An Adaptive Computational Intelligence Algorithm for Simulation-driven Optimization Problems

Yoel Tenne
Formerly with the Department of Mechanical Engineering and Science,
Faculty of Engineering,
Kyoto University,
Kyoto, Japan
email: ytennex-e04@yahoo.com

Kazuhiro Izui
Department of Mechanical Engineering and Science,
Faculty of Engineering,
Kyoto University,
Kyoto, Japan
email: izui@prec.kyoto-u.ac.jp

Shinji Nishiwaki
Department of Mechanical Engineering and Science,
Faculty of Engineering,
Kyoto University,
Kyoto, Japan
email: shinji@prec.kyoto-u.ac.jp

***Abstract*—Modern engineering design optimization often evaluates candidate designs with computer simulations. In this setup, there will often exist candidate designs which cause the simulation to fail and would have no objective value assigned to them. This, in turn, can degrade the effectiveness of the design optimization process and lead to a poor final result. To address this issue, this paper proposes a new computational intelligence optimization algorithm which incorporates a classifier into the optimization process. The latter predicts which candidate designs are expected to cause a simulation failure, and its prediction is used to bias the search towards candidate designs for which the simulation is expected to succeed. However, the effectiveness of this approach depends on the classifier being used, but it is typically not known a-priori which classifier best suits the problem being solved. To address this issue, the proposed algorithm employs a statistically rigorous procedure to autonomously select the classifier type, and to adjust the classifier selection procedure with the goal of improving its accuracy. A performance analysis with a simulation-driven design problem demonstrates the effectiveness of the proposed algorithm.**

Index Terms*—expensive optimization problems; computational intelligence; modelling; classification; model selection.

I. Introduction

Nowadays engineers often use *computer simulations* to evaluate candidate designs, with the goal of reducing the duration and cost of the product design process. Such simulations, which must be properly validated with laboratory experiments, transform the design process into an optimization problem having three distinct features [2]:

- The simulation acts as the objective function, namely, it assigns candidate designs their corresponding objective values. However, the simulation is often a legacy code or a commercial software whose inner workings are inaccessible to the user, and so an analytic expression for this function is unavailable. Such a *black-box function* precludes the use of optimizers which require an analytic function.
- Each simulation run is *computationally expensive*, that is, it requires considerable computer resources, and this severely restricts the number of candidate designs which can be evaluated.
- Both the real-world physics being modelled, and the numerical simulation process, may result in an objective function having a complicated nonconvex landscape, which makes it difficult to locate an optimum.

Accordingly, such optimization scenarios are commonly termed in literature as *expensive black-box optimization problems* [2].

A framework which has proven effective in such challenging problems is that of *metamodel-assisted computational intelligence* (CI) *algorithms*. It combines a *metamodel* which approximates the expensive black-box function and provides predicted objective values at a much lower computational cost, with a *CI* optimizer which seeks an optimum of the metamodel. Due to its explorative nature, a CI optimizer often performs well in challenging nonconvex landscapes.

While the above optimization framework has proven effective, simulation-driven optimization problems often present another challenge, namely, some candidate designs will cause the simulation to fail, and would therefore not provide the expected objective value. We refer to such designs as *simulator-infeasible* (SI), while those for which the simulation completes successfully are termed *simulator-feasible* (SF). SI designs have two main implications on the optimization search:

- Since they do not have a corresponding objective value, the objective function becomes discontinuous, and this exacerbates the difficulty of the optimization search.

and

- Such designs can consume a large portion of the allotted computational resources without providing any objective values, and can therefore degrade the search effectiveness and lead to a poor final result.

A fundamental assumption in this study is that the simulation failures are caused by an unknown limitation of the simulation code, and that they are not random. This implies that repeated evaluations of a SF candidate solution will consistently succeed, while repeated evaluations of a SI candidate solution will consistently fail. Limitations of the simulation code can be attributed to a variety of reasons, for example, the inability to handle complex geometries, or an attempt to simulate physical conditions which are not supported by the numerical approximations employed in the simulation.

Based on the description so far, we summarize the underlying settings and core assumptions on which this paper is based:

- The optimization problem involves a black-box objective function which is computationally expensive to evaluate.
- The black-box objective function may have a complicated nonconvex landscape which exacerbates the optimization difficulty.
- Some candidate solutions will cause the simulation to fail, namely it will return no objective value. Such failures are nonrandom, but their cause is unknown.

Numerous studies have referred to such simulation failures and the difficulties they introduce into the optimization search, for example, Büche et al. [3], Okabe [4], and Poloni et al. [5]. The multitude of such references indicates that SI candidate designs are common in real-world applications, and therefore that it is important to effectively handle them. Two main strategies for handling SI vectors include discarding such vectors altogether, or assigning them a penalized objective value and then incorporating them into the metamodel. However, both of these strategies have significant demerits, for example, they discard information which can be beneficial to the search, or they result in a metamodel whose landscape is severely deformed.

In these settings, this study proposes a new approach in which a metamodel-assisted CI algorithm incorporates a *classifier* into the optimization search. The role of the classifier is to predict if a candidate design is SI or not, and its prediction is then used to bias the search towards candidate designs predicted to be SF. However, the effectiveness of this approach depends on the type of classifier being used. Typically, it is not known prior to the optimization search which classifier best suits the problem being solved, while an unsuitable classifier can degrade the search effectiveness. To circumvent this, this study employs a procedure which autonomously selects the most suitable classifier type during the search, based on the statistical procedure of *cross-validation* (CV). To further enhance this procedure, the proposed algorithm also calibrates during the search the *split ratio* parameter related to this procedure.

To the best of our knowledge, such a computational intelligence algorithm which incorporates a metamodel and a classifier, and which autonomously selects the classifier type and calibrates the CV procedure, is new. To evaluate its effectiveness, the proposed algorithm was tested using a representative simulation-driven problem of airfoil shape optimization. Analysis of the test results demonstrates the effectiveness of the proposed algorithm, and the contribution of the proposed classifier selection procedure.

The remainder of this paper is as follows: Section II provides the pertinent background information, Section III describes in detail the proposed algorithm, and Section IV provides an extensive performance analysis. Lastly, Section V concludes this paper.

II. Background

This section provides background information on expensive optimization problems, SI vectors in optimization, and statistical accuracy estimation.

A. Expensive optimization problems

As mentioned in Section I, expensive optimization problems are common in engineering, and Figure 1 shows the layout of such problems in which the simulation is viewed as a black-box function, namely, it assigns objective values to candidate designs, while its analytic expression is unknown. In this setup, the candidate designs are represented as vectors of design variables, and are provided as inputs to the simulation. Overall, such optimization problems arise in domains ranging from the design of electronic devices to the design of aircraft, and a representative problem is described in Section IV-A.

Also as mentioned, the resultant objective function often has a complicated, nonconvex landscape, which can lead gradient-based optimizers to converge to a poor final result. This has motivated the use of CI optimizers in such problems, as they tend to be more explorative, and hence often perform better in complicated nonconvex objective landscapes. Such optimizers typically employ a *population* of candidate solutions and manipulate them using a variety of operators. One such widely used CI optimizer, which is also employed in this study, is the *evolutionary algorithm* (EA), whose mechanics are inspired by the paradigms of adaptation and survival of the fittest. A baseline EA applies the following operators [6]:

- Selection: The candidate solutions (vectors) with the best objective value are selected as *parents*.
- Recombination: Two parents are selected, and their vectors are combined to yield an offspring. This is repeated several times to generate a population of offspring.
- Mutation: Offspring are selected at random, and some of their vector components are randomly changed.

The offspring population is then evaluated, and the fittest candidate solutions, namely, those with the best objective values, are taken to be the population of the next ‘generation’. The process then repeats until a termination criterion is met, for example, if the maximum number of generations has been reached. Through these operators, the EA drives the population to adapt to the function landscape, and to converge to an optimum. While the above description is representative of many EAs in literature, other variants have been proposed which may employ different operators. Algorithm 1 gives a pseudocode of a baseline EA.

Since CI optimizers directly evaluate candidate solutions and do not use gradient information, they often require many thousands of function evaluations to yield a satisfactory solution. This is a major obstacle in applying them to expensive optimization problems, where the objective function can be evaluated only a small number of times. As mentioned in Section I, an established framework to circumvent this is to

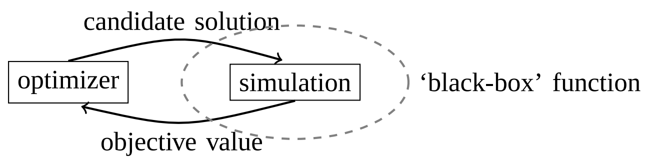


Fig. 1. The layout of an expensive black-box optimization problem. The optimizer generates candidate solutions, and these are evaluated by the simulation to obtain their corresponding objective values. The optimizer views the simulation as a black-box function, that is, having no analytic expression.

Algorithm 1: A baseline evolutionary algorithm (EA)

```
initialize a population of candidate solutions;
evaluate each candidate solution in the population;
/* main loop                                          */
repeat
    select a group of candidate solutions and designate
    them as parents;
    recombine the parents to create offspring;
    mutate some of the offspring;
    evaluate the offspring;
    select the candidate solutions which will comprise
    the population of the next generation;
until convergence, or maximum number of generations
reached;
```

employ a metamodel which approximates the true expensive function and provides the optimizer with predicted objective values at a much lower computational cost. Metamodels are typically interpolants trained with previously evaluated vectors, and variants include artificial neural networks, Kriging, polynomials, and radial basis functions (RBF) [7]. Numerous metamodel-assisted CI algorithms have proposed, and examples include the Kriging assisted EA by Ratle [8] which is also described in Section IV-C, and Booker et al. [9] which coupled a pattern search optimizer with quadratic metamodels. Later examples include Emmerich et al. [10] which used an evolutionary strategies (ES) optimizer coupled with a Kriging metamodel, and Liang et al. [11] which coupled an EA with a least-square fit polynomial metamodel. Poloni et al. [5] and Muyl et al. [12] studied algorithms coupled with an artificial neural networks (ANN). Later, Büche et al. [3] studied an ES optimizer assisted with a Kriging metamodel, and employed an elaborate sampling scheme which sought solutions based on different trade-offs of exploration-exploitation, as described in Section IV-C. More recent examples include Tenne and Armfield [13], Neri et al. [14] and Zhou et al. [15]. Given the established effectiveness of the metamodelling framework, it is also employed in this study.

While metamodels address the issue of computationally expensive evaluations, they introduce the challenge of *prediction inaccuracy*. Specifically, due to the restricted number of expensive function evaluations, only a small number of vectors will be available to train the metamodel, which degrades its accuracy. In severe cases, the optimizer may even converge to a *false optimum*, namely, an optimum of the metamodel which is a not an optimum of the true expensive function [16], and it is therefore necessary to safeguard the metamodel accuracy to ensure the progress of the optimization search. The proposed algorithm accomplishes this by leveraging on the *trust-region* (TR) approach which originated in the field of nonlinear programming [17], where initially a *trial step* is performed to seek an optimum of the metamodel in the TR, namely, the region where the metamodel is assumed to be accurate. Next, the TR and metamodel are updated based on the optimum found, and the process repeats until a termination condition is met. A merit of the TR approach is that it ensures asymptotic convergence to an optimum of the true expensive function [17]. Section III gives a detailed description of the TR approach implemented in this study.

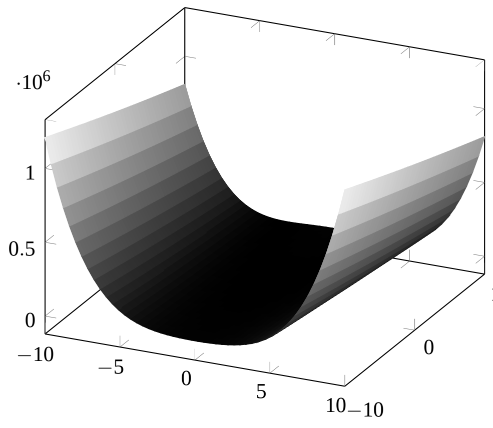
B. Simulator-infeasible vectors

As mentioned in Section I, this study focuses on expensive optimization problems with simulator-infeasible (SI) vectors, namely, which cause the simulation to fail. A multitude of studies have referred to such vectors and to the difficulties they introduce into the optimization search. For example, Poloni et al. [5] described an optimization problem which involved a computational fluid dynamics analysis, and noted that some candidate designs caused "failure of the simulation code". In another study, Booker et al. [9] described a rotor blade structural optimization problem in which "attempts to evaluate the objective function failed". Similarly, Büche et al. [3] described an aerodynamics shape optimization problem in which "evaluation of all points fails". Additional pertinent studies include Liang et al. [11], Conn et al. [18] and Okabe [4].

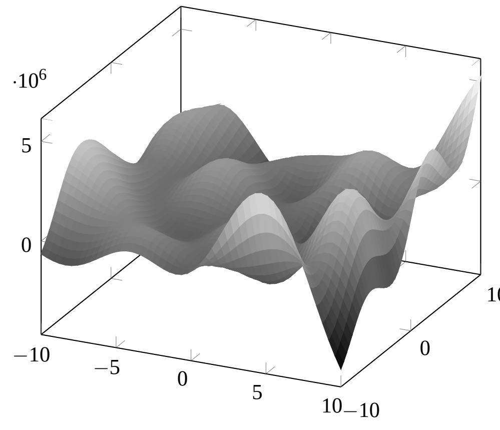
Several techniques have been explored in an effort to handle SI vectors. For example, Rasheed et al. [19] described an aircraft design optimization problem in which an EA directly called the expensive simulation, and no metamodels were employed. A classifier was used to screen candidate design prior to the simulation call, and those predicted to be SI were assigned a 'death penalty', namely, a fictitious and highly penalized objective value, to quickly eliminate them from the population, but no metamodels were employed. In another related study, Emmerich et al. [10] also used the penalty approach, but incorporated the penalized vectors into the metamodel in an attempt to bias the search towards SF vectors. In contrast, Büche et al. [3] discarded the SI vectors altogether, so that the metamodel was trained using only the SF vectors.

These strategies, and similar ones, have several demerits in the context of expensive optimization problems: a) assigning SI vectors a penalized objective value and then incorporating them into the metamodel can severely deform the metamodel landscape and degrade its accuracy, while b) discarding SI vectors results in a loss of information which might have been useful in enhancing the optimization search. As an example, Figure 2 shows the effect of penalizing SI vectors and incorporating them into a Kriging metamodel, which is described in Section III. Figure 2(a) shows the metamodel resulting from a sample of 30 SF vectors, while Figure 2(b) shows the resultant metamodel when 20 SI vectors were added to the baseline sample and were assigned the worst objective value from the baseline sample. The metamodel landscape was severely deformed and consequently locating an optimum of the true objective function became more difficult.

Such issues have motivated exploring alternative approaches for handling SI vectors. For example, Tenne and Armfield [20] proposed an approach which employed two metamodels, where one was used for approximating the objective function and another for generating a penalty which was based on the distance of a new candidate solution to previously encountered SI ones. Other studies have examined using classifiers for constrained non-linear programming, though unrelated to SI vectors [21]. Further exploring the use of classifiers, Tenne et al. [22] obtained preliminary results with a classifier-assisted algorithm for handling SI vectors. However, the algorithm used a single type of classifier, and it did not attempt to select the classifier during the search. Recently, Tenne et al. [1] presented a preliminary investigation on a framework which adapts the classifier type based on the problem being solved. The present study leverages on the latter framework and extends it by proposing to also adapt the CV split ratio used in the classifier selection step. The present study also provides a more extensive performance analysis.



(a)



(b)

Fig. 2. An example of the effect of SI vectors on the metamodel. The objective function was Rosenbrock, whose optimum is at (1, 1). (a) shows a Kriging metamodel trained using a sample of 30 SF vectors, and (b) shows the resultant metamodel when 20 SI vectors were added to the sample and were assigned the worst objective value of the sample in (a). The landscape of the resultant metamodel was severely deformed, and the optimum of the true objective function was masked.

C. Accuracy estimation

As mentioned in Section I, the proposed algorithm employs a classifier to predict which candidate designs will cause the simulation to fail, and to improve the effectiveness of this approach, it selects the classifier deemed most accurate out of a family of candidates.

Accuracy estimation is rigorously addressed in the general statistical framework of *model selection*, in which a *model* refers to any functional relation which is used to explain an inputs–outputs relation [23]. In the model selection procedure, several candidate models are prescribed and their accuracy is estimated, after which the model deemed as the most accurate is selected as the optimal one. An established procedure for estimating the model accuracy is that of *cross-validation* (CV), in which the sample of vectors is split into a *training sample* and a *testing sample*. A candidate model is trained using the former, and its predictions for the testing vectors are compared to their already known exact function values.

The CV procedure relies on the *split ratio* parameter, which determines which portion of data set will be designated as the training sample and which as the testing sample. This suggests that the accuracy of the procedure will be affected by the split ratio used. To verify this, the CV procedure was used to estimate the accuracy of two candidate classifiers, namely, *k nearest neighbours* (*k*NN) and *support vector machine* (SVM), whose details are given in Appendix A, and the tests were performed using the two well-established data sets `iris` and `yeast` provided by Frank and Asuncion [24]. The accuracy measure used was the total classification error, namely, the number of items in the testing sample to which the classifier assigned an incorrect class. To check the effect of different split ratios, the full data set was initially split in a 80–20 training-testing ratio, and the accuracy of each classifier was estimated. The accuracy estimates from this step are considered as the reference results, since they employed the full data set. Next, the training sample was used as the baseline sample, and the accuracy of each classifier was estimated by using each of the following training-testing split ratios in turn: 0.8–0.2, 0.5–0.5, and 0.2–0.8 . Table I shows the test results and the rankings of the two classifiers. It follows that the rankings corresponding to the 0.5–0.5 and 0.2–0.8 split ratios matched those obtained with the full sample, while those of the 0.8–0.2 split ratio differed. This in turn verifies the above assumption, namely, that the split ratio affected the accuracy of the CV procedure.

Since the optimal split ratio is unknown prior to the accuracy estimation step, it is possible that an unsuitable value would be used, which in turn would degrade the accuracy of the CV procedure. To circumvent this, the proposed algorithm employs a procedure to autonomously select a suitable split ratio, as described in Section III.

III. Proposed algorithm

This section describes the proposed algorithm in detail, and explains how it addresses the optimization challenges discussed in Sections I and II. The algorithm leverages on three paradigms:

- Classification of candidate vectors: Each candidate vector is treated as having two attributes, namely, its *objective value*, which is predicted by a metamodel, and its *class*, namely, if it is SI or SF, which is predicted by a classifier.
- Selection of the classifier type: Typically, it is not known prior to the optimization search which classifier type is most suitable to the problem being solved. To circumvent this, during the optimization search the proposed algorithm uses the CV procedure to autonomously select the most suitable type of classifier. To further improve the accuracy of this approach, the proposed algorithm also continuously selects during the search the most suitable split ratio value.
- Trust-region (TR) optimization: Given the inherent metamodel inaccuracy, a TR framework is employed to ensure convergence to an optimum of the true expensive function.

The proposed algorithm operates in five main steps: initialization, training a metamodel, selecting the classifier type and training a corresponding classifier, performing a TR trial step to seek an optimum, and performing the TR updates. The details of these steps are as follows:

Step 1) *Initialization*: The proposed algorithm begins by generating an initial sample of vectors using a Latin hypercube (LH) design of experiments [25]. This is a statistically oriented sampling method which ensures that the sample is space-filling, namely, that the vectors are distributed throughout the search space, which improves the accuracy of the resultant metamodel. A sample of s vectors is generated as follows. The range of each variable is split into s equally sized intervals, and one point is sampled at random in each interval. Next, a sample point is selected at random and without replacement for each variable, and these samples are combined to produce a vector. This sequence is repeated for s times to create the complete sample, which is then evaluated with the expensive simulation and is stored in memory. After this step, the main optimization loop begins.

TABLE I
CLASSIFIER ACCURACY RANKINGS BY DIFFERENT CV SPLIT RATIOS

Split ratio	Error	Rank	Error	Rank
Full	2	1	3	2
0.8	1	2	0	1
0.5	1	1	2	2
0.2	5	1	7	2

Highlighted lines have the same ranks as those obtained with the full sample.

Step 2) *Metamodel training*: In this step, the proposed algorithm trains a metamodel by using the SF vectors stored in memory and ignores the SI vectors. In this study a Kriging metamodel was employed, based on its prevalence in literature [3, 26, 27]. This metamodel is statistically-oriented and combines two components: a 'drift' function, which is a global coarse approximation of the true expensive function, and a local correction based on the correlation between the interpolation vectors. Given a set of evaluated vectors, $\boldsymbol{x}_i \in \mathbb{R}^d$, $i = 1 \dots n$, the Kriging metamodel is trained such that it exactly interpolates the observed values, that is, $m(\boldsymbol{x}_i) = f(\boldsymbol{x}_i)$, where $m(\boldsymbol{x})$ and $f(\boldsymbol{x})$ are the metamodel and true objective function, respectively. Using a constant drift function [28] gives the Kriging metamodel

$$m(\boldsymbol{x}) = \beta + \kappa(\boldsymbol{x}), \tag{1}$$

with the drift function β and local correction $\kappa(\boldsymbol{x})$. The latter is defined by a stationary Gaussian process with mean zero and covariance

$$Cov[\kappa(\boldsymbol{x})\kappa(\boldsymbol{y})] = \sigma^2 c(\theta, \boldsymbol{x}, \boldsymbol{y}), \tag{2}$$

where $c(\theta, \boldsymbol{x}, \boldsymbol{y})$ is a user-prescribed correlation function. A common choice for the latter is the Gaussian correlation function [28], defined as

$$c(\theta, \boldsymbol{x}, \boldsymbol{y}) = \Pi_{i=1}^{d} \exp\left(-\theta (x_i - y_i)^2\right), \tag{3}$$

and combining it with the constant drift function transforms the metamodel from (1) into the following form

$$m(\boldsymbol{x}) = \hat{\beta} + \boldsymbol{r}(\boldsymbol{x})^{\mathrm{T}} \boldsymbol{R}^{-1} (\boldsymbol{f} - \boldsymbol{1}\hat{\beta}). \tag{4}$$

Here, $\hat{\beta}$ is the estimated drift coefficient, $\boldsymbol{R}$ is the symmetric matrix of correlations between all interpolation vectors, $\boldsymbol{f}$ is the vector of objective values, and $\boldsymbol{1}$ is a vector with all elements equal to 1. $\boldsymbol{r}^{\mathrm{T}}$ is the correlation vector between a new vector $\boldsymbol{x}$ and the sample vectors, namely,

$$\boldsymbol{r}^{\mathrm{T}} = [c(\theta, \boldsymbol{x}, \boldsymbol{x}_1), \dots, c(\theta, \boldsymbol{x}, \boldsymbol{x}_n)]. \tag{5}$$

The estimated drift coefficient $\hat{\beta}$ and variance $\hat{\sigma}^2$ are obtained from

$$\hat{\beta} = \left(\boldsymbol{1}^{\mathrm{T}} \boldsymbol{R}^{-1} \boldsymbol{1}\right)^{-1} \boldsymbol{1}^{\mathrm{T}} \boldsymbol{R}^{-1} \boldsymbol{f}, \tag{6a}$$

$$\hat{\sigma}^2 = \frac{1}{n}\left[(\boldsymbol{f} - \boldsymbol{1}\hat{\beta})^{\mathrm{T}} \boldsymbol{R}^{-1} (\boldsymbol{f} - \boldsymbol{1}\hat{\beta})\right]. \tag{6b}$$

Fully defining the metamodel requires the correlation parameter θ, whose optimal value, $\theta^{\star}$, is typically taken as the maximizer of the metamodel likelihood. In practise, the latter is obtained by minimizing the negative log-likelihood, namely

$$\theta^{\star} : \min - \left(n \log(\hat{\sigma}^2) + \log(|\boldsymbol{R}|)\right). \tag{7}$$

While a different correlation parameter can be used for each variable, this study follows the practise prevalent in literature in which the metamodel employs a single correlation parameter. This results in a univariate likelihood function, which is relatively easy to optimize.

Step 3) *Classifier selection and training*: In the next step, the proposed algorithm trains a classifier to predict if a vector is SF or SI. To further improve this technique, the proposed algorithm employs the CV to select during the search a classifier deemed as most suitable out of a family of candidates. To further enhance the accuracy of this procedure, the proposed algorithm employs an additional step to identify a suitable split ratio for the CV procedure, out of a prescribed set of candidate ratios. The details of the procedure are as follows:

3.1) The set of vectors stored in memory is split into sample A and sample B in a 80–20 split ratio.

3.2) Using only sample A, the proposed algorithm loops over the prescribed set of candidate split ratios, s_i, $i = 1 \dots n_s$, where n_s is the number of candidate of ratios, and for each it performs the following steps:

3.2.1) It generates a training sample and a testing sample based on sample A.

3.2.2) For each candidate type of classifier, the proposed algorithm trains a corresponding classifier using the training sample, and then estimates the classifier's accuracy by using the testing sample, where the accuracy measure is the *total classification error*, defined as

$$e = \sum_{i=1}^{l} \big(\hat{c}(\boldsymbol{x}_i) \neq F(\boldsymbol{x}_i) \big) , \tag{8}$$

where $\boldsymbol{x}_i$, $i = 1 \dots l$, are the vectors in the testing sample, $\hat{c}(\boldsymbol{x})$ is the prediction of the classifier which was trained using the training sample, and $F(\boldsymbol{x}_i)$ is the true and known class of the testing vectors. For the latter, $F(\boldsymbol{x}_i) = 1$ was used for a SF vector, and $F(\boldsymbol{x}_i) = -1$ for a SI vector.

3.2.3) The candidate classifiers are ranked based on their obtained total classification errors, which yields a vector of ranks $\boldsymbol{r}_i$, where i is the index of the current split ratio being considered.

3.3) After completing the above procedure for all candidate split ratios, the proposed algorithm loops over the set of candidate classifier types, and using the samples A and B obtained in step 3.1, it trains a classifier using sample A, and estimates the classifier's accuracy using sample B. The classifier types are ranked based on their estimated accuracies, which yields a vector of ranks $\boldsymbol{r}_0$.

3.4) The proposed algorithm selects as the optimal split ratio (s^*) the one whose corresponding ranks vector $\boldsymbol{r}_i$ is most similar to the reference ranks vector $\boldsymbol{r}_0$. This similarity is measured by the l_1 norm, namely

$$s^* = s_{i^*} , \quad i^* : \min_{i=1\dots n_s} \|\boldsymbol{r}_i - \boldsymbol{r}_0\|_1 , \tag{9}$$

where i^* is the index of the optimal split ratio. The basis for this procedure is that the most suitable split ratio should yield a ranks prediction which is relatively insensitive to the sample size. Therefore, the ranks vectors obtained in step 3.2.3, namely, $\boldsymbol{r}_i$, $i = 1 \dots n_s$, which were obtained based on the training sample derived from sample A, are compared to the ranks vector from step 3.4, namely, $\boldsymbol{r}_0$, which was obtained based on the full set of vectors stored in memory.

3.5) After identifying the most suitable split ratio, the proposed algorithm selects the classifier type which had the lowest prediction error in the selected split ratio, and trains a classifier, designated as $c(\boldsymbol{x})$, using all the vectors stored in memory. This classifier is then used in the optimization search performed in step 4.

In this study the proposed algorithm selected between three classifiers types: *k nearest neighbours* (*k*NN), *linear discriminant analysis* (LDA), and *support vector machine* (SVM), whose details are given in Appendix A. The candidate values for the training-testing split ratios were 0.8–0.2, 0.5–0.5, and 0.2–0.8.

Step 4) *TR trial step*: The best vector in the memory storage is taken as the TR centre ($\boldsymbol{x}_\text{b}$), and a TR trial-step is performed, namely, an optimizer is invoked to find an optimum in the bounded region

$$\mathscr{T} = \{\boldsymbol{x} : \|\boldsymbol{x} - \boldsymbol{x}_\text{b}\|_2 \leqslant \Delta\} , \tag{10}$$

where Δ is the TR radius. The optimizer used is the real-coded EA of Chipperfield et al. [29], which follows the setup described in Section II-A, namely, it begins by selecting a set of parents, recombines them to produce offspring, mutates some of the offspring, and selects the population of the next generation from the union of the offspring and the best parents. Table II gives the complete parameter settings of this EA, which are based on those suggested in literature [29, 30].

During the trial step, the EA uses the following *modified objective function* which combines the prediction of the metamodel from Step 2 and the classifier from Step 3, as follows

$$\hat{m}(\boldsymbol{x}) = \begin{cases} m(\boldsymbol{x}) & \text{if } c(\boldsymbol{x}) \text{ is SF} \\ p & \text{if } c(\boldsymbol{x}) \text{ is SI} \end{cases} \tag{11}$$

TABLE II
INTERNAL PARAMETERS OF THE EA UTILIZED IN THIS STUDY [29]

Population size	100
Generations	100
Selection	Stochastic universal selection (SUS)
Recombination	Intermediate, applied with probability $p = 0.7$
Mutation	Breeder Genetic Algorithm (BGA) mutation, applied with probability $p = 0.1$
Elitism	10%

namely, the EA receives the objective value predicted by the metamodel $m(\boldsymbol{x})$ if the classifier predicts a vector is SF, but it receives the penalized objective value p otherwise. The latter is taken as the worst objective value in the initial LH sample.

This formulation enhances the optimization search in two main aspects. First, the classifier accumulates the information about the SI vectors encountered during the search and uses it to predict the distribution of such vectors in the search space, so this potentially beneficial information is not discarded. Second, the classifier's prediction is used to bias the search but without affecting the metamodel landscape, and this avoids the issues discussed in Section II-B. To visualize the effect of this setup, and to demonstrate the predictions of the metamodel, the classifier, and how they are combined into a single modified objective function, Figure 3 gives a synthetic example with the Rosenbrock function. The plots show that the landscape predicted by the modified objective function closely follows that of the baseline metamodel, *and* embeds the knowledge on the location of the SI vectors, but it is only minimally deformed.

Step 5) *TR updates*: The optimum found by the EA, $\boldsymbol{x}^\star$, is evaluated with the true expensive function, which yields the exact objective value $f(\boldsymbol{x}^\star)$. Following the classical TR framework [17], the proposed algorithm performs the following updates:

- If $f(\boldsymbol{x}^\star) < f(\boldsymbol{x}_\mathrm{b})$: The trial step was successful since the predicted optimum is indeed better than the current best solution, namely, $\boldsymbol{x}_\mathrm{b}$. Accordingly, the TR is centred at the new optimum, and the TR is enlarged by doubling its radius.
- If $f(\boldsymbol{x}^\star) \geqslant f(\boldsymbol{x}_\mathrm{b})$ *and* there are sufficient SF vectors inside the TR: The search was unsuccessful since the predicted optimum is not better than the current best vector. However, since there are sufficient SF vectors in the TR, the metamodel is deemed as being sufficiently accurate to justify contracting the TR. Accordingly, the TR is contracted by halving its radius.
- If $f(\boldsymbol{x}^\star) \geqslant f(\boldsymbol{x}_\mathrm{b})$ *and* there are insufficient SF vectors inside the TR: The search was unsuccessful, but this may be since the metamodel is inaccurate due to an insufficient number of SF vectors in the TR. Therefore, the algorithm samples new vectors ($\boldsymbol{x}_\mathrm{n}$) inside the TR, as explained below.

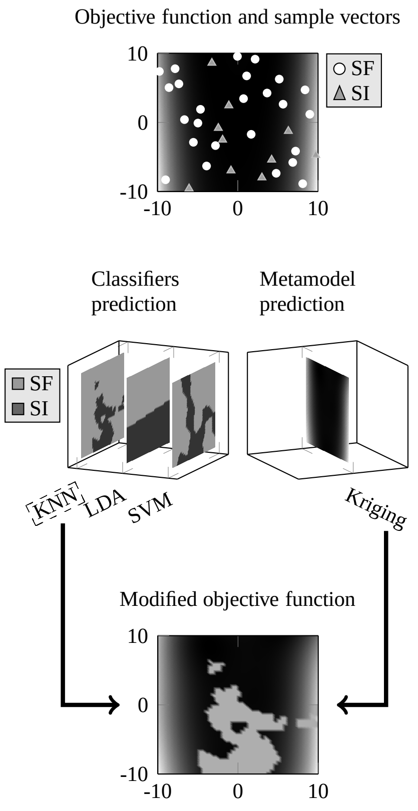


Fig. 3. An example showing how the proposed algorithm generated the modified objective function. The objective function was Rosenbrock, and the sample was comprised of 26 SF vectors and 9 SI vectors. The proposed algorithm trained a Kriging metamodel using the SF vectors, and trained the classifiers using the entire sample. The *k*NN classifier was deemed as the most accurate, and therefore its prediction was used in the modified objective function. The landscape of the latter was modified based on the predictions regarding SI vectors, but it was only minimally deformed.

As a change from the classical TR framework, the proposed algorithm contracts the TR only if it contains a sufficient number of SF vectors, to avoid a too rapid TR contraction and premature convergence [17]. To select a suitable threshold value (q) for the number of these vectors, numerical experiments have been performed and are described in Section IV-B.

Another change from the classical TR framework is the sampling of new vectors to improve the accuracy of the metamodel in the TR. There are two considerations in selecting these vectors: i) they should improve the metamodel accuracy locally around the current optimum, and alternatively ii) they should improve the metamodel accuracy over the entire TR, and particularly in regions sparse with vectors [31]. Since these are typically two opposing considerations, the proposed algorithm generates several new vectors which correspond to different trade-offs between these considerations. The vectors are taken as the minimizers of the following objective function

$$h(\boldsymbol{x}) = wh_1(\boldsymbol{x}) + (1-w)h_2(\boldsymbol{x}), \tag{12}$$

where the minimization is performed by the real-coded EA described earlier. Here, $h_1(\boldsymbol{x})$ is the *rank* of the vector $\boldsymbol{x}$ based on its objective value, such that the best vector in the EA population is assigned a rank of 1, the following one a rank of two, and so on. Also, $h_2(\boldsymbol{x})$ is the *rank* of the vector $\boldsymbol{x}$ based on its distance from existing vectors in the TR, where the vector in the EA population which is farthest is given a rank of one, the vector having the 2*nd* largest distance is given a rank of two, and so on. The weight w defines the trade-off between the two considerations, where $w = 1$ implies a vector is searched based only on its objective value, which will result in the new vector being in the vicinity of the TR centre, while $w = 0$ implies a vector is searched based only on its distance to existing vectors in the TR, which will result in a vector being away from existing vectors. Equation (12) uses a rank based approach to make the search more consistent across different objective functions. To identify suitable weights, numerical experiments have been performed and are described in Section IV-B.

To complete the algorithm description, several additional points are noted:

- While in the description above the proposed algorithm used a Kriging metamodel and selected between a *k*NN, linear discriminant analysis (LDA), and SVM classifiers, other types of metamodels and classifiers can be readily used.
- To avoid a numerical breakdown of the metamodel training process, the proposed algorithm evaluates a new vector and adds it to the memory storage only if it differs from those already stored.
- There is some computational overhead introduced by the proposed algorithm due to the classifier selection step. However, since no expensive evaluations are involved in this step, and since the classifiers' training phase is computationally cheap, the overhead is minimal.

To complete the description of the proposed algorithm, Figure 4 gives a schematic layout of its optimization iteration, and Algorithm 2 gives its pseudocode.

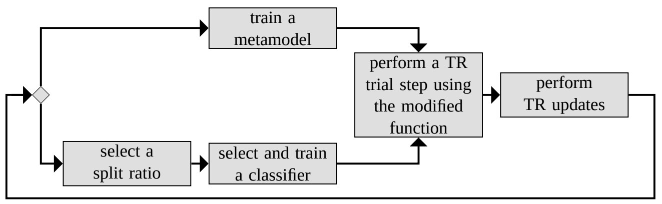

train a metamodel
perform a TR trial step using the modified function
perform TR updates
select a split ratio
select and train a classifier

Fig. 4. The layout of an optimization iteration of the proposed algorithm. The iteration begins by training a metamodel and classifier, including selection of the classifier type and CV split ratio. This is followed by a TR trial step to locate an optimum of the modified objective function, and lastly updates of the TR to ensure the progress of the search.

Algorithm 2: Proposed optimization algorithm

```
/* initialization                                        */
generate an initial LH sample;
evaluate the sample vectors and store in memory;
/* main optimization loop                                */
repeat
    /* train a metamodel                                 */
    use the SF vectors stored in memory to train a metamodel;
    /* select and train a classifier                     */
    split the vectors stored in memory into a sample A and a sample B;
    for each candidate split ratio do
        split sample A into a training sample and testing sample using the candidate split ratio;
        for each candidate classifier type do
            train a classifier using the training sample;
            estimate the classifier accuracy using the testing sample;
        rank the classifiers based on their accuracies;
    for each candidate classifier type do
        train a classifier using sample A;
        estimate the classifier accuracy using sample B;
    rank the classifiers based on their accuracies to obtain a ranks vector $\boldsymbol{r}_0$;
    select the split ratio $s^*$ whose corresponding ranks vector is most similar to $\boldsymbol{r}_0$;
    select the classifier type which produced the lowest prediction error when $s^*$ was used, and train a classifier using all vectors stored in memory;
    /* perform a TR trial step                           */
    set the TR centre to the best vector stored in memory;
    use a real-coded EA to find an optimum of the modified objective function in the TR;
    /* perform TR updates                                */
    evaluate the predicted optimum with the true expensive function;
    if the new optimum is better than the best vector in memory then
        double the TR radius
    else if the new optimum is not better than the best vector in memory and there are sufficient vectors in the TR then
        halve the TR radius;
    else if the new optimum is not better than the best vector in memory and there are insufficient vectors in the TR then
        add new vectors in the TR to improve the metamodel accuracy;
    add to the memory storage all the new vectors evaluated with the true expensive function;
until maximum number of analyses completed;
```

IV. Numerical experiments

This section describes the numerical experiments used to evaluate the performance of the proposed algorithm. It begins by describing the test problem employed, it then describes a parameter sensitivity analysis which was used to select suitable settings for the algorithm parameters, and lastly it describes and analyzes a set of benchmark tests.

A. Problem description

The test problem employed was that of airfoil shape optimization, as it is both simulation-driven and contains SI vectors, as explained below. The formulation of the problem is as follows. During flight, an aircraft generates *lift*, namely, the force which counters the aircraft weight and keeps it airborne, and *drag*, which is an aerodynamic friction force obstructing the aircraft movement. Both the lift and drag result from the flow of air around the aircraft wing whose cross-section is the airfoil. The optimization goal is then to identify an airfoil shape which maximizes the lift and minimizes the drag. In practise, the design requirements for airfoils are specified in terms of the nondimensional lift and drag coefficients, c_l and c_d , respectively, defined as

$$c_l = \frac{L}{\frac{1}{2}\rho V^2 S} \tag{13a}$$

$$c_d = \frac{D}{\frac{1}{2}\rho V^2 S} \tag{13b}$$

where L and D are the lift and drag forces, respectively, ρ is the air density, V is the aircraft speed, and S is a reference area, such as the wing area. Also important is the *angle of attack* (AOA), which is the angle between the aircraft velocity and the airfoil *chord line*, defined as the straight line joining the leading and trailing edges of the airfoil. Figure 5 gives a schematic layout of the airfoil problem.

Candidate airfoils were represented with the Hicks-Henne parameterization [32], in which the profile of a candidate airfoil is defined as

$$y = y_b + \sum_{i=1}^{h} \alpha_i b_i(x)\,, \tag{14}$$

where y_b is a baseline airfoil profile, taken as the NACA0012 symmetric airfoil, b_i are basis functions, which following [33], are defined as

$$b_i(x) = \left[\sin\left(\pi x^{\frac{\log(0.5)}{\log(i/(h+1))}}\right)\right]^4 , \tag{15}$$

and $\alpha_i \in [-0.01, 0.01]$ are coefficients, which are the problem's design variables. Ten basis functions were used for the upper and lower airfoil profile, respectively, resulting in 20 design variables overall. Figure 5 shows the layout of the airfoil problem and the Hicks-Henne parametrization. The lift and drag coefficients of candidate airfoils were obtained by using *XFoil*, a computational fluid dynamics simulation for analysis airfoils operating in the subsonic regime [34]. Each airfoil evaluation required up to 30 seconds on a desktop computer. To ensure structural integrity, the thickness of an airfoil (t) between 0.2 to 0.8 of its chord length had to be equal to or larger than a critical value $t^\star = 0.1$.

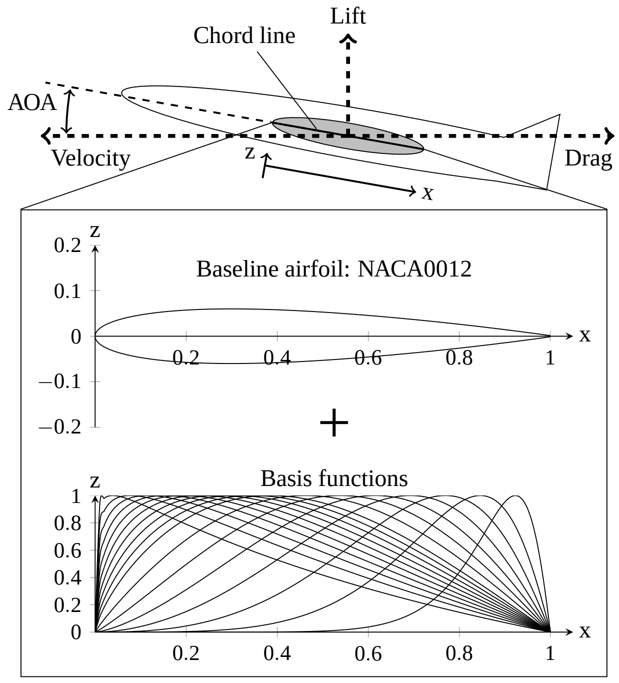


Fig. 5. A schematic layout of the airfoil problem showing the physical quantities involved, and the Hicks-Henne parameterization setup.

The airfoil shape optimization problem is a pertinent test case since it contains SI vectors, and their prevalence is strongly affected by the angle of attack (AOA) defined earlier. Specifically, since the turbulence of the flow field increases with the AOA, at higher angle values it will be more difficult to complete the aerodynamics analysis, which will result in more simulation failures, and therefore, more SI trial designs. To verify this, 30 different airfoils were sampled and evaluated in identical flight conditions, except for the AOA which was increased from 20° to 50° . Figure 6 shows the obtained results, where, as expected, the number of failed analyses increased with the AOA. Therefore, by changing the AOA we could change the density of SI vectors in the search space, and hence the relative difficulty of the tests. In view of these results, the numerical experiments included three optimization scenarios, namely, with AOA $= 20^\circ, 30^\circ$, and 40°, which following Figure 6, correspond to a low, medium and high density of SI vectors in the search space, respectively.

B. Parameter sensitivity analysis

As described in Section III, the proposed algorithm relies on two main parameters, namely:

- q: The minimum number of vectors in TR to invoke a TR contraction.

and

- w_i: The weights used for generating new vectors to improve the metamodel accuracy ($\boldsymbol{x}_{\mathrm{n}}$).

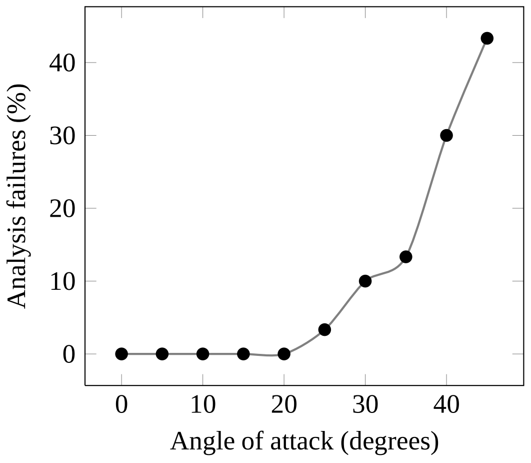


Fig. 6. Simulation failures as a function of the angle of attack (AOA).

To identify suitable values for these parameters, in turn, each parameter was assigned one of three candidate values, and ten optimization runs were repeated with the AOA $= 30^\circ$ setting.

Table III shows the obtained results, where the best mean statistic is emphasized. It follows that suitable parameter settings are a threshold value of $q = 20$, and a set of weights $w = \{0.8, 0.5, 0.2\}$, resulting in three new vectors being added in the TR to improve the metamodel accuracy. Accordingly, these settings were used in the benchmark tests described in the next section.

C. Benchmark tests: Results and analysis

For a comprehensive evaluation, the proposed algorithm was benchmarked against two representative algorithms from literature:

- *EA with Periodic Sampling* (EA–PS) [35]: The algorithm safeguards the metamodel accuracy by periodically evaluating a small subset of the population with the true objective function, and incorporating them into the metamodel. The algorithm begins by generating an initial sample of vectors (candidate solutions), evaluating them with the expensive function, and training a Kriging metamodel. It then uses a real-coded EA to seek an optimum of the metamodel, where the EA is run for 10 generations. The ten best members of the resultant population are then evaluated with the true expensive function, and are incorporated into the metamodel. This process repeats until the maximum number of expensive function evaluations is reached. In the benchmark tests, the EA was identical to the one used by the proposed algorithm and used the same parameters given in Table II.
- *Expected Improvement with a Covariance Matrix Adaptation Evolutionary Strategies optimizer* (EI–CMA-ES) [3]: The algorithm combines a covariance matrix adaptation evolutionary strategy (CMA-ES) optimizer [36] with a Kriging metamodel, and updates the latter based on the expected improvement framework [37]. The algorithm begins by generating an initial sample of vectors, and evaluates them with the true function. Its main loop then begins, where at each generation it trains a Kriging metamodel by using both the recently evaluated vectors and those stored in memory which are nearest to the best solution. A CMA-ES optimizer is then invoked to locate an optimum of the metamodel in a bounded region defined by the metamodel training sample. In the spirit of the expected improvement framework [37], the function being minimized is
$$\hat{f}(\boldsymbol{x}) = m(\boldsymbol{x}) - \rho\zeta(\boldsymbol{x}) , \qquad (16)$$
where $m(\boldsymbol{x})$ is the Kriging metamodel prediction, ρ is a prescribed coefficient, and $\zeta(\boldsymbol{x})$ is the estimated Kriging prediction error, which is zero at sampled vectors since there the true objective value is known. The search is repeated for $\rho = 0, 1, 2,$ and 4, to obtain four solutions corresponding to different search profiles, namely, ranging from a local search ($\rho = 0$) to a more explorative one ($\rho = 4$). All non-duplicate solutions found are evaluated with the true expensive function, and are stored in memory. In case no new solutions were evaluated, for example, because they already match those stored in memory, a new solution is generated by perturbing the current best one. Following Büche et al. [3], the algorithm used a training set of 100 vectors comprising of the 50 most recently evaluated ones and 50 nearest-neighbours, and the CMA-ES used the default settings given in Reference [36].

TABLE III
TEST STATISTICS FOR THE PARAMETER SENSITIVITY ANALYSIS

(a) q: Number of vectors in the TR needed to contract the TR

	2 (=0.1d)	10 (=0.5d)	20 (=d)
Mean	9.794e-01	8.772e-01	**8.568e-01**
SD	5.631e-02	6.540e-02	7.462e-02
Median	9.984e-01	8.674e-01	8.434e-01
Min(best)	8.409e-01	7.875e-01	7.527e-01
Max(worst)	1.028e+00	1.001e+00	1.005e+00

d: Dimension of objective function.

(b) w: Weights used for generating new vectors in the TR

	$\{0.8, 0.2\}$	$\{0.8, 0.5, 0.2\}$	$\{0.8, 0.6, 0.4, 0.2\}$
Mean	9.126e-01	**8.741e-01**	8.772e-01
SD	6.099e-02	7.245e-02	6.540e-02
Median	9.400e-01	8.847e-01	8.674e-01
Min(best)	7.880e-01	7.175e-01	7.875e-01
Max(worst)	9.661e-01	9.947e-01	1.001e+00

To also study the contribution of the proposed procedure of selecting the classifier type and calibrating the CV split ratio during the search, the benchmark tests also included the following two variants of the proposed algorithm which were identical to it in operation, except that they used a fixed type of classifier, and therefore did not use the procedures in question: variant i) VK: a variant which used a kNN classifier, and variant ii) VS: a variant which used an SVM classifier.

For all algorithms the limit of simulation calls was set to 200, and their initial sample was generated by a LH procedure and consisted of 20 candidate solutions. To support a valid statistical analysis, 30 runs were repeated for each algorithm at each test case. This was done only for the benchmarking purpose, and is not required in an ordinary optimization search.

For a thorough evaluation, three performance measures were analyzed: i) the final objective value obtained by each algorithm, ii) the number of SI vectors generated by each algorithm during its optimization search, and iii) the classifier type and split ratio which were selected by the proposed algorithm during its optimization search. The details of these analyses are as follows:

- *Final objective value*: To compare the effectiveness of the algorithms, Table IV gives the test statistics of mean, standard deviation (SD), median, best, and worst final objective values obtained by each algorithm in each optimization scenario, with the best mean and median statistics emphasized. The table also gives the significance level (α) at which results of the proposed algorithm were better than those achieved by the other algorithms, where the significance levels considered were 0.01, 0.05, or an empty entry otherwise. Statistical significance was determined using the Mann–Whitney nonparametric test [38, p.423–432].
  It follows that the proposed algorithm consistently performed well, as indicated by its mean and median statistics. Also, statistical significance comparisons show that it outperformed the two reference algorithms from literature, namely, EA–PS and EI–CMA-ES in the AOA=20° case, and outperformed the EA–PS algorithm in the AOA=30° and 40° cases. In contrast, the EA–PS algorithm typically achieved either the worst or second worst mean statistic, which highlights the demerit of the penalty approach it employs, namely, that incorporating penalized vectors into the training sample can result in deformation of the metamodel landscape and consequently degrade the search effectiveness. It is noted that the performance gains achieved by the proposed algorithm varied depending on the problem setting (the AOA), and were more modest in the high AOA settings where the high prevalence of SI vectors exacerbated the optimization difficultly. However, even modest performance gains can be significant, which justifies the minor added computational overhead incurred by the proposed algorithm.
  Lastly, test statistics also show the contribution of the procedure for selecting the classifier type and split ratio, as indicated by the comparisons to the two variants VK and VS. This indicates that adapting the optimization algorithm to the problem being solved improved the search effectiveness.
- *Number of SI vectors encountered during the search*: Table V gives the resultant test statistics for the number of SI vectors generated by each algorithm in the three optimization scenarios. These statistics are important since they indicate the efficiency of each algorithm, namely, the extent of computer resources wasted during its search. Results show that the EA–PS algorithm consistently obtained the best mean statistic, which indicates that it typically generated the least amount of SI vectors. This is attributed to the penalty approach it employs which deforms the metamodel landscape and consequently biases the optimization search away from the vicinity of previously encountered SI vectors. However, this setup also resulted in poor final objective values, as indicated by the test statistics in Table IV.
  In contrast, the optimization search of the proposed algorithm resulted in a higher number of SI vectors in all scenarios, which suggests that in this test problem locating good candidate solutions required exploring SI candidate solutions.
- *Variation of the classifier type and split ratio during the search*: The goal of this analysis was to study the contribution of the procedure for selecting the classifier type and split ratio, namely, if predominantly a single classifier type and split ratio were selected during the search, which would imply the procedure was redundant, or if the selected types were varied frequently.
  Figure 7 shows representative results from a test run with AOA=20°, and from another run with AOA=40°. It follows that in both runs, the classifier type and the split ratio were frequently updated during the search. In the AOA=20° case, the SVM and *k*NN classifiers were selected a similar number of times, while LDA was selected less frequently, which indicates that it was deemed as less accurate. With the split ratio, all settings were selected during the search, with the 0.8 setting being selected more frequently than the 0.5 and 0.2 settings.
  In the AOA=40° case, the SVM and *k*NN classifiers were both frequently selected. However, the LDA classifier was not selected during the run, which indicates that it was consistently deemed as the least accurate. With respect to the split ratio, and similarly to the AOA=20° case, all settings were selected during the search, with the 0.8 setting being used more frequently.
  Overall, these results, coupled with the test statistics in Table IV indicate that: a) the optimal classifier type and split ratio varied not only between different optimization scenarios, but also during the optimization search itself, and b) adapting the optimization algorithm during the search improved the search effectiveness.

V. Conclusion and future work

In modern engineering, computer simulations are often used to evaluate candidate designs. This setup yields an optimization problem of a computationally expensive black-box function, namely, whose analytic expression is unknown and which can be evaluated only a small number of times. Often, such problems will also involve candidate designs which cause the simulation to fail. Therefore, such designs would not have

TABLE IV
STATISTICS FOR THE BEST SOLUTION FOUND

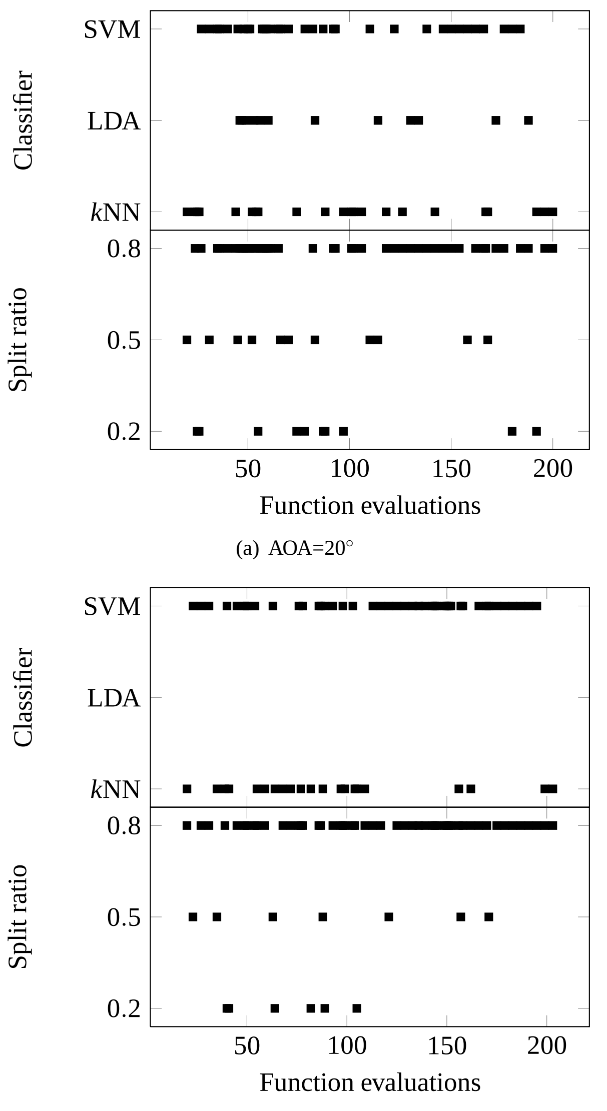
AOA		Proposed	VK	VS	EA–PS	EI–CMA-ES
20	Mean	**3.737e-01**	3.890e-01	3.885e-01	4.418e-01	5.675e-01
	SD	7.018e-03	2.845e-02	3.926e-02	5.773e-02	2.102e-01
	Median	**3.717e-01**	3.857e-01	3.718e-01	4.333e-01	5.052e-01
	Min(best)	3.649e-01	3.603e-01	3.626e-01	3.674e-01	3.584e-01
	Max(worst)	3.902e-01	4.392e-01	4.911e-01	5.686e-01	9.638e-01
	α				0.01	0.05
30	Mean	**9.273e-01**	9.475e-01	9.616e-01	9.842e-01	9.322e-01
	SD	7.314e-02	8.652e-02	4.662e-02	1.237e-01	8.289e-02
	Median	9.388e-01	9.495e-01	9.521e-01	1.026e+00	**9.180e-01**
	Min(best)	7.989e-01	7.836e-01	8.832e-01	7.113e-01	8.197e-01
	Max(worst)	1.003e+00	1.081e+00	1.022e+00	1.099e+00	1.205e+00
	α				0.05	
40	Mean	**1.023e+00**	1.027e+00	1.032e+00	1.112e+00	1.044e+00
	SD	3.888e-02	4.637e-02	4.319e-02	4.635e-02	4.347e-02
	Mmedian	**1.015e+00**	1.029e+00	1.045e+00	1.116e+00	1.034e+00
	Min(best)	9.490e-01	9.505e-01	9.675e-01	1.006e+00	9.746e-01
	Max(worst)	1.088e+00	1.111e+00	1.090e+00	1.204e+00	1.154e+00
	α				0.01	

(Algorithm spans Proposed through EI–CMA-ES.)

TABLE V
STATISTICS FOR THE NUMBER OF SI VECTORS ENCOUNTERED DURING THE SEARCH

AOA		Proposed	VK	VS	EA–PS	EI–CMA-ES
20	Mean	1.610e+01	4.800e+00	1.280e+01	**4.267e+00**	9.033e+00
	SD	3.242e+01	4.894e+00	1.561e+01	2.477e+00	1.785e+01
	Median	4.500e+00	4.000e+00	5.000e+00	4.500e+00	**2.000e+00**
	Min(best)	1.000e+00	0.000e+00	0.000e+00	1.000e+00	0.000e+00
	Max(worst)	1.070e+02	1.400e+01	4.900e+01	9.000e+00	8.100e+01
30	Mean	3.330e+01	3.970e+01	2.630e+01	**9.967e+00**	2.467e+01
	SD	1.532e+01	2.445e+01	1.778e+01	4.846e+00	1.489e+01
	Median	3.350e+01	3.700e+01	2.000e+01	**1.050e+01**	2.200e+01
	Min(best)	1.200e+01	1.400e+01	9.000e+00	1.000e+00	9.000e+00
	Max(worst)	5.500e+01	8.100e+01	6.500e+01	1.900e+01	8.700e+01
40	Mean	6.290e+01	4.790e+01	5.720e+01	**2.267e+01**	4.743e+01
	SD	2.429e+01	1.795e+01	2.296e+01	8.515e+00	1.618e+01
	Median	6.500e+01	4.700e+01	5.600e+01	**2.100e+01**	4.850e+01
	Min(best)	2.100e+01	2.200e+01	1.700e+01	1.300e+01	2.100e+01
	Max(worst)	9.600e+01	7.800e+01	9.900e+01	4.300e+01	8.400e+01

(Algorithm spans Proposed through EI–CMA-ES.)



(a) AOA=20°

(b) AOA=40°

Fig. 7. The classifier type and split ratio selected by the proposed algorithm during two test runs.

an objective value assigned to them, and consequently they can degrade the effectiveness of the optimization search.

Existing approaches for handling such candidate designs include assigning them a penalized objective value or discarding them altogether, but both of these approaches have significant demerits, as discussed in Section II. In these settings, this study has proposed a new computational intelligence optimization algorithm which incorporates a classifier into the optimization search. The latter predicts which candidate designs are expected to cause the simulation to fail, and this prediction is used to bias the search towards candidate designs for which the simulation is expected to succeed. However, the effectiveness of this setup depends on the type of classifier being used, but typically it is not known prior to the optimization search which classifier type is most suitable to the problem being solved. To address this, the proposed algorithm autonomously selects during the search an optimal classifier type out of a family of candidates, based on the CV procedure. To further enhance the accuracy of this approach, it also selects during the search the split ratio of the CV procedure.

The effectiveness of the proposed algorithm was evaluated with a simulation-driven test problem of airfoil shape optimization which is representative of real-world problems. Analysis of the experiments results shows that:

- Incorporating a classifier into the optimization search was an effective approach to handle SI vectors, as indicated by the test statistics of the final function value.
- Penalizing SI vectors and incorporating them into the metamodel training sample reduced the number of failed evaluations, but also yielded a poorer final result. In contrast, the proposed algorithm typically evaluated a larger number of SI vectors, which indicates that obtaining a good SF solution may require exploring a multitude of SI ones.
- The optimal classifier type and split ratio varied not

only between optimization scenarios, but also during the optimization search itself. Also, adapting the optimization algorithm during the search improved the search effectiveness, as indicated by the comparisons to the two variants of the proposed algorithm which did not employ this procedure.

Overall, the proposed algorithm effectively performed an optimization of a computationally expensive black-box function in the presence of SI candidate designs. Prospective future work includes improving the algorithm's effectiveness in problems with a high prevalence of SI vectors, and dynamic optimization problems, namely, which vary with time.

Appendix A
Candidate classifiers

Classifiers originated in the domain on machine learning with the goal of class prediction. Mathematically, given a set of vectors $\boldsymbol{x}_i \in \mathbb{R}^d$, $i = 1 \ldots n$, which are grouped into several classes such that each vector has a corresponding class label $F(\boldsymbol{x}_i) \in \mathbb{I}$, for example, $\mathbb{I} = \{-1, +1\}$, a classifier performs the mapping

$$c(\boldsymbol{x}) : \mathbb{R}^d \to \mathbb{I} , \tag{17}$$

where $c(\boldsymbol{x})$ is the class assigned by the classifier.

In this study, the proposed algorithm selects from three established classifier variants [39]:

- *k Nearest Neighbours* (KNN): The classifier assigns the new vector the class of its closest training vector, namely:

$$c(\boldsymbol{x}) = F(\boldsymbol{x}_{\text{NN}}) : d(\boldsymbol{x}, \boldsymbol{x}_{\text{NN}}) = \min d\big(\boldsymbol{x}, \boldsymbol{x}_i\big) , \ i = 1 \ldots n , \tag{18}$$

  where $d(\boldsymbol{x}, \boldsymbol{y})$ is a distance measure such as the l_2 norm. An extension of this technique is to assign the class most frequent among the $k > 1$ nearest neighbours (kNN). In this study the classifier used $k = 3$.
- *Linear Discriminant Analysis* (LDA): In a two-class problem, where the class labels are $F(\boldsymbol{x}_i) \in \mathbb{I} = \{-1, +1\}$, the classifier attempts to model the conditional probability density functions of a vector belonging to each class, where the latter functions are assumed to be normally distributed. The classifier considers the separation between classes as the ratio of: a) the variance between classes, and b) the variance within the classes, and obtains a vector $\boldsymbol{w}$ which maximizes this ratio. The vector $\boldsymbol{w}$ is such that it is orthogonal to the hyperplane separating the two classes. A new vector $\boldsymbol{x}$ is classified based on its projection with respect to the separating hyperplane, that is,

$$c(\boldsymbol{x}) = \text{sign}\,(\boldsymbol{w} \cdot \boldsymbol{x}) . \tag{19}$$

- *Support Vector Machines* (SVM): The classifier projects the data into a high-dimensional space where it can be more easily separated into disjoint classes. In a two-class problem, and assuming class labels $F(\boldsymbol{x}_i) \in \mathbb{I} = \{-1, +1\}$, an SVM classifier tries to find the best classification function for the training data. For a linearly separable training set, a linear classification function is the separating hyperplane passing through the middle of the two classes. Once this hyperplane has been fixed, new vectors are classified based on their relative position to this hyperplane, that is, whether they are "above" or "below" it. Since there are many possible separating hyperplanes, an SVM classifier adds the condition that the hyperplane should maximize its distance to the nearest vectors from each class. This is accomplished by maximizing the Lagrangian

$$L_P = \frac{1}{2}\|\boldsymbol{w}\| - \sum_{i=1}^{n} \alpha_i F(\boldsymbol{x}_i)(\boldsymbol{w} \cdot \boldsymbol{x}_i + b) + \sum_{i=1}^{n} \alpha_i , \tag{20}$$

  where n is the number of samples (training vectors), $F(\boldsymbol{x}_i)$ is the class of the ith training vector, and $\alpha_i \geqslant 0$, $i = 1 \ldots n$, are the Lagrange multipliers, such that the derivatives of L_P with respect to α_i are zero. The vector $\boldsymbol{w}$ and scalar b define the hyperplane.

References

[1] Y. Tenne, K. Izui, and S. Nishiwaki, "A computational intelligence algorithm for simulation-driven optimization problems," in *Proceedings of the Third International Conference on Future Computational Technologies and Applications, Future Computing 2011*, Rome, Italy, International Academy, Research, and Industry Association (IARIA). IARIA XPS Press, 2011, pp. 127–134.

[2] Y. Tenne and C. K. Goh, Eds., *Computational Intelligence in Expensive Optimization Problems*, ser. Evolutionary Learning and Optimization. Springer, 2010, vol. 2.

[3] D. Büche, N. N. Schraudolph, and P. Koumoutsakos, "Accelerating evolutionary algorithms with Gaussian process fitness function models," *IEEE Transactions on Systems, Man, and Cybernetics–Part C*, vol. 35, no. 2, pp. 183–194, 2005.

[4] T. Okabe, "Stabilizing parallel computation for evolutionary algorithms on real-world applications," in *Proceedings of the 7th International Conference on Optimization Techniques and Applications–ICOTA 7*. Tokyo: Universal Academy Press, 2007, pp. 131–132.

[5] C. Poloni, A. Giurgevich, L. Onseti, and V. Pediroda, "Hybridization of a multi-objective genetic algorithm, a neural network and a classical optimizer for a complex design problem in fluid dynamics," *Computer Methods in Applied Mechanics and Engineering*, vol. 186, no. 2–4, pp. 403–420, 2000.

[6] K. A. de Jong, *Evolutionary Computation:A Unified Approach*. MIT Press, Cambridge, Mass. 2006.

[7] T. W. Simpson, J. D. Poplinski, P. N. Koch, and J. K. Allen, "Metamodels for computer-based engineering design: Survey and recommendations," *Engineering with Computers*, vol. 17, pp. 129–150, 2001.

[8] A. Ratle, "Accelerating the convergence of evolutionary algorithms by fitness landscape approximations," in *Proceedings of the 5th International Conference on Parallel Problem Solving from Nature–PPSN V*, A. E. Eiben,

T. Bäck, and H.-P. Schwefel, Eds. Berlin: Springer, 1998, pp. 87–96.

[9] A. J. Booker, J. E. Dennis, P. D. Frank, D. B. Serafini, V. Torczon, and M. W. Trosset, "A rigorous framework for optimization of expensive functions by surrogates," *Structural Optimization*, vol. 17, no. 1, pp. 1–13, 1999.

[10] M. T. M. Emmerich, A. Giotis, M. Özedmir, T. Bäck, and K. C. Giannakoglou, "Metamodel-assisted evolution strategies," in *The 7th International Conference on Parallel Problem Solving from Nature–PPSN VII*, ser. Lecture Notes in Computer Science, J. J. Merelo Guervós, Ed., no. 2439. Berlin: Springer, 2002, pp. 361–370.

[11] K.-H. Liang, X. Yao, and C. Newton, "Evolutionary search of approximated N-dimensional landscapes," *International Journal of Knowledge-Based Intelligent Engineering Systems*, vol. 4, no. 3, pp. 172–183, 2000.

[12] F. Muyl, L. Dumas, and V. Herbert, "Hybrid method for aerodynamic shape optimization in automotive industry," *Computers and Fluids*, vol. 33, no. 5–6, pp. 849–858, 2004.

[13] Y. Tenne and S. W. Armfield, "A framework for memetic optimization using variable global and local surrogate models," *Journal of Soft Computing*, vol. 13, no. 8, pp. 781–793, 2009.

[14] F. Neri, X. del Toro Garcia, G. L. Cascella, and N. Salvatore, "Surrogate assisted local search on PMSM drive design," *International Journal for Computation and Mathematics in Electrical and Electronic Engineering*, vol. 27, no. 3, pp. 573–592, 2008.

[15] Z. Zhou, Y.-S. Ong, P. B. Nair, A. J. Keane, and K. Y. Lum, "Combining global and local surrogate models to accelerate evolutionary optimization," *IEEE Transactions on Systems, Man, and Cybernetics–Part C*, vol. 37, no. 1, pp. 66–76, 2007.

[16] Y. Jin, M. Olhofer, and B. Sendhoff, "A framework for evolutionary optimization with approximate fitness functions," *IEEE Transactions on evolutionary computation*, vol. 6, no. 5, pp. 481–494, 2002.

[17] A. R. Conn, K. Scheinberg, and P. L. Toint, "On the convergence of derivative-free methods for unconstrained optimization," in *Approximation Theory and Optimization: Tributes to M.J.D. Powell*, A. Iserles and M. D. Buhmann, Eds. Cambridge; New York: Cambridge University Press, 1997, pp. 83–108.

[18] ——, "A derivative free optimization algorithm in practice," in *Proceedings of the Seventh AIAA/USAF/NASA/ISSMO Symposium on Multidisciplinary Analysis and Optimization.* Reston, Virginia: American Institute of Aeronautics and Astronautics, 1998, AIAA paper number AIAA-1998-4718.

[19] K. Rasheed, H. Hirsh, and A. Gelsey, "A genetic algorithm for continuous design space search," *Artificial Intelligence in Engineering*, vol. 11, pp. 295–305, 1997.

[20] Y. Tenne and S. W. Armfield, "A versatile surrogate-assisted memetic algorithm for optimization of computationally expensive functions and its engineering applications," in *Success in Evolutionary Computation*, ser. Studies in Computational Intelligence, A. Yang, Y. Shan, and L. Thu Bui, Eds. Berlin; Heidelberg: Springer-Verlag, 2008, vol. 92, pp. 43–72.

[21] S. Handoko, C. K. Kwoh, and Y.-S. Ong, "Feasibility structure modeling: An effective chaperon for constrained memetic algorithms," *IEEE Transactions on Evolutionary Computation*, vol. 14, no. 5, pp. 740–758, 2010.

[22] Y. Tenne, K. Izui, and S. Nishiwaki, "Handling undefined vectors in expensive optimization problems," in *Proceedings of the 2010 EvoStar Conference*, ser. Lecture Notes in Computer Science, C. Di Chio, Ed., vol. 6024/2010. Berlin: Springer, 2010, pp. 582–591.

[23] K. P. Burnham and D. R. Anderson, *Model Selection and Inference: A Practical Information-theoretic Approach.* New York: Springer, 2002.

[24] A. Frank and A. Asuncion, "UCI Machine Learning Repository," 2010. [Online]. Available: http://archive.ics.uci.edu/ml

[25] M. D. McKay, R. J. Beckman, and W. J. Conover, "A comparison of three methods for selecting values of input variables in the analysis of output from a computer code," *Technometrics*, vol. 21, no. 2, pp. 239–245, 1979.

[26] K. S. Won and T. Ray, "Performance of Kriging and Cokriging based surrogate models within the unified framework for surrogate assisted optimization," in *The 2004 IEEE Congress on Evolutionary Computation–CEC 2004.* Piscataway, NJ: IEEE, 2004, pp. 1577–1585.

[27] H. You, M. Yang, D. Wang, and X. Jia, "Kriging model combined with Latin hypercube sampling for surrogate modeling of analog integrated circuit performance," in *Proceedings of the Tenth International Symposium on Quality Electronic Design–ISQED 2009.* Piscataway, NJ: IEEE, 2009, pp. 554–558.

[28] J. R. Koehler and A. B. Owen, "Computer experiments," in *Handbook of Statistics*, S. Ghosh, C. R. Rao, and P. R. Krishnaiah, Eds. Amsterdam: Elsevier, 1996, pp. 261–308.

[29] A. Chipperfield, P. Fleming, H. Pohlheim, and C. Fonseca, *Genetic Algorithm TOOLBOX For Use with MATLAB, Version 1.2*, Department of Automatic Control and Systems Engineering, University of Sheffield, Sheffield, 1994.

[30] K. A. de Jong and W. M. Spears, "An analysis of the interacting roles of population size and crossover in genetic algorithms," in *Proceedings of the 1st Workshop on Parallel Problem Solving from Nature–PPSN I*, H.-P. Schwefel and R. Männer, Eds. Berlin: Springer, 1990, pp. 38–47.

[31] W. R. Madych, "Miscellaneous error bounds for multiquadric and related interpolators," *Computers and Mathematics with Applications*, vol. 24, no. 12, pp. 121–138, 1992.

[32] R. M. Hicks and P. A. Henne, "Wing design by numerical optimization," *Journal of Aircraft*, vol. 15, no. 7, pp. 407–

412, 1978.

[33] H.-Y. Wu, S. Yang, F. Liu, and H.-M. Tsai, "Comparison of three geometric representations of airfoils for aerodynamic optimization," in *Proceedings of the 16th AIAA Computational Fluid Dynamics Conference*. American Institute of Aeronautics and Astronautics, 2003, AIAA 2003-4095.

[34] M. Drela and H. Youngren, *XFOIL 6.9 User Primer*, Department of Aeronautics and Astronautics, Massachusetts Institute of Technology, Cambridge, MA, 2001.

[35] A. Ratle, "Optimal sampling strategies for learning a fitness model," in *The 1999 IEEE Congress on Evolutionary Computation–CEC 1999*. Piscataway, New Jersey: IEEE, 1999, pp. 2078–2085.

[36] N. Hansen and A. Ostermeier, "Completely derandomized self-adaptation in evolution strategies," *Evolutionary Computation*, vol. 9, no. 2, pp. 159–195, 2001.

[37] D. R. Jones, M. Schonlau, and W. J. Welch, "Efficient global optimization of expensive black-box functions," *Journal of Global Optimization*, vol. 13, pp. 455–492, 1998.

[38] D. J. Sheskin, *Handbook of Parametric and Nonparametric Statistical Procedures*, 4th ed. Boca Raton, Florida: Chapman and Hall, 2007.

[39] X. Wu, V. Kumar, R. J. Quinlan, J. Ghosh, Q. Yang, H. Motoda, G. J. McLachlan, A. Ng, B. Liu, P. S. Yu, Z.-H. Zhou, M. Steinbach, D. J. Hand, and D. Steinberg, "Top 10 algorithms in data mining," *Knowledge and Information Systems*, vol. 14, pp. 1–37, 2008.