An Optimization Algorithm Employing Multiple Metamodels and Optimizers

Yoel Tenne

Department of Mechanical and Mechatronic Engineering, Ariel University Centre, Ariel 40700, Israel

Abstract: Modern engineering design optimization often relies on computer simulations to evaluate candidate designs, a setup which results in expensive black-box optimization problems. Such problems introduce unique challenges, which has motivated the application of metamodel-assisted computational intelligence algorithms to solve them. Such algorithms combine a computational intelligence optimizer which employs a population of candidate solutions, with a metamodel which is a computationally cheaper approximation of the expensive computer simulation. However, although a variety of metamodels and optimizers have been proposed, the optimal types to employ are problem dependant. Therefore, a priori prescribing the type of metamodel and optimizer to be used may degrade its effectiveness. Leveraging on this issue, this study proposes a new computational intelligence algorithm which autonomously adapts the type of the metamodel and optimizer during the search by selecting the most suitable types out of a family of candidates at each stage. Performance analysis using a set of test functions demonstrates the effectiveness of the proposed algorithm, and highlights the merit of the proposed adaptation approach.

Keywords: Expensive optimization problems, computational intelligence, adaptive algorithms, metamodeling, model selection.

1 Introduction

Nowadays, researchers extensively rely on computer simulations to evaluate candidate designs and to reduce the time and the cost of the engineering design process. Such simulations, which must be properly validated with real-world experiments, transform the design process into an optimization problem having three distinct features:

1) Objective values are obtained from the simulation, which is often a legacy code or a commercial software available only as an executable. As such, the simulation is treated as a black-box function, i.e., a function with no analytic expression.

2) Each simulation run is expensive, i.e., it requires extensive computational resources, often minutes to weeks of CPU time. This implies that the expensive function (the simulation) can be called only a small number of times.

3) Due to the numerical simulation and the underlying physics being modelled, the objective landscape may contain many local optima and exhibit other complicated features, which exacerbate the optimization difficulty.

Accordingly, such problems are often referred to as expensive optimization problems^[1].

Due to these difficulties, classical optimization algorithms may perform poorly in such settings, and this has motivated researchers to explore new optimization strategies. One such well-established strategy is that of metamodel-assisted computational intelligence (CI) optimization algorithms, which combines two powerful approaches:

1) In contrast to classical gradient-based optimizers which operate on a single candidate solution, CI optimizers employ a population of candidate solutions and manipulate them at each iteration to produce a new set of candidate solutions^[2]. CI optimizers typically perform well over a range of different objective functions, do not require gradi-

ent information, and can handle challenges such as multimodality and discontinuities in the objective function landscape.

2) Metamodels are computationally cheaper approximations of the true expensive function, typically interpolants trained using previously evaluated solutions. During the optimization search, the optimizer receives a predicted response from the metamodel at a fraction of the computational cost when compared to using the true expensive function. Metamodels allow to use CI optimizers efficiently, which typically require many thousands of evaluations to converge, on a limited computational budget.

Although different types of optimizers and metamodels have been proposed, the optimal type to employ is typically problem dependant, i.e., different optimizers and metamodels suit different problems. Therefore, a priori prescribing the optimizer and metamodel type may result in a setup which is unsuitable to the problem being solved, and can degrade the search effectiveness.

Leveraging on this issue, this study proposes a new CI metamodel-assisted algorithm which adapts the type of optimizer and metamodel during the search, and selects an optimal combination of the latter two out of a family of candidates. This eliminates the need to a priori prescribing the type of the metamodel and optimizer, and allows the optimization search to self-adapt to different problems. To the best of our knowledge, such an adaptive metamodel-assisted CI algorithm is new. Extensive performance analysis, using both mathematical test functions and a representative engineering application, demonstrates the effectiveness of the proposed algorithm and highlights the contribution of the proposed approach.

The remainder of this paper is organized as follows. Section 2 provides the pertinent background information on CI optimizers and expensive optimization problems. Section 3 describes the proposed algorithm. Section 4 provides a de-

tailed performance analysis. And Section 5 concludes this paper.

2 Background

2.1 Expensive optimization problems

Expensive optimization problems arise in diverse domains, and as mentioned in Section 1, a typical scenario is an engineering design optimization problem driven by computer simulations. Fig. 1 shows the layout of such problem, where the simulation is viewed as a black-box function, i.e., it assigns objective values to candidate designs while its analytic expression is unknown. In such optimization problem candidate designs are represented as vectors of design variables, and are provided as inputs to the simulation.

Fig. 1 The layout of an expensive black-box optimization problem

Also as mentioned, the resultant objective function often has a complicated, nonconvex landscape, which can lead gradient-based optimizers to converge to a poor final result. This has motivated the application of CI optimizers in such problems, since these optimizers are more explorative and therefore often perform better in nonconvex function landscapes. Such optimizers typically employ a population of candidate solutions, and manipulate them using a variety of operators. For example, the evolutionary algorithm (EA) employs operators inspired by the paradigms of adaptation and survival of the fittest in nature, such that the best candidate solutions in the population are combined to produce offspring. As another example, the particle swarm optimizer (PSO) employs a set of particles which explore the search space, and it modifies their trajectories based on the observed function values. Through their respective operators, the various CI optimizers emphasize the best performing member in the population, which leads to gradual adaptation of the population to the function landscape and convergence to an optimum.

Since CI optimizers rely on a population of candidate solutions and do not utilize gradient information, they often require many thousands of function evaluations to obtain a satisfactory solution. This is a major obstacle in applying them to expensive optimization problems where the objective function can be evaluated only a small number of times. An established methodology to circumvent this is to employ a metamodel which is an approximation of the true expensive function, but is computationally much cheaper to evaluate. Metamodels (also termed as surrogates or response surfaces) are typically interpolants trained with previously evaluated vectors, and variants include artificial neural networks, Kriging, polynomials, and radial basis functions (RBF), which are described in Appendix A^[3,4].

Much research has been devoted to the development of

metamodel-assisted optimization algorithms. Early examples include the algorithm in [5] which studied an evolutionary algorithm assisted with a Kriging metamodel, and the algorithm in [6] which coupled a pattern search optimizer with quadratic polynomials. Later examples include the algorithm which used an evolutionary strategies optimizer coupled with a Kriging metamodel^[7], and the algorithm which coupled an evolutionary algorithm with a least-square fit polynomial^[8]. Poloni et al.^[9] and Muy et al.^[10] studied algorithms coupled with an artificial neural network.

In [11], Goel et al. explored the concept of a metamodel ensemble, in which multiple metamodels were employed concurrently, and their predictions were aggregated based on the estimated accuracy of each metamodel. Similar concepts were explored by Jin and Sendhoff^[12], which used an ensemble of artificial neural networks, and Lim et al.^[13] which used an ensemble comprised of an RBF and a polynomial metamodel. Acar and Rais-Rohani^[14] proposed to optimize the weights of the individual ensemble metamodels to improve the overall ensemble accuracy, which transforms the ensemble training process into an optimization problem of its own. Acar^[15] studied the use of the pointwise cross-validation error estimation as a mean for determining the ensemble composition, and observed that the resultant ensembles did not perform well since this measure was not a reliable accuracy estimator. Zhou et al.^[16] proposed to configure the ensemble based on a recursive arithmetic averaging procedure, and showed this improved the ensemble prediction accuracy.

Another strategy which considers multiple metamodels is that of model selection, in which the accuracy of several metamodels is estimated, and a single metamodel, which is deemed as the most accurate, is employed. Following this methodology, Gorissen et al.^[17] proposed a framework in which the metamodels were selected by an evolutionary algorithm, a setup which resulted in a dynamic selection of the metamodel type and its complexity during the search. In a related study, Tenne and Armfield^[18] presented a framework which used global and local metamodels, and selected an optimal type for each based on statistical cross-validation techniques. In this setup, the algorithm autonomously selected the optimal global and local metamodel types at each phase of the optimization search. Recently, Gorissen et al.^[19] described a toolbox for training and selection of metamodels.

Also pertinent is the variable fidelity approach, in which the optimization algorithm relies on several simulations which differ by their accuracy, and accordingly by their computational cost. The goal is then to try and maximize the use of the computationally cheaper simulations while ensuring convergence to an optimum of the problem, i.e., an optimum of the objective function of the highest fidelity simulation. In two early examples, both Rodriguez et al.^[20] and Alexandrov et al.^[21] proposed frameworks which adjusted the simulation fidelity based on the trust-region approach. Sefrioui and Periaux^[22] proposed a hierarchical setup which consisted of a bottom layer in which an evolutionary algorithm with a large population used a low fidelity simulation. Also, a top layer was employed in which an EA with a small population used a high fidelity simulation. The

best performing individuals from the bottom layer would then migrate into a top layer for more exact evaluation, and vice versa. Also in the class, Lim et al.^[23] explored an evolutionary based optimization algorithm which adjusted the simulation fidelity during the search, and based on the correlation between adjacent members in the population of candidate solutions.

An extensive survey on expensive optimization problems and metamodel-assisted optimization is given by [3], which provides a detailed discussion on a variety of related topics. Forrester and Keane^[4] addressed similar topics, but focused on the Kriging metamodel and metamodel updating strategies based on the expected-improvement framework of [24]. Tenne and Goh^[1] provide a collection of recent studies which focus on CI and metamodel-assisted optimization.

2.2 Related studies: Algorithms employing multiple metamodels or optimizers

Researchers have explored algorithms which employ multiple metamodels or optimizers with the goal of improving the search effectiveness.

With respect to multiple metamodels, an early example, which was also mentioned in Section 2.1, is the framework of hierarchical optimization of [22], which employed multiple simulations of different fidelity. A similar setup was explored by [25] and used evolutionary strategies optimizers. Also as mentioned, Tenne and Armfield^[18] proposed an optimization framework which employed global and local metamodels, and autonomously selected the type of each metamodel during the optimization search. Zhou et al.^[26] used a hybrid algorithm which coupled an EA and an SQP optimizer, and used an radial basis functions and a polynomial metamodels to benefit from the inaccuracy of the metamodel predictions. Cohen and Intrator^[27] used an artificial neural network (ANN) comprising of clusters of both radial basis functions and multilayer perceptron (MLP) components, and used tests, such as error levels and confidence intervals, to configure the network topology.

With respect to employing multiple optimizers, Neri et al.^[28] studied the problem of optimizing the topology of an MLP neural network, and optimized its weights using an EA coupled with a simulated annealing (SA) and a Hook-Jeeves optimizers. Caponio et al.^[29] coupled an EA with a Hooke-Jeeves and a simplex optimizer for the problem of optimizing the performance of a motor. Egea et al.^[30] proposed an algorithm for optimization of chemical processes which employed a scatter search optimizer as a global search procedure, while promising solutions were then refined with a simplex optimizer. Wang et al.^[31] studied an algorithm for dynamic optimization problems in which an EA optimizer was coupled with a hill climbing optimizer. They also described various specialized EA operators to enhance the search effectiveness.

Most studies, such as those referenced above, used a fixed set of metamodels or optimizers, and did not select which types to use based on the problem being solved. However, the optimal type of the latter two is problem dependant. To demonstrate this, Table 1 shows the results of the two numerical experiments. In the first, three types of metamodels (radial basis function, radial basis functions network, and

Kriging), which are described in Appendix A, were used to approximate the functions Ackley-10D, Rastrigin-20D, and Rosenbrock-20D. For each function, the three metamodels were trained using the same set of 30 vectors, and their prediction error, measured by the root mean square error (RMSE), was calculated using the same set of 20 new test vectors. Table 1 (a) shows the resultant prediction error, from which it follows that no single metamodel was overall best, and that the most suitable metamodel varied with the function and dimension. In the second experiment, three different types of optimizers, namely, differential evolution (DE), evolutionary algorithm, and particle swarm optimizer, which are described in Appendix B, were used to search for the optimum of several metamodels. For each test function, a metamodel was trained using a sample of 50 vectors, and each optimizer was applied repeatedly for 5 times. Table 1 (b) shows the best solution found by each optimizer in each case, from which it follows that no single optimizer was overall best, and the best optimizer to use again varied between the test cases.

Table 1 Comparison of metamodel accuracy and optimizer performance across several test functions (RMS error)
(a) Prediction accuracy of different metmodels (RMSE)

Function	RBF	RBFN	Kriging
Ackley-10D	1.583E-01	2.699E-01	2.329E-01
Griewank-10D	2.238E-03	9.899E-03	1.487E-03
Rastrigin-20D	2.911E+01	2.673E+01	4.167E+01
Rosenbrock-20D	5.218E+04	7.064E+04	7.139E+04

(b) The best solution obtained by different optimizers

Function	Metamodel	DE	EA	PSO
Ackley-10D	RBF	2.072E+01	1.966E+01	2.005E+01
Griewank-10D	Kriging	1.388E+00	1.387E+00	1.388E+00
Rastrigin-20D	RBFN	1.683E+02	1.726E+02	1.705E+02
Rosenbrock-20D	Kriging	7.496E+05	5.404E+05	5.269E+05

These results imply that a priori prescribing the optimizer and metamodel types may degrade the search effectiveness. This observation motivates the development of algorithms which adapt the type of metamodel and optimizer to the problem being solved. In this context, Gorissen et al.^[17] proposed an approach in which multiple metamodels were considered, and EA was used to select an optimal type of metamodel during the search. Taking a different approach, the algorithm proposed in this study adapts both the type of optimizer and metamodel during the search by autonomously selecting the best performing ones from a family candidates. Section 3 provides a detailed description of the proposed algorithm.

3 Proposed algorithm

Before providing a detailed description of the proposed algorithm, it is highlighted that it leverages on two main concepts:

1) Adaptation of the type of optimizer and metamodel during search: The algorithm autonomously selects the best performing metamodel and optimizer out of a family of candidates at each iteration.

2) Trust-region (TR) optimization: To ensure convergence to an optimum of the true expensive objective function in the presence of metamodel inaccuracy, the proposed algorithm employs a TR approach which involves a trial-step search followed by the updates of the TR.

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

Step 1. Initialization: The proposed algorithm begins by generating an initial sample of vectors using a Latin hypercube (LH) design of experiments^[32]. This technique is used to ensure that the sample is space-filling, as this improves the metamodel accuracy. 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 without replacement for each variable, and these samples are combined to produce a vector. This procedure is repeated for l times to generate the complete sample, which is then evaluated with the expensive simulation, and the vectors are stored in memory. After this initialization step, the main optimization loop begins.

Step 2. Metamodel selection: As mentioned, the proposed algorithm selects an optimal type of metamodel out of a family of candidates. To accomplish this in a mathematically rigorous manner, it leverages on the statistical model selection theory^[33], and operates as follows. The set of vectors, which have already been evaluated with the true expensive function and stored in memory is split into a training set and a testing set. For each candidate metamodel type, the proposed algorithm trains a metamodel using the training set, tests its accuracy using the testing set, a procedure termed in the literature as cross-validation (CV). In this procedure, the accuracy of each metamodel is estimated using the mean squared error (MSE) of prediction, which is calculated as

$$\text{MSE} = \frac{1}{l} \sum_{i=1}^l (\hat{m}(\mathbf{x}_i) - f(\mathbf{x}_i))^2 \quad (1)$$

where $\mathbf{x}_i$, $i = 1, \dots, l$, are the testing vectors, $\hat{m}(\mathbf{x})$ is a metamodel trained using the training set, and $f(\mathbf{x}_i)$ is the exact and already known function value at $\mathbf{x}_i$. A merit of this approach is that there is no restriction on the type or number of the candidate metamodels which can be considered. As the CV procedure relies on splitting the candidate solutions stored in memory into two sets, numerical experiments have been performed to identify a suitable splitting ratio, and the details are given in Section 4.1. With respect to the metamodel types, the proposed algorithm selects between three candidates: radial basis functions (RBF), radial basis functions network (RBFN) and Kriging, which are described in Appendix A. After the selection phase, the proposed algorithm trains a new metamodel by using all the vectors stored in memory, and this metamodel is then used during the TR trial-step, as described in the following step.

Step 3. TR step and optimizer selection: The best vector stored in memory is taken as the TR centre ($\mathbf{x}_b$), and the

proposed algorithm performs a TR trial-step, i.e., it seeks the optimum of the metamodel in the bounded region

$$g = \{\mathbf{x} : \|\mathbf{x} - \mathbf{x}_b\|_2 \leq \Delta\} \quad (2)$$

where $\mathbf{x}_b$ is the best solution found so far, and Δ is the TR radius.

Next, a procedure is employed to identify the best performing optimizer out of a family of candidates. To accomplish this, in turn, each candidate optimizer is used to seek the optimum of the metamodel trained in the previous step. In this setup, the TR trial-step is repeated with each optimizer. But since obtaining the objective values from each of the metamodels is computationally cheap, the entire procedure requires only a few seconds. After completing this step, the proposed algorithm selects the overall best solution $\mathbf{x}^*$ found, and uses it during the TR trial-step as described below. It also records the best performing optimizer, and uses it in the metamodel improvement step as described below. The proposed algorithm selects between three candidate optimizers: a DE algorithm, an EA, and a PSO, which are described in Appendix B.

Step 4. TR updates: The best solution found in the previous step $\mathbf{x}^*$, is evaluated with the true expensive function, which yields the exact objective value $f(\mathbf{x}^*)$. Following the classical TR framework^[34], the proposed algorithm updates the metamodel and the TR as follows:

1) If $f(\mathbf{x}^*) < f(\mathbf{x}_b)$, the trial step was successful since the predicted optimum is indeed better than the current best solution $\mathbf{x}_b$. Accordingly, the TR is centred at the new optimum, and the TR is enlarged by doubling its radius.

2) If $f(\mathbf{x}^*) \geq f(\mathbf{x}_b)$ and there are sufficient vectors in the TR, the search was unsuccessful since the predicted optimum is not better than the current best vector. However, since there are sufficient vectors in the TR, the metamodel is considered to be sufficiently accurate to justify contracting the TR, and therefore the TR radius is halved.

3) If $f(\mathbf{x}^*) \geq f(\mathbf{x}_b)$ and there are insufficient vectors inside the TR, the search was unsuccessful. But this may be due to poor accuracy of the metamodel, i.e., there are too few vectors in the TR. As such, the algorithm adds new vectors in the TR to improve the metamodel accuracy as explained below.

As a change from the classical TR framework, the proposed algorithm monitors the number of vectors in the TR to avoid contracting it too quickly, which can result in premature convergence^[34]. Numerical experiments have been performed to identify a suitable threshold value q , and the details are given in Section 4.1.

Another change from the classical TR framework is the addition of new vectors to improve the accuracy of the metamodel in the TR, as mentioned above. There are two considerations in selecting these vectors:

1) They should improve the metamodel accuracy locally, around the current optimum (the TR centre).

2) Alternatively, they should improve the metamodel over the entire TR, particularly in regions sparse with vectors^[35].

Since these are typically two opposing considerations, the proposed algorithm generates several new vectors which represent different trade-offs between the above two considerations. The vectors are taken as the minimizers of the

objective function

$$h(\mathbf{x}) = wh_1(\mathbf{x}) + (1 - w)h_2(\mathbf{x}) \quad (3)$$

where the minimization of (3) is performed by the best performing optimizer selected in Step 3.

The variables used in (3) are:

$h_1(\mathbf{x})$ is the rank of a vector based on its objective value. The best vector in the population is assigned a rank of 1, the following one is assigned a rank of 2, etc.

$h_2(\mathbf{x})$ is the rank of a vector based on its distance from the existing vectors in the TR. The vector in the population which is farthest is given a rank of 1, the vector having the second largest distance is given a rank of 2, etc.

The weight w defines the trade-off between the two considerations. The setting $w = 1$ implies that a vector is searched only based on its objective value, while $w = 0$ implies that a vector is searched only based on its distance to existing ones.

Equation (3) uses a rank-based approach to make the search insensitive to the magnitude and sign of the objective function values. Numerical experiments have been performed to identify a suitable set of weights, and the details are given in Section 4.1.

While in this study, the proposed algorithm used specific metamodels (Kriging, radial basis functions, and radial basis functions network) and optimizers (DE, EA, and PSO), other types or numbers of the latter can be readily used. To complete the description of the proposed algorithm, Fig. 2 gives a schematic layout of its optimization iteration, and Algorithm 1 gives its pseudo code.

4 Numerical experiments: Results and discussion

This section analyzes in detail the performance of the proposed algorithm. It begins by describing a parameter sensitivity analysis which was used to identify suitable settings for the algorithm parameters, it then describes benchmark tests based on a set of mathematical test functions, and concludes with another set of benchmark tests based on an engineering application of airfoil shape optimization.

4.1 Parameter sensitivity analysis

As described in Section 3, the proposed algorithm relies on three main parameters:

- 1) q : The minimum number of vectors needed in the TR to invoke a TR contraction.
- 2) w_i : The weights used for generating the new vectors in the metamodel improvement step.
- 3) The split ratio between the training and the testing sets, as used in the CV procedure when estimating the metamodel accuracy.

For a rigorous sensitivity analysis, the following setup was used. A baseline set of parameter values was defined, and for each parameter in question, three candidate values were selected. In turn, only one parameter was changed and assigned a candidate setting, and the proposed algorithm was run for a full optimization search.

Algorithm 1. Proposed optimization algorithm

```

/* Initialization */
Generate an initial LH sample;
Evaluate the sample vectors and store in memory;
/* Main optimization loop */
repeat
    /* Select a metamodel */
    Split the evaluated vectors stored in memory into a
    training set and a testing set;
    for each type of candidate metamodel do
        train a metamodel using the training set;
        Estimate the metamodel prediction error using CV;
        Select the metamodel type having the lowest prediction
        error, and train a new metamodel using all vectors stored
        in memory;
    /* Select and optimizer and perform a TR step */
    Set the TR centre to the best vector stored in memory;
    for each type of candidate optimizer do
        Search for an optimum of the metamodel in the TR;
        Select the overall best optimum found and its corresponding
        optimizer;
    /* Update the TR */
    Evaluate the predicted optimum with the true expensive
    function;
    if the new optimum is better than the TR centre then
        Double the TR radius
    else if the new optimum is not better than the TR centre
    and there are sufficient vectors in the TR then
        Halve 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 new vectors in the TR to improve the metamodel
        accuracy;
    Add to the vectors stored in memory all the new vectors
    evaluated with the true expensive function;
until maximum number of analyses completed.

```

To examine the impact of different parameter settings across a variety of objective functions and function dimensions, for each candidate setting, the algorithm was run with the Rastrigin-10D, Rosenbrock-10D, Rastrigin-20D, and Rosenbrock-20D functions, and with 10 repeats per case. Overall, for each parameter, 120 test runs were performed (3 candidate settings per parameter $\times$ 4 objective

Fig. 2 The layout of an optimization iteration of the proposed algorithm

functions $\times 10$ repetitions per setting). The candidate parameter settings were (baseline settings are underlined):

- 1) q : $0.1d$, $0.5d$, d , where d is the function dimension.
- 2) w : $\{0.8, \underline{0.6}, 0.4, 0.2\}$, $\{0.8, 0.5, 0.2\}$, $\{0.8, 0.2\}$.
- 3) Training-testing split ratio (in percent): $\underline{80-20}$, 50-50, 20-80.

For each candidate setting, Table 2 shows the mean objective value, the rank, and the overall rank, where a lower score is better. From analyzing these test results, it follows:

1) q : Using $q = d$ was the optimal setting for the Rastrigin function, while it was $q = 0.1d$ for the Rosenbrock function. This is attributed to the differences between the function landscapes: Rastrigin is highly multimodal, and therefore, a higher threshold number delays the TR contraction and diminishes the changes of converging to a poor local optimum. In contrast, Rosenbrock is unimodal and therefore a lower threshold number accelerates the TR contraction and enables the algorithm to obtain a better approximation of the single global optimum. Since no single setting was overall optimal, we selected the intermediate setting of $q = 0.5d$.

2) w : The optimal setting was that of using four weights, $w = \{0.8, 0.6, 0.4, 0.2\}$ which corresponds to adding four new vectors in the metamodel improvement step. This is attributed to the weights which result in vectors that improve the metamodel accuracy both near the optimum and over the entire TR.

3) Training-testing split ratio: The 50-50 split ratio was selected as it yielded the best results.

The selected parameter setting, as described above, were used in the benchmark tests described in the following sections.

4.2 Benchmark tests based on mathematical test functions

The performance of the proposed algorithm was evaluated using the set of established test functions (Ackley, Griewank, Rastrigin, Rosenbrock, Schwefel 2.13, and Weierstrass) suggested by [36], and in dimension 10 to 40. The functions details are given in Table 3.

For a thorough evaluation, the proposed algorithm was benchmarked against four reference algorithms:

1) V1: Variant 1 of the proposed algorithm, which is identical in operation, except that it used a single metamodel (RBF) and a single optimizer (DE).

2) V2: Variant 2 of the proposed algorithm, which was identical to the V1 variant in operation, except that it used a single metamodel (RBFN) and a single optimizer (EA).

3) Metamodel-assisted EA with periodic sampling (EA-PS)^[5]: The algorithm combines a Kriging metamodel and an EA. It safeguards the metamodel accuracy by periodically evaluating a small subset of the population with the true objective function and then incorporating the latter into the metamodel.

4) Expected-improvement with a metamodel-assisted CMA-ES (EI-CMA-ES)^[37]: The algorithm combines a covariance matrix adaptation evolutionary strategy (CMA-ES) optimizer with a Kriging metamodel, and uses the expected improvement framework to update the metamodel.

Full details of the latter two reference algorithms are given in Appendix C. This benchmark layout was employed

to:

1) Study the contribution of selecting the metamodel and optimizer during the search, by comparing the proposed algorithm to the V1 and V2 variants which did not employ such a selection.

2) Compare the proposed algorithm to the representative metamodel-assisted algorithms from the literature.

Every algorithm-objective function combination was repeated for 30 times to support a valid statistical analysis.

Table 4 gives the resultant test statistics of mean, standard deviation (SD), median, minimum (best) and maximum (worst) objective value for each algorithm-objective function combination, where the test functions were Ackley and Griewank in dimension 10, Rastrigin and Rosenbrock in dimension 20, and Schwefel 2.13 and Weierstrass in dimension 40. Also, the statistic α shows the significance-level at which the performance of the proposed algorithm was better than those of the other algorithms. An empty entry indicates that there was no statistically significant performance advantage at the 0.05 level. The α statistic was obtained using the Mann-Whitney nonparametric test^[38].

Table 2 Results for the parameter sensitivity analysis

(a) Mean statistic as a function of q (number of TR vectors to enable contraction)

Function	$q = 0.1d$		$q = 0.5d$		$q = d$	
	Mean	Rank	Mean	Rank	Mean	Rank
Ras-10	5.245E+01	02	5.514E+01	03	3.852E+01	01
Ros-10	4.524E+01	02	3.131E+01	01	8.856E+01	03
Ras-20	7.176E+01	02	1.174E+02	03	5.838E+01	01
Ros-20	4.038E+02	02	2.384E+02	01	4.928E+02	03
Overall		08		08		08

d : Dimension of the objective function.

(b) Mean statistic as a function of w_i (weights used for generating vectors for metamodel improvement)

Function	$w = \{0.8, 0.6, 0.4, 0.2\}$		$w = \{0.8, 0.5, 0.2\}$		$w = \{0.8, 0.2\}$	
	Mean	Rank	Mean	Rank	Mean	Rank
Ras-10	5.245E+01	03	4.180E+01	01	4.539E+01	02
Ros-10	4.524E+01	01	4.600E+01	02	5.324E+01	03
Ras-20	7.176E+01	01	9.213E+01	03	8.981E+01	02
Ros-20	4.038E+02	02	4.431E+02	03	2.534E+02	01
Overall		07		09		08

(c) Mean statistic as a function of the training-testing split ratio

Function	80-20		50-50		20-80	
	Mean	Rank	Mean	Rank	cMean	Rank
Ras-10	5.245E+01	03	3.683E+01	01	4.429E+01	02
Ros-10	4.524E+01	01	6.088E+01	02	7.405E+01	03
Ras-20	7.176E+01	02	8.319E+01	03	6.931E+01	01
Ros-20	4.038E+02	02	3.752E+02	01	5.266E+02	03
Overall		08		07		09

Table 3 Mathematical test functions

Function	Definition	Domain
Ackley	$f(x) = -20 \exp(-0.2 \sqrt{\frac{1}{d} \sum_{i=1}^d x_i^2}) - \exp(\sum_{i=1}^d \cos \frac{2\pi x_i}{d}) + 20 + e$	$[-32, 32]^d$
Griewank	$f(x) = \sum_{i=1}^d \frac{x_i^2}{4000} - \prod_{i=1}^d \cos\left(\frac{x_i}{\sqrt{i}}\right) + 1$	$[-100, 100]^d$
Rastrigin	$f(x) = \sum_{i=1}^d \{x_i^2 - 10 \cos(2\pi x_i) + 10\}$	$[-5, 5]^d$
Rosenbrock	$f(x) = \sum_{i=1}^{d-1} \{100(x_i^2 - x_{i+1})^2 + (x_i - 1)^2\}$	$[-10, 10]^d$
Schwefel 2.13	$f(x) = \sum_{i=1}^d \sum_{j=1}^d [(a_{i,j} \sin(\alpha_j) + b_{i,j} \cos(\alpha_j)) - (a_{i,j} \sin(x_j) + b_{i,j} \cos(x_j))]^2$	$[-\pi, \pi]^d$
Weierstrass	$f(x) = \sum_{i=1}^d \sum_{k=0}^{20} 0.5^k \cos(2\pi 3^k (x_i + 0.5)) - df(x) = \sum_{k=0}^{20} 0.5^k \cos(\pi 3^k)$	$[-0.5, 0.5]^d$

Test results show that the proposed algorithm performed well. With the Ackley-10D function, it obtained the best mean and had the second best median, closely following the V1 algorithm. Its SD was the third best after the EA-PS and EI-CMA-ES algorithms, which indicates that its performance was stable. Its performance had a statistically significant advantage over that of the V2, EA-PS, and EI-CMA-ES algorithms at 0.05 level or better.

A similar pattern is observed with the Griewank-10D function, in which the proposed algorithm obtained both the best mean and the best median statistics. Its SD was the fourth best, indicating it had a higher variability in its performance when compared to the other algorithms. Its performance had a statistically significant advantage over that of the V2, EA-PS and EI-CMA-ES algorithms at 0.01 level, which indicates solid performance gains.

With the Rastrigin-20D function, the proposed algorithm maintained its performance advantage, and obtained the best mean and median statistics. Its SD statistic was the second best after the EA-PS algorithm, which indicates that its performance was stable. Lastly, its performance had a statistically significant advantage over that of the other four algorithms at the 0.01 level, again indicating solid performance gains.

With the Rosenbrock-20D function, a similar trend is observed, and the proposed algorithm obtained the best mean and median statistics while its SD was the second best after the EA-PS algorithm. These results are consistent with those observed with the Rastrigin-20D function, i.e., that the proposed algorithm obtained both a good final result and that its performance was stable. Its performance had a statistically significant advantage over that of the V2, EA-PS, and EI-CMA-ES algorithms at the 0.01 level.

For the Schwefel 2.13-40D function, the proposed algorithm obtained both the best mean and median statistics, while its SD was the second best behind the EA-PS algorithm. Its performance had a statistically significant advantage over the other four algorithms at the 0.01 level.

Finally, for the Weierstrass-40D function, the proposed

algorithm again obtained the best mean and median statistics, while its SD was the fourth best, which indicates that its performance had a higher variability than that of the V1, V2, and EA-PS algorithms. Its performance had a statistically significant advantage over the V1 and V2 algorithms at the 0.01 level.

Overall, test results show that the proposed algorithm achieved the best mean and median statistics in all cases except for the Ackley-10D, where it achieved the best mean and the second best median. Analysis of the SD statistic shows that the proposed algorithm performance was typically stable, as it obtained the best SD statistic with the Schwefel 2.13-40D function, the second best with the Rastrigin-20D and Rosenbrock-20D, and the fourth best with the Ackley-10D function.

Analysis also shows that the proposed algorithm had a statistically significant performance advantage over the reference algorithms in 19 out of 24 comparisons (it was compared to 4 reference algorithms across 6 test functions), which indicates that it was the best performing algorithm overall.

Another performance aspect which was studied was the pattern of metamodel and optimizer updates, in order to understand if the latter two were updated frequently or if a single type of metamodel or optimizer dominated the search. Fig. 3 shows representative plots of the metamodel and optimizer updates from a test run with functions Ackley-10D and Rosenbrock-20D. It follows that both were updated frequently during the search, which indicates that no single type was overall optimal, and that the optimal type varied as the search progressed. These results also highlight the demerit of using a fixed type of metamodel and optimizer, as this can result in an unsuitable configuration which can degrade the search effectiveness. This observation is also supported by the test results, where the proposed algorithm consistently outperformed algorithms V1 and V2 which were identical in operation to the proposed algorithm, but employed a single type of metamodel and optimizer.

Table 4 Test statistics for the mathematical test functions
(a) Ackley-10D

	Proposed	V1	V2	EA-PS	EI-CMA-ES
Mean	3.085E+00	5.286E+00	4.699E+00	5.241E+00	1.796E+01
SD	1.681E+00	7.091E+00	1.687E+00	5.590E-01	1.529E+00
Median	2.880E+00	2.269E+00	4.718E+00	5.408E+00	1.797E+01
Min(best)	5.919E-01	7.488E-01	2.250E+00	4.098E+00	1.443E+01
Max(worst)	6.125E+00	1.889E+01	8.075E+00	6.010E+00	1.988E+01
α			0.05	0.01	0.01

(b) Griewank-10D

	Proposed	V1	V2	EA-PS	EI-CMA-ES
Mean	2.235E-01	2.507E-01	8.642E-01	9.579E-01	9.338E-01
SD	3.098E-01	2.163E-01	4.569E-02	1.076E-01	2.435E-01
Median	9.341E-02	1.539E-01	8.603E-01	9.862E-01	1.007E+00
Min(best)	1.660E-02	5.273E-02	7.996E-01	7.146E-01	2.441E-01
Max(worst)	8.087E-01	6.517E-01	9.262E-01	1.046E+00	1.050E+00
α			0.01	0.01	0.01

(c) Rastrigin-20D

	Proposed	V1	V2	EA-PS	EI-CMA-ES
Mean	5.543E+01	1.049E+02	1.824E+02	1.223E+02	2.105E+02
SD	1.937E+01	2.875E+01	2.141E+01	1.219E+01	3.914E+01
Median	5.385E+01	1.044E+02	1.809E+02	1.230E+02	2.296E+02
Min(best)	2.864E+01	6.983E+01	1.557E+02	1.046E+02	1.395E+02
Max(worst)	8.720E+01	1.660E+02	2.284E+02	1.429E+02	2.507E+02
α		0.01	0.01	0.01	0.01

(d) Rosenbrock-20D

	Proposed	V1	V2	EA-PS	EI-CMA-ES
Mean	5.593E+02	9.600E+02	2.278E+04	8.435E+02	3.967E+03
SD	3.224E+02	8.066E+02	1.192E+04	3.012E+02	9.406E+02
Median	4.569E+02	5.861E+02	2.318E+04	7.782E+02	3.685E+03
Min(best)	2.547E+02	2.028E+02	3.242E+03	4.676E+02	3.141E+03
Max(worst)	1.366E+03	2.278E+03	4.496E+04	1.439E+03	6.144E+03
α			0.01	0.01	0.01

(e) Schwefel 2.13-40D

	Proposed	V1	V2	EA-PS	EI-CMA-ES
Mean	5.668E+05	1.116E+06	2.723E+06	1.774E+06	1.667E+06
SD	2.116E+05	5.734E+05	5.315E+05	2.509E+05	6.520E+05
Median	5.022E+05	8.565E+05	2.581E+06	1.744E+06	1.528E+06
Min(best)	3.432E+05	6.562E+05	1.971E+06	1.415E+06	8.933E+05
Max(worst)	1.012E+06	2.530E+06	3.754E+06	2.104E+06	2.871E+06
α		0.01	0.01	0.01	0.01

(f) Weierstrass-40D

	Proposed	V1	V2	EA-PS	EI-CMA-ES
Mean	3.641E+01	4.482E+01	5.447E+01	2.343E+01	3.598E+01
SD	4.428E+00	4.152E+00	2.157E+00	1.265E+00	1.463E+01
Median	3.587E+01	4.556E+01	5.407E+01	2.304E+01	2.597E+01
Min(best)	2.985E+01	3.569E+01	5.083E+01	2.214E+01	2.100E+01
Max(worst)	4.274E+01	4.906E+01	5.784E+01	2.567E+01	5.817E+01
α		0.01	0.01

Fig. 3 Examples of variation of the selected metamodel type and optimizer type during the search

To compare the convergence trends by each of the five benchmarked algorithms, Fig. 4 shows the representative convergence plots with functions Ackley-10D and Rosenbrock-20D. The plots show that the proposed algorithm achieved either the best or near-best convergence rate, while it still provided the overall best final result. The contribution of adapting the metamodel and optimizer is evident from the comparison to algorithms V1 and V2, which had a slower convergence rate and achieved an inferior final result. Plots also show that the proposed algorithm outperformed the two reference algorithms from the literature, which follows the trends observed in the test statistics (Table 4).

4.3 Engineering test problem

For a comprehensive evaluation, the proposed algorithm was also tested with an engineering application of airfoil shape optimization which is representative of a real-world simulation-driven design application, and is formulated as follows. During flight, an aircraft generates lift, i.e., the force which counters the aircraft weight and keeps it airborne, and drag which is an aerodynamic friction force that obstructs the aircrafts movement. Both the lift and drag are strongly determined by the wing cross-section, i.e., the airfoil. The optimization goal is then to find an airfoil shape which maximizes the lift and minimizes the drag.

In practise, the design requirements for airfoils are specified in terms of the nondimensional lift and drag coefficients, c_l and c_d , respectively, defined as

$$c_l = \frac{L}{\frac{1}{2}\rho V^2 S} \quad (4a)$$

$$c_d = \frac{D}{\frac{1}{2}\rho V^2 S} \quad (4b)$$

Fig. 4 Representative convergence plots for the benchmarked algorithms

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

Candidate airfoils were represented using the Hicks-Henne parameterization^[39], which defines the profile of a candidate airfoil as

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

where y_b is a baseline airfoil profile, taken as the NACA0012 symmetric airfoil, b_i is a basis function, which is defined as^[40]

$$b_i(x) = \left[\sin \left(\pi x^{\frac{\log(0.5)}{\log \left(\frac{i}{h+1} \right)}} \right) \right]^4 \quad (6)$$

and $\alpha_i \in [-0.01, 0.01]$ are coefficients, which are the problem's design variables. 10 basis functions were used for the upper and lower airfoil profile, respectively, resulting in

Table 5 Test statistics for the airfoil problem

	Proposed	V1	V2	EA-PS	EI-CMA-ES
Mean	-3.310E+00	-3.296E+00	-3.308E+00	-3.231E+00	-3.278E+00
SD	9.746E-02	7.399E-02	6.027E-02	7.164E-02	9.597E-02
Median	-3.326E+00	-3.277E+00	-3.315E+00	-3.227E+00	-3.267E+00
Min(best)	-3.422E+00	-3.420E+00	-3.391E+00	-3.335E+00	-3.395E+00
Max(worst)	-3.156E+00	-3.200E+00	-3.217E+00	-3.134E+00	-3.098E+00
α				0.05	

a problem consisting of 20 design variables. Fig. 5 summarizes the nomenclature of the physical quantities and airfoil parametrization involved in this test problem. The lift and drag coefficients of candidate airfoils were obtained using XFOil, a computational fluid dynamics simulation for analysis of airfoils operating in the subsonic regime Drela^[41]. Each airfoil evaluation required up to 30 s on a desktop computer. To ensure structural integrity, between 0.2 to 0.8 of the chord line, the airfoil thickness (t) had to be equal to or larger than a critical value $t^* = 0.1$. The flight conditions were set as a cruise altitude of 30 kft (30000 feet), a cruise speed of Mach 0.7, i.e., 70 % of the speed of sound, and an AOA of 2° . As mentioned, the optimization goal was to find an airfoil shape which maximizes the lift coefficient (c_l) and minimizes the aerodynamic (c_d) at the prescribed flight altitude, speed, and AOA. Also, to ensure structural integrity, between 0.2 to 0.8 of the chord length, the airfoil's thickness (t) had to be equal to or larger than a critical value $t^* = 0.1$.

Fig. 5 Schematic layout of the airfoil optimization problem

To facilitate the metamodel-assisted optimization, i.e., to account both for the objective function and constraint using a single metamodel, the thickness constraint was incorporated into the objective function, which resulted in the following augmented objective function

$$f = -\frac{c_l}{c_d} + p \quad (7a)$$

where p is the penalty for airfoils which violate the thickness constraint, and was defined as

$$p = \begin{cases} \frac{t^*}{t} \times \left\| \frac{c_l}{c_d} \right\|, & \text{if } t < t^* \\ 0, & \text{otherwise.} \end{cases} \quad (7b)$$

The benchmark tests followed the setup described in Section 4.2, and Table 5 gives the resultant test statistics. It follows that the proposed algorithm outperformed the other candidate algorithms also in this test problem, which shows that it also performed well in a representative simulation-driven test problem. It obtained the best mean and median statistics, and had the highest SD. Its performance had a statistically significant advantage over that of the EA-PS algorithm at the 0.05 level.

Following Section 4.2, Fig. 6 shows the metamodel and optimizer updates from one test run. They follow the trend observed earlier: the metamodel and optimizer types were updated frequently during the search, and no single type was dominant. In this test run, the RBFN metamodel was used much less than the RBF and Kriging ones.

Fig. 6 Updates of the metamodel and optimizer type during one run with the airfoil problem

Fig. 7 gives representative convergence plots for the five algorithms, showing that the proposed algorithm had the fastest convergence rate and obtained the overall best final result. The V1 and V2 variants followed closely but obtained inferior final results, which again highlights the contribution of adapting the metamodel and optimizer types during the search. The EA-PS algorithm did not improve on its initial best solution, while the EI-CMA-ES algorithm obtained a result inferior to that of the proposed algorithm and its two variants.

5 Conclusions

The modern engineering design optimization process often replaces laboratory experiments with computer simulations, which results in expensive black-box optimization problems. Such problems introduce several new challenges into the optimization process, e.g., a lack of gradient information, a high computational cost for evaluating candidate solutions, and a complicated nonconvex objective landscape.

Fig. 7 Representative convergence plots for the five algorithms in the airfoil problem

Metamodel-assisted CI optimization algorithms have shown to be effective in such challenging problems, but a variety of metamodels and optimizers have been proposed, which introduces the problem of selecting a combination suitable to the problem being solved. The type of metamodel and optimizer being used directly impacts the effectiveness of the optimization search, and a priori prescribing their type may result in a setup which is unsuitable to the problem being solved and degrades the search effectiveness. Furthermore, the high cost of evaluating candidate solutions makes it impractical to experiment with different metamodels and optimizers by performing multiple optimization runs.

Leveraging on this issue, this study has proposed a new metamodel-assisted CI algorithm which autonomously adapts the type of metamodel and optimizer during the search, by selecting the most suitable ones out of a family of candidates.

In an extensive evaluation, the proposed algorithm was benchmarked against two metamodel-assisted algorithms from the literature and two variants without metamodel or optimizer selection. Analysis shows that overall the proposed algorithm outperformed the reference algorithms as it consistently achieved the best final result. Also, the type of metamodel and optimizer were updated frequently during the search, which highlights the merit of the proposed approach.

Appendix

A. Candidate metmodels

As mentioned in Section 2, a metamodel serves as a computationally-cheaper mathematical approximation of the true expensive objective function, and it is typically an interpolant trained using previously evaluated vectors. In this study, the proposed algorithm selected between three metamodel variants^[3, 4].

1) Kriging: The metamodel takes a statistical approach to interpolation by combining two components: a drift function which is a global coarse approximation of the true function and a local correction based on the correlation between the interpolation vectors. Given a set of evaluated vectors, $\mathbf{x}_i \in \mathbf{R}^d$, $i = 1, \dots, n$, the Kriging metamodel is trained

such that it exactly interpolates the observed values, i.e., $m(\mathbf{x}_i) = f(\mathbf{x}_i)$, where $m(\mathbf{x})$ and $f(\mathbf{x})$ are the metamodel and true objective function, respectively. Using a constant drift function^[42] gives the Kriging metamodel

$$m(\mathbf{x}) = \beta + \kappa(\mathbf{x}) \quad (8)$$

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

$$\text{Cov}[\kappa(\mathbf{x})\kappa(\mathbf{y})] = \sigma^2 c(\theta, \mathbf{x}, \mathbf{y}) \quad (9)$$

where $c(\theta, \mathbf{x}, \mathbf{y})$ is a user-prescribed correlation function. A common choice for the correlation is the Gaussian correlation function^[42], defined as

$$c(\theta, \mathbf{x}, \mathbf{y}) = \prod_{i=1}^d \exp(-\theta(x_i - y_i)^2). \quad (10)$$

Combining the latter expression with the constant drift function transforms the metamodel from (8) into the following form

$$m(\mathbf{x}) = \hat{\beta} + \mathbf{r}(\mathbf{x})^T \mathbf{R}^{-1}(\mathbf{f} - \mathbf{1}\hat{\beta}) \quad (11)$$

where $\hat{\beta}$ 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. $\mathbf{r}^T$ is the correlation vector between a new vector $\mathbf{x}$ and the sample vectors, i.e.,

$$\mathbf{r}^T = [c(\theta, \mathbf{x}, \mathbf{x}_1), \dots, c(\theta, \mathbf{x}, \mathbf{x}_n)]. \quad (12)$$

The estimated drift coefficient $\hat{\beta}$ and variance $\hat{\sigma}^2$, which are required in (11), are obtained as

$$\hat{\beta} = (\mathbf{1}^T \mathbf{R}^{-1} \mathbf{1})^{-1} \mathbf{1}^T \mathbf{R}^{-1} \mathbf{f} \quad (13a)$$

$$\hat{\sigma}^2 = \frac{1}{n} [(\mathbf{f} - \mathbf{1}\hat{\beta})^T \mathbf{R}^{-1} (\mathbf{f} - \mathbf{1}\hat{\beta})]. \quad (13b)$$

Fully defining the metamodel requires the correlation parameters θ , which are commonly taken as the maximizers of the metamodel likelihood. This is achieved by minimizing the expression

$$\psi(\theta) = \|\mathbf{R}\|^{\frac{1}{n}} \hat{\sigma}^2 \quad (14)$$

which is a function only of the correlation parameters θ and the sample data. While a different correlation parameter can be used for each variable, the metamodel used in this study, as is commonly done in literature, employs a single correlation parameter. This results in a univariate likelihood function, which is relatively easy to optimize.

2) Radial basis functions: The metamodel approximates the objective function by a superposition of basis functions of the form

$$\phi_i(\mathbf{x}) = \phi(\|\mathbf{x} - \mathbf{x}_i\|_2) \quad (15)$$

where $\mathbf{x}_i$ is a sampled vector. Given the sample vectors $\mathbf{x}_i \in \mathbf{R}^d$, $i = 1, \dots, n$, and the corresponding objective values $f(\mathbf{x}_i)$, the radial basis function metamodel is then given by

$$m(\mathbf{x}) = \alpha_i \sum_{i=1}^n \phi_i(\mathbf{x}) + c \quad (16)$$

where α_i and c are coefficients determined from the interpolation conditions

$$m(\mathbf{x}_i) = f(\mathbf{x}_i), i = 1, \dots, n \quad (17a)$$

$$\sum_{i=1}^n \alpha_i = 0. \quad (17b)$$

A common choice is the Gaussian basis function^[43], defined as

$$\phi_i(\mathbf{x}) = \exp\left(-\frac{\|\mathbf{x} - \mathbf{x}_i\|^2}{\tau}\right) \quad (18)$$

where τ controls the width of the Gaussians and is determined by cross-validation^[4, 44], as explained below. Mathematically, the metamodel training process involves the solution of the linear equation system

$$\begin{pmatrix} \Phi & \mathbf{1} \\ \mathbf{1}^T & 0 \end{pmatrix} \boldsymbol{\alpha} = \begin{bmatrix} \mathbf{f} \\ 0 \end{bmatrix} \quad (19)$$

where Φ is the Gram matrix such that $\Phi_{i,j} = \phi_i(\mathbf{x}_j)$, $\mathbf{1}$ is a vector whose elements are all one, and $\mathbf{f}$ is the vector of objective values of the sample vectors. The linear system (19) is often ill-conditioned, and is therefore solved using the singular value decomposition (SVD) procedure^[4].

A main aspect of the training process of the radial basis function metamodel is the identification of an optimal width coefficient (τ) for the Gaussian basis functions. An established procedure to achieve this is by generating a set of candidate values, estimating the accuracy of the radial basis function metamodel with each candidate value using a CV procedure, and selecting the metamodel deemed as the most accurate^[4, 44].

3) RBFN: The metamodel aims to mimic the operation of a biological neural network, and uses radial basis functions as its neurons or processing units. In contrast to the baseline radial basis function metamodel, the radial basis functions network typically uses fewer basis functions than sample vectors, which is done to improve the generalization ability and prediction accuracy. Given a set of sample vectors, $\mathbf{x}_i, i = 1, \dots, n$, and their respective objective values $f(\mathbf{x}_i)$, the radial basis functions network metamodel is given by

$$m(\mathbf{x}) = \sum_{i=1}^{\hat{n}} \alpha_i \phi_i(\mathbf{x}) \quad (20)$$

where the processing units $\phi_i(\mathbf{x})$ are

$$\phi_i(\mathbf{x}) = \exp\left(-\frac{\|\mathbf{x} - \hat{\mathbf{x}}_i\|^2}{\tau}\right) \quad (21)$$

and $\hat{n} \leq n$. The neurons' centres $\hat{\mathbf{x}}_i, i = 1, \dots, \hat{n}$, need to be determined such that they provide a good representation of the sample vectors. An established approach to achieve this is to determine them by clustering, and often with the k -means clustering algorithm to determine these centres^[45]. The parameter τ , which determines the width of the Gaussian functions of the neurons, is selected by cross-validation CV^[4, 44]. The coefficients α_i are determined from the least-squares solution

$$\Phi^T \Phi \boldsymbol{\alpha} = \Phi^T \mathbf{f} \quad (22)$$

where

$$\Phi_{i,j} = \phi_j(\mathbf{x}_i) \quad (23)$$

$\mathbf{x}_i, i = 1, \dots, n$, are the sample vectors, and $\mathbf{f}$ is the vector of corresponding objective function values. As mentioned in Section 3, the proposed algorithm used an radial basis functions network with $\hat{n} = 0.8n$, i.e., the number of neurons was 80 % of the sample size.

B. Candidate optimizers

Computational intelligence and nature-inspired optimizers employ a population of candidate solutions and manipulate them at each iteration to produce a new set of candidate solutions. This manipulation is performed with operators which are often inspired by phenomena in biology and physics. In this study, the proposed algorithm selects between three established CI optimizers which represent different optimization paradigms:

1) Differential evolution^[46]: The optimizer employs a population of candidate solutions. At each iteration, it selects a candidate solution $\mathbf{x}$ from the population at random, and generates a new solution by probabilistic rules governing how to combine components both from the original candidate solution and other population members. In this study, a DE variant known as DE/rand/1/bin was used, which operates as follows. A new candidate solution $\mathbf{y}$ is generated to replace $\mathbf{x}$ using the following component-wise rule:

$$y_i = \begin{cases} a_i + w(b_i - c_i), & \text{if } r_i \leq p \\ x_i, & \text{otherwise} \end{cases} \quad (24)$$

where $\mathbf{a}$, $\mathbf{b}$, and $\mathbf{c}$ are three distinct candidate solutions selected from the population at random, w is a user-prescribed weight, r_i is a random number drawn from a uniform probability distribution $U[0, 1]$, and p is the user-prescribed crossover probability. The DE parameters were set based on the recommended settings^[47] as: a population size of 18 candidate solutions, a crossover probability of $p = 0.502$, and a differential weight of $w = 0.671$.

2) Evolutionary algorithm : The algorithm uses a population of candidate solutions. At each iteration, it applies a series of operators to generate the population of the next iteration. A typical sequence of operators is: a) selection: the best candidate solutions in the population are selected as parents, b) recombination: parents are selected at random and their vectors are combined to produce offspring, c) mutation: a few offsprings are selected at random and some of their design variables are assigned new values. In this study, the real-coded EA of [48] was used, which employs the stochastic universal selection operator, intermediate recombination with a probability of $p = 0.7$, and the breeder genetic algorithm mutation operator with probability $p = 0.1$.

3) Particle swarm optimizer^[49]: The algorithm uses a swarm of candidate solutions, termed particles, each having its own velocity. Based on the function values of the particles, the algorithm updates their velocities and positions, such that the swarm is driven to explore more promising regions of the search space. At iteration t , for a swarm

particle at position $\mathbf{x}^{(t)}$ travelling at a velocity $\mathbf{v}^{(t)}$, its new velocity is obtained using the rule

$$\mathbf{v}^{(t+1)} = \omega \mathbf{v}^{(t)} + \phi_p r_p (\mathbf{p} - \mathbf{x}^{(t)}) + \phi_g r_g (\mathbf{g} - \mathbf{x}^{(t)}) \quad (25)$$

where ω is a user-defined parameter termed the inertia weight, and ϕ_p and ϕ_g are the user-defined attraction weights towards the particle's best known position $\mathbf{p}$ and swarm best position $\mathbf{g}$. r_p and r_g are weights taken as random numbers drawn from the uniform distribution $U[0, 1]$. After obtaining $\mathbf{v}^{(t+1)}$, the particle's new position is updated using the rule

$$\mathbf{x}^{(t+1)} = \mathbf{x}^{(t)} + \mathbf{v}^{(t+1)} \quad (26)$$

The PSO parameters were set based on the recommended settings in [50]: a swarm size of 150 particles, $\omega = -0.324$, $\phi_p = -0.114$, and $\phi_g = 3.979$.

C. Reference algorithms

As mentioned in Section 4.2, the benchmark tests included the following two reference algorithms from literature:

1) Metamodel-assisted with EA-PS^[5]: The algorithm combines a Kriging metamodel and an EA, and safeguards the metamodel accuracy by periodically evaluating a small subset of the population with the true objective function, and using them to update the metamodel. The algorithm begins by generating candidate solutions using an LH design, evaluating them with the expensive function, and training an initial Kriging metamodel. A real-coded EA then runs for 10 generations while evaluating only the metamodel. Next, the algorithm evaluates the best ten candidate solutions in the population with the true expensive function, and incorporates them into the metamodel. The process repeats until the maximum number of expensive function evaluations is reached. In the tests, the EA was identical to the one used by the proposed algorithm, and which was described in Appendix B.

2) EI-CMA-ES^[37]: The algorithm combines a CMA-ES optimizer^[51] with a Kriging metamodel, and updates the latter based on the expected improvement framework^[24]. The algorithm begins by generating an initial sample of vectors, and evaluates them with the true function. Its main loop then begins, where at each generation, it trains a local Kriging metamodel using both the recently evaluated vectors, and the vectors stored in memory which are nearest to the best solution. A CMA-ES algorithm then searches for an optimum of the metamodel in a bounded region defined by the training sample. In the spirit of the expected improvement framework^[24], it uses the merit function

$$\hat{f}(\mathbf{x}) = m(\mathbf{x}) - \rho \zeta(\mathbf{x}) \quad (27)$$

where $m(\mathbf{x})$ is the metamodel prediction, ρ is a prescribed coefficient, and $\zeta(\mathbf{x})$ is the estimated Kriging prediction error, which is zero at sampled vectors since 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 are stored in memory. If no new vectors were evaluated, e.g., they match vectors which have already been

stored in memory, the algorithm generates a new vector by randomly changing the current best one. Following [37], the algorithm used a training set of 100 vectors comprising of the 50 most recently evaluated ones and 50 nearest-neighbours, and CMA-ES used the default values in [51].

References

- [1] Y. Tenne, C. K. Goh, Eds. Computational Intelligence in Expensive Optimization Problems. *Adaption, Learning, and Optimization*, vol. 2, New York: Springer, 2010.
- [2] Z. Michalewicz. *Genetic Algorithms + Data Structures = Evolution Programs*, Berlin: Springer-Verlag, 1998.
- [3] N. V. Queipo, R. T. Haftka, W. Shyy, T. Goel, R. Vaidyanathan, P. K. Tucker. Surrogate-based analysis and optimization. *Progress in Aerospace Sciences*, vol. 41, no. 1, pp. 1–28, 2005.
- [4] A. I. J. Forrester, A. J. Keane. Recent advances in surrogate-based optimization. *Progress in Aerospace Science*, vol. 45, no. 1–3, pp. 50–79, 2008.
- [5] A. Ratle. Optimal sampling strategies for learning a fitness model. In *Proceedings of the Congress on Evolutionary Computation (CEC)*, IEEE, Piscataway, New Jersey, pp. 2078–2085, 1999.
- [6] A. J. Booker, J. E. Dennis, P. D. Frank, D. B. Serafini, V. Torczon, M. W. Trosset. A rigorous framework for optimization of expensive functions by surrogates. *Structural Optimization*, vol. 17, no. 1, pp. 1–13, 1999.
- [7] M. Emmerich, A. Giotis, M. Özdemir, T. Bäck, K. Gianakoglou. Metamodel-assisted evolution strategies. In *Proceedings of the 7th International Conference on Parallel Problem Solving from Nature*, Springer, Berlin, Germany: pp. 361–370, 2002.
- [8] K. H. Liang, X. Yao, C. Newton. Evolutionary search of approximated N -dimensional landscapes. *International Journal of Knowledge-Based Intelligent Engineering Systems*, vol. 4, no. 3, pp. 172–183, 2000.
- [9] C. Poloni, A. Giurgevich, L. Onseti, V. Pediroda. Hybridization of a multi-objective genetic algorithm, a neural network and a classical optimizer for a complex design problem in fluid dynamics. *Computer Methods in Applied Mechanics and Engineering*, vol. 186, no. 2–4, pp. 403–420, 2000.
- [10] F. Muyl, L. Dumas, V. Herbert. Hybrid method for aerodynamic shape optimization in automotive industry. *Computers and Fluids*, vol. 33, no. 5–6, pp. 849–858, 2004.
- [11] T. Goel, R. T. Haftka, W. Shyy, N. V. Queipo. Ensemble of surrogates. *Structural and Multidisciplinary Optimization*, vol. 33, no. 3, pp. 199–216, 2007.
- [12] Y. C. Jin, B. Sendhoff. Reducing fitness evaluations using clustering techniques and neural network ensembles. In *Proceedings of the Genetic and Evolutionary Computation Conference*, Springer, Berlin, Germany, pp. 688–699, 2004.

- [13] D. Lim, Y. S. Ong, Y. C. Jin, B. Sendhoff. A study on metamodeling techniques, ensembles, and multi-surrogates in evolutionary computation. In *Proceedings of the 9th Annual Conference on Genetic and Evolutionary Computation*, New York, USA: ACM Press, pp. 1288–1295, 2007.
- [14] E. Acar, M. Rais-Rohani. Ensemble of metamodels with optimized weight factors. *Structural and Multidisciplinary Optimization*, vol. 37, no. 3, pp. 279–294, 2009.
- [15] E. Acar. Various approaches for constructing an ensemble of metamodels using local measures. *Structural and Multidisciplinary Optimization*, vol. 42, no. 6, pp. 879–896, 2010.
- [16] X. J. Zhou, Y. Z. Ma, Z. Q. Cheng, L. P. Liu, J. J. Wang. Ensemble of metamodels with recursive arithmetic average. In *Proceedings of the 2nd International Conference on Industrial Mechatronics and Automation*, IEEE, Wuhan, China, pp. 178–182, 2010.
- [17] D. Gorissen, T. Dhane, F. De Turck. Evolutionary model type selection for global surrogate modeling. *The Journal of Machine Learning Research*, vol. 10, pp. 2039–2078, 2009.
- [18] Y. Tenne, S. W. Armfield. A framework for memetic optimization using variable global and local surrogate models. *Soft Computing*, vol. 13, no. 8–9, pp. 781–793, 2009.
- [19] D. Gorissen, I. Coukuyt, P. Demeester, T. Dhane, K. Crombecq. A surrogate modeling and adaptive sampling toolbox for computer based design. *Machine Learning*, vol. 11, pp. 2051–2055, 2010.
- [20] J. F. Rodríguez, J. E. Renaud, L. T. Watson. Convergence of trust region augmented Lagrangian methods using variable fidelity approximation data. *Structural Optimization*, vol. 15, no. 3–4, pp. 141–156, 1998.
- [21] N. M. Alexandrov, R. M. Lewis, C. R. Gumbert, L. L. Green, P. A. Newman. Optimization with variable-fidelity models applied to wing design, Technical report NASA/CR-1999-209826, ICASE Report No. 99–49, Langley Research Center, Hampton, Virginia, 2000.
- [22] M. Sefrioui, J. Périaux. A hierarchical genetic algorithm using multiple models for optimization. In *Proceedings of the 6th International Conference on Parallel Problem Solving from Nature, number 1917 in Lecture Notes in Computer Science*, Springer-Verlag, New York, pp. 879–888, 2000.
- [23] D. Lim, Y. S. Ong, Y. C. Jin, B. Sendhoff. Evolutionary optimization with dynamic fidelity computational models. *Advanced Intelligent Computing Theories and Applications* Springer, Berlin, pp. 235–242, 2008.
- [24] D. R. Jones, M. Schonlau, W. J. Welch. Efficient global optimization of expensive black-box functions. *Journal of Global Optimization*, vol. 13, no. 4, pp. 455–492, 1998.
- [25] I. C. Karpolis, A. S. Zymaris, V. G. Asouti, K. C. Giannakoglou. Multilevel optimization strategies based on metamodel-assisted evolutionary algorithms, for computationally expensive problems. In *Proceedings of the IEEE Conference on Evolutionary Computation*, IEEE, Piscataway, New Jersey, pp. 4116–4123, 2007.
- [26] Z. Z. Zhou, Y. S. Ong, M. H. Lim, B. Lee. Memetic algorithm using multi-surrogates for computationally expensive optimization problems. *Soft Computing*, vol. 11, no. 10, pp. 957–971, 2007.
- [27] S. Cohen, N. Intrator. Automatic model selection in a hybrid perceptron/radial network. *Information Fusion*, vol. 3, no. 4, pp. 259–266, 2002.
- [28] F. Neri, N. Kotilainen, M. Vapa. A memetic-neural approach to discover resources in P2P networks. *Recent Advances in Evolutionary Computation for Combinatorial Optimization*, vol. 153, pp. 113–129, 2008.
- [29] A. Caponio, G. L. Cascella, F. Neri, N. Salvatore, M. Sumner. A fast adaptive memetic algorithm for online and offline control design of PMSM drives. *IEEE Transactions on Systems, Man, and Cybernetics-Part B*, vol. 37, no. 1, pp. 28–41, 2007.
- [30] J. A. Egea, E. Balsa-Canto, M. S. G. Garcia, R. J. Banga. Dynamic optimization of nonlinear processes with an enhanced scatter search method. *Journal of Industrial Chemical Engineering Research*, vol. 48, no. 9, pp. 4388–4401, 2009.
- [31] H. F. Wang, D. W. Wang, S. X. Yang. A memetic algorithm with adaptive hill climbing strategy for dynamic optimization problems. *Soft Computing*, vol. 13, no. 8–9, pp. 763–780, 2009.
- [32] M. D. McKay, R. J. Beckman, W. J. Conover. A comparison of three methods for selecting values of input variables in the analysis of output from a computer code. *Technometrics*, vol. 42, no. 1, pp. 55–61, 2000.
- [33] K. P. Burnham, D. R. Anderson. *Model Selection and Inference: A Practical Information-theoretic Approach*. New York: Springer, 2002.
- [34] A. R. Conn, K. Scheinberg, P. L. Toint. On the convergence of derivative-free methods for unconstrained optimization. *Approximation Theory and Optimization: Tributes to M. J. D. Powell*, New York: Cambridge University Press, pp. 83–108, 1997.
- [35] W. R. Madych. Miscellaneous error bounds for multiquadric and related interpolators. *Computers and Mathematics with Applications*, vol. 24, no. 12, pp. 121–138, 1992.
- [36] P. N. Suganthan, N. Hansen, J. J. Liang, K. Deb, Y. P. Chen, A. Auger, S. Tiwari. Problem definitions and evaluation criteria for the CEC 2005 special session on real-parameter optimization. Technical Report KanGAL 2005005, Nanyang Technological University, Singapore and Kanpur Genetic Algorithms Laboratory, Indian Institute of Technology Kanpur, India, 2005.
- [37] D. Büche, N. N. Schraudolph, P. Koumoutsakos. Accelerating evolutionary algorithms with Gaussian process fitness function models. *IEEE Transactions on Systems, Man, and Cybernetics-Part C: Applications and Reviews*, vol. 35, no. 2, pp. 183–194, 2005.
- [38] D. J. Sheskin. *Handbook of Parametric and Nonparametric Statistical Procedures*, 4th ed., Boca Raton, Florida: Chapman and Hall, 2007.

- [39] R. M. Hicks, P. A. Henne. Wing design by numerical optimization. *Journal of Aircraft*, vol. 15, no. 7, pp. 407–412, 1978.
 - [40] H. Y. Wu, S. Yang, F. Liu, H. M. Tsai. Comparison of three geometric representations of airfoils for aerodynamic optimization. In *Proceedings of the 16th AIAA Computational Fluid Dynamics Conference*, Orlando, Florida, USA 2003.
 - [41] M. Drela, H. Youngren. XFOIL 6.9 User Primer, Department of Aeronautics and Astronautics, Massachusetts Institute of Technology, Cambridge, MA, 2001.
 - [42] J. R. Koehler, A. B. Owen. Computer experiments. *Handbook of Statistics*, S. Ghosh, C. R. Rao, P. R. Krishnaiah, Ed., Amsterdam: Elsevier, pp. 261–308, 1996.
 - [43] M. J. D. Powell. Radial basis function methods for interpolation to functions of many variables. In *Proceedings of the 5th Hellenic-European Conference on Computer Mathematics and its Applications*, LEA Press, Athens, pp. 2–24, 2001.
 - [44] N. Benoudjit, C. Archambeau, A. Lendasse, J. A. Lee, M. Verleysen. Width optimization of the Gaussian kernels in radial basis function networks. In *Proceedings of the 10th European Symposium on Artificial Neural Networks-ESANN*, Belgium, pp. 425–432, 2002.
 - [45] J. Moody, C. J. Darken. Fast learning in networks of locally-tuned processing units. *Neural Computation*, vol. 1, no. 2, pp. 281–294, 1989.
 - [46] R. Storn, K. Price. Differential evolution — a simple and efficient heuristic for global optimization over continuous spaces. *Journal of Global Optimization*, vol. 11, no. 4, pp. 341–359, 1997.
 - [47] M. E. H. Pedersen. Good parameters for differential evolution, Technical Report HL1002, Hvas Laboratories, Norway, 2010.
 - [48] A. Chipperfield, P. Fleming, H. Pohlheim, C. Fonseca. Genetic Algorithm TOOLBOX for Use with MATLAB, Version 1.2, Department of Automatic Control and Systems Engineering, University of Sheffield, Sheffield, England, 1994.
 - [49] J. Kennedy, R. Eberhart. Particle swarm optimization. In *Proceedings of the 1995 IEEE International Conference on Neural Networks*, IEEE, Perth, WA, Australia, pp. 1942–1948, 1995.
 - [50] M. E. H. Pedersen. Good parameters for particle swarm optimization, Technical Report HL1001, Hvas Laboratories, 2010.
 - [51] N. Hansen, A. Ostermeier. Completely derandomized self-adaptation in evolution strategies. *Evolutionary Computation*, vol. 9, no. 2, pp. 159–195, 2001.
- Yoel Tenne** has obtained his Ph.D. degree in mechanical engineering from Sydney University, Australia. He was an Australian Endeavour Research fellow, and a Japan Society for Promotion of Science fellow at Kyoto University. He is currently a senior research fellow at the Ariel University Centre, Israel.
- His research interests include design optimization, computational intelligence, and systems engineering.
- E-mail: y.tenne@ariel.ac.il