Soft Computing
https://doi.org/10.1007/s00500-019-03761-6

METHODOLOGIES AND APPLICATION

Smart adaptive run parameterization (SArP): enhancement of user manual selection of running parameters in fluid dynamic simulations using bio-inspired and machine-learning techniques

Hatem Ghorbel[1] · Nicolas Zannini[1] · Salma Cherif[1] · Florian Sauser[2] · David Grunenwald[3] · William Droz[1] · Mahamadou Baradji[4] · Djamel Lakehal[4]

© The Author(s) 2019

Abstract
Computational fluid dynamic (CFD) simulations present numerous challenges in the domain of artificial intelligence. Computational time, resources and cost that can reach disproportional size before leading a simulation to its fully converged solution are one of the central issues in this domain. In this paper, we propose a novel algorithm that finds optimal parameter settings for the numerical solvers of CFD software. Indeed, this research proposes an alternative approach; rather than going deeper in reducing the mathematical complexity, it suggests taking advantage of the history of previous runs in order to estimate the best parameters for numerical equation resolution. In fact, our approach is bio-inspired and based on a genetic algorithm (GA) and evolutionary strategies enhanced with surrogate functions based on machine-learning meta-models. Our research method was tested on 11 different use cases using various configurations of the GA and algorithms of machine learning such as regression trees extra trees regressors and random forest regressors. Our approach has achieved better runtime performance and higher convergence quality (an improvement varying between 8 and 40%) in all of the test cases when compared to a basic approach which requires manually selecting the parameters. Moreover, our approach outperforms in some cases manual selection of parameters by reaching convergent solutions that couldn't otherwise be achieved manually.

Keywords Computational fluid dynamics · Genetic algorithms · Surrogate functions · Machine learning

1 Introduction

Applying parameter optimization techniques to CFD software is very challenging since CFD simulations can require weeks of computation on expensive high-performance clusters. Typical optimization strategies (e.g., Monte Carlo, Gradient Descent or genetic algorithms) require thousands of simulation runs and are often still debatable (Whitley 1994).

Recently there has been an increasing interest in exploiting machine-learning techniques in the aim of accelerating and enhancing evolutionary computation such as the work of Asouti et al. (2016). Our approach is carried out in accordance with this trend and combines evolution strategies with learning algorithms in order to reach good numerical parameters within as few trials and errors as possible. Results will reduce the complexity of the task of parameter choice for users without sufficient background in numerical simulation methods.

The current state of simulation resolution by CFD software presents many issues. In fact, some simulations cannot reach convergence by manual parameter selection. Some others achieve convergence but suffer from a lack of performance, poor solution quality and costly resolution time. Reaching and improving numerical convergence is still quite a hard task, costly to obtain and often not achievable in a traditional trial-and-error method. Therefore, we need an

Communicated by V. Loia.

✉ Hatem Ghorbel
hatem.ghorbel@he-arc.ch

[1] Data Analytics Group, Haute École Arc Ingénierie, University of Applied Sciences and Arts Western Switzerland, Rue de la Serre 7, 2610 Saint-Imier, Switzerland

[2] Embedding Computing System Group, Haute École Arc Ingénierie, University of Applied Sciences and Arts Western Switzerland, Rue de la Serre 7, 2610 Saint-Imier, Switzerland

[3] Interaction Technologies Group, Haute École Arc Ingénierie, University of Applied Sciences and Arts Western Switzerland, Rue de la Serre 7, 2610 Saint-Imier, Switzerland

[4] Ascomp AG, Zurich, Switzerland

Published online: 19 January 2019

algorithm that would explore the space of possible solutions while keeping an intelligent selection aiming at optimizing the problem resolution. This concept should be expanded using meta-heuristics that would minimize the time and resources within the numerical resolution.

A first intuitive answer to our needs is a genetic algorithm that explores the search space of solutions and applies various mechanisms to guarantee the outcome of the fittest ones. Yet, to ensure that the genetic algorithm covers a sufficiently large search space, it is necessary to launch it with a relatively large population size in each generation. And then we need to call for each candidate solution the original fitness function for evaluation. Although it ensures an optimal solution, this strategy represents the main drawback of the GA: it is time-consuming and resource intensive. Indeed, each of the solutions found through mutation and crossover operations should be assessed using the numerical solver itself to assign an objective fitness (i.e., with a population of 10 individuals and 10 generations, numerical solver is executed 100 times).

To remedy this problem, we propose to use machine-learning techniques to provide prediction functions as fitness surrogate in the GA. It works as a data-driven process based on historical data collected in log files and used to estimate new fitness from the perspective of previous cases. Hence it becomes easier for the GA evaluate solutions and send the most adapted ones to the numerical solver.

In a learning phase, parameter settings that are observed to work well for specific problems will be fingerprinted and recorded into a learning model. In a production phase, the model will be used to speed up the genetic algorithm and quickly find optimal parameter settings. Effectively, this consists in aggregating the knowledge and experience of more advanced users into a database that is then used to the benefit of all users. In fact, historical data are first filtered, categorized and used to train a machine-learning-based surrogate model in the GA. In order to find out the best surrogate fitness function, we have conducted a comparison between different supervised machine-learning models such as regression trees, extra trees regressors, and random forest regressors.

By combining a bio-inspired approach with machine-learning techniques, the task of choosing the optimal numerical parameters will effectively be shifted from the user to the software. This will make a significant impact on the usability of the CFD software. Consequently, the CFD software will be accessible to users that do not necessarily have strong expertise in numerical methods. The remainder of this paper is structured as follows. We will first lay out the related work previously conducted to our research in Sect. 2, then we will present our methodology to solve the problem in Sect. 3, namely the GA and surrogate model. Then, in Sect. 4, we will illustrate and discuss the results we obtained and evaluate the approach using metrics of the domain. Finally, in Sect. 5 we will end up with a conclusion of our work.

2 Related work

In CFD, the nonlinear Navier–Stokes equations are commonly solved using a combination of iterative methods. In the literature, there is a quite large number of such methods, each parameterized by several free parameters, and performing differently in different scenarios. The simplest example is perhaps the relaxation method used to update the solution at each iteration using a combination of the current values and the predicted values. Convergence properties of the solver are then highly dependent on the choice of the relaxation factors. Low relaxation stabilizes the method at the expense of an increased computational cost. Conversely, high relaxation factors tend to speed up the numerical method but also decrease stability, which may lead to numerical blowup. Recent studies Dragojlovica and Kaminskib (2004) have shown that optimum relaxation factors can be found using exploratory computation. Accelerating the convergence of CFD solver in general has been seen as an optimization problem where optimum numerical values are explored in a large search space. Different optimization methods using either local optimization techniques (such as gradient methods, quasi-Newton methods, and simplex methods), or global techniques (such as simulated annealing, Monte Carlo, genetic algorithms) have been generally used by scholars to improve resolution performance (Asouti et al. 2016).

2.1 Global optimization techniques: genetic and evolutionary approach

Local optimization techniques such as gradient methods, quasi-Newton methods, and simplex methods depend strongly on the solution domain and tend to be tied to the initial guess. As a result, this tight coupling enables these methods to take advantage of the solution space characteristics, resulting in relatively fast convergence to local maxima or minima. However, differentiability and/or continuity rapidly create constraints on the solution domain. These limitations make local methods generally restricted to smooth and unimodal objective functions and therefore not unsuitable for real-world research fraught with discontinuities, multimodal, and noisy search spaces (Haupt et al. 1998; Goldberg 2006).

Global optimization techniques such as simulated annealing, genetic algorithms, and Monte Carlo methods are largely independent of the solution space and place few constraints. They better cope with solution spaces having discontinuities, constrained variables, nonlinear relations, or a large number of dimensions. They are more robust than local techniques and yield optimum or near-optimum solutions instead of local optimum (Whitley 1994).

Genetic algorithms (GA) are considered more efficient when compared to simulated annealing as they are based on a population method rather than a single-state method. GA provides faster convergence and is more adapted to parallelism. They are particularly suitable for complex engineering problems (Gen and Cheng 2000) such as CFD problem solving. For instance, Marco et al. (1999) presented a GA based on a multi-objective optimization to solve the problem of the design of an airfoil in Eulerian flow. In Kotragouda (2007) firstly developed and optimized a four-jet control system using a continuous GA and the results are compared with the EARND GA results (Huang 2004). Secondly, an unsteady two-jet control system (synthetic jets) is setup and optimized using the Continuous GA. Fabritius (2014) used GA to improve different turbulence models applied to various types of flows, particularly to improve for the $k - \epsilon$ and the SpalartAllmaras models.

2.2 Surrogate modeling

GA is found to be slow when the search space is huge, especially in tangible engineering problems such as CFD solvers. Several attempts to accelerate the solution search using surrogate functions have been proposed. Buche et al. (2005) have used Gaussian process to model fitness function in the optimization of stationary gas turbine compressor profiles. Voutchkov and Keane (2010) discuss the idea of using surrogate models for multi-objective optimization and demonstrate this idea using several response surface methods on a pre-selected set of test functions from the literature such as F5 and ZDT1-ZDT6.

Zvoianu et al. (2013) have proposed an on-the-fly automated creation of highly accurate and stable surrogate fitness functions based on artificial neural networks to solve the problem of performance optimization of electrical drives. They achieved to enhance the computational time of the optimization process by 4672%. Similarly, Brownlee and Wright (2015) discuss surrogate fitness models to solve the problem of building design. Proposed surrogates are based on radial basis function networks, combined with a deterministic scheme to deal with approximation error in the constraints by allowing some infeasible solutions in the population. Forrester et al. (2006) investigate polynomial regression-based surrogate methods for improving CFD simulations for global aerodynamic optimization. They build a strategy of combining expecting improvement updates with partially converged CFD results in order to improve the efficiency of the global approximation. They finally show that this strategy outperforms the traditional surrogate-based optimization.

Moreover, Dragojlovica and Kaminskib (2004) have reported a novel method for accelerating convergence of iterative CFD solvers achieved by a control algorithm which uses fuzzy logic as a decision-making technique in order to guide the under-relaxation of the discretized Navier–Stokes equations. The control criteria are derived from observation over a large interval of previous consecutive iterations.

The evaluation of the method has shown a five-time acceleration of the number of iterations in case of mixed convection and of a two-time acceleration in case of natural convection. Other approaches proposed by Lee and Takagi (1993), Alba et al. (1996) have introduced controlling algorithms from the perspective of the combination of fuzzy logic to genetic algorithms.

Several works have proposed systems designed to use experience from previous designs and/or simple modeling tools. Giannakoglou (2002) have reviewed the use of genetic algorithms to solve problems in aeronautics, particularly numerical optimization methods such as CFD solvers. They have pointed out the typical problems of the population-based search algorithms such as the excessive number of candidate solutions and proposed surrogate models mainly based on neural networks to reduce computing cost. They stored previously evaluated individuals along with their fitness values in databases. For each new individual, the database is scanned and the closest neighbors to the new individual are identified and used to train the local neural network which finally estimates the new individual fitness. They have pointed out that several versions of neural networks outperformed the conventional genetic algorithms (without surrogate model) in terms of CPU time efficiency.

Similarly, the enhancement of genetic algorithms by means of machine-learning techniques such as neural networks (Llor et al. 2007; Kyriacou et al. 2014), support vector machines (Barros et al. 2007), and symbolic regression (Schmidt and Lipson 2008) was applied in developing CFD controller systems. Shirayama (2005) have viewed the acceleration of CFD convergence from a recommendation system perspective, i.e., for each evolution of the algorithm, the search space of individuals is reduced using knowledge rules extracted from historical data. They modeled the problem as a genetic algorithm where a recommendation system proposes the best computational parameters in each evolution according to a knowledge database gathered from previous experiences. Authors have reported that the proposed system was implemented and that the recommendation system is highly efficient and parallelized.

Similar to previous work in surrogate modeling applied to accelerate convergence in evolutionary computing such as Forrester et al. (2006), Schmidt and Lipson (2008), Barros et al. (2007), Zvoianu et al. (2013), we explore in the present work a machine-learning approach where we learn from previous CFD resolution to calculate an on-the-fly estimation of the fitness function. As machine learning needs a careful choice of training data, the strategy of CFD partial convergence as in Forrester et al. (2006) is applied. Best partial convergence solutions compete to become the best-fitted solutions.

3 Methodology

Applying parameter optimization techniques to CFD software is very challenging since CFD simulations are computationally expensive. Typical optimization strategies (e.g., Monte Carlo, Gradient Descent or genetic algorithms) require thousands of simulation runs and are often not feasible. Our approach will instead combine evolution strategies with learning algorithms as surrogate model, so as to reach good numerical parameters in as few trials and errors as possible. In a learning phase, parameter settings that are observed to work well for specific problems will be fingerprinted and recorded into a learning model. In a production phase, the model will be used to speed up the evolutionary algorithm and quickly find optimal parameters. Effectively, this consists in aggregating the knowledge and experience of more advanced users into a database that is then used to the benefit of all users.

In this section, our genetic algorithm approach is firstly presented. Secondly, the combined machine-learning surrogate model is described.

3.1 Genetic algorithm

Genetic algorithms (GA) are a family of computational models inspired by Darwin's evolution theory and Mendel's genetic laws. These models encode a potential solution to a specific problem on a simple chromosome-like structure and apply bio-inspired recombination operators to these structures in such a way as to create more adapted solution to the problem. The biological inspiration consists in preserving significant features and searching for those that would be better adapted to the problem resolution, over successive generations. Genetic algorithms are often viewed as function optimizers, although the range of problems to which genetic algorithms have been applied is quite broad (Goldberg 2006).

An implementation of a genetic algorithm begins with a population of chromosomes. Each one is evaluated and allocated reproductive opportunities by mutation and crossover in such a way that those which hold a better solution to the problem are given more chances to be selected to "reproduce" than those which hold poorer solutions. The rightness of a solution is typically defined with respect to the current population and generation. In this research, it is measured by an original fitness function chosen according to the quality of a partial convergence. The selected solutions represent a new population on which we iteratively conduct the same process. Hence, a new generation of solutions is created after each iteration until the stopping criteria are reached. The stopping criterion could be related to the number of generations, time constraint or the quality of the achieved solutions (stationary fitness score).

In this paper, the proposed GA is developed as a controller wrapped around the CFD software with the aim of automatically selecting the free parameters before running numerical simulations. It is designed to select the fittest solution in a finite population of solutions evolved over different generations according to evolution rules and genetic operations as will be detailed below. The only information the GA requires is one payoff value per objective for each candidate solution according to an elaborated fitness function.

3.1.1 GA launching conditions

Basically, the GA are robust methods for optimization in high-dimensional search spaces. However, they typically require a large number of calls of the fitness function. In the case of CFD solvers, evaluating the fitness function is very costly and this may drastically decrease the overall computational performance. To avoid this problem, the GA should be activated only when it is needed. This is when it could provide a capital gain on the computation time, typically when the simulation diverges or does not converge fast enough. This is explained through Fig. 1.[1]

Moreover, the activation conditions vary depending on the simulated cases: steady or unsteady cases. More specifically, the simulation is launched with the user initial parameters and once one of the following conditions occurs, the GA is activated:

(i) For steady cases:
   - Simulation converges slowly or diverges (according to the slope of residuals);
   - Pressure solver reaches the maximum number of iterations too often (a threshold of consecutive times is reached).

(ii) For unsteady cases:
   - Too small time step is reached (a threshold value is reached);
   - Time step decreases rapidly (according to the slope of residuals);
   - Pressure solver reaches its maximum number of iterations too often (a threshold of percentage/ratio of iterations in a time step are reached);
   - Time step does not converge for a threshold value of consecutive times.

3.1.2 GA structure

The initial population for this GA is generated from the chromosome formed from the initial parameters (Fig. 2)

[1] TransAT is the CFD software used in this article.

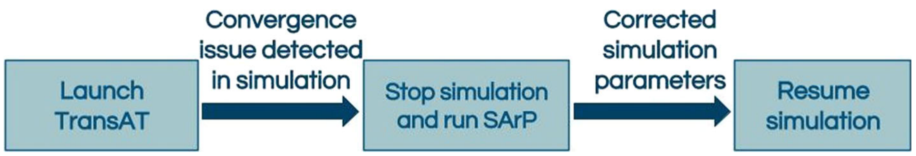

Launch TransAT
Convergence issue detected in simulation
Stop simulation and run SArP
Corrected simulation parameters
Resume simulation

Fig. 1 GA launching after detecting convergence problems

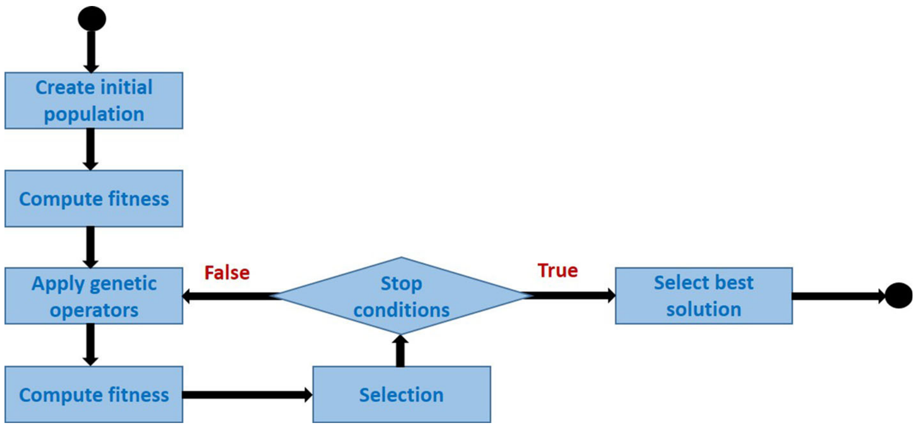


Fig. 2 The different stages of the GA

setup by the user. Then $N-1$ mutations are applied to obtain an N-individuals' population containing $N-1$ mutated chromosomes in addition to the initial chromosome as illustrated in Fig. 4. A value of N varying between 5 and 11 has been used in various GA experiments. Since computational time is an issue, $N = 5$ has given the best performance in most of the cases. Mutation is detailed in Sect. 3.1.3.

Chromosomes are composed of genes. These genes hold the free parameters used to optimize numerical solvers (Fig. 3).[2] With the help of CFD experts and based on the history of previous simulations, fixed ranges for parameters were defined. These parameters can have one of the three types:

(i) numerical type, defined by a range of values delimited by [min, max] (e.g., CFLMIN, the CourantFriedrichsLewy condition [0.05, 1.0]);
(ii) enumerate type, represented by a list of possible values. (e.g., eqn_solver(2): solver1", solver2", solver3", "solver4, solver);
(iii) boolean type, have one of two possible values (e.g., autorelaxation: True, False).

Details of these parameters are stored alongside in a different description file where we describe their types, ranges, possible values. We constructed these descriptions by a preprocessing phase that analyses the experts' tuning process of solutions that are either originally non-convergent or badly convergent ones. This analysis gave us a concrete idea about the different parameters and their relevance to the solvers as well as their varying behavior and ranges, and their effects on the simulation.

Finally, the new chromosomes we create applying genetic operations are the result of multiple combinations derived from the intersection of chromosome description file with current simulation case.

3.1.3 Genetic operations

Genetic operations create new chromosomes based on an existing population. In this paper, it consists in applying crossover followed by a mutation on chromosomes in each generation, apart from the first generation where we directly apply a mutation on the initial chromosome (Fig. 4). Regarding **mutation**, it is applied on a random number of newly created chromosomes in each generation. Mutation consists in muting random parts of genes according to a predefined mutation probability. Mutation when applied enables to diversify solutions from the search space and thus prevents the algorithm to be trapped in local minima. In the implemented GA, 30% of new chromosomes undergo a mutation of 20% of their genes. Mutation operations are varied according to parameters' types as following.

– **Numerical parameters** Pull a random number according to a Gaussian distribution. The Gaussian is centered on the current value and is bounded by the min and max values allowed for the parameter;
– **Listed** Select randomly another item in the list of possible choices;
– **Boolean** Invert Boolean value.

[2] The values stated in the example below are not correct and are merely there to serve the purpose of comprehension.

Fig. 3 An example of a chromosome

```
{ "eqn_solver(1)": "solver4", "eqn_relax_factor(1)": 0.5, "eqn_preconditioner(3)":
"valueprecond1",  "eqn_relresiter(3)": 0.005, "eqn_relax_factor(3)": 1,
"eqn_solver_library(1)": "value2",  "eqn_nswp(4)": 50,  "eqn_nswp(2)": 50,
"eqn_solver_library(3)": "value3",  "eqn_preconditioner(1)": "none",
"eqn_subpreconditioner(4)": "valuesuprecond4",  "autorelaxation": true,
"scarborough": 2, "eqn_solver(3)": "solver5", "CFLMIN": 1,  "eqn_solver_library(4
)": "value2","eqn_subpreconditioner(2)": "valuesuprecond4", "eqn_relresiter(1)":
0.00001,  "STNMAX": 3,  "eqn_relax_factor(4)": 0.5, "eqn_relresiter(4)": 0.001,
"eqn_relax_factor(2)": 0.5, "eqn_preconditioner(2)": "value1",
"eqn_subpreconditioner(3)": "valuesuprecond4",  "eqn_relresiter(2)": 0.001,
"eqn_nswp(3)": 100, "eqn_nswp(1)": 100, "eqn_preconditioner(4)": "value1",
"eqn_solver_library(2)": "value3",  "eqn_solver(4)": "solver5",
"eqn_solver(2)": "solver5", "eqn_subpreconditioner(1)": "none", "CFLMAX": 3 }
```

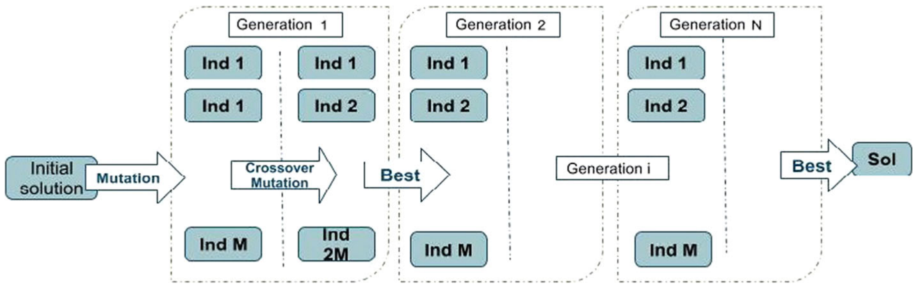


Fig. 4 Adopted GA strategy

Regarding **crossover**, we chose to work with single-point crossover randomly defined.

According to the encoding method and the predefined physical rules, we always have to be certain to maintain the overall consistency of the underlying solution and the validity of the individuals in our population. These physical rules are general constraints defined by CFD experts and applied to ensure the resolution coherence within a solver. This is to avoid having a solution with a good score, but having no physical sense when interpreted. Thus, following each genetic operation, a consistency check process is applied to each potential solution. In case duplicated chromosomes are created, further mutations are applied to create separate ones. The main reason behind is to stimulate the diversity of the population considered as relatively small-sized one and so very to exposed to uniformity.

3.1.4 Selection

The basic part of the selection process is to stochastically select which individuals from one generation that would survive and populate the next generation. The assumption is that the fittest individuals have a greater chance of survival than least fitted ones. The latter are not without a chance. They may have genetic coding that shows usefulness to future generations. Different selection strategies are applied while dealing with this problem:

- **Truncation** From one generation to the other, we only keep the P (population size) best chromosomes;
- **Roulette** This technique is analogous to a roulette wheel with each slice proportional in size to the fitness score assigned in such a way that higher score is always more favorable to be selected for next generation;
- **Tournament** Provides selective pressure by holding a tournament competition among S individuals $S \leq P$. The best individual from the tournament is the one with the highest fitness, which is the winner of the P;

Finally, in all different cases, we found out that keeping a portion of the generation's solutions selected using truncation strategy always enhances the results and guarantees that the best solution is constantly selected for the next generation. Thus, we apply truncation alongside with a second selection strategy in each generation. This hybrid solution accentuates the randomness of the algorithm and guarantees a sufficient exploration of the search space.

3.1.5 Halt criteria

We dealt with two families of cases, steady and unsteady ones. In steady flows, conditions (velocity, pressure and cross section) may vary from point to point but do not vary with time. Simulations are executed during n iterations. However, the properties of an unsteady flow vary with time at any point of the fluid. Simulations are executed during N time steps. Genetic evolution stops when one of the halt criterium is reached. These criteria vary from steady to unsteady cases.

(i) For steady cases

- The best solution found so far hasn't been improved for X consecutive generations (default $X = 4$);

- The maximum number of generations is reached without the GA convergence criteria being reached. In this case, if one or more solutions have been found during the process, the fittest one is selected. If no fit solution is found, the simulation stops.

(ii) For unsteady cases

- The last time step of a solution has converged;
- The best solution found so far hasn't been improved for X consecutive generations (default $X = 4$);
- Similar to the steady case, the maximum number of generations is reached without the GA convergence criteria being reached.

3.1.6 Fitness function

In each generation, individuals are evaluated with the CFD software in order to score their fitness. This is conducted by calling the CFD software which starts solving the CFD equations for the next k iterations (steady cases) or time steps (unsteady cases). The obtained residuals are analyzed and then scored according to which extent the objectives of obtaining an efficient, high quality and stable convergence are achieved.

Even if we are in a multi-objective problem (maximize the convergence quality, the stability quality, and minimize the distance to the target), we have solved the problem using a single-objective approximation given by a scoring function defined as a weighted sum of these three objectives. Consequently, the fitness function is defined as a linear combination of the following three objectives: (i) distance of the residual to the target, (ii) quality and (iii) stability of convergence. The weight of each indicator in the scoring function is adjusted experimentally by observing historical data related to simulations performed by beginners and expert users alike.

In order to quantify the stability and the quality of convergence at a point i, we apply the method of least squares approximation on the window $[i-w, i]$. The scoring function will, consequently, take the following form:

$$\mathbf{f}_i(\mathbf{d}_i, \mathbf{c}_i, \mathbf{e}_i) = -\alpha|\lg(\mathbf{d}_i)| - \beta\mathbf{c}_i - \gamma\lg(\mathbf{e}_i), \quad i \in [w, n] \tag{1}$$

where

w = the size of the least square analysis windows
d_i = distance of the current residual to the target
c_i = convergence quality (slope of the least square estimation)
e_i = stability quality(mean error of least square estimation)
α = distance to the target coefficient, $\alpha \in [0, 1]$
β = convergence quality coefficient, $\beta \in [0, 1]$
γ = stability quality coefficient, $\gamma \in [0, 1]$
$\alpha + \beta + \gamma = 1$
n = number of iterations

The fitness function defined in Eq. 1 indicates the current simulation performance at the progress point i and serves as a metaheuristic to give insights to the genetic algorithm. For steady simulations, it is used as it is in the next steps of the selection. However, for unsteady simulations, two extra qualitative contraints are appended to select the fittest chromosomes. One is related to the convergence of the last time step and the other to the duration of time steps. In these cases, the selection of the fittest chromosomes is done sequentially as following:

1. Select the chromosomes with converged time steps;
2. Select the chromosomes with the highest time steps out of the set created in 1;
3. Select the chromosomes with the highest fitness at the last time step out of the set created in 2.

3.2 Combining genetic algorithm and machine-learning-based surrogate modeling

In order to accelerate the GA runtime, we propose to surrogate the fitness function by an estimated function using machine-learning techniques (Witten et al. 2016). This represents the second challenge of this research. In fact, currently, the fitness of a solution is evaluated according to its capability to efficiently reach the CFD solver convergence. This evaluation is quantified by calling the CFD software and starts running the analytic resolution using parameters set up of the solution, but only for the first k iterations or time steps (the progress point i in Eq. 1). In more details, the obtained residuals are first analyzed then scored according to which extent the objectives of obtaining an efficient, high quality and stable convergence are achieved.

Nevertheless, this is clearly a time expensive strategy especially if we extend the search space to larger number of generations and individuals. Therefore, we propose to develop an estimation fitness function capable of predicting the fitness value of some candidate solutions from the perspective of historical data; previous runs; and previous user expertise of the tool. Thus, instead of keeping the evolution process search for a best solution "naturally" (by evaluating them every time using the CFD software), we redefine the rules of evolution by integrating an associative memory that evaluates the current solution according to learnt experiences. Such a bias in the evolution process gives rise to a self-adapting strategy and decreases the number of calls to the original fitness function.

This approach has been the focus of several academic studies (Llor et al. 2007; Schmidt and Lipson 2008; Shirayama 2005; Barros et al. 2007) and has been shown to improve the computational performance. The estimation modeling we conduct is founded on machine-learning techniques regression trees, and random forest regressors.

The idea, as detailed in Fig. 5, is to launch the GA in a regular conduct in order to feed the machine-learning algorithm with its output data labeled by the scores of the original fitness function (CFD Software). Based on the so far generated history, when the size of the labeled data is sufficient, we train the machine-learning model. As for the following generations, we use the trained model to predict the fitness of the candidate solution without having to launch each time the CFD software. Subsequently, based on this predictive model selection, only 50% of the best-predicted solutions are evaluated with original fitness function using the CFD software. Thus, we reduce by half the number of calls of the CFD software.

3.2.1 Training corpus

The training corpus is composed of samples of previous solutions labeled with their score as quantified by the fitness function. To construct this dataset, we collected a sample of about 10,000 solutions; 2000 per case in all the 5 cases on which we worked along with the GA. Steady and unsteady cases are separated; hence two different training corpora are constructed.

Although a considerable amount of datasets was accumulated, the data need to be filtered and preprocessed before being used to feed the machine-learning algorithm. Depending on the type of simulation (steady or unsteady), the filtering takes into consideration the actual progress of the simulation, the fitness previously allocated to each simulation, and the physical and geometric properties of the cases. Data filtering allows for better targeting the training set by selecting only similar cases to improve prediction performance.

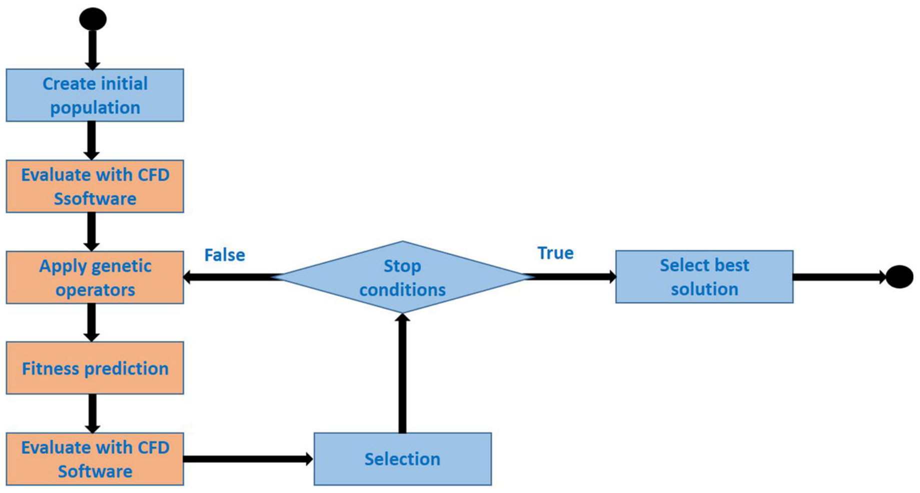
The generated database includes two sets of data depending on the way they are generated. First, *history data* include chromosomes generated and scored during previous runs of same/similar cases. This dataset is collected during previous user simulations and stored in dedicate databases within the CFD software. Similar cases are found using nearest neighborhood (NN) algorithm (Altman 1992) where neighbor is the closest case in the multidimensional feature space formed by physical properties such as boundaries, grid properties, Block-based Mesh Refinement (BMR) levels, and surface tree. Details are out of the scope of this article but the approach is very similar to that of Morbitzer et al. (2003). Second, *local data* include generated scored chromosomes generated on-the-fly during the current run with previous generations of the GA.

3.2.2 Training and prediction strategies

The strategy we adopted once the fitness prediction module is activated is as follows (see Fig. 6):

– Verify whether *history data* exist for this particular simulation or if only the chromosomes created during the current GA run should be relied on;
– Once the appropriate sample of data is located, proceed to selecting chromosomes that are similar to the current chromosome in terms of state and progress level. The selection is solely applied on data from chromosomes that are in the same phase, progress or state as the current chromosome for which the fitness is to be predicted;

Fig. 5 Fitness prediction module added to the GA



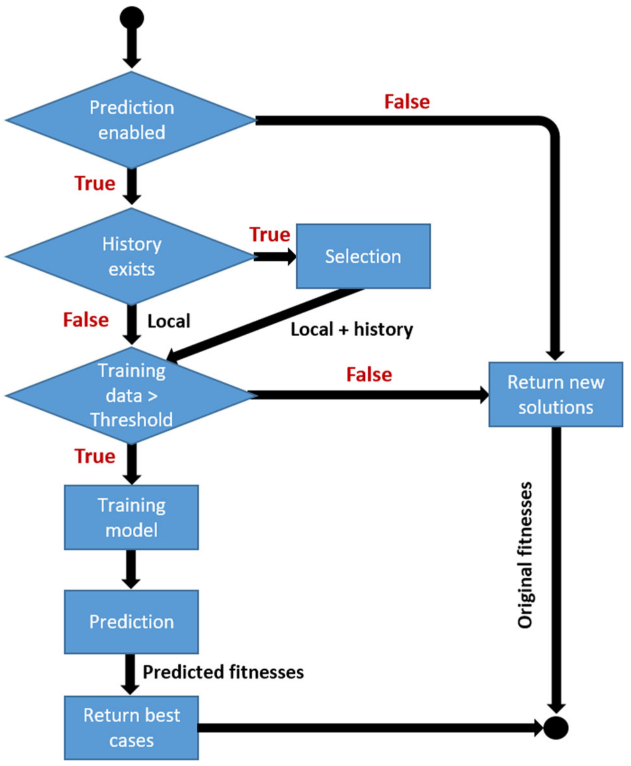


Fig. 6 Detailed flow of the fitness prediction module

- Check whether the training sample size complies with the training size fixed threshold. Training dataset size thresholds were defined empirically from the study of the different models performances in varying the size of the training sample and analyzing the results (see details in Sect. 3.2.4). As a result, minimum thresholds were fixed for each model in order to get satisfactory prediction results;
- If the training sample selected after a cross-validation process is not large enough to get satisfactory model performances, the fitness prediction model is not applied and the GA proceeds to evaluate all the chromosomes;
- If training data are sufficient, actively train the machine learning algorithm on the selected data to generate the prediction model;
- The fitness of the chromosomes of the current generation is predicted based on the applied model;
- Return the chromosomes with the best-predicted fitness to be passed on to the next generation.

3.2.3 Machine-learning techniques

Two types of machine-learning models were implemented: classifiers and regressors (Friedman et al. 2001).

The former type of model has a binary behavior. Chromosomes can be either fit or unfit depending on their fitness. These chromosome classes are uniformly distributed within the training data. This classification method is an efficient way to eliminate cases that are quickly diverging. Although this model is not able to order two solutions in the same class, it remains faster than regressors.

With regressors, regression is used to predict the actual value of the fitness value. It gives more accurate results than with classifiers. The estimation of the fitness function as a continuous function allows the differentiation and ordering of chromosomes. Nonlinear regression models are used to handle the complexity of the problems at hand.

The list of trained algorithms is as follows. Python Scikit-learn framework is used for training and predicting (Pedregosa et al. 2011; Raschka and Vahid 2017).

- **Regression trees** A non-parametric supervised learning method used for classification. The goal is to create a model that predicts the value of a target variable by learning simple decision rules inferred from the data features (Ross 1986);
- **Bagging regressor** An ensemble meta-estimator that fits base regressors each on random subsets of the original dataset and then aggregates their individual predictions (either by voting or by averaging) to form a final prediction. The base estimator used is a regression tree one. This regressor improves the stability and accuracy of machine-learning algorithms and reduces the variance and helps to avoid overfitting (Breiman 1996a, b);
- **extra trees regressor** A meta-estimator that fits a number of randomized regression trees (a.k.a. extra trees) on various subsamples of the dataset and use averaging to improve the predictive accuracy and control overfitting (Matloff 2017);
- **Random forest regressor** A meta-estimator that fits a number of classifying regression trees on various subsamples of the dataset and use averaging to improve the predictive accuracy and control overfitting. The subsample size is always the same as the original input sample size (Matloff 2017).

3.2.4 Cross-validation

In order to validate the stability of our machine-learning models, a tenfold cross-validation method was applied. This did not only give us an idea about how well our model does on data used to train it, but also the error estimation is averaged over all 10 trials to get total effectiveness of our model. This significantly reduces bias as we are using most of the data for fitting and also significantly reduces variance as most of the data are also being used in validation set.

The cross-validation method was used to test the different machine-learning algorithms for fitness estimation. This method is a model validation technique for assessing how results of a statistical analysis generalize to an independent data set. It is mainly used to avoid overfitting. It works as a rotation estimator as it randomly splits data into two different datasets, one for the training and one for the testing.

A tenfold cross-validation method was applied to the generated *local data* by running locally stored test cases with the GA.

To guarantee the prediction models good performance, a certain size of data is required to train. The dataset size threshold ensuring significant results was found empirically by analyzing the performances of the different prediction models with varying size of datasets ranging from 72 to 429 as detailed in Table 1. If this threshold is not reached, the prediction model step is skipped altogether.

4 Comparative results

In this section, both results of the genetic algorithm and the combined machine-learning surrogate model are presented and compared.

4.1 Genetic algorithm results

The choice of the cases is motivated by the convergence speed and the complexity of the physical properties. Simple, moderate and complex cases are selected by numerical simulation experts to form test samples for steady and unsteady cases. GA parameters are adjusted for each class, namely the size of the population is set bigger for unsteady cases as these are slower in convergence. And mutation rates are set higher for small populations to stimulate diversity.

4.1.1 Steady cases

The GA was first tested with the following set of steady cases:

- **Case3** Natural convection
- **Channel_no_cond** Flow of a liquid around a cube with 2 levels of turbulence
- **Channel_flow_flux_pitched** Force convection in an inclined channel
- **Backwardstep** Laminar flow past a backward-facing step modeled with an embedded object
- **Impinging_jet** Jet impinging on a flat plate
- **Ra_10p8** Natural convection in a square cavity at a Rayleigh number of 10^8.

(i) **Baseline**
Our baseline is the user manual selection of parameter when manipulating the CFD software. We distinguish two sorts of users: beginners and experts. The former are generally students and trainers who have sufficient skills to manipulate the CFD software but limited experience in manual selection of physical models. The latter have, however, high skills in all the simulation processes. The baseline simulations for Backwardstep, Ra_10p8 and Impinging_jet did not converge, the aim was to fix the convergence issue. The remaining baseline cases—i.e., Case3, Channel_no_cond and Channel_flow_flux_pitched—were already converging. The aim was to accelerate the convergence and enhance its quality. The Case3 test case was particularly slow.
The results obtained in each of these cases are reported in

Table 1 Steady-case threshold variation, cross-validation results representing the percentage of correctly predicted chromosome fitness

Dataset size	72	143	214	286	357	429
Bagging regressor	81.82%	95.35%	87.5%	88.37%	94.39%	96.90%
Random forest regressor	72.73%	95.35%	87.5%	86.05%	90.65%	93.80%
Extra trees regressor	81.82%	95.39%	90.63%	90.70%	94.39%	93.80%
Regression trees	81.82%	90.70%	93.85%	100%	100%	96.90%

Table 2 Baseline experiments showing user performance with manual parameter selection in **steady cases**

Simulation	Beginner		Expert	
	Time	Iterations	Time	Iterations
Case3	104 s	582	49 s	322
Channel_no cond	–	–	46 s	123
Channel_flow_flux_pitched	–	–	79 s	187
Backward_step	No convergence	No convergence	41 s	375
Impinging_jet	No convergence	No convergence	322 s	1705
Ra_10p8	–	–	No convergence	No convergence

Beginners, generally students, and experts' performances are separated. It is important to note that there are cases which ended up without achieving convergence. Empty values stand for cases that were not proposed to beginners due to their complexity

Table 3 Runtime and iteration number of converged solutions after the application of the GA in steady cases

Simulation	Best			Average		
	GA time (s)	Iterations	Total time (s)	GA time (s)	Iterations	Total time (s)
Case3	50	538	129	95	528	173
Channel_mo cond	44	122	82	51	122	91
channel_flow_flux_pitched	203	186	276	255	186	324
Backward_step	47	269	88	110	456	166
Impinging_jet	1245	405	1324	2087	1031	2242
Ra_10p8	352	857	554	1567	857	1752

The average values were calculated on 10 different runs

the following two Tables 2 and 3 containing details that correspond before and after the application of the GA, respectively.

(ii) **Evaluation metrics**

To compare User's solutions and those found by the GA, we used the three following metrics:

– **Load GA** Ratio of the time spent in the GA compared to the total execution time. The aim is to measure the time load of the added GA with respect to the entire runtime as it is expected that the GA should slow down the CFD resolution. This metric was quite significant to chose the appropriate GA parameters (population size, mutation rates, etc.) to achieve a GA load that doesn't exceed 60%.

$$\text{Load GA} = \frac{\text{GA.Time}}{\text{Total.Time}} \tag{2}$$

– **Quality** Indicator related to the number of iterations (n) in a simulation. Best quality is obtained with as few as possible iterations.

$$\text{Quality} = \frac{1}{n} \tag{3}$$

– **Performance** Indicator related to the computation time required for the numerical simulation when we do not consider the GA time. The aim is to measure the performance of the numerical simulation conducted using the parameters provided by the GA.

$$\text{Performance} = \frac{1}{\text{Total.Time} - \text{GA.Time}} \tag{4}$$

Evaluation using load, quality, and performance is depicted in Table 4.

4.1.2 Discussion

Concerning "Backward_step" and "Impinging_jet" cases, the GA achieved, starting from a beginner's solution, to outperform experts' solutions regarding the performance and quality metrics. However, for "Case3", even if it achieved to outperform beginners' solutions and the quality of the experts' solution, the performance of the latter remains the best. Furthermore, curiously, the GA succeeded in finding a solution to the case "Ra_10p8" which had no prior known user solution. As for "channel_no_cond" and "chan-

Table 4 Comparison between GA and expert results in steady cases in terms of load, quality, and performance

Simulation	Load GA	Quality			Performance		
	Best GA (%)	Best GA	Expert	Gain (%)	Best GA	Expert	Gain (%)
Backward_step	53.41	3.72E−03	2.67E−03	39.41	2.44E−02	2.44E−02	0.00
Case3	38.76	1.86E−03	3.11E−03	40.15	1.27E−02	2.04E−02	− 37.97
Ra_10p8	63.54	1.17E−03	0.00E+00	0.00	4.95E−03	0.00E+00	100.00
Channel_mo cond	53.66	8.20E−03	8.13E−03	0.82	2.63E−02	2.17E−02	21.05
channel_flow_flux_pitched	73.55	5.38E−03	5.35E−03	0.54	1.37E−02	1.27E−02	8.22
Impinging_jet	94.03	2.47E−03	5.87E−04	320.99	1.27E−02	3.11E−03	307.59
Average	62.82			66.99			66.48
Variance	19.19			125.92			126.53

Even if the GA load is relatively high (an average of 62.8% and a variance of), an average gain in the quality (53.65%) and the performance (66.4%) of the convergence is achieved. The results are pronounced in some cases more than others, which explains the high value of the variance

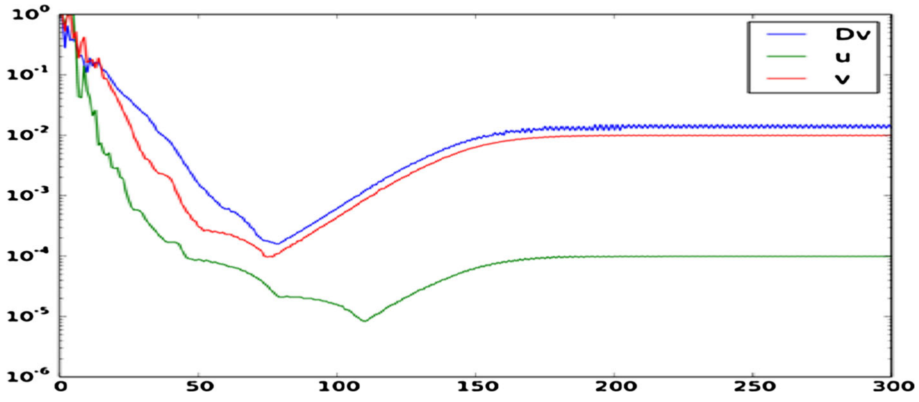


Fig. 7 Graph representing the residual across the number of iterations in backward_step

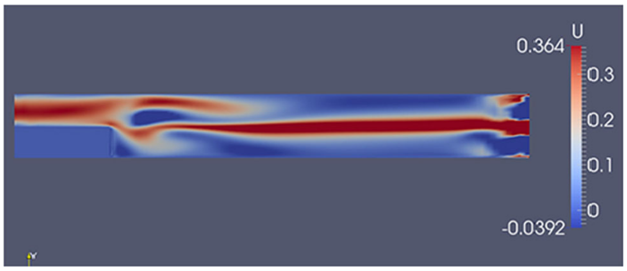


Fig. 8 Physical simulation of the found solution representing the velocity

nel_flow_flux_pitched" cases, their initial solutions were already good, but the GA still managed to improve their quality and performance. In all cases, the GA has significantly improved the initial input user solution.

To sum up, in spite of the high load of the GA, i.e., the additional time it takes in the convergence runtime, the gain in the quality of convergence (measured in terms of gain in number of iterations) and in the performance (measured by the gain in iterative convergence runtime) is achieved. Therefore, further investigation should focus on GA acceleration using by fitness surrogate using prediction models.

From a physical viewpoint, Fig. 7 depicts the physical interpretation of the results. The convergence graph of the Backward_step simulation before applying the GA is plotted. It is noticed that from iteration 70, the residual starts to diverge. The obtained solution after 300 iterations is physically meaningless as seen in Fig. 8. However, after the call of the GA at iteration 70 triggered by the residual divergence, the algorithm achieves to find a better solution (span between iteration 70 and 85 in Fig. 9) that finally converges in 200 iterations and gives a physically meaningful simulation (Fig. 10).

4.1.3 Unsteady cases

The GA was also tested with the following set of unsteady test cases:

– **Ra_10p10** Natural convection in a rectangular cavity at a Rayleigh number of 10^{10}
– **Network_smart** Two-phase flow through a network of pipes
– **Ra_10p5 (Windows only)** Natural convection modeling in a square cavity at Rayleigh number of 10^5 using a compressible flow
– **Elbow_Water_Annulus** Two-phase upward flow in an elbow with water air coming in at the center of the pipe and water in an annulus around air inflow
– **CapillaryTube:** capillary tube flow

(i) **Baseline**
The original simulations for CapillaryTube, Elbow_Water _Annulus and Ra_10p5 (Windows only) did not converge when parameters are set by either beginners or experts. For the Ra_10p5 test case, no known set of parameters making it converge was found until then on Windows whereas it converged on Linux with the hyper solver (not available on Windows) (see Table 5). The results obtained for each of these cases are reported in the following two Tables 5 and 6 containing results before and after the use the GA, respectively.

(ii) **Evaluation metrics**
To compare user solutions and those found by the GA, we used the three following metrics:

– **Load GA**
Ratio of the time spent in the GA compared to the total execution time. The aim is to measure the time load of the added GA with respect to the entire runtime as it is expected that the GA should slow down the CFD resolution. This metric was quite significant to chose the appropriate GA parameters (population size, mutation rates, etc.) to achieve a GA load that doesn't exceed 40%. This ratio is the same as used in steady cases, nevertheless, in unsteady where CFD resolution takes relatively more time, the time load of the GA should not be too high so as to prevent slowing down the total runtime.

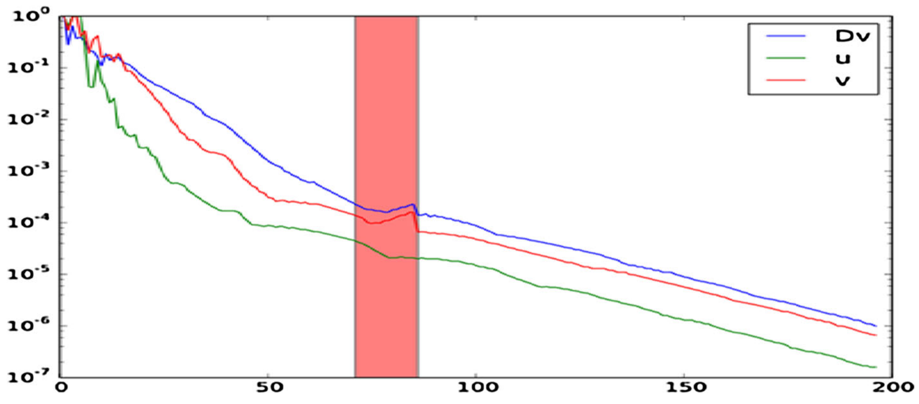


Fig. 9 Graph representing the residual across the number of iterations in backward_step after the call of the GA at iteration 70

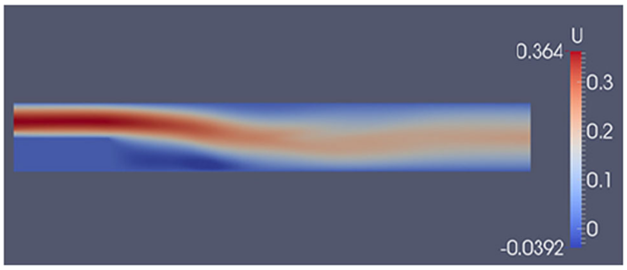


Fig. 10 Physical simulation of the found solution representing the velocity

$$\text{Load GA} = \frac{\text{GA.Time}}{\text{Total.Time}} \tag{5}$$

- **Efficiency**
  Instead of taking about quality in the steady cases, we talk about efficiency which is a time-related indicator since in these cases we talk about time steps instead of iterations. Efficiency is hence calculated as the ratio of the *physical* duration of the simulation and the number of time steps N need to converge the CFD resolution. When N decreases, this indicates that CFD resolution takes less time.

$$\text{Efficiency} = \frac{\text{Phys.Time}}{N} \tag{6}$$

- **Performance** Indicator related to the computation time required for the numerical simulation when we do not consider the GA time. The aim is to measure the performance of the numerical simulation conducted using the parameters provided by the GA.

$$\text{Performance} = \frac{1}{\text{Total.Time} - \text{GA.Time}} \tag{7}$$

4.1.4 Discussion

According to Table 7, the interesting point in unsteady results is that all the simulations with the GA converged to a solution even when experts failed. As a matter of fact, unsteady simulations take in general much more time to run than steady ones; for instance, the runtime of steady cases is between 82 and 1324 s whereas the unsteady cases take between 4100 and 384,888 s (4.5 days). Consequently, we notice that the load of GA decreases as the total runtime of the simulations becomes high (for instance 88% of in Ra_10p5 and capillaryTube cases whose runtime is, respectively, in the range of 2000–4000 s.

The GA has also achieved to converge the three cases ("Ra_10p5", "network_smart" and Elbow Water Annulus") for which the experts have found no solution. The simulated physical time of the expert solution of the case "Ra_10p10"

Table 5 Baseline experiments showing user performance with manual parameter selection in **unsteady cases**

Simulation	Beginner			Expert		
	Time step	Phys. time	Total time	Time step	Phys. time	Total time
Ra_10p10	–	–	–	10,000	49.73 s	50,948 s
Network_smart	–	–	–	No converg.	No converg.	No converg.
Ra_10p5 (Windows)	–	–	–	No converg.	No converg.	No converg.
Elbow water annulus	–	–	–	No converg.	No converg.	No converg.
CapillaryTube	No converg.	No converg.	No converg.	1000	0.2356 s	1083 s

Beginners, generally students, and experts' performances are separated. It is important to note that there are cases ended up without achieving convergence. Empty values stand for cases that were not proposed to beginners due to their complexity

Table 6 Runtime and number of iterations of converged solutions after the application of the GA in unsteady cases

Simulation	Best				Average			
	Step	Phys. time (s)	GA time (s)	Total time (s)	Step	Phys. time (s)	GA time (s)	Total time (s)
Ra_10p10	10,000	170.8	2089	43,807	10,000	167.87	2608	46,573
Network_smart	5614	1	5714	22,112	5003	1	8730	25,757
Ra_10p5 (Windows)	500	2.384	1916	2168	500	2.384	2263	2495
Elbow water annulus	15,272	0.3292	16,137	384,888	15,272	0.3292	16,137	384,888
CapillaryTube	439	0.5	1961	4096	566	0.5	2408	5953

The average values were calculated on 10 different runs

Table 7 Comparison between GA and expert results for unsteady cases in terms of load, quality, and performance

Simulation	Load GA	Performance			Efficiency		
	Best GA (%)	Best GA	Expert	Gain (%)	Best GA	Expert	Gain (%)
Ra_10p10	4.77	3.89E−03	9.76E−04	299.44	1.71E−02	4.97E−03	243.45
Ra_10p5 (Windows)	88.38	1.09E−03	None	100	4.77E−03	None	100
network_smart	25.87	4.52E−05	None	100	1.78E−04	None	100
capillaryTube	47.88	1.22E−04	2.18E−04	− 43.89	1.14E−03	2.36E−04	383.43
Elbow water annulus	4.19	8.55E−07	None	100	2.16E−05	None	100
Average	34.21			125.42			185.38
Variance	35.21			122.34			126.95

Even if the GA load is relatively high (an average of 62.82% and a variance of 19.19%), an average gain in the quality (66.99%) and the performance (66.48%) of the convergence is achieved. The results are pronounced in some cases more than others, which explains the high value of the variance

has tripled with the GA (maintaining the same number of time steps). The GA performance is, however, mitigated. For instance, in the capillaryTube case, starting from a beginner's solution that does not converge, the GA found a solution but could not outperform the experts. The total runtime GA solution is still higher (5953 s) than that of the expert (1083 s). 47.88% of the GA runtime solution is consumed by the GA itself to search for the best parameter setup and the rest for the numerical resolution. Although the physical time simulated with the expert solution (0.5 s) is less than that simulated by the GA one, we believe that there is still much to do to improve the GA performance.

In fact, enlarging the search space by creating more generations and more numerous populations often makes the GA able to find out best performing solutions. This is because it provides the algorithm the capabilities of exploring further new solutions by adding new individuals through mutation to its population, and exploiting pertinent ones by making them survive several generations to ripen and reach their full potential. Nevertheless, doing so is also often time and resource consuming, since every individual needs to be assessed by the CFD software in different stages of the simulation. Thus, a high number of calls to the original fitness function and hence to the CFD software is needed, which may drastically decrease computational performance and consume important resources.

4.2 Machine-learning surrogate results

4.2.1 Steady cases

The proposed surrogate functions predict the fitness value of a solution in each iteration using a regressive machine-learning models that were previously trained on labeled data. The following Table 8 depicts the obtained results of simulations conducted to solve the "backwardstep" case after applying several machine-learning models. A comparison to the baseline simulation (without the use of the surrogate function) is also provided and the rate of the improvement is calculated.

4.2.2 Discussion

We note that fitness prediction using machine-learning models achieved a 40% reduction in total calculation time when compared to the baseline model. This is substantially achieved due to a 60% reduction in the number of calls to TransAT CFD numeric solver. These results are considered as quite significant improvement in the simulation tool that outperforms human expert performance in parameter selection. The quality of the solution (number of iterations) has, however, slightly deteriorated as the number of iterations increases by 13%. But, this hasn't affected the physical correctness of the solution.

Table 8 Results of the integration of the aggregate model to the GA with the "backwardstep" steady case

Algorithm	Nb GA	GA time	Iterations	Total time	GA impact	Nb TransAT	Nb FP
Without prediction	2.8	350.2	516.8	506	0.65	28	0
Regression trees	3.2	226.4	571.2	386.8	0.54	18.6	12.2
Bagging regressor	1.4	114.2	631	273.4	0.39	9.2	4.8
Extra trees regressor	1.4	103	546.2	237.4	0.43	8.4	5.6
Random forest regressor	1.8	146	594	299	0.44	11.6	6.4
Mean without prediction	2.8	350.2	516.8	506	0.65	28	0
Mean with prediction	1.95	147.4	585.6	299.15	0.45	11.95	7.25
Improvement (%)	30.4	58	− 13.3	40.9	31.4	57.3	

The baseline simulation conducted without prediction model is compared to a mean result of the performances of different machine-learning regression models. Comparison is based on the number of times the GA is called (Nb GA), the runtime of the GA including predictions (GA time), the number of iterations of the retained solution (Iterations), the global simulation runtime until convergence (Total time), the time portion allocated to the GA with respect to the total simulation time (GA Impact), the number of times the TransatAT is called to evaluate the retained solution (Nb TransAT), and the number of times the prediction model is trained for fitness prediction (Nb FP)

Comparison is provided on the basis of the mean performance of all tested regression models. Nevertheless, if we compare the best results found by *extra trees regressor* to the baseline, the former significantly outperforms the latter.

Indeed, random forest and extra trees differ in the sense that the splits of the trees in the former case are deterministic whereas they are random in the latter case. According to bias/variance analysis shown by Geurts et al. (2006), extra trees work by decreasing variance while at the same time increasing bias. When the randomization is increased above the optimal level, variance decreases slightly while bias increases often significantly. This technique is quite efficient particularly in problems where the proportion of irrelevant attributes is high, the case of our study. This is why extra trees performs the best among the others.

4.2.3 Unsteady cases

Similarly, we applied the same fitness prediction using the proposed surrogate model to the "Capillary Tube" case. Results are depicted in Table 9. Even if the unsteady cases are physically more complex than steady cases, quite significant results are also achieved.

4.2.4 Discussion

Similar to steady cases, fitness prediction in unsteady cases using machine-learning models achieved an 8% reduction in total calculation time when compared to the baseline model. This is substantially achieved due to a 24% reduction in the number of calls to TransAT CFD numeric solver. These results are considered as quite a significant improvement in the simulation tool as manual parameter selection in unsteady

Table 9 Results of the integration of the aggregate model to the GA with the "Capillary Tube" unsteady case

Algorithm	Nb GA	GA time	N	Total time	GA impact	Physical time	Nb TransAT	Nb FP
Without prediction	2.6	3645.4	545.6	6628.4	0.55	0.5	54	0
Regression trees	2.4	2405.8	614.2	5665.2	0.41	0.5	36	2.4
Bagging regressor	2.2	2726	599.2	6313.6	0.41	0.5	37	3
Extra trees regressor	2.6	2509.4	569.6	6012	0.39	0.5	39	2.6
Random forest	3	3480.4	554.8	6525.2	0.47	0.5	53	3.8
Without prediction	2.6	3645.4	545.6	6628.4	0.55	0.5	54	0
With prediction	2.55	2780.4	584.45	6129	0.42	0.5	41.25	2.95
Improvement (%)	2	24	− 7	8	23	0	24	

The baseline simulation conducted without prediction model is compared to a mean result of the performances of different machine-learning regression models. Comparison is based on the number of times the GA is called (Nb GA), the runtime of the GA including predictions (GA time), the number of time steps (N), the global simulation runtime until convergence (Total time), the time portion allocated to the GA with respect to the total simulation time (GA Impact), the physical time of the simulation (physical Time), the number of times the TransatAT is called to evaluate the retained solution (Nb TransAT), and the number of times the prediction model is trained for fitness prediction (Nb FP) simulations is considered as a hard and time-consuming task conducted basically using a trial-and-error approach. The quality of the solution (number of iterations) has, however, slightly deteriorated as the number of iterations increases by 7%. But, this hasn't affected the physical correctness of the solution.

Comparison is provided on the basis of the mean performance of all tested regression models. Nevertheless, if we compare the best results found by *extra trees regressor* to the baseline, the former largely outperforms the latter for the same reasons described above in steady cases.

5 Conclusion

As we have shown in the first part of this paper, evolutionary and genetic approach has enhanced the performance and the quality of parameter selection in numeric simulation in both steady and unsteady simulation cases. We have formulated the problem of genetic optimization using a heuristic fitness function defined by the distance of the solution to the target, its quality and its stability.

However, it was noticed that the calculus of such original fitness function during simulation is a time-consuming task. To overcome these limitations, we proposed a machine-learning approach for estimating the fitness. The proposed solution has significantly improved the simulation runtime by a ratio varying between 8 and 40% according to the tested case.

Nevertheless, the application of machine-learning techniques is not free of challenges, as performance drastically varies according to the adopted strategies for the selection of training data and learning algorithms. The obtained results in this project are relevant for the use cases tested here. Moreover, customized tools are provided for human operator in order to adapt machine-learning strategies to the simulation context. This work is now deployed and integrated in the SarP module of the TransAt CFD software and currently user exploitation by users.

Acknowledgements This work was funded by the Swiss Commission for Technology and Innovation (CTI) Project No. 17412.1 PFIW-IW.

Compliance with ethical standards

Conflict of interest All authors declare that they have no conflict of interest.

Ethical approval This article does not contain any studies with human participants or animals performed by any of the authors.

Open Access This article is distributed under the terms of the Creative Commons Attribution 4.0 International License (http://creativecommons.org/licenses/by/4.0/), which permits unrestricted use, distribution, and reproduction in any medium, provided you give appropriate credit to the original author(s) and the source, provide a link to the Creative Commons license, and indicate if changes were made.

References

Alba E, Cotta C, Troya JM (1996) Type-constrained genetic programming for rule-base definition in fuzzy logic controllers. In: Proceedings of the 1st annual conference on genetic programming. MIT Press

Altman NS (1992) An introduction to kernel and nearest-neighbor nonparametric regression. Am Stat 46(3):175–185

Asouti VG, Kyriacou SA, Giannakoglou KC (2016) PCA-enhanced metamodel-assisted evolutionary algorithms for aerodynamic optimization. In: Application of surrogate-based global optimization to aerodynamic design. Springer, pp 47–57. http://link.springer.com/chapter/10.1007/978-3-319-21506-8_3. Accessed 14 June 2017

Barros M, Guilherme J, Horta N (2007) GA-SVM feasibility model and optimization kernel applied to analog IC design automation. In: Proceedings of the 17th ACM Great Lakes symposium on VLSI (GLSVLSI'07), 11–13 March, Stresa-Lago Maggiore, Italy

Breiman L (1996a) Stacked regressions. Mach Learn 24(1):41–64

Breiman L (1996b) Bagging predictors. Mach Learn 26(2):123–140

Brownlee AEI, Wright JA (2015) Constrained, mixed-integer and multi-objective optimisation of building designs by NSGA-II with fitness approximation. Appl Soft Comput 33:114–126

Buche D, Schraudolph NN, Koumoutsakos P (2005) Accelerating evolutionary algorithms with Gaussian process fitness function models. IEEE Trans Syst Man Cybern Part C (Appl Rev) 35(2):183–194

Deb K et al (2002) A fast and elitist multiobjective genetic algorithm: NSGA-II. IEEE Trans Evol Comput 6(2):182–197

Dragojlovica Z, Kaminskib DA (2004) A fuzzy logic algorithm for acceleration of convergence in solving turbulent flow and heat transfer problems. Numer Heat Transf Part B Fundam Int J Comput Methodol 46(4):301–327

Fabritius B (2014) Application of genetic algorithms to problems in computational fluid dynamics. Dissertation

Forrester AIJ, Bressloff NW, Keane AJ (2006) Optimization using surrogate models and partially converged computational fluid dynamics simulations. In: Proceedings of the royal society of London A: mathematical, physical and engineering sciences, vol 462, no 2071. The Royal Society

Friedman J, Hastie T, Tibshirani R (2001) The elements of statistical learning. Springer series in statistics, vol 1. Springer, New York

Gen M, Cheng R (2000) Genetic algorithms and engineering optimization, vol 7. Wiley, Hoboken

Geurts P, Ernst D, Wehenkel L (2006) Extremely randomized trees. Mach Learn 63(1):3–42

Giannakoglou KC (2002) Design of optimal aerodynamic shapes using stochastic optimization methods and computational intelligence. Prog Aerosp Sci 38(1):43–76

Goldberg DE (2006) Genetic algorithms. Pearson Education India, London

Haupt RL, Haupt SE, Haupt SE (1998) Practical genetic algorithms, vol 2. Wiley, New York

Huang L (2004) Optimization of blowing and suction control on NACA0012 airfoil using genetic algorithm with diversity control. Ph.D. thesis, University of Kentucky

Kotragouda NB (2007) Application of genetic algorithms and CFD for flow control optimization. Dissertation

Kyriacou SA, Asouti VG, Giannakoglou KC (2014) Efficient PCA-driven EAs and metamodel-assisted EAs, with applications in turbomachinery. Eng Optim 46(7):895–911

Lee ML, Takagi H (1993) Dynamic control of genetic algorithms using fuzzy logic techniques. In: Forrest S (ed) Proceedings of the 5th international conference on genetic algorithms. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, pp 76–83

Llor X, Sastry K, Yu T-L, Goldberg DE (2007) Do not match, inherit: fitness surrogates for genetics-based machine learning techniques. In: Proceedings of the 9th annual conference on genetic and evolutionary computation (GECCO '07), 07–11 July, London, UK

Marco N, Désidéri J-A, Lanteri S (1999) Multi-objective optimization in CFD by genetic algorithms. Dissertation, INRIA

Matloff N (2017) Statistical regression and classification: from linear models to machine learning. CRC Press, Boca Raton

Morbitzer C, Strachan P, Simpson C (2003) Application of data mining techniques for building simulation performance prediction analysis. In: Proceedings of the 8th international IBPSA conference Eindhoven, Netherlands, 11–14 August

Pedregosa F et al (2011) Scikit-learn: machine learning in Python. J Mach Learn Res 12:2825–2830

Raschka S, Mirjalili V (2017) Python machine learning. Packt Publishing Ltd, Birmingham

Ross QJ (1986) Induction of decision trees. Mach Learn 1(1):81–106

Schmidt MD, Lipson H (2008) Predicting solution rank to improve performance. In: Proceedings of the 12th annual conference on genetic and evolutionary computation (GECCO '10), 12–16 July, Atlanta, GA, USA

Shirayama S (2005) The framework of a system for recommending computational parameter choices. In: New developments in computational fluid dynamics. Notes on numerical fluid mechanics and multidisciplinary design (NNFM), vol 90, pp 186–197

Voutchkov I, Keane A (2010) Multi-objective optimization using surrogates. In: Computational intelligence in optimization. Springer, Berlin, pp 155–175

Whitley D (1994) A genetic algorithm tutorial. Stat Comput 4(2):65–85

Witten IH, Frank E, Hall MA, Pal CJ (2016) Data mining: practical machine learning tools and techniques. Morgan Kaufmann, Burlington

Zitzler E, Laumanns M, Thiele L (2001) SPEA2: improving the strength Pareto evolutionary algorithm. TIK-report 103

Zvoianu AC, Bramerdorfer G, Lughofer E, Silber S, Amrhein W, Klement EP (2013) Hybridization of multi-objective evolutionary algorithms and artificial neural networks for optimizing the performance of electrical drives. Eng Appl Artif Intell 26(8):1781–1794

Publisher's Note Springer Nature remains neutral with regard to jurisdictional claims in published maps and institutional affiliations.