A Computational Intelligence Algorithm for Simulation-driven Optimization Problems

Yoel Tenne
Faculty of Engineering,
Kyoto University, Japan
yoel.tenne@ky3.ecs.kyoto-u.ac.jp

Kazuhiro Izui
Faculty of Engineering,
Kyoto University, Japan
izui@prec.kyoto-u.ac.jp

Shinji Nishiwaki
Faculty of Engineering,
Kyoto University, Japan,
shinji@prec.kyoto-u.ac.jp

***Abstract*—Modern engineering design optimization often replaces laboratory experiments with computer simulations, resulting in what is commonly termed as *expensive black-box optimization problems*. In such problems, there will often exist candidate solutions which 'crash' the simulation and can thus lead to search stagnation and to a poor final result. Existing approaches to handle such solutions include discarding them altogether or assigning them a penalized fitness, but such approaches have significant demerits. Accordingly, this paper explores the fusion of a classifier into the optimization search to predict which solutions are expected to crash the simulation, and uses a modified objective function to bias the search towards valid ones, namely, which are expected *not* to crash the simulator. The further improve its performance, the proposed algorithm also continuously selects during the search an optimal type of classifier out of a family of candidates. To ensure the progress of the optimization search, it also employs a trust-region approach. Performance analysis using a representative real-world shape design optimization test case shows the efficacy of the proposed algorithm.**

***Keywords*-expensive optimization problems, computational intelligence**

I. Introduction

In the modern design process, engineers often replace laboratory experiments with *computer simulations*. This transforms the design process into an optimization problem having three distinct characteristics [20]:

- The simulation acts as the objective function, assigning candidate designs their corresponding objective values. However, the simulation is often a legacy or a commercial code available only as an executable, and so there is no analytic expression for this "objective function". Accordingly, it is referred to as a *black-box function*, and requires using gradient-free optimizers.
- Each simulation run is *computationally expensive*, that is, having a lengthy execution time, and this severely restricts the number of evaluations allowed during the optimization search.

and

- Often, both the underlying physics being modelled, and the numerical simulation, may result in a complicated, multimodal objective landscape, which makes it difficult to locate an optimum.

A promising optimization strategy for such expensive black-box problems is to couple a computational intelligence (CI) optimizer, which is gradient-free and handles complicated landscapes well, with *models*, namely, mathematical approximations of the true expensive objective function, but which are significantly cheaper to evaluate. During the optimization, the model replaces the expensive function (simulation), and economically provides the CI optimizer with approximate objective values.

While this approach works well, in practise another difficulty arises, as often some candidate solutions will cause the simulation to fail. We refer to such vectors as *simulator-infeasible* (SI), while those for which the simulation completes successfully and provides the objective value are *simulator-feasible* (SF). SI vectors have two main implications for the optimization problem: a) as they do not have an objective value assigned to them, since the simulation has crashed, the objective function becomes discontinuous, which increases the optimization difficulty, and b) they can consume a large portion of the limited number of calls to the expensive simulation without providing new information to the optimizer, thus potentially leading to search stagnation and a poor final result. Numerous papers, for example [1, 15, 16], have mentioned the difficulties SI vectors introduce, and so it is important effectively to handle them. Common strategies include discarding them altogether, or incorporating them into the model with a penalized fitness. 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 model with a severely deformed landscape.

Accordingly, to effectively handle such SI vectors in model-assisted optimization, this paper explores the fusion of a classifier into the optimization search, and offers two main contributions. First, a classifier is used to predicts if a new vector is SI or not, and the proposed algorithm then combines this prediction with the model's prediction to bias the search to vectors predicted to be SF via a dedicated modified objective function. Second, to further enhance performance, the proposed algorithm continuously selects during the search an optimal classifier type from a family of candidates. To ensure convergence to an optimum of the true expensive function, the proposed algorithm also employs a trust-region (TR) approach. Performance analysis using an engineering design application of airfoil shape optimization shows the efficacy of the proposed algorithm. The remainder of this paper is as follows: Section II describes existing approaches and open challenges in handling SI vectors, Section III describes the proposed algorithm and Section IV gives a detailed performance analysis. Lastly, Section V summarizes this paper.

II. Existing Approaches and Challenges

As mentioned in Section I, SI vectors are a common in simulation-driven optimization problems, and numerous studies mention their existence and the difficulties they introduce, for example [1, 5, 12, 15, 16].

As such vectors are both common in real-world applications, and pose the risk of hampering the optimization search, several approaches have been proposed to handle them. In reference [17], the authors used an evolutionary algorithm (EA) as the optimizer and proposed using a classifier to screen vectors before evaluating them. Those predicted to be SI were assigned a 'death penalty', that is, a fictitious and highly penalized objective value, to quickly eliminate them from the population. The study did not consider models, and the EA evaluated the expensive function directly. In a related approach described in reference [5], severely penalized SI] vectors and incorporated them into the model in order to bias the search away from them. Alternatively, in reference [1] the authors proposed to completely discard SI vectors from the training set of the model.

However, within the domain of model-assisted optimization such approaches suffer from several shortcomings: a) assigning SI vectors a fictitious penalized objective value and then incorporating them into the training set can result in a model with a severely deformed landscape, while b) excluding SI vectors altogether discards information which may be beneficial to the optimization search. As an example, Figure 1 compares two Kriging models of the Rosenbrock function: (a) shows a model trained using 30 vectors which were all SF, while (b) shows the resultant model when the sample was augmented with 20 SI vectors which were assigned a penalized fitness, taken as the worst objective value from the baseline sample. The resultant model has a landscape which is severely deformed and contains many false optima, making it difficult to locate an optimum of the true objective function.

Such issues have motivated alternative approaches to handle SI vectors in model-assisted optimization. For example, in reference [19] the authors proposed a dual model approach, where one model interpolated the objective function and the other interpolated the penalty value between SF and SI vectors. Other studies have explored the use of classifiers for constrained non-linear programming (not focusing on SI vectors), for example reference [6]. Further exploring the use of classifiers, reference [21] presented preliminary results with a classifier-assisted algorithm for handling SI vectors.

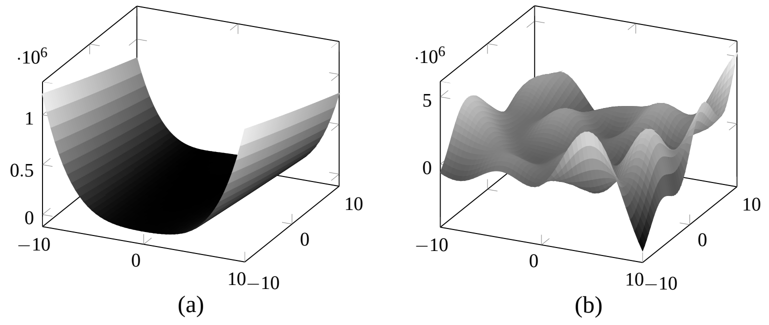


Fig. 1. Kriging models of the Rosenbrock function trained using: (a) a baseline sample of 30 vectors all SF, and (b) the baseline sample augmented with 20 SI vectors which were assigned the worst objective value from the baseline sample.

III. Proposed Algorithm

To address the issues introduced by SI vectors, as discussed in Section I and Section II, this paper proposes a model-assisted CI algorithm which incorporates a classifier into the search. To further improve the search efficacy, the algorithm continuously selects during the search an optimal type of classifier, out of a prescribed family of candidates. To ensure convergence to an optimum of the true expensive function, the algorithm also employs a TR approach. The following sections describe the model, the candidate classifiers, the classifier selection stage, and lastly, the overall workflow of the algorithm.

A. Modelling

As mentioned, the proposed algorithm uses a model to approximate the true expensive objective function. The algorithm does not impose any restrictions on the model type, and in this study the well-established Kriging model was used [11]. This model takes a statistical approach to interpolation by combining two components: a 'drift' function, which is a global coarse approximation to the true expensive function, and a local correction based on the correlation between the interpolation points. Given a set of evaluated vectors, $\mathbf{x}_i \in \mathbb{R}^d$, $i = 1 \ldots n$, the Kriging model is trained such that it exactly interpolates the observed values, that is, $m(\mathbf{x}_i) = f(\mathbf{x}_i)$, where $m(\mathbf{x})$ and $f(\mathbf{x})$ are the model and true objective function, respectively. Using a constant drift function, as in reference [11], gives the Kriging model

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

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

$$Cov[\kappa(\mathbf{x})\kappa(\mathbf{y})] = \sigma^2 \mathscr{R}(\theta, \mathbf{x}, \mathbf{y}), \tag{2}$$

where $\mathscr{R}$ is a user-prescribed correlation function. A common choice for the correlation is the Gaussian function [11], which is defined as

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

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

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

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

$$\mathbf{r}^{\mathrm{T}} = [\mathscr{R}(\theta,\mathbf{x},\mathbf{x}_1),\ldots,\mathscr{R}(\theta,\mathbf{x},\mathbf{x}_n)] \,. \tag{5}$$

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

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

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

Fully defining the model requires the correlation parameter θ, which is commonly taken as the maximizer of the model likelihood. In practise, maximizing the model likelihood is performed by minimizing the negative logarithm of the likelihood, which is given by

$$\mathscr{L} = -\left(n\log(\hat{\sigma}^2) + \log(|\mathbf{R}|)\right) \,. \tag{7}$$

B. Candidate Classifiers

Given a set of input vectors with corresponding *labels*, that is, indices assigning them to a group, the goal of a classifier is to map a new input vector into one of the existing groups based on some similarity between the new and existing vectors [23]. Mathematically, given a set of vectors $\mathbf{x}_i \in \mathbb{R}^d$, $i = 1\ldots n$, with corresponding class labels, for example, $F(\mathbf{x}_i) \in \mathbb{I} = \{-1,1\}$, a classifier $c(\mathbf{x})$ is the mapping

$$c(\mathbf{x}) : \mathbb{R}^d \to \mathbb{I}. \tag{8}$$

The proposed algorithm puts no restriction on the number or the type of candidates classifiers, and in this study we used three well-established classifier variants [23]:

- *nearest neighbour* (NN): The classifier assigns the new vector the class of the nearest training vector, designated $\mathbf{x}_{\mathrm{NN}}$, where the latter is determined by a distance measure such as the l_2 norm, namely,

$$\begin{gathered} c(\mathbf{x}) = F(\mathbf{x}_{\mathrm{NN}}) : \\ d(\mathbf{x},\mathbf{x}_{\mathrm{NN}}) = \min_{i=1\ldots n} d\big(\mathbf{x},\mathbf{x}_i\big) \,. \end{gathered} \tag{9}$$

  An extension of the approach, termed k nearest neighbours (k-NN) assigns the class most frequent among the k nearest neighbours. In this study the classifier used $k = 3$.
- *linear discriminant analysis* (LDA): In a two-class problem, where the class labels are $F(\mathbf{x}_i) \in \mathbb{I} = \{-1,+1\}$, 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 $\mathbf{w}$ which maximizes this ratio. The vector $\mathbf{w}$ is orthogonal to the hyperplane separating the two classes. A new vector, $\mathbf{x}$, is classified based on its projection with respect to the separating hyperplane, that is,

$$c(\mathbf{x}) = \operatorname{sign}(\mathbf{w}\cdot\mathbf{x}) \,. \tag{10}$$

- support vector machine (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, with $F(\mathbf{x}_i) \in \mathbb{I} = \{-1,+1\}$, 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 the distance between the hyperplane and the nearest vectors to it from each class. This is accomplished by maximizing the Lagrangian

$$L_P = \frac{1}{2}\|\mathbf{w}\| - \sum_{i=1}^{n}\alpha_i F(\mathbf{x}_i)(\mathbf{w}\cdot\mathbf{x}_i + b) + \sum_{i=1}^{n}\alpha_i \,, \tag{11}$$

  where n is the number of samples (training vectors), $F(\mathbf{x}_i)$ 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 $\mathbf{w}$ and scalar b define the hyperplane.

C. Classifier Selection

As mentioned, during the optimization search the proposed algorithm continuously selects an optimal classifier out of a family of candidates. To accomplish this in a mathematically rigorous way, the proposed algorithm leverages on statistical model-selection theory and selects a classifier using an *accuracy estimator*. Specifically, the accuracy of each candidate classifier is estimated, and the classifier chosen is the one having the best estimated accuracy. The procedure, termed cross-validation (CV), proceeds as follows. A cache, which contains vectors evaluated with the expensive function, is split into a *training set* and a *testing set*, in a 80–20 ratio. A candidate classifier is trained using the former set and its prediction is tested on the latter set, where the classification error is

$$e = \sum_{i=1}^{l}\big(\hat{c}(\mathbf{x}_i) \neq F(\mathbf{x}_i)\big) \,, \tag{12}$$

where $\hat{c}$ is the prediction of the trained classifier, $\mathbf{x}_i$, $i = 1\ldots l$, are the CV testing vectors, and $F(\mathbf{x}_i)$ is the true and known class of the latter vectors, where in this study $F(\mathbf{x}_i) = 1$ was used for a SF vector, and $F(\mathbf{x}_i) = -1$ for a SI one. These class labels were used as they are common in literature, for example

[23]. As mentioned in Section III-B, in this study, the proposed algorithm selects between three well-established classifiers, namely k-NN, LDA, and SVM. The algorithm selects the classifier type having the smallest prediction error, as defined in Equation (12), and then uses all the cached vectors, namely, both SF and SI, to train a new classifier with the selected type, which serves as the classifier during the TR trial step, as explained in the following section.

D. Workflow

The algorithm begins by sampling a set of vectors which will serve as the initial training sample. The vectors are generated using the Latin hypercube design (LHD) method for experiments design [14], as it provides a space-filling sample which improves the accuracy of the model. Briefly, for a sample of k vectors the range of each variable is split into k equal intervals, and one point is sampled at random in each interval. Next, a sample point is selected at random (without replacement) for each variable, and these samples are combined to give a vector. This procedure is repeated for k times to generate the complete sample. After generating the sample, the vectors are evaluated with the expensive function and are then cached.

The main optimization loop then begins, where the algorithm first trains a Kriging model using all the SF in the cache. It then uses the procedure described in Section III-C to select a classifier type, and then trains a classifier using all the cached vectors, namely, both SF and SI, as these are two vector classes.

Next, the proposed algorithm performs an optimization search, and to ensure convergence to an optimum of the true expensive function it follows the trust-region (TR) approach. Specifically, the TR is the region where the model is assumed to be accurate, and defined as

$$\mathscr{T} = \{\mathbf{x} : \|\mathbf{x} - \mathbf{x}_\mathrm{b}\|_2 \leqslant \Delta\}\,, \qquad (13)$$

where $\mathbf{x}_\mathrm{b}$ is the best vector found so far, and is taken as the TR centre, and Δ is the TR radius. The proposed algorithm seeks the optimum of the model in the TR, and as the optimizer it uses the real-coded EA from reference [2]. Since evaluating the model is computationally cheap, the EA uses a large population and many generations to improve its search, and Table I gives the complete EA parameter settings. During this optimization trial-step the EA does not use the model predictions directly, but instead it obtains the fitness values from the following *modified objective function*, defined as

$$\overline{m}(\mathbf{x}) = \begin{cases} m(\mathbf{x}) & \text{if } c(\mathbf{x}) \text{ is SF} \\ p & \text{if } c(\mathbf{x}) \text{ is SI} \end{cases} \qquad (14)$$

where $m(\mathbf{x})$ is the model prediction, and p is a penalized fitness taken to be the worst function value from the initial Latin hypercube (LH) sample. As such, the EA receives the model prediction if the classifier predicts a vector is SF, but receives the penalized fitness otherwise. In this setup, the information about SI vectors is preserved in the classifier, but they are *not* incorporated into the model with a penalized fitness and hence do not deform its landscape.

The optimum found by the EA, designated as $\mathbf{x}^\star$, is then evaluated with the true expensive function at a cost of one function evaluation, which provides $f(\mathbf{x}^\star)$. Following the classical TR framework [3], the algorithm manages the model and TR based on the outcome of the trial step, as follows:

- *A successful trial step*: The trial step located a new optimum which is better than the current best, that is, $f(\mathbf{x}^\star) < f(\mathbf{x}_\mathrm{b})$. Following the classical TR approach, the new optimum is taken as the new the TR centre, and the TR is enlarged by doubling its radius.
- *An unsuccessful trial step*: The trial step did not locate a new optimum, that is, $f(\mathbf{x}^\star) \geqslant f(\mathbf{x}_\mathrm{b})$. This can happen since either the TR is too large, or since there are not enough vectors in the TR, resulting in a model which is too inaccurate. Accordingly, to gauge the accuracy of the model, the proposed algorithm checks the number of vectors inside the TR. This number is then compared to the dimension of the objective function (d), as it is an indicator to the number of function evaluations required to estimate the gradient by finite-differences and hence is indicative of the number of function evaluation needed to find a new optimum. To manage the accuracy of the model, the following steps are performed:
  - If there are less than d SF vectors inside the TR: The unsuccessful step may be since the model or classifier are inaccurate in the TR. As such, the algorithm adds a new point ($\mathbf{x}_\mathrm{n}$) inside the TR to improve their local accuracy. The procedure for adding the point is explained below.
  - If there are more than d SF vectors in the TR: The model and classifier are considered to be sufficiently accurate in the TR. In this case, and following the classical TR framework, the TR is contracted by halving its radius.

Compared to the classical TR framework, the above steps also monitor the number of interior vectors in the TR, since they determine the local model accuracy. Monitoring the number of these vectors ensures the TR is not contracted too quickly when the search stagnates due to poor accuracy of the model or the classifier [3].

Another change from the classical framework is the addition of a new vector ($\mathbf{x}_\mathrm{n}$) to improve the local model accuracy. To accomplish this, the new vector should be far from existing ones, so it improves the model in an region sparse with

TABLE I
EA PARAMETERS

population size	100
generations	100
selection	stochastic universal selection (SUS), $p = 0.7$
recombination	intermediate, $p = 0.7$
mutation[1]	Breeder Genetic Algorithm (BGA) mutation, $p = 0.1$
elitism	10%

[1] Based on [2]

sampled points [13]. Mathematically, finding such a point translates to the following max-min optimization problem,

$$\mathbf{x}_\mathrm{n} : \max_{\mathbf{x}\in\mathscr{T}} \min_{\mathbf{x}_i\in\mathscr{T}} \{\|\mathbf{x}-\mathbf{x}_i\|_2\} \tag{15}$$

where $\mathbf{x}_i$, $i = 1\ldots r$, are the existing interior TR points [9]. To simplify the solution of Equation (15), the proposed algorithm generates a LH sample in the TR and chooses the sample point with the largest minimum distance.

Lastly, if the TR has been contracted for q consecutive iterations, which suggests convergence to a local optimum, the algorithm adds a point outside the TR to improve the accuracy of model globally, which assists in locating new optima. The point is generated using the same procedure described above for the new interior point, namely, $\mathbf{x}_\mathrm{n}$, but now considering the entire search space instead of just the TR. Based on numerical experiments, we identified $q = 2$ as a suitable setting. To complete the description, Algorithm 1 gives the pseudocode of the proposed algorithm.

Algorithm 1: Proposed Optimization Algorithm with Adaptive Model and Classifier

```
generate an initial LHD sample;
evaluate and cache the sample vectors;
repeat
    train a new Kriging model using all SF vectors in the cache;
    /* classifier selection */
    for candidate classifier = {k-NN, LDA, SVM} do
        use CV to find the classification error (12);
    select the classifier with the lowest error and train a new classifier using all the vectors in the cache;
    /* TR trial step */
    set the best vector in the cache as the TR centre;
    search for the model optimum using an EA and the modified objective function (14);
    evaluate the predicted optimum with the expensive objective function;
    /* manage the model and TR */
    if the new optimum is better than the TR centre then
        increase the TR radius
    else if the new optimum is not better than the TR centre and there are insufficient vectors in the TR then
        add a new vector in the TR to improve the model and classifier;
    else if the new optimum is not better than the TR centre and there are sufficient vectors in the TR then
        decrease the TR radius;
    /* check search stagnation */
    if there have been q consecutive TR contractions then
        add a new vector outside the TR to improve the accuracy of the model globally;
    cache all new vectors evaluated;
until optimization budget exhausted;
```

IV. Performance Analysis

For its evaluation, the proposed algorithm was applied to an engineering application of airfoil shape optimization. The problem is pertinent to this study as it is both representative of real-world expensive black-box optimization problems, *and* contains SI vectors, as explained below.

The setup of the problem is as follows. During flight an aircraft generates *lift*, namely, the beneficial aerodynamic force which keeps it airborne, and also *drag*, that is, an aerodynamic friction force which obstructs the aircraft's movement. Accordingly, the optimization goal is to find an airfoil shape which maximizes the ratio of the lift to drag at some prescribed flight conditions, namely, the flight altitude, the flight speed, and the angle of attack (AOA) which is the angle between the airfoil chord and the aircraft velocity. Figure 2(a) shows the physical quantities involved.

To ensure structural integrity, the minimum airfoil thickness (t) between 0.2 to 0.8 of the airfoil chord must be equal to or larger than a critical value $t^\star = 0.1$. Also, in practise the design requirements for airfoils are specified in terms of the non-dimensional lift and drag coefficients, c_l and c_d, respectively, defined as

$$c_l = \frac{L}{\frac{1}{2}\rho V^2 S} \tag{16a}$$

$$c_d = \frac{D}{\frac{1}{2}\rho V^2 S} \tag{16b}$$

where L and D are the lift and drag forces, respectively, ρ is the air density, V is aircraft speed, and S is a reference area, such as the wing area. Accordingly, maximizing the lift and minimizing the drag is formulated as a minimization problem using the following objective function

$$f = -\frac{c_l}{c_d} + p\,, \tag{17a}$$

where c_l and c_d were defined above, and p is a penalty for airfoils which violate the thickness constraint, and is defined as

$$p = \begin{cases} \frac{t^\star}{t}\cdot\left|\frac{c_l}{c_d}\right| & \text{if } t < t^\star \\ 0 & \text{otherwise} \end{cases} \tag{17b}$$

Airfoils were represented with the Hicks-Henne parameterization [8], which uses a baseline airfoil and adds the basis functions

$$b_i(x) = \left[\sin\left(\pi x^{\frac{\log(0.5)}{\log(i/(h+1))}}\right)\right]^4 , \tag{18}$$

with $i = 1\ldots h$, where h is user-prescribed, to smoothly modify the baseline shape [22]. The lower and upper curves of a candidate airfoil are then given by

$$y = y_b + \sum_{i=1}^{h} \alpha_i b_i(x)\,, \tag{19}$$

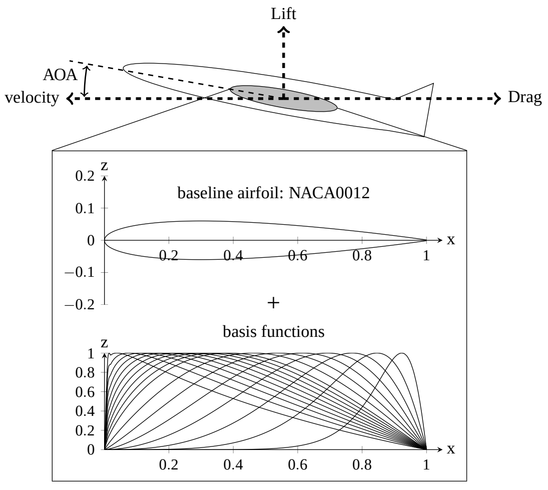
where y_b is the baseline upper/lower curve, which was taken as the NACA0012 symmetric airfoil, and $\alpha_i \in [-0.01, 0.01]$ are the coefficients (design variables) to be found. In this study, we used $h = 10$ functions for the upper and lower curve, respectively, or a total of 20 coefficients, namely, design variables, per airfoil. To obtain the lift and drag of candidate airfoils, we used XFoil–a computational fluid dynamics simulation for analysis of subsonic airfoils [4]. Each airfoil evaluation required up to 30 seconds on a desktop computer. Figure 2(a) gives the layout of the Hicks-Henne parametrization.

As mentioned above, the airfoil optimization problem is a pertinent test case since it contains SI vectors. The prevalence of these vectors depends on two major factors: the AOA *and* the operating conditions, namely, the altitude and velocity. To illustrate the effect of the AOA, 30 different airfoils were evaluated at identical flight conditions, except for the AOA which was increased from 0° to 40°, and the number of failed evaluations, namely, SI vectors, was recorded at each AOA. Figure 2(b) shows the obtained results, which indicate a consistent trend where a higher AOA resulted in more failed evaluations, namely, more SI vectors. Accordingly, we have selected the settings $\text{AOA} = 20^\circ, 30^\circ$, and 40° for the optimization tests. With respect to the altitude and velocity, we have experimented with different operating conditions, and have chosen a speed of $Ma = 0.775$, namely, 0.775 of the speed of sound, and an altitude of 32 kft.

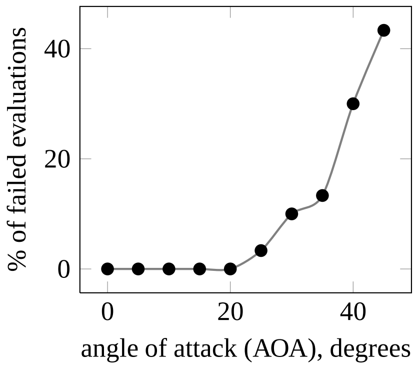
For a comprehensive evaluation, the proposed algorithm was also benchmarked against the following two representative model-assisted EAs:

- *Model-assisted EA with periodic sampling* (EA–PS) [18]: The algorithm begins by generating an initial LH sample and training a Kriging model. A real-coded EA then runs for 10 generations while evaluating only the model, and next, the top 10 elites in the population are evaluated with the true expensive function and are incorporated into the model. The goal of this procedure is to safeguard the accuracy of the model by periodically updating it with the evaluated elites. This optimization loop repeats until the optimization budget is exhausted. In the benchmarks, the EA was identical to the one in the proposed algorithm, and SI vectors were assigned a fictitious penalty taken to be the mean objective value in the initial LH sample.
- *Expected-Improvement with a model-assisted CMA-ES* (EI–CMA-ES) [1]: The algorithm begins by generating an initial sample of points and trains an initial Kriging model. The main loop then begins, where at each generation the algorithm trains a local Kriging model around the current elite using both the recently evaluated vectors, and the cached vectors which are nearest to the elite. The algorithm excludes SI vectors from the model training set. A covariance matrix adaptation evolutionary strategy (CMA-ES) algorithm then searches for an optimum of the model in a bounded region defined by the latter two sets of solutions, namely, the recently evaluated ones and the nearest neighbours, and in the spirit of the Expected-Improvement framework [10], uses the merit function

$$\hat{f}(\mathbf{x}) = m(\mathbf{x}) - \rho\zeta(\mathbf{x}), \qquad (20)$$



(a)



(b)

Fig. 2. Aspects of the airfoil optimization problem. (a) shows the physical quantities and Hicks-Henne airfoil parametrization setup. (b) shows the effect of the AOA on the prevalence of SI vectors.

where $m(\mathbf{x})$ is the Kriging model prediction, ρ is a prescribed coefficient, and $\zeta(\mathbf{x})$ is the estimate of the Kriging model prediction error, which is zero at sampled points 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 cached. In case no new solutions were evaluated, for example, because they already exist in the cache, the algorithm generates a new solution by perturbing the current elite. Following reference [1], the algorithm used a training set of 100 vectors (50 most recently evaluated ones and 50 nearest-neighbours) and the CMA-ES used the default values in the source code [7].

We have also used a variant of the proposed algorithm with a fixed classifier type, namely, only k-NN, to gauge the contribution of the classifier selection step. The variant is designated KK (Kriging–k-NN).

In all tests the optimization budget was 200 evaluations of the true objective function, that is, simulation runs, and the size of the initial sample was 20. To support a valid statistical analysis, 30 trials were repeated for each algorithm–test case combination.

Table II gives the test statistics for the three AOA cases, as well as the significance-level (α) at which the proposed algorithm was better than each of the other algorithms, namely, EA–PS, EI–CMA-ES, and KK, where an empty entry indicates no statistically-significant difference up to the 0.05 level. Statistical significance tests were done using the Mann–Whitney nonparametric test. For each AOA case, the best mean and median results are emphasized. From studying the test results for each AOA setting it follows:

- AOA=20°: The proposed algorithm obtained the best mean score, and its performance was statistically-significant better than the KK and EA–PS variants at the $\alpha = 0.01$ level. The EI–CMA-ES algorithm had the best median result. followed by the proposed algorithm. With respect to the standard deviation, the EA–PS algorithm had the best (lowest) result, followed by the proposed algorithm.
- AOA=30°: The KK variant obtained the best mean, followed by the proposed algorithm, a setup which was also repeated for the median statistic. The proposed algorithm was statistically-significant better than the EI–CMA-ES algorithm at the 0.01 level. With respect to the standard deviation, the KK algorithm had the best result, followed by the EA–PS algorithm, followed by the proposed algorithm.
- AOA=40°: The proposed algorithm had the best statistic, while the EA–PS algorithm had the best median, closely followed by the proposed algorithm. The proposed algorithm was statistically-significant better than the EI–CMA-ES algorithm at the 0.01 level. With respect to the standard deviation, the KK algorithm had the best result, followed by the proposed algorithm.

Overall, results show the proposed algorithm performed well, as it obtained either the best or near-best mean statistic, and consistently obtained the near-best median statistic, showing its performance was robust across different problem settings. Its standard deviation was intermediate between the extremal results by the other algorithms, indicating there was some variability in its performance, but it was still competitive and never the worst performing algorithm with respect to this statistic. Results also highlight the contribution of the classifier selection stage, as in two cases (AOA $= 20^\circ$ and 40°) the proposed algorithm outperformed the KK variant which does not select a classifier, and obtained results which were nearly as good in the AOA $= 30^\circ$ case.

Lastly, to demonstrate the optimization outcomes, Figure 3 shows representative airfoils obtained by the proposed algorithm at each of the three optimization cases.

TABLE II
STATISTICS FOR OBJECTIVE VALUE

AOA		P	KK	EA–PS	EI–CMA-ES
20°	mean	**-1.035e+01**	-8.091e+00	-6.889e+00	-1.023e+01
	SD	1.326e+00	1.697e+00	6.526e-01	2.025e+00
	median	-1.049e+01	-7.283e+00	-6.843e+00	**-1.107e+01**
	min	-1.302e+01	-1.138e+01	-8.837e+00	-1.192e+01
	max	-7.143e+00	-5.880e+00	-5.794e+00	-5.442e+00
	α		0.01	0.01	
30°	mean	-3.155e+00	**-3.192e+00**	-3.146e+00	-2.910e+00
	SD	4.694e-02	3.105e-02	3.345e-02	4.761e-02
	median	-3.140e+00	**-3.183e+00**	-3.140e+00	-2.916e+00
	min	-3.270e+00	-3.298e+00	-3.223e+00	-3.005e+00
	max	-3.091e+00	-3.145e+00	-3.092e+00	-2.813e+00
	α				0.01
40°	mean	**-2.793e+00**	-2.784e+00	-2.784e+00	-2.552e+00
	SD	3.380e-02	2.230e-02	4.732e-02	4.249e-02
	median	-2.785e+00	-2.782e+00	**-2.786e+00**	-2.557e+00
	min	-2.875e+00	-2.827e+00	-2.869e+00	-2.637e+00
	max	-2.726e+00	-2.742e+00	-2.717e+00	-2.455e+00
	α				0.01

P: proposed algorithm with model and classifier adaptation.
KK: proposed algorithm restricted to a Kriging model and a *k*-NN classifier.
EA–PS: EA with periodical sampling [18].
EI–CMA-ES: Expected Improvement framework with a CMA-ES optimizer [1].

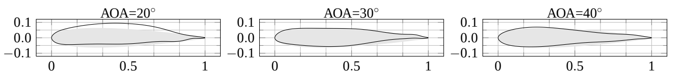


Fig. 3. Representative airfoils obtained by the proposed algorithm, shown in black. For comparison, the baseline NACA0012 airfoil is shown in gray.

V. SUMMARY

The modern engineering design process is often a simulation-driven optimization problem. In practise, there may exist candidate designs which 'crash' the simulation and thus can lead to search stagnation and to a poor final result. To effectively handle this scenario, this paper has proposed a model-assisted computational intelligence optimization algorithm which introduces a classifier into the search. The latter predicts which solutions are expected to crash the simulation, and its prediction is incorporated with the model prediction to bias the search towards valid solutions, that is, which are expected not to crash the simulator. To improve its efficacy, the proposed algorithm continuously selects during the search an optimal type of classifier, out of a prescribed family of candidates. To safeguard the optimization search in the face of model inaccuracy, the proposed algorithm also employs a TR approach. Performance analysis using a representative real-world application of airfoil shape optimization showed the efficacy of the proposed algorithm. In future work, we consider applying the proposed algorithm to additional challenging real-world applications with SI vectors.

ACKNOWLEDGEMENT

The first author thanks the Japan Society for Promotion of Science for its support.

REFERENCES

[1] 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*, 35(2):183–194, 2005.

[2] 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.

[3] A. R. Conn, N. I. M. Gould, and P. L. Toint. *Trust Region Methods.* SIAM, Philadelphia, Pennsylvania, 2000.

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

[5] M. T. M. Emmerich, A. Giotis, M. Özedmir, T. Bäck, and K. C. Giannakoglou. Metamodel-assisted evolution strategies. In J. J. Merelo Guervós, editor, *The 7th International Conference on Parallel Problem Solving from Nature–PPSN VII*, number 2439 in Lecture Notes in Computer Science, pages 361–370. Springer, Berlin, 2002.

[6] S. Handoko, C. K. Kwoh, and Y.-S. Ong. Feasibility structure modeling: An effective chaperon for constrained memetic algorithms. *IEEE Transactions on Evolutionary Computation*, 14(5):740–758, 2010.

[7] N. Hansen and A. Ostermeier. Completely derandomized self-adaptation in evolution strategies. *Evolutionary Computation*, 9(2):159–195, 2001. http://www.lri.fr/~hansen/cmaes_inmatlab.html.

[8] R. M. Hicks and P. A. Henne. Wing design by numerical optimization. *Journal of Aircraft*, 15(7):407–412, 1978.

[9] M. E. Johnson, L. M. Moore, and D. Ylvisaker. Minimax and maximin distance designs. *Journal of Statistical Planning and Inference*, 26(2): 131–148, 1990.

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

[11] J. R. Koehler and A. B. Owen. Computer experiments. In S. Ghosh, C. R. Rao, and P. R. Krishnaiah, editors, *Handbook of Statistics*, pages 261–308. Elsevier, Amsterdam, 1996.

[12] K.-H. Liang, X. Yao, and C. Newton. Evolutionary search of approximated N-dimensional landscapes. *International Journal of Knowledge-Based Intelligent Engineering Systems*, 4(3):172–183, 2000.

[13] W. R. Madych. Miscellaneous error bounds for multiquadric and related interpolators. *Computers and Mathematics with Applications*, 24(12): 121–138, 1992.

[14] 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*, 21(2):239–245, 1979.

[15] 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*, pages 131–132. Universal Academy Press, Tokyo, 2007.

[16] 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*, 186(2–4):403–420, 2000.

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

[18] A. Ratle. Optimal sampling strategies for learning a fitness model. In *The 1999 IEEE Congress on Evolutionary Computation–CEC 1999*, pages 2078–2085. IEEE, Piscataway, New Jersey, 1999.

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

[20] Y. Tenne and C. K. Goh, editors. *Computational Intelligence in Expensive Optimization Problems*, volume 2 of *Evolutionary Learning and Optimization.* Springer, 2010. URL `http://www.springerlink.com/content/v81864`.

[21] Y. Tenne, K. Izui, and S. Nishiwaki. Handling undefined vectors in expensive optimization problems. In C. Di Chio, editor, *Proceedings of the 2010 EvoStar Conference*, volume 6024/2010 of *Lecture Notes in Computer Science*, pages 582–591. Springer, Berlin, 2010.

[22] 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.

[23] 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*, 14:1–37, 2008.