Int. J. Appl. Math. Comput. Sci., 2017, Vol. 27, No. 1, 105–118
DOI: 10.1515/amcs-2017-0008

MACHINE–LEARNING IN OPTIMIZATION OF EXPENSIVE BLACK–BOX FUNCTIONS

YOEL TENNE [a]

[a] Department of Mechanical and Mechatronic Engineering
Ariel University, Ariel 40700, Israel
e-mail: y.tenne@ariel.ac.il

Modern engineering design optimization often uses computer simulations to evaluate candidate designs. For some of these designs the simulation can fail for an unknown reason, which in turn may hamper the optimization process. To handle such scenarios more effectively, this study proposes the integration of classifiers, borrowed from the domain of machine learning, into the optimization process. Several implementations of the proposed approach are described. An extensive set of numerical experiments shows that the proposed approach improves search effectiveness.

Keywords: simulations, metamodels, classifiers, machine learning.

1. Introduction

Computer simulations are being extensively used in engineering and science as a partial substitute for laboratory experiments. This allows us to investigate scenarios which may be complicated to evaluate by real-world experiments, and reduces the costs associated with product development. Such computer simulations, which must be properly validated with laboratory experiments, transform the design process into an optimization problem which is characterized by the following aspects (Tenne and Goh, 2010):

- The simulation maps candidate designs (vectors) into output values, and so it acts as the problem's objective function. The output value of the simulation may be a result of intricate numerical calculations, or the simulation code itself may be either a legacy or a commercial code whose inner mechanics are inaccessible to the user. In either case, this renders the simulation as a *black-box function*, i.e., a function for which an analytic expression in unavailable. Therefore, optimization algorithms which rely on analytic functions and gradient expressions cannot be used.

- Each run of the simulation requires considerable computer resources, and therefore, it is a *computationally expensive* process, and in turn this severely restricts the number of candidate designs (simulation runs) which can be evaluated.

- Both the underlying physics of the problem being solved and the numerical simulation process itself, often result in an objective function which has complicated features such as multiple local optima, and this further complicates the optimization process.

Such optimization problems are commonly referred to in the literature as *expensive black-box optimization problems*, and a wide range of algorithms has been proposed in an attempt to effectively handle them (Tenne and Goh, 2010; Regis and Shoemaker, 2013; Muller and Shoemaker, 2014).

On top of the above challenges, such problems often present an additional difficulty which is related to the evaluation process: for some candidate designs the simulation will fail, and no objective value will be provided. In this study, such designs are termed *simulator-infeasible* (SI), while those for which the simulation succeeds are termed *simulator-feasible* (SF). The existence of SI designs has two main implications on the optimization process:

- Since they do not have a corresponding objective value, the objective function becomes discontinuous, and this introduces an additional optimization

challenge.

- Although such designs can consume a large portion of the allowed computational budget, they do not provide any objective values to the optimization algorithm. This in turn can lead to search stagnation and a poor final result.

In this study there are two underlying assumptions regarding simulation failures:

- They are deterministic and are not caused by some random malfunction of the simulation code. Therefore, evaluations of an SF design will consistently succeed, while those of an SI design will consistently fail.
- The reason for simulation failures is unknown prior to the optimization search, and hence such SI designs cannot be excluded *a priori* from the search space.

Numerous studies, such as those by Büche *et al.* (2005), Jin *et al.* (2002), and Poloni *et al.* (2000), have reported encountering SI designs in simulation-driven problems, which indicates that the issue is both common and requires an effective strategy to address it. In these settings, this paper describes the integration of classifiers, borrowed from the domain of machine-learning, into the optimization search as a means of more effectively handling SI vectors. Specifically, the role of the classifier is to predict, prior to any evaluation, if a candidate design will lead to a simulation failure or not. Based on this prediction, the search is then biased towards designs for which the simulation is expected to succeed, thereby reducing the number of failed evaluations and focusing the search on successful designs, which in turn can yield a better final result. The effectiveness of the approach is demonstrated through several implementations and their application to simulation-driven problems.

The remainder of this paper is organized as follows. Section 2 provides the pertinent background information, Section 3 describes a baseline implementation of a classifier-assisted optimization algorithm, while Sections 4 and 5 describe implementations which select an optimal type of classifier dynamically during the search or employ an ensemble of multiple classifiers, respectively. Lastly, Section 6 concludes the paper.

2. Background

2.1. Optimization problem. Basically, simulation-driven optimization problems often arise in engineering and science, and Fig. 1 shows their typical layout. In such problems the simulation acts as the black-box function, and the optimization algorithm explores the design search space and generates new candidate designs for evaluation. In this setup, candidate designs are represented as vectors of design variables and are sent as inputs to the simulation.

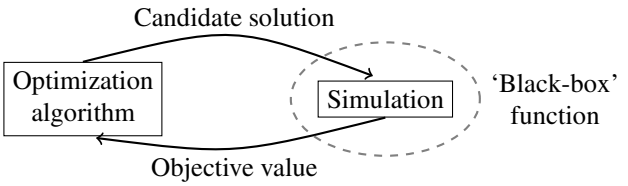

Fig. 1. Layout of an expensive black-box optimization problem.

A common strategy to circumvent the issues mentioned in Section 1 (no analytic expression, high evaluation cost, and challenging function features) is to use a *metamodel*, also termed in the literature a *response surface* or a *surrogate model*, which is a mathematical approximation of the true expensive function that provides predicted objective values at a much lower computational cost. Common metamodel variants include artificial neural networks, Kriging, polynomials, and radial basis functions (RBFs) (Wortmann *et al.*, 2015; Forrester and Keane, 2008; Queipo *et al.*, 2005; Regis, 2014; Viana *et al.*, 2013; Tenne, 2013). Metamodel-assisted frameworks typically operate by first training a metamodel and then seeking its optimum. The process could also include the sampling of additional points to update the metamodel. The search for an optimum is often performed by combining a global search algorithm such as an evolutionary algorithm (EA), particle swarm optimizer (PSO), and alike, followed by a deterministic local search performed by algorithms such as sequential quadratic programming (SQP) or steepest-descent, where gradient information is obtained from the metamodel (Smoczek, 2013; Smołka *et al.*, 2015). Algorithm 1 gives the pseudocode of the baseline algorithm.

Algorithm 1. Baseline metamodel-assisted optimization.

```
sample an initial set of vectors;
while stopping criterion not met do
    train a metamodel with the vectors evaluated so far;
    search for an optimum of the metamodel;
    evaluate the solution found with the true expensive function;
    possibly sample additional vectors, to update the metamodel;
return the best solution found;
```

Recent studies have explored more involved frameworks which include using multiple metamodels concurrently in an ensemble setup (see, e.g., Viana *et al.*, 2013; Tenne, 2015; Muller and Shoemaker, 2014), selecting an optimal metamodel type dynamically during the search (Gorissen *et al.*, 2008; Tenne, 2015), or

using more involved optimization procedures (Wortmann *et al.*, 2015; Regis, 2014).

2.2. Simulation failures. SI vectors have been reported in numerous studies, with references such as "failure of the simulation code" (Poloni *et al.*, 2000) or "attempts to evaluate the objective function failed" (Booker *et al.*, 1999), while additional examples include the works by Büche *et al.* (2005), Conn *et al.* (1998), and Okabe (2007). Accordingly, several approaches have been explored to handle such SI vectors. Rasheed *et al.* (1997) used a classifier to screen vectors before evaluating them with the simulation. Those predicted to be SI were assigned a 'death penalty' to eliminate them from consideration. It is emphasized that in their study the authors did not consider the use of metamodels. Emmerich *et al.* (2002) also used a penalty approach, but incorporated the penalized vectors into the metamodel to bias the search towards SF solutions. In contrast, Büche *et al.* (2005) ignored the SI vectors altogether when training the metamodel, and used only the SF ones. Such approaches exhibit several shortcoming in the context of expensive optimization problems:

- Incorporating penalized vectors into the metamodel training process can severely degrade its prediction accuracy.
- Ignoring SI vectors altogether does not leverage on the potentially beneficial information they present, which could be used to improve the search effectiveness.

As an example, Fig. 2 shows the effect of incorporating penalized SI vectors into the metamodel. The left panel presents a metamodel which was trained by using 30 SF vectors, while the right one the resultant metamodel when 20 SI penalized vectors were incorporated, where the penalized value was the worst objective value of the SF vectors. It is evident that the resultant metamodel is a poor approximation of the true objective function, which would in turn hamper the optimization search.

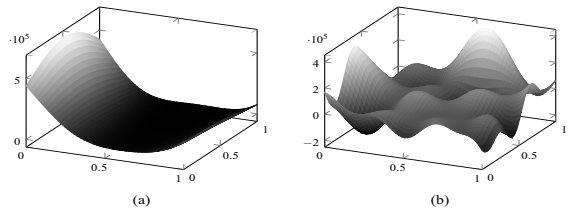

Fig. 2. Kriging metamodels of the Rosenbrock function: using a sample of 30 SF vectors (a), adding 20 SI vectors, assigned the worst objective value of the SF vectors (b).

Such shortcomings have motivated the development of alternative strategies for handling SI vectors. For example, Tenne and Armfield (2008) proposed a dual metamodel approach, in which one metamodel was used for the objective function and another provided an interpolated penalty which was proportional to the distance from the SI vectors. Other studies used classifiers for constrained nonlinear programming, though unrelated to SI vectors (Handoko *et al.*, 2010). Further exploring the use of classifiers, Tenne *et al.* (2011) obtained preliminary results with a classifier-assisted algorithm for handling SI vectors.

3. Implementation 1

This section describes a baseline implementation in which a single classifier is incorporated into the search.

3.1. Workflow. The proposed implementation begins by generating an initial sample of vectors based on a Latin hypercube (LH) design of experiments (McKay *et al.*, 1979). This method is used to ensure that the sample is space-filling, which improves the accuracy of the resultant metamodel. Briefly, for a sample of l vectors, the range of each variable is split into l equal 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 procedure is repeated l times to generate the complete sample, which is then evaluated with the expensive simulation, and the vectors are stored in memory.

The main optimization loop then begins, in which a metamodel is trained by using the SF vectors which have been evaluated so far, and then a classifier is trained based on *all* the vectors which have been evaluated, i.e., both the SF and SI ones, to account for both the vector classes. In this implementation, a k nearest neighbours (kNN) classifier and a Kriging metamodel were employed, whose details are given in Appendices A and B. Since a metamodel is trained based on a small sample of vectors, it will be inherently inaccurate, and so it is necessary to safeguard the search to ensure that it converges towards an optimum of the true objective function. To accomplish this, a trust-region (TR) approach is used, in which a trial step is performed and updates are done based on the trial outcome (Conn *et al.*, 2000). Specifically, the best vector found so far is designated as the *TR centre* ($\boldsymbol{x}_{\mathrm{b}}$), and the algorithm seeks the optimum of the metamodel in the TR, which is ball centred at $\boldsymbol{x}_{\mathrm{b}}$. The search is performed by a real-coded EA (Chipperfield *et al.*, 1994) and followed by an SQP solver. Since objective values are obtained from the metamodel, the EA uses a large population and many generations, as shown in Table 1, to improve its search effectiveness. It is

emphasized that during this trial step, the values passed on to the EA are those from the following *modified objective function*:

$$\hat{m}(\boldsymbol{x}) = \begin{cases} m(\boldsymbol{x}) & \text{if the classifier predicts } \boldsymbol{x} \text{ is SF,} \\ p & \text{if the classifier predicts } \boldsymbol{x} \text{ is SI,} \end{cases} \quad (1)$$

where $m(\boldsymbol{x})$ is the metamodel prediction and p is a penalized objective value taken to be the worst function value from the initial LH sample. Accordingly, the EA and SQP receive the metamodel prediction if the classifier predicts a vector is SF, but they receive the penalized objective value otherwise. In this setup, the classifier perseveres the knowledge about the SI vectors encountered, but they do not affect the resultant metamodel, which avoids the issues discussed in Section 2.2.

The resultant vector which was obtained in the TR step ($\boldsymbol{x}^\star$) is now evaluated with the true expensive function, and the following updates are performed (assuming a minimization problem):

- 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 ($\boldsymbol{x}_\mathrm{b}$). This implies that the metamodel is accurate at least in the TR, and hence its radius is doubled.
- If $f(\boldsymbol{x}^\star) \geq f(\boldsymbol{x}_\mathrm{b})$ *and* there are sufficient SF vectors inside the TR: The search was unsuccessful since the solution found is not better than the current best vector. This implies that the metamodel is inaccurate in the TR. However, since there are sufficient SF vectors in the TR, this inaccuracy is attributed to the TR being too large, and hence the TR radius is halved.
- If $f(\boldsymbol{x}^\star) \geq f(\boldsymbol{x}_\mathrm{b})$ *and* there are insufficient SF vectors inside the TR: As above, but the metamodel inaccuracy is attributed to the small number of vectors in the TR. Therefore, the algorithm samples new vectors in the TR, as explained below.

Compared with the classical TR framework, the framework described contracts the TR only when there are sufficient vectors in the TR. This is done to safeguard against premature convergence, which could result if the TR was to be contracted too rapidly (Conn *et al.*, 2000).

Another change from the classical TR approach is the addition of a new vector ($\boldsymbol{x}_\mathrm{n}$) to improve the metamodel accuracy in the TR. To achieve this, the new vector should be far from the existing vectors in the TR. To accomplish this efficiently, a Latin hypercube design (LHD) sample is generated in the TR, and the sample vector which has the largest distant to existing TR vectors is evaluated with the true objective function.

To complete this discussion, Algorithm 2 gives a pseudocode of the baseline implementation.

Algorithm 2. Implementation 1.

generate an initial sample and evaluate with the objective function;
repeat
  train a metamodel with the SF vectors evaluated so far;
  train a classifier with all the vectors evaluated so far;
  perform a trust-region trial step and updates;
until *maximum number of analyses reached*;

Table 1. Internal parameters of the EA utilized in this study.

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%

3.2. Performance analysis. To evaluate its effectiveness, Implementation 1 was applied to a simulation-driven problem of airfoil shape optimization in which the goal is to obtain an airfoil shape that maximizes the ratio of the lift to the drag (the aerodynamic friction) at some prescribed flight conditions. The two aerodynamic forces are represented by the lift and drag coefficients, c_l and c_d, respectively. Accordingly, the mathematical formulation of the problem is

$$\begin{aligned} \min \quad & f(\boldsymbol{x}) = -\frac{c_l}{c_d} \\ \text{s.t.} \quad & \boldsymbol{x_L} \leq \boldsymbol{x} \leq \boldsymbol{x_U}, \end{aligned} \quad (2)$$

where $\boldsymbol{x}$ is the vector of design variables and $\boldsymbol{x_L}$ $\boldsymbol{x_U}$ are the lower and upper bounds, as defined in Table 2. Additional relevant physical quantities involved are the aircraft altitude, speed, and *angle of attack* (AOA), which is the angle between the airfoil velocity vector and the *chord line*. Candidate airfoils were represented with the parametric sections (PARSEC) parameterization (Sobieszczanski-Sobieski and Haftka, 1997), which defines 11 design variables representing geometrical features. To ensure a closed airfoil shape, the condition $dz_{TE} = 0$ was enforced, while bounds on the other variables were set based on the work Tenne *et al.* (2010), and are given in Table 2. To complete the description, Fig. 3 shows the formulation of the airfoil problem.

The lift and drag coefficients of candidate airfoils were obtained by using *XFoil*, a computational fluid

Table 2. Bounds on the parametric sections (PARSEC) variables.

Variable	Geometric feature	Lower bound	Upper bound
$r_{\mathtt{LE}}$	leading-edge radius	0.001	0.003
$x_{\mathtt{upp}}$	max. upper thickness location	0.175	0.5
$z_{\mathtt{upp}}$	max. upper thickness	0	0.2
$z''_{\mathtt{upp}}$	max. upper curvature	-2	-0.05
$x_{\mathtt{low}}$	max. lower thickness location	0.175	0.5
$z_{\mathtt{low}}$	max. upper thickness	-0.2	0
$z''_{\mathtt{low}}$	max. lower curvature	0.05	2
$z_{\mathtt{TE}}$	trailing edge height	-0.1	0.1
$dz_{\mathtt{TE}}$	trailing edge thickness	0	0
$\alpha^{[1]}$	upper trailing edge angle (deg.)	150	180
$\beta^{[1,2]}$	lower trailing edge angle (deg.)	150	210

[1] measured anti-clockwise from the x-axis.
[2] $\beta \geq \alpha$ to avoid intersecting curves.

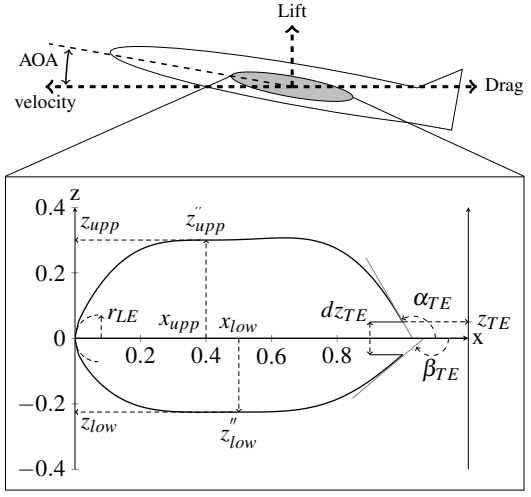

Fig. 3. Formulation of the airfoil problem in Implementation 1.

dynamics simulation tool for analysis of subsonic isolated airfoils (Drela and Youngren, 2001). Each airfoil evaluation required up to 30 seconds on a desktop computer. Three AOA settings were examined (2°, 5°, 10° and 15°) to evaluate the effectiveness of the framework in different problem settings. In particular, higher AOA values result in more frequent simulation failures, since the corresponding flow field being modeled is more turbulent and is therefore more difficult to be simulated numerically. Two variants of Implementation 1 were employed:

- I1-SVM: a variant which employed a support vector machine (SVM) classifier with Gaussian kernels,
- I1-KNN: a variant which employed a kNN classifier ($k = 3$).

For completeness, details of these classifiers are given in Appendix B. Also included in the tests were two penalty-based variants (termed *reference algorithms*) which use the same optimization steps as in Implementation 1, *except* that they did not use a classifier, but instead assigned to the SI vectors a penalized objective value and incorporated them into the metamodel. The two reference algorithms used were:

- R-1: the penalized objective value was taken to be the worst objective value in the initial sample,
- R-10: the penalized objective value was taken to be 10 times the worst objective value in the initial sample.

This test setup was employed since it highlights the effect of the proposed classifier-assisted approach. In all tests a limit of 200 function evaluations (simulation runs) was enforced, and for a valid statistical analysis, tests were repeated 30 times with each combination of the algorithm and the AOA.

Table 3 shows the resultant test statistics with the best mean and median emphasized at each AOA setting. It follows that the I1-SVM algorithm typically outperformed the other variants, as evident from the best mean median statistics it obtained for AOA $= 5°$, $15°$, and the best mean which it obtained for AOA $= 2°$, $10°$. Statistical significance analysis (at the $\alpha = 0.05$ level) shows that the performance gains of the I1-SVM variant were not statistically significant at AOA $= 2°$. For AOA $= 5°$ gains were significant over the R-10 variant, and for AOA$=10°$, they were significant over the R-1 and R-10 variants. At AOA $= 15°$, gains were borderline significant over the R-1, and significant over the R-10 variant. Overall, these results show that incorporating a classifier improved the effectiveness of the optimization search in the presence of SI vectors.

Table 3. Test statistics: Implementation 1.

α		Classifier-assisted I1-SVM	Classifier-assisted I1-KNN	R-1	R-10
2	Mean	**-2.873e+02**	-9.995e+01	-9.637e+01	-2.550e+02
	SD	5.418e+02	3.149e+01	2.660e+01	8.370e+02
	Median	-9.815e+01	-9.876e+01	-9.768e+01	**-1.126e+02**
	Min(best)	-2.395e+03	-1.835e+02	-1.670e+02	-4.604e+03
	Max(worst)	-3.192e+01	-1.084e+01	-3.425e+01	-3.070e+01
5	Mean	**-1.634e+03**	-1.498e+03	-2.820e+02	-3.104e+02
	SD	7.492e+03	4.461e+03	3.973e+02	5.424e+02
	Median	**-8.841e+01**	-1.022e+02	-9.320e+01	-8.405e+01
	Min(best)	-3.999e+04	-2.488e+04	-1.515e+03	-2.210e+03
	Max(worst)	-3.615e+01	-1.879e+01	-3.635e+01	-1.518e+01
10	Mean	**-2.435e+01**	-2.231e+01	-1.697e+01	-2.103e+01
	SD	1.999e+01	1.202e+01	9.226e+00	1.102e+01
	Median	-2.031e+01	**-2.103e+01**	-1.478e+01	-1.636e+01
	Min(best)	-1.216e+02	-7.533e+01	-4.130e+01	-4.233e+01
	Max(worst)	-1.237e+01	-8.926e+00	-6.545e+00	-4.784e+00
15	Mean	**-6.079e+00**	-5.172e+00	-5.690e+00	-6.043e+00
	SD	2.042e+00	1.502e+00	2.111e+00	3.200e+00
	Median	**-5.322e+00**	-5.321e+00	-5.009e+00	-4.598e+00
	Min(best)	-9.923e+00	-8.834e+00	-1.119e+01	-1.701e+01
	Max(worst)	-3.441e+00	-2.720e+00	-3.272e+00	-3.183e+00

I1-SVM: Implementation 1 with an SVM classifier.
I1-KNN: Implementation 1 with a kNN classifier.

4. Implementation 2

The effectiveness of the framework described in the previous section depends on the type of classifier being used, but the optimal classifier type is problem dependent and is typically unknown prior to the optimization search. This implies that the search effectiveness could be further improved by identifying the optimal classifier type for the problem being solved. In theory, this could be accomplished by performing several trial optimization runs, but in practice this approach is unsuitable due to the high computational cost of function evaluations. In these settings, this section describes a second implementation which introduces a *classifier selection stage* to autonomously select an optimal classifier type out of a family of candidates.

4.1. Workflow. The second implementation follows the main steps of Implementation 1 from Section 3, with the addition of the classifier selection step. To achieve this in a mathematically valid way, the cross-validation (CV) procedure from the domain of *model selection* is employed. Specifically, the vectors which have been evaluated with the expensive simulation (and thus have been saved in memory) are split into a *training set* and a *testing set* in a 80–20 ratio. A candidate classifier is trained by using the training set and tested on the testing set. The prediction error is calculated as

$$e = \sum_{i=1}^{l} \left(\hat{c}(\boldsymbol{x}_i) \neq F(\boldsymbol{x}_i) \right), \tag{3}$$

where $\hat{c}$ is the prediction of the trained classifier, $\boldsymbol{x}_i$, $i = 1, \ldots, l$, are the CV testing vectors, and $F(\boldsymbol{x}_i)$ is the true and known class of the latter vectors. The class labels used were $F(\boldsymbol{x}_i) = 1$ for an SF vector, and $F(\boldsymbol{x}_i) = -1$ for an SI one, as the ± 1 labels are commonly used in the literature (Wu *et al.*, 2008). Three well-established classifiers were used in the tests: kNN, linear discriminant analysis (LDA), and the SVM, and for completeness their details are given in Appendix B. After identifying the best performing classifier variant, a new classifier, which corresponds to the selected variant, is trained by using all the vectors stored in memory, and it is then used during the TR trial step. In this setup, an optimal classifier variant is re-selected at each iteration, and so the classifier being used is frequently updated during the search. To complete the description, Algorithm 3 gives the pseudocode of Implementation 2.

4.2. Performance analysis. The effectiveness of the second implementation was evaluated with the airfoil problem from Section 3.2, but here the parameterization of Hicks and Henne (1978) was used, in which a new airfoil shape is generated by combining a baseline airfoil

Algorithm 3. Implementation 2.

generate an initial sample and evaluate with the objective function;
repeat
  train a metamodel with the SF vectors evaluated so far;
  select a classifier based on cross-validation;
  train a classifier of the selected type by using all the vectors evaluated so far;
  perform a trust-region trial step and updates;
until *maximum number of analyses reached*;

shape with several shape functions, defined as

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

where h is user-prescribed (Wu *et al.*, 2003). The lower and upper curves of the new airfoil are then given by

$$y = y_b + \sum_{i=1}^{h} \alpha_i b_i(x)\,, \quad (5)$$

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. To complete this description, Fig. 4 gives the problem formulation. It is emphasized that in the figure the basis functions are shown for illustration only, and so the number of basis functions shown is different from that used in the tests.

In the numerical tests, $h = 10$ functions were used for the upper and lower airfoil curves, respectively, which resulted in a total of 20 design variables (coefficients) per airfoil. Accordingly, the formulation of the optimization problem was

$$\begin{aligned} \min \quad & f(\boldsymbol{\alpha}) = -\frac{c_l}{c_d}, \\ \text{s.t.} \quad & -0.01 \le \alpha_i \le 0.01, \end{aligned} \quad (6)$$

where $\boldsymbol{\alpha}$ is the vector of design variables from Eqn. (5). As before, candidate airfoils were evaluated with the XFoil analysis code.

For a comprehensive evaluation, the proposed algorithm was benchmarked against the following two representative metamodel-assisted algorithms:

- *Metamodel assisted EA with periodic sampling* (EA–PS): The algorithm leverages on the concepts by Ratle (1999) and de Jong (2006), where the accuracy of the metamodel is safeguarded by periodically evaluating a small subset of the EA population with the true objective function, and then incorporating them into the metamodel to improve its accuracy. The algorithm begins by generating an initial sample of vectors and evaluating them with the true expensive function. The main optimization loop then begins, in which the algorithm trains a metamodel by using the vectors evaluated so far and then employing a real-coded EA to search for the optimum of the metamodel. After the search is completed, the algorithm evaluates the ten best candidate solutions in the population with the true expensive function. The entire process is then repeated, until the maximum number of expensive function evaluations is reached. Following the above references, the algorithm used the Kriging metamodel, which is described in Appendix A.

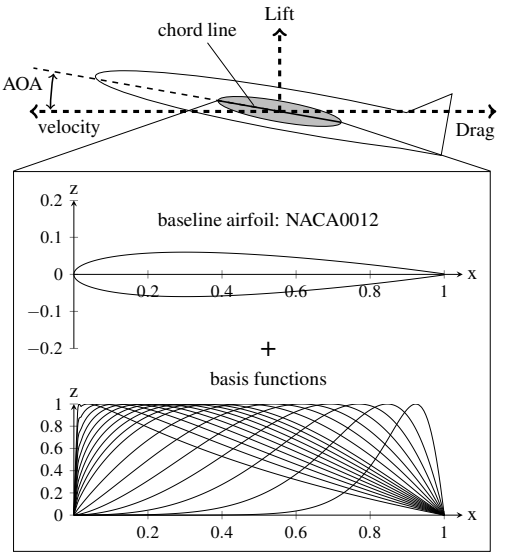

Fig. 4. Formulation of the airfoil problem in Implementation 2.

- *Expected-improvement with a metamodel-assisted CMA-ES* (EI–CMA-ES): The algorithm builds on the approach of Büche *et al.* (2005), in which a covariance matrix adaptation evolutionary strategy (CMA-ES) optimizer (Hansen and Ostermeier, 2001) is combined with a Kriging metamodel, and where metamodel updates are performed based on the expected improvement framework of Jones *et al.* (1998). The algorithm begins by generating an initial sample of vectors and evaluating them with the true function. Its main loop then stars, where at each generation it trains a local Kriging metamodel based both on the recently evaluated vectors and those stored in memory which are nearest to the best solution. A CMA-ES algorithm is then used to search for an optimum of the metamodel in the region enclosing the vectors which were employed to train the metamodel. In the spirit of the expected

improvement framework (Jones *et al.*, 1998), the objective function used was

$$\hat{f}(\boldsymbol{x}) = m(\boldsymbol{x}) - \rho\zeta(\boldsymbol{x})\,, \tag{7}$$

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 the 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, i.e., 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 stored in memory. In case no new solutions are evaluated, for example, because they already match solutions stored in memory, the algorithm generates a new solution by perturbing the current best one. Following Büche *et al.* (2005), the algorithm used a training set of 100 vectors comprising the 50 most recently evaluated ones and the 50 nearest neighbours, and the CMA-ES used the values suggested by Hansen and Ostermeier (2001).

The tests also included a variant of Implementation 2 but with a fixed classifier type, namely, only kNN, to study the contribution of the classifier selection step. This variant is designated as *I2-KK* (Implementation 2 with a Kriging–kNN combination) in the following test results. Tests were performed in three AOA settings (20°, 30°, and 40°), and with an optimization budget of 200 evaluations of the true expensive function, i.e., simulation runs. The size of the initial sample was 20. To support a valid statistical analysis, 30 trials were repeated for each algorithm and test case combination.

Table 4 gives the test statistics for the three AOA scenarios, as well as the significance-level (α) at which Implementation 2 was better than each of the other algorithms, i.e., EA–PS, EI–CMA-ES, and KK, where an empty entry indicates that no statistically significant difference was observed up to the 0.05 level. Statistical significance was measured by using the Mann–Whitney nonparametric test (Sheskin, 2007). Also, in each AOA setting, the best mean and median results, are emphasized. From the test results the following trends are observed:

- AOA $= 20°$: Implementation 2 achieved the best mean, and its performance had a statistically significant advantage over the KK and EA–PS variants at the $\alpha = 0.01$ level.
- AOA $= 30°$: Implementation 2 achieved the second best mean and median scores, following the KK variant. Its performance had a statistically significant advantage over the EI–CMA-ES algorithm at the 0.01 level.
- AOA $= 40°$: Implementation 2 achieved the best mean scores, and its performance had a statistically significant advantage over the EI–CMA-ES algorithm at the 0.01 level.

Overall, test results show that Implementation 2 typically performed better than the reference algorithms, which proves that selecting the classifier type during the search further improved its effectiveness.

5. Implementation 3

A further possibility to enhance the classifier-assisted approach is to leverage on the concept of *ensembles*, namely, employing multiple metamodels and classifiers concurrently, to improve the prediction accuracy. This section describes a third implementation which is based on this concept.

5.1. Workflow. The proposed third implementation leverages on the preceding discussion in Sections 3 and 4, but adds an ensemble training step to generate the metamodel and classifier ensembles. The specific mechanics of this training step are as follows:

(1) For each prescribed metamodel variant, a corresponding metamodel is trained by using the SF vectors which have been evaluated so far, which results in a set of metamodels $m_i(\boldsymbol{x})\,, i = 1, \ldots, n_{\mathrm{m}}\,,$ where n_{m} is the number of ensemble metamodels.

(2) The accuracy of each metamodel variant is estimated by using the CV procedure, such that the SF vectors which have been evaluated and stored in memory are split into a training set and a testing set. For each candidate variant, a corresponding metamodel is trained by using the training set, and its prediction accuracy is measured on the testing set through the mean prediction error defined as

$$\mathrm{MPE} = \frac{1}{l}\sum_{i=1}^{l}\big(\hat{m}(\boldsymbol{x}_i) - f(\boldsymbol{x}_i)\big)^2\,, \tag{8}$$

where $\hat{m}(\boldsymbol{x})$ is the metamodel trained with the training set, and $\boldsymbol{x}_i\,,\ f(\boldsymbol{x})\,,\ i = 1, \ldots, l\,,$ are the testing vectors and their corresponding objective values, respectively.

Before proceeding, it should be noted that other accuracy estimation methods exist, such as the *leave-one-out CV* (LOOCV) (Golberg *et al.*, 1996), and the related generalized cross-validation (Arlot, 2010; Molinaro *et al.*, 2005). However, the baseline CV was used here since it is both computationally efficient and can be applied to any metamodel.

Table 4. Test statistics: Implementation 2.

α		Classifier-assisted I2	Classifier-assisted I2-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(best)	-1.302e+01	-1.138e+01	-8.837e+00	-1.192e+01
	Max(worst)	-7.143e+00	-5.880e+00	-5.794e+00	-5.442e+00
	α		1.000e-02	1.000e-02	
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(best)	-3.270e+00	-3.298e+00	-3.223e+00	-3.005e+00
	Max(worst)	-3.091e+00	-3.145e+00	-3.092e+00	-2.813e+00
	α				1.000e-02
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(best)	-2.875e+00	-2.827e+00	-2.869e+00	-2.637e+00
	Max(worst)	-2.726e+00	-2.742e+00	-2.717e+00	-2.455e+00
	α				1.000e-02

I2: Proposed Implementation 2 algorithm with classifier selection.
KK: Implementation 2 with only a Kriging metamodel and a kNN classifier.

(3) Each metamodel is assigned a dedicated ensemble weight, u_i, which is calculated as

$$u_i = \frac{\text{MPE}_i{}^{-1}}{\sum_{j=1}^{n_\text{m}} \text{MPE}_j{}^{-1}}, \quad i = 1, \ldots, n_\text{m}, \tag{9}$$

where MPE_i, MPE_j are the mean prediction errors of the i-th and j-th metamodels, respectively, as defined in Eqn. (8), and n_m is the number of metamodels in the ensemble. In the numerical tests, the ensemble was comprised of the Kriging and radial basis function (RBF) metamodels, which for completeness are described in Appendix A.

(4) The predictions of the individual metamodels are aggregated to obtain the overall ensemble prediction

$$\bar{m}(\boldsymbol{x}) = \sum_{i=1}^{n_\text{m}} u_i m_i(\boldsymbol{x}), \tag{10}$$

where $m_i(\boldsymbol{x})$ and u_i are the prediction and weight of the i-th metamodel, respectively, as in the preceding steps.

In an analogous step, the algorithm then generates an ensemble of classifiers by using both the SF and SI vectors evaluated so far, to account for the two classes:

(1) For each prescribed classifier type, a classifier is trained by using all the vectors which have been evaluated so far, which results in a set of classifiers $c_i(\boldsymbol{x})$, $i = 1, \ldots, n_\text{c}$, where n_c is the number of ensemble classifiers.

(2) The accuracy of each classifier is estimated by using the CV procedure described earlier, which uses a training and a testing set, to obtain its mean classification error defined as

$$\text{MCE} = \frac{1}{l} \sum_{i=1}^{l} \big(\hat{c}(\boldsymbol{x}_i) \neq F(\boldsymbol{x}_i)\big), \tag{11}$$

where $\hat{c}(\boldsymbol{x})$ is the prediction of the classifier trained by using the training set, and $\boldsymbol{x}_i$, $F(\boldsymbol{x}_i)$, $i = 1, \ldots, l$, are the CV testing vectors and their class labels, respectively. For the latter, $F(\boldsymbol{x}_i) = 1$ was used for an SF vector and $F(\boldsymbol{x}_i) = -1$ for SI one.

(3) Each classifier is assigned a dedicated ensemble weight, v_i, which is calculated as

$$v_i = \frac{\text{MCE}_i^{-1}}{\sum_{j=1}^{n_\text{c}} \text{MCE}_j^{-1}}, \quad i = 1, \ldots, n_\text{c}, \tag{12}$$

where MCE_i, MCE_j are the classification errors of the i-th and j-th classifiers, respectively, as defined in Eqn. (11). In the numerical tests the classifier ensemble was comprised of a kNN, an LDA, and an SVM classifier, which for completeness are described in Appendix B.

Lastly, the individual classifiers predictions are aggregated to yield the overall classifier ensemble prediction

$$\bar{c}(\boldsymbol{x}) = \begin{cases} +1 \text{ (=SF)} & \text{if } \sum\limits_{i=1}^{n_\text{c}} v_i c_i(\boldsymbol{x}) > 0, \\ -1 \text{ (=SI)} & \text{otherwise.} \end{cases} \tag{13}$$

To complete this description, Algorithm 4 gives the pseudocode of the proposed algorithm.

Algorithm 4. Implementation 3.

generate an initial sample and evaluate with the objective function
repeat
  train a metamodel ensemble by using the SF vectors evaluated so far;
  train a classifier ensemble by using all the vectors evaluated so far;
  perform a trust-region trial step based on the ensemble outputs, and perform updates;
until *maximum number of analyses reached*;

5.2. Performance analysis. Performance analysis used the problem from Section 4, but flight conditions were different and, accordingly, the optimal values attained. For evaluation, Implementation 3 was benchmarked against the EA–PS and CMA-ES reference algorithms described in Section 4, along with two variants of Implementation 3: (i) *I3-KK*, which used a Kriging metamodel and a kNN classifier, and (ii) *I3-RS*, which employed an RBF metamodel and an SVM classifier. The latter two variants used the same steps as in Implementation 3 except that they employed a single type of metamodel and classifier. They were added to the numerical tests to gauge the contribution of using the ensemble approach. In all tests, the optimization budget was 200 expensive evaluations, that is, calls to the numerical simulation. To support a valid statistical analysis, 30 runs were repeated with each algorithm in each test case.

Table 5 gives the test statistics for the three AOA scenarios, as well as the significance-level (α) at which Implementation 3 was better than each of the other algorithms, i.e., EA–PS, EI–CMA-ES, KK, and RS, where an empty entry indicates no statistically significant difference up to the 0.05 level. Statistical significance was measured by using the Mann–Whitney nonparametric test (Sheskin, 2007).

It follows that Implementation 3 performed well, as evident from its corresponding mean and median statistics, which typically outperformed those of the other algorithms. It obtained an intermediate standard deviation (SD) score, which indicates that there was some variability in its performance, but it was comparable to that of the other algorithms. Lastly, its performance had a statistically significant advantage in 6 out of 12 comparisons (four algorithms in three AOA settings), which further indicates it performed well. Overall, the test results show that using an ensemble setup further improved search effectiveness when compared to employing a single metamodel and classifier.

6. Conclusion

Expensive-black optimization problems which rely on computer simulations are nowadays common in engineering and science. Such problems will often contain candidate designs which cause the simulation to fail, but where the reason for failure is unknown. This, in turn, may degrade search effectiveness, and lead to a poor final result. To address this issue, this study described three implementations which incorporate classifiers into the optimization search. The role of the classifiers is to predict if a candidate design would result in a simulation-failure, and then the search is biased towards designs for which the simulation evaluation is expected to succeed. A baseline implementation of this concept employs a single type of classifier, while two more involved implementations either select the classifier type during the search, or generate an ensemble of metamodels and classifiers. Performance analysis based on a simulation-driven problem of airfoil shape optimization, which is representative of real-world engineering problems, showed that incorporating classifiers improved search effectiveness, and yielded promising results when compared with several existing algorithms from the literature.

References

Arlot, S. (2010). A survey of cross-validation procedures for model selection, *Statistics Survey* **4**: 40–79.

Benoudjit, N., Archambeau, C., Lendasse, A., Lee, J. and Verleysen, M. (2002). Width optimization of the Gaussian Kernels in radial basis function networks, *Proceedings of the 10th European Symposium on Artificial Neural Networks, ESANN 2002, Bruges, Belgium*, pp. 425–432.

Booker, A.J., Dennis, J.E., Frank, P.D., Serafini, D.B., Torczon, V. and Trosset, M.W. (1999). A rigorous framework for optimization of expensive functions by surrogates, *Structural Optimization* **17**(1): 1–13.

Büche, D., Schraudolph, N.N. and Koumoutsakos, P. (2005). Accelerating evolutionary algorithms with Gaussian process fitness function models, *IEEE Transactions on Systems, Man, and Cybernetics C* **35**(2): 183–194.

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

Conn, A.R., Gould, N.I.M. and Toint, P.L. (2000). *Trust Region Methods*, SIAM, Philadelphia, PA.

Conn, A.R., Scheinberg, K. and Toint, P.L. (1998). A derivative free optimization algorithm in practice, *Proceedings of the 7th AIAA/USAF/NASA/ISSMO Symposium on Multidisciplinary Analysis and Optimization, St. Louis, MO, USA*, Paper No. AIAA-1998-4718.

de Jong, K.A. (2006). *Evolutionary Computation: A Unified Approach*, MIT Press, Cambridge, MA.

Table 5. Test statistics: Implemenation 3 benchmarks.

α		Classifier-assisted I3	I3-KK	I3-RS	EA–PS	EI–CMA-ES
20	Mean	**3.669e-01**	3.718e-01	3.814e-01	4.418e-01	5.675e-01
	SD	7.573e-03	8.892e-03	4.515e-02	5.773e-02	2.102e-01
	Median	**3.672e-01**	3.700e-01	3.687e-01	4.333e-01	5.052e-01
	Min(best)	3.518e-01	3.577e-01	3.579e-01	3.674e-01	3.584e-01
	Max(worst)	3.780e-01	3.924e-01	5.395e-01	5.686e-01	9.638e-01
	α		5.000e-02		1.000e-02	1.000e-02
30	Mean	8.357e-01	8.113e-01	8.298e-01	9.842e-01	**7.768e-01**
	SD	7.070e-02	6.413e-02	5.166e-02	1.237e-01	6.908e-02
	Median	8.230e-01	8.035e-01	8.293e-01	1.026e+00	**7.650e-01**
	Min(best)	7.617e-01	6.617e-01	7.397e-01	7.113e-01	6.831e-01
	Max(worst)	1.012e+00	9.466e-01	9.011e-01	1.099e+00	1.005e+00
	α				1.000e-02	
40	Mean	**9.529e-01**	9.719e-01	9.705e-01	1.112e+00	9.761e-01
	SD	4.361e-02	5.160e-02	4.218e-02	4.635e-02	4.063e-02
	Median	**9.528e-01**	9.599e-01	9.592e-01	1.116e+00	9.668e-01
	Min(best)	8.568e-01	8.814e-01	8.971e-01	1.006e+00	9.111e-01
	Max(worst)	1.054e+00	1.042e+00	1.046e+00	1.204e+00	1.079e+00
	α				1.000e-02	

I3: Proposed Implementation 3 algorithm with ensembles.
KK: Proposed Implementation 3 but only with a Kriging metamodel and a kNN classifier.

Drela, M. and Youngren, H. (2001). Xfoil 6.9 user primer, *Technical report*, MIT, Cambridge, MA.

Emmerich, M.T.M., Giotis, A., Özedmir, M., Bäck, T. and Giannakoglou, K.C. (2002). Metamodel-assisted evolution strategies, *in* J.J. Merelo Guervós (Ed.), *7th International Conference on Parallel Problem Solving from Nature—PPSN VII*, Lecture Notes in Computer Science, Vol. 2439, Springer, Berlin, pp. 361–370.

Forrester, A.I.J. and Keane, A.J. (2008). Recent advances in surrogate-based optimization, *Progress in Aerospace Science* **45**(1–3): 50–79.

Golberg, M.A., Chen, C.S. and Karur, S.R. (1996). Improved multiquadric approximation for partial differential equations, *Engineering Analysis with Boundary Elements* **18**(1): 9–17.

Gorissen, D., De Tommasi, L., Croon, J. and Dhaene, T. (2008). Automatic model type selection with heterogeneous evolution: An application to RF circuit block modeling, *Proceedings of the IEEE Congress on Evolutionary Computation (CEC), Hong Kong, China*, pp. 989–996.

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

Hansen, N. and Ostermeier, A. (2001). Completely derandomized self-adaptation in evolution strategies, *Evolutionary Computation* **9**(2): 159–195.

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

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

Jones, D.R., Schonlau, M. and Welch, W.J. (1998). Efficient global optimization of expensive black-box functions, *Journal of Global Optimization* **13**(4): 455–492.

McKay, M.D., Beckman, R.J. and Conover, W.J. (1979). 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.

Molinaro, A.M., Simon, R. and Pfeiffer, R.M. (2005). Prediction error estimation: A comparison of resampling methods, *Biometrika* **21**(15): 3301–3307.

Muller, J. and Shoemaker, C.A. (2014). Influence of ensemble surrogate models and sampling strategy on the solution quality of algorithms for computationally expensive black-box global optimization problems, *Journal of Global Optimization* **60**(2): 123–144.

Okabe, T. (2007). Stabilizing parallel computation for evolutionary algorithms on real-world applications, *Proceedings of the 7th International Conference on Optimization Techniques and Applications (ICOTA 7), Kobe, Japan*, pp. 131–132.

Poloni, C., Giurgevich, A., Onseti, L. and Pediroda, V. (2000). 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.

Powell, M.J.D. (2001). Radial basis function methods for interpolation of functions of many variables, *Proceedings of the 5th Hellenic-European Conference on Computer Mathematics and Its Applications (HERCMA-01), Athens, Greece*, pp. 2–24.

Queipo, N.V., Haftka, R.T., Shyy, W., Goel, T., Vaidyanathan, R. and Tucker, K.P. (2005). Surrogate-based analysis and optimization, *Progress in Aerospace Science* **41**(1): 1–28.

Rasheed, K., Hirsh, H. and Gelsey, A. (1997). A genetic algorithm for continuous design space search, *Artificial Intelligence in Engineering* **11**(3): 295–305.

Ratle, A. (1999). Optimal sampling strategies for learning a fitness model, *1999 IEEE Congress on Evolutionary Computation—CEC 1999, Washington, DC, USA*, pp. 2078–2085.

Regis, R.G. (2014). Particle swarm with radial basis function surrogates for expensive black-box optimization, *Journal of Computational Science* **5**(1): 12–23.

Regis, R.G. and Shoemaker, C.A. (2013). A quasi-multistart framework for global optimization of expensive functions using response surface models, *International Journal of Global Optimization* **56**(4): 1719–1753.

Sacks, J., Welch, W.J., Mitchell, T.J. and Wynn, H.P. (1989). Design and analysis of computer experiments, *Statistical Science* **4**(4): 409–435.

Sheskin, D.J. (2007). *Handbook of Parametric and Nonparametric Statistical Procedures*, 4th Edn., Chapman and Hall, Boca Raton, FL.

Smoczek, J. (2013). Evolutionary optimization of interval mathematics-based design of a TSK fuzzy controller for anti-sway crane control, *International Journal of Applied Mathematics and Computer Science* **23**(4): 749–759, DOI: 10.2478/amcs-2013-0056.

Smołka, M., Schaefer, R., Paszyński, M., Pardo, D. and Álvarez Aramberri, J. (2015). An agent-oriented hierarchic strategy for solving inverse problems, *International Journal of Applied Mathematics and Computer Science* **25**(3): 483–498, DOI: 10.1515/amcs-2015-0036.

Sobieszczanski-Sobieski, J. and Haftka, R.T. (1997). Multidisciplinary aerospace design optimization: Survey of recent developments, *Structural Optimization* **14**(1): 1–23.

Tenne, Y. (2013). An optimization algorithm employing multiple metamodels and optimizers, *International Journal of Automation and Computing* **10**(3): 227–241.

Tenne, Y. (2015). An adaptive-topology ensemble algorithm for engineering optimization problems, *Optimization and Engineering* **16**(2): 303–334.

Tenne, Y. and Armfield, S.W. (2008). A versatile surrogate-assisted memetic algorithm for optimization of computationally expensive functions and its engineering applications, *in* A. Yang *et al.* (Eds.), *Success in Evolutionary Computation*, Studies in Computational Intelligence, Vol. 92, Springer-Verlag, Berlin/Heidelberg, pp. 43–72.

Tenne, Y. and Goh, C.K. (Eds.) (2010). *Computational Intelligence in Expensive Optimization Problems*, Springer, Berlin.

Tenne, Y., Izui, K. and Nishiwaki, S. (2010). Handling undefined vectors in expensive optimization problems, *in* C. Di Chio (Ed.), *Proceedings of the 2010 EvoStar Conference*, Lecture Notes in Computer Science, Vol. 6024, Springer, Berlin, pp. 582–591.

Tenne, Y., Izui, K. and Nishiwaki, S. (2011). A classifier-assisted framework for expensive optimization problems: A knowledge-mining approach, *in* C.A. Coello-Coello (Ed.), *Proceedings of the 5th Learning and Intelligent Optimization Conference (LION 5)*, Lecture Notes in Computer Science, Vol. 6683, Springer, Berlin/Heidelberg, pp. 161–175.

Viana, F.A.C., Haftka, R.T. and Watson, L.T. (2013). Efficient global optimization algorithm assisted by multiple surrogate technique, *Journal of Global Optimization* **56**(2): 669–689.

Wortmann, T., Costa, A., Nannicini, G. and Schroepfer, T. (2015). Advantages of surrogate models for architectural design optimization, *Artificial Intelligence for Engineering Design, Analysis and Manufacturing* **29**(4): 471–481.

Wu, H.-Y., Yang, S., Liu, F. and Tsai, H.-M. (2003). Comparison of three geometric representations of airfoils for aerodynamic optimization, *Proceedings of the 16th AIAA Computational Fluid Dynamics Conference, Orlando, FL, USA*, pp. 1–11, Paper no. AIAA 2003-4095.

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

Yoel Tenne obtained his PhD at Sydney University, Australia, and was then an Australia Endeavour fellow at the Korea Advanced Institute of Technologu (KAIST), S. Korea, and a JSPS fellow at Kyoto University, Japan. He is currently a senior lecturer at Ariel University, Israel. His research domains include applied computational intelligence, systems engineering, and applied optimization.

Appendix A
Metamodel variants

The details of the metamodels used in this study are as follows.

Kriging: This is a statistically based metamodel which combines a global coarse approximation with a local correction which is based on the correlation between the interpolation vectors. The metamodel replicates the training sample, namely,

$$m(\boldsymbol{x}_i) = f(\boldsymbol{x}_i)\,, \quad i = 1, \ldots, n\,, \tag{A1}$$

where $m(\boldsymbol{x})$ and $f(\boldsymbol{x})$ are the metamodel and the true objective function, respectively, and $\boldsymbol{x}_i$ are the sample vectors. Using a constant global function gives the Kriging metamodel

$$m(\boldsymbol{x}) = \beta + \kappa(\boldsymbol{x})\,, \tag{A2}$$

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

mean zero and covariance,

$$\mathrm{Cov}[\kappa(\boldsymbol{x})\kappa(\vec{y})] = \sigma^2 c(\theta\,,\boldsymbol{x},\vec{y})\,, \tag{A3}$$

where $c(\theta\,,\boldsymbol{x},\vec{y})$ is a user-prescribed correlation function. A common choice for the latter is the Gaussian correlation function (Forrester and Keane, 2008), defined as

$$c(\theta\,,\boldsymbol{x},\vec{y}) = \prod_{i=1}^{d} \exp\left(-\theta\,(x_i - y_i)^2\right)\,, \tag{A4}$$

and combining it with the constant drift function transforms the metamodel from (A2) into the following form:

$$m(\boldsymbol{x}) = \hat{\beta} + \vec{r}(\boldsymbol{x})^{\mathrm{T}} R^{-1}(\vec{f} - \vec{1}\hat{\beta})\,. \tag{A5}$$

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

$$\vec{r}^{\mathrm{T}} = [c(\theta, \boldsymbol{x}\,, \boldsymbol{x}_1), \ldots, c(\theta\,, \boldsymbol{x}\,, \boldsymbol{x}_n)]\,. \tag{A6}$$

The estimated drift coefficient $\hat{\beta}$ and variance $\hat{\sigma}^2$, which are required in Eqn. (A5), are obtained as follows:

$$\hat{\beta} = \left(\vec{1}^{\mathrm{T}} R^{-1} \vec{1}\right)^{-1} \vec{1}^{\mathrm{T}} R^{-1} \vec{f}, \tag{A7a}$$

$$\hat{\sigma}^2 = \frac{1}{n}\left[(\vec{f} - \vec{1}\hat{\beta})^{\mathrm{T}} R^{-1}(\vec{f} - \vec{1}\hat{\beta})\right]\,. \tag{A7b}$$

Fully defining the metamodel requires the correlation parameters $\boldsymbol{\theta}$, which are commonly taken as the maximizers of the metamodel likelihood. This is achieved by minimizing the expression (Sacks *et al.*, 1989)

$$\psi(\boldsymbol{\theta}) = |R|^{1/n}\hat{\sigma}^2\,, \tag{A8}$$

which is a function only of $\vec{\theta}$ and the sample data.

Radial basis functions (RBFs): The metamodel approximates the objective function as a superposition of *basis functions* of the form

$$\phi_i(\boldsymbol{x}) = \phi(\|\boldsymbol{x} - \boldsymbol{x}_i\|_2)\,, \tag{A9}$$

where $\boldsymbol{x}_i$ is a sampled vector. Given the training sample vectors and responses $\boldsymbol{x}_i$, $f(\boldsymbol{x}_i)$, $i = 1, \ldots, n$, the metamodel is given by

$$m(\boldsymbol{x}) = \alpha_i \sum_{i=1}^{n} \phi_i(\boldsymbol{x}) + c\,, \tag{A10}$$

where α_i and c are coefficients which are determined from the interpolation conditions

$$m(\boldsymbol{x}_i) = f(\boldsymbol{x}_i), \quad i = 1, \ldots, n\,, \tag{A11a}$$

$$\sum_{i=1}^{n} \alpha_i = 0\,. \tag{A11b}$$

A common choice is the Gaussian basis function (Powell, 2001)

$$\phi_i(\boldsymbol{x}) = \exp\left(-\frac{\boldsymbol{x} - \boldsymbol{x}_i}{\tau}\right)\,, \tag{A12}$$

where τ controls the width of the Gaussians and is determined by cross-validation (Forrester and Keane, 2008; Benoudjit *et al.*, 2002). The weight coefficients α_i are obtained by solving the linear system

$$\begin{pmatrix} \Phi & \vec{1} \\ \vec{1}^T & 0 \end{pmatrix} \vec{\alpha} = \begin{bmatrix} \vec{f} \\ \vec{0} \end{bmatrix}\,, \tag{A13}$$

where $\boldsymbol{\Phi}$ is the Gram matrix such that $\Phi_{i,j} = \phi_i(\boldsymbol{x}_j)$, $\vec{1}$ is a vector whose elements are all one, and $\vec{f}$ is the vector of objective values of the sample vectors.

Appendix B
Classifier variants

The details of the classifier variants used in this study are as follows.

k nearest neighbours (kNN): The classifier assigns the new vector the class of the closest training vector (the nearest neighbor), i.e.,

$$c(\boldsymbol{x}) = F(\boldsymbol{x}_{\mathrm{NN}}) : d(\boldsymbol{x}, \boldsymbol{x}_{\mathrm{NN}}) = \min\, d\big(\boldsymbol{x}, \boldsymbol{x}_i\big)\,, \quad i = 1, \ldots, n\,, \tag{B1}$$

where $d(\boldsymbol{x}, \vec{y})$ is 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 was $k = 3$.

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 $\vec{w}$ which maximizes this ratio. The vector $\vec{w}$ is such that it is orthogonal to the hyperplane separating the two classes. A new vector $\vec{x}$ is classified based on its projection with respect to the separating hyperplane, that is,

$$c(\boldsymbol{x}) = \mathrm{sign}\,(\vec{w} \cdot \vec{x})\,. \tag{B2}$$

Support vector machines (SVMs): The method projects the data into a high-dimensional space so it can be more easily separated into disjoint classes. For a linearly separable training set comprising of two classes, 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. To fix the hyperplane, a condition is added that the hyperplane should maximize its distance to the nearest vectors from each class. This is accomplished by maximizing the Lagrangian

$$L_P = \frac{1}{2}\|\vec{w}\| - \sum_{i=1}^{n} \alpha_i F(\boldsymbol{x}_i)(\vec{w} \cdot \vec{x}_i + b) + \sum_{i=1}^{n} \alpha_i \,, \quad \text{(B3)}$$

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

Received: 3 March 2016
Revised: 4 September 2016
Accepted: 11 October 2016