Surrogate Assisted Feature Computation for Continuous Problems

Nacim Belkhir[1,2(✉)], Johann Dréo[1], Pierre Savéant[1], and Marc Schoenauer[2]

[1] Thales Research & Technology, Palaiseau, France
{nacim.belkhir,johann.dreo,pierre.saveant}@thalesgroup.com
[2] TAO, Inria Saclay Île-de-France, Orsay, France
marc.schoenauer@inria.fr

Abstract. A possible approach to Algorithm Selection and Configuration for continuous black box optimization problems relies on problem features, computed from a set of evaluated sample points. However, the computation of these features requires a rather large number of such samples, unlikely to be practical for expensive real-world problems. On the other hand, surrogate models have been proposed to tackle the optimization of expensive objective functions. This paper proposes to use surrogate models to approximate the values of the features at reasonable computational cost. Two experimental studies are conducted, using a continuous domain test bench. First, the effect of sub-sampling is analyzed. Then, a methodology to compute approximate values for the features using a surrogate model is proposed, and validated from the point of view of the classification of the test functions. It is shown that when only small computational budgets are available, using surrogate models as proxies to compute the features can be beneficial.

Keywords: Empirical study · Black-box continuous optimization · Surrogate modelling · Problem features

1 Introduction

Different optimization algorithms, or, equivalently, different parameterizations of the same algorithm, will in general perform non-uniformly on different classes of optimization problems. Today, it is widely acknowledged in the domain of optimization at large that the quest for a general optimization algorithm, that would solve all problems at best, is vain, as proved by different works around the *No Free Lunch theorem* [1,12]. Hence, tackling an unknown optimization problem amounts to choose the best algorithm among a given set of possible algorithms (*algorithm selection*), and/or the best parameter set for a given algorithm (*algorithm configuration*).

Such a choice can be considered as an optimization problem by itself, thus pertaining to the *Programming by Optimization* (PbO) paradigm proposed by [4]: given a new optimization problem instance (an objective function to minimize or maximize), a set of algorithms with domains for their parameters, and

P. Festa et al. (Eds.): LION 2016, LNCS 10079, pp. 17–31, 2016.
DOI: 10.1007/978-3-319-50349-3_2

a performance measure, find the best[1] algorithm or parameter setting to solve this problem. However, such a meta-optimization problem is in general difficult to solve (hierarchical search space, multi-modal landscape, ...) and thus requires running different algorithms with different parameterizations, where each of these runs will in turn call the objective function a large number of times: in total, the number of calls to the objective function will be huge, making such an approach intractable in most real-world situations with expensive objective functions.

The PbO approach can however be applied to classes of objective functions: the best algorithm/parameter setting can be learned off-line, once and for all for a given class of functions, and the optimal setting (algorithm + parameters) applied to all members of that class. This is the case for instance in operational contexts, where the same type of problem, but with slightly different settings, has to be solved again and again. However, in the general case of black-box optimization problems, very little domain knowledge is known (the type of search space for sure, maybe some relevant parameters) and such characteristics are not sufficient to reliably choose an algorithm and its parameters.

Another approach is then to compute some characteristics of objective functions, aka *features*, without much domain knowledge, and, thanks to a large example base of algorithms performances on known objective functions, to learn a *performance model* of several algorithms and their parameters. A well-known success in that direction has been obtained in the SAT domain [13], but dozens of features had been proposed in the literature to describe SAT problems and try to understand what makes a SAT problem hard for this or that algorithm. This situation is quite unique, and is an appeal for research regarding the design and study of features in other domains.

In particular, in the domain of continuous optimization (the search space is a subspace of $\mathbb{R}^d$, for some d), several recent works [8,9,11] have proposed many different features to try to understand the landscape of continuous optimization and, ultimately, solve the Algorithm Selection and/or Configuration problem. Note that a large body of mathematical programming algorithms exist, and are proved to be optimal for specific classes of objective functions: Linear Programming should be applied if the function (and the constraints) are linear; gradient-based algorithms should be applied if the function is convex, twice differentiable and well-conditioned, etc. But these are rare exceptions in the real world, where the general problem remains open, and the feature-based approach seems worth investigating, in particular considering the promising initial results obtained by [9] (more in Sect. 2).

Unfortunately, the computation of all proposed features is based on many sample points, i.e., values of the objective function at given (generally random) points of the search space. In real-world situations, where the objective

[1] The performance measure generally involves time-to-solution (CPU time, or number of function evaluations) and precision/accuracy of the solution returned by the algorithm (precision of the solution for continuous optimization problems, number of constraints violated for Constraint Programming problems, etc.).

function is expensive and the computational budget limited, the computation of the features as proposed in [9] might simply be impossible. A first research question is hence to study how badly the features behave when the size of the sample,used to compute those features, decreases.

Yet, a prominent approach has already been proposed to handle expensive objective functions in optimization. It relies on *surrogate models*, i.e., models of regression or interpolation of the objective function built upon sample points gathered during the run of the algorithm, and used from time to time in the optimization algorithm in lieu of the actual objective function.

Building on this idea, the present work investigates the use of surrogate models to compute problem features: based on a small sample set, a surrogate model is developed. The features of the surrogate model can then be easily computed, since the cost of evaluating a surrogate model is negligible compared to the original objective function. The second issue investigated in this paper relates to the accuracy of the features of the surrogate model as approximations of the features of the original objective function, and will be studied here experimentally, using the well-known BBOB test set of functions [3]. Note that this paper will not directly tackle the algorithm selection or algorithm configuration problem, left for further work. The accuracy of feature sets will be assessed first by a direct comparison with the features computed using a very large sample set, as well as by their abilities to correctly recover the BBOB (hand-designed) classes, as in [9] (see Sect. 4 for more details on the BBOB testbench).

The paper is organized as follow, Sect. 2 surveys the features proposed in the literature. Section 3 will introduce the methodology proposed in this paper. Section 4 describes the experimental context, while Sect. 5 describes the experimental protocol in detail, including the performance measures used to validate the results presented in Sect. 6. Finally these results are summarized and hints for further works are given in Sect. 7.

2 Problem Features for Continuous Optimization

Let $\mathcal{F}$ be the objective function at hand, defined on a domain $\mathcal{D} \subset \mathbb{R}^d$ for some given dimensions d: d is the only high-level feature given a priori as domain knowledge. All (other) features considered here will be computed from a sample set $\mathcal{X} = (x_1, \ldots x_p)$ of points of $\mathcal{D}$, and the corresponding values of the objective function $\mathcal{Y} = (y_1, \ldots, y_p)$ (with $y_i = \mathcal{F}(x_i)$ for all i).

A first series of features used here is taken from [9], where low-level features are computed directly from $(\mathcal{X}, \mathcal{Y})$. High-level features,or fitness landscape properties(e.g. multi-modality,separability, convexity,etc), are then built from different statistics on these low-level features. A total of 62 low-level features is computed, giving in turn 8 high-level features. The low-level features are grouped in the following classes:

- *Distribution*: these features consider the distribution of objective values in $\mathcal{Y}$, computing the skewness and the kurtosis of the distribution, but also estimating the number of peaks of the distribution.

- *Level Set*: from a given threshold in the objective function values (e.g., some quantiles), a classification technique such as LDA, QDA or MDA is trained to predict the position w.r.t. the threshold. The distributions of the misclassification errors for the different classifiers are used as proxies to the multi-modality of the objective function.
- *Meta-Model*: First, linear and quadratic regression models for $(\mathcal{X}, \mathcal{Y})$ are computed. The resulting R^2 coefficients for the accuracy of the models, as well as some statistics on the relative sizes of the coefficients of the models give some indication on the shape of the landscape.
- *Convexity*: estimate the probability of convexity and linearity of the objective function by selecting two random points from the sample $\mathcal{X}$ and generating a new sample point in the segment and comparing the objective value of the new sampled point and the same convex combination of the two initial objective values. The probability of convexity is computed by averaging the number of trials where the computed difference is lower than a predefined negative threshold, while the probability of linearity is computed by considering all absolute differences that are smaller than the absolute value of the predefined threshold.
- *Curvature*: considers the numerical approximation of the gradient in each point of a sub-sample of $\mathcal{X}$ by the Richardson's extrapolation method, and the resulting features consider the basic statistics of the respective derivatives, the condition number of the similar numerical approximation of the Hessian.
- *Local Search*: a local search algorithm, e.g., Nelder-Mead, is run from starting points in a sub-sample of $\mathcal{X}$; the final solutions of all runs are clustered in order to identify the local optima of the objective function, the basin sizes are approximated by the number of local searches which terminate there, and other statistics gathered during the different runs give other indicators of the landscape properties.

Furthermore, a series of features termed *Dispersion* was originally proposed in [8]. Their computation analyzes the distance between candidate solutions for a percentile of the best solutions of the optimization problem by comparing them to the mean or median distance between solutions in the whole initial sample. Different percentiles are considered, giving in total 16 features.

Finally, features related to *Information Content* have been proposed in [11]. They are related to the number of the binary decisions needed to find the information, such that each candidate solution is binarized w.r.t. the fitness value of their nearest neighbors' fitness. These metrics give information about the smoothness, or the existence of a global structure of the objective function. A total of 5 dispersion features are considered in [11].

The recent works that defined those features [9,11] successfully demonstrated that these features could be used in order to classify the optimization problems w.r.t. their BBOB classes (see definition in Sect. 4). However, some of these features (local search, curvature, convexity) require additional samples. On the other hand, in [2,11], only features that can be computed with a fixed initial sample are considered; and the results demonstrate that such limited number of

features (33 features) can nevertheless be used to correctly identify the BBOB classes. It should be emphasized, however, that all the above-mentioned works consider large samples of candidate solution (between $500 \times d$ and $5000 \times d$), which is not practical if the objective function is expensive.

3 Surrogate Assisted Feature Computation

3.1 Sub-sampling

Real world applications where numerical simulations are involved, usually result in very expensive objective functions, for which the computation of a single value might take a few hours. In such case, the features described in previous Sect. 2 might help choosing the right algorithm, and/or the right parameters of a given algorithm, thus saving computation time for its optimization process. The computation of these features should not cost more than the optimization – which might be the case if around $10^3 \times d$ evaluations are needed, as in [2,9,11]. Hence making the use of features no applicable in real world applications.

A first solution is of course to simply use a smaller sample set. But the price to pay will be a poor approximation of the actual features, which in turn might result in a wrong choice for the optimization algorithm or its parameters. As it can be expected (see Sect. 6.1), a too small sample set results in a poor approximation with a high variance w.r.t. the choice of the sample set.

3.2 Surrogate Modeling

Coming from another field, numerical engineers have tackled this problem using *Response Surface Methods* for many decades now: after few iterations of any optimization algorithm, many points of the search space have already been evaluated, and this sample set can be used to build a *surrogate model* of the objective function, that can in turn be used in the optimization algorithm as a proxy for the actual objective, being costless to compute (see e.g., [6,7] for surveys in the engineering domain and in in Evolutionary Computation domain, respectively). The most critical issue to be addressed when using surrogate modelling techniques is the choice of the function space where to look for a model. Several approaches have been proposed in the literature to solve such regression problem, from Neural Networks to Gaussian Processes (aka Kriging) to Support Vector Machines and Regression Random Forests, and it is beyond the scope of this paper to describe them in detail.

Although a surrogate model to be used within an optimization process should be accurate, taking into account the local characteristic of the problem. A surrogate model whose purpose is to compute features of the function at hand should globally approximate the objective function. Both goals are different, and this should be taken into account.

3.3 Accuracy vs Efficiency

The baseline of the approach proposed in this paper is to use small sample sets to build a surrogate model of the objective function at hand, to compute the features of that surrogate model using as many samples as needed, and to use these features in lieu of the unreachable actual features of the true objective function.

However, this approach must be validated – empirically – against the simple (without surrogate) sub-sampling approach (compute the features on small sample sets). Furthermore, some parameters of the approach (the model for the surrogate and its hyper-parameters, the number of samples to be used for evaluating the features of the surrogate model) must also be tuned. This raises the question of what measure should be used to assess the quality of the approximated features.

An obvious measure is simply the error made on the feature values: for each feature, the "exact" value can be computed using as large sample sets as needed (say, same sizes than in [2,9,11]). Then the L^2 norm of the error vector, difference between the approximated and "exact" values for all features gives a first idea of how good the approximation is.

However, the ultimate reason for computing the features is to solve the algorithm selection and/or configuration problem. And it might be the case that the error in solving the latter problem varies differently from the L^2 error on the feature values (e.g., some features that are poorly approximated are also not important for the algorithm selection and/or configuration).

Because there is not yet any standard algorithm selection or classification problem that would allow us to do such a validation, some obvious proxy classification problems that are used in [2,9,11], retrieving the known classes of problems manually defined on BBOB testbench (described in next Section). Next Section will give the technical details of these experiments, and their results will be presented in the following Section.

4 Experimental Settings

BBOB testbench

All the experiments presented here use test functions from the Black Box Optimization Benchmark (BBOB)[2] [3]. The BBOB benchmark contains 24 analytically defined continuous objective functions with known global optimum. All these functions are defined on the d-dimensional domain $[-5,5]^d$, with $d \in \{2,3,5,10,20,40\}$. In order to avoid possible biases, 15 variants of each function are considered, obtained from the original function by a translation of the position of the optimum, plus a rotation of the coordinate system for the non-separable functions.

These 24 functions have been manually classified in five classes of problem, with 5 separable functions, 4 uni-modal functions with low or moderate conditioning, 5 uni-modal functions with high conditioning, 5 multi-modal

[2] http://coco.gforge.inria.fr.

functions with regular global structure, 5 multi-modal functions with weak global structure.

Sample Sets
All sample sets are drawn uniformly on $[-5, 5]^d$ for a BBOB function in dimension d (see above). In the remaining of the paper, all sample set sizes are normalized w.r.t. the dimension d of the definition domain of the objective function. For the sake of brevity, we will only mention the ratio between the sample set size and the dimension: "a sample of size k" will actually mean "a sample of size $k \times d$". In all experiments, $k \in \{30, 50, 100, 500, 1000, 2000\}$ (the largest value 2000 was decided after the first experiments, see Sect. 6.1).

Features
As discussed in Sect. 2, only part of the features described in [8,9,11] will be used here, namely: 3 Distribution features, 9 Meta-Model features, 16 Dispersion features, and 5 Information Contents features. All considered features are implemented as an R package publicly available at http://github.com/flacco, thanks to Pascal Kerschke.

Surrogate Models
As discussed in Sect. 3.2, several approaches to surrogate modelling will be used and compared: Gaussian Processes, Random Forests, Support Vector Machines with polynomial or RBF kernel, denoted respectively GP, RF, SVM_P and SVM_{RBF}. But learning a surrogate model requires some hyper-parameters to be tuned: a grid search is performed in the hyper-parameter space (4 parameters for GP, 5 for the other models), using a 5-folds cross-validation procedure — randomly re-sample (Bootstrap) the sample set, using 80% for training the surrogate and the remaining 20% for testing— for 300 iterations, optimizing the approximation accuracy on the test set. All surrogate modelling procedures are implemented w.r.t the python scikit-learn library[3].

5 Experimental Protocol and Validation

Experimental Protocol and Notations
For a given objective function $\mathcal{F}$ (one trial of one instance of one $d-$dimensional function from BBOB), the basic experiment goes as follows: one sample sets of given size is drawn from the definition domain of $\mathcal{F}$; the features are computed on this sample set, and a surrogate model using one of the chosen modelling techniques. The sample set is then completed with more samples, using the surrogate model in lieu of the original function. Approximate features are then computed using this extended sample set.

An immediate validation of such approximated feature values can be made by comparing them to the "exact" values: a proxy for these values will be the features computed with the largest initial sample set, of size 2000 (see Sect. 6.1). However, the global validation (see below) requires to compute such

[3] http://scikit-learn.org/.

approximated features for all BBOB functions, and for several different sample set sizes.

Let $\mathcal{X}^s, s = 1, \ldots, S$ be some sample sets from the domain of $\mathcal{F}$[4]. Features $\Phi(\mathcal{F}^s)$ (vector of $\mathbb{R}^F$ if F is the number of features) are computed for $\mathcal{F}$ from sample set $(\mathcal{X}^s, \mathcal{Y}^s)$ (with $y_i = \mathcal{F}(x_i)$ for all $(x_i, y_i) \in (\mathcal{X}^s, \mathcal{Y}^s)$, see Sect. 2). Let us denote by $\Phi(\mathcal{F}^*)$ the features computed from the largest sample set (of size 2000).

Each sample set $\mathcal{X}^s$ is also used to learn some surrogate models $\widehat{\mathcal{F}_t^s}$ using different surrogate modelling techniques $t = T_1, \ldots, T_T$. For each (s, t), the set of features $\Phi(\widehat{\mathcal{F}_t^{s,s'}})$ is computed for $\widehat{\mathcal{F}_t^s}$, after completing the sample set $\mathcal{X}^s$ with s' new points using $\widehat{\mathcal{F}_t^s}$, resulting in sample set $(\widehat{\mathcal{X}_t^{s,s'}}, \widehat{\mathcal{Y}_t^{s,s'}})$ of size $s + s'$ (i.e., $\mathcal{X}^s \subset \widehat{\mathcal{X}_t^{s,s'}}$ and, for all $(x_i, y_i) \in (\widehat{\mathcal{X}_t^{s,s'}}, \widehat{\mathcal{Y}_t^{s,s'}})$, $y_i = \mathcal{F}(x_i)$ if $x_i \in \mathcal{X}^s$ and $y_i = \widehat{\mathcal{F}_t^s}(x_i)$ otherwise). All approximate features $\Phi(\mathcal{F}^s)$ and $\Phi(\widehat{\mathcal{F}_t^{s,s'}})$ can then be compared to the "exact" features $\Phi(\mathcal{F}^*)$, and their accuracy assessed using the L^2 norm in $\mathbb{R}^F$ of the errors on the feature values ($Err(\mathcal{F}^s) = ||\Phi(\mathcal{F}^s) - \Phi(\mathcal{F}^*)||_2$, $Err(\widehat{\mathcal{F}_t^s}) = ||\Phi(\widehat{\mathcal{F}_t^{s,s'}}) - \Phi(\mathcal{F}^*)||_2$).

However, as discussed in Sect. 3.3, another comparison is needed between the approximate features and the "exact" values, that relates to the ability of the approximate features to be sufficient to correctly classify the BBOB classes. Such validation requires the computation of all approximate features with same sizes of sample sets for all instances of functions of BBOB testbench.

Classification Efficiency

However, measuring the efficiency of a set of approximated features as a whole goes through using them as input for learning a classifier in order to discriminate the five BBOB classes. This is done using a 5-folds cross-validation, repeated 100 times, procedure as follows. Let us denote $Cl(\mathcal{F})$ the class (in 1..5) a given function $\mathcal{F}$ belongs to. For a given sample set size s, the example set for the classification task consists of $(\Phi(\mathcal{F}^s), Cl(\mathcal{F}))$ pairs when dealing with features computed on $\mathcal{F}$ and $(\Phi(\widehat{\mathcal{F}_t^{s,s'}}), Cl(\mathcal{F}))$ when dealing with surrogate model t built on $\mathcal{F}$ ($t \in \{GP, RF, SVM_P, SVM_{RBF}\}$). Such example set is made of 5 trials $\times$ 5 instances $\times$ 24 functions $\times$ 5 dimensions. Out of these 3000 examples, 80% are randomly chosen *without replacement and avoid any overlapping the training and test set*, equally distributed in the 5 classes, to build the training set, on which a Random Forest classifier is trained (with default hyper-parameters from scikit-learn).

The accuracy of the resulting classifier should then be assessed on the remaining 20% of the global example set. However, different scenarii are possible in real-world situations. A first scenario is when the training phase is done on easy functions, for which it is possible to compute the features with large enough sample sets, and the unknown functions on which to perform algorithm selection/configuration (the test phase) are all expensive. An "orthogonal"

[4] By abuse of notation, s will denote both the sample set and the (normalized) size of the sample set.

scenario is when the functions available for training are also expensive. In the latter case, only approximate features will be available for training, either computed on small samples, or computed using a surrogate of the functions used for training.

Two situations similar to the ones described above will be experimented with here, involving different sample sizes for the training of the classifier and its test.

When no surrogate model is involved (Sect. 6.1), an experiment studying the efficiency of the approximate features as a basis for classification (Sect. 6.1) is defined with only two parameters: the size s_{train} of the sample set used to learn the features for the training set, and the size s_{test} of the sample set used to learn the features for the test set. The classification accuracy of the resulting classifier will be denoted $Eff(s_{train}, s_{test})$.

But when analyzing the efficiency of surrogate assisted feature computation (Sect. 6.2), an experiment is defined with 5 parameters: the type T of surrogate model (in $\{GP, RF, SVM_P, SVM_{RBF}\}$), and, for both the training features and the test features, the sizes of the original sample sets used to learn the surrogate models (respectively s^{org}_{train} and s^{org}_{test}), and the additional number of points added to these original sample sets using the surrogate models (respectively s^{surr}_{train} and s^{surr}_{test}). The classification accuracy of the resulting classifier will be denoted $\widehat{Eff}(T, s^{org}_{train}, s^{surr}_{train}; s^{org}_{test}, s^{surr}_{test})$. Note that if one of the s^{surr}_{train} or s^{surr}_{test} is 0, only the true values of $\mathcal{F}$ are used in the corresponding step. In particular $\widehat{Eff}(T, s^{org}_{train}, 0; s^{org}_{test}, 0) = Eff(s^{org}_{train}, s^{org}_{test})$ (the surrogate model is never used).

6 Results

Two series of experiments will be presented here. The first one (Sect. 6.1) doesn't involve any surrogate model, and aims at studying how the features diverge from their "exact" baseline values when the size of the sample decreases. The goal of the second series (Sect. 6.2) is to check whether using a surrogate model built on the same available small sample set to complement it can help to cope with such divergence. In both series, the divergence with the baseline values will be assessed by the accuracy of the approximated values (individual comparison for a given feature and a given function, and their aggregation in the L^2 error – Sects. 6.1 and 6.2 respectively), and by the efficiency of the whole set of approximate features, using them to discriminate the 5 BBOB classes, as explained in previous Section – Sects. 6.1 and 6.2 respectively).Due to space constraints, only few typical figures are displayed[5].

6.1 Effects of Sub-sampling

Accuracy of Sub-Sampled Features. Five test functions (F1, F8, F13, F17, F23, one per BBOB class) are used here to assess the effect of sub-sampling

[5] Additional plots are available at https://drive.google.com/open?id=0B9GuQcCjvwtFM2VLeVEyMGtFQnM.

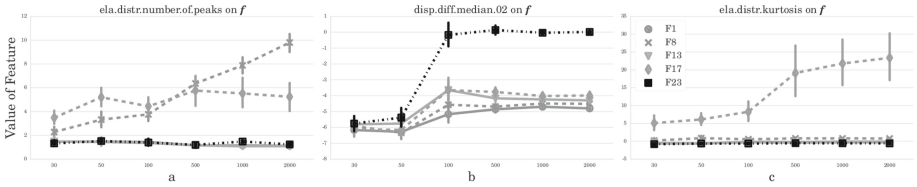

Fig. 1. Examples of effect of sub-sampling (x-axis) on feature values for the different test functions in dimension $d = 5$.

on the feature values. Figure 1 shows some typical feature behaviors on those 5 functions: whereas plot (a) display a smooth behavior, where feature values stabilize for $s \geq 100$, both other plots show that even $s = 2000$ might still be somehow too small for the multi-modal functions F17 and F23. However, most features on most functions exhibited a smooth behavior, and were stable for $s \geq 500$, justifying the decision to take $\Phi(\mathcal{F}^{2000})$ as $\Phi(\mathcal{F}^*)$. It is also clear from Fig. 1 that small sample sizes (e.g., 30) will provide a poor approximation of the feature values, and might not allow to discriminate among different functions.

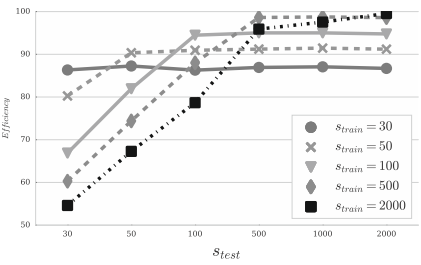

Fig. 2. $Eff(s_{train}, s_{test})$ vs s_{test} for different values of k_{train}.

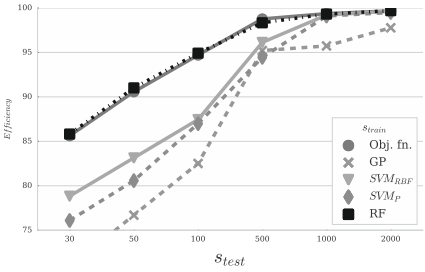

Fig. 3. $Eff(s, s)$ (black circles) and $\widehat{Eff}(T, s, *; s, *)$ for different T.

Efficiency of Sub-Sampled Features. Figure 2 displays the efficiency of the approximated features to discriminate among BBOB classes (see Sect. 5). Each line corresponds to a sample size s_{train} used to train the classifier, and each point corresponds to a different sample size s_{test} used to compute the features of the test instance to be classified.

Two conclusions can be drawn from this figure. First, there is no reason to use a larger test sample size s_{test} than the size s_{train} that was used to train the classifier, as the efficiency does not increase after s_{test} has reached s_{train}. Second, if you know that only a limited budget will be available at test time (i.e., all new instances will be very expensive), then you should train the classifier with a small budget too: for a given s_{test}, the best efficiency is obtained by the classifier trained with $s_{train} = s_{test}$.

A possible explanation, to be investigated deeper in further work, is that sub-sampling does not only increase the variance of the feature values, but it also induces some bias that might be also tracked by using the same sample size for training than for testing.

6.2 Surrogate Assisted Feature Approximation

In this Section, we try to improve the accuracy and efficiency of the approximate features by adding samples computed using some surrogate model, as described in Sect. 3.

Accuracy of Surrogate-Assisted Features. Figure 4 displays feature values for 3 different features (columns) and 2 different surrogate models (top: Gaussian Processes, bottom: Random Forests), for the 5 test functions F1, F8, F13, F17, F23. The effects of small sample sizes are here rather similar to those on the function alone as displayed in Fig. 1. However, it should be noted that the different surrogate models can give different behaviors on the same feature.

An alternative point of view and a comparison with the approximated features directly computed using the initial small sample with exact objective values is given in Fig. 5. Here, the Random Forest surrogate model is used to add 2000 points to the initial sample set. It is clear (and results on other feature confirm this trend) that Random Forests give a much smaller error than Support Vector Machines (with both polynomial and RBF kernel), and even more so with Gaussian Processes (not shown here for space reason). More interestingly, using the Random Forest surrogate model often results in more accurate approximate features than computing their values only on the few available exact values (see the $s = 30$ histograms on Fig. 5).

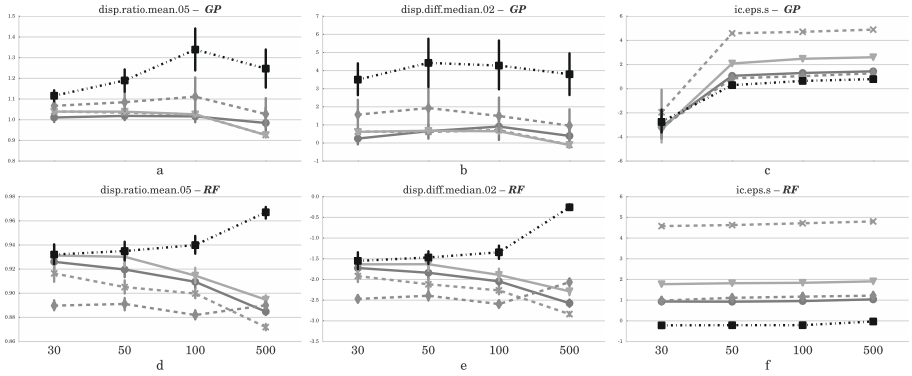

Fig. 4. Values of 3 features vs initial sample size s, for the 5 test functions ($d = 20$), using a surrogate model (GP for (a, b, c), RF for (d, e, f)) to add $s' = 2000$ points to the sample set.

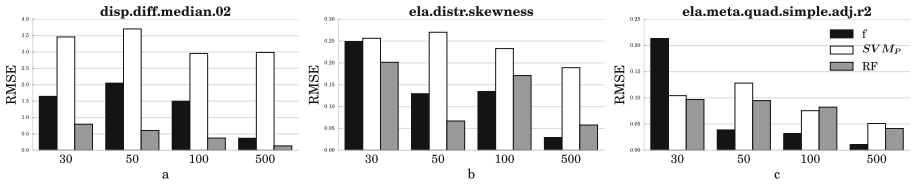

Fig. 5. Accuracy error for 3 features on F13 ($d = 5$), for different values of s (x-axis). For each s, $Err(s, s)$, $Err(\widehat{\mathcal{F}^{s,2000}_{SVM_P}})$ and $Err(\widehat{\mathcal{F}^{s,2000}_{RF}})$ are plotted.

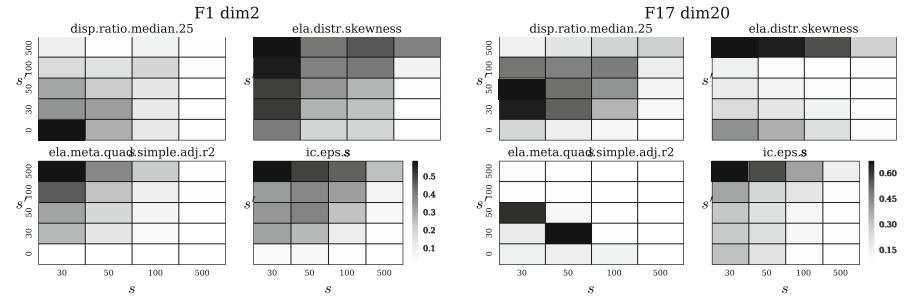

Fig. 6. Accuracy error of 4 features on F1, d=2 (left), and on F17, d=20 (right). For each (s, s'), the error $Err(\widehat{\mathcal{F}^{s,s'}_{RF}})$ is plotted (the darker the higher).

But whereas adding as many points as possible (2000 here) using the surrogate model seems a natural way to go, one can wonder if this is always the best thing to do. And the answer is no: Fig. 6 plots the error on the feature values obtained when adding s' (y-axis) sample points using a surrogate model (here, a Random Forest), to a sample set of size s (x-axis). In many cases, the error increases with s', or displays an unstructured behavior, in particular for small values of s. Only for large-enough values of s (500 and more, not shown here) does the error decrease.

Efficiency of Surrogate-Assisted Features. Let us now look at the efficiency of the approximated features to discriminate among BBOB classes (as explained in Sect. 5). Figure 3 displays $Eff(s, s)$ (the upper hull of the plots on adjacent Fig. 2) as well as the different $\widehat{Eff}(T, s, *; s, *)$, for $T \in [GP, SVM_P and SVM_{RBF}, RF]$. It is again obvious here that the Random Forest model outperforms all others, which is consistent with previous results (as well as with all other not presented results). From now on, only RF surrogate model will be considered. But another observation that can be made here is that there is no gain to be expected by using the surrogate model during the learning phase too, as both plots for $Eff(s, s)$ and $\widehat{Eff}(RF, s, *; s, *)$ are almost identical.

A final experiment will try to answer the main question that motivated this work: can the use of a surrogate model improve the efficiency of the approximated

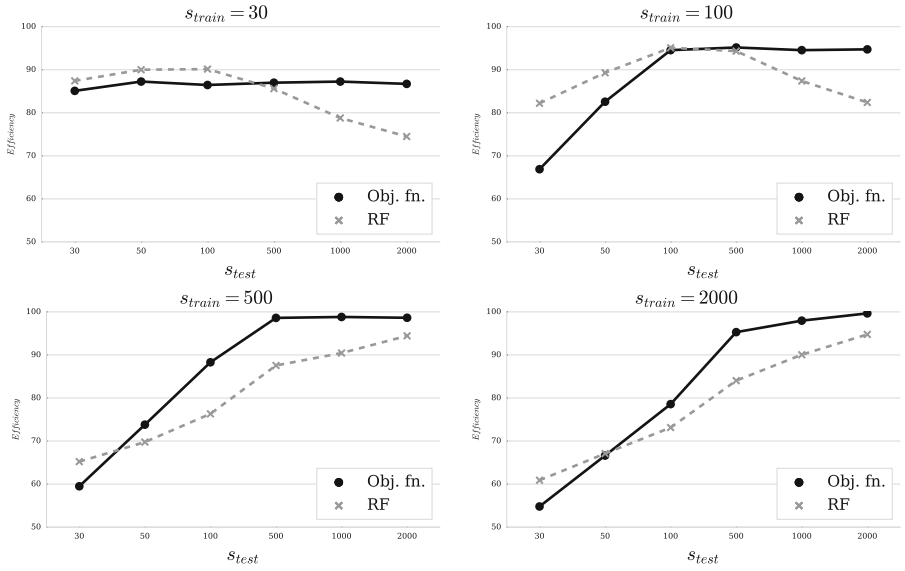

Fig. 7. Comparison, for different values of s_{train}, of the efficiency of sub-sampled features $Err(s_{train}, s_{test}$ (continuous black line) with that of surrogate assisted features $\widehat{Eff}(RF, s_{train}, 0; s_{test}, *)$ (grey dotted line).

Table 1. Mean and standard deviation of the efficiency of sub-sampled features $Eff(s_{train}, s_{test}$ (columns Obj. Fn.) and surrogate assisted features $\widehat{Eff}(RF, s_{train}, 0; s_{test}, *)$ (columns RF). Statistically significantly better results (Wilcoxon signed test with 95% confidence) are in bold.

$s_{train} = 30$		
s_{test}	Obj. fn.	RF
30	85.1±1.6	**87.4±1.5**
50	87.3±1.4	**90.0±1.3**
100	86.5±1.5	**90.2±1.2**
500	**87.0±1.6**	85.6±2.0

$s_{train} = 100$		
s_{test}	Obj. fn.	RF
30	66.9±3.7	**82.2±1.5**
50	82.6±3.0	**89.3±1.2**
100	94.6±0.9	**95.1±1.0**
500	**95.2±1.1**	94.3±1.0

$s_{train} = 500$		
s_{test}	Obj. fn.	RF
30	59.5±3.7	**65.2±3.1**
50	**73.8±2.1**	69.8±3.1
100	**88.3±2.0**	76.3±2.8
500	**98.6±0.5**	87.6±1.8

$s_{train} = 2000$		
s_{test}	Obj. fn.	RF
30	54.8±3.5	**60.9±3.1**
50	66.6±3.0	**67.1±3.3**
100	**78.6±3.6**	73.1±3.0
500	**95.3±1.1**	84.0±2.4

features in the context of expensive objective functions? Fig. 7 displays 4 plots, corresponding to different values of s_{train}. On each plot, the black continuous line is $Eff(s_{train}, s_{test})$, i.e., the corresponding line of Fig. 2, and the dotted grey line shows $\widehat{Eff}(RF, s_{train}, 0; s_{test}, *)$, i.e., the efficiency obtained when using the RF surrogate model to augment the sample set with 2000 new samples. And indeed, there is some advantage in using the surrogate model during the test phase, the more so for small training budgets. Furthermore, this advantage of using the surrogate is statistically significant, as witnessed in Table 1 where the same data are given together with the standard deviations. Figures in bold are statistically better than the corresponding non-bold figures according to a Wilcoxon signed rank test with 95% confidence.

7 Discussion and Conclusion

In the context of continuous black-box optimization, this paper has proposed a methodology to compute features describing the characteristics of the problem at hand, as proposed in [2,9,11], using surrogate models to cope with expensive objective functions: to-date methods to compute such features rely on large sample sets of evaluated points, that are not practically available when dealing with expensive real-world problems. The performance of approximated features have been measured both with their *accuracy*, related to the error on their values when compared to values computed on very large sample sets, and with their *efficiency*, ability to train a classifier that can correctly discriminate the five classes of the test functions in the BBOB testbench [3].

The paper has first experimentally studied the loss of accuracy of the features due to sub-sampling, identifying a reasonable sample size beyond which the computed features exhibit stable values with small variance – $2000\times$ the problem dimension. The study of the efficiency of the sub-sampled features led to a first conclusion: if the budget at test time is going to be small (expensive objective function), then the budget allocated to training should be small too.

Experiment involving surrogate models first surprisingly demonstrated that, in the context of the present work, only Random Forest surrogate models gave satisfactory results when used to augment the sample set used to compute the features, whereas for instance Gaussian Processes, very popular today when it comes to expensive optimization per se, gave the worst results of all.

But the most interesting observation is that when only few samples are available, using a surrogate model built on these small sample sets to augment the sample set can lead to better classification results (on BBOB classes) than the sub-sampled features directly computed on the small available sample sets. However, if some large training budgets are available, i.e., if there exists cheap functions that are representative enough of the expensive real-world objective functions that will be encountered later, then using a surrogate model on these expensive unknown function does not seem to be beneficial.

There are several directions for further work. First, several features that have been proposed in the literature have been discarded because they require additional function evaluations (e.g., the local search, curvature, and convexity features [9]). Surrogate models might also be useful in order to compute those features at low cost. Such progress would then make even more important to use some feature selection method rather than using all features for the classification task at hand.

Different paths can also be explored regarding the way the surrogate models are computed. First, as of today, only global surrogate models are considered. But it might be interesting to restrict the scope of the surrogate learning to better compute local search features for instance. Similarly, the performance measure used to construct the surrogate models is the standard approximation error on the known samples. But in the context of feature computation, some other measures could be considered – ultimately, surrogate models should be optimized for the quality of the approximate features they allow to compute.

Only global results on all features have been presented here. In particular, it could be the case that different surrogate models are the best choice to compute different features (e.g., SVM for convexity features, random forests for multi-modality features, etc.). A deeper insight on feature accuracy is needed here. Along the same line, some results presented here suggest that there might be a systematic bias induced by sub-sampling – something that requires deeper analysis too.

Last but not least, the link between those features and Algorithm selection and configuration remains to be established. Indeed, retrieving BBOB classes is a much easier problem than learning an empirical performance model, as in [5] for instance. Or those features might be used together with some latent features identification method, as in [10].

References

1. Auger, A., Teytaud, O.: Continuous lunches are free plus the design of optimal optimization algorithms. Algorithmica **57**, 121–146 (2009). https://hal.inria.fr/inria-00369788
2. Bischl, B., Mersmann, O., Trautmann, H., Preuß, M.: Algorithm selection based on exploratory landscape analysis and cost-sensitive learning. In: Proceedings of the 14th Annual Conference on Genetic and Evolutionary Computation, pp. 313–320. ACM (2012)
3. Hansen, N., Auger, A., Finck, S., Ros, R.: Real-parameter black-box optimization benchmarking 2010: experimental setup. Technical report RR-7215, INRIA (2010)
4. Hoos, H.H.: Programming by optimization. Commun. ACM **55**(2), 70–80 (2012)
5. Hutter, F., Xu, L., Hoos, H.H., Leyton-Brown, K.: Algorithm runtime prediction: methods and evaluation. Artif. Intell. **206**, 79–111 (2014)
6. Jin, R., Chen, W., Simpson, T.W.: Comparative studies of metamodeling techniques under multiple modeling criteria. Struct. Multidiscip. Optim. **23**, 1–13 (2000)
7. Jin, Y.: A comprehensive survey of fitness approximation in evolutionary computation. Soft Comput. **9**(1), 3–12 (2005)
8. Lunacek, M., Whitley, D.: The dispersion metric and the CMA evolution strategy. In: Proceedings of the 8th Annual Conference on Genetic and Evolutionary Computation, pp. 477–484. ACM (2006)
9. Mersmann, O., Bischl, B., Trautmann, H., Preuss, M., Weihs, C., Rudolph, G.: Exploratory landscape analysis. In: Proceedings of 13th GECCO, pp. 829–836. ACM (2011)
10. Mısır, M., Sebag, M.: Algorithm selection as a collaborative filtering problem. Technical report, INRIA-Saclay (2013). http://hal.inria.fr/hal-00922840
11. Munoz, M., Kirley, M., Halgamuge, S.K., et al.: Exploratory landscape analysis of continuous space optimization problems using information content. IEEE Trans. Evol. Comput. **19**(1), 74–87 (2015)
12. Wolpert, D., Macready, W.: No free lunch theorems for optimization. IEEE Trans. Evol. Comput. **1**(1), 67–82 (1997)
13. Xu, L., Hutter, F., Hoos, H.H., Leyton-Brown, K.: Satzilla: portfolio-based algorithm selection for SAT. J. Artif. Intell. Res. **33**, 565–606 (2008)

http://www.springer.com/978-3-319-50348-6

Learning and Intelligent Optimization
10th International Conference, LION 10, Ischia, Italy, May 29 -- June 1, 2016, Revised Selected Papers
Festa, P.; Sellmann, M.; Vanschoren, J. (Eds.)
2016, XI, 309 p. 74 illus., Softcover
ISBN: 978-3-319-50348-6