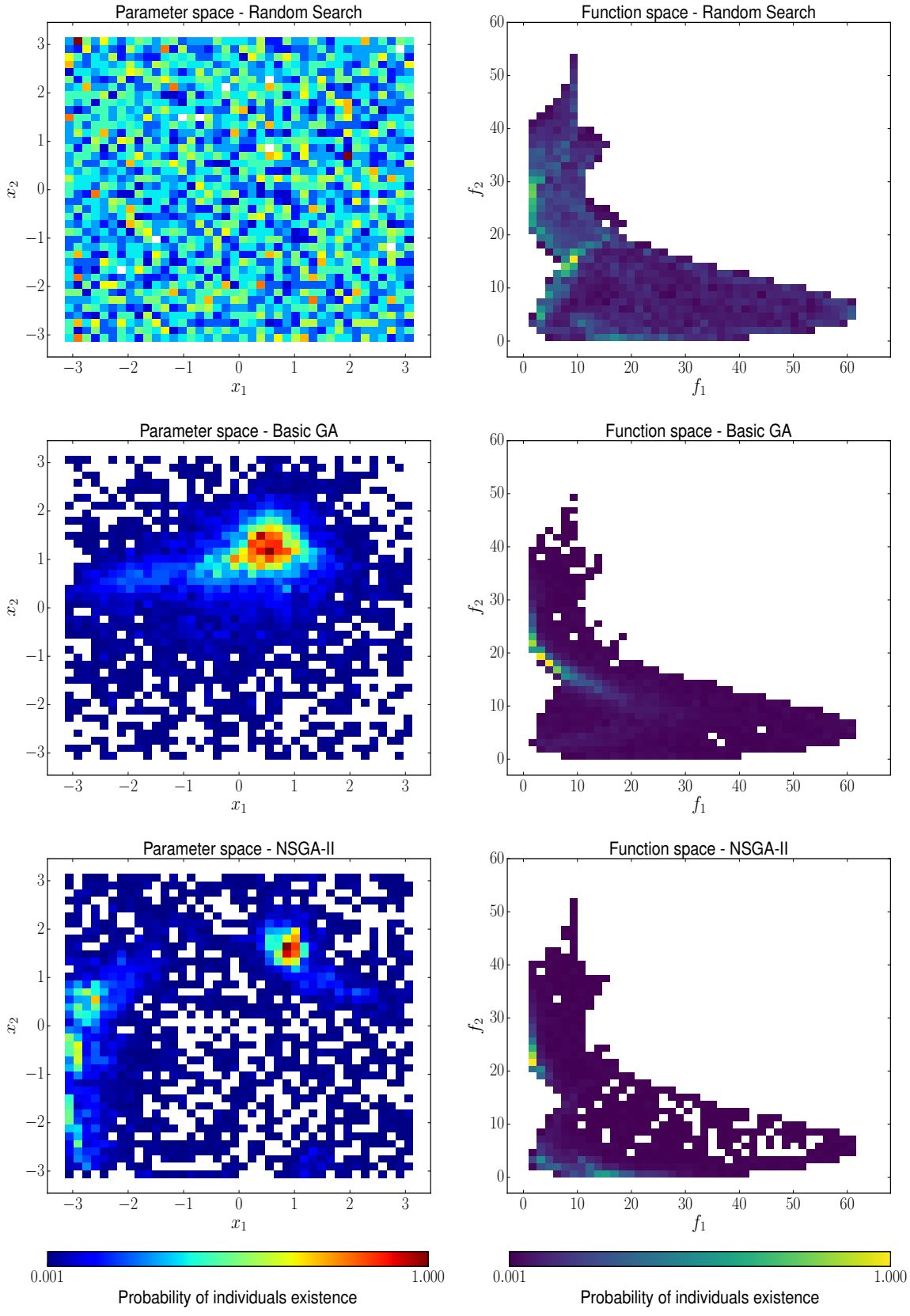

Figure 3.5: Method comparison for the Poloni's function

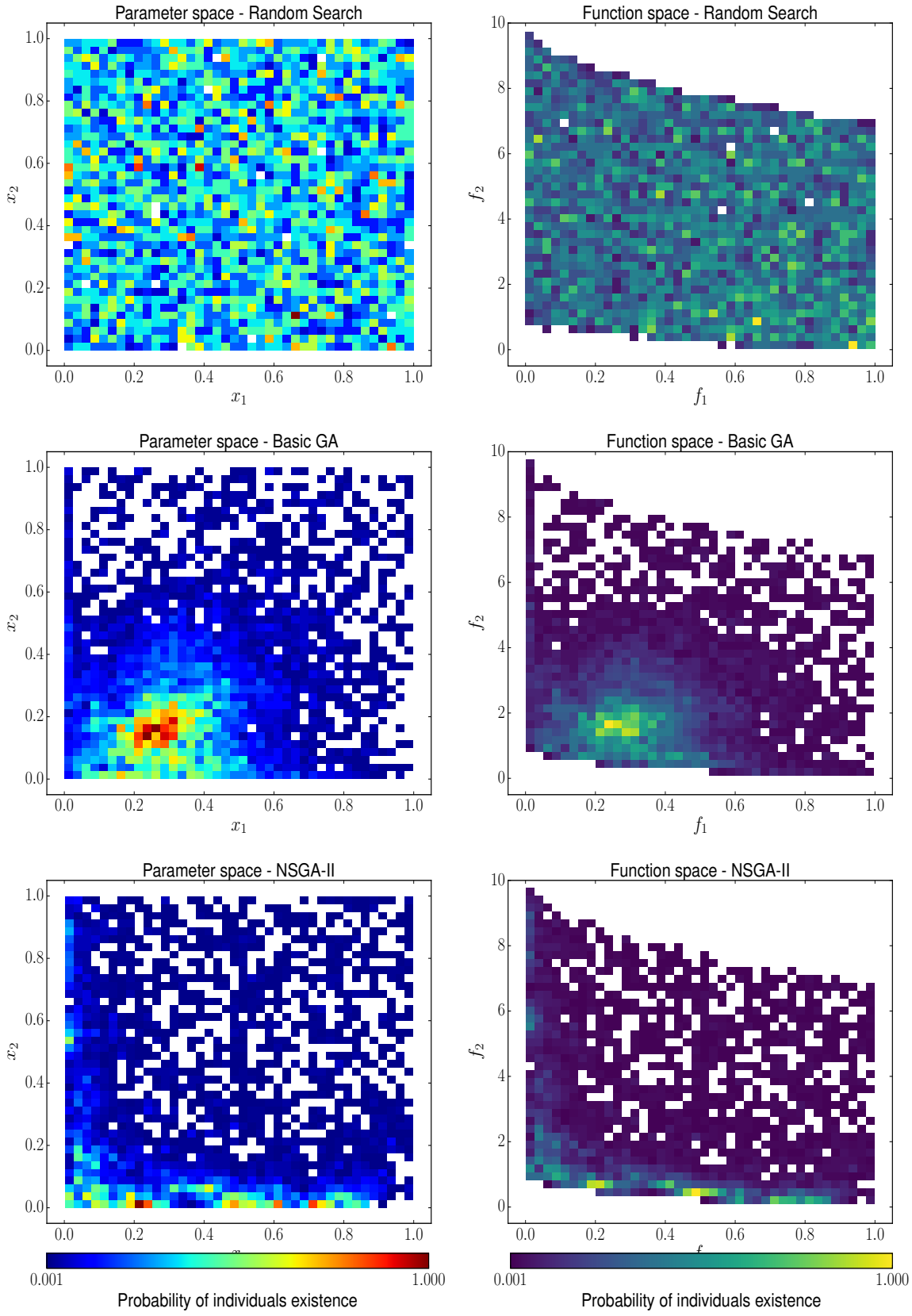

Figure 3.6: Method comparison for the ZDT2 function

It can be seen that for the three functions, the random search cover the whole parameter space uniformly. However, this doesn't mean that the function space is searched well enough to determine the Pareto front (see Figure 3.6). Thus the random search is not valid even for a two variable case because it does not converge nor evolve towards a solution. The basic GA and the NSGA-II points converge to certain areas, although the basic GA does not cover the whole Pareto front as it should (see Figure 3.5) or does not capture it at all (see Figure 3.6). Thus the only method that is really valid for searching in complex parameter spaces and converge to a true Pareto front is the NSGA-II (see Figures 3.4, 3.5 and 3.6). Although this comparison may be seen biased for the different algorithms, the optimization in CFD is usually performed with Monte Carlo methods which are a random search that may not be the most effective methods.

3.1.2 Number of evaluations versus number of generations

Given that NSGA-II is the algorithm chosen for solving multiobjective optimization problems, a deep knowledge of how the dynamics of the system really work is essential to achieve an accurate solution. The NSGA-II depends only on two variables, which are the number of individuals per generation and the limit in the number of generations. The same number of function evaluations (translated in the future to CFD simulations) may be distributed in different combinations of individuals and generations. In order to prove if it is better to have more individuals or generations (for the same number of function evaluations) two measures proposed by [37] are used:

- Diversity metric (Δ): it measures the extent of spread achieved in the solutions by taking the distance between each one of the points that form the Pareto-optimal front. The best value will be zero, having that all individuals are at the same distance one to each other. This metric is only valid for bi-objective optimization, but a Voronoi triangularization may be used for Pareto front in higher dimensions:

$$\Delta = \frac{d_f + d_l + \sum_{i=1}^{N-1} |d_i - \bar{d}|}{d_f + d_l + (N-1)\bar{d}} \tag{3.3}$$

- Convergence metric (Υ): this metric may only be used in cases where the Pareto front may be known, because it is defined as the minimum distance to the true Pareto front:

$$\Upsilon = \min(\text{dist}(PF_{GAi}, \mathcal{PF}_0)) \tag{3.4}$$

The three test functions described before (2.2, 3.1 and 3.2) are used again as a way of knowing how the algorithm really behaves. It was tested with all the cases shown in the Table 3.2, running each one of the possible configurations 20 times in order to capture the trend.

Table 3.2: Generations and individuals combinations tested for performance

	INDIVIDUALS PER GENERATION				
TOTAL FUNCTION EVALUATIONS	**8**	**16**	**32**	**64**	**128**
640	79	39	19	9	4
512	63	31	15	7	3
384	47	23	1	5	2
256	31	15	7	3	*(trivial)*
128	15	7	3	*(trivial)*	

As said before, it must be noted that the first generation of the NSGA-II has twice the size of the rest of populations. This also created some *trivial* cases, because if the number of individuals is N and the total function evaluations are fixed to be $2N$, the first population will cover all the possible function evaluations (so it will be a random search and not an evolutionary computation algorithm).

The values shown in the figures are the mean and standard deviation of the two metrics, having that Figure 3.7 has error bars with the mean μ_Δ and standard deviation σ_Δ of the analyzed cases, while the error bars in Figure 3.8 represent μ_Υ and σ_Υ. The values are very high when compared with the ones obtained in [37] for the same test functions, but as said before the number of function evaluations is very low when compared to those in the paper.

The results in Figure 3.7 for the divergence metric (Δ) show not conclusive results. There is not a clear trend seen in all three functions. The vertical axis has the same limit in all subfigures in order to compare more easily the results. Although it may seem that for a small number of generations the metric has a higher value (having that the best approach will be to have a greater number of generations), the changes are slight and they do not serve to draw any conclusion. It is interesting to see that the Bihn & Korn function yield the same results without any dependence on the combination (reducing only the standard deviation with more evaluations).

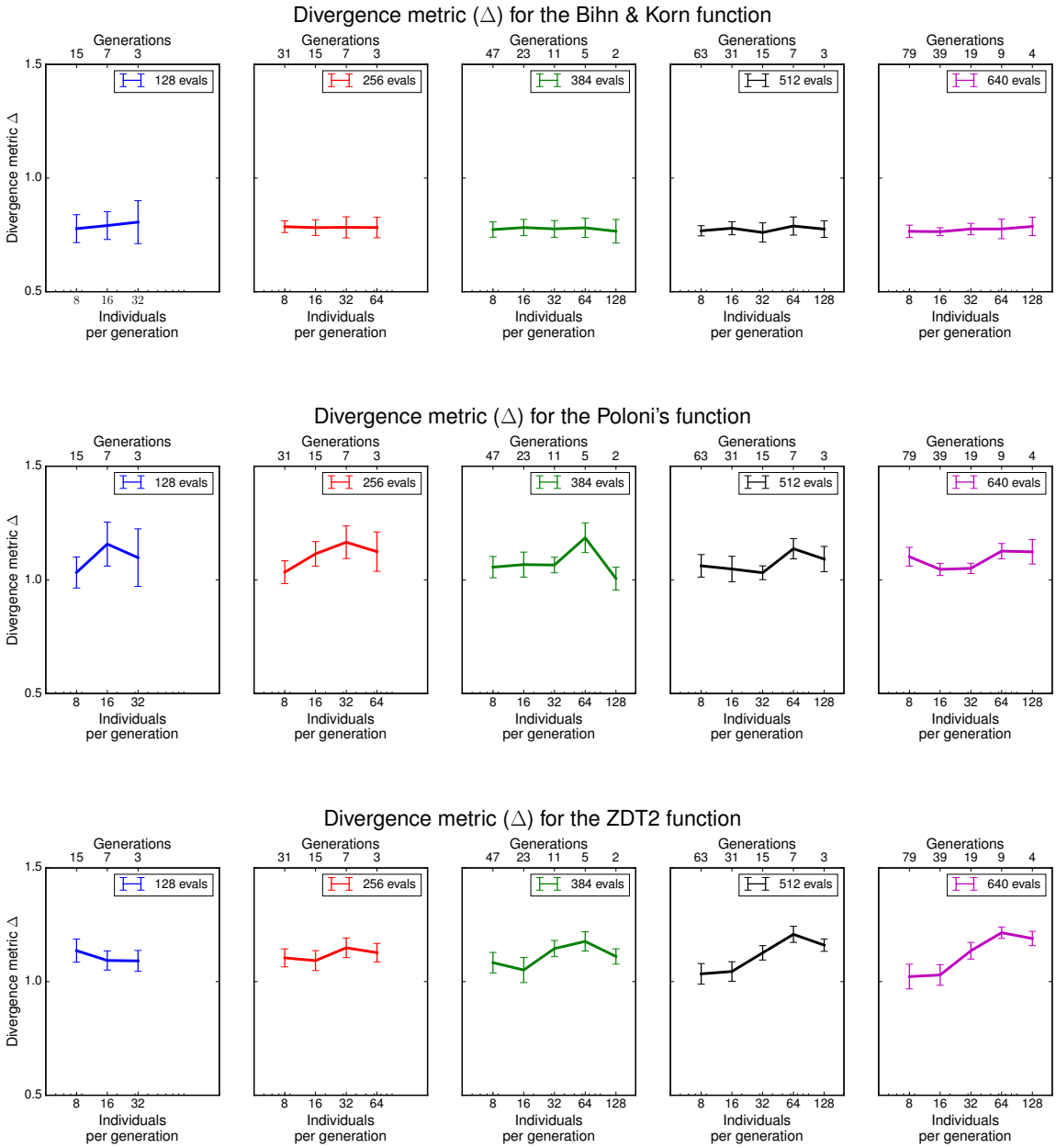

Figure 3.7: Divergence metric for the three different test functions

The convergence metric (Υ) results are shown in Figure 3.8. In this case, there is a clear trend: the convergence to the True Pareto front increased with more individuals per generation that with more generations of smaller populations. It must be noted that the first figure in 3.8 has different vertical limits. However, what is really important is the trend that the metric follow. The results for the ZDT2 function also reveal that the number of generations may be kept small but allowing the population converge to a solution. Thus, creating a population of very large size and evaluating it just for 3 or 4 generations will not give accurate solutions.

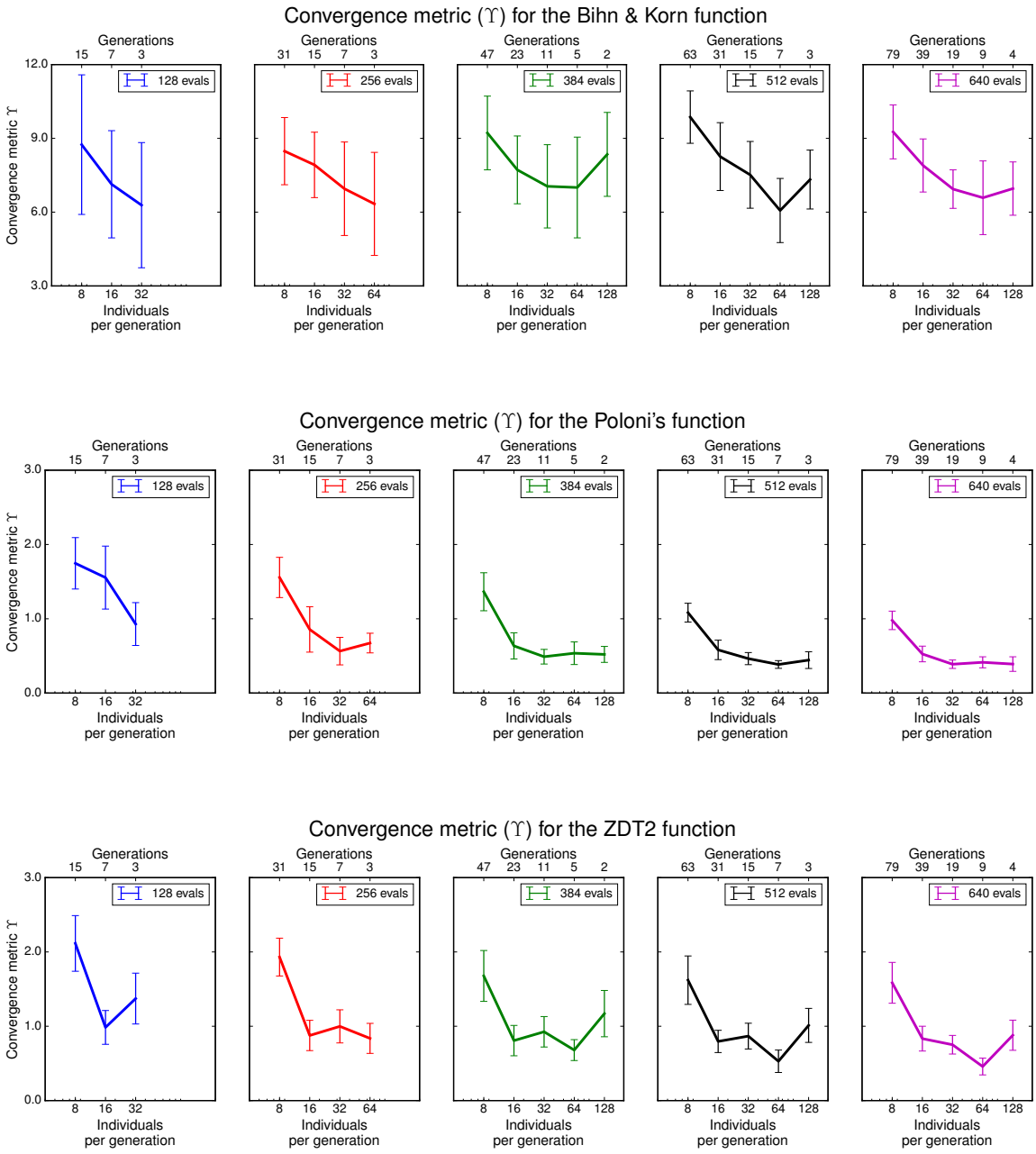

Figure 3.8: Convergence metric for the three different test functions

3.2 CFD cases

In this subsection, different CFD setups will be presented and explained (showing the results in the following section). The code of the NSGA-II was implemented in Python for a two variable and two function case (although it may be easily upgraded to a higher number of variables and objective functions). The different individuals were generated from Python scripts that transformed the variables into a mesh or a configuration file for a case. Each individual has been simulated with OpenFOAM in a parallel fashion (using MPI for multiprocessor parallel computing). The results obtained from the simulation were analyzed with ParaView (in the `pvbatch` mode) or directly with Python to get the fitness value of each individual. With that fitness value, the genetic algorithm computed the next generation, sending it to OpenFOAM as before. All this loop was controlled with Bash scripting: from running the simulations and waiting for them to finish before sending the next set to calling the different scripts for every individual and generation. Only 2D cases were analyzed, although the procedure for 3D cases will be the same.

The workstation used to perform the different cases was a machine with 32 processors (Intel Xeon CPU E5-2650 @ 2.00GHz), 64 Gb of RAM memory and a 275 Gb SSD hard disk. The machine was running under Ubuntu 16.04 LTS, with OpenFOAM version 5.00, ParaView v5.4.0 and Python 3.6.4. The amount of data obtained after the simulation process almost fill the whole solid state drive, so listing the whole code is unfeasible. However, the basic files for performing the different cases are available in GitHub (`https://github.com/jlobatop/GA-CFD-MO`) as well as the code and Jupyter notebooks developed along the process of this thesis.

3.2.1 Suppression of cylinder vortex oscillations

The first one of the cases is the flow around a cylinder. Although cylinders may seem a very simple case, the results obtained for a cylinder may be extrapolated and used in other cases, such as airfoils. Vortex shedding is a phenomenon that happens when the equilibrium flow around a cylinder suffers a Hopf bifurcation (instability caused by background distances) and the flow enters into a new equilibrium which is time-periodic (at the Strouhal frequency) [41]. This is a major problem when mixing structural behavior and flow dynamics, because if the Strouhal frequency matches the natural frequency of the structure, the consequences may be catastrophic because an excited natural frequency usually leads to unstable behavior of the structure [42].

There are a lot of way of suppressing the vortex shed oscillations that are present in the wake of the cylinder. The methods are classified in passive and active, depending if they are looped with some kind of feedback of the state of the system. Vortex shedding may be controlled with a lot of techniques, where the most common ones are magnetic field, rotary oscillations, secondary flow and surface roughness. In this analysis, a passive flow control with a secondary flow will be used. Instead of using a simple blowing jet, a blowing and suction jet will be installed in the rear part of the cylinder. Using a sinusoidal wave that introduces and extracts momentum (instead of a pulse jet that only introduces momentum) avoid issues related to mass conservation in the CFD simulation [43].

Case setup

As said, the flow control will be performed with a sinusoidal wave type of function that depends on two variables: the amplitude (A or v given that the variation in momentum insertion is performed with changes in the velocity) and the frequency (f). The initialization of the flow control is performed once the von Karman vortex shedding is already developed, so the phase shift of the sinusoidal wave is fixed. The objective of the optimization is to minimize the force oscillations in both the horizontal and vertical axis. In order to quantify the oscillations, different approaches were considered. The output of the OpenFOAM solver returns a force-time data file that contains the forces that the cylinder is suffering. In order to capture the amplitude of the data obtained, a Fourier transform was at first considered. The amplitude values are very small so the `FFT` (Fast Fourier Transform) implemented in Python did not work properly. Computing the root mean square (RMS) error was also viewed as a possible approach. The RMS for a data set is defined as [44]:

$$x_{rms} = \sqrt{\frac{1}{n}\left(x_1^2 + x_2^2 + ... + x_n^2\right)} \tag{3.5}$$

$$x_{rms}^2 = \overline{x}^2 + \sigma_x^2 = \overline{x^2} \tag{3.6}$$

where $\bar{x}$ is the mean and σ_x is the standard deviation. The problem that this metric has is that it cannot go to zero because the mean $\overline{x}$ will not go to zero given that there is only one exit in the rear part and the force exerted is not compensated with a front membrane (having that $\overline{x} \neq 0$). Minimizing x_{rms} will try to minimize the oscillations but also it will try to make the force zero.

The metric that better represent the problem is the standard deviation σ. Computing it in the last n cycles (where n depends on the complete cycles that have been captured in the simulation), an oscillatory force will only have $\sigma = 0$ if the amplitude of the oscillations is zero. Thus, the algorithm will try to minimize σ_Y and σ_X for a sum of the pressure and viscous forces.

The genetic algorithm setup was done with 64 individuals in the first generation and just 10 individuals for each other generation. A 6 generation limit was also established. These limits may be very strict but they are chosen for computational resources limitations. A more powerful study may be performed, surely achieving more accurate solutions. The search domain is $A \in [0.01, 2]$ and $f \in [0.01, \pi/2]$. The solver used is `pimpleFoam`, which is an incompressible solver for transient cases. The maximum time of simulation was 150 seconds. Newtonian transport model with laminar turbulence modeling was chosen. The case was done in a non-dimensionalized fashion, with a density value of $\rho = 1kg \cdot m^{-3}$, a viscosity value of $\nu = 1 \times 10^{-5} m^2 \cdot s^{-1}$, and a cylinder diameter of 1 m facing a flow at a velocity of 1 $m \cdot s^{-1}$. This values give a Reynolds number of $Re = 100,000$. Vortex shedding at this Reynolds number happens to occur in a turbulent way. Most of the schemes were chosen as `linear` and the time differentiation scheme was chosen to be `backward`. The real mesh can be seen in Figure 3.9 and the mesh details are shown in Figure 3.10.

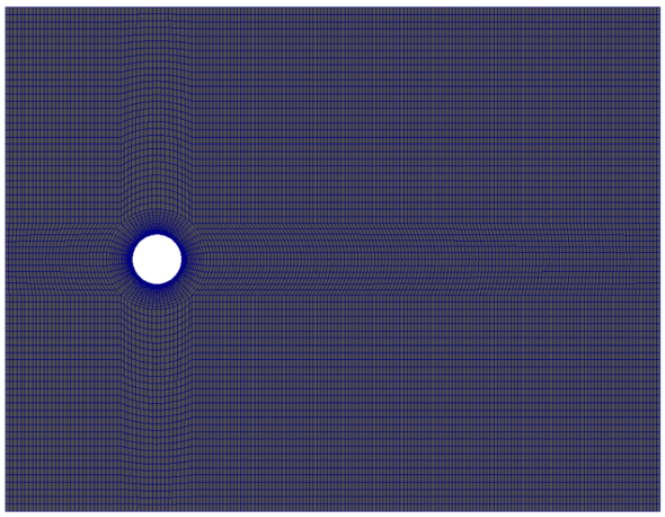

Figure 3.9: Picture of the mesh for the cylinder case

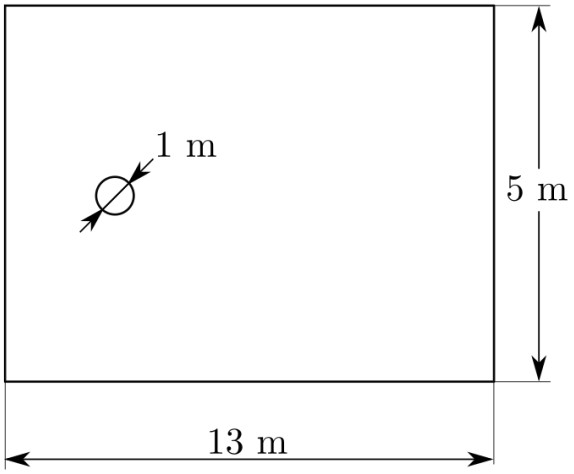

Figure 3.10: Schematics of the mesh for the cylinder analysis

The boundary conditions imposed in the different faces are listed with an exploded view of the mesh.

Table 3.3: Boundary conditions for the cylinder case

Inlet (yellow)	
U	`fixedValue (1 0 0)`
p	`zeroGradient`
Outlet (green)	
U	`zeroGradient`
p	`fixedValue 0`
Flow control membrane (grey)	
U	`uniformFixedValue (tableFile)`
p	`zeroGradient`
Upper/Lower (grey)	
U	`symmetry`
p	`symmetry`
Cylinder (red)	
U	`fixedValue (0 0 0)`
p	`zeroGradient`
Front/Back (white)	
*	empty

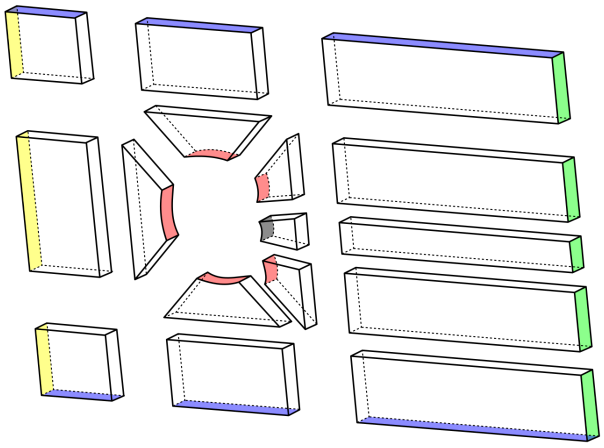

Figure 3.11: Exploded view of the mesh for the cylinder case

Processor convergence

Apart from the typical mesh convergence studies, the processor convergence type of analysis is critical in this highly parallelizable cases. One simulation may be divided for a different number of processors, having that time reaches a minimum in some number of processors. Total simulation time is the sum of the simulation time, case decomposition time and case reconstruction time. For 1 processor there are no decomposition nor reconstruction times, so the total time is the same as the simulation time.

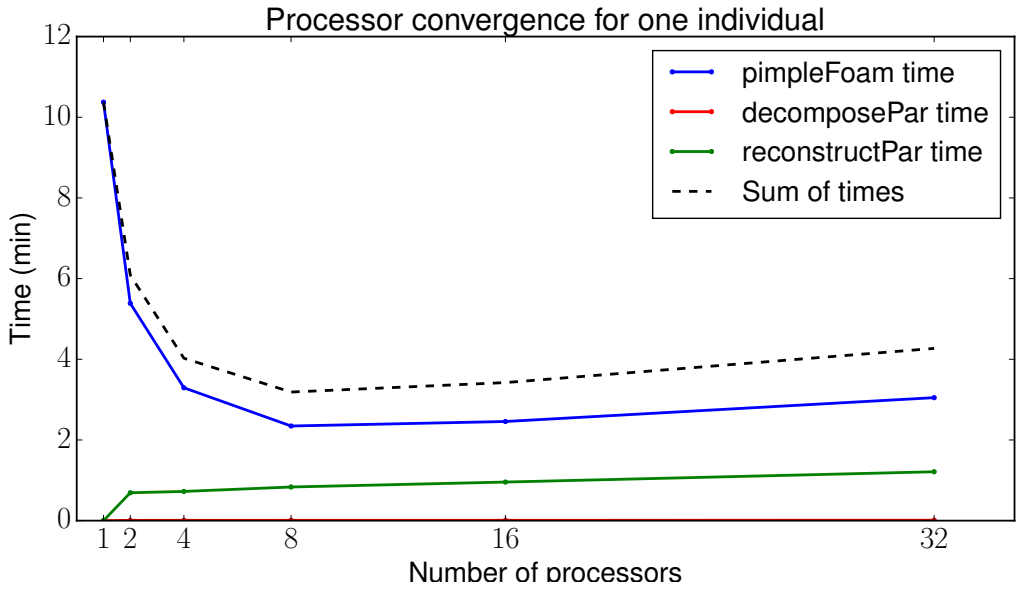

Figure 3.12: Processor convergence for 1 individual

The results of the processor convergence in Figure 3.12 shown that using 8 processor gives the fastest results. However, when applying genetic algorithms or another population-based algorithm, there are a lot of similar simulations that might be performed at the same time. If each individual is computed in 1 processor (and constraining the problem to a 32-processors machine), then 32 individuals may be computed at the same time. If each individual is simulated in 2 processors, only 16 individuals may be run at the same time (so if each population is formed by 32 individuals, 16 must be simulated first and 16 once the latter have finished). If each individual is run in 4 processors, only 8 individuals may be simulated at the same time, sending the population in 4 groups of 8.

If the sum of times from one individual is multiplied by the divisions that must be made to a generation of 32 individuals for simulating the whole population, an estimation of the simulation time is obtained. Nevertheless, the real times were computed and plotted in Figure 3.13, seeing that there is a difference between the actual values and the estimated ones (probably due to the fact that there are computational resources that were not affected in the one individual simulation times).

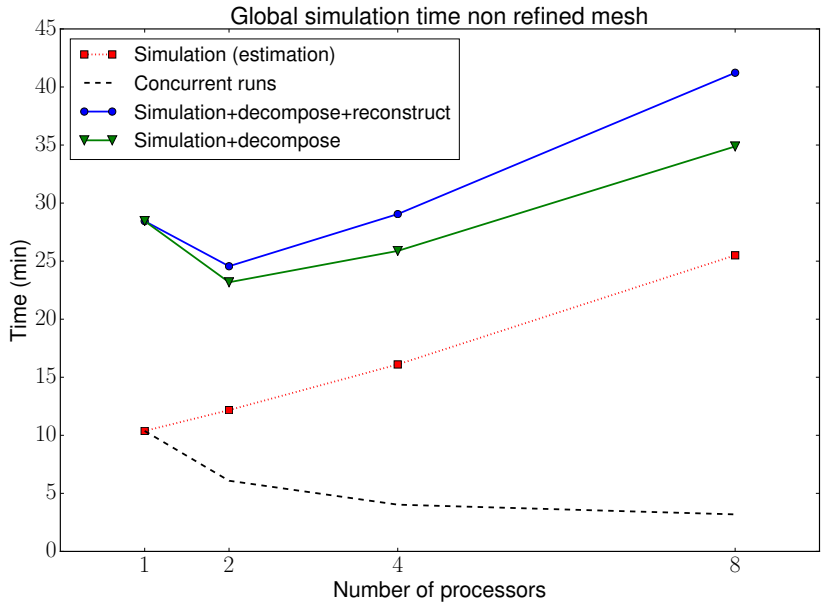

Figure 3.13: Processor convergence for a 32 individual population

The reconstructed time has been separated from the decomposition time because in order to analyze the results it is not strictly necessary to reconstruct the case. This does not make any difference because, for all the cases, the most optimal configuration is to use 2 processors.

3.2.2 Inlet of diffuser geometry

Optimizing the inlet of a diffuser is the first step for increasing the efficiency and performance of the engine as a whole. Inlet diffuser geometry is a well-known problem that has been studied in multiple approaches [45], [46]. However, there are a lot of shapes in which the inlet may be optimized, e.g., if multiple variables are analyzed, the shape may be parametrized with splines or with control nodes. For simplification purposes, in this study the shape was computed with two values: a length (L) and an angle (θ), which are the search space variables. The function space is formed by the total pressure ratio (computed between the total pressure in the outlet of the diffuser and the freestream total pressure) and by the Mach number in the diffuser outlet. Both values will be maximized. The first one seems evident to maximize: total pressure ratio is related to entropy generation and the lower the entropy generation, the better. The maximization of the Mach number in the outlet of the diffuser may be useful in supersonic combustion engines, where there is not turbomachinery and the flow may enter at supersonic speeds to the diffuser.

Case setup

In this case, instead of varying the flow conditions as before, the optimization will be performed to the mesh. Thus, each combination of L and θ will have an associated mesh, having all the meshes with the same boundary conditions of M_∞, p_∞ and T_∞. A combination of the two may be performed, having, for example, L and Ma as the search space variables to maximize both the pressure ratio and the Mach number at the diffuser outlet (having the problem of the moving nose of the SR-71 Blackbird). Both the Mach number and the total pressure ratio are computed in Python with values extracted from Paraview. The genetic algorithm used 32 individuals per generation and 6 generations limit. This simulation was performed with a compressible steady state solver (`rhoSimpleFoam`) with 2500 maximum iterations, where the residuals were stable. The turbulence modeling was done with a $k-\epsilon$ model under Newtonian fluid and perfect gas assumptions. The freestream velocity was $u_\infty = 590\ m \cdot s^{-1}$ which for a temperature of $T_\infty = 216\ K$ yields a Mach number of $Ma = 2$, freestream pressure was $P_\infty = 19930\ Pa$ which corresponds to a altitude of 12 km, viscosity was $\nu = 1 \times 10^{-5} m^2 \cdot s^{-1}$ and the density $\rho = 0.31\ kg\dot{m}^{-3}$. k, ϵ, ν_t and α_t were chosen based on the tutorials in the OpenFOAM library. The Reynolds number for this case will vary depending on both the horizontal length L and the inclination angle θ, but it will be located near the $Re \sim 10^6 - 10^7$ range.

The geometry of the mesh is sketched in Figure 3.14, having some fixed distances for all possible individuals and some variable dimensions: L and θ. Fixed distances will constrain the possible dimensions that the step may have and they are necessary to avoid having a completely undefined problem. Python took each combination of L and θ, created a `blockMeshDict` with the corresponding dimensions and then run the file to get the mesh.

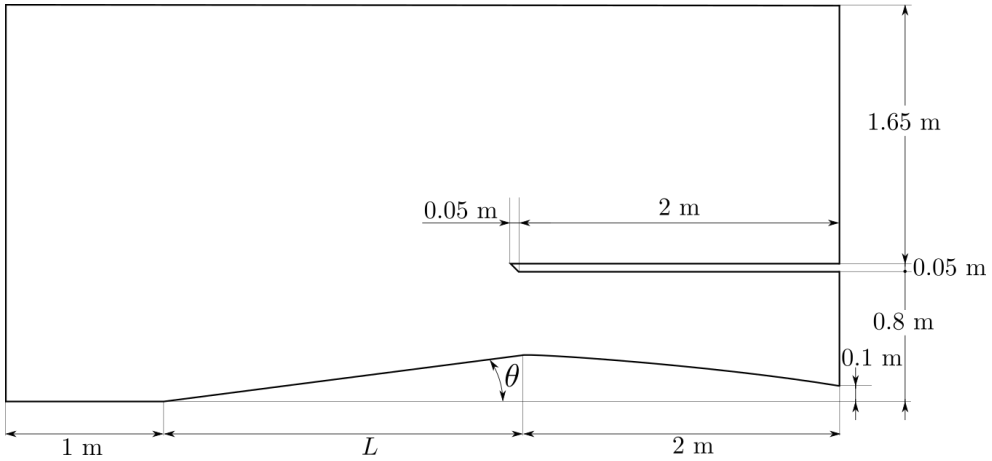

Figure 3.14: Schematics of the mesh for the diffuser mesh

One possible set of values $L-\theta$ returned the mesh shown in Figure 3.15. It can be seen that the grading of the mesh is concentred in some zones while other zones have a more coarse grid. Given that `blockMesh` uses hexahedral blocks, the refinement must be performed along the dimensions of the hexahedral. This is a 2D case so in the spanwise direction z there is only one cell (not shown in the picture).

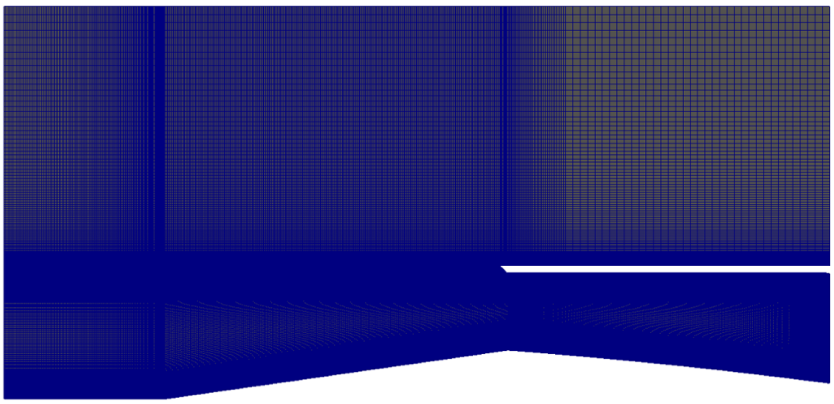

Figure 3.15: Picture of the mesh for diffuser case (random L and θ)

As it has already been mentioned, the fixed dimensions of the mesh will bound the possible values that L and θ take. The heights of 0.1 m and 0.8 m inside the diffuser will constraint the maximum and the minimum value of θ for each L. There is also physical maximum theta given by the attached oblique shock wave theory. It will be independent of L and it will only depend on the value of M_∞, having a θ_{max}^{phys} independent of the length of the step. Finally, a maximum length L_{max} was fixed in order to avoid unrealistic lengths of the step.

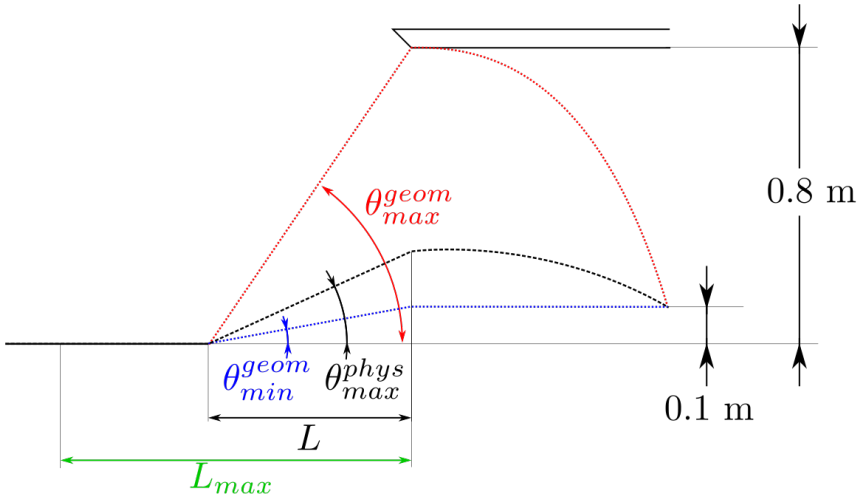

Figure 3.16: Constraints for the diffuser case

The search space is located in the grey zone (Figure 3.17):

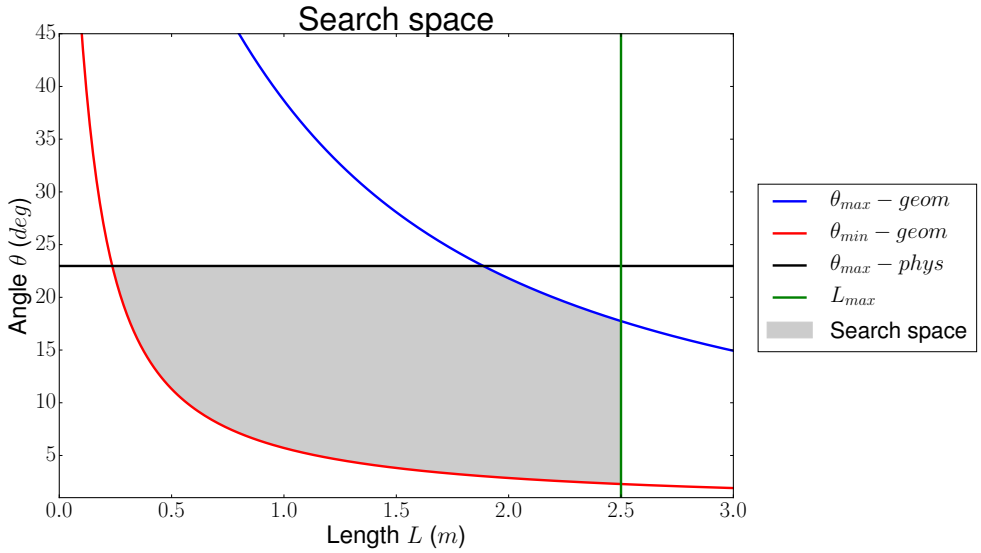

Figure 3.17: Search space for the diffuser analysis

The same procedure to determine the best combination of processors as the one followed in the cylinder was done for this case. Here, the optimum number of processors was 16, using a decomposition in 16 subdomains for each one of the cases. Mesh convergence was also performed, taking as most optimum values the ones obtained from a high-resolution mesh simulated with the same conditions as the case here presented. One consideration when selecting the mesh size is the time for performing each simulation, given that the simulation of one generation of 32 individuals took 2 hours, without including the fitness evaluation.

The mesh is composed by 8 blocks with the sizes described in Figure 3.14. The boundary conditions are shown in Figure 3.18 and the type and value for each one of the variables is collected in Table 3.4.

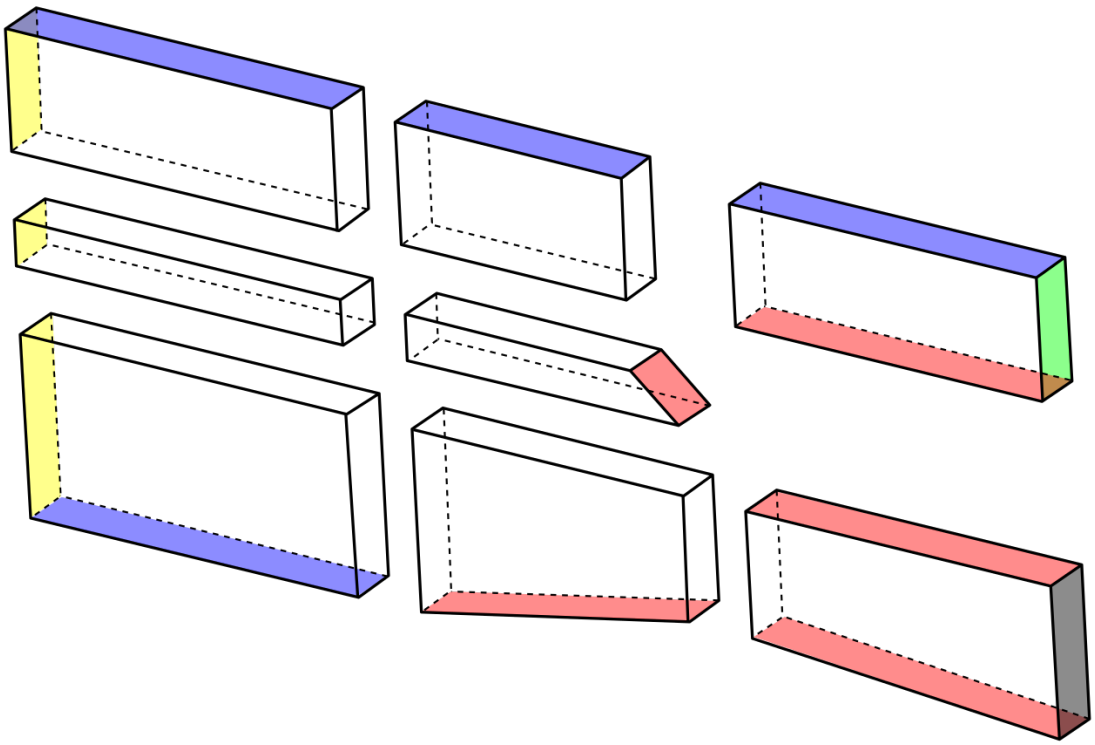

Figure 3.18: Exploded view of the mesh for the diffuser case

Table 3.4: Boundary conditions for the diffuser case

	Inlet (yellow)
U	`fixedValue (590 0 0)`
p	`fixedValue 19930`
T	`fixedValue 216`
k	`turbulentIntensityKineticEnergyInlet 2`
ϵ	`turbulentMixingLengthDissipationRateInlet 200`
ν_t	`calculated 0.00001`
α_t	`calculated 0.01`
	Outlet (green)
U	`zeroGradient`
p	`zeroGradient`
T	`zeroGradient`
k	`inletOutlet 2`
ϵ	`inletOutlet 200`
ν_t	`calculated 0.00001`
α_t	`calculated 0.01`
	Engine (grey)
U	`zeroGradient`
p	`zeroGradient`
T	`zeroGradient`
k	`inletOutlet 2`
ϵ	`inletOutlet 200`
ν_t	`calculated 0.00001`
α_t	`calculated 0.01`
	Wall: cowl & axis (red)
U	`fixedValue (0 0 0)`
p	`zeroGradient`
T	`zeroGradient`
k	`kqRWallFunction 2`
ϵ	`epsilonWallFunction 200`
ν_t	`nutkWallFunction 0.0`
α_t	`alphatWallFunction 0.01`
	Upper/Lower (blue)
U	`inletOutlet (590 0 0)`
p	`zeroGradient`
T	`zeroGradient`
k	`inletOutlet 2`
ϵ	`inletOutlet 200`
ν_t	`calculated 0.00001`
α_t	`calculated 0.01`
	Front/Back (white)
*	empty

3.2.3 Airfoil design

The most typical use of optimization algorithms in CFD techniques is the design of airfoils. However, the classical approach is done via adjoint shape methods, achieving the most optimum airfoil by varying the mesh shape. Other approaches try to adjust the different splines that form the airfoil applying then panel methods theory. In this study, the application of genetic algorithms and computer fluid dynamics to achieve the most optimal airfoil will be addressed.

As said there are a lot of way of parametrizing one airfoil. There are also families of airfoils described by some number, such as the NACA family. But for a first approach, it will be interesting to begin with a more straightforward parametrization. Joukowsky airfoils are one of those sets that is usually used for potential flow but they may be compared with some NACA airfoils, having that the transformation is more than a mathematical construction [47].

Joukowsky transformation is a mathematical operation that takes one circle in the ζ complex plane and transforms it to an airfoil in the z plane. The position of the circle and its radius will determine the shape of the airfoil, so the way of parametrizing the airfoil will be the μ_x and μ_y as the coordinates for the center and the radius R. Joukowsky transform is:

$$z = \zeta + \frac{1}{\zeta} \tag{3.7}$$

where $z = x + y\ i$ and $\zeta = \chi + \eta\ i$.

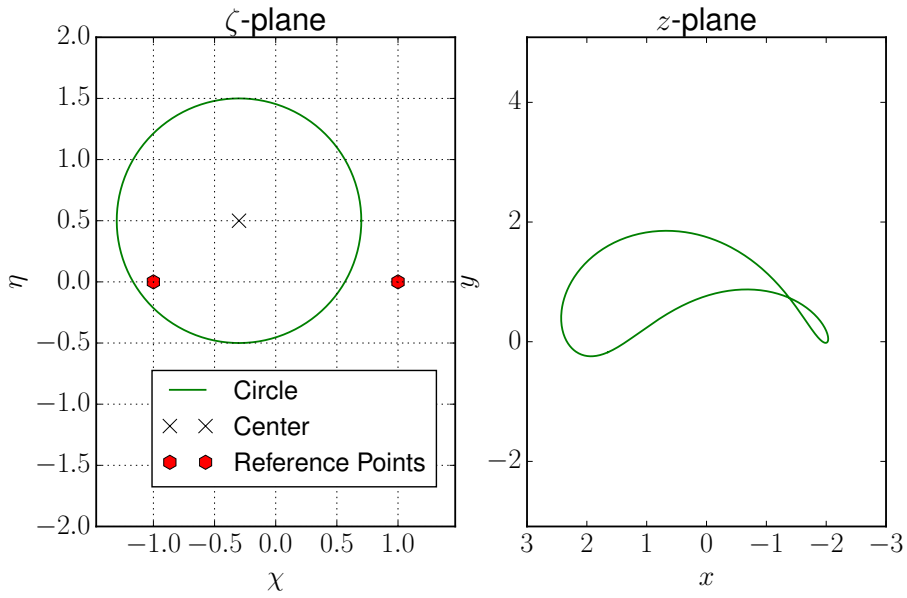

Figure 3.19: Example of a Joukowsky airfoil with non valid μ_x, μ_y and R

Taking a closer look to Joukowsky airfoils, they only have realistic shapes if the circle in ζ intercepts the point $\zeta = (-1, 0)$ or the point $\zeta = (1, 0)$ (as it can be seen in Figure 3.19). This way the radius will be determined with the position of the center of the circle, having $R = f(\mu_x, \mu_y)$. Thus the search variables that form the parameter space are μ_x and μ_y, which are the coordinates of the center of the circle in the ζ plane in the χ and η.

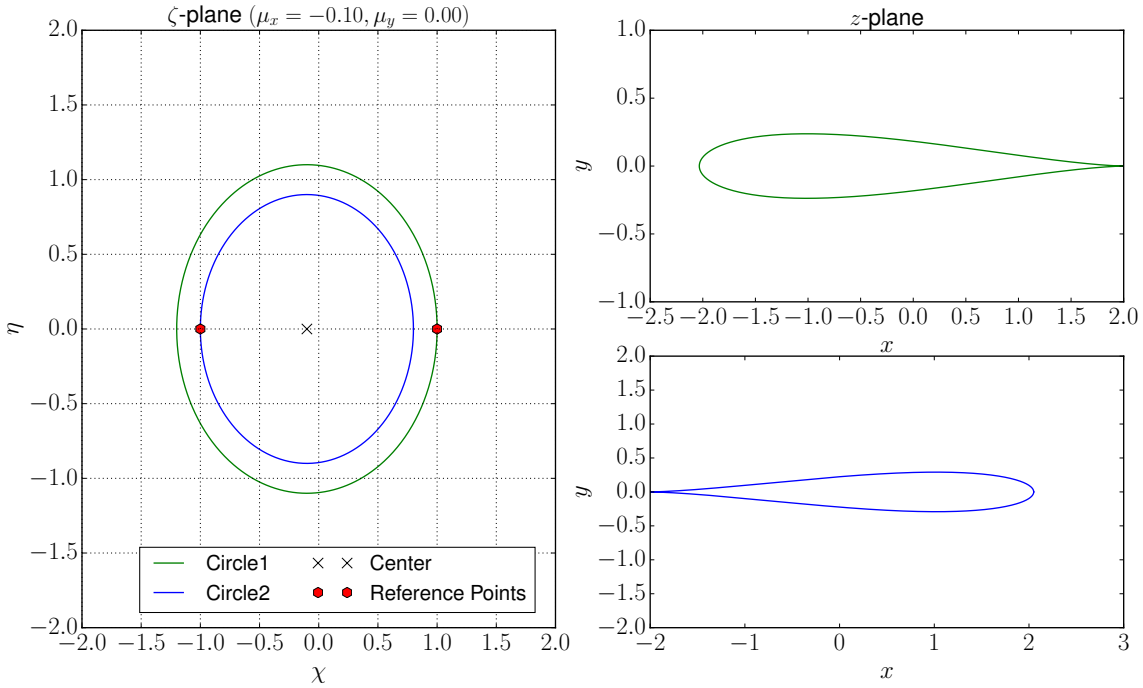

Figure 3.20: Example of a Joukowsky airfoil defined just with μ_x and μ_y

Two simulations were carried out, selecting different objectives to define the function space. These objectives can be easily changed by simply varying the `fitness.py` script, where the objectives are selected and saved to select the next generation:

- Maximization of lift and minimization of drag: these are the most common objectives when trying to optimize an airfoil. Nevertheless, it is impossible to get the optimum value of both at the same time.

- Maximization of lift-to-drag ratio and maximization of the airfoil area: this kind of objectives also try to maximize lift and minimize drag (mixed in the L/D ratio) while keeping the inner area of the airfoil the bigger as possible (provided that airfoils carry the fuel in the wings it will be interesting to maximize it).

Case setup

In this case, the individuals will have the characteristics represented in the mesh. The shape of the airfoil will depend on the value of μ_x and μ_y, which are the search variables. The search space will be bounded inside the range $\mu_x \in [-0.3, -0.1]$ and $\mu_y \in [0.0, 0.15]$. The simulation was performed with an incompressible solver in a steady-state fashion (`simpleFoam`) for at least 2500 iterations. The turbulence was modeled with the Spalart-Allmaras RAS model and a Newtonian flow was assumed. The density was chosen at sea level ($\rho = 1.225\ kg \cdot m^{-3}$) as well as the viscosity value of $\nu = 1.789 \times 10^{-5}\ m^2 \cdot s^{-1}$. The airfoil faced a free stream velocity of 30 $m \cdot s^{-1}$ with turbulence values as selected for a similar OpenFOAM tutorial case ($\tilde{\nu} = 0.14$ and $\nu_t = 0.14$). This corresponds to a Reynolds number of $Re = 2.05 \times 10^6$.

The mesh geometry consisted of a rectangle with a circular inlet that encased the airfoil. Joukowsky airfoils may have a wide range of possible chords, but for this analysis, the chord has been normalized to a value of one. The other dimensions of the mesh are with chosen with respect to that chord value and choosing them far enough so boundary conditions do not affect the flow and close enough so the computational domain is not too big. The refinement of the boundary layer was done in a very generic way, provided that the airfoil may acquired a great variety of shapes. However, and taking advantage of the fact that the mesh was done with hexahedral blocks, some boundary layer inflation was done (with simple expansion ratio gradings).

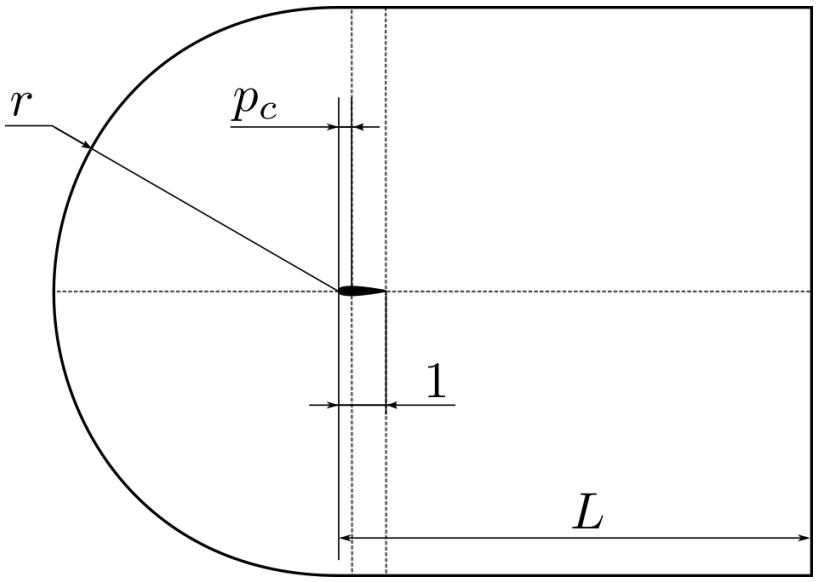

Figure 3.21: Mesh schematics of the airfoil

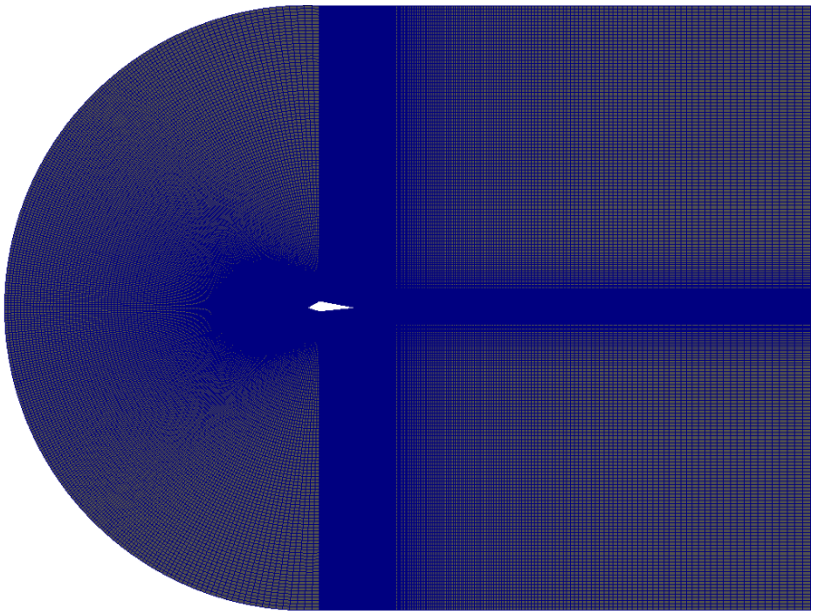

Figure 3.22: Mesh for the airfoil optimization case

The process of creating the airfoil was automatically done with Python, but the progress of introducing the points of the spline that represents the airfoil is shown in Figures 3.23 and 3.24, where the airfoil is delimited by the hexahedral blocks, converting it to arcs that represent the value of μ_x and μ_y.

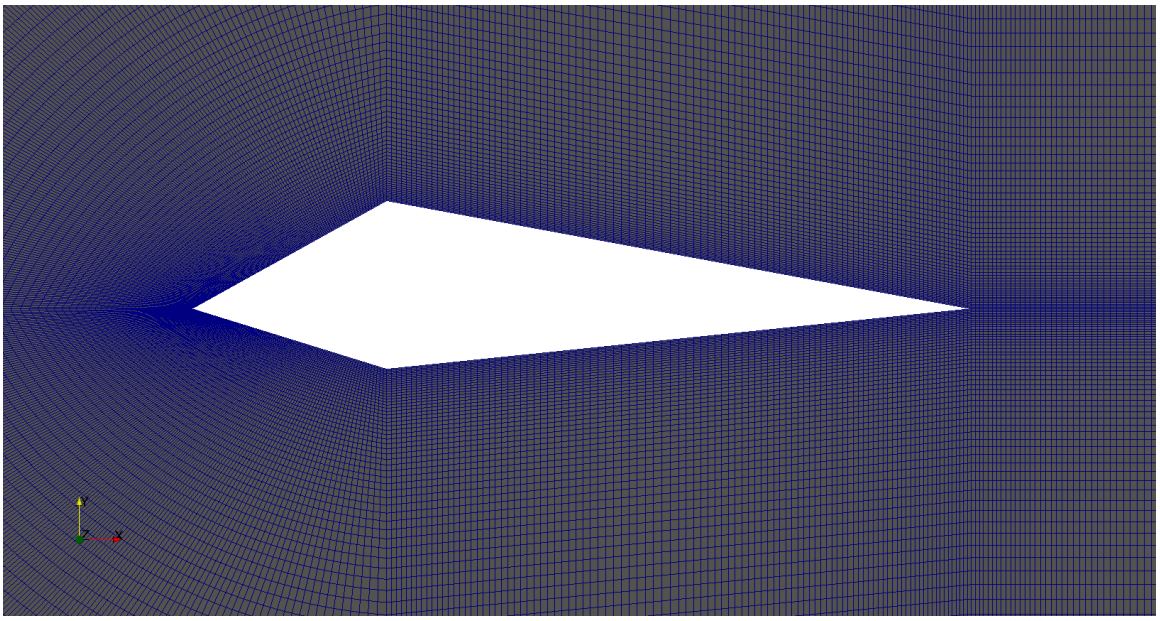

Figure 3.23: Mesh picture with the uncomputed airfoil

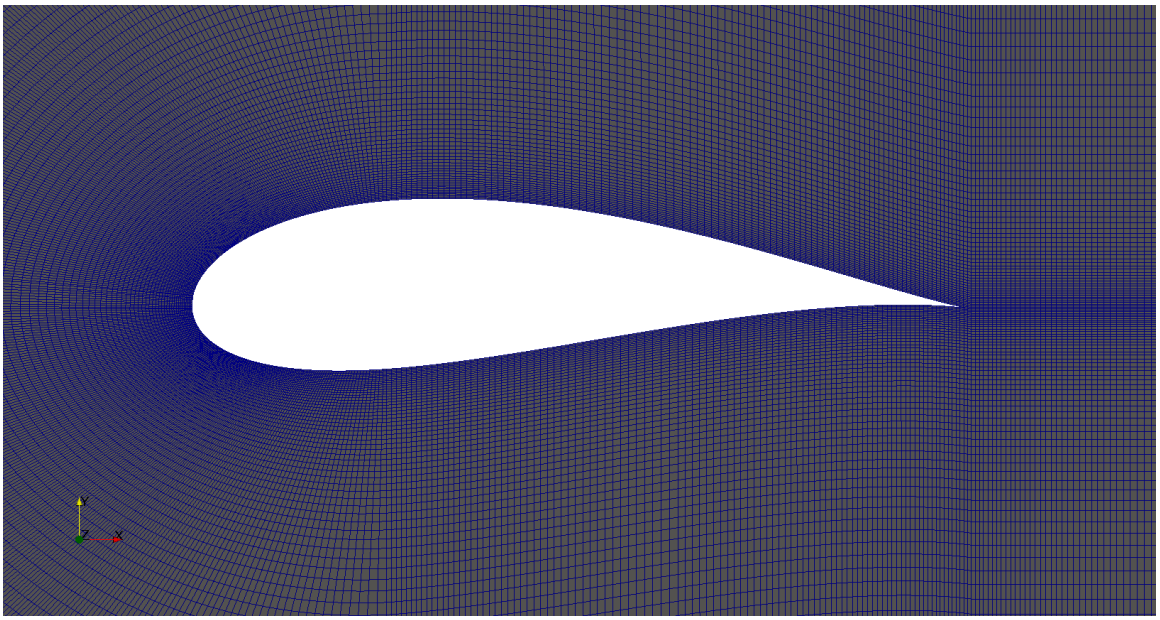

Figure 3.24: Computed airfoil in the mesh

The boundary conditions used for the two optimizations performed are the same. A freestream enters the domain through the yellow face and leaves it through the green one. Given that the angle of attack α is zero, there is no need of any special refinements or critical considerations about mass conservation in the domain (whereas doing angle of attack simulations will require a refined mesh and take into account the possible problems that boundary conditions may have).

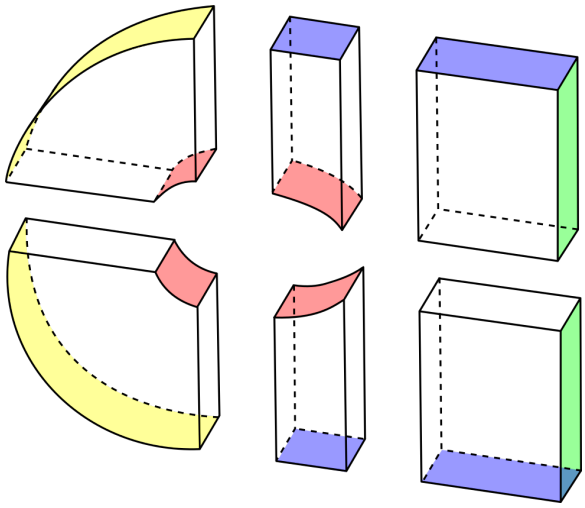

Figure 3.25: Airfoil exploded view of the mesh with the boundary conditions

The boundary conditions are listed in Table 3.5:

Table 3.5: Boundary conditions for the airfoil cases

Inlet (yellow)	
U	`freestream (30 0 0)`
p	`freestreamPressure`
$\tilde{\nu}$	`freestream 0.14`
ν_t	`freestream 0.14`
Outlet (green)	
U	`freestream (30 0 0)`
p	`freestreamPressure`
$\tilde{\nu}$	`freestream 0.14`
ν_t	`freestream 0.14`
Airfoil (red)	
U	`noSlip`
p	`zeroGradient`
$\tilde{\nu}$	`fixedValue 0.0`
ν_t	`nutUSpaldingWallFunction 0.0`
Upper/Lower (blue)	
U	`freestream (30 0 0)`
p	`freestreamPressure`
$\tilde{\nu}$	`freestream 0.14`
ν_t	`freestream 0.14`
Front/Back (white)	
*	empty

Chapter 4

Results

In this chapter, the results of the four optimizations explained in the previous methodology discussion will be presented. Thus, this chapter will be divided into different sections for each case. First, both the search space and the function space will be presented, showing where the individuals are as generations advance. Given that populations are too large to show all individuals, a sample from the population will be chosen in order to show the convergence of the process to a set of non-dominated solutions. Just the results will be shown, the correspondent discussion will be done in the following sections.

Supression of cylinder vortex oscillations

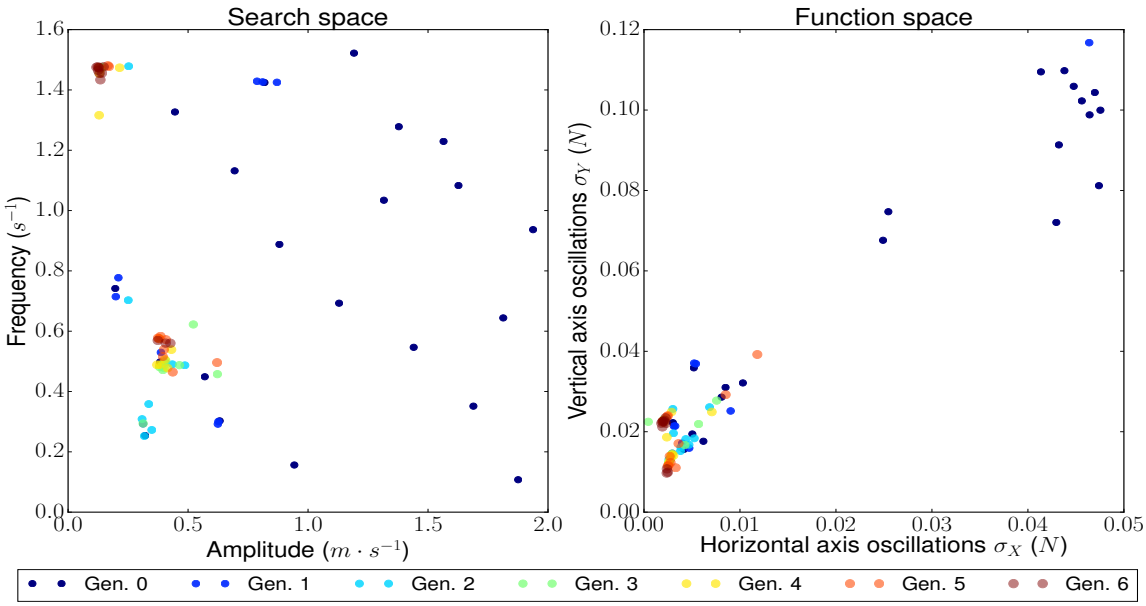

Figure 4.1: Evolution of the generations for the cylinder case

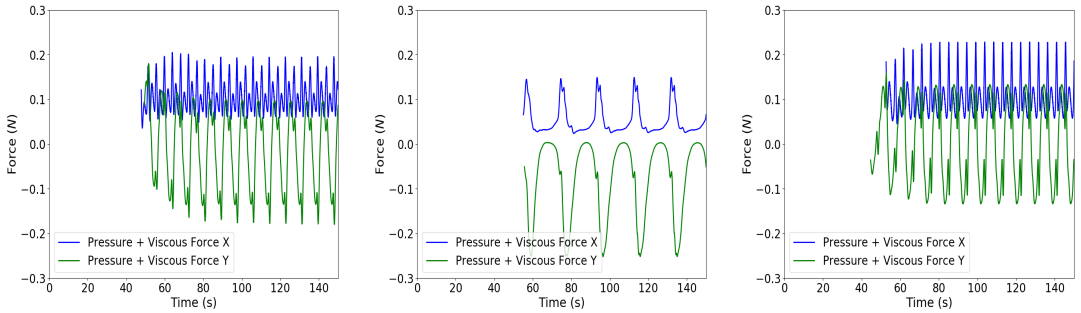

Figure 4.2: Sample of the initial generation for the cylinder case

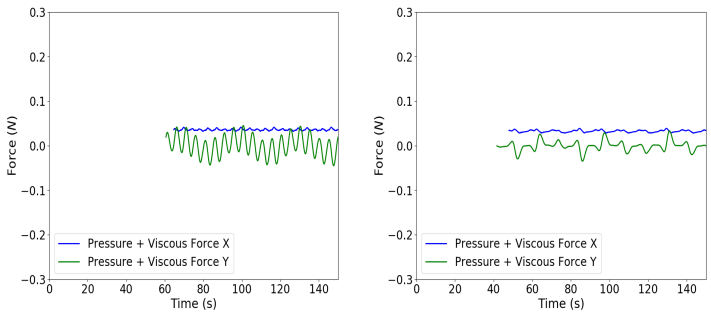

Figure 4.3: Sample of the final generation for the cylinder case

Inlet of diffuser geometry

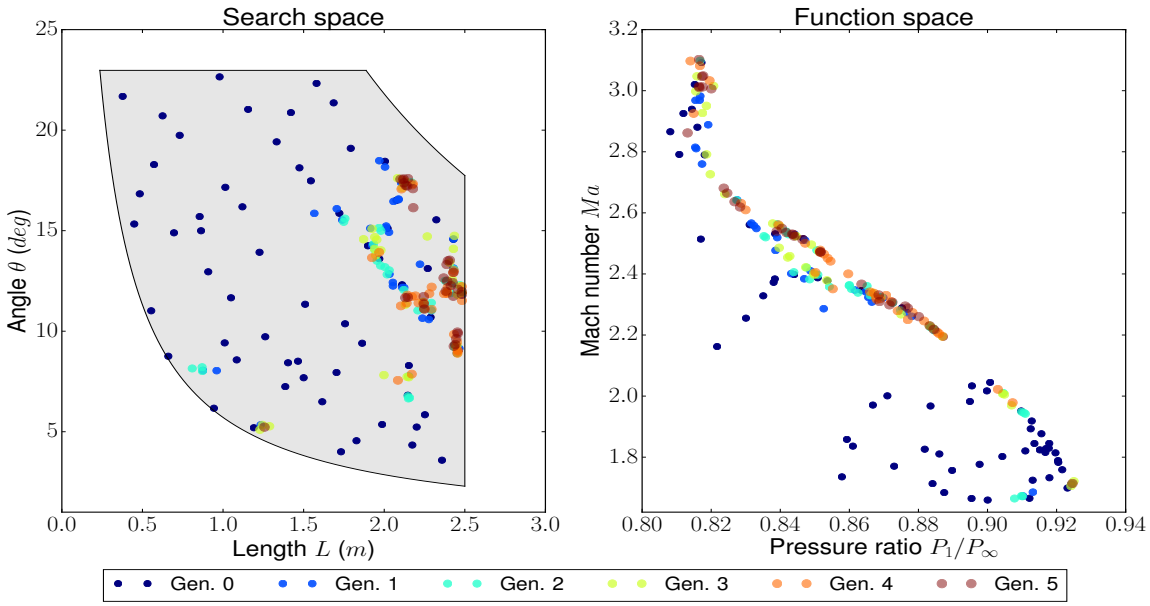

Figure 4.4: Evolution of the generations for the diffuser case

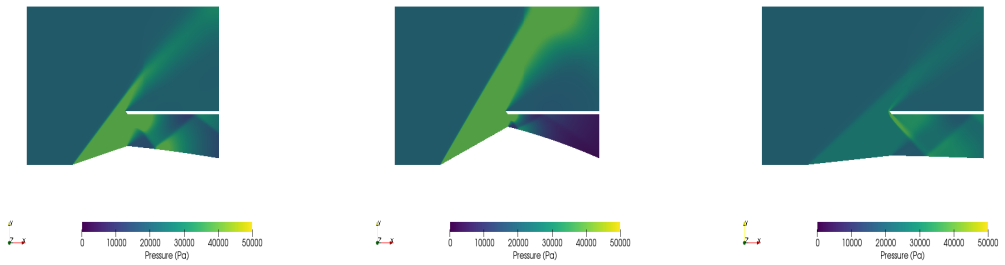

Figure 4.5: Sample of the initial generation for the diffuser case

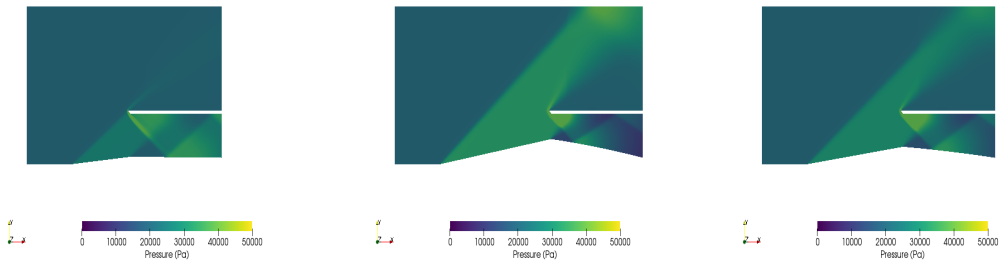

Figure 4.6: Sample of the final generation for the diffuser case

Airfoil design: lift maximization and drag minimization

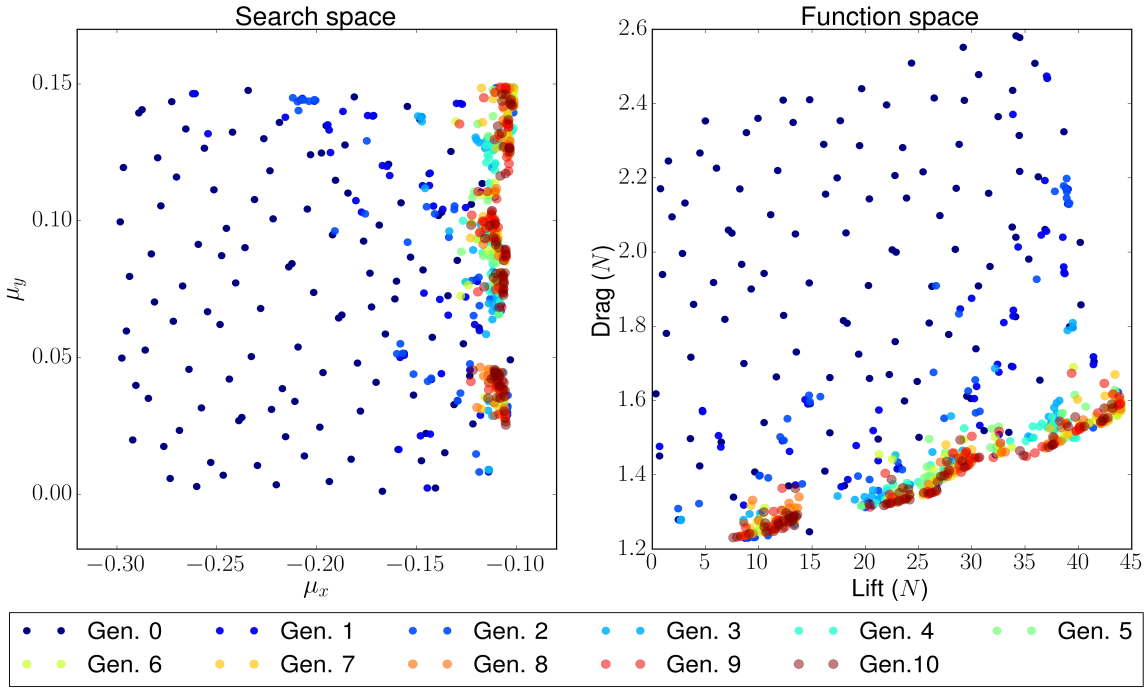

Figure 4.7: Evolution of the generations for the airfoil case I

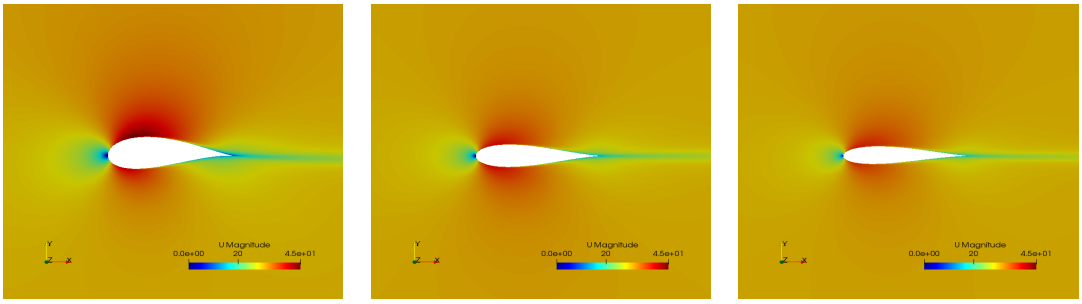

Figure 4.8: Sample of the initial generation for the airfoil case I

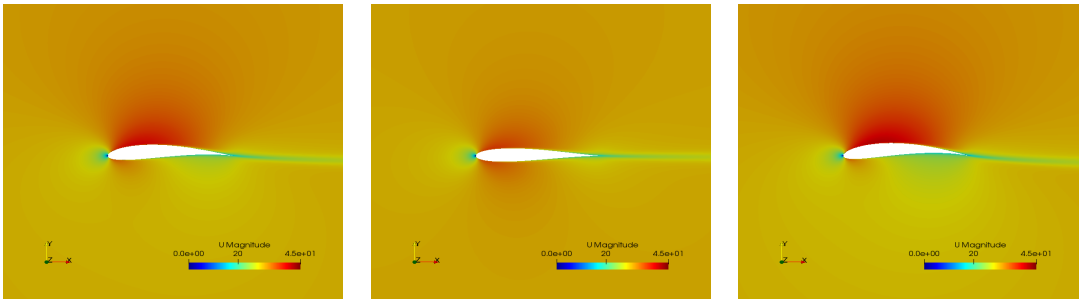

Figure 4.9: Sample of the final generation for the airfoil case I

Airfoil design: lift-to-drag ratio and area maximization

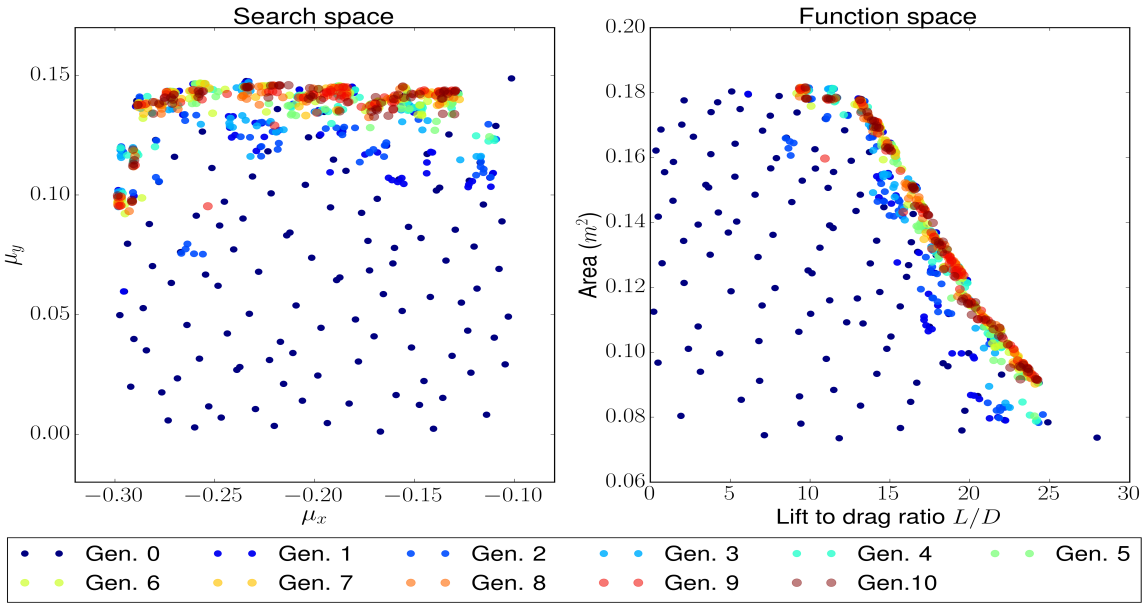

Figure 4.10: Evolution of the generations for the airfoil case II

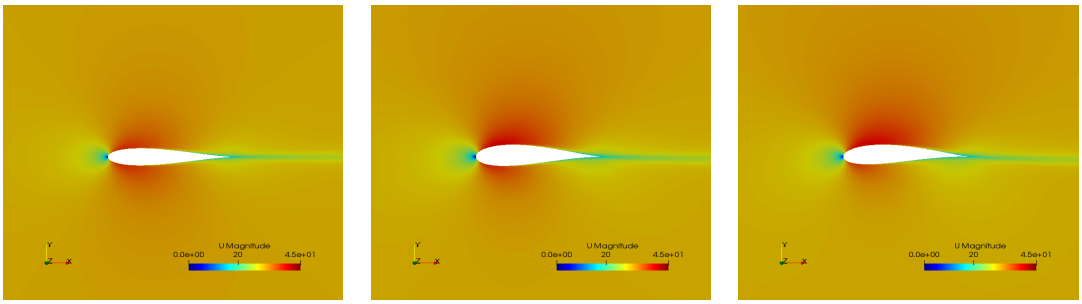

Figure 4.11: Sample of the initial generation for the airfoil case II

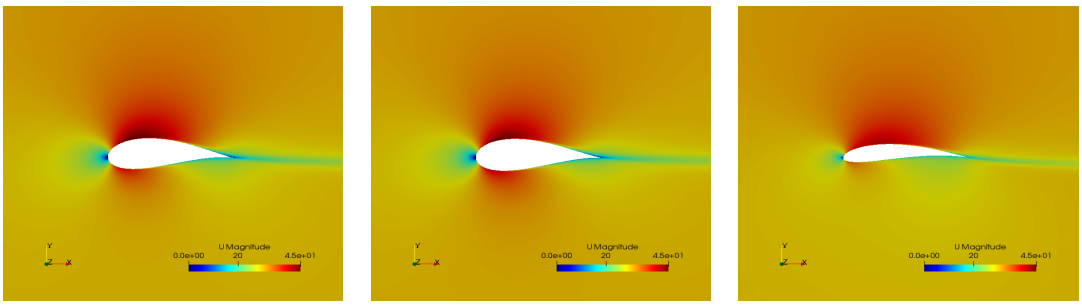

Figure 4.12: Sample of the final generation for the airfoil case II

Chapter 5

Discussion

Once the results of all the four optimizations performed here have been shown, they will be briefly discussed. The four cases showed how the individuals of the populations move throughout the search space with its corresponding position in the function space. Also, some sample individuals of the population are presented in order to see how the individuals behave and how they evolve towards a solution.

Supression of cylinder vortex oscillations

The search space is too large and, due to time restrictions, the number of individuals simulated did not cover evenly the search domain. Also, the solution converges to two points instead of a full Pareto front. The algorithm, however, behaves as it should, because it moves the points towards the zero oscillations points, which is not reached but it is close to some solutions.

In Figure 4.2 some individuals from the first population have been randomly chosen and the figures with its oscillations are shown. The blue line represents the sum of the pressure and viscous force in the x axis while the green line is the pressure and viscous force in the y axis. The horizontal axis represents the time evolution while the vertical axis is the force in Newtons. The last point is chosen as a reference to pick a number of cycles that realistically represents the oscillations suffered by the cylinder. As it can be seen, there is a great variety of amplitudes and frequencies in the search space.

In Figure 4.3 the force plots for the two points where the algorithm converges are shown. The oscillations have been damped and the amplitude is smaller than the amplitudes of the initial population which was the objective of the algorithm. Thus it can be said that, although not being a Pareto front, the solutions of the algorithm have reduced the amplitude of the oscillations.

Inlet to diffuser geometry

In this case, Figure 4.4 shows not only the evolution of the individuals but also the search space where they are bounded in. In contrast with the cylinder analysis, the function space shows a neat and well-defined Pareto front. The individuals may have a large pressure ratio and a low Mach number or a large Mach number and a small pressure ratio: the trade-off between the two variables has been captured.

In Figure 4.5 some randomly chosen individuals from the initial generation are chosen. It can be seen that there is a great variety of L and θ possible values. The cases shown do not look as optimal, and there are shapes (as the middle one) that look even unrealistic. However, in Figure 4.6 it can be seen how the individuals have evolved towards a most optimal solution. The left image shows the expected solution where the shock wave hits the cowl of the engine. However, there are plenty of other solutions that are not dominated, having higher velocities for some cases and higher pressure ratios for others.

Airfoil design: lift maximization and drag minimization

Given that the airfoil setup was quicker than the other cases, much more individuals have been simulated (see Figure 4.7). The search space is well covered and the points are spread enough to see where the optimal region is. Points eventually converge to the location of that optimal area, creating some unwanted clustering zones. This is translated in the function spaces as a well-defined Pareto front that captures the trade-off between lift and drag, given that both are related.

In Figure 4.8 it can be seen that there is a wide range of possible airfoils obtained through the Joukowsky transform. This case was initialized with a Sobol sampling, which is quasi-random low discrepancy initialization that tries to cover the whole search space as much as possible. This initialization will be useful for the second optimization performed to the airfoil case, given that the same initial population was used to reduce as much as possible computational time. This type of initialization covers better the search space and, while for a genetical algorithm not applied to CFD this will yields the same results after a large enough number of generations, in this case, it is very interesting to reduce the simulations as much as possible. The wide range that the airfoils in the first generation have airfoils converge to more aerodynamic shapes (see Figure 4.9) with a low thickness and slightly cambered. This shape enhances as much as possible the lift while affecting the less possible the drag.

Airfoil design: lift-to-drag and area maximization

Finally, Figure 4.10 shows the search space (which is the same as the previous one) and the function space with a completely different Pareto front as the one seen in Figure 4.7. Although the first generation was formed by the same individuals, the selection pressure based on assigning the fitness depending on different objectives yield completely opposed results. The Pareto front is well captured and it shows the most optimal solutions for maximizing the L/D ratio and the area.

Figure 4.11 show randomly chosen individuals from the initial generation with some possible airfoils (having that individuals from Figure 4.8 also are possible individuals from the first generation for the airfoil case II). The results of the final generation (see Figure 4.12) are way different from the results of Figure 4.9. Here the thickness is (in general) larger than in the previous case. However, there are individuals that have a smaller thickness than other individuals from the population - but being still thicker than the individuals in Figure 4.9.

Chapter 6

Conclusions and further developments

As it can be seen, genetic algorithms are a great approach to all kind of optimization problems: from CFD cases with multiobjective optimization to single objective optimization of other engineering systems. The results obtained show that this approach is (although time-consuming) efficient compared with other approaches. Genetic algorithms can be not very efficient, computationally speaking. But they are the only approach to some problems as global optimization because methods like gradient search only locate local minimum. Genetic algorithms may also be applied to black-box functions, where only an evaluation from the function is required.

The results shown in Figures 4.4, 4.7 and 4.10 have the Pareto front correctly captured with the NSGA-II. However, there are zones in the Pareto front that are not correctly covered, leaving gaps in the non-dominated set and having areas with sparse individuals combined with very populated areas. There are different ways to improve these behaviors and before applying the algorithm to other cases, a refinement should be done to eliminate these discrepancies [48], [49]. However, these problems may be due to a little number of individuals and/or generations, having that larger numbers will give a more detailed and evenly distributed Pareto front.

The algorithm has proved to be robust in the sense that if there is a function space without a Pareto front, it will converge to the most optimal solutions, as done in the cylinder case. Nevertheless, this lack of Pareto front may also be due to a small population size. Cylinder case also requires a more in-depth analysis to extract more information from the data, such as the frequencies of the oscillations. With this detailed analysis, other parameters may be chosen for achieving better performance in the optimization and smaller oscillations in the flow.

Complex search spaces have been also considered within this analysis and the results show that this genetic algorithms may perform optimization in constrained problems. Although the constraints were only imposed in the search space, the NSGA-II also allows the use of constraints in the function space [37].

The adaptability of the code is one of the most powerful characteristics of this optimization method. Slight changes in the code (just replacing some variables) will allow the code to optimize different objectives, as it was shown with the airfoil case. This makes the code versatile and open to a broad variety of possibilities when looking for new optimization problems.

Although genetic algorithms have been presented as one of the better solutions, they also have problems and limitations. The most evident one is that GA are heuristic methods that only achieve an approximation of the actual solution. Due to the stochasticity of the algorithm, each run will return a slightly different approximation of that solution. However, finding the "optimal" solution may not be feasible and the solution obtained with genetic algorithms may consist of a robust solution for multi-objective purposes.

There are other evolutionary algorithms that may also be implemented for CFD optimization, such as evolutionary algorithms or particle swarm optimization. All these new machine learning techniques will be of high importance in the future of computer fluid simulation and the optimization of cases, codes, and shapes.

Another major improvement that may be done to the application of genetic algorithms in the computer fluid dynamics field is data management. Each simulation returns a huge amount of data (roughly 50 Gb per case) and everything is reduced to just some bytes that contain the fitness value. Taking advantage of all the data of the simulations may be used for improving the convergence of the algorithm without being necessary the use of more generations or individuals.

Another update in the code can be a higher parallelization level. Evolutionary algorithms have the advantage to be easily and efficiently parallelizable [29]. The code only performs the CFD simulations in a parallel fashion. However, fitness evaluation (extracting data from the simulations) is also a very computer demanding operation. If this process was also run in parallel instead of in serial processors, the whole algorithm will run in shorter amounts of time. The big bottleneck of genetic algorithms in computer fluid dynamics is exactly this: the evaluation time of each individual is high when repeating it generation after generations. Analysis like processor convergence are more than justified to reduce as much as possible the simulation time.

The use of genetic algorithms in 3D simulations is possible, but the increase in computational time make it an unfeasible analysis for this report. The amount of data from a 3D simulation will also be way larger than the data of a 2D simulation, being necessary more space to handle all the files. Apart from computational limitations, the use of GA in 3D cases will also be a topic of high importance in the future.

Finally, the algorithm has been implemented for 2 variables and 2 objective functions. However, the real potential of the genetic algorithm lays on cases with more than two variables, where the search space is complex and can not be visualized in 2 dimensions. This search will also require higher computational resources and more time will be spent in doing simulations. In 2D cases, the results may seem more or less intuitive and the Pareto front may be "located" in both the search and function space. In the case of tenths of variables, this visualization will be a little trickier and the direct relationship between search and function spaces will not be that straightforward.

References

[1] K. Miettinen, *Nonlinear Multiobjective Optimization*, ser. International Series in Operations Research & Management Science. Springer US, 1999, ISBN: 9780792382782.

[2] T. T. Binh and U. Korn, "Mobes: A multiobjective evolution strategy for constrained optimization problems," in *The Third International Conference on Genetic Algorithms (Mendel 97)*, vol. 25, 1997, p. 27.

[3] I. P. Stanimirovic, M. L. Zlatanovic, and M. D. Petkovic, "On the linear weighted sum method for multi-objective optimization,"

[4] I. Y. Kim and O. De Weck, "Adaptive weighted sum method for multiobjective optimization: A new method for pareto front generation," *Structural and multidisciplinary optimization*, vol. 31, no. 2, pp. 105–116, 2006.

[5] W. Jakob and C. Blume, "Pareto optimization or cascaded weighted sum: A comparison of concepts," *Algorithms*, vol. 7, no. 1, pp. 166–185, 2014.

[6] C. M. Fonseca and P. J. Fleming, "An overview of evolutionary algorithms in multiobjective optimization," *Evolutionary computation*, vol. 3, no. 1, pp. 1–16, 1995.

[7] Y. Collette and P. Siarry, *Multiobjective optimization: Principles and case studies*. Springer Science & Business Media, 2013.

[8] N. I. Jaini and S. V. Utyuzhnikov, "A fuzzy trade-off ranking method for multi-criteria decision-making," *Axioms*, vol. 7, no. 1, p. 1, 2017.

[9] E. Triantaphyllou, "Multi-criteria decision making methods," in *Multi-criteria decision making methods: A comparative study*, Springer, 2000, pp. 5–21.

[10] A. V. Lotov, V. A. Bushenkov, and G. K. Kamenev, *Interactive decision maps: Approximation and visualization of Pareto frontier*. Springer Science & Business Media, 2013, vol. 89.

[11] K. Deb, *Multi-objective optimization using evolutionary algorithms*. John Wiley & Sons, 2001, vol. 16.

[12] Y. Li, "Momcmc: An efficient monte carlo method for multi-objective sampling over real parameter space," *Computers & Mathematics with Applications*, vol. 64, no. 11, pp. 3542–3556, 2012.

[13] J. C. Spall, *Introduction to stochastic search and optimization: Estimation, simulation, and control*. John Wiley & Sons, 2005, vol. 65.

[14] G. Altekar, S. Dwarkadas, J. P. Huelsenbeck, and F. Ronquist, "Parallel metropolis coupled markov chain monte carlo for bayesian phylogenetic inference," *Bioinformatics*, vol. 20, no. 3, pp. 407–415, 2004.

[15] W. R. Gilks and P. Wild, "Adaptive rejection sampling for gibbs sampling," *Applied Statistics*, pp. 337–348, 1992.

[16] J. Brownlee, *Clever algorithms: Nature-inspired programming recipes*. Jason Brownlee, 2011.

[17] R. E. Caflisch, "Monte carlo and quasi-monte carlo methods," *Acta numerica*, vol. 7, pp. 1–49, 1998.

[18] Y. Cui, Z. Geng, Q. Zhu, and Y. Han, "Multi-objective optimization methods and application in energy saving," *Energy*, vol. 125, pp. 681–704, 2017.

[19] R. S. Michalski, J. G. Carbonell, and T. M. Mitchell, *Machine learning: An artificial intelligence approach*. Springer Science & Business Media, 2013.

[20] J. R. Koza, M. A. Keane, M. J. Streeter, W. Mydlowec, J. Yu, and G. Lanza, *Genetic programming IV: Routine human-competitive machine intelligence*. Springer Science & Business Media, 2006, vol. 5.

[21] N. L. Cramer, "A representation for the adaptive generation of simple sequential programs," in *Proceedings of the first international conference on genetic algorithms*, 1985, pp. 183–187.

[22] S. Das and P. N. Suganthan, "Differential evolution: A survey of the state-of-the-art," *IEEE transactions on evolutionary computation*, vol. 15, no. 1, pp. 4–31, 2011.

[23] G. J. Klir and B. Yuan, *Fuzzy sets, fuzzy logic, and fuzzy systems: Selected papers by Lotfi A. Zadeh*. World Scientific Publishing Co., Inc., 1996.

[24] S. Droste, T. Jansen, and I. Wegener, "Upper and lower bounds for randomized search heuristics in black-box optimization," *Theory of computing systems*, vol. 39, no. 4, pp. 525–544, 2006.

[25] D. H. Wolpert and W. G. Macready, "No free lunch theorems for optimization," *IEEE transactions on evolutionary computation*, vol. 1, no. 1, pp. 67–82, 1997.

[26] M. Mitchell, *An introduction to genetic algorithms*. MIT press, 1998.

[27] S. Ronald, "Genetic algorithms and permutation-encoded problems: Diversity preservation and a study of multimodality," PhD thesis, 1995.

[28] J. Zhong, X. Hu, J. Zhang, and M. Gu, "Comparison of performance between different selection strategies on simple genetic algorithms," in *Computational Intelligence for Modelling, Control and Automation, 2005 and International Conference on Intelligent Agents, Web Technologies and Internet Commerce, International Conference on*, IEEE, vol. 2, 2005, pp. 1115–1121.

[29] D. Thévenin and G. Janiga, *Optimization and Computational Fluid Dynamics*, ser. Optimization and Computational Fluid Dynamics. Springer Berlin Heidelberg, 2008.

[30] R. R. Yager and L. A. Zadeh, *An introduction to fuzzy logic applications in intelligent systems*. Springer Science & Business Media, 2012, vol. 165.

[31] D. E. Goldberg, *Genetic algorithms*. Pearson Education India, 2006.

[32] M. D. Vose and G. E. Liepins, "Punctuated equilibria in genetic search," *Complex systems*, vol. 5, no. 1, pp. 31–44, 1991.

[33] D. Whitley, "An executable model of a simple genetic algorithm," in *Foundations of genetic algorithms*, vol. 2, Elsevier, 1993, pp. 45–62.

[34] ——, "A genetic algorithm tutorial," *Statistics and Computing*, vol. 4, pp. 65–85, 1994.

[35] J. Holland, *Adaptation in Natural and Artificial Systems: An Introductory Analysis with Applications to Biology, Control, and Artificial Intelligence*, ser. A Bradford book. M.I.T.P., 1992.

[36] A. Konak, D. W. Coit, and A. E. Smith, "Multi-objective optimization using genetic algorithms: A tutorial," *Reliability Engineering & System Safety*, vol. 91, no. 9, pp. 992–1007, 2006.

[37] K. Deb, A. Pratap, S. Agarwal, and T. Meyarivan, "A fast and elitist multiobjective genetic algorithm: Nsga-ii," *IEEE transactions on evolutionary computation*, vol. 6, no. 2, pp. 182–197, 2002.

[38] H. Versteeg and W. Malalasekera, *An Introduction to Computational Fluid Dynamics: The Finite Volume Method*. Pearson Education Limited, 2007.

[39] C. Poloni, "Hybrid GA for multi objective aerodynamic shape optimisation," 1995.

[40] E. Zitzler, K. Deb, and L. Thiele, "Comparison of multiobjective evolutionary algorithms: Empirical results," *Evolutionary computation*, vol. 8, no. 2, pp. 173–195, 2000.

[41] T. K. Sengupta, N. Singh, and V. Suman, "Dynamical system approach to instability of flow past a circular cylinder," *Journal of Fluid Mechanics*, vol. 656, pp. 82–115, 2010.

[42] D. Green and W. G. Unruh, "The failure of the Tacoma bridge: A physical model," *American journal of physics*, vol. 74, no. 8, pp. 706–716, 2006.

[43] S. Rashidi, M. Hayatdavoodi, and J. A. Esfahani, "Vortex shedding suppression and wake control: A review," *Ocean Engineering*, vol. 126, pp. 57–80, 2016.

[44] C. Bissell and D. Chapman, *Digital signal transmission*. Cambridge University Press, 1992.

[45] B. Djebedjian, "Two-dimensional diffuser shape optimization," in *Proceedings of IMEC2004, International Mechanical Engineering Conference*, 2004.

[46] B. Schmandt and H. Herwig, "Diffuser and nozzle design optimization by entropy generation minimization," *Entropy*, vol. 13, no. 7, pp. 1380–1402, 2011.

[47] N. R. Kapania, K. Terracciano, and S. Taylor, "Modeling the fluid flow around airfoils using conformal mapping," *SIAM Undergraduate Research Online*, vol. 1, no. 2, 2008.

[48] K. J. Chichakly and M. J. Eppstein, "Improving uniformity of solution spacing in biobjective evolution," in *Proceedings of the 15th annual conference companion on Genetic and evolutionary computation*, ACM, 2013, pp. 87–88.

[49] Y. Yuan, H. Xu, and B. Wang, "An improved NSGA-III procedure for evolutionary many-objective optimization," in *Proceedings of the 2014 Annual Conference on Genetic and Evolutionary Computation*, ACM, 2014, pp. 661–668.