A Framework for Evaluating Meta-models for Simulation-based Optimisation

Pepijn van Heiningen
LIACS, Leiden University
Niels Bohrweg 1, 2333 CA
Leiden, The Netherlands
pepijnvanheiningen@hetnet.nl

Bas van Stein
Natural Computing Group
LIACS, Leiden University
Niels Bohrweg 1, 2333 CA
Leiden, The Netherlands
b.van.stein@liacs.leidenuniv.nl

Thomas Bäck
Natural Computing Group
LIACS, Leiden University
Niels Bohrweg 1, 2333 CA
Leiden, The Netherlands
t.h.w.baeck@liacs.leidenuniv.nl

Abstract—**In order to evaluate complex and computationally expensive experiments, data-driven *meta-models* are used to replace costly experiments and approximate the real experiments' outcome. In this study, an evaluation framework is proposed for measuring the performance of these models. Explored is how the *performance* is related to the difference between the benchmark optimum and the model optimum, and a novel method to measure the performance of this '*optima-fit*' is proposed. The evaluation framework is presented using an experimental setup of four meta-modeling techniques (Decision Tree, Random Forests, Support Vector Regression and Kriging), which are systematically compared to each other. The techniques are fitted to eleven benchmarks, with dimensionality ranging from 2 to 32. The meta-models are trained with varying input sample sizes and sampling strategies. In addition, the relations between the performance to the various sampling strategies and sizes on these benchmarks are explored. Our novel designed metric can provide additional insight in the performance of a specific range of meta-models.**
Index Terms—**Meta-models; Evaluation; Optimisation.**

I. Introduction

In competitive engineering environments such as the automotive industry, the product designs are often optimized through non-linear, global optimizers where the goodness of designs are determined by costly physical experiments or heavy computer codes (simulators).

However, these simulations are often expensive to evaluate. Due to the high time complexity, these simulations may only lead to a trial-and-error approach where the designer, instead of deliberately changing variables to optimize the design, tries varying parameters until a sufficiently good solution is found (see Simpson et al. [21]).

For example; most (if not all) major automotive companies use computer simulations to analyze and test the safety of their new models. According to the researchers at Ford, simulation time can take from six to twenty hours (see Akkerman [1]).

The enormous computational cost of some tests makes it infeasible for state-of-the-art optimizers, which usually take up many simulation runs (Jin et al. [10]). Developing optimization algorithms that can produce decent results in a limited number of evaluations is a great challenge. Considering the previously mentioned example of a car crash: even with a limited budget of only 50 iterations and under the assumption that every iteration needs only one simulation, a minimum of 300 hours of simulation time would be required.

One way to solve this problem is by building *meta-models* (models of the simulation models) to provide approximations of these simulations. Provided that they are sufficiently *well-fitted* (providing predictions close to the truth, even on unseen data), these models are able to replace the simulation models, and are especially useful in the early phases of the design process, where many different blueprints can be tested and designs are still optimized on a larger scale. Once a suitable design has been found, the actual simulation could then be used to perform the final, more locally oriented optimization. The major advantage of this approach is the speed-up obtained by leaving out simulation, allowing for more evaluations by the optimization algorithms. Other benefits are easier integration of optimization algorithms on different problem domains, and a better understanding of the relationship between input variables and their output (see Simpson et al. [21]).

Using meta-models for optimization is called *Meta-model-based Design Optimization* (MBDO) (see e.g. Ryberg [19]). Many different variants of meta-models, experimental designs and optimization methods exist and have been tested. Subsequently, a great deal of studies comparing different meta-models with varying dimensions or sampling techniques have been performed (see Jin et al. [10], Shi et al. [20], Fang et al. [6], Li et al. [14], Kim et al. [13] etc.). Jin et al. [10] note in their future work that more test problems with large dimensions, as well as with more medium dimensions should be tested. Furthermore, the quality of the meta-models with respect to optimization algorithms were tested by Li et al. [14]. Since one of the main usages of these meta-models is optimization, it is important to study this more thoroughly. A well-fitted model might not be the best to optimize on, if the global optimum of the model does not correspond with the true global optimum. On the contrary, a simple model which is unable to represent the whole space well, might have a good fit in the neighbourhood of the location of the global optimum.

In the literature, there seems to be only limited consensus on what the correct number of samples is in order to train a model properly. In Akkerman et al. [1] an initial sampling size between $3N$ and $4N$ is recommended, with N being the number of dimensions of the problem. However, Jones et al. [12] take approximately $10N$ points in the initialization of their model, although it is only used as an initial step in their proposed algorithm. More thorough work is done by Shi [20], who concludes that $3N$ is a good choice for sample size when considering computational cost. Simpson et al. [22] conducted several tests with different sample sizes and different models, but since they only apply their methods on two problems, no solid general conclusions can be drawn. Finally, only a few comparisons have been made, where besides the number of samples, also the number of dimensions and models were tested on a set of representative benchmarks.

Contributions: a framework for measuring the performance of meta-models with respect to optimization is proposed, taking both the meta-models' optimum and the true optimum into account. In addition, a new metric is introduced to give more insight in the performance for a particular meta-model. One of our goals is to give a well argued recommendation for sample sizes and investigate what effect the distribution of samples has on the performance of models. In Section II we will present the methods studied in this paper in a more formal way. The approach taken in our experiments is highlighted in Section IV and the results are discussed in Section V.

II. Related research

When designing meta-models for optimization algorithms, typically two choices have to be made:
1) the selection of a sampling strategy, where a number of *samples* are picked in the design space, and
2) the actual meta-model itself. In this chapter the sampling strategies and meta-models used in the experimental setup are explained.

A. Sampling strategies

The design of a meta-model optimization strategy starts with the selection of an experimental design, where a number of sampling points are picked in the design space. The theory of placing design points in this space is called the *Design of Experiments*. Since in most cases only little is known about the simulation function that needs to be modeled, some strategies try to fill the space, by spreading out sampling points as evenly as possible. These methods are referred to as 'space-filling' designs. Examples are: *Latin Hypercube sampling* (McKay et al. [15]), *minimax* and *maximin* designs (Johnson et al. [11]) and *orthogonal arrays* (Owen et al. [16]). However, little is known from literature as to what performance can be obtained when the distribution of samples is not space-filling or even uniformly distributed. In a real-world scenario, it might not always be possible to select design points in advance due to budget or time constraints. When this is the case, one would have to work with the existing data that are not necessarily evenly spaced across the search space. Therefore, only a single space-filling distribution of samples is selected in our experiments, and compared with theoretically less favorable distributions in order to determine the effect of the following sampling strategies.

1) Uniform Sampling: Samples are drawn from a uniform distribution within a range between values a and b for each dimension. The probability density function of the uniform distribution is:

$$p(x) = \frac{1}{b-a}, \tag{1}$$

between the values of a and b, and zero everywhere else. Uniform sampling is used in our experiments as one of the sampling strategies, because it is the most naïve and simple way to select samples spread over the entire domain. However, due to the random nature there is no guarantee that the discrepancy of the sample is low.

2) Normal Sampling: Another sampling technique is normal random sampling, where samples are drawn from a normal distribution, with $\sigma = 1$ and $\mu = 0$ (the data is normalized). Selected samples are multiplied by $\frac{1}{4}$, such that the probability of selecting samples outside the given range is almost 0. The probability density function of the Gaussian distribution in 1D:

$$p(x) = \frac{1}{\sqrt{2\pi\sigma^2}} \cdot e^{-\frac{(x-\mu)^2}{2\sigma^2}} \tag{2}$$

where μ is the mean, and σ is the standard deviation. Normal random sampling simulates datasets where samples show clustering patterns. This might influence the accuracy of the model.

3) Latin Hypercube Sampling: Latin Hypercube sampling was first proposed by McKay, Conover and Beckman [15]. It ensures that each sample is the only one in each axis-aligned hyperplane. The co-domain of each random variable X_i is split into N strata of equal probability, and on each stratum one sample is generated uniformly. This works especially well when only a few samples are needed. Latin Hypercube sampling is one of the most well-known and often-used space-filling sampling methods, spreading the samples over the design space. This method is expected to give the best results.

4) Normal-centered sampling: Samples are drawn from a normal distribution, with $\sigma = 1$ and $\mu = 0$. However, this time the value is set to $\mu = opt$ per dimension, where *opt* is the value of where the optimum is located on that axis. This ensures that samples are centered on the optimum of our benchmark. Note that this would not be possible in real-world scenarios where the optimum of the benchmark is unknown. Normal-centered sampling is included in order to test the results when samples are generated around the optimum of the benchmark functions.

5) Wald-centered sampling (inverse Gaussian): This technique draws samples from a Wald distribution, with $\mu = 1, \lambda = 10$.

$$P(x; \mu, \lambda) = \sqrt{\frac{\lambda}{2\pi x^3}} \cdot e^{\frac{-\lambda(x-\mu)^2}{2 \cdot \mu^2 \cdot x}} \tag{3}$$

Again, values are scaled by multiplying by $\frac{1}{4}$. Since the Wald distribution is strictly positive, the value of our sample is negated in 50% of cases. Finally, the 'center' of our distribution is shifted to the optimum of the tested benchmark, in order to test what would happen if samples are located away from the optimum. Wald sampling mimics poorly sampled design spaces, when samples are not evenly spread over the design space and are located away from the optimum.

B. Meta-models

Many different models exist that can learn from data and predict values that have not been seen before. Frequently used ones are: Neural Networks, RBF-networks, Multivariate Adaptive Regression Splines, Decision Trees, Random Forests, Kriging and Support Vector Regression. In this section we will look at four most commonly used meta-models that have been used in our experiments.

1) Decision Trees: Decision Trees (DT) are commonly used in data mining (see Alpaydin [2]). A Decision Tree model consists of nodes, where in each interior node, input values are split into two or more branches to build a tree. The *leaves*, the exterior nodes, represent the output values of the target variable, given the path from the *root* of the tree along the different splits to the leaf. Decision Trees can be used for both *classification* as well as *regression*. When target values are represented by a finite set of discrete classes, Decision Trees are called classification trees, when target values are continuous, they are called regression trees.

Trees can be learned with algorithms such as *ID3*, *CART* and *C4.5*, belonging to the class of top-down induction of decision trees (TDIDT) (see Quinlan [18]). In this process the input dataset is recursively partitioned into subsets of items. When all remaining items in a subset have the same value, the recursion stops.

Different metrics can be used to determine the splits. In classification trees, *Information Gain* [18] or the *Gini impurity* [5] can be used as the splitting criterion. In regression trees, the splits are often determined by maximum variance reduction (which is equal to minimizing the sum of squared errors S in equation 4).

$$S = \sum_{l \in L} \sum_{i \in l} (y_i - \mu_l)^2 \tag{4}$$

where L is the set of leaves of the tree, and $\mu_l = \frac{1}{n_l} \sum_{i \in l} y_i$ is the prediction for leaf L (where n_l is the number of items in leaf l).

2) Random Forest Regression: Random Forests is the name for an ensemble of learning techniques. It was first coined by Leo Breiman [4]. In Random Forests, the notion of *bagging* [3] is used. In bagging, multiple Decision Trees (or other learners) are built and combined into a 'forest' of trees. The technique of using Random Forests for regression is called *Random Forest Regression* (RFR).

To ensure differences in the tree structures, each tree depends on a random subset of selected samples. Furthermore, trees no longer split on the best split found (as usually is the case of Decision Trees), but instead on the best split found in a random subset of the features. This increases the 'randomness' of each tree, and thus reduces the biasness of the estimation. After the generation of a number of trees, all trees vote on a class (in classification), and the most popular class is picked as a result. In regression, a prediction is simply the average of the predicted outcomes of each tree.

The main advantages of Random Forests, compared to Decision Trees, are that the increased randomness counteracts over-fitting and exhibits more smooth behavior while retaining the good qualities of Decision Trees. A drawback is that they cannot be interpreted easily.

3) Support Vector Regression: The original *Support Vector Machines* (SVM) were invented by Vapnik [23] in 1963. They are well described by Alpaydin [2]. In Support Vector Machines, an optimal separating hyperplane is constructed, allowing for tasks such as classification and regression. The hyperplane is optimal when it has a maximal margin between the data-points of the classes, minimizing the *generalization error* (the error in predicting new, unseen input values). In order to construct these hyperplanes, only a limited number of samples, called the *support vectors*, are necessary. Only this (usually small) subset of samples is used to determine the margin.

In order to generalize the method to regression, the ϵ-sensitive loss function can be used. In regression, the goal is not to separate the data, but to contain all samples within a margin ϵ, and then minimize the margin. For the details, please see

Alpaydin. [2]

4) Kriging: Kriging uses *Gaussian Processes* (GP) as the assumption on the data. GP can be seen as an extension of a multivariate Gaussian distribution on infinite dimensions. The prediction of Kriging is a sum of a non-linear trend and a GP:

$$G(X) = \sum_{j=0}^{k} \beta_j f_j(x) + Z(X) \tag{5}$$

where $Z(X)$ is a centered GP with 0 mean and covariance function K. The correlation between two samples is modeled only by the covariance function K. Different covariance functions can be chosen, a popular choice is the squared exponential:

$$k(x, x') = \sigma_f^2 \exp\left[-\frac{(x-x')^2}{2l^2}\right] \tag{6}$$

In order to predict new values, the hyper-parameters of this function, such as σ and l, have to be learned from the data. This can be done with any multivariate optimization algorithm.

C. Quality Measurement

The first measure used to determine the performance of the model is the R^2 score:

$$R^2 \equiv 1 - \frac{\sum_i (y_i - f_i)^2}{\sum_i (y_i - \bar{y})^2}, \tag{7}$$

with data $y_1 \ldots y_n$, with predicted value $f_1 \ldots f_n$. $\bar{y}$ is the mean of the actual value of the data.
R^2 scores are widely used to provide an estimate of the accuracy of the model. Scores have an upper bound of 1.0, and can be negative. A constant model, predicting only the mean value of y would get an R^2-score of 0. A drawback of using the R^2 score is that it will for some regression techniques never decrease when additional variables are added to the model, which can cause irrelevant data to be learned. Since our models are fitted to mathematical benchmarks, this is not a problem, as each dimension is necessary to correctly fit the model. Therefore the R^2-score can be safely used to compare with our own metrics, presented below in Section III.

D. Optimization

In order to determine the optimum in the model, a *Covariance Matrix Adaptation Evolution Strategy* (CMA-ES) algorithm is used. CMA-ES was originally invented by Hansen (see Hansen and Ostermeier [9]). It is considered the state-of-the-art in evolutionary strategies, and widely used for black-box optimization problems.
In Algorithm 1 the pseudo-code of the CMA-ES algorithm is presented. First, a set of μ parents are initialized and evaluated. A new population of λ offspring is generated, by sampling from a multivariate normal distribution:

```
Initialize and evaluate $\mu$ initial parents
while not terminated do
    Mutate parents into $\lambda$ children
    Evaluate, select and recombine children into $\mu$ parents
    Update covariance matrix and $\sigma$
end
```

Algorithm 1: Pseudo-code of CMA-ES

$$x_k^{g+1} = m^g + \sigma^g N\left(0, C^g\right) \text{ for } k = 1, \cdots, \lambda \tag{8}$$

The best μ individuals are selected and averaged.

$$m^{g+1} = \sum_{i=0}^{\mu} w_i x_{i:\lambda}^{g+1}, \tag{9}$$

where $i:\lambda$ denotes the i-th best individual in λ. The weights w_i can be set to $\frac{1}{\mu}$. Finally the covariance matrix is updated. From selected individuals, the new mean of the next generation is computed. For a detailed explanation please see Hansen [8].

The major advantages are that it conducts an adaptive search to local function topology with properly controlled step-size on the benchmarks that are considered in this paper. This is necessary in order to find the optimum in the model, which approximates the benchmarks. It should be noted that derivative based methods that converge much faster, usually get stuck in local optima.

Our settings are as follows: 250 generations, population size of $20N$, (with N the number of dimensions), starting with $\sigma = 0.2$. In order to maintain all individuals in the feasible range of $[-1, 1]$ per dimension, the value of individuals outside the hypercube is set to 10^{10}.

III. Framework for evaluating meta-models

Commonly, the performance assessment of meta-models are made on the accuracy or some other properties. However, in simulation-optimization, these models are used to find (global) optima. Therefore, the *optima-fit* (the relation between the optimum of our model and the optimum in the benchmark) is of major importance.
As explained in Section II-C, the R^2-score can be used to provide an estimate of the accuracy of the predictions of a trained model. However, it might not accurately represent the usability of the model for optimization. Therefore the Δx and Δf metrics are proposed. Besides testing the model accuracy, the difference between the optimum in the benchmark and the optimum found in the model is compared both in the input space (Δx) and in the objective space (Δf). In Figure 1, a graphical overview is provided of the two metrics.

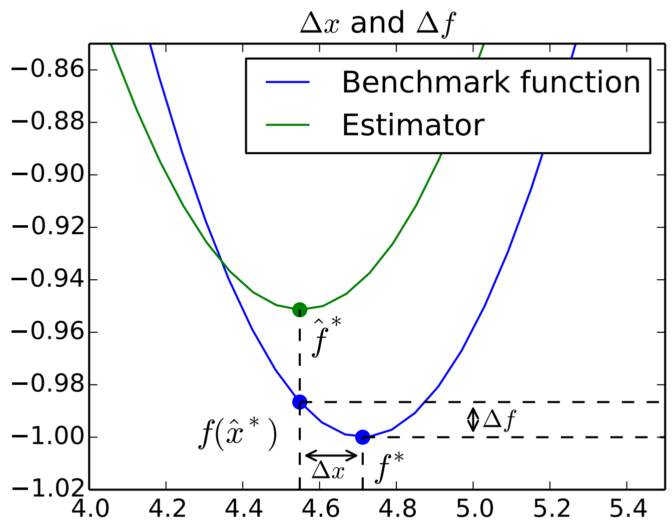


Fig. 1: Graphical interpretation of Δx and Δf

The distance between objects (or Δx) is simply the *L2 norm*, or the *Euclidean distance* between two points:

$$\Delta x = \|\hat{x}^* - x^*\| \tag{10}$$

$$= \sqrt{(\hat{x}_1^* - x_1^*)^2 + (\hat{x}_2^* - x_2^*)^2 + \cdots + (\hat{x}_n^* - x_n^*)^2} \tag{11}$$

In real-world problems, the location of the optimum is not known in advance. Therefore we are also interested in the difference in fitness (or Δf) between the estimated optimum and the real optimum on different functions, as this can give us an insight as to how close these models can come to the true value.

Δf is the difference between the fitness of the best individual found by the model, and the fitness of the actual optimal point:

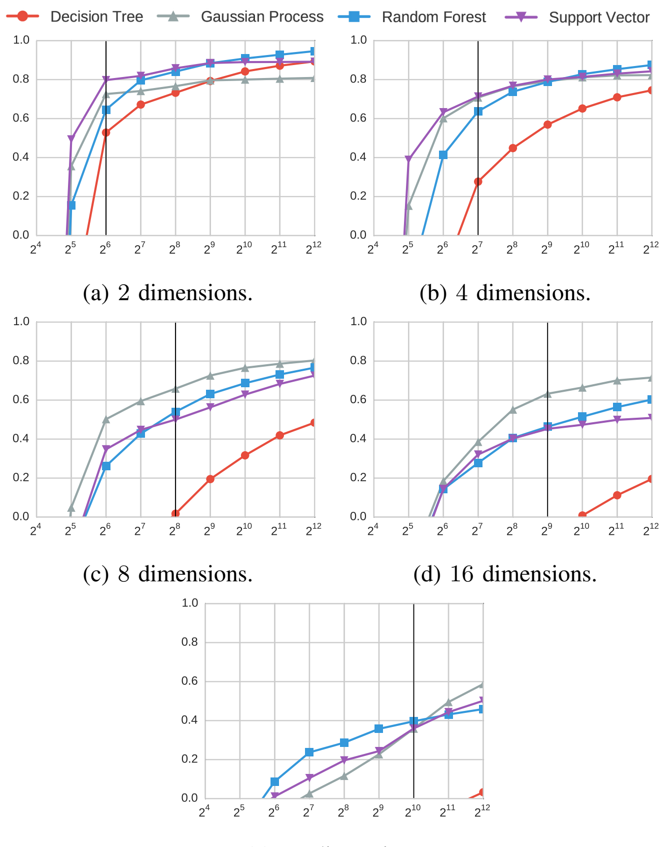
$$\Delta f = f(\hat{x}^*) - f(x^*)$$

Note that small values of Δx do not necessarily imply small values of Δf, as the fitness landscape can be highly irregular. Vice versa, high values of Δx might not necessarily be inferior, as functions might have similar local optima at different locations in the search space. One might argue that in this case, either of the points could be sufficient.

As the input values of our benchmarks are all normalized, a comparison between the different benchmarks can be made for Δx. For Δf however, this produces some complications as the maximum in our input space is not always known. Therefore, the maximum values from the two dimensional problem are used to normalize the output values. The ranges used to normalize can be found in Table I.

IV. Experimental setup

The following parameters are varied in our set of experiments:



(a) 2 dimensions.

(b) 4 dimensions.

(c) 8 dimensions.

(d) 16 dimensions.

(e) 32 dimensions.

Fig. 2: Comparison of modeling techniques on R^2 scores in $2, 4, 8, 16$ and 32 dimensions with varying sampling sizes.

1) Dimensions (ranging from 2^1 to 2^5, increasing with powers of two)
2) Number of samples (ranging from 2^4 to 2^{12}, increasing with powers of two)
3) Meta-models (Decision Tree, Random Forest Regression, Support Vector Regression and Gaussian Process Regression)
4) Sampling strategies (uniform, normal, Latin Hypercube, normal-centered, and wald-centered sampling)
5) Benchmark functions (11 functions, see Table I)

The benchmarks are normalized on their input for equal comparison, each input dimension ranges between $[-1, 1]$. Samples are generated per distribution and dimension, after which all models are trained with the same dataset. In total $5 \times 9 \times 4 \times 5 \times 11 = 9900$ tests are conducted. Each test is 5-fold cross-validated. In cross-validation, the dataset is split up into n folds. One of these folds is taken out in order to serve as a validation set, the other folds are used for training the model. This is repeated until each fold has been used exactly once as validation set. In this way, scores obtained for each fold, which are averaged in order to determine how well the model fits to this dataset. All algorithms used were taken from Scikit-learn [17]. The benchmarks are taken from DEAP [7].

TABLE I: Functions used in the experimental setup.

Number:	Name:	Function:	Range:	Optima:	Δf range
1	Ackley	$f(x) = 20 - 20 \cdot \exp\left(-0.2\sqrt{\frac{1}{N}\sum_{i=1}^{N} x_i^2}\right) + e - \exp\left(\frac{1}{N}\sum_{i=1}^{N}\cos(2\pi x_i)\right)$	$x_i \in [-15, 30]$	$x_i = 0, \forall i \in \{1 \dots N\}, f(x) = 0$	[0, 25]
2	Bohachevsky	$f(x) = \sum_{i=1}^{N-1}\left(x_i^2 + 2x_{i+1}^2 - 0.3\cos(3\pi x_i) - 0.5\cos(4\pi x_{i+1} + 0.7)\right)$	$x_i \in [-100, 100]$	$x_i = 0, \forall i \in \{1 \dots N\}, f(x) = 0$	[0, 30000]
3	Griewank	$f(x) = \frac{1}{4000}\sum_{i=1}^{N} x_i^2 - \prod_{i=1}^{N}\cos\left(\frac{x_i}{\sqrt{i}}\right) + 1$	$x_i \in [-600, 600]$	$x_i = 0, \forall i \in \{1 \dots N\}, f(x) = 0$	[0, 200]
4	Plane	$f(x) = x_0$	$x_i \in [-100, 100]$	$x_0 = -\infty, f(x) = x_0$	[-100, 100]
5	Rastrigin	$f(x) = 10N\sum_{i=1}^{N} x_i^2 - 10\cos(2\pi x_i)$	$x_i \in [-5.12, 5.12]$	$x_i = 0, \forall i \in \{1 \dots N\}, f(x) = 0$	[0, 90]
6	Rastrigin scaled	$f(x) = 10N + \sum_{i=1}^{N}\left(10^{\left(\frac{i-1}{N-1}\right)} x_i\right)^2 - 10\cos\left(2\pi 10^{\left(\frac{i-1}{N-1}\right)} x_i\right)$	$x_i \in [-5.12, 5.12]$	$x_i = 0, \forall i \in \{1 \dots N\}, f(x) = 0$	[0, 3000]
7	Rastrigin skewed	$f(x) = 10N\sum_{i=1}^{N} y_i^2 - 10\cos(2\pi x_i)$ with $y_i = \begin{cases} 10 \cdot x_i & \text{if } x_i > 0, \\ x_i & \text{otherwise} \end{cases}$	$x_i \in [-5.12, 5.12]$	$x_i = 0, \forall i \in \{1 \dots N\}, f(x) = 0$	[0, 6000]
8	Rosenbrock	$f(x) = \sum_{i}^{N-1}(1 - x_i)^2 + 100\left(x_{i+1} - x_i^2\right)^2$	$x_i \in [-2, 3]$	$x_i = 1, \forall i \in \{1 \dots N\}, f(x) = 0$	[0, 12000]
9	Schaffer	$f(x) = \sum_{i=1}^{N-1}\left(x_i^2 + x_{i+1}\right)^{0.25} \cdot \left[\sin^2\left(50 \cdot \left(x_i^2 + x_{i+1}^2\right)^{0.10}\right) + 1.0\right]$	$x_i \in [-100, 100]$	$x_i = 0, \forall i \in \{1 \dots N\}, f(x) = 0$	[0, 25]
10	Schwefel	$f(x) = 418.9828872724339 \cdot N - \sum_{i=1}^{N} x_i \sin\left(\sqrt{\lvert x_i \rvert}\right)$	$x_i \in [-500, 500]$	$x_i = 420.9687, \forall i \in \{1 \dots N\}, f(x) = 0$	[0, 1800]
11	Sphere	$f(x) = \sum_{i=1}^{N} x_i^2$	$x_i \in [-100, 100]$	$x_i = 0, \forall i \in \{1 \dots N\}, f(x) = 0$	[0, 20000]

V. Results

The performance of the models are assessed by comparing the Δx, Δf and R^2 scores obtained from the experiments. In addition, the influence of the number of samples and the sampling distribution is discussed in the section below.
The results of our tests, Δx, Δy and R^2-scores, are shown in Figure 3. Our newly designed metrics can provide additional information about the performance of the models when comparing them with the R^2 score.

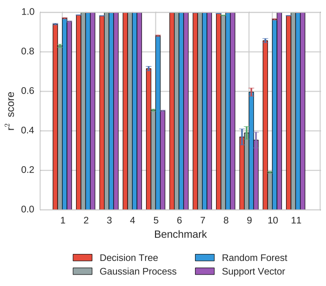
1) Δx versus R^2 score: In order to investigate whether higher R^2 scores correspond with better found optima, we can have a look at the R^2 and Δx scores in Figure 3. This shown us that higher values of the R^2-score indeed seem to correlate with lower Δx, improving the quality of our search when the model corresponds closely to the benchmark.

2) Δf versus R^2-score: In Figure 3, something interesting can be observed: larger values of Δx do not necessarily imply larger values of Δf. In two dimensions, Figure 3e paints a much brighter picture than Figure 3c. This leads us to believe that the Δf score might be a better representation than Δx, as this shows how good the found optimum is, had we assumed that it was located at the actual benchmark optimum. Note that the values in 32 dimensions are much higher, caused by the fact that the output of the models is normalized by the ranges in 2 dimensions.

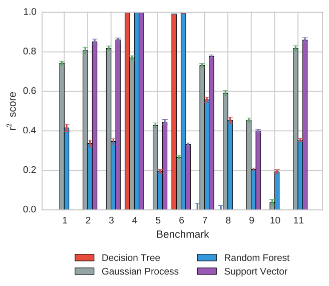
It can also be observed that even with the maximum number of samples, RFR cannot compete with SVR and GP when used for optimization. This can be explained by the smoothness of the models, as the predictions of RFR are not necessarily smooth. Our optimization might get stuck in one of the local optima present in the model.

A. Influence of number of samples and distribution

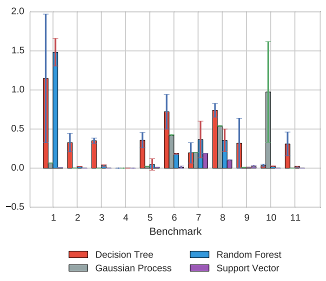
In Figure 2 our four meta-models are compared on different numbers of samples, distributions and dimensions. The values are averaged over the benchmarks: each point represents the score of the model, averaged over the 11 five-fold



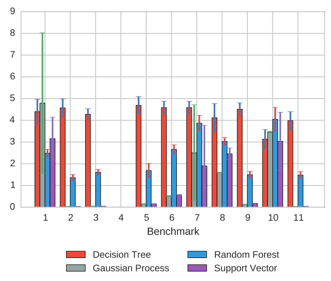
(a) R^2-scores, 2 dimensions



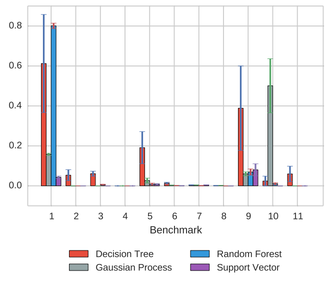
(b) R^2 scores, 32 dimensions



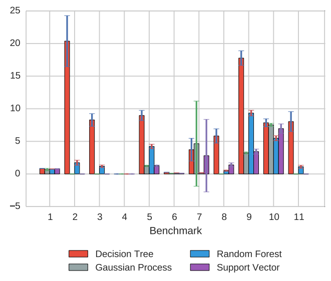
(c) Δx, 2 dimensions



(d) Δx, 32 dimensions



(e) Δf, 2 dimensions



(f) Δf, 32 dimensions

Fig. 3: Comparing R^2 score, Δx and Δf for all benchmarks in both 2 and 32 dimensions for 4096 samples according to a Latin Hypercube design.

cross-validated benchmarks.

As expected, increasing the number of samples leads to higher scores (higher R^2 scores are better). Comparing our results to the recommendations made by Akkerman et al. [1] and Shi [20], who conclude that 10N or 3N samples should be sufficient, our results show that more samples are necessary to obtain good average scores on all benchmarks. To demonstrate this, the *knee-point* (the closest point to the point (0,1)) of the function is computed for each distribution and dimension. The average of these values is 44N with a median of 32N. The guideline of 32N is visualized in the plots in Figure 2. This would mean that our recommended sample size is 2^6 for 2 dimensions, increasing to 2^{10} for 32 dimensional problems. These results will of course also vary per benchmark, which explains the difference and makes the creation of rough guidelines a difficult task.

As the dimensionality of the problem grows it becomes increasingly difficult to obtain well-fitted models. Especially, a simple Decision Tree is no longer sufficient to fully model the problem in higher dimensions. Depending on the sampling method, either Gaussian Process Regression (Kriging) or Support Vector Regression (SVR) has the best average performance on all benchmarks. Random Forests can be a good choice as well, as it is usually a close second or third, but is almost always outperformed.

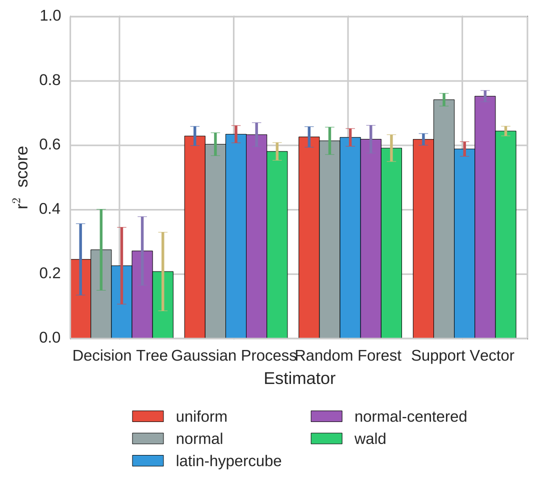


Fig. 4: Comparing distributions

Some interesting results can be seen when comparing different distributions. Surprisingly, the uniform sampling method works almost as well as the widely used Latin Hypercube sampling. As expected, Wald sampling has the worst performance of all sampling methods, becoming more apparent as the dimensionality of the problem grows. However, the differences between distributions are relatively small. This can be seen in Figure 4, where the performance of the models is shown on different distributions (with 512 samples). SVR combined with normal and normal-centered sampling gives surprisingly good results, with much better scores than, for example, Latin Hypercube sampling.

VI. Conclusions and Outlook

In this paper, a new framework for testing the performance of meta-models is proposed. In this framework, in addition to only testing model accuracy, also the location of the optimum is taken into account in order to know whether a model has been well-fitted. This *optima-fit* consists of two metrics: Δx (measuring distance between optima in the input space) and Δf (measuring difference in the objective space).

In order to evaluate this framework, four different meta-modeling techniques (Decision Tree, Random Forest, Kriging and Support Vector Regression) have been tested on eleven benchmarks. These models were all trained with nine sampling sizes, each taken from five distributions. An analysis of each of the models is performed, to provide insight in their behaviour and in order to compare different techniques. In particular, the properties of the model for optimization were examined, by running a CMA-ES optimizer on the output of the model. We see that the modelling techniques providing smooth output (Support Vector Regression and Kriging) not only outperform the other models solely in model quality, but also in quality of the optimum (both Δx and Δf). Random Forests seem to be more robust than these techniques in lower dimensions. Decision Trees are the fastest to train and predict, but they only perform well on simple benchmarks.

When comparing different sampling strategies, the distribution of the samples does not seem to influence the scores as much as initially was expected. Uniform random sampling produces very similar results to the widely-spread Latin Hypercube design. Sampling around the optimum of the benchmark, or sampling away from the optimum does have an influence on the results, as the latter produces lower accuracy models, as expected. However, even though a difference is observable, it proved to be less than initially expected.

In contrast to other approaches, our results show that we need at least 32N samples in order to get well-fitted models and good estimations of the optimum for all benchmarks. The difference with previous (lower) recommendations in other papers, might be due to some of the difficult and non-linear benchmarks included in the experiments.

A. Future work

From our research, we see that in 32 dimensions most models do no longer provide reliable approximations of our benchmarks. In engineering practices, problems with 32 parameters are not uncommon, however, increasing the dimensionality also increases the difficulty in both the learning of the model, as well as the optimization. One interesting idea would be to investigate these models in higher dimensions, and find methods that make them work better, or devise completely new approaches designed specifically for high-dimensional data. Additionally, only benchmarks with single-objectives were considered in this paper. Applying the experimental setup also to multi-objective optimization would be an interesting challenge.

Although our research shows the distribution of samples only has little influence on the accuracy of the models, many other space-filling distributions exist. In our research, the only space-filling experimental design that is tested is Latin Hypercube sampling, and it does not strictly outperform the uniform random sampling as expected. Other designs of experiments, such as Hammersley sequences and orthogonal arrays could be analyzed more thoroughly in future research.

References

[1] A. Akkerman, D. Anderson, and L. Gu. Robustness optimization for vehicular crash simulations. *Computing in Science Engineering*, 2(6):8–13, Nov 2000.

[2] E. Alpaydin. *Introduction to Machine Learning*. The MIT Press, 2nd edition, 2010.

[3] L. Breiman. Bagging predictors. *Machine Learning*, 24(2):123–140, 1996.

[4] L. Breiman. Random forests. *Machine Learning*, 45(1):5–32, 2001.

[5] L. Breiman, J. Friedman, R. Olshen, and C. Stone. *Classification and Regression Trees*. Wadsworth and Brooks, Monterey, CA, 1984.

[6] H. Fang, M. Rais-Rohani, Z. Liu, and M. Horstemeyer. A comparative study of metamodeling methods for multiobjective crashworthiness optimization. *Computers & Structures*, 83(2526):2121 – 2136, 2005.

[7] F.-A. Fortin, F.-M. De Rainville, M.-A. Gardner, M. Parizeau, and C. Gagné. DEAP: Evolutionary algorithms made easy. *Journal of Machine Learning Research*, 13:2171–2175, jul 2012.

[8] N. Hansen. The cma evolution strategy: A tutorial, 2005.

[9] N. Hansen and A. Ostermeier. Adapting arbitrary normal mutation distributions in evolution strategies: the covariance matrix adaptation. In *Evolutionary Computation, 1996., Proceedings of IEEE International Conference on*, pages 312–317, May 1996.

[10] R. Jin, W. Chen, and T. Simpson. Comparative studies of metamodelling techniques under multiple modelling criteria. *Structural and Multidisciplinary Optimization*, 23(1):1–13, 2001.

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

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

[13] B.-S. Kim, Y.-B. Lee, and D.-H. Choi. Comparison study on the accuracy of metamodeling technique for non-convex functions. *Journal of Mechanical Science and Technology*, 23(4):1175–1181, 2009.

[14] Y. Li, S. Ng, M. Xie, and T. Goh. A systematic comparison of metamodeling techniques for simulation optimization in decision support systems. *Applied Soft Computing*, 10(4):1257 – 1273, 2010. Optimisation Methods & Applications in Decision-Making Processes.

[15] W. J. C. M. D. McKay, R. J. Beckman. A comparison of three methods for selecting values of input variables in the analysis of output from a computer code. *Technometrics*, 21(2):239–245, 1979.

[16] A. Owen. Orthogonal arrays for computer experiments, integration, and visualization. *Statistica Sinica*, 2:439–452, 1992.

[17] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay. Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, 12:2825–2830, 2011.

[18] J. R. Quinlan. Induction of decision trees. *Mach. Learn.*, 1(1):81–106, Mar. 1986.

[19] A.-B. Ryberg. Metamodel-based design optimization – a multidisciplinary approach for automotive structures. Master's thesis, Linköping University, January 2013.

[20] L. Shi, R. Yang, and P. Zhu. A method for selecting surrogate models in crashworthiness optimization. *Structural and Multidisciplinary Optimization*, 46(2):159–170, 2012.

[21] T. Simpson, J. Poplinski, P. N. Koch, and J. Allen. Metamodels for computer-based engineering design: Survey and recommendations. *Engineering with Computers*, 17(2):129–150, 2001.

[22] T. W. Simpson, D. K. J. Lin, and W. Chen. Sampling strategies for computer experiments: Design and analysis, 2001.

[23] V. Vapnik. *Estimation of Dependences Based on Empirical Data: Springer Series in Statistics (Springer Series in Statistics)*. Springer-Verlag New York, Inc., Secaucus, NJ, USA, 1982.