An Iterative Two-stage Multi-fidelity Optimization Algorithm for Solving Computationally Expensive Problems

Angus Kenny, Tapabrata Ray, and Hemant Kumar Singh

***Abstract*—The abstract goes here.**

[Angus: Angus can make comments using `\angus{comment}`]

[Ray: Ray can make comments using `\ray{comment}`]

[Hemant: Hemant can make comments using `\hemant{comment}`]

***Index Terms*—IEEE, IEEEtran, journal, LaTeX, paper, template.**

I. Introduction

MANY real-world engineering design problems require estimation of responses that are intractable for exact or analytical methods. In such cases, two alternatives are commonly resorted to: numerical simulations and physical testing of a prototype [1]. When used in-loop during design optimization using iterative methods such as evolutionary algorithms (EAs), both of these methods tend to be prohibitively slow; as well as potentially cost and resource intensive [2]. Such optimization problems where each design evaluation incurs significant cost in any form are referred to as computationally expensive optimization problems.

Simulation-based optimization (SO) refers to the methods that deal with optimization problems involving numerical simulations[1]. Typically, the these methods make use of surrogate approximations, also referred to as metamodels, to reduce the computational expense [3]. The basic principle is to use historical information from previously evaluated designs to build a surrogate model of the response landscape. The predicted values from these surrogate models can be then utilized to determine new candidate solutions that are likely to be competent when evaluated using the true (time consuming) simulation. Through this informed pre-selection, the number of expensive evaluations can be significantly reduced during the optimization.

Some numerical simulation processes allow control over the resolution, or fidelity, of the samples they produce. For example, in finite element analysis (FEA) or computational fluid dynamic (CFD) simulations, the mesh size can be controlled to yield solutions with different fidelities [4], [5]. A coarse mesh yields a low-fidelity (LF) performance estimate that is relatively fast but less accurate, while a fine mesh yields a high-fidelity (HF) estimate that is relatively more time consuming but more accurate. Multi-fidelity SO methods (MFSO) are a special class of SO methods that attempt to efficiently combine the information from different fidelities with an eventual aim of searching for optimal HF designs.

A number of different approaches to MFSO have been explored in the literature. Many of these approaches are based on the co-kriging technique described by Forrester et al. [6] which correlates the two sets of samples, low- and high-fidelity, to produce a single prediction model. This technique is an extension of the autoregressive model first introduced by Kennedy and O'Hagan [7]. The two key aspects in which the different methods in this category deviate from each other, is in how they collect these samples and also how they search the model for potential candidates. Laurenceau and Sagaut [8] investigated a number of different sampling methods for use with kriging and co-kriging in an airfoil design problem; Huang et al. [9] combine a genetic algorithm with co-kriging to optimize wing-body drag reduction in aerodynamic design; Perdikaris et al. [10] incorporate elements of statistical learning into a co-kriging framework to cross-correlate ensembles of multi-fidelity surrogate models; Yang et al. [11] take a physics-informed approach, constructing models based on sparsely observed domain knowledge, representing unknowns as random variables or fields which are regressed using elements of co-kriging; and Giraldo et al. [12] provide an extension to co-kriging for use when the secondary variable is functional, based on the work of Goulard and Voltz [13].

Among the approaches that do not use co-kriging models, some of the prominent ones include the following: Lv et al. [14] employ a canonical correlation analysis-based model, in which the least squares method is used to determine optimal parameters; Ariyarit and Kanazaki [15] use a hybrid method which employs a kriging model to estimate local deviations and a radial basis function to approximate the global model in airfoil design problems; Hebbal et al. [16] and Cutajar et al. [17] both use machine learning techniques that treat the layers of a deep Gaussian process as different fidelity levels to capture non-linear correlations between fidelities; Xu et al. [18] use a two-stage

The authors are with the School of Engineering and Information Technology, The University of New South Wales, Australia.
Emails: {angus.kenny, t.ray, h.singh}@adfa.edu.au.

[1]For brevity, the discussion is restricted to simulation-based design, but the same principles can be applied to other expensive optimization such as those involving physical prototyping.

process which first uses ordinal transformation to transform the original multi-dimensional design space into a one-dimensional ordinal space and then samples from this ordinal space using a method based on the optimal computational budget allocation (OCBA) algorithm proposed by Chen and Lee [19]; Branke et al. [4] and Lim et al. [20] both take evolutionary approaches to solving MFSO problems; Bryson and Rumpfkeil [21] propose a quasi-Newton method framework which can be used with many different surrogate model techniques; and Ng and Willcox [22] propose a number of approaches for multi-fidelity optimization under uncertainty.

Many of these approaches sample low- and high-fidelity solutions *a priori*, then build models based on these samples and use some global search method to optimize them. To ensure these constructed models are properly representative, it is important to maintain a diversity of samples across the entire the design space; however, sampling without any prior knowledge can result in many computational resources being expended in areas which do not contain any promising candidates. Of those which do sample iteratively, information is typically only shared in one direction between the two datasets. Low-fidelity solutions are sampled randomly, or using some indepenent process, and used to inform where the high-fidelity solutions should be sampled from; but no information is then shared in the reverse direction, to inform the low-fidelity sampling in the next iteration. Again, this can result in computational resources being consumed unnecessarily in regions of the search space which are unproductive, especially in high-dimensional problems.

To overcome these limitations and improve the performance for MFSO methods, this paper proposes an iterative two-stage, bound-constrained, single-objective multi-fidelity optimisation problems, referred to here as $MFITS$. It uses previously obtained information about promising areas of the search space to define a restricted neighbourhood using a guided differential evolution (DE) [23] process on a kriging model of the low-fidelity samples. This neighbourhood is then sampled from and searched, using a method derived from OCBA, to determine a set of candidates to undergo low-fidelity simulation. The information from these simulations is used to update the low-fidelity model and also a co-kriging-based surrogate model of the high-fidelity samples, which is searched globally using DE to find a suitable candidate for high-fidelity simulation. Finally, these high-fidelity samples are used to update the surrogate model and also to help determine the restricted neighbourhood in the next iteration. By using the high-fidelity simulation information to inform and restrict the region of interest while searching the low-fidelity model, $MFITS$ allows two-way information sharing between the sets of samples. The performance of the $MFITS$ model is compared against a baseline co-kriging-based MFSO algorithm on two separate datasets. The first is a common set of multi-fidelity test-functions from the literature, and the second is a set of multi-fidelity test functions that are generated from standard test functions using the methods described in the paper by Wang et al. In addition to this, some important properties of $MFITS$ are also investigated.

The remainder of this paper is organized as follows. Section II provides the fundamentals and background of the proposed model, along with a description of the type of problems tackled and related work. The $MFITS$ algorithm is detailed in Section III, describing all of its constituent parts and detailing some similarities and differences with a related technique from the literature. Experimental design and datasets are discussed in Section IV, with Section V giving the results of these experiments and a discussion of their implications. Finally, Section VI provides the conclusion and outlines any future directions of research.

II. Background and Related Work

This section details the type of problem $MFITS$ is designed to address; provides information about two critical components that are used in its construction; and gives a brief summary of another method which is similar to $MFITS$ from the literature.

A. Problem type

The algorithm presented here is designed to address bound-constrained, single-objective, multi-fidelity optimisation problems with continuous variables. Here, the terms bound-constrained and continuous imply that a variable may have an upper- or lower-bound which it cannot exceed, but that it may take any value (precision issues notwithstanding) between these bounds in the decision space; and that there are no regions in the objective space which are forbidden.

Let P be a problem instance with D decision variables and let $\boldsymbol{x} \in \prod_{i=1}^{D}[l_i, u_i]$ be a *solution* to P, represented by a real vector[2] $\boldsymbol{x}$ such that $l_i \leq x_i \leq u_i$ for all $i \in [D]$, where $\boldsymbol{l}, \boldsymbol{u} \in \mathbb{R}^D$ are the lower- and upper-bounds, respectively[3]. The *objective value* of a solution to P is given by the function

$$f : \prod_{i=1}^{D}[l_i, u_i] \to \mathbb{R}, \ \boldsymbol{x} \mapsto f(\boldsymbol{x}) \,, \tag{1}$$

which is typically smooth. When this objective value is minimized, the so-called *optimal* solution to P is $\boldsymbol{x}^*$, such that $f(\boldsymbol{x}^*) \leq f(\boldsymbol{x})$ for all $\boldsymbol{x} \in \prod_{i=1}^{D}[l_i, u_i]$, although it may be maximized without loss of generality.

Examples of these types of problems abound in the field of engineering design [1], [24] and often they require some form of numerical simulation to compute the objective value of their solutions. This numerical simulation can be very computationally expensive, therefore researchers often employ artificial test functions when developing algorithms. While not generally being an accurate model for the behaviour of real-world problems, test functions are very fast

[2]Bold type is used here to indicate a vector (indexed by superscript) and regular type is used for elements (indexed by subscript). E.g., $\boldsymbol{x}^i$ is the ith indexed vector, and x^i_j is the jth element of the ith vector.

[3]To conserve space, the following shorthand is used in this paper: $[k] = \{1, 2, \ldots, k\}$ and $[k^*] = \{0, 1, \ldots, k-1\}$.

to evaluate and can be customized to gauge the performance of algorithms on a variety of fitness landscapes.

The dataset in Lv et al. [14] uses the customized addition, removal and modification of terms to produce a transformed function with a desired correlation to the original. While this method allows control over the shape of the fitness landscape, it requires analysing each test function individually and only permits a single level of fidelity for each transformation.

The second strategy introduces external noise which models the errors that occur when simulation fidelity is decreased. Wang et al. [25] analysed the behaviour of many numerical simulations under different fidelity conditions and formulated a generic transformation function to turn any test function f of the form in Equation 1 into a low-fidelity version $\tilde{f}$:

$$\tilde{f} : \prod_{i=1}^{D} [l_i, u_i] \times [0, 10000] \to \mathbb{R},\ (\boldsymbol{x}, \phi) \mapsto f(\boldsymbol{x}) + e(\boldsymbol{x}, \phi)\,, \tag{2}$$

where ϕ is the *fidelity level*, with $\tilde{f}(\boldsymbol{x}, 10000) = f(\boldsymbol{x})$ and $\tilde{f}(\boldsymbol{x}, 0)$ having the worst possible correlation to $f(\boldsymbol{x})$. The *error function*, $e(\boldsymbol{x}, \phi)$ — which returns a single real value — can be one of ten different functions, all of which are independent of $f(\boldsymbol{x})$.

Because the error function is always independent of $f(\boldsymbol{x})$, this method can be applied to any test function at all; and because ϕ is real-valued, many different fidelity levels can be modelled easily— although some analysis must still be performed if specific correlation coefficients are required.

B. Kriging and co-kriging

Originally arising from geostatistical methods used for ore valuation in mining research — but since being applied across a number of domains — is the so-called *kriging* method [1]. Kriging is an interpolation technique that uses a limited set of samples to predict the objective value at a point that has not been sampled yet.

Let P be a problem instance with D decision variables and let $X = \{\boldsymbol{x}^1, \boldsymbol{x}^2, \ldots, \boldsymbol{x}^n\}$ be a set of n sample points with observed objective values $\boldsymbol{y} = \{y_1, y_2, \ldots, y_n\}$, where $\boldsymbol{x}^i \in \mathbb{R}^D$ for $i \in [n]$. A kriging model of these samples is the Gaussian process

$$Y(\boldsymbol{x}) = f(\boldsymbol{x}) + Z(\boldsymbol{x}), \tag{3}$$

where, $f(\boldsymbol{x})$ is a polynomial regression function based on $\boldsymbol{y}$ and encapsulates the main variations in the samples. The function $Z(\boldsymbol{x})$ is a Gaussian process with mean 0, which models the residual error. As the mean of $Z(\boldsymbol{x})$ is 0, $f(\boldsymbol{x})$ is the mean of $Y(\boldsymbol{x})$.

Low-fidelity samples are typically much cheaper to produce than high-fidelity ones, a fact which the *co-kriging* technique exploits to great effect. Adapted from Kennedy and O'Hagan's autoregressive model [7], co-kriging correlates multiple sets of samples with different fidelities to produce a single model that approximates the high-fidelity samples. The residual error for a sample point evaluated at a given fidelity is recursively modelled as a function of the error of the same point evaluated at the fidelity below it — until some foundational model with lowest fidelity is reached. Because of this, the Markov property must be assumed, that given a sample point evaluated at some fidelity, no more information about that point can be gained by evaluating it at a lower fidelity.

Let $X_H = \{\boldsymbol{x}_H^1, \boldsymbol{x}_H^2, \ldots, \boldsymbol{x}_H^n\}$ be the set of n high-fidelity sample points with observed objective values $\boldsymbol{y}_H = \{y_{H(1)}, y_{H(2)}, \ldots, y_{H(n)}\}$ and $X_L = \{\boldsymbol{x}_L^1, \boldsymbol{x}_L^2, \ldots, \boldsymbol{x}_L^m\}$ be the set of m low-fidelity[4] sample points with observed objective values $\boldsymbol{y}_L = \{y_{L(1)}, y_{L(2)}, \ldots, y_{L(m)}\}$. A kriging model $Y_L(\boldsymbol{x})$ of the low-fidelity samples are constructed according to Equation 3. Using this, the high-fidelity model is

$$Y_H(\boldsymbol{x}) = \rho(\boldsymbol{x})\mu_L(\boldsymbol{x}) + Z_d(\boldsymbol{x}), \tag{4}$$

where $\rho(\boldsymbol{x})$ is a scaling factor, determined as part of the MLE of the second model; $\mu_L(\boldsymbol{x})$ is the mean of $Y_L(\boldsymbol{x})$; and $Z_d(\boldsymbol{x})$ is a Gaussian process which models the difference between $Y_H(\boldsymbol{x})$ and $\rho(\boldsymbol{x})\mu_L(\boldsymbol{x})$.

An in-depth mathematical treatment of kriging, co-kriging and how to use these models to make predictions can be found in [1], [6], [7], [26].

C. Optimal computing budget allocation

Stochastic simulation is a noisy process, requiring multiple simulation replications in order to accurately approximate the true fitness value of a given design.

Assuming a normal distribution, the mean fitness value for a design, μ, is unknown, but it can be estimated by its sample mean $\hat{\mu}$. Given a set of k candidate designs and a finite computing budget T, the optimal computing budget allocation (OCBA) [19] method aims to find an allocation such that $N_1 + N_2 + \cdots + N_k = T$, where N_i is the total replications allocated to design i, in order to select the best design, $b \in [k]$, such that $\hat{\mu}_b < \hat{\mu}_i$ for all $i \in [k]$.

The probability that design b actually *is* the the best design is called the probability of correct selection (PCS). The PCS can be estimated using Monte Carlo simulation, but this is time-consuming; therefore, the following problem is formulated in [19] to compute the approximate probability of correct selection (APCS):

$$\begin{aligned} \underset{N_1,\ldots,N_k}{\text{maximize}} \quad & 1 - \sum_{i=1, i\neq b}^{k} P\{\tilde{\mu}_b < \tilde{\mu}_i\}\,, \\ \text{such that} \quad & \sum_{i=1}^{k} N_i = T\,, \quad N_i \geq 0, \end{aligned} \tag{5}$$

[4]Although only two fidelity levels are employed here, without loss of generality, the method can be extended to an arbitrary number of fidelities.

Based on the work in [19], the APCS is asymptotically maximized (as $T \to \infty$) when

$$\frac{N_i}{N_j} = \left(\frac{\sigma_i / \delta_{b,i}}{\sigma_j / \delta_{b,j}} \right)^2 , \tag{6}$$

$$N_b = \sigma_b \sqrt{\sum_{i=1, i \neq b}^{k} \frac{N_i^2}{\sigma_i^2}} , \tag{7}$$

where N_i is the total replications allocated to design i and $\delta_{b,i} = \hat{\mu}_b - \hat{\mu}_i$, for all $i, j \in \{1, 2, \ldots, k\}$ with $i \neq j \neq b$.

The implications of Equations 6 and 7 are such that additional computing resources are not only allocated to those designs with a small sample mean, but also to potentially promising designs that have a high variance. A high variance indicates uncertainty in the prediction, which increased volume of replications will help to address.

Using the above results, Algorithm 1 details an iterative process to select a design, based on maximising APCS.

Algorithm 1: $OCBA$ procedure

Input: k, number of designs; T, computing budget; Δ, replications per update; n_0, initial replications.
Output: b, index of best design.

$b \leftarrow \emptyset$ {Initialize b}
$N_i \leftarrow n_0 \,, \ \forall i \in [k]$ {Count initial replications}
$\boldsymbol{S}^i \leftarrow Sim(n_0) \,, \ \forall i \in [k]$ {Perform initial replications}
while $\sum_{i \in [k]} N_i < T$ **do**
  $\boldsymbol{N}' \leftarrow \boldsymbol{N}$ {Store old allocations}
  $\hat{\boldsymbol{\mu}}, \hat{\boldsymbol{\sigma}} \leftarrow Stats(\boldsymbol{S}), \ \forall i \in [k]$ {Compute sample statistics}
  $b \leftarrow \text{argmin}_i \ \hat{\mu}_i, \ \forall i \in [k]$ {Update b}
  $\boldsymbol{N} \leftarrow Allocate(\hat{\boldsymbol{\mu}}, \hat{\boldsymbol{\sigma}}, \boldsymbol{N}, \Delta)$ {Allocate Δ replications}
  $S_i \leftarrow Sim(N_i - N_i'), \ \forall i \in [k]$ {Perform additional simulations}
end while

In this algorithm, $Sim(n)$ is a function that returns the output of running the stochastic simulation n times; $Stats(\boldsymbol{S})$ computes the sample statistics for a set of simulation outputs; and $Allocate(\hat{\boldsymbol{\mu}}, \hat{\boldsymbol{\sigma}}, \boldsymbol{N}, \Delta)$ is a function that allocates Δ replications among the designs, in accordance with Equations 6 and 7.

D. MO^2TOS

Optimal computing budget allocation (OCBA) operates on the assumption that each stochastic simulation being performed will have the same input parameters, and does not consider the case where the fitness of a solution may be evaluated with different fidelities. The multi-fidelity optimisation with ordinal transformation and optimal sampling (MO^2TOS) [18] framework is a two-stage algorithm that addresses this consideration, by combining ideas from OCBA with ordinal transformation [27].

Given a problem instance P and a finite computing budget T, the MO^2TOS algorithm takes advantage of the relatively low cost of low-fidelity evaluations to identify promising regions of the search space that can subsequently be exploited by iterative high-fidelity evaluations.

A large population of solutions are evaluated in low-fidelity and then ranked by their evaluated value. This ranked population is then partitioned into equal-sized groups. The reason for ranking the solutions before partitioning, is to increase the probability that solutions which share a group are proximate to each other in the transformed objective space, as opposed to the original design space. As a result, regardless of where solutions may be, it is more likely that solutions within a group will have a similar performance.

These partitioned groups are treated as de facto "designs" for the purposes of OCBA allocation; however, instead of multiple simulation replications being performed on the same design, the allocations will determine how many solutions are selected from a group for high-fidelity evaluation.

Initially, each group has n_0 solutions selected — without replacement — and evaluated in high-fidelity. These results are used to determine the sample statistics of the groups. Equations 6 and 7 are used to allocate a portion of the total budget, Δ, to the set of groups. Solutions are selected from these groups in accordance with this allocation, and evaluated in high-fidelity. The sample statistics are computed once again and the process repeats, while the total budget consumed is less than T.

The value of Δ is typically quite small, as there may be significant bias between the low- and high-fidelity evaluated solutions. By keeping Δ small, it ensures that the sample statistics of the groups are updated more frequently, meaning that the allocations are based more on the statistics of each group, as opposed to their intial partitioning.

The MO^2TOS framework has some limitations, due to the fact that it samples the low-fidelity candidates *a priori* and the fact that is a two-step process that operates directly on the high- and low-fidelity functions, as opposed to an iterative one. By sampling once, without any prior information, the sampling must occur uniformly across the entire design space. This means that unnecessary computational budget is likely to be expended in areas which are not helpful to the search, and the only solutions that can be considered are those which were part of the initial set of samples. For example, if $\boldsymbol{x}^* = (1.0005, 0.0005)$, but $\boldsymbol{x}^*$ is not in the initial low-fidelity samples, there is no way for the search to produce it without the addition of some kind of local search operation — even if $\boldsymbol{x} = (1.0005, 0.0006)$ is in the low-fidelity samples. Similarly, it is necessary to initially evaluate n_0 solutions with high-fidelity from each partitioned group, which can result in further unnecessary expenditure of computational budget.[Hemant: Is it possible to demonstrate this limitation through a small conceptual example ? If so I think it would really add weight to the motivation. A 1 or 2-dimensional test problem with LF and HF curves and some samples. If you can point out what the algorithm will miss in the given snapshot of the search which could be avoided by what you're going to propose, it'll certainly catch the reader's attention better. Then, after you've discussed your algorithm, you can revisit the snapshot and explain how your algorithm will evaluate a different, more useful, new point

to expedite the search.]

Algorithm 5 in Section IV gives the basic structure of the MO^2TOS procedure used for numerical comparison in this paper, more detail can be found in [18].

III. $MFITS$ Algorithm

The multi-fidelity optimisation algorithm proposed in this paper is an iterative two-stage process, which maintains two separate surrogate models that share information between them. The first surrogate is a kriging model of the low-fidelity samples. This model is searched to find the best potential candidates, within a restricted region, to evaluate with low-fidelity. The information from this search is used to update the first model and a co-kriging model that combines both low- and high-fidelity samples to approximate the high-fidelity objective function. This second model is globally searched to determine a suitable candidate for high-fidelity evaluation, and these samples is used to update the co-kriging model and to determine the neighbourhood for the next search of the low-fidelity surrogate. Algorithm 2 gives a description of this process. The functions within this procedure are defined as a general framework in the remainder of this section, with the specific parameters and methods given in Section IV.

Algorithm 2: $MFITS$ procedure

Input: P, problem data; $N_{L_{max}}$, max size of low-fidelity population; $N_{e_{max}}$, maximum number of evaluations; ρ, parameter set for LocalOCBA.
Output: Best solution found $\boldsymbol{x}^\beta$.

$X_v \leftarrow LHS(P),\ \forall v \in \{L, H\}$ {Generate initial populations}
$A_v, N_e \leftarrow f_v(X_v, 0),\ \forall v \in \{L, H\}$ {Evaluate initial populations}
$\boldsymbol{x}^\beta \leftarrow \emptyset$ {Initialize $\boldsymbol{x}^\beta$}
while $N_e < N_{e_{max}}$ **do**
  $M_L \leftarrow Krige(A_L)$ {Update low-fidelity kriging model}
  $M_C \leftarrow CoKrige(A_L, A_H)$ {Update co-kriging model}
  $\boldsymbol{x} \leftarrow GlobalSearch(M_C)$ {Globally search co-kriging model}
  $\boldsymbol{\alpha}, N_e \leftarrow f_H(\boldsymbol{x}, N_e)$ {Evaluate and update total cost}
  $A_H \leftarrow A_H \cup \{\boldsymbol{\alpha}\}$ {Add $\boldsymbol{\alpha}$ to high-fidelity archive}
  $\boldsymbol{x}^\beta \leftarrow \min(f_H(\boldsymbol{x}^\beta), f_H(\boldsymbol{x}))$ {Update best solution}
  $\epsilon \leftarrow Sigmoid(N_e, N_{e_{max}})$ {Determine size of neighbourhood}
  $A_L, N_e \leftarrow LocalOCBA(\rho, M_L, A_L, \boldsymbol{x}^\beta, \epsilon)$ {Locally search low-fidelity model}
  if $|A_L| > N_{L_{max}}$ **then**
    $A_L \leftarrow Winnow(A_L, N_{L_{max}})$ {Control population size}
  end if
end while

A. Initialisation, evaluation and surrogate models

The function $LHS(P)$ returns an initial population sampled from a latin hypercube, he bounds of which are defined in P. Latin hypercube sampling is used in order to start with as broad a picture of the search space as possible. Solutions are evaluated using the function $f_v(X, N_e)$, at some fidelity $v \in \{L, H\}$, which returns a set of "archive" pairs comprising the solution and its objective value. It also updates the total cost incurred, N_e.

Two surrogate models are used, $Krige(A_L)$ takes the low-fidelity samples and returns a kriging model approximating it, and $CoKrige(A_L, A_H)$ takes both the low- and high-fidelity archive and returns a co-kriging model that approximates the fitness landscape of the high-fidelity objective function. The function $GlobalSearch(M_C)$ takes the co-kriging model as input and returns the best solution that can be found using some global search method. This output solution is evaluated in high-fidelity, the information from which is used to update the co-kriging model, and also to determine the restricted neighbourhood used to locally search the low-fidelity kriging model.

B. Restricted neighbourhood

The correlation between the low- and high-fidelity samples can vary wildly between problems, meaning identifying a promising region in the low-fidelity kriging model does not necessarily mean identifying a promising region in the high-fidelity co-kriging model. Therefore, instead of searching the low-fidelity model globally, it makes sense to search for promising candidates within a restricted neighbourhood, defined using information from the high-fidelity model. As the goal of performing low-fidelity evaluations in co-kriging models is to gain information about the shape of the fitness landscape, more than it is to find the "best" low-fidelity solution, restricting the neighbourhood in this way allows the algorithm to focus on the region of interest — without getting "distracted" by global optima that might not be useful in the over-all search.

In order to restrict the region of interest, a so-called *guided DE* process is employed. In differential evolution (DE), assuming complete recombination occurs, a candidate child solution x_c is produced from three parent solutions $\boldsymbol{x}^1$, $\boldsymbol{x}^2$ and $\boldsymbol{x}^3$ with the following formula [23]:

$$\boldsymbol{x}^c = \boldsymbol{x}^1 + F(\boldsymbol{x}^2 - \boldsymbol{x}^3)\,, \tag{8}$$

with F being a constant called the mutation factor. This is simply scaled vector addition, therefore this produced child $\boldsymbol{x}^c$ can be "nudged" further towards a reference point (in this case x_β) by a similar process:

$$\boldsymbol{x}^{c'} = \boldsymbol{x}^c + \boldsymbol{g}(\boldsymbol{x}^\beta - \boldsymbol{x}^c)\,, \tag{9}$$

where $\boldsymbol{G}$ is the so-called *guide factor*. This is illustrated in Figure 1.

In this figure, it can be seen that $\boldsymbol{x}^{c_1}$ is produced using $\boldsymbol{x}^1$, $\boldsymbol{x}^2$ and $\boldsymbol{x}^3$, and then translated using Equation 9 to produce $\boldsymbol{x}^{c_1'}$. Child solutions $\boldsymbol{x}^{c_2}$ and $\boldsymbol{x}^{c_3}$ are produced using two other sets of solutions that are not pictured, and then translated using the same equation. If this process is continued, the result is a "cloud" of candidate solutions in the vicinity of $\boldsymbol{x}^\beta$, the centre of the region of interest.

The guide factor determines how far the resulting solution is translated in the direction of $\boldsymbol{x}^\beta$. At the beginning of the search, $\boldsymbol{x}^\beta$ is determined by optimising a very coarse model built with sparse information, and cannot be trusted to be indicative of a promising region of the search space; as the search continues, the model becomes more accurate and the information it produces more trustworthy. Therefore,

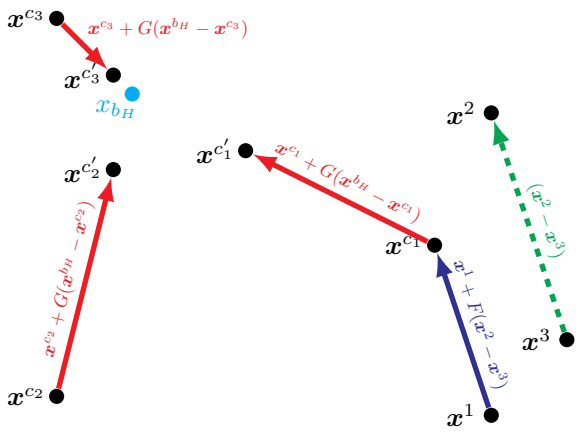


Fig. 1: The DE recombination process can be "guided" towards the best high-fidelity solution found so far.

the algorithm should be explorative in the beginning of the search, but exploitative towards the end. One function which exhibits these properties is the logistic sigmoid curve

$$\epsilon = \frac{L}{1 + e^{-k(x - x_0)}}\,, \tag{10}$$

where x_0 is the x value of the sigmoid's midpoint, L is the curve's maximum and k is the steepness parameter, which controls how fast the function "ramps up". The function $Sigmoid(N_e, N_{e_{max}})$ uses the total cost and maximum cost to compute this ϵ value. Figure 2 gives a plot of the logistic sigmoid curve with $L = 0.99$, $k = 10$ and $x_0 = 0.2$. Using

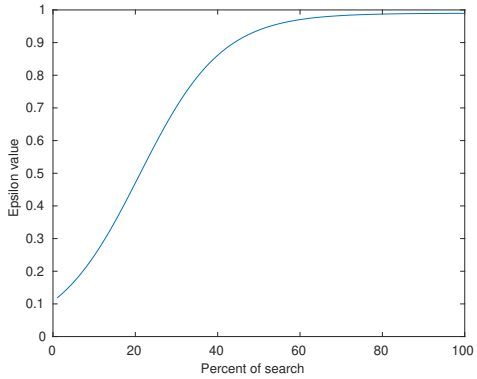


Fig. 2: The logistic sigmoid curve flattens at around 80%.

this value, $\boldsymbol{g}$ can be computed as:

$$\boldsymbol{g} = \epsilon + (1 - \epsilon)\boldsymbol{r}\,, \tag{11}$$

where $\boldsymbol{r} \in [0, 1]^D$ is a random vector with D components, making $\boldsymbol{g} \in [\epsilon, 1]^D$ also a random vector. This ensures the cloud of candidate solutions around $\boldsymbol{x}^\beta$ is sparse at the beginning of the search, allowing for better global exploration, while focusing the search more tightly on promising regions towards the end — crucially, without adding any extra parameters.

C. LocalOCBA

Once the candidate solutions have been been generated in the vicinity of $\boldsymbol{x}^\beta$, they must be selected from. As already stated, the goal of evaluating solutions in low-fidelity is not to optimize the low-fidelity objective function, but to provide as much information for the model as possible. The optimal computing budget allocation (OCBA) algorithm selects from a set of solutions with the goal of reducing uncertainty within simulation models (by allocating computing resources). This principle can be applied here, with some modifications, as described in Algorithm 3.

Algorithm 3: *LocalOCBA* procedure

Input: $\rho = \{\Delta_1$, total to be sampled; Δ_2, samples per statistics update$\}$; M, low-fidelity model; A_L, low-fidelity archive; $\boldsymbol{x}^\beta$, best high-fidelity solution ; ϵ, sigmoid value; N_e, total cost incurred.
Output: A_L, updated low-fidelity archive; N_e, updated total cost.

$X \leftarrow GuidedDE(A_L, \boldsymbol{x}^\beta, \epsilon)$ {Generate child population}
$A_X \leftarrow f_M(X)$ {Approximate each child by model M}
$G, k \leftarrow Partition(A_X)$ {Partition solutions}
$S_i \leftarrow \emptyset,\ \forall i \in [k]$ {Empty groups for selected solutions}
while $\sum_{i \in [k]} |S_i| < \Delta_1$ **do**
  $\hat{\mu}_i, \hat{\sigma}_i \leftarrow$ sample statistics for $S_i,\ \forall i \in [k]$ (if $|S_i| < 2$, use G_i)
  $R \leftarrow GetRatios(\hat{\boldsymbol{\mu}}, \hat{\boldsymbol{\sigma}})$ {compute allocation ratios}
  $D \leftarrow Allocate(R, S, G) : \sum_{i \in [k]} |D_i| = \Delta_2$ {Allocate Δ_2 solutions according to ratios R}
  $D_i, N_e \leftarrow f_L(D_i, N_e),\ \forall i \in [k]$ {Evaluate allocated solutions}
  $S_i \leftarrow S_i \cup D_i,\ \forall i \in [k]$ {Add to selected solutions}
  $G_i \leftarrow G_i \setminus D_i,\ \forall i \in [k]$ {Selection without replacement}
end while
$A_L \leftarrow A_L \cup \bigcup_{i \in [k]} S_i$ {Combine all selected solutions}

Here, the $GuidedDE(A_L, \boldsymbol{x}^\beta, \epsilon)$ process is used to generate a set of candidate solutions, which are approximated using $f_M(X)$. This function takes a set of solutions and returns a set of pairs comprising the solution and its approximation on some model M. The solutions are ranked and partitioned using a clustering algorithm, based on their approximated value. The purpose of ranking the solutions first is to increase the probability that solutions within a clustered group will tend to have a similar performance to each other, regardless of their proximity in the decision space. This helps to ensure a diversity of solutions will be selected, while still preferencing the more promising candidates. The function $Partition(X)$ returns a set of solution groups G and the number of groups k.

OCBA principles are used to select — without replacement — from these groups to populate a set of empty groups S. First, sample statistics are computed for all groups of selected solutions in S. If there are fewer than two solutions in a group, then its corresponding group in G is used. The function $GetRatios(\hat{\boldsymbol{\mu}}, \hat{\boldsymbol{\sigma}})$ uses these statistics to compute allocation ratios in accordance for each group with standard OCBA practice. These ratios are used by $Allocate(R, S, G)$ to allocate Δ_2 solutions to be evaluated as low-fidelity and added to the selected solutions S.

Once Δ_1 solutions have been selected, they are added to the low-fidelity archive, and the updated archive is returned.

D. Population size control

Due to the fact that many more low-fidelity solutions are added to the archive between each high-fidelity evaluation, the size of the low-fidelity archive can become too big for some kriging and co-kriging algorithms, or too concentrated if the search is focused on the same area for too long. Therefore, a maximum population size $N_{L_{max}}$ should be set such that once it reaches that threshold, the population should be maintained at that level. Choosing a steady-state method such as ranking the solutions by their value and selecting the top $N_{L_{max}}$ will cause the population to converge and lose diversity over time. As the purpose of maintaining the low-fidelity population is to provide the co-kriging model with information about the shape of the fitness landscape, this loss of diversity can be very detrimental.

The function $Winnow(A_L, N_{L_{max}})$ takes an archive and a maximum size and returns a winnowed archive with exactly $N_{L_{max}}$ solutions. It does this by partitioning the solutions — in the decision space — into $N_{L_{max}}$ different groups using a clustering algorithm, such as k-means clustering. Most of these clusters will contain only one solution which is added to the winnowed archive; for those that have more than one solution, only the solution with the best value is selected. This ensures that the archive never has more than $N_{L_{max}}$ solutions, but diversity is maintained throughout the population.

E. Similarities and differences to MO^2TOS

There exist some similarities between $MFITS$ and the MO^2TOS framework. For example, both use a two-step process of ordering a population of solutions and then selecting from them using ideas from OCBA. Despite the similarities, there are several key aspects which differentiate the two algorithms from each other.

The biggest difference is that $MFITS$ is an iterative process, whereas MO^2TOS is a two-step algorithm which is only run through one time. As MO^2TOS only performs a single iteration, it cannot use any prior knowledge to determine where it should concentrate its resources when evaluating the initial low-fidelity population. Therefore, it must evaluate uniformly across the whole search space, and subsequently expend computational budget in areas which are not beneficial to the search. In contrast, $MFITS$ uses information from previous iterations, to identify promising areas of the search space that it can exploit.

Another difference is that MO^2TOS operates on the high- and low-fidelity objective functions directly, across the entire search space. Solutions are selected using information from low-fidelity evaluations, to be evaluated in high-fidelity. $MFITS$ uses its ranking and selection phases in order to select from a neighbourhood of solutions that have been identified in a promising region of the search space. These selections are informed by a kriging model of the low-fidelity function, and selected for low-fidelity evaluation. The high-fidelity evaluations are determined by a separate search that is perfomed on the co-kriging model, updated by the selected low-fidelity evaluations. Because of this, it is not necessary to perform the initial n_0 evaluations before computing the sample statistics, as the goal is not to optimize the low-fidelity objective function, just select from a set of ranked candidates.

The fact that MO^2TOS must determine all of its candidates *a priori* means that it can only sample from a pre-defined, finite set of solutions, and is unable to refine what it produces. By doing on-line sampling, $MFITS$ is theoretically able to produce any feasible solution and improve upon any promising ones by focusing the search on the local neighbourhood.

Finally, MO^2TOS partitions the ranked population into a fixed number of equal-sized groups, whereas $MFITS$ uses a clustering algorithm to determine the number and size of partitions. This ensures that the average distance between groups in objective space is maximized. [Hemant: Referring back to my comment in the previous section, here is where you can revisit/redraw the figure and show how the next selected point(s) are of higher quality for your algorithm. Of course it will not demonstrate every component of your algorithm, but at least the concept demonstration why your selection method is likely to improve performance.]

IV. Numerical Experiments

The experiments were carried out on [Angus: insert details of machine and OS here]. All code was implemented and executed in MATLAB[Angus: version number], with kriging and co-kriging models constructed using the ooDACE toolbox [28].

The $MFITS$ algorithm was compared against a baseline co-kriging algorithm and an implementation of the MO^2TOS [18] method proposed by Xu et al. All algorithms were run 30 times on each problem instance for the equivalent of 2000 low-fidelity evaluations, recording the best objective value, the mean and standard deviation of the resulting solutions produced by each run.

A. Datasets

The experiments were run on a collection of bound-constrained, single-objective, multi-fidelity test functions that can be divided into two different datasets. The first set (dataset A) is a selection of problem instances taken from Lv et al. [14]. Problems $f10$ to $f17$ were chosen from this dataset as they contain between three and eight decision variables, determined as a representative range of easy to difficult problems. The problem instances here are provided with explicit definitions for both the high- and low-fidelity functions.

The second set of problem instances (dataset B) is generated using the Griewank [29] and Michalewicz [30] test problems (Figures 3 and 4), taken from the literature.

Each problem was instantiated with versions for three, five and eight decision variables. This matches the range of the first dataset, and also allows the performance of the algorithms to be judged across different sized problems in a more controlled environment. These test problems are not naturally multi-fidelity optimisation problems, so low-fidelity evaluations must be made by applying the methods described in [25]. Two different error functions were tested for each instance, modelling two types of fidelity error: resolution and stochastic. An appropriate fidelity level ϕ for each problem was chosen by computing the square of the Pearson correlation coefficient r^2 for different fidelity values and selecting a value for ϕ such that r^2 was between 0.65 and 0.85, to be commensurate with the first dataset.

See Tables III and IV in Appendix A for test function and error function equations, respectively.

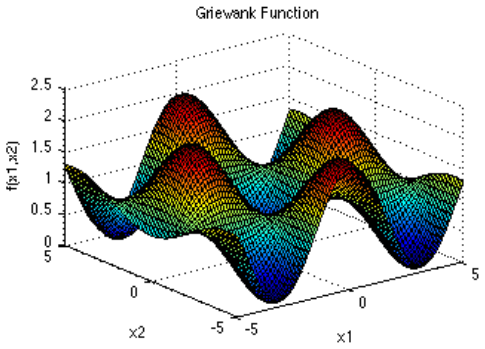


Fig. 3: The Griewank test function in 2D.[Angus: will do a better version]

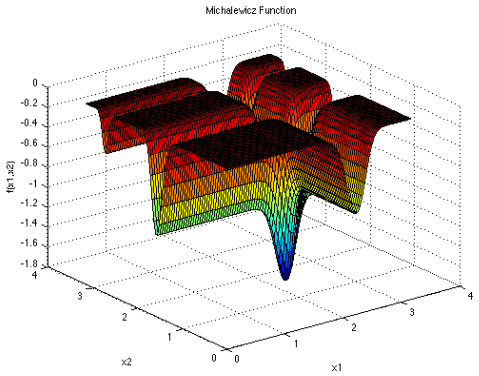


Fig. 4: The Michalewicz test function in 2D.[Angus: will do a better version]

B. $MFITS$ parameters

The $MFITS$ algorithm can be viewed as a general framework into which various search and modeling methods can be "plugged". Therefore, its components can be divided into two categories: $MFITS$ specific components, which are intrinsic to its operation and whose parameters are also intrinsic; and, generic method components, such as global search techniques and modelling methods which can be substituted for other similar techniques, which come with their own parameters. The $MFITS$ specific parameters are discussed in general in Section III; this section will detail the specific values for these parameters and for the generic search method used.

To reduce the number of arbitrary parameters, initial population sizes are defined as a function of problem size. The high-fidelity population has a size of $6D$, as this is the smallest number of high-fidelity solutions that can be used to generate a co-kriging model for all of the instances, with the low-fidelity population being $18D$, which was chosen using sensitivity analysis to find the smallest value which still produces good solutions across the whole set of instances.

The maximum size that the low-fidelity archive can reach before it must be truncated is 400, this was chosen because values much higher than this significantly slow the procedure down, for minimal-to-no improvment.

In the $LocalOCBA$ procedure, the total number of solutions to be sampled, Δ_1, is 25 and the number of samples per statistics update, Δ_2, is 5. These were chosen to provide a good balance between speed and efficiency. The clustering method used to partition the solutions is k-means clustering, with the value for k being chosen using the so-called *elbow method*.

Differential evolution (DE) [23] was used to globally search the updated co-kriging model. This DE was run over 30 generations with a crossover rate of 0.9, a mutation factor of 0.5 and a population size of 100.

The kriging models used are from the ooDACE toolbox [28], with the default paramters.

C. Baseline co-kriging algorithm

Co-kriging is a popular approach to solving many multi-fidelity problems. The outer-loop of the $MFITS$ algorithm functions by iteratively updating a co-kriging model using data from a modified version of the OCBA procedure. In order to demonstrate the effectiveness of this new technique, the performance of $MFITS$ is compared against the baseline simple co-kriging algorithm given in Algorithm 4.

Algorithm 4: Baseline co-kriging procedure

Input: P, problem data; $N_{L_{max}}$, max size of LF pop; $N_{e_{max}}$, maximum number of evaluations; Δ, new LF per iteration.
Output: Best solution found $\boldsymbol{x}^\beta$.

$X_v \leftarrow LHS(P),\ \forall v \in \{L, H\}$ {Generate initial populations}
$\boldsymbol{x}^\beta \leftarrow \emptyset$ {Initialize x_β}
while $N_e < N_{e_{max}}$ **do**
  $A_L \leftarrow A_L \cup LHS(\Delta)$ {Add Δ new solutions to LF archive}
  if $|A_L| > N_{L_{max}}$ **then**
    $A_L \leftarrow winnow(A_L, N_{L_{max}})$ {Control population size}
  end if
  $M_C \leftarrow CoKrige(A_L, A_H)$ {Update co-kriging model}
  $\boldsymbol{x} \leftarrow GlobalSearch(M_C)$ {Globally search co-kriging model}
  $\boldsymbol{\alpha}, N_e \leftarrow f_H(\boldsymbol{x}, N_e)$ {Evaluate and update total cost}
  $A_H \leftarrow A_H \cup \{\boldsymbol{\alpha}\}$ {Add $\boldsymbol{\alpha}$ to high-fidelity archive}
  $\boldsymbol{x}^\beta \leftarrow \min(f_H(\boldsymbol{x}^\beta), f_H(\boldsymbol{x}))$ {Update best solution}
end while

The functions here have the same definitions as in Algorithm 2. This procedure iteratively updates a co-kriging model by randomly sampling the low-fidelity data using a latin hypercube sampling (LHS) technique; however, the global search method remains the same.

D. MO^2TOS algorithm

The second method used for comparison with $MFITS$ is the MO^2TOS algorithm described in Section II. No source code for this algorithm was available online, so an implementation based on [18] was made, the pseudocode for which is given in Algorithm 5.

Algorithm 5: MO^2TOS procedure

Input: P, problem data; $N_{e_{max}}$, max LF equivalent evals; n_0, initial HF evals; ρ, HF eval proportion; k, number of partitions; Δ, HF evals per statistics update.
Output: Best solution found $\boldsymbol{x}^\beta$.

$X_L \leftarrow LHS(P, N_{e_{max}}, \rho)$ {Generate initial population}
$A_L, N_e \leftarrow f_L(X_L, 0)$ {Evaluate initial LF population}
$G \leftarrow Partition(A_L, k)$ {Partition solutions into k groups}
$D_i \leftarrow randSelect(G_i, n_0),\ \forall i \in [k]$ {Select initial HF candidates}
$S_i, N_e \leftarrow f_L(D_i, N_e),\ \forall i \in [k]$ {Evaluate selected solutions}
$G_i \leftarrow G_i \setminus D_i,\ \forall i \in [k]$ {Selection without replacement}
while $N_e < N_{e_{max}}$ **do**
  $\hat{\mu}_i, \hat{\sigma}_i \leftarrow$ sample statistics for $S_i, \forall i \in [k]$
  $R \leftarrow GetRatios(\hat{\boldsymbol{\mu}}, \hat{\boldsymbol{\sigma}})$ {compute allocation ratios}
  $D \leftarrow Allocate(R, S, G) : \sum_{i \in [k]} |D_i| = \Delta$ {Allocate solutions}
  $D_i, N_e \leftarrow f_L(D_i, N_e),\ \forall i \in [k]$ {Evaluate allocated solutions}
  $S_i \leftarrow S_i \cup D_i,\ \forall i \in [k]$ {Add to selected solutions}
  $G_i \leftarrow G_i \setminus D_i,\ \forall i \in [k]$ {Selection without replacement}
end while
$\boldsymbol{x}^\beta \leftarrow \min\left(\bigcup_{i \in [k]} S_i\right)$ {Find best selected solution}

Here, most of the subroutines are defined the same as those described in Sections II and III, except for a couple. The total computing budget, $N_{e_{max}}$, must be divided between the initial low-fidelity evaluations and the subsequent high-fidelity ones, with the proportion of the total computing budget allocated to the high-fidelity evaluations given by the parameter ρ. In [18], Xu et al. do not provide any guidelines for setting this parameter, however sensitivity analysis experiments showed that a value of around 0.25 for ρ provided the best results for the given problem instances and computing budget.

The function $LHS(P, N, \rho)$ takes the problem instance, total budget and ρ as input and returns an initial population of solutions sampled using latin hypercube sampling (LHS) with size $\frac{N}{1+\rho}$. This population is evaluated in low-fidelity and partitioned using $Partition(A, k)$ which takes an archive of solutions and a number of groups, ranks the solutions by objective value and returns k equal-size groups based on their ranks. Xu et al. used $k = 10$ for their experiments. The remainder of the procedure is similar to Algorithm 3, with $\Delta = 5$ corresponding to the value of Δ_2 in that algorithm.

V. Results and Discussion

The results presented in this section are divided into the two datasets as described in Section IV. Dataset A are taken from [14] and demonstrate the performance of $MFITS$ on a diversity of problems with different sizes; while dataset B investigates the effect of problem size and error type.

Dataset A

Table I gives the results from 30 runs of $MFITS$ and the base-line co-kriging algorithm on problems $f10$ to $f17$ from [14], comprising dataset A.

In this table, it can be seen that $MFITS$ performs as well as, or better than, the base-line co-kriging algorithm, with a few exceptions. For the tests on instances $f11$, $f14$ and $f16$ the base-line co-kriging algorithm was able to find at least one solution that was better than the best solution produced by $MFITS$, and for the instances $f10$ and $f14$ it produced better solutions on average. In all of these instances, the differences between the algorithms are small, with both performing very similarly; whereas $MFITS$ outperforms the co-kriging by some margin in a lot of the other instances.

This is supported by the convergence plots in Figure 5. Here, it is clear that when the size of the problem is small, the performance of the two algorithms is very similar, with similar convergence rates; but as the size increases, the both the average performance, and convergence rate, of $MFITS$ over co-kriging improves significantly — with the most marked improvement being for problem $f17$.

Dataset B

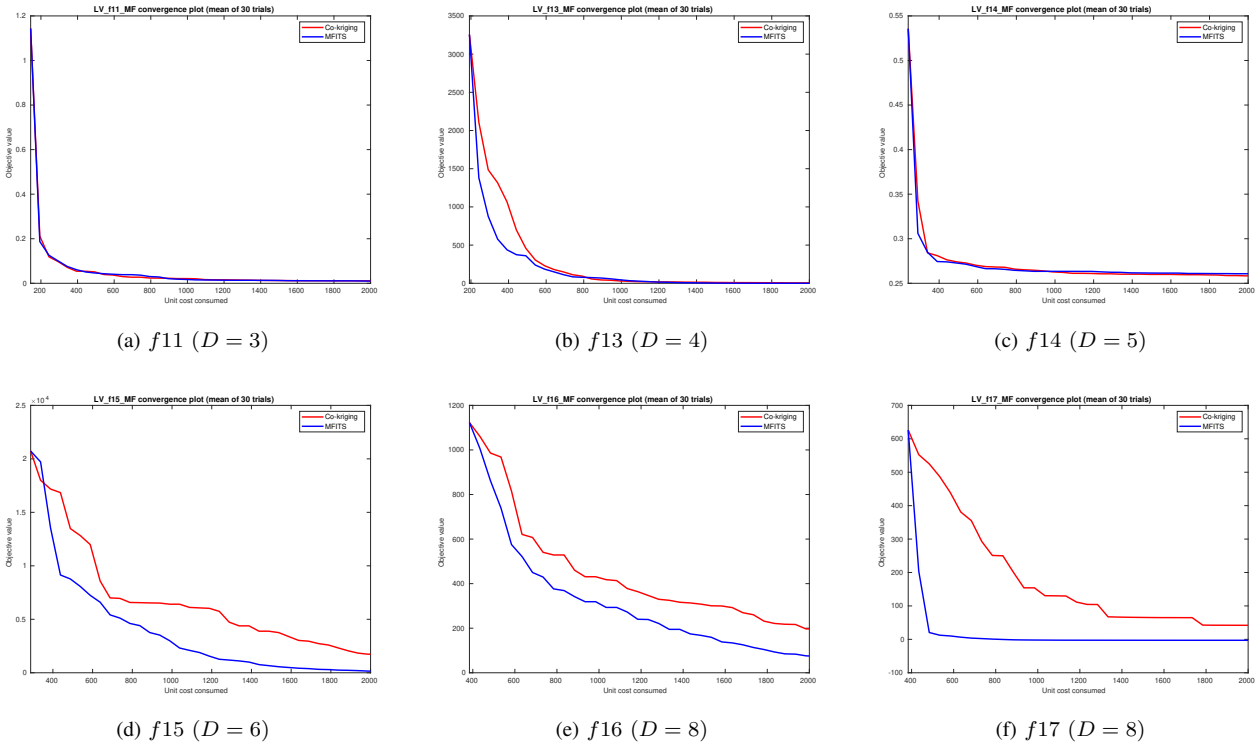
The purpose of dataset B was to observe the effect of problem size on the performance of $MFITS$, and to do so using two different error functions to ensure the results obtained are consistent, and not dependent on the simulation error model used.

The results in Table II compare the performance of $MFITS$ and the baseline co-kriging algorithm and are taken over 30 runs on two problem instances from the literature, with varying numbers of decision variables using error functions e_2 and e_6 as described in [25].

The table shows that in the majority of instances $MFITS$ outperforms the co-kriging baseline algorithm, with a few exceptions. For the Michaelwicz problem using error function e_6 and has been instantiated with five decision variables, the co-kriging base-line algorithm managed to find the best over-all solution, however $MFITS$ still performed better on average. The only time when the co-kriging base-line outperformed $MFITS$ on average was the two smallest instantiations of the Griewank problem. The convergence plots in Figures 6a and 6d show that the difference between the performance of both algorithms for these instances is reasonably negligible. The remaining plots in Figure 6 agree with the results on dataset A.

TABLE I: Results on dataset A comparing the base-line co-kriging algorithm to $MFITS$. Given are the number of decision variables (D), the square of the Pearson correlation coefficient (r^2), the best objective obtained, the mean best objective over the full set of runs (μ) and the corresponding standard deviation (σ).

			Co-kriging			MO^2TOS			$MFITS$		
Instance	D	r^2	best	μ	σ	best	μ	σ	best	μ	σ
$f10$	3	0.64	**0**	**2.2960**	3.5445	4.9076	29.6201	33.3903	**0**	3.9189	5.3296
$f11$	3	0.74	**0.0001**	0.0110	0.0077	0.0199	0.2043	0.1710	0.0004	**0.0098**	0.0064
$f12$	4	0.79	-8.1107	-3.8007	1.3426	-3.8370	-1.7674	0.7283	**-9.5783**	**-5.8853**	1.5123
$f13$	4	0.89	0.6290	4.9366	5.0965	14.0887	159.2747	180.3111	**0.0519**	**0.3457**	0.1971
$f14$	5	0.75	**0.2509**	**0.2583**	0.0039	0.2815	0.4025	0.0605	0.2522	0.2607	0.0037
$f15$	6	0.78	104.2304	1700.28	2015.04	894.6061	6069.7722	4689.5802	**24.6278**	**152.9817**	144.6451
$f16$	8	0.82	**7.3904**	196.810	171.7771	152.3790	502.9491	261.0805	7.9240	**75.2898**	59.3423
$f17$	8	0.79	-2.5859	42.0074	153.4414	87.9290	293.7373	104.7241	**-3.0161**	**-2.8355**	0.0967



(a) $f11$ ($D = 3$) (b) $f13$ ($D = 4$) (c) $f14$ ($D = 5$)

(d) $f15$ ($D = 6$) (e) $f16$ ($D = 8$) (f) $f17$ ($D = 8$)

Fig. 5: Mean convergence plots for problem instances $f11$, $f13$, $f14$, $f15$, $f16$ and $f17$ of dataset A, over 30 runs. [Angus: will do pgfplots version]

Summary

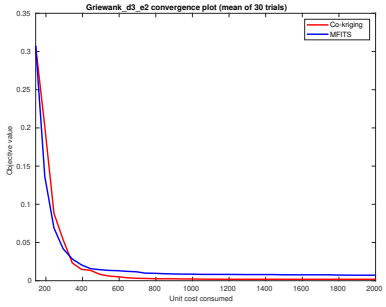
The results from both datasets suggest a similar pattern. For problems with few decision variables, there is very little difference between $MFITS$ and the base-line co-kriging algorithm, and as the size of the problem increases, the convergence rate and performance of $MFITS$ improves over the simple co-kriging algorithm with the greatest improvements being observed on the largest problem instances.

Although both algorithms contain the co-kriging technique as their principal constituent, the base-line algorithm uses a random sampling technique which gives it a disadvantage in problems with more decision variables, as it must cover an exponentially larger space with the same computational budget. By using the $LocalOCBA$ procedure to focus the sampling on areas of the search space that have been identified as promising, $MFITS$ can use its budget more efficently to exploit these regions, which is evidenced by the faster convergence rates at larger problem sizes.

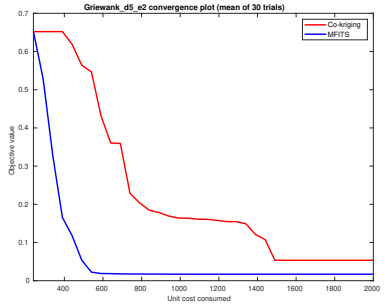
Finally, although the numerical results and convergence plots do indicate that there is not a significant difference between the performance of the two algorithms on the smaller instances; it is worth noting that for three out of four of these instances, the co-kriging baseline did perform slightly better on average. One possible explanation for this is that the $LocalOCBA$ procedure makes $MFITS$ more greedy, and therefore more likely to become trapped in local optima — especially for multimodal instances like the Griewank problem (as demonstrated in Figure 3). In these

TABLE II: Results on Griewank and Michalewicz test problems using Wang error functions 2 and 6 (indicated by subscript), comparing the base-line co-kriging algorithm to $MFITS$. Given are the number of decision variables (D), the square of the Pearson correlation coefficient (r^2), the best objective obtained, the mean best objective over the full set of runs (μ) and the corresponding standard deviation (σ).

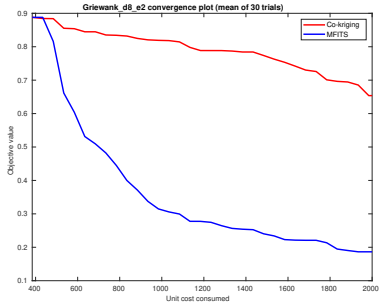
			Co-kriging			MO^2TOS			$MFITS$		
Instance	D	r^2	best	μ	σ	best	μ	σ	best	μ	σ
$Griewank_2$	3	0.73	**0**	**0.0018**	0.0083	0.0019	0.0735	0.0440	**0**	0.0072	0.0137
	5	0.54	**0**	0.0535	0.0733	0.1428	0.3886	0.1245	**0**	**0.0485**	0.1055
	8	0.37	**0**	0.6538	0.3363	0.5606	0.8036	0.1100	**0**	**0.1864**	0.2531
$Griewank_6$	3	0.74	**0**	**0.0056**	0.0114	0.0076	0.0564	0.0290	**0**	0.0104	0.0130
	5	0.76	**0**	0.0730	0.1145	0.1139	0.3792	0.1328	**0**	**0.0361**	0.0778
	8	0.76	**0**	0.4560	0.4238	0.5565	0.8108	0.1249	**0**	**0.2089**	0.3539
$Michalewicz_2$	3	0.77	-2.7239	-2.3582	0.2587	-2.4612	-1.8939	0.2366	**-2.7360**	**-2.5305**	0.1557
	5	0.73	-3.5164	-2.9470	0.2921	-3.2841	-2.4554	0.3928	**-3.5653**	**-2.9953**	0.3219
	8	0.64	-4.1813	-3.0214	0.5671	-4.2150	-3.0818	0.7827	**-4.5542**	**-3.3046**	0.4142
$Michalewicz_6$	3	0.76	-2.7194	-2.2412	0.2978	-2.4657	-1.8847	0.2927	**-2.7409**	**-2.3142**	0.2411
	5	0.83	**-3.6198**	-2.7140	0.3948	-3.2980	-2.5380	0.3328	-3.5511	**-2.9104**	0.2823
	8	0.86	-3.9978	-3.1032	0.4929	-3.9627	-3.1363	0.3851	**-4.2922**	**-3.1672**	0.0909



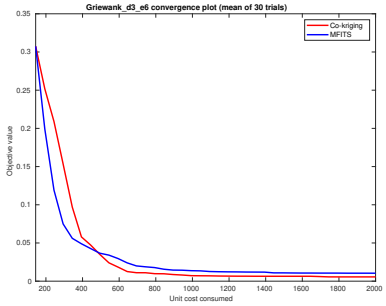
(a) $Griewank_2$ $(D = 3)$



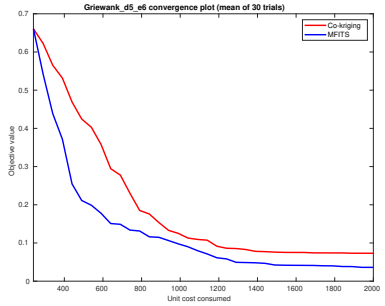
(b) $Griewank_2$ $(D = 5)$



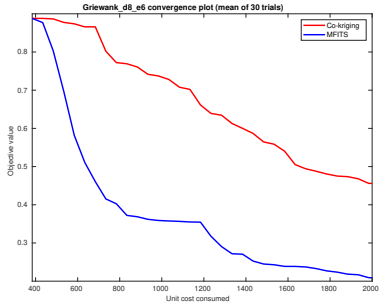
(c) $Griewank_2$ $(D = 8)$



(d) $Griewank_6$ $(D = 3)$



(e) $Griewank_6$ $(D = 5)$



(f) $Griewank_6$ $(D = 8)$

Fig. 6: Mean convergence plots for $Griewank$ problem instance of dataset B, over 30 runs. [Angus: will do pgfplots version]

smaller instances, the random sampling of the base-line co-kriging algorithm is not as much of a disadvantage and it is more likely to come across good areas of the search space by chance.

VI. Conclusion and Future Work

Real-world engineering design problems can be very complex, requiring computationally intensive simulation-based techniques to optimize (SO). One property of SO techiniques is that they often allow the fidelity of output results to be specified, with lower-fidelity solutions comsuming less of the allowed computational budget.

This paper presented a multi-fidelity simulation-based optimization algorithm designed to solve bound-constrained, single-objective problems called $MFITS$. It is an iterative, two-stage process, built on the commonly-used co-kriging method, which updates its model by employing a modified version of the optimal computing budget allocation algorithm which samples from a restricted neighbourhood.

In order to demonstrate the effectiveness of the im-

proved sampling technique of $MFITS$, numerical tests were carried out that compared it to a base-line co-kriging implementation, which updates its model using random sampling. The experiments were conducted on two separate datasets, one multi-fidelity dataset from the literature, and one dataset made by applying a generic method used to model simulation errors to two standard test functions. The experiments on the first dataset demonstrated the versatility of $MFITS$ on a variety of test functions, while those on the second dataset investigated the effects of problem size and error type on $MFITS$.

The results of the numerical experiments confirmed that $MFITS$ produced solutions that were competitive with, or superior to, the base-line co-kriging algorithm across all instances. The main finding illustrated by the results is that by sampling the search space more selectively when updating its surrogate model, $MFITS$ is more efficient in its computing budget use, resulting in faster convergence rates — especially as the size of the problem increases.

Further research in this direction could include investigating the use of different global search techniques to optimize the co-kriging model and applying it to a wider range of test functions and real-world problems, including problems with integer and categorical decision variables, and linear and non-linear constraints.

APPENDIX A
FUNCTION EQUATIONS

Tables III and IV below give the equations for the test functions and error functions, respectively.

REFERENCES

[1] A. Forrester, A. Sobester, and A. Keane, *Engineering design via surrogate modelling: a practical guide*. John Wiley & Sons, 2008.

[2] Y. Jin and B. Sendhoff, "A systems approach to evolutionary multi-objective structural optimization and beyond," *IEEE Computational Intelligence Magazine*, vol. 4, no. 3, pp. 62–76, 2009.

[3] S. Amaran, N. V. Sahinidis, B. Sharda, and S. J. Bury, "Simulation optimization: a review of algorithms and applications," *Annals of Operations Research*, vol. 240, no. 1, pp. 351–380, 2016.

[4] J. Branke, M. Asafuddoula, K. S. Bhattacharjee, and T. Ray, "Efficient use of partially converged simulations in evolutionary optimization," *IEEE Transactions on Evolutionary Computation*, vol. 21, no. 1, pp. 52–64, 2016.

[5] D. J. Toal, "Some considerations regarding the use of multi-fidelity kriging in the construction of surrogate models," *Structural and Multidisciplinary Optimization*, vol. 51, no. 6, pp. 1223–1245, 2015.

[6] A. I. Forrester, A. Sóbester, and A. J. Keane, "Multi-fidelity optimization via surrogate modelling," *Proceedings of the royal society a: mathematical, physical and engineering sciences*, vol. 463, no. 2088, pp. 3251–3269, 2007.

[7] M. C. Kennedy and A. O'Hagan, "Predicting the output from a complex computer code when fast approximations are available," *Biometrika*, vol. 87, no. 1, pp. 1–13, 2000.

[8] J. Laurenceau and P. Sagaut, "Building efficient response surfaces of aerodynamic functions with kriging and cokriging," *AIAA journal*, vol. 46, no. 2, pp. 498–507, 2008.

[9] L. Huang, Z. Gao, and D. Zhang, "Research on multi-fidelity aerodynamic optimization methods," *Chinese Journal of Aeronautics*, vol. 26, no. 2, pp. 279–286, 2013.

[10] P. Perdikaris, D. Venturi, J. O. Royset, and G. E. Karniadakis, "Multi-fidelity modelling via recursive co-kriging and gaussian–markov random fields," *Proceedings of the Royal Society A: Mathematical, Physical and Engineering Sciences*, vol. 471, no. 2179, p. 20150018, 2015.

[11] X. Yang, D. Barajas-Solano, G. Tartakovsky, and A. M. Tartakovsky, "Physics-informed cokriging: A gaussian-process-regression-based multifidelity method for data-model convergence," *Journal of Computational Physics*, vol. 395, pp. 410–431, 2019.

[12] R. Giraldo, L. Herrera, and V. Leiva, "Cokriging prediction using as secondary variable a functional random field with application in environmental pollution," *Mathematics*, vol. 8, no. 8, p. 1305, 2020.

[13] M. Goulard and M. Voltz, "Geostatistical interpolation of curves: a case study in soil science," in *Geostatistics Tróia'92*. Springer, 1993, pp. 805–816.

[14] L. Lv, C. Zong, C. Zhang, X. Song, and W. Sun, "Multi-fidelity surrogate model based on canonical correlation analysis and least squares," *Journal of Mechanical Design*, vol. 143, no. 2, p. 021705, 2021.

[15] A. Ariyarit and M. Kanazaki, "Multi-fidelity multi-objective efficient global optimization applied to airfoil design problems," *Applied Sciences*, vol. 7, no. 12, p. 1318, 2017.

[16] A. Hebbal, L. Brevault, M. Balesdent, E.-G. Talbi, and N. Melab, "Multi-fidelity modeling with different input domain definitions using deep gaussian processes," *Structural and Multidisciplinary Optimization*, vol. 63, no. 5, pp. 2267–2288, 2021.

[17] K. Cutajar, M. Pullin, A. Damianou, N. Lawrence, and J. González, "Deep gaussian processes for multi-fidelity modeling," *arXiv preprint arXiv:1903.07320*, 2019.

[18] J. Xu, S. Zhang, E. Huang, C.-H. Chen, L. H. Lee, and N. Celik, "Mo2tos: Multi-fidelity optimization with ordinal transformation and optimal sampling," *Asia-Pacific Journal of Operational Research*, vol. 33, no. 03, p. 1650017, 2016.

[19] C.-H. Chen and L. H. Lee, *Stochastic simulation optimization: an optimal computing budget allocation*. World scientific, 2011, vol. 1.

[20] D. Lim, Y.-S. Ong, Y. Jin, and B. Sendhoff, "Evolutionary optimization with dynamic fidelity computational models," in *International Conference on Intelligent Computing*. Springer, 2008, pp. 235–242.

[21] D. E. Bryson and M. P. Rumpfkeil, "Multifidelity quasi-newton method for design optimization," *AIAA Journal*, vol. 56, no. 10, pp. 4074–4086, 2018.

[22] L. W. Ng and K. E. Willcox, "Multifidelity approaches for optimization under uncertainty," *International Journal for numerical methods in Engineering*, vol. 100, no. 10, pp. 746–772, 2014.

[23] R. Storn and 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.

[24] K. Deb, *Optimization for engineering design: Algorithms and examples*. PHI Learning Pvt. Ltd., 2012.

[25] H. Wang, Y. Jin, and J. Doherty, "A generic test suite for evolutionary multifidelity optimization," *IEEE Transactions on Evolutionary Computation*, vol. 22, no. 6, pp. 836–850, 2017.

[26] M. C. Kennedy and A. O'Hagan, "Bayesian calibration of computer models," *Journal of the Royal Statistical Society: Series B (Statistical Methodology)*, vol. 63, no. 3, pp. 425–464, 2001.

[27] J. Xu, S. Zhang, E. Huang, C.-H. Chen, L. H. Lee, and N. Celik, "An ordinal transformation framework for multi-fidelity simulation optimization," in *2014 IEEE International Conference on Automation Science and Engineering (CASE)*. IEEE, 2014, pp. 385–390.

[28] I. Couckuyt, T. Dhaene, and P. Demeester, "oodace toolbox: A flexible object-oriented kriging implementation," *Journal of Machine Learning Research*, vol. 15, pp. 3183–3186, 2014.

[29] A. O. Griewank, "Generalized descent for global optimization," *Journal of optimization theory and applications*, vol. 34, no. 1, pp. 11–39, 1981.

[30] Z. Michalewicz, *Genetic algorithms+ data structures= evolution programs*. Springer Science & Business Media, 2013.

TABLE III: Test functions used for experiments in this paper. Given are the instance name, the function, the number of decision variables (D), the bounds (B) and the reference the test function is taken from.

Instance	Function	D	B	Ref.
$f10$	$f_H(\boldsymbol{x}) = 100\left(\exp\left(-\frac{2}{x_1^{1.75}}\right) + \exp\left(-\frac{2}{x_2^{1.5}}\right) + \exp\left(-\frac{2}{x_3^{1.25}}\right)\right)$ $f_L(\boldsymbol{x}) = 100\left(\exp\left(-\frac{2}{x_1^{1.75}}\right) + \exp\left(-\frac{2}{x_2^{1.5}}\right)\right)$	3	$[0,1]^D$	[14]
$f11$	$f_H(\boldsymbol{x}) = 4\left(x_1 - 2 + 8x_2 - 8x_2^2\right)^2 + (3-4x_2)^2 + 16\sqrt{x_3+1}\,(2x_3-1)^2$ $f_L(\boldsymbol{x}) = 4\left(x_1 - 2 + 8x_2 - 8x_2^2\right)^2 + (3-4x_2)^2 + 5\sqrt{x_3+1}\,(2x_3-1)^2$	3	$[0,1]^D$	[14]
$f12$	$f_H(\boldsymbol{x}) = -\sum_{i=1}^{m}\left(\sum_{j=1}^{4}(x_j - C_{ji})^2 + \beta_i\right)^{-1}$ $f_L(\boldsymbol{x}) = -\sum_{i=1}^{m}\left(\sum_{j=1}^{4}(x_j - C_{ji})^2 + 0.9\beta_i\right)^{-1}$ $m = 10, \quad \beta = \frac{1}{10}(1,2,2,4,4,6,3,7,5,5)^T$ $C = \begin{pmatrix} 4.0 & 1.0 & 8.0 & 6.0 & 3.0 & 2.0 & 5.0 & 8.0 & 6.0 & 7.0 \\ 4.0 & 1.0 & 8.0 & 6.0 & 7.0 & 9.0 & 3.0 & 1.0 & 2.0 & 3.6 \\ 4.0 & 1.0 & 8.0 & 6.0 & 3.0 & 2.0 & 5.0 & 8.0 & 6.0 & 7.0 \\ 4.0 & 1.0 & 8.0 & 6.0 & 7.0 & 9.0 & 3.0 & 1.0 & 2.0 & 3.6 \end{pmatrix}$	4	$[0,10]^D$	[14]
$f13$	$f_H(\boldsymbol{x}) = (x_1-1)^2 + \sum_{i=2}^{d} i\left(2x_i^2 - x_{i-1}\right)^2$ $f_L(\boldsymbol{x}) = (x_1-1)^2 + x_2^4 + 4x_3^4 + 4x_4^4$	4	$[-10,10]^D$	[14]
$f14$	$f_H(\boldsymbol{x}) = \sum_{i=1}^{5}\left(0.3 + \sin\left(\frac{16}{15}x_i - 1\right) + \sin\left(\frac{16}{15}x_i - 1\right)^2\right)$ $f_L(\boldsymbol{x}) = \sum_{i=1}^{5}\left(0.3 + \sin\left(\frac{13}{15}x_i - 1\right) + \sin\left(\frac{13}{15}x_i - 1\right)^2\right)$	5	$[-1,1]^D$	[14]
$f15$	$f_H(\boldsymbol{x}) = \sum_{i=1}^{5}\left(100\left(x_{i+1} - x_i^2\right)^2 + (x_i - 1)^2\right)$ $f_L(\boldsymbol{x}) = \sum_{i=1}^{5}\left(100\left(x_{i+1} - x_i^2\right)^2 + 4\,(x_i - 1)^4\right)$	6	$[0,1]^D$	[14]
$f16$	$f_H(\boldsymbol{x}) = \sum_{i=1}^{2}\begin{bmatrix}(4x_{4i-3} - 10x_{4i-2})^2 + 5\,(x_{4i-1} - x_{4i})^2 + \\ (x_{4i-2} - 2x_{4i-1})^4 + 10\,(x_{4i-3} - x_{4i})^2\end{bmatrix}$ $f_L(\boldsymbol{x}) = \sum_{i=1}^{2}\begin{bmatrix}(4x_{4i-3} - 10x_{4i-2})^2 + 5\,(x_{4i-1} - x_{4i})^2 + \\ (x_{4i-2} - 2x_{4i-1})^4 + 4\,(x_{4i-3} - x_{4i})^2\end{bmatrix}$	8	$[-4,5]^D$	[14]
$f17$	$f_H(\boldsymbol{x}) = \sum_{i=1}^{8}\left(x_i^4 - 16x_i^2 + 5x_i\right)$ $f_L(\boldsymbol{x}) = \sum_{i=1}^{8}\left(0.8x_i^4 - 16x_i^2 + 5x_i\right)$	8	$[-5,5]^D$	[14]
Griewank	$f_H(\boldsymbol{x}) = \sum_{i=1}^{D}\frac{x_i^2}{4000} - \prod_{i=1}^{D}\cos\left(\frac{x_i}{\sqrt{i}}\right) + 1$	$\{3,5,8\}$	$[-5,5]^D$	[29]
Michalewicz	$f_H(\boldsymbol{x}) = \sum_{i=1}^{D}\sin(x_i)\sin^{2m}\left(\frac{ix_i^2}{\pi}\right), \quad m = 10$	$\{3,5,8\}$	$[0,\pi]^D$	[30]

TABLE IV: Error functions used, as defined in Wang et al. [25]. Function e_2 models resolution errors and e_6 models stochastic errors. Here, $\phi \in [0, 10000]$ determines the fidelity level and D is the number of decision variables.

Name	Function
e_2	$e_2(\boldsymbol{x},\phi) = \sum_{i=1}^{D} a(\phi)\cos\left(w(\phi)x_i + b(\phi) + \pi\right)$ where, $a(\phi) = \theta(\phi),\ w(\phi) = 10\pi\theta(\phi),\ b(\phi) = 0.5\pi\theta(\phi),\ \theta(\phi) = e^{-0.00025\phi}$
e_6	$e_6 = \mathcal{N}\left(\mu(\boldsymbol{x},\phi), \sigma(\phi)\right)$ where, $\mu^2(\phi) = 0,\ \sigma(\phi) = 0.1\vartheta(\phi),\ \vartheta(\phi) = e^{-0.0005\phi}$