Department of
Information Engineering
and Computer Science DISI

UNIVERSITY
OF TRENTO - Italy

DISI - Via Sommarive, 5 - 38123 POVO, Trento - Italy
http://disi.unitn.it

ACTIVE LEARNING OF PARETO FRONTS

Paolo Campigotto, Andrea Passerini, Roberto Battiti

January 2013

Technical Report # DISI-13-001

Active learning of Pareto fronts

Paolo Campigotto, Andrea Passerini, and Roberto Battiti

***Abstract*—This work introduces the Active Learning of Pareto fronts (ALP) algorithm, a novel approach to recover the Pareto front of a multi-objective optimization problem. ALP casts the identification of the Pareto front into a supervised machine learning task. This approach enables an analytical model of the Pareto front to be built. The computational effort in generating the supervised information is reduced by an active learning strategy. In particular, the model is learnt from a set of informative training objective vectors. The training objective vectors are approximated Pareto-optimal vectors obtained by solving different scalarized problem instances. The experimental results show that ALP achieves an accurate Pareto front approximation with a lower computational effort than state-of-the-art Estimation of Distribution Algorithms and widely-known genetic techniques.**

***Index Terms*—Multi-objective optimization, Gaussian process regression, active learning, uncertainty sampling.**

I. Introduction

REAL-world optimization tasks usually require the optimization of several conflicting objectives: a solution simultaneously optimizing all of them does not exist. Therefore, the solution to the multi-objective optimization problem (MOP) becomes the quantitative identification of trade-offs between the multiple objectives. The trade-offs between the competing objectives are captured by the *Pareto-optimal* solutions, for which any single objective cannot be improved without compromising at least one of the others.

The set of the Pareto-optimal solutions in the decision space (Pareto set or PS) and of the corresponding objective vectors in the objective space (Pareto front or PF) typically have a large or even infinite cardinality, as it is the case for continuous problems. In these cases, the typical solution consists of an approximation of the Pareto set (Pareto front) by a finite number of representative solutions (objective vectors). The better the approximation of the Pareto front, the better the choice of the favorite compromise between the objectives that is offered to the decision maker.

The traditional approach to obtain a finite set of Pareto-optimal solutions (and the corresponding images in the PF) involves the sequential generation and solution of scalarized instances of the MOP. Scalarization [1], [2] consists of transforming the original MOP into a *single*-objective optimization problem (SOP). The generated SOP is a parametric combination of the multiple objectives of the original MOP into a single objective. This combination may involve the generation of additional constraints not included in the original MOP. Different Pareto-optimal solutions can be obtained by appropriately varying the scalarization parameters. The generated SOP can be solved by applying common methods and widely developed theory for single-objective optimization. However, scalarization-based methods are usually sensitive to the shape or continuity of the Pareto front [3]. For example, non-convex parts of the Pareto front cannot be recovered by optimizing convex combinations of the objective functions.

Rather than relying on scalarization techniques, current state of the art approaches for MOPs are represented by the Evolutionary Multi-objective optimization algorithms (EMOAs) [4], which are less susceptible to the shape of the Pareto front, handling, e.g., discontinuous or concave shapes. EMOAs are heuristic techniques generating an approximation of the whole PF in a single run and without using any derivative (i.e., gradient) information. The approximation is obtained by repeatedly improving a set of candidate solutions in the decision space (referred to as “population” within the Evolutionary metaphor) until their images in the objective space have converged to the PF. However, neither the convergence to the PF, nor, in case of convergence, a uniform distribution of the population over the PF are guaranteed. Furthermore, the performance is typically sensitive to the setting of the algorithm parameters (e.g., population size, number of iterations, genetic operators parameters), whose tuning depends on the specific problem instance being solved.

The Active Learning of Pareto fronts algorithm introduced in this paper is different from existing EMOAs. Because the PF has infinite cardinality in the case of continuous MOPs, ALP generates an *analytical representation* of the PF, rather than an approximation by a finite and preset number of points. An analytical representation of the entire Pareto front may significantly improve the decision making process, particularly when the preferences of the Decision Maker (DM) cannot be stated *a priori*. The compact analytical representation of the PF offers to the DM the possibility of visually inspecting the front, and of focusing on the preferred regions and selecting the favorite solution $\hat{\mathbf{z}}$, as the desired compromise between the different objectives. ALP can then identify the solution in the decision space corresponding to $\hat{\mathbf{z}}$. With a large number of objectives, dimensionality reduction techniques [5] may be employed to enable the investigation of the learnt analytical PF representation in a three or two dimensional visualization.

The ALP algorithm generates the analytical PF representation by learning a model of the PF from a *training set* of approximated Pareto-optimal vectors, which are obtained by solving different SOP instances. In order to minimize the computational effort (measured as number of evaluations of the MOP objective functions), informative training objective vectors are selected by applying active learning principles (AL). In particular, the learning stage is accomplished by Gaussian process (GP) regression [6], as it provides an explicit measure of predictive uncertainty that can be used to guide

P. Campigotto, A. Passerini and R. Battiti are with the Department of Information Engineering and Computer Science, University of Trento, Via Sommarive 14, I-38123 Povo, TN, Italy. E-mail: {campigotto,passerini,battiti}@disi.unitn.it.

the selection of the training examples (active learning by uncertainty sampling [7]). The adoption of the AL paradigm also favors the *anytime* property: when increasing the number of function evaluations, the accuracy of the analytical PF representation improves. Furthermore, the GP method offers a natural termination criterion for ALP: when the *information gain* obtained by including any additional training example is negligible, the algorithm stops. The training objective vectors are generated by solving different instances of a scalarized optimization problem. ALP does not require any derivative information and is a generic framework: neither the ML method learning the PF model, nor the AL principles and the optimization techniques for solving SOPs instances are limited to the ones discussed in this paper.

Let us note that there are two possible connections between machine learning and multi-objective optimization. The first connection arises because machine learning problems contain challenging multi-objective optimization tasks, from general cases like the trade-off between intra-cluster similarity and inter-cluster dissimilarity in clustering problems, to specific cases such as, reinforcement learning problems with multiple conflicting reward functions [8], [9]. The second connection, implemented by our work, goes in the opposite direction: it uses learning techniques from the machine learning community to solve the general multi-objective optimization problem.

The next Section describes the problem settings assumed in this work. Details of the ALP algorithm are provided in Sec. III, while Sec. IV describes the Gaussian process regression technique used to model the Pareto front. Related work is discussed in the following section. An experimental comparison between ALP and state-of-the-art EMOAs is reported in Sec. VI, while the ability of ALP in learning disconnected Pareto fronts is evaluated in Sec. VII and Sec. VIII. The experiments in Sec. IX validate the choice of the active learning strategy based on the predictive uncertainty of the GP regression model. Finally, possible generalizations of ALP and interesting directions for future research are discussed in Sec. X.

II. Problem settings

The ALP algorithm introduces a new strategy to tackle multi-objective optimization problems (MOPs). In this Section, we give the formal definition of the MOP, specifying the terminology used throughout the paper, and define the properties of the Pareto front which our algorithm relies on.

A. Multi-objective optimization problem

A multi-objective optimization problem (MOP) is formulated as:

$$\begin{array}{ll} \min\limits_{\mathbf{x}} & \mathbf{f}(\mathbf{x}) = \{f_1(\mathbf{x}), \ldots, f_m(\mathbf{x})\} \\ \text{subject to} & \mathbf{x} \in \Omega \end{array} \tag{1}$$

where $\mathbf{x} \in \mathbb{R}^n$ is a vector of n decision variables; $\Omega \subset \mathbb{R}^n$ is the *feasible region* and is typically specified as a set of constraints on the decision variables; $\mathbf{f} : \Omega \to \Phi \subset \mathbb{R}^m$ is made of m objective functions which need to be jointly minimized. Objective vectors are images of decision vectors and can be written as $\mathbf{z} = \mathbf{f}(\mathbf{x}) = \{f_1(\mathbf{x}), \ldots, f_m(\mathbf{x})\}$. Problem (1) is ill-posed whenever objective functions are conflicting, a situation which typically occurs in real-world applications. In these cases, an objective vector is considered optimal if none of its components can be improved without worsening at least one of the others. An objective vector $\mathbf{z}$ is said to *dominate* $\mathbf{z}'$, denoted as $\mathbf{z} \succ \mathbf{z}'$,if $z_k \leq z'_k$ for all k and there exists at least one h such that $z_h < z'_h$. A point $\mathbf{x}$ is Pareto-optimal if there is no other $\mathbf{x}' \in \Omega$ such that $\mathbf{f}(\mathbf{x}')$ dominates $\mathbf{f}(\mathbf{x})$. The set of Pareto-optimal solutions is called *Pareto set* (PS). The corresponding set of Pareto-optimal objective vectors is called *Pareto front* (PF). The *ideal* objective vector represents the best value of each objective. It is generally infeasible and is obtained by separately minimizing each objective function in the feasible region, i.e., $z_k^{ID} = \min_{\mathbf{x} \in \Omega} f_k(\mathbf{x})$. Analogously, the *worst* objective vector contains the maximum possible value of each objective over the entire search space. Another notable vector in the objective space is the *Nadir* point, whose k-*th* component is the maximum value of the k-*th* objective among all the Pareto-optimal vectors: $z_k^N = \max_{\mathbf{x} \in \Omega^*} f_k(\mathbf{x})$, where $\Omega^* \subset \Omega$ is the set of the Pareto-optimal solutions. The ideal and Nadir points define the range of values that the Pareto-optimal objective vectors may attain.

B. Assumptions

Under mild smoothness conditions, the Pareto front of continuous MOPs is an $(m-1)$-dimensional piecewise-continuous manifold [1], [10], [11], with m being the number of objectives. The dominance relation defined in the objective space enables a further characterization of the PF by expressing an arbitrary objective z_d (dependent function variable) as a function of the remaining objectives $\mathbf{z}_I$ (independent function variables): $z_d = g(\mathbf{z}_I)$. Without loss of generality the m-*th* objective of a m-objective problem is considered the dependent objective, i.e., $z_m = g(\mathbf{z}_I)$, with $\mathbf{z}_I$ including objectives $z_i, i = 1, \ldots, m-1$.

In this work, we focus on bi-objective optimization problems, where the PF is characterized by the continuous function $z_2 = g(z_1)$. The only assumption of ALP is the knowledge of the domain of g. When the PF is connected, the domain of g is completely specified by a lower and an upper bound point. While the lower bound can be easily obtained by computing the value z_1^{ID} of the ideal point, the upper bound can be specified by the decision maker herself, which is often aware of a critical threshold defining the set of interesting values for a given objective. When the decision maker cannot provide the desired information, the upper bound defining the domain of g is represented by the value z_1^N of the Nadir point, which, in the case of bi-objective problems, can be computed exactly (see Fig. 1). In fact, in the case of bi-objective optimization problems, each component of the Nadir point is obtained from the Pareto-optimal objective vector containing the ideal value for the other component. Therefore, the component z_1^N can be determined by solving the following program:

$$\begin{aligned} z_1^N = \min_{\mathbf{x}} \quad & f_1(\mathbf{x}) \\ \text{subject to} \quad & f_2(\mathbf{x}) = z_2^{ID} \\ & \mathbf{x} \in \Omega \end{aligned} \tag{2}$$

For MOPs with more than two objectives or bi-objective MOPs with a disconnected PF, the domain of function g cannot be defined by just a couple of (user supplied) points. In Sec. VII, our approach is generalized to bi-objective MOPs with disconnected Pareto fronts, where the a priori knowledge of the domain of g is an impractical assumption. Finally, Sec. X outlines a possible generalization of ALP to tackle MOPs with a number of objectives larger than two.

III. The ALP algorithm

Our approach casts the MOP into a *machine learning task*: given a training set of approximated Pareto-optimal vectors, the PF is identified by learning a model explaining the training data. In particular, as the PF is an (unknown) continuous function $z_m = g(\mathbf{z}_I)$, the learning task is naturally formulated as a *regression* problem in the objective space. In particular, the dependent objective z_m of function g and the independent objectives $\mathbf{z}_I$ constitute the target variable and the input feature vector of the regression task, respectively. Let us assume that both a set of training vectors and a supervisor, which provides the value z_m for each of the vectors, are available. The target of the learning task is a model $\tilde{g}$ which, given a new vector $\mathbf{z}_I$, returns the estimated value $\tilde{z}_m$ of the target variable z_m. Model $\tilde{g}$ provides the analytical representation of the PF returned by ALP.

Given an initial set T of training examples in the form $(\mathbf{z}_I, z_m)$ (*initialization phase*), ALP iteratively learns $\tilde{g}$ (*refinement phase*). Each refinement iteration consists of training the model $\tilde{g}$ on the current training set T and, based on the learnt model, selecting a new training vector $\hat{\mathbf{z}}_I$. The new training example $(\hat{\mathbf{z}}_I, \hat{z}_m)$, where $\hat{z}_m$ is the supervised information, is included in T. The update of the training set T completes the refinement iteration.

The pseudo-code of ALP is in Fig 2. In the following, we provide the details of the algorithm.

1) Initialization phase. The initial training set T is generated by selecting v vectors uniformly at random within the feature space S of the regression task, spanned by the $m-1$ independent objectives $\mathbf{z}_I$. Supervised information for each selected vector is generated.

2) Refinement phase (active learning). Training examples generation is expensive, as computing the supervised information involves the evaluation of the objective functions $\mathbf{f}$ of the MOP. In order to minimize the number of training examples without affecting the accuracy of the learnt model, training inputs are selected by the *uncertainty sampling* principle [7]. At each refinement iteration, the new training input is the feature vector $\hat{\mathbf{z}}_I$ in S whose prediction the current model is *most uncertain*. The couple $(\hat{\mathbf{z}}_I, \hat{z}_m)$, with $\hat{z}_m$ being the supervised regression score, is considered the *most informative* training data for the current model.

Given the training set T, the model $\tilde{g}$ approximating the PF is learnt by applying *Gaussian process* regression (GPR). This choice is motivated by the ability of GP learners in quantifying prediction uncertainties, which enables a suitable application of the uncertainty sampling principle. GP learners provide a Gaussian distribution $\mathcal{N}(\mu(\bar{\mathbf{z}}_I), \sigma(\bar{\mathbf{z}}_I))$ of the values predicted for any single test input $\bar{\mathbf{z}}_I$. The mean $\mu(\bar{z_I})$ of the predictive distribution can be interpreted as the prediction $\tilde{z}_m$ of the GPR model $\tilde{g}$ when applied on test input $\bar{\mathbf{z}}_I$, while the variance of the distribution quantifies the confidence of the model about the prediction. Large variance means that the test sample is not represented by the model learnt on the current training examples. Therefore, the point $\hat{\mathbf{z}}_I$ in S maximizing $\sigma(\mathbf{z}_I)$ is used to generate an *informative* training example for the current model $\tilde{g}$. A detailed description of GPR including all aspects relevant to our algorithm is reported in Section IV.

3) Supervised information generation. Supervised information consists of the value of the dependent objective $\hat{z}_m$ for a given input feature vector $\hat{\mathbf{z}}_I$. The couple $(\hat{\mathbf{z}}_I, \hat{z}_m)$ constitutes a training example. The value $\hat{z}_m$ is obtained by solving the following mathematical problem:

$$\begin{aligned} \min_{\mathbf{x},\epsilon} \quad & f_m(\mathbf{x}) \\ \text{subject to} \quad & \\ & f_j(\mathbf{x}) = \hat{z}_j + \epsilon_j,\ j = 1, \ldots, m-1, \\ & |\epsilon_j| \leq 10^{-2} \\ & \mathbf{x} \in \Omega \end{aligned} \tag{3}$$

where $\Omega \subseteq \mathbb{R}^n$ identifies the feasible region of the MOP. Let $\hat{\mathbf{x}}$ be the solution of problem (3). Then, $\hat{z}_m = f_m(\hat{\mathbf{x}})$. The objective vector $\hat{\mathbf{z}} = \{\hat{\mathbf{z}}_I, \hat{z}_m\}$ is Pareto-optimal. Slack variables ϵ_j in problem (3) relax the equality constraints $f_j(\mathbf{x}) = \hat{z}_j$. Equality constraints relaxation is introduced to solve instances of problem (3) suffering from the presence of local minima. When $|\epsilon_j| > 0$ for at least one value of index j, the new training input is a point $\tilde{\mathbf{z}}_I$ in the neighborhood of $\hat{\mathbf{z}}_I$. As position $\tilde{\mathbf{z}}_I$ is located in the region where the predictive uncertainty is higher, it represents an informative query point. The tolerance value of 10^{-2} in problem (3) is in general related to the input range and to the objective functions, the value of 10^{-2} is adequate for all benchmark problems (and the results are very robust with respect to this choice).

4) Dominance-based filtering. When the solution of problem (3) is an *approximation* $\tilde{\mathbf{x}}$ of the Pareto-optimal solution $\hat{\mathbf{x}}$, a *noisy* training example $(\tilde{\mathbf{z}}_I, \tilde{z}_m)$, with $\tilde{z}_m = f_m(\tilde{\mathbf{x}}) > \hat{z}_m$, may be generated. Therefore, when new training examples are included in the training set T, a "dominance check" is performed to detect and remove training examples which are dominated by the new training examples.

5) Termination criterion. The ALP approach is a general framework to solve MOPs, enabling different termination criteria. A simple one is provided by the AL paradigm. When the information gain obtained from *any* additional training example is negligible, the accuracy of the learnt PF model is not expected to improve in a significant manner, and the algorithm stops. The information gain is estimated by the uncertainty of the model about its predictions for the

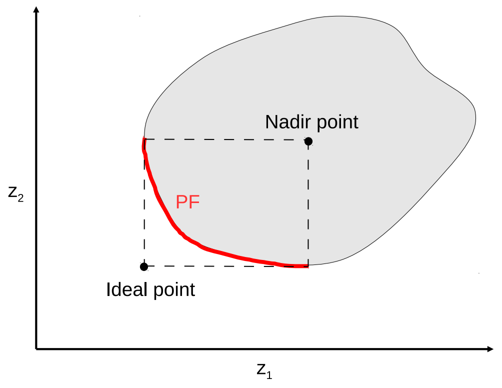

Fig. 1. Bi-objective minimization problem. The gray-shaded area denotes the feasible objective space, with the PF being depicted by the bold-marked curve. The ideal and Nadir points bounding the PF are highlighted.

```
1.  **procedure** ALP
2.      **input:** multi-objective optimization problem
3.      **output:** analytical PF representation
4.      **Let** $S$ the feature space of the regression task
5.      /* Initialization phase */
6.      Select $v$ training inputs $\mathbf{z}_I$ uniformly at random in $S$
7.      Generate regression score $z_m$ for each training input by solving $v$ instances of problem (3) /* Supervision */
8.      Initialize training set $T$ by the $v$ instances $(\mathbf{z}_I, z_m)$
9.      Remove dominated training examples from $T$ /* Dominance-based filtering */
10.     /* Refinement phase */
11.     **do**
12.         Train GP regression model on set $T$ /* Modeling */
13.         Select most informative training input $\hat{\mathbf{z}}_I$ /* Active learning */
14.         Generate regression score $\hat{z}_m$ for $\hat{\mathbf{z}}_I$ by solving problem (3) /* Supervision */
15.         Include training example $(\hat{\mathbf{z}}_I, \hat{z}_m)$ in $T$
16.         Remove dominated training examples from $T$ /* Dominance-based filtering */
17.     **until** (termination_criterion)
18.     **return** learnt GP model /* Analytical PF representation */
```

Fig. 2. Pseudo-code for the ALP algorithm generating the analytical PF representation by iteratively refining the learnt GP model.

candidate training examples. Alternative termination criteria for the iterative learning process include a limit on the number of refinement iterations or an upper bound on the number of evaluations of the MOP objectives (limit on the computational effort).

6) Computational complexity. The computational complexity of ALP is dominated by the GP training, which takes time $O(|T|^3)$ (see Sec. IV). However, because of the limited number of training examples that can be used by ALP to efficiently recover the PF, the GP training is completed in a negligible amount of time. Furthermore, in real-world MOPs, the run-time bottleneck is typically represented by the evaluation of the objective functions. The fitness computation may involve, for example, time-consuming experiments or simulations. In some real-world cases, the analytical formulation of the fitness surface may even not exist, e.g., when one objective is the optimization of quantitative judgments provided by the decision maker.

Training the GP does not require the evaluation of the objective functions, which is instead needed by the generation of the training scores (Eq. (3)). Therefore, the computational cost of ALP corresponds to the effort spent in generating the supervised information.

IV. Gaussian Process Regression

The ALP algorithm learns an analytical approximating of the PF by applying *Gaussian process* regression (GPR) [6], [12]. This choice is motivated by the ability of GP learners to provide an explicit uncertainty model for the individual predictions, therefore enabling a suitable application of the uncertainty sampling principle [7]: an input on which the current learner has maximum uncertainty is selected as new training example.

In this section we describe the regression based on Gaussian processes. First, we show how the traditional regression task is tackled by using Gaussian processes and elucidate the components (i.e., the mean and covariance functions) needed to completely specify a GP regression model. Subsection IV-B details the Gaussian noise model assumption and describes how to make predictions from a trained Gaussian process. The training (i.e., learning) of a GP model is explained in Subsection IV-C. It consists of selecting a functional form for the mean and the covariance functions and of tuning their parameters, together with the Gaussian noise variance, to fit the current data. In particular, we focus on model selection by likelihood maximization, discussing also its computational complexity. Finally, we show how GPR enables a sound application of the uncertainty sampling principle. A summary of the GPR features concludes the section.

A. Gaussian process model

The traditional regression task consists of estimating a latent function $g(\mathbf{x})$ from a noisy training dataset $T = \{(\mathbf{x}_i, y_i), i = 1, \ldots, |T|\}$. Figure 3 (left) depicts an arbitrary single-valued latent function $g(x)$ and a couple of training observations ($|T| = 2$). A Gaussian process (GP) is a collection of random variables, any finite subset of which have consistent joint Gaussian distributions. The consistency requirement means that the distributions of any finite subset of random variables satisfies the marginalization property. That is, the joint distribution of any finite subset of random variables is obtained from the joint distribution of any arbitrary superset of the original subset by marginalizing out the additional variables (e.g., $\mathrm{p}(v_1) = \int \mathrm{p}(v_1, v_2)\,\mathrm{d}v_2$, where v_1 and v_2 are two random variables).

When a GP is used to model the regression task, the random variables represent the values of the latent function $g(\mathbf{x})$ at different locations $\mathbf{x}$. Therefore, the random (function) variables can be indexed by the continuous function $g(\mathbf{x})$. In this paper, g_i denotes the random function variable associated with the input $\mathbf{x}_i$ which characterizes the possible values for $g(\mathbf{x}_i)$. By definition of GP, n arbitrary random function variables $\mathbf{g} = \{g_1, g_2, \ldots, g_n\}$ have a joint Gaussian distribution:

$$\mathrm{p}(\mathbf{g}|X) = \mathcal{N}(\bar{\mathbf{g}}, K) \tag{4}$$

where X is the set of corresponding inputs (indexes), $X = \{\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_n\}$, and $\mathcal{N}(\bar{\mathbf{g}}, K)$ denotes a multivariate Gaussian distribution with mean vector $\bar{\mathbf{g}}$ and covariance matrix K. In Fig. 3 (right), the function variables g_1 and g_3 characterize the possible values of the single-valued latent function (dotted line) at locations $x = 1$ and $x = 3$, respectively. Both random variables follow a Gaussian distribution, obtained by marginalizing their joint Gaussian distribution $\mathrm{p}(g_1, g_3|\{x_1, x_3\})$. Cross-marked points show the mean of the Gaussians, corresponding to the unknown values $g(1)$ and $g(3)$, respectively, while error bars denote the mean value +/- three times the standard deviation (corresponding to the 99.7% confidence interval).

A GP is a *conditional* probabilistic model. The distribution $\mathrm{p}(\mathbf{x})$ on the inputs is not specified, and only the conditional distribution $\mathrm{p}(\mathbf{g}|X)$ is actually modeled. For notational simplicity, in the rest of this paper the explicit conditioning on the input is omitted, with the convention that $\mathrm{p}(\mathbf{g})$ stands for $\mathrm{p}(\mathbf{g}|X)$, where X represents the appropriate inputs where the function variables $\mathbf{g}$ are conditioned on.

A Gaussian process is completely specified by the mean $\mu(\mathbf{x})$ and the covariance $c(\mathbf{x}_i, \mathbf{x}_j)$ functions. The mean function is usually set to zero (bias and offsets can be subtracted from the data without loosing generality), therefore the definition of the GP reduces to the choice of the suitable covariance function, which measures the similarity between inputs $\mathbf{x}$ by computing the covariance among the function variables g associated with the inputs. The covariance matrix K in Eq. (4) is calculated from the covariance function: $K_{i,j} = \mathrm{cov}(g_i, g_j) = c(\mathbf{x}_i, \mathbf{x}_j)$.

A *sample function* of a Gaussian process is a single realization of each of its random variables. The joint Gaussian distribution of the random variables defining a Gaussian process induces a Gaussian distribution over sample functions. The covariance function defines the properties of the functions sampled from the GP (e.g., their smoothness, length-scale, amplitude, etc.), which can be used as estimations of $g(\mathbf{x})$. Therefore the covariance function incorporates the *prior* belief (or bias) about the function $g(\mathbf{x})$ to model. Consider, e.g., the squared exponential covariance function (often referred to as radial basis or Gaussian covariance function):

$$c(\mathbf{x}_i, \mathbf{x}_j) = \sigma_f^2 \, \exp(-\frac{1}{2l^2}||\mathbf{x}_i - \mathbf{x}_j||^2) \tag{5}$$

where σ_f^2 is the "signal variance" which controls the order of magnitude of the sample functions (i.e., the scale of the output) and l defines the characteristic length-scale of the process, which informally can be thought as "roughly the distance you have to move in input space before the function value can change significantly" [6], i.e., the distance in the input space for the random function variables to become almost uncorrelated. The covariance $\mathrm{cov}(g_i, g_j)$ between variables g_i, g_j decreases as the distance between their corresponding inputs $\mathbf{x}_i$, $\mathbf{x}_j$ increases. Adopting the squared exponential covariance function is equivalent to approximating the latent function $g(\mathbf{x})$ by a linear combination of an infinite number of Gaussian basis functions [6], [13]. Furthermore, the squared exponential covariance function generates infinitely differentiable functions. Figure (4) contains five single-valued functions sampled from a zero-mean GP with squared exponential covariance function.

B. Gaussian process prediction

Gaussian process regression (GPR) assumes that the observations $y_i = g(\mathbf{x}_i) + \epsilon$ are affected by white noise ϵ, following an independent and identically distributed Gaussian distribution with zero mean and variance σ_n^2. For example, the observations $\{y_1, y_3\}$ depicted in Fig. 3 (left) are generated by setting σ_n^2= 0.2. Let $\mathbf{y} = \{y_1, y_2, \ldots, y_{|T|}\}$. Analogously to the symbol $\mathrm{p}(\mathbf{g})$, in this paper $\mathrm{p}(\mathbf{y})$ is used as short notation for $\mathrm{p}(\mathbf{y}|X)$, where X is the set of training inputs corresponding to the observations $\mathbf{y}$.

GPR is a *Bayesian* probabilistic method. A Gaussian process is used to express the *prior* belief about the function $g(\mathbf{x})$ to be modeled, *before* considering any training observation. A *noise model* (or likelihood) is designed to link the observations $\mathbf{y}$ to the latent function. Bayesian inference can then be applied

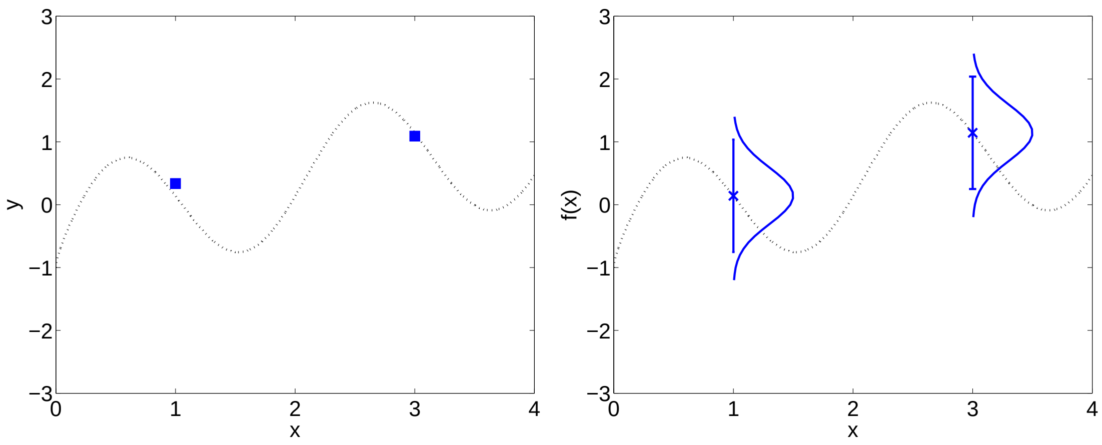

Fig. 3. (Left) Traditional regression task. The single-valued latent function $g(x)$ is represented by the dotted line. Square-marked points represent a couple of noisy training examples. (Right) GP model. The random variables g_1 and g_3, following a Gaussian distribution, characterize the possible values for the latent function (dotted line) at locations $x = 1$ and $x = 3$, respectively. See text for details.

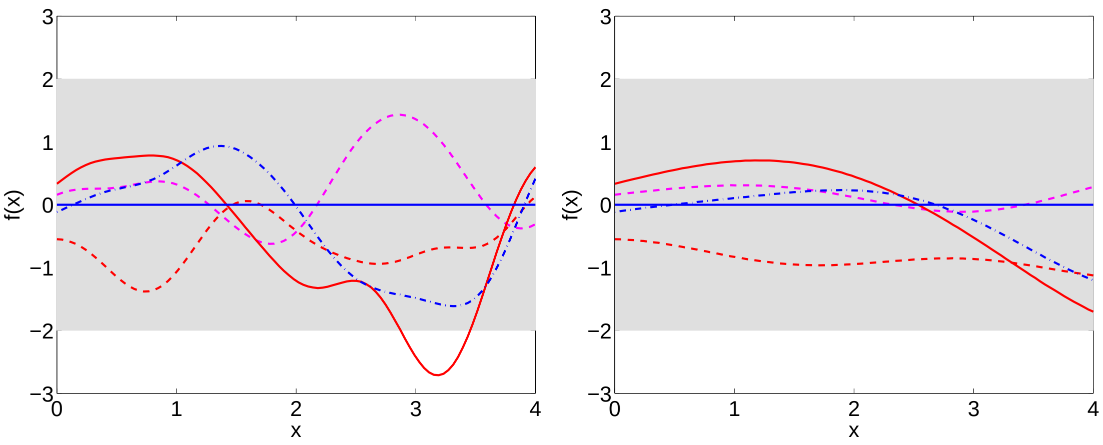

Fig. 4. Five sample functions from a zero-mean GP with a squared exponential covariance function with $\sigma_f^2 = 1$, $l = 0.5$ (left) and $\sigma_f^2 = 1$, $l = 1.5$ (right). In both plots the straight line represents the mean function of the GP and the shaded area is obtained by summing the point-wise mean function +/- two times the standard deviation for each input value (corresponding to the 95% confidence region).

to make predictions from the available training examples and the prior belief.

When using a Gaussian process, the joint distribution of the latent function variables $\mathbf{g}$ given the training inputs X is a multivariate Gaussian:

$$\mathrm{p}(\mathbf{g}) = \mathcal{N}(\bar{\mathbf{g}}, K) \tag{6}$$

with $\bar{\mathbf{g}}$ and K being the mean vector and the covariance matrix, respectively. Equation (6) defines the prior for Bayesian inference and the corresponding Gaussian process is referred to as *prior* GP. The components of $\bar{\mathbf{g}}$ are usually set to zero, thus Equation (6) can be rewritten:

$$\mathrm{p}(\mathbf{g}) = \mathcal{N}(\mathbf{0}, K) \tag{7}$$

Under the assumption of Gaussian white noise affecting the observations $\mathbf{y}$, a suitable noise model is:

$$\mathrm{p}(\mathbf{y}|\mathbf{g}) = \mathcal{N}(\mathbf{g}, \sigma_n^2 I) \tag{8}$$

where I is the identity matrix. By integrating over the unobserved function variables $\mathbf{g}$, the *marginal likelihood* (or evidence) is obtained:

$$\mathrm{p}(\mathbf{y}) = \int \mathrm{p}(\mathbf{y}|\mathbf{g})\mathrm{p}(\mathbf{g})\mathrm{d}\mathbf{g} = \mathcal{N}(\mathbf{0}, K + \sigma_n^2 I) \tag{9}$$

The term *marginal* refers to the marginalization over $\mathbf{g}$. With the prior (Eq. (7)), the likelihood (Eq. (8)) and the evidence (Eq. (9)) components, Bayesian inference can be applied to make a prediction on a test input $\mathbf{x}_*$. For this purpose, the joint distribution *under the prior GP* of the observed outputs $\mathbf{y}$ and the unknown function value $g_* = g(\mathbf{x}_*)$ at the test location $\mathbf{x}_*$ is derived:

$$\mathrm{p}(\mathbf{y}, g_*) = \begin{bmatrix} \mathbf{y} \\ g_* \end{bmatrix} = \mathcal{N}\left(\begin{bmatrix} \mathbf{0} \\ 0 \end{bmatrix}, \begin{bmatrix} K + \sigma_n^2 I & \mathbf{k}_* \\ \mathbf{k}_*' & k_* \end{bmatrix} \right) \tag{10}$$

where $k_* = c(\mathbf{x}_*, \mathbf{x}_*)$ and the $|T| \times 1$ vector $\mathbf{k}_*$ contains the covariances $c(\mathbf{x}_i, \mathbf{x}_*)$ among the training and the test inputs. Matrix $K + \sigma_n^2 I$ is the covariance matrix of the noisy observations, that is generated by simply adding the diagonal matrix $\sigma_n^2 I$ to the matrix K. In fact, under the additive, independent and identically distributed noise assumption, the covariance among observations is formulated as:

$$cov(y_i, y_j) = c(\mathbf{x}_i, \mathbf{x}_j) + \sigma_n^2 \delta_{i,j}$$

where δ indicates the Kronecker delta function ($\delta_{i,j} = 1$ if $i = j$, 0 otherwise).

The predictive distribution over the candidate values for $g(x_*)$ is obtained by conditioning on the observed training outputs $\mathbf{y}$:

$$\mathrm{p}(g_*|\mathbf{y}) = \frac{\mathrm{p}(\mathbf{y}, g_*)}{\mathrm{p}(\mathbf{y})} \tag{11}$$

The result is the predictive Gaussian distribution:

$$\mathrm{p}(g_*|\mathbf{y}) = \mathcal{N}(\bar{g}_*, \sigma^2(g_*)) \tag{12}$$

where

$$\bar{g}_* = \mu(\mathbf{x}_*) = \mathbf{k}_*'(K + \sigma_n^2 I)^{-1}\mathbf{y} \tag{13}$$

$$\sigma^2(g_*) = k_* - \mathbf{k}_*'(K + \sigma_n^2 I)^{-1}\mathbf{k}_* \tag{14}$$

Note that GPR estimates the probability distribution $\mathrm{p}(g_*|\mathbf{y})$ over the candidate values for $g(\mathbf{x}_*)$, rather than providing a single prediction (estimated best guess) for $g(\mathbf{x}_*)$. The value $\mu(\mathbf{x}_*)$ is used as the estimation for g_*, while the variance $\sigma^2(g_*)$ defines how much the GP model is confident with the estimation (larger variance means smaller confidence).

The Cholesky decomposition of matrix $K + \sigma_n^2 I$ enables a fast and numerically stable implementation of Equations (13) and (14). However, the complexity of inverting the matrix $K + \sigma_n^2 I$ is $O(|T|^3)$, where $|T|$ is the number of training observations. The computation of the predictive mean (Eq. (13)) can be reduced to $O(|T|)$, as it can be rewritten as $\mu(\mathbf{x}_*) = \mathbf{k}_*'\boldsymbol{\beta}$, with the factor $\boldsymbol{\beta} = (K + \sigma_n^2 I)^{-1}\mathbf{y}$ calculated just once for multiple predictions. A similar speed-up cannot be achieved for the computation of the predictive variance (Eq. (14)), which costs $O(|T|^2)$ per single test case. Therefore, GP training scales cubically with the number of training examples $|T|$, while predictions are quadratic in $|T|$, making prohibitive the application of GPR with more than few thousands of training examples. However, within our problem settings, much smaller data sets are used by the ALP algorithm, as the generation of training examples is expensive.

The predictive distribution in Eq. (12) can be computed for any *set* X_* of test inputs $\mathbf{x}_*$. In this case, Equation (10) is reformulated as follows:

$$\mathrm{p}(\mathbf{y}, \mathbf{g}_*) = \begin{bmatrix} \mathbf{y} \\ \mathbf{g}_* \end{bmatrix} = \mathcal{N}\left(\begin{bmatrix} \mathbf{0} \\ \mathbf{0} \end{bmatrix}, \begin{bmatrix} K + \sigma_n^2 I & K_* \\ K_*' & K_{**} \end{bmatrix} \right) \tag{15}$$

where the $|T| \times |X_*|$ matrix K_* contains the covariances among training and test locations, while K_{**} is the covariance matrix of test inputs. The predictive joint distribution $\mathrm{p}(\mathbf{g}_*|\mathbf{y})$ of the test outputs X_* has the following non-zero mean vector and covariance matrix:

$$\bar{\mathbf{g}}_* = K_*'(K + \sigma_n^2 I)^{-1}\mathbf{y} \tag{16}$$

$$\mathrm{cov}(\mathbf{g}_*) = K_{**} - K_*'(K + \sigma_n^2 I)^{-1} K_* \tag{17}$$

The joint distribution $\mathrm{p}(\mathbf{g}_*|\mathbf{y})$ is the posterior distribution for a *finite set* of test cases. While the predictive marginal distributions $\mathrm{p}(g_*|\mathbf{y})$ provides information about a single test input, the predictive joint distribution $\mathrm{p}(\mathbf{g}_*|\mathbf{y})$ defines the estimated correlation among the test inputs. In fact, when combining the prior Gaussian process with the observations by Bayesian inference and under the Gaussian noise model assumption, a Gaussian process on its own is actually obtained, referred to as the *posterior* GP. It has the following non-zero mean and covariance functions:

$$\mu_T(\mathbf{x}) = \mathbf{k}'(K + \sigma_n^2 I)^{-1}\mathbf{y} \tag{18}$$

$$\mathrm{cov}_T(g_i, g_j) = c_T(\mathbf{x}_i, \mathbf{x}_j) = c(\mathbf{x}_i, \mathbf{x}_j) - \mathbf{k}_i'(K + \sigma_n^2 I)^{-1}\mathbf{k}_j \tag{19}$$

where the $|T| \times 1$ vector $\mathbf{k}$ contains the covariances among the training inputs and input $\mathbf{x}$. The index T in function symbols μ_T and c_T highlights that the posterior GP *depends* on the training observations. The posterior variance $c_T(\mathbf{x}, \mathbf{x})$ is equal to the prior variance $c(\mathbf{x}, \mathbf{x})$ minus a positive term, which depends on the training inputs. Thus the posterior variance is always smaller than the prior variance, due to the additional information provided by the observations. Note that the predictive distributions $\mathrm{p}(g_*|\mathbf{y})$ and $\mathrm{p}(\mathbf{g}_*|\mathbf{y})$ for single and multiple test cases, respectively, are consistent with Eq. (18) and Eq. (19). Figure 5 (left) depicts five sample functions from the GP posterior, obtained by using the prior GP defined in Fig. 4 (left) and by setting the noise variance σ_n^2 to the value 0.2. Usually, the mean function $\mu_T(\mathbf{x})$ (solid blue line in Fig. 5 (right)) is used as the estimation of the latent function $g(\mathbf{x})$. In fact, the mean function can be interpreted as the average of a large number of sample functions drawn from the GP posterior, each one providing an estimation of $g(\mathbf{x})$. When using GPR within the ALP algorithm, the mean function of the posterior GP represents the Pareto front learnt from approximated Pareto-optimal training vectors.

C. *Gaussian process training (model selection)*

Gaussian process regression is a non-parametric technique, as no parametric formulation of the latent function (e.g., a weighted sum of fixed basis functions) is assumed. However, covariance functions typically have a set of free parameters. In the GPR literature, these parameters are usually referred to as *hyperparameters*, to emphasize that they are parameters of a non-parametric model [6]. The hyperparameters control the shape of the sample functions drawn from the parametric prior. For example, the two pictures in Fig. 3 show how the hyperparameter l in the squared exponential covariance function (Eq. (5)) affects the length-scale of the sample functions. In the rest of this paper, the hyperparameters of the GP model, consisting of the covariance function parameters and the noise variance σ_n^2, will be collectively referred to by the vector $\boldsymbol{\theta}$. For example, when considering the squared exponential covariance function, $\boldsymbol{\theta} = [\sigma_f^2, l, \sigma_n^2]$.

In principle, as GPR is a probabilistic model, a prior distribution $\mathrm{p}(\boldsymbol{\theta})$ encoding the belief about hyperparameters can be defined. The predictive distribution $\mathrm{p}(g_*|\mathbf{y})$ for a test input $\mathbf{x}_*$ can then be obtained by marginalizing over the uncertainty in the hyperparameters:

$$\mathrm{p}(g_*|\mathbf{y}) = \int \mathrm{p}(g_*|\mathbf{y}, \boldsymbol{\theta})\mathrm{p}(\boldsymbol{\theta}|\mathbf{y})d\boldsymbol{\theta} \tag{20}$$

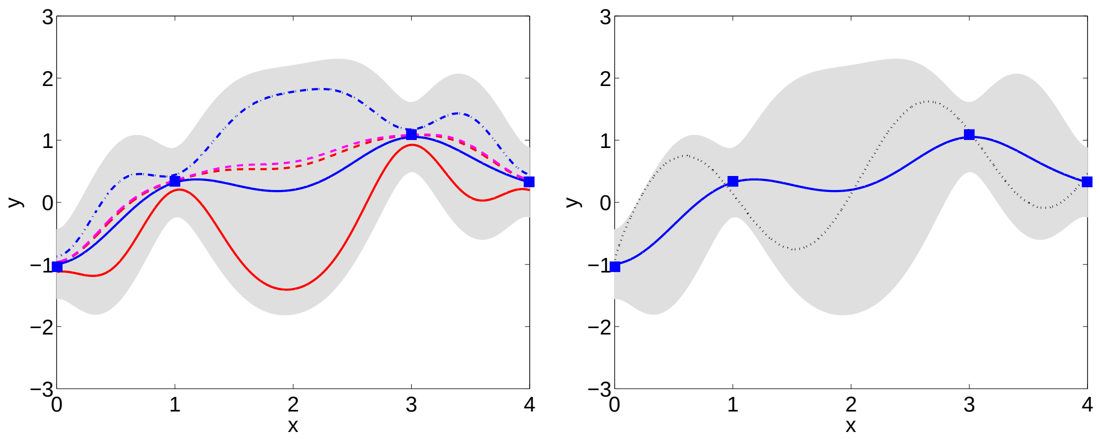

Fig. 5. (Left) Sample functions from the posterior GP. (Right) Mean function (solid line) of the posterior GP approximating the latent function g (dotted line). In both plots, squared points show the training examples and the shaded area depicts the 95% confidence region, which is defined at each input location x by the interval $[\mu_T(x) - 2\sigma^2(x), \mu_T(x) + 2\sigma^2(x)]$, with $\sigma^2(x) = c_T(x, x)$.

That is, the predictive distribution is obtained by considering the predictions from *all* the possible $\boldsymbol{\theta}$ values weighted by their posterior probability $\mathrm{p}(\boldsymbol{\theta}|\mathbf{y})$. The inclusion of the term $\mathrm{p}(\boldsymbol{\theta}|\mathbf{y})$ enables to rely on predictions of *plausible* hyperparameters settings. However, in general the above Bayesian formulation cannot be evaluated analytically and expensive numerical methods have to be considered [6]. Therefore this approach cannot be adopted by the ALP algorithm, which re-trains the GP model several times with an increasing training set.

Instead of considering the predictions from all possible hyperparameters settings, a *single* value for $\boldsymbol{\theta}$ can be inferred from the observations $\mathbf{y}$. A popular choice consists of selecting the value $\bar{\boldsymbol{\theta}}$ maximizing the posterior probability $\mathrm{p}(\boldsymbol{\theta}|\mathbf{y})$, defined as:

$$\mathrm{p}(\boldsymbol{\theta}|\mathbf{y}) \approx \mathrm{p}(\mathbf{y}|\boldsymbol{\theta})\mathrm{p}(\boldsymbol{\theta}) \tag{21}$$

The term $\mathrm{p}(\mathbf{y}|\boldsymbol{\theta})$ corresponds to the marginal likelihood defined in Eq. (9), which can be rewritten as follows to highlight the dependency of the function variables $\mathbf{g}$ on the GP hyperparameters:

$$\mathrm{p}(\mathbf{y}|\boldsymbol{\theta}) = \int \mathrm{p}(\mathbf{y}|\mathbf{g},\boldsymbol{\theta})\mathrm{p}(\mathbf{g}|\boldsymbol{\theta})\mathrm{d}\mathbf{g} = \mathcal{N}(\mathbf{0}, K + \sigma_n^2 I) \tag{22}$$

Although a suitable prior distribution $\mathrm{p}(\boldsymbol{\theta})$ in Eq. (21) helps in discarding unreasonable values of $\boldsymbol{\theta}$ [14], when the prior belief about the hyperparameters is vague, the component $\mathrm{p}(\boldsymbol{\theta})$ is typically set to the uniform distribution (i.e., ignored). Therefore Eq. (21) reduces to the maximization of the marginal likelihood. For numerical reasons, the log-marginal likelihood is used. The value $\bar{\boldsymbol{\theta}}$ is thus obtained by maximizing w.r.t. $\boldsymbol{\theta}$ the quantity:

$$\begin{aligned}\log\,\mathrm{p}(\mathbf{y}|\boldsymbol{\theta}) &= \log\,\mathcal{N}(\mathbf{0}, K + \sigma_n^2 I) = \\ &= -\frac{|T|}{2}\log\,2\pi - \frac{1}{2}\log\,|K + \sigma_n^2 I| - \\ &\quad \frac{1}{2}\mathbf{y}'(K + \sigma_n^2 I)^{-1}\mathbf{y}\end{aligned} \tag{23}$$

The quantity $\log\,\mathrm{p}(\mathbf{y}|\boldsymbol{\theta})$ represents the log-evidence for the specific GP model defined by the hyperparameters settings $\boldsymbol{\theta}$.

In general, maximizing the quantity $\log\,\mathrm{p}(\mathbf{y}|\boldsymbol{\theta})$ is a non-convex optimization task. Standard gradient-based optimization algorithms, e.g., conjugate gradient techniques or quasi-Newton methods, are typically adopted to search for locally-optimal points. The cost of computing the log-marginal likelihood and its gradient is dominated by the inversion of the covariance matrix $K + \sigma_n^2 I$. The inversion is performed at each of the l iterations of the optimization algorithm, yielding a complexity of $O(l|T|^3)$. When the number of hyperparameters is small w.r.t. the number of training examples, local optimizers are not usually a problem [15]. However, with small-size training datasets, local optimizers may decrease the performance of the optimization technique. Random restarts of the optimization algorithm overcomes the search stagnation into the local optima. Furthermore, investigating the $\boldsymbol{\theta}$ configurations corresponding to the local minimizers may be worthwhile. In fact, every locally-optimal configuration corresponds to a particular interpretation of the training data. When a small number of training examples is available, locally-optimal configurations with significantly different log-likelihood values may provide *plausible alternative* interpretations of the training examples [6] (e.g., low noise level σ_n^2 and short length-scale l w.r.t. high noise level and long length-scale).

In addition to the optimization of the covariance function hyperparameters, the model selection procedure may also include the discrete choice between different functional forms (i.e., models) for the covariance function. In particular, the functional form can be selected by simply comparing the maximum likelihood values computed for each candidate model. Finally, rather than resorting to a zero-mean prior GP, a parametric formulation for the mean function can also be specified. Its hyperparameters are included in the vector $\boldsymbol{\theta}$ and optimized together with the noise model and the covariance function hyperparameters.

D. Uncertainty sampling with GPR

Acquiring training examples for supervised learning tasks is a typically expensive and time consuming process. Active learning approaches attempt to reduce this cost by actively suggesting examples for which supervision should be collected, in an iterative process alternating learning and feedback elicitation. To minimize the number of training examples generated, *active* learning methods select the inputs that provide the highest information gain for the learnt model. The informativeness of the query inputs can be defined in different ways and several active learning techniques exist (see [16] for a recent review). One of the most popular is the uncertainty sampling principle [7], which considers the input with highest predictive uncertainty as the most informative training example for the current model.

The ability of GPR in estimating the confidence for individual predictions enables a suitable application of the uncertainty sampling principle. A large variance $\sigma^2(g_*)$ of the predictive distribution $\mathcal{N}(\bar{g}_*, \sigma^2(g_*))$ for a single test input $\mathbf{x}_*$ means that the test sample is not represented by the GP model learnt from the training data. The predictive variance quantifies the predictive uncertainty of the GP model, and therefore the input maximizing the predictive variance is selected by the uncertainty sampling principle.

With GPR, predictive uncertainty grows in regions away from training inputs. The plot in Fig. 6 (left) is obtained by adopting a squared exponential covariance function with fixed hyperparameters $\sigma_f^2 = 1$, $l = 0.5$, $\sigma_n^2 = 0.2$. When the GP model is re-trained with the inclusion of the most uncertain input (triangular marker in Fig. 6 (left)) in the training set, the prediction accuracy improves, as clearly showed by Fig. 6 (right).

E. Summary

Regression based on Gaussian process (GPR) assumes a *Gaussian noise model*. In particular, at each input location $\mathbf{x}$ the noise ϵ follows an independent Gaussian distribution with zero-mean and identical variance σ_n^2 over the whole input domain (*homoscedasticity* property). If $g(\mathbf{x})$ is the latent function to be estimated, the training value y_i observed at input $\mathbf{x}_i$ is given as $y_i = g(\mathbf{x}_i) + \epsilon$. Furthermore, the joint distribution of the noise over each arbitrary subset of the inputs is a *multivariate* Gaussian satisfying the *marginalization* property. Predictions are obtained by Bayesian inference from the training data and from the prior belief about the latent function g. The prior information is encoded by a prior Gaussian process, completely specified by its mean and covariance functions. The covariance function defines the correlation between two arbitrary random function variables g_i, g_j, and thus the similarity between the corresponding inputs $\mathbf{x}_i$,$\mathbf{x}_j$. The covariance function characterizes the properties of the regression functions (e.g., smoothness) that can be generated by the GP. Both the mean and the covariance functions are parametric, and their (hyper-)parameters, together with the σ_n^2 parameter of the noise model, are usually tuned by maximizing the *log-likelihood* of the training data. Log-likelihood optimization trades off the model complexity (regularization) and the model fit to the noisy input data [6].

The Gaussian noise model enables *exact* Bayesian inference of the predictive distribution on the test inputs and exact computation of the log-likelihood. However, the *time complexity* of both training the GP (i.e., log-likelihood maximization) and making predictions is $O(|T|^3)$, with $|T|$ being the number of training examples. Therefore GPR does not scale well with the number of training examples: neither the predictive distribution nor the marginal likelihood can be computed exactly with large datasets. In this case, one can use approximate and sparse methods, which decrease the complexity to $O(|S|^3)$ by selecting an informative subset $S \subset T$ of the training examples [6]. However, in our settings exact inference is possible because of the limited size of T. The ALP algorithm adopts GPR because it can quantify the confidence about the predicted regression function $\tilde{g}$. The estimation of the predictive uncertainty enables a sound application of the *uncertainty sampling* principle. In particular, the predictive uncertainty on a test input $\mathbf{x}_*$ is quantified by the variance $\sigma^2(g_*)$ of the predictive Gaussian distribution of the g_* values.

V. Related work

Estimation of Distribution algorithms (EDAs) are Evolutionary techniques which generate a probability model of promising solutions based on statistical information extracted from the current population. New candidate solutions are sampled from the model and used to update the current population.

ALP shares with EDAs the extraction of statistical global information from current data. However, our technique is not developed within the Evolutionary framework. EDAs typically generate a probability distribution of promising solutions in the decision space, which is used by the genetic operators to refine the current population. A finite PF approximation is provided by the population. On the contrary, the model learnt by ALP is an analytical PF approximation and the training examples are Pareto-optimal points generated by solving different SOPs. Furthermore, the SOPs are not fixed at the initialization phase, but they are dynamically generated based on the predictive uncertainty of the current PF model. Selecting the new training inputs based on the predictive model uncertainty provides two advantages. First, it enables a computationally cheaper and accurate PF approximation. Second, because the predictive uncertainty increases when moving away from the current training inputs, a diversified set of Pareto-optimal solutions, corresponding to objective vectors distributed over the whole PF, is provided to the decision maker.

Rather than learning a model of the promising solutions like the EDAs, surrogate-based EMOAs [17], [18] adopt machine learning methods to generate surrogates of the MOP functions. Surrogates generation is particularly convenient when the evaluation of the original MOP objectives is computationally expensive. The aim of surrogate-based evolutionary algorithms is the reduction of the run-time and the computational cost spent to drive the population towards the PF.

The rest of this section includes a description of EDAs, mostly examining the state-of-the-art MMEA algorithm, which is the ALP benchmark in our experimental studies. A discussion about surrogate-based (Evolutionary) optimization follows, focussed in particular on the recent MOEA/D-EGO approach, which employs the Gaussian process optimization paradigm.

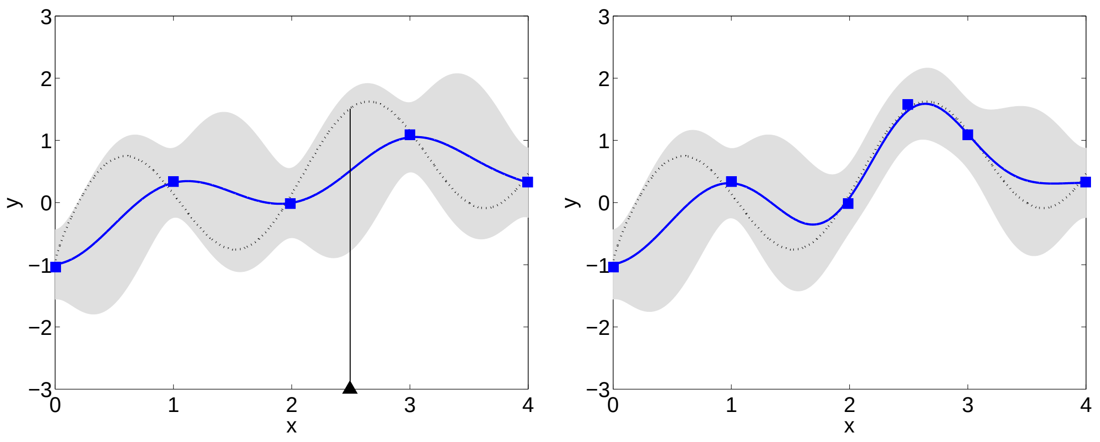

Fig. 6. Uncertainty sampling with GPR. The interpretation of the solid, dotted lines, of the squared points and the shaded area is the same as in Fig. 5. (Right) The triangular mark on the x-axis highlights the input $\hat{\mathbf{x}}$ where the GP prediction is most uncertain. (Left) The GP prediction accuracy improves when including example $(\hat{\mathbf{x}}, \hat{y})$ in the training set.

A. Multi-objective Estimation of Distribution algorithms

Estimation of Distribution Algorithms (EDAs) learn a joint probability density function over the decision variables of the optimization problem. The probability density function models the distribution of the promising solutions in the current population, with the purpose of learning the pattern characterizing the set of promising solutions. The pattern is generated by dependencies among the decision variables, often referred to as *variable linkages* [4]. The learnt information is used to predict the distribution of new high-quality solutions, thus driving the search towards the promising areas of the space. In particular, at each generation of the algorithm, the probability density function is sampled to generate a set of candidate solutions. This sample is then evaluated with respect to the objective functions of the problem and it is used to refine the current population. Based on the refined population, a more accurate model generating better-quality solutions with higher probability can be learnt in the next iteration. The generic framework of EDAs is in Fig. 7.

The motivation for EDAs is given by the limitation of common genetic operators: when two parents are in the PS, their offspring may not be close to the PS. An analogous behavior is observed when applying the mutation operator: if the original solution is Pareto-optimal, the modified solution may be dominated. Rather than resorting to traditional genetic operators, EDAs refine the population by updating and re-sampling the probability model of promising solutions. Therefore, the generated probability model should describe accurately the promising areas of the search space. Furthermore, a complicated model requiring expensive generation and sampling operations is not suitable in the context of EDAs, as these operations are repeated several times during the search.

Different probability distributions can be used in EDAs, and different machine learning methods can be adopted to learn and sample them [19]. In particular, the dependencies among the decision variables can be modeled by univariate [20], bivariate [21] and multivariate [22] probability distributions. Univariate models are based on the assumption of independent decision variables. While bivariate distributions model pairwise dependencies only, more complex interactions among subsets of variables are represented by multi-variate models. These models are usually the product of marginal probability distributions, called factors, defined over (overlapping) subsets of the decision variables. The increased expressiveness of the more sophisticated models, like the multi-variate distributions, has to be traded off with their larger computational cost.

A common approach to build computationally tractable models relies on the Gaussian distribution assumption for the decision variables. The multivariate Gaussian distribution is defined by the vector of mean values (one for each decision variable) and the covariance matrix. The mean vector determines the bias of each variable values and the variances, i.e., the entries along the diagonal of the covariance matrix, define the spread of the values. The covariances (the off-diagonal entries in the matrix) define the pairwise dependencies among the decision variables.

Different EDAs for multi-objective optimization (acronym MOEDAs) have been developed. The works in [23], [22] learns a single probability distribution modeling the whole search space. To promote the diversity of the population, the approaches in [24], [20], [11] partition the population into different clusters and learn a separate local model for each cluster. Depending on the specific algorithm, clustering may be performed in the decision space [11] or in the objective space [24], [20]. A mixture distribution can be used to aggregate the local probabilistic models [20].

Among the evolutionary methods, MOEDAs [19] are closest to our approach, sharing with ALP the extraction of global statistical information from the current data. Therefore in the following, we focus on the MMEA algorithm [24], a state-of-the-art MOEDA which will be the touchstone in our experiments.

B. The MMEA algorithm

Pareto-optimal solutions are typically not scattered at random over the decision space. Their distribution indeed exhibits a clear geometric shape. In particular, under mild smoothness conditions, both the PF in the objective space and the PS in

```
1.  procedure EDA
2.      input: multi-objective optimization problem, population size $|P|$
3.      output: finite set of points (i.e., population) $P$ approximating the PS
4.      generate the initial population $P$ /* Initialization */
5.      do
6.          Generate a model $M$ for the distribution of the individuals in $P$ /* Modeling */
7.          Sample $M$ to generate a set of new solutions $Q$ /* Reproduction */
8.          Select $|P|$ from $P \cup Q$ and replace the individuals in $P$ by them /* Selection */
9.      until (termination_criterion)
10.     return population $P$
```

Fig. 7. Generic EDAs framework. The termination criterion is typically represented by a pre-defined number of iterations (i.e., population generations within the Evolutionary metaphor).

the decision space are piecewise continuous manifolds [11]. While the dimensionality of the PF manifold is $m-1$, with m the number of objective functions, the dimensionality of the PS manifold is typically larger and unknown. The RM-MEDA algorithm [11] has been designed to tackle MOPs where the PS dimensionality is $m-1$. MMEA [24] extends RM-MEDA to handle the case of unknown larger PS dimensionality. The rationale for MMEA is the assumption that the population of the EMOAs becomes scattered around the PS as the search goes on. Therefore, the PS approximation generated by EMOAs is provided by the candidate solutions located within the central region of the area containing the current population. Under this assumption, promising candidate solutions can be locally modeled by a probability distribution ξ whose centroid is the approximation to the PS. Each solution of the current population is considered an independent observation of vector ξ.

In order to generate a model of the Pareto set, MMEA clusters the population members in the decision space into a number of sub-populations based on the distribution of their images in the objective space. This approach favours diversity in the objective space. As the PF dimensionality is assumed to be $m-1$, the PF is approximated by fitting a $m-1$-dimensional simplex over the objective vectors. In detail, the i-*th* vertex of the simplex S, $i = 1, 2, \dots, m$, is the vertex corresponding to the non-dominated solution with largest f_i value. Simplex S is shifted along its normal direction by the minimum distance enabling each point in S to be non-dominated by the vectors in the population. Finally, the volume of S is arbitrarily increased by a factor α to favour the exploration of the objective space. The objective vectors are clustered around k reference points uniformly distributed over the simplex.

By clustering in the objective space, the solutions in the decision space are partitioned into k (possibly overlapping) sub-populations. Each subpopulation provides a local approximation of the PS. In particular, a linear local approximation of the PS is obtained by applying the principal component analysis technique (PCA) over each subpopulation independently. The unknown PS dimensionality d is locally estimated by the number $\tilde{d}_j$, $j = 1, 2, \dots k$, of principal components explaining a preset percentage θ of the variation in the j-*th* subpopulation data, where θ is a control parameter of MMEA. The first $\tilde{d}_j$ principal components obtained by PCA identify the axes of the hyper-cuboid Ψ_j locally approximating the PS. The range of each axis is provided by the smallest hyper-cuboid, centered at the empirical mean of the subpopulation solutions, containing the projections of all the subpopulation solutions on the space spanned by the $\tilde{d}_j$ axes from the center of the cuboid. The range of each axis is then enlarged to increase the cuboid volume by a factor β, with β being a control parameter of MMEA. The rationale for this choice is the promotion of the exploration of promising regions in the decision space (with respect to the directions explaining most of the variance in the data).

The local distribution of promising solutions ξ_j for the j-*th* subpopulation is modeled by a uniform probability vector on the j-*th* hyper-cuboid and a Gaussian additive noise component. The k local probability model ξ_j are sampled to obtain a set of new candidate solutions (reproduction by sampling) which are used to refine the current population. The sampling procedure consists of selecting the hyper-cuboid Ψ_j with probability P_j, picking uniformly at random a point $\mathbf{x}^r$ from Ψ_j and perturbing $\mathbf{x}^r$ by an additive noise vector sampled from the noise component. When the sampled solution does not belong to the feasible region Ω in the decision space, it is discarded and replaced by a new candidate selected uniformly at random within Ω.

C. *Surrogate-based EMOAs*

EDAs use statistical models to generate the offspring of the population, which is then evaluated w.r.t the original objective functions to build the new generation. A different usage of statistical models in EMOAs consists of learning a *surrogate* of the original objective functions. The approximated objective values predicted by the surrogate model are used to estimate the quality of candidate solutions, without evaluating the original MOP functions. The surrogate model (also referred to as *meta-model* or *response surface model*) is therefore an approximation of the original fitness surface. This approximation is used to reduce the number of evaluations of the original MOP functions. The surrogate model is usually learnt from solutions already evaluated. Different techniques from the statistical learning community have been used to learn the surrogate models [17]. Analogously to the model generation in EDAs, because a surrogate model is usually trained, refined and evaluated several times, a trade-off among its accuracy and its computational cost has to be considered.

With respect to ALP, a relevant surrogate-based approach is the recent MOEA/D-EGO algorithm [25], a variant of the MOEA/D algorithm [26]. The Multi-Objective Evolutionary Algorithm based on Decomposition (MOEA/D) decomposes the original MOP into s SOPs, obtained by scalarization. The objective of the i-*th* SOP, $i = 1, 2, \ldots s$, is a parametric combination $h(\mathbf{x}|\lambda_i)$ of the original MOP objectives, with λ_i being the vector containing the scalarization parameters. The generated SOPs differ from each other in the values of the parameter vector. A neighbourhood relation among the SOPs is defined, based on the distances between their parameter vectors. Each SOP is optimized simultaneously by using information mainly from its neighbouring SOPs. This approach assumes that neighborhood problems are likely to have similar solutions, and therefore they can profitably exchange information during their joint optimization. The population of MOEAD/D consists of the current solutions of the s SOPs.

MOEA/D-EGO learns a surrogate GP model for each objective of the original MOP. The training set is the current population which contains all the solutions evaluated so far. A predictive model is then derived for each SOP, i.e., for each parametric combination $h(\mathbf{x}|\lambda_i)$ of the MOP objectives. In particular, the i-*th* predictive model estimates the probability distribution of the values of $h(\mathbf{x}_*|\lambda_i)$ at each test input $\mathbf{x}_*$. The probability distribution is used in combination with the expected improvement metric $\psi_i(\mathbf{x})$, which measures the merit of evaluating input $\mathbf{x}$ (by the original MOP objectives) to optimize $h(\mathbf{x}|\lambda_i)$. The s expected improvements $\psi_i(\mathbf{x})$, $i = 1, 2, \ldots s$, are optimized simultaneously to obtain points $\tilde{\mathbf{x}}_i$, with $\tilde{\mathbf{x}}_i$ being the approximate solution maximizing $\psi_i(\mathbf{x})$. The simultaneous optimization is obtained by applying the MOEA/D algorithm. MOEA/D optimizes $\psi_i(\mathbf{x})$ by exploiting the solutions that maximize the expected improvements of the neighbouring SOPs. A subset of the points $\tilde{\mathbf{x}}_i$, $i = 1, 2, \ldots s$ is then selected to be evaluated by the original MOP objectives. The selected points are finally included into the current population.

Even if MOEA/D and MOEA/D-EGO share with ALP the generation of scalar SOPs and the adoption of GP models, their approach is very different from ALP. MOEA/D and MOEAD/D-EGO use a pre-specified number of parametric SOPs with uniformly distributed parameter vectors defined at initialization. Pareto-optimal solutions (or approximation thereof) are obtained by solving the subproblems generated by the set of weighting vectors. However, it may be difficult to select suitable parameter vectors for obtaining solutions evenly distributed over the whole PF. For example, in the weighted sum aggregation method [1], an uniform distribution of parameter vectors does not necessarily generate a set of solution evenly distributed over the PF [1].

On the other hand, ALP learns a GP global model of the PF from a training set of approximate Pareto-optimal vectors. The training set is generated on-line, with each training example being the solution of a SOP. Therefore, the SOPs are dynamically generated during the search process rather than being defined at the initialization stage. In particular, the SOPs generation is driven by the predictive uncertainty of the learnt GP model. Furthermore, a single Gaussian process model is learnt by ALP to solve a regression task, i.e, generating an analytic approximation of the PF. In MOEAD/D-EGO the GP models are used to efficiently solve SOPs, i.e., to find the optimum of a function with the lowest number of function evaluation possible. Therefore, in MOEA/D the learnt GP model is used as a response surface to search for likely candidate global optima.

In ALP, the training examples are generated by solving the non-linear problem in Eq. (3). Our future research will consider surrogate-based techniques for single-objective optimization, in order to reduce the computational cost of ALP supervised information.

VI. Experimental results

ALP is tested over the RM-MEDA benchmark introduced in [11] to evaluate the ability of EDAs in recovering the Pareto front. As the best results over the RM-MEDA benchmark has been achieved by the MMEA algorithm [24], the latter represents the touchstone in our experimental studies. In particular, the optimal setting of MMEA parameters is taken from paper [24].

The experimental studies are described as follows. First, the RM-MEDA benchmark is outlined. The setting of the ALP algorithm is then detailed and a single ALP run is showed, in order to provide a paradigmatic case of the PF approximation refinement observed during the learning iterations. The comparison with the MMEA algorithm is given in Subsec. VI-E, following the description of the comparison metrics adopted. Finally, ALP is applied on the welded beam design MOP [27], a widely-used benchmark introduced in the spirit of real world optimization tasks.

A. The RM-MEDA benchmark

For a comparison with ALP, we consider the bi-objective problems of the RM-MEDA benchmark [11]. By using the numbering adopted in paper [24], the benchmark set is thus formed by the instances ZZJ08-*F1*, ZZJ08-*F2*, ZZJ08-*F3*, ZZJ08-*F5*, ZZJ08-*F6* and ZZJ08-*F7*. These instances are derived from the widely-known Zitzler-Deb-Thiele (ZDT) test problems [28]. The PS of the bi-objective ZDT problems is parallel to the coordinate axes, because of the lack of dependencies among the decision variables. This deficiency introduces a bias in favor of the commonly used crossover operator: if two solutions are Pareto-optimal, their offspring is likely to be Pareto-optimal. Furthermore, variable linkages exist in many applications and their inclusion into test suites has been suggested by different works [29], proposing variable transformations for introducing dependencies among the decision variables. The RM-MEDA instances F1, F2, F3 introduce linear linkages in the formulation of the ZDT1, ZDT2 and ZDT6 test problems [28]. Nonlinear dependencies among the decision variables have been added to the definition of the same ZDT instances to generate the RM-MEDA problems F5, F6, F7. Their Pareto sets consist of bounded continuous curves.

B. ALP setting

In order to avoid any bias favouring our method over the competitor, the PF manifold is arbitrarily expressed by considering z_2 as a function of z_1: $z_2 = g(z_1)$. To apply the ALP algorithm, the Ideal and Nadir points $\mathbf{z}^{ID}$ and $\mathbf{z}^{N}$ of the above MOPs are computed offline. In particular, the domain $[z_1^{ID}, z_1^{N}]$ of the regression task is equal to $[0,1]$ for all the considered MOPs, with the exception of problems *F3* and *F7* where the domain is the interval $[0.281, 1]$. ALP is implemented in Matlab R2008a. In particular, the optimization problem (3) generating supervised information is solved by using the continuous local search algorithms for constrained optimization provided by the Matlab Optimization Toolbox$^{\text{TM}}$ library. In detail, a sequential quadratic programming algorithm [30], [31] is used, except for problems *F3* and *F7*, where an interior-point method [32] is adopted. The choice of the optimization strategy is based on the observed convergence rate to the (local) optima of the function f_m. Both strategies are implemented by the "fmincon" Matlab routine. A single run of "fmincon" algorithm is performed, initializing at random the starting point and limiting the number of scalarized function evaluations to 3000. For the considered MOPs, within the value of 3000, the algorithm usually converges to a (local) minimum of the function f_m. In general, ALP is a flexible framework enabling the usage of alternative constrained optimization algorithms (e.g., derivative-free approaches) rather than the "fmincon" routine to solve problem (3).

ALP training set is initialized by selecting two training examples uniformly at random within the input space. In the performed experiments, the mean function of the prior GP is the line $a * x + b$, with $\{a, b\}$ being the mean function hyperparameters. Three candidate forms for the covariance function are considered: the squared exponential (Eq. (5)), the neural network and the Matérn covariance functions (see book [6] for their formulation). In combination with the mean and covariance functions, we make use of a white Gaussian noise model with variance ρ_n^2. The hyperparameters vector $\boldsymbol{\theta}$ thus includes ρ_n^2, the mean function hyperparameters a and b and the covariance function hyperparameters. The model selection phase consists of two tasks: 1) the choice of the functional form for the covariance function, 2) the optimization of the hyperparameters $\boldsymbol{\theta}$. Both tasks are accomplished in one step by evidence maximization (see Eq. (23)): for each candidate covariance function, the vector $\boldsymbol{\theta}$ is optimized. The setting with largest likelihood value is then selected. In particular, the optimization task is solved by applying a conjugate gradient algorithm. Ten runs of the algorithm are performed, each one consisting of 50 conjugate gradient steps. A different starting point $\boldsymbol{\theta}_{init}$ is used for each run. A suitable selection of the starting point, based on the prior knowledge about the desired PF model, drives the optimization algorithm towards (local) minima which provide plausible interpretations of the training data. In particular, the slope a of the mean function in vector $\boldsymbol{\theta}_{init}$ is initialized to the value -1. In fact, any connected PF of a bi-objective minimization problem can be modelled by a strictly decreasing function. The gain from the inclusion of prior knowledge in the starting point design is more significant at the initial iterations of the ALP algorithm, when the number of training examples is limited.

C. A single ALP run in detail

A single run of the ALP algorithm over MOP ZZJ08-*F2* is shown in Fig. 8. The predictive GP model is depicted by the solid line, while the dotted line represents the unknown Pareto front $z_2 = g(z_1)$. The shaded area represents the 95% confidence interval for the predictive GP model. At the first iteration, with a couple of training examples only, the learnt PF model is a line, while the order of magnitude of the predictive variance is 10^{-6} and thus the shaded area cannot be visualized. The inclusion of the new training example located at input $z_1 = 0.01$ (triangular marker over the x-axis) changes the slope of the line, while constant predictive uncertainty over the input space is observed. In this case, the ALP algorithm places the new query point at the input $\mathbf{z}_1$ maximizing the minimum distance from the training examples. At the third iteration, the accuracy of the learnt model improves. Predictive uncertainty increases when moving away from training examples and the shaded area entails the Pareto front to be modeled. The uncertainty sampling method selects the input $z_1 = 0.29$ as the new training example. At the fourth iteration, with 1536 evaluations of the MOP objective functions $\mathbf{f}(\mathbf{x})$, the PF is successfully recovered. Additional refinement iterations slightly improve the accuracy of the learnt model (however the improvements cannot be visually observed), while the predictive uncertainty becomes null. In general, the adoption of the active learning to generate the training examples favours the anytime property: when increasing the number of functions evaluations, the PF approximation improves.

D. Comparison metrics

The performance of the ALP and MMEA algorithms is evaluated by measuring the quality of the recovered PF approximation w.r.t its cost. The cost is expressed by the number of objective function evaluations rather than the CPU time, as the evaluation of complex and expensive objective functions is the computational bottleneck of the real-world MOPs [3]. Furthermore, the number of function evaluations is the commonly used and generally accepted measure for the run time in the multi-objective optimization literature. The quality of the PF approximation is estimated by the inverted generational distance (IGD) and the hypervolume difference (IH^{-}) metrics. Each metric evaluates the quality of the PF approximation recovered by an evolutionary multi-objective algorithm (EMOA) by measuring both the convergence and the diversity of its population. The adopted metrics are calculated according to the procedure described in paper [24].

A test set P^* of Pareto-optimal objective vectors uniformly spread over the PF is generated. With P denoting the population of the EMOA, the IGD metric is defined as follows:

$$\text{IGD}(P^*, P) = \frac{\sum_{\mathbf{z}^* \in P^*} d(\mathbf{z}^*, P)}{|P^*|} \qquad (24)$$

where $d(\mathbf{z}^*, P)$ is the *minimum* Euclidean distance from the Pareto-optimal vector $\mathbf{z}^*$ to any objective vector in P, while

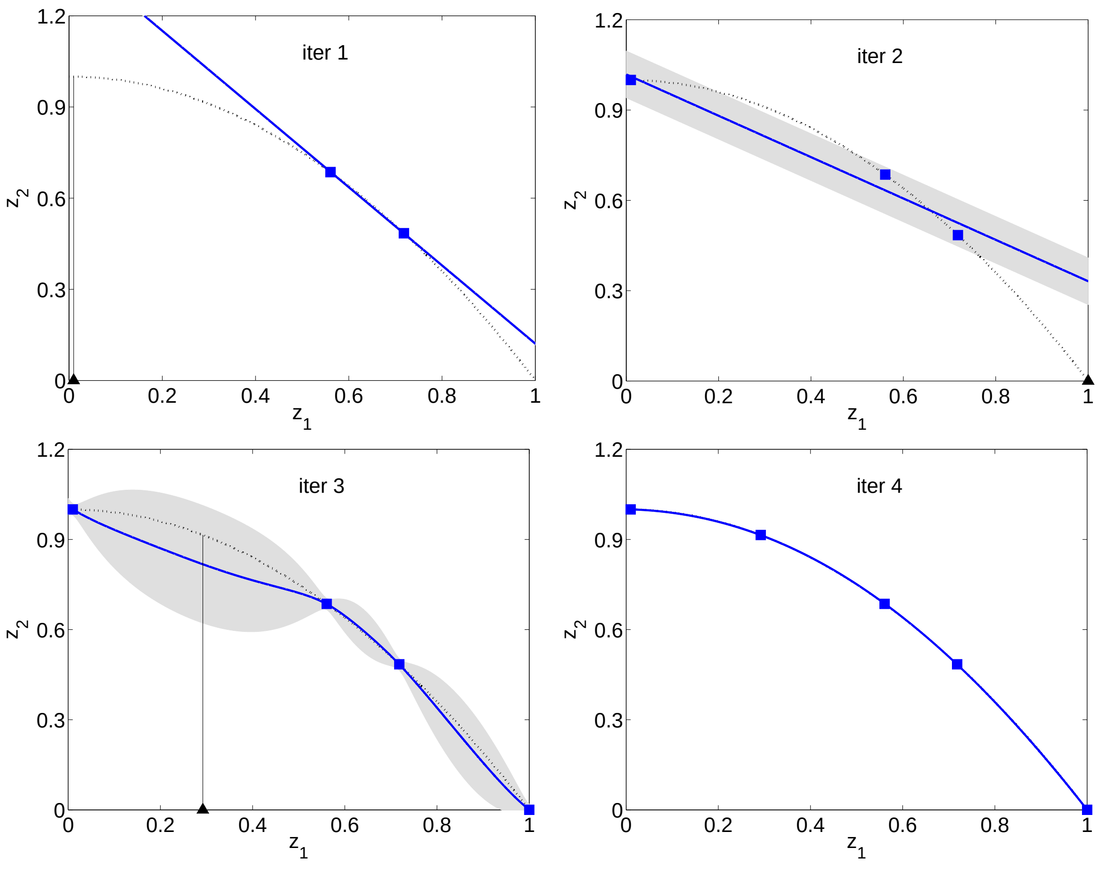

Fig. 8. Generation of most informative training examples while solving MOP *F2*. Each figure depicts the PF (dotted line) of the learnt model (solid line). The Figures refer to the first, second, third and fourth refinement iterations, respectively. Shaded area denotes the 95% confidence interval for the predictive GP model. On the *x*-axis, the most informative training example, selected by active learning, is showed by a triangular marker. At the fourth iteration (second row, right figure) the PF is correctly learnt.

$|P^*|$ indicates the cardinality of P^*. According to the notation used in paper [24], the IGD metric is denoted as IGDF because it measures the quality of the PF approximation (i.e., P is a set of objective vectors and $d(\mathbf{z}^*, P)$ is the Euclidean distance in the objective space). The dominated hypervolume metric IH(P) (hypervolume for short, also known as Lebesgue measure or S-metric) measures the size of the objective space dominated by the population P and bounded by a fixed reference point [33]. The hypervolume difference metric IH$^-(P^*, P)$ is defined as:

$$\mathrm{IH}^-(P^*, P) = \mathrm{IH}(P^*) - \mathrm{IH}(P) \qquad (25)$$

The IGDF and the IH$^-$ are alternative measures of *both* the diversity and the convergence of P, provided that the value $|P^*|$ is large enough to represent the whole PF. The lower is the value IGDF(P^*, P) and IH$^-(P^*, P)$, the better is the quality of the population P approximating the PF. To have a low value of IGDF(P^*, P) and IH$^-(P^*, P)$, the population P must be close to the PF and cannot miss any part of the whole PF. The reference point for the hypervolume computation and the cardinality $|P^*|$ used in the experiments reported here are taken from paper [24].

The above metrics measure the quality of a finite set of points (population) approximating the PF. When adopting these metrics to evaluate the ALP algorithm, which recovers an analytical PF model $\tilde{g}$ rather than a discrete PF representation, the population P is computed by sampling the analytical PF model. In particular, $|P^*|$ sample vectors of the form $\tilde{\mathbf{z}} = \{\mathbf{z}_I^*, \tilde{g}(\mathbf{z}_I^*)\}$ are generated by evaluating $\tilde{g}(\mathbf{z}_I^*)$ for each $\mathbf{z}^* \in P^*$. Any dominated objective vector among the $|P^*|$ samples generated is discarded. Furthermore, as the ALP algorithm may generate PF models which are (partially) infeasible (this behavior has been occasionally observed during the initial refinement iterations), the absolute value of the hypervolume difference metric (Eq. (25)) is considered.

E. Detailed comparative results

Fig. 9 and Fig. 10 compare the performance of the ALP and MMEA algorithms. The curves depict the evolution of the IGDF (left column) and IH$^-$ (right column) metrics over the number of evaluations of the MOP objective vector $\mathbf{f}(\mathbf{x})$. The dashed and solid lines correspond to the ALP and MMEA algorithms, respectively. Each point is the median value over 20 runs, executed with different seeds. Error bars denote the interquartile range (IQR) of data distribution.

With the exception of MOPs *F3* and *F5*, the ALP algorithm dominates the MMEA technique, achieving on average a PF approximation with sensibly better quality, in terms of both performance metrics. For example, over MOP *F1*, within 10000 function evaluations, a more accurate Pareto front is recovered by the ALP algorithm, and after 10000 evaluations, the quality of ALP solution is at least twelve times better than the quality of MMEA solution.

A stable behavior is observed for both algorithms. The large size of the ALP error bars in the case of MOP *F2* is due to the logarithmic scale. In fact, the order of magnitude of ALP IQR values for MOP *F2* is actually lower than that of MMEA ones. For example, when 70000 function evaluations are performed, the orders of magnitude of the IQR values for ALP and MMEA results (measured by the IGDF metric) are 10^{-5} and 10^{-3}, respectively.

There is no qualitative difference in terms of the IGDF metric among ALP and MMEA results for MOP *F5*, but the ALP technique achieves a better average performance in terms of the IH^{-} measure. When more than 30000 function evaluations are used to solve MOP ZZJ08-*F3*, the performance of ALP is not better than the MMEA algorithm. However, within 20000 function evaluations, the quality of ALP solution is better by at least a factor ten than the PF recovered by MMEA. Reasonable approximations obtained within few function evaluations are in many cases preferred to more accurate solutions which require a heavier computational effort. In fact, as noted by Lovison in work [10], "even a roughly sketched global picture of the whole situation can give crucial information on the problem at hand, suggesting correctly the location of paramount zones".

F. Real world MOP

ALP has been applied to solve a benchmark problem in the spirit of real-world applications, the welded beam design MOP [27]. It is a widely-studied *constrained* optimization problem, consisting of the design of a beam that is welded onto another beam to carry a certain load. In particular, the (point) load consists of a force $F = 6000$ lb acting on the free end of the beam. The overhang portion of the beam has a length of 14 inch. Four decision variables are defined: the beam thickness b, the beam width t, the weld length l, and the weld thickness h. The geometry and the loading condition of the beam are shown in Fig. 11.
Denoting the variable vector $\mathbf{x} = (b, t, l, h)$, the problem consists of finding the optimal configuration $\mathbf{x}^*$ which minimizes both the beam cost (first objective, $f_1(\mathbf{x})$) and the vertical deflection of the beam end (second objective, $f_2(\mathbf{x})$). The objectives are conflicting. In fact, cost minimization is achieved by minimizing the beam dimension, corresponding to small values for all the four decision variables. However, the smaller the beam dimensions, the larger the end deflection tends to be. Minimum deflection at the beam end (i.e., maximum rigidity of the beam) is obtained by increasing the values of the decision variables. The different trade-offs among the conflicting objectives can be obtained by recovering the Pareto front of the problem. The mathematical formulation of the problem is as follows:

$$\min_{\mathbf{x}} \quad z_1 = f_1(\mathbf{x}) = 1.10471\ h^2\ l + 0.04811\ t\ b\ (14.0 + l) \tag{26}$$

$$\min_{\mathbf{x}} \quad z_2 = f_2(\mathbf{x}) = 2.1952/(t^3\ b) \tag{27}$$

$$\text{subject to} \quad 13600 - \tau(\mathbf{x}) \geq 0 \tag{28}$$

$$30000 - \sigma(\mathbf{x}) \geq 0 \tag{29}$$

$$b - h \geq 0 \tag{30}$$

$$P_c(\mathbf{x}) - 6000 \geq 0 \tag{31}$$

$$0.125 \leq h, b \leq 5.0$$

$$0.1 \leq l, t \leq 10.0$$

where:

$$\begin{aligned}
\tau(\mathbf{x}) &= \sqrt{(\tau')^2 + (\tau'')^2 + (l\ \tau'\ \tau'')/\sqrt{0.25\ (l^2 + (h+t)^2)}} \\
\tau' &= \frac{6000}{\sqrt{2}\ h\ l} \\
\tau'' &= \frac{6000\ (14 + 0.5\ l)\sqrt{0.25\ (l^2 + (h+t)^2)}}{1.414\ h\ l\ (l^2/12 + 0.25\ (h+t)^2)} \\
\sigma(\mathbf{x}) &= \frac{504000}{t^2\ b} \\
P_c(\mathbf{x}) &= 64746.022\ (1 - 0.0282346\ t)\ t\ b^3
\end{aligned}$$

The first constraint (Eq. (28)) guarantees that the shear stress $\tau(\mathbf{x})$ at the support location of the beam is smaller than the maximum shear strength of the material (13600 psi). Furthermore, the normal stress $\sigma(\mathbf{x})$ at the support location of the beam must be smaller than the maximum yield strength of the material, 30000 psi (second constraint). Finally, the thickness of the beam must not be smaller than the weld thickness (Eq. (30)) and the buckling load $P_c(\mathbf{x})$ of the beam along direction t must be greater than the applied load F (fourth constraint). A configuration $\mathbf{x}$ violating any of the four constraints is infeasible.

ALP learns the Pareto front of the welded beam MOP by considering the end deflection objective as a function g of the cost objective (z_1). The domain of g is $[z_1^{ID}, z_1^{N}]$, with the values $z_1^{ID} = 2.381$ and $z_1^{N} = 36.421$ of the Ideal and Nadir points, respectively, taken from paper [34].

The best Pareto front model over 20 runs recovered by ALP within 5000 function evaluations is depicted in Figure 12 (left). The "true" Pareto front, obtained by enumeration, is showed by a dotted line, while the solid line represents the ALP approximation. Figure 12 (right) presents the aggregate performance results over 20 runs of ALP. The performance is measured by the root mean squared error (RMSE), which evaluates the difference between ALP approximation and "true" PF of the welded beam problem. The curve is obtained by reporting the median RMSE values over the computational effort. The error bars denote the 25-*th* and the 75-*th* percentiles. When increasing the number of function evaluations, the accuracy of the PF approximation rapidly improves, until ALP converges to a RMSE value lower than 2×10^{-3}. Furthermore, after 500 initial function evaluations, ALP results in less variability than the more unstable performance observed within the 500 initial function evaluations, confirming the efficiency of the active learning strategy.

VII. Handling disconnected Pareto fronts

The ALP framework described in Sec. III assumes to known the domain of the function $z_d = g(\mathbf{z}_I)$ characterizing the PF. In the case of bi-objective MOPs with connected PF

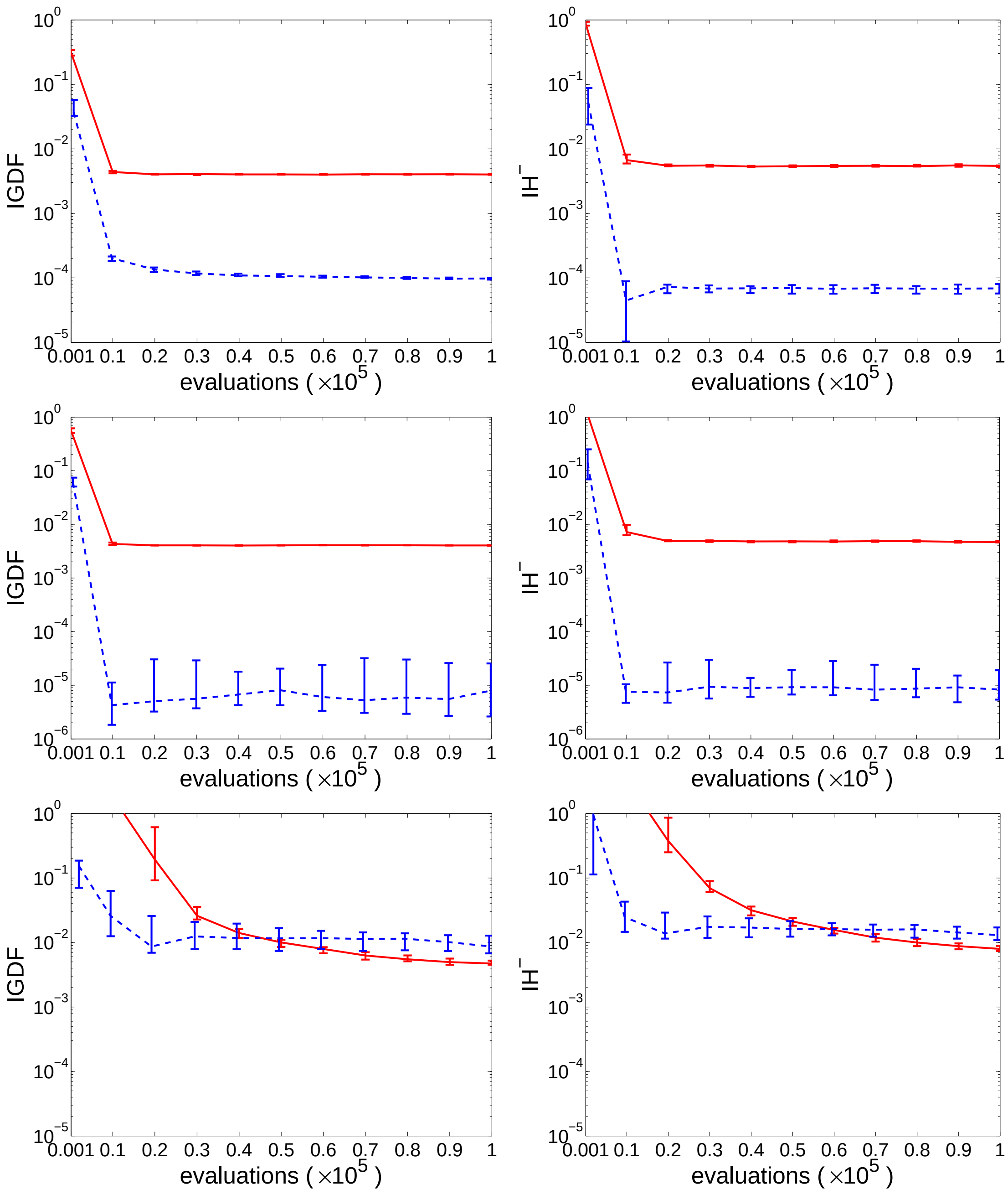

Fig. 9. Performance metrics (median values) evolution over the number of function evaluations. Error bars represent the IQR of data distribution. Figures in rows one, two and three refer to MOP *F1*, *F2*, *F3*, respectively. Dashed lines plot ALP results, while solid lines depict MMEA performance.

characterized by $z_2 = g(z_1)$, this prior information can be easily obtained by computing the components z_1^{ID} and z_1^N of the ideal and the Nadir point, respectively, or by asking the decision maker for the interval of the interesting values of the objective z_1.

When the PF is *disconnected*, the function g is discontinuous and its domain cannot be defined by a couple of z_1 values only. Therefore, assuming a priori knowledge of the domain of g is impractical. However, a disconnected PF can be identified by learning a regression function $z_2 = h(z_1)$ approximating the entire lower boundary of the feasible objective region, including the dominated portions of the boundary (e.g., the concave parts). As a matter of fact, the PF is a subset of the objective boundary and thus a model of the disconnected PF can be extracted straightforwardly by sampling the function h and keeping the non-dominated samples only (e.g., the points lying in the concavities of the function h are discarded). For example, Fig. 13 (left) shows the feasible objective region of the ZDT3 problem, taken from the ZDT benchmark suite [28]. The boundary of the feasible region (gray-shaded area) is the thin black curve, consisting of both convex and concave parts. The disconnected PF is depicted by the bold-marked portions of the black curve.

When the ALP algorithm is used to learn the function h,

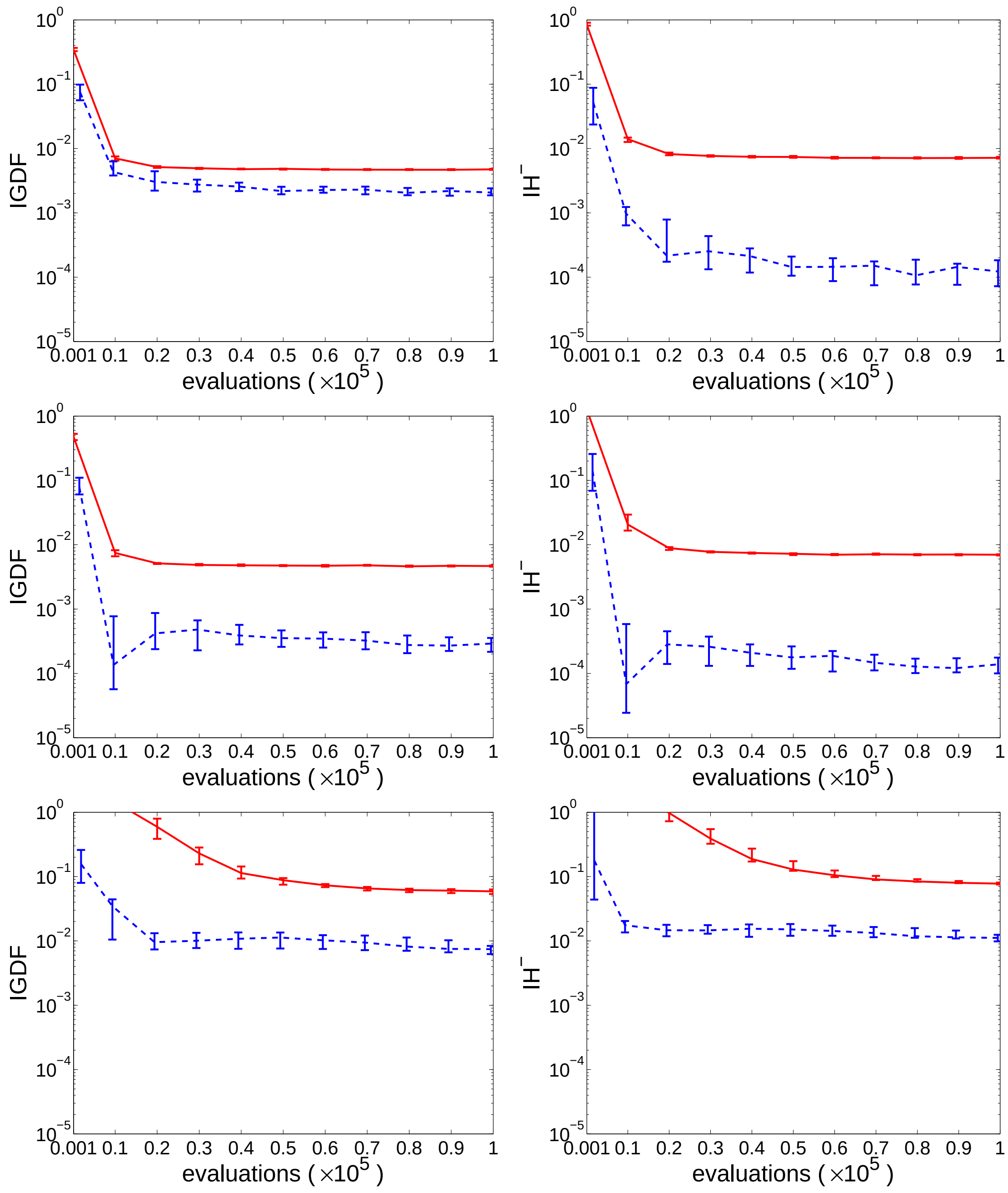

Fig. 10. Performance metrics evolution over the number of function evaluations in the case of MOP *F5* (row one), *F6* (row two) and *F7* (row three). Median values with IQR error bars are reported. Dashed and solid lines depict ALP and MMEA results, respectively.

both dominated and non-dominated examples are needed. The former are in fact used to model the concave portions of the boundary (to be discarded when presenting the PF). In order to allow this, the dominance-based filtering discarding false Pareto-optimal training examples has to be switched off. Otherwise, the learning phase may get stuck when trying to model the concavities of the boundary, as other dominating training examples will likely be present in this case (see Fig. 13 (right)).

In order to model entire feasible region boundaries, an additional functional form has been included in the set of candidate covariance functions considered by the GP model selection procedure. This consists of a combined function, obtained summing up a periodic covariance function c_p with a squared exponential covariance function c_s (see [6] for their formulation). The rationale for the choice of function c_p is the modelling of the concavities characterizing the boundary of the feasible area. The combination with function c_s enables a decay away from exact periodicity, as the boundary shape is not expected to be exactly periodic. Furthermore, the squared exponential covariance function models the smooth trend characterizing disconnected Pareto fronts, which, in the case of bi-objective minimization problems, are monotonically decreasing piecewise continuous functions.

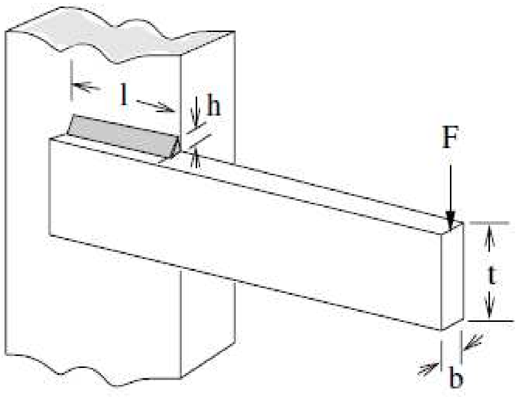

Fig. 11. The welded beam design MOP. The four decision variables b, t, l, h are marked. The force F acts on the free end of the beam.

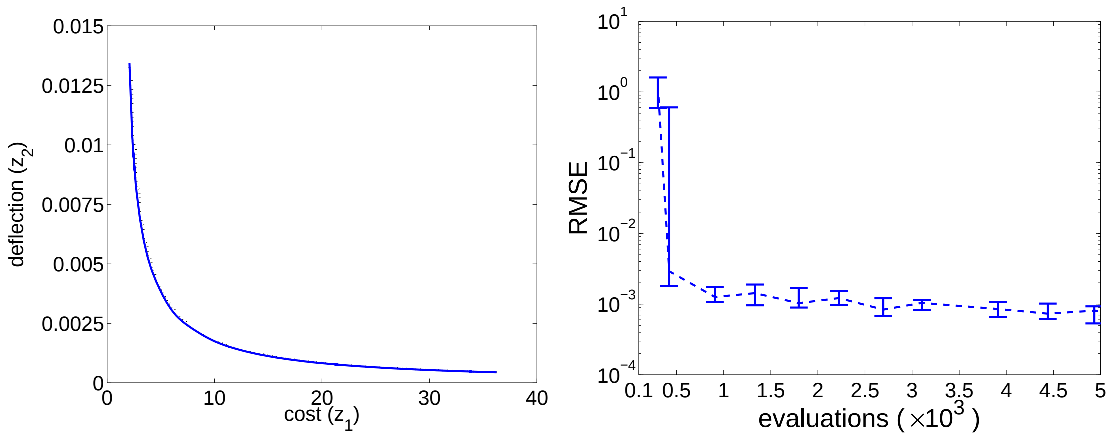

Fig. 12. Welded Beam problem. (Left) The best Pareto front approximation (solid line) recovered by ALP with less than 5000 function evaluations. The “true” PF is represented by the dotted line, overlapping with the solid line. (Right) ALP performance measured by the RMSE evolution over the function evaluations number. Median values over 20 runs are reported. Error bars represent the interquartile range.

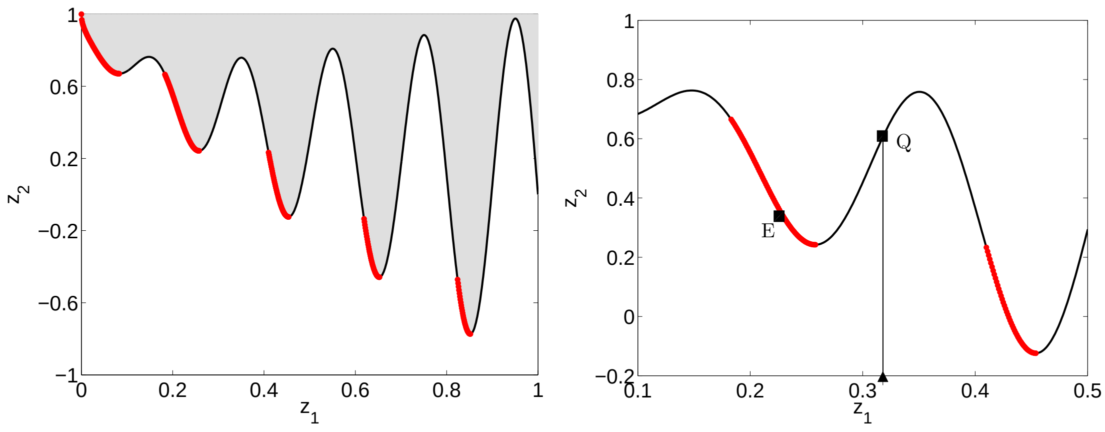

Fig. 13. (Left) ZDT3 problem. The boundary (black thin curve) of the feasible region (gray-shaded area) contains both convex and concave parts. The disconnected PF is formed by the bold-marked curve segments. (Right) When the query point (triangular marker on the z_1-axis) does not belong to the domain of g, the *exact* solution of the scalarized program providing supervised information generates an objective vector Q that may be dominated by the current training examples (point E in the graph).

Let us note that this combined covariance function is not designed for a specific MOP, but is a general choice driven by the typical form of feasible region boundaries. Furthermore, we are not imposing this covariance function in the prior GP, but only suggesting it to the model selection procedure as a candidate covariance function. The identification of the covariance function (and its parameters setting) that best fits (i.e., explains) the training data is performed by the model selection procedure.

No further modifications are required to adapt the ALP algorithm presented in Sec. VI to model disconnected PF.

VIII. Case studies with disconnected PF

The ability of ALP in learning disconnected Pareto fronts is evaluated over four well known MOPs, ZDT3 [28], D99 [3], [35], TNK [36] and KUR [37]. The selected MOPs differ from each other in several features. While ZDT3 has 30 decision variables, the decision space of both D99 and TNK is bi-dimensional and KUR has three decision variables. The Pareto optimal solutions of ZDT3, D99 and TNK lie in a one-dimensional piecewise continuous manifold, while three dimensions are needed to represent the Pareto optimal solutions of the KUR problem. Furthermore, in the case of ZDT3, D99 and TNK the manifold formed by the Pareto-optimal solutions lies on the boundary of the decision space, while the Pareto optimal solutions of KUR are located around the center. The shape of the PF also varies among the MOPs considered. For example, the PF of ZDT3 is formed by five convex curves, while those of D99 and KUR are represented by six non-convex curves and by three non-convex curves plus an isolated point, respectively. Finally, TNK differs from the rest of the MOPs by being a constrained optimization problem.

The MMEA algorithm, which was used as touchstone in the experiments of Sec. VI, has been designed for connected Pareto fronts and cannot represent a fair ALP competitor over the four selected problems. ALP is therefore compared with the popular state-of-the-art genetic algorithm NSGA-II [38]. We used the original NSGA-II code provided by the authors (version $1.1.6$).

The experimental results detailed in the rest of this section show that the ALP algorithm outperforms NSGA-II over the ZDT3, D99 and TNK MOPs, while it fails in learning an accurate model of the KUR PF. A cross-over point is observed when comparing the algorithms over the TNK MOP. As a matter of fact, within 6000 function evaluations ALP converges much faster than NSGA-II to a reasonable PF model. More than 12000 function evaluations are instead needed to observe NSGA-II performance exceeding ALP performance. The failure of ALP learning process in the case of KUR MOP is due to the noisy training dataset obtained when generating the supervised information. ALP can however detect this suboptimal behavior, by observing the predictive uncertainty of the GP model, which does not decrease when increasing the number of learning iterations. In particular, the predictive uncertainty remains exceptionally high also in regions of the regression domain densely populated by training examples.

A. Detailed results over the ZDT3 MOP

There are no discontinuities in the decision space of the ZDT3 problem (in particular, the Pareto set is connected). Its PF is convex, but disconnected. In particular, it is formed by the five different bold-marked curves in Fig. 13 (left). The formulation of the ZDT3 problem is as follows:

$$\min_{\mathbf{x}} \quad z_1 = f_1(\mathbf{x}) = x_1$$
$$\min_{\mathbf{x}} \quad z_2 = f_2(\mathbf{x}) = g(\mathbf{x}) * [1 - \sqrt{f_1(\mathbf{x})/g(\mathbf{x})} - (f_1(\mathbf{x})/g(\mathbf{x}) * \sin(10\pi f_1(\mathbf{x}))]$$

subject to

$$g(\mathbf{x}) = 1 + \tfrac{9}{(n-1)} * \sum_{i=2}^{n} x_i$$
$$x_i \in [0,1], i = 1 \ldots 30$$

In running the NSGA-II algorithm, we use the specific parameter setting for ZDT3 which is available with the code. In this setting, SBX and polynomial mutation are the operators for crossover and mutation, respectively. The distribution indexes for both operators are $\eta_c = 15$ and $\eta_m = 20$, respectively. The crossover probability is 0.9 and the mutation probability is $1/n$, where $n = 30$ is the number of decision variables of ZDT3. A population of size 100 is used, while the algorithm has been kept running until generation 200.

Twenty runs with different seeds are executed for both the ALP and NSGA-II algorithms. The convergence to the Pareto front of the problem is depicted in Fig. 14. In particular, the left (right) graph reports the evolution of the IGDF (IH$^-$) metric over the number of function evaluations. The dashed and solid curves correspond to the ALP and NSGA-II algorithms, respectively. The curves denote the average performance (median value over the 20 runs) of the algorithms, while the error bars plot the interquartile range (IQR) of the data distribution.

Within 6000 function evaluations, the performance of ALP and NSGA-II are comparable. However, when progressively increasing the number of function evaluations from 6000 up to 20000, the quality of the ALP model improves more rapidly than that of the NSGA-II one. This behavior is observed for both the performance metrics reported. In detail, ALP performance converges to an IGDF value less than 10^{-3}, about one order of magnitude better than that of NSGA-II. An even more pronounced behavior is observed for the IH$^-$ metric, where ALP performance is about two orders of magnitude better.

The sample percentiles denoted by the error bars demonstrate a stable behavior of the NSGA-II algorithm. The variability of ALP results is comparable with the dispersion of the NSGA-II samples, with the exception of the values observed at 8000 function evaluations. In this case, ALP data are more unstable than NSGA-II results. The higher variability of ALP performance is due to the GP model selection procedure, which in some occasional ALP runs at 8000 evaluations generates a model over-fitting the training data. A similar but less pronounced (variance halved) phenomenon is still observed at 10000 function evaluation in terms of the IH$^-$ metric (Fig. 14, right). When increasing the number of training

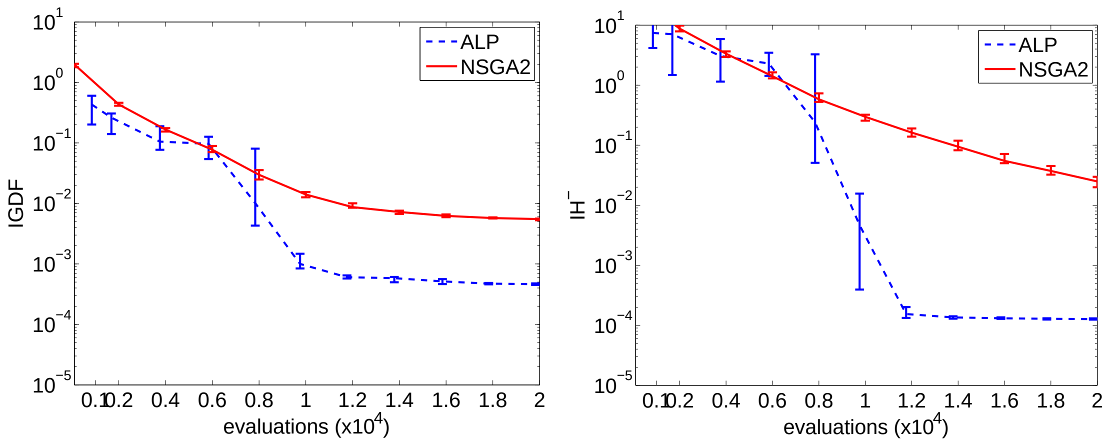

Fig. 14. Performance metrics (median values) evolution over the number of function evaluations observed in the case of ZDT3. Error bars represent the IQR of data distribution. The dashed line plots the ALP results, while the solid line shows the NSGA-II performance. In the left graph, the performance is measured by the IGDF metric, while the right graph plots the IH^{-} metric evolution.

examples (and thus the number of functions evaluations), a GP model that fits the data is learnt in each run of ALP and a high-quality PF approximation is obtained.

Finally let us note that the combined covariance function, involving a periodic component and an exponential decrease of the input correlation with the input distance, was selected 94% of the times by the model selection procedure, confirming that more informative prior information improves the performance of Gaussian process regression.

B. *Detailed results over the D99 MOP*

The second benchmark with disconnected PF is one of the main classical unconstrained MOPs described in [3] and originally introduced in [35]. It consists of two real decision variables, taking values in the range $[0, 1]$. The optimal solutions lie in the boundary of the decision space, and both the PS and the PF are disconnected. In detail, the MOP formulation is defined by the following equations:

$$\begin{array}{ll} \min\limits_{\mathbf{x}} & z_1 = f_1(\mathbf{x}) = x_1 \\ \min\limits_{\mathbf{x}} & z_2 = f_2(\mathbf{x}) = g(\mathbf{x}) * h(\mathbf{x}) \\ \text{subject to} & \end{array}$$

$$g(\mathbf{x}) = 1 + 10x_2$$
$$h(\mathbf{x}) = 1 - (\tfrac{f_1(\mathbf{x})}{g(\mathbf{x})})^{\alpha} - \tfrac{f_1(\mathbf{x})}{g(\mathbf{x})} * \sin(2\pi \mathrm{q} f_1(\mathbf{x}))$$
$$x_i \in [0, 1], i = 1 \ldots 2$$

The component $h(\mathbf{x})$ is a parametric function with parameters α and q. According to the setting used in [3], in our work the parameters α and q assume the values two and six, respectively. The parameter q defines the number of discrete PF curves in a unit interval of f_1. Therefore, as $f_1(\mathbf{x}) = x_1 \in [0, 1]$, the PF consists of six disconnected curves (Fig. 15 (left)).

To tackle the D99 MOP, the default setting of the NSGA-II algorithm suggested in the original NSGA-II paper [39] is adopted, with SBX and polynomial mutation being the operators for crossover and mutation, respectively. The distribution indexes for both operators are $\eta_c = 20$ and $\eta_m = 20$, respectively. The crossover probability is 0.9, while the mutation probability is $1/n$, where $n = 2$ is the number of decision variables of the problem. The population size is set to 100, and the non-dominated individuals in the tenth generation (obtained by performing 1000 evaluations of the objective vector $\mathbf{z}$) are used to approximate the PF.

The convergence to the Pareto front of both the ALP and NSGA-II algorithms is depicted in Fig. 17, reporting the evolution of the IGDF (left graph) and the IH^{-} (right graph) metrics over the number of function evaluations. The dashed and solid curves plots the average performance (median value over 20 runs) of the ALP and NSGA-II algorithms, respectively, while the error bars plot the interquartile range (IQR) of the data distribution.

The superior performance of ALP w.r.t. NSGA-II observed for ZDT3 is confirmed. For both performance metrics, the quality of the PF approximation returned by ALP rapidly improves after 300 evaluations, converging to a value smaller than 10^{-6} after 600 evaluations. On the other hand, after 1000 objective function evaluations the performance of NSGA-II is still about five orders of magnitude worse than ALP performance. A stable behavior is observed for both algorithms when increasing the number of function evaluations.

Fig. 16 compares the PF model learnt by ALP (left) w.r.t. the approximation generated by NSGA-II (right) when 500 function evaluations are performed (and ALP performance is converging). Only nine over the 100 individuals of the NSGA-II population are non-dominated, and only a couple of them is Pareto-optimal (the IGDF value is equal to 0.10). On the contrary, by interpreting and selecting an informative set of training data, ALP has recovered an accurate PF model (asterisk-marked line, overlapping with the solid line denoting the PF), corresponding to an IGDF value lower than 10^{-5}.

Finally, as for the ZDT3 problem, the combined covariance function including a periodic component was selected more than the 95% of the times by the model selection procedure of the ALP algorithm. This result is consistent with intuition, when observing the boundary of the feasible objective space

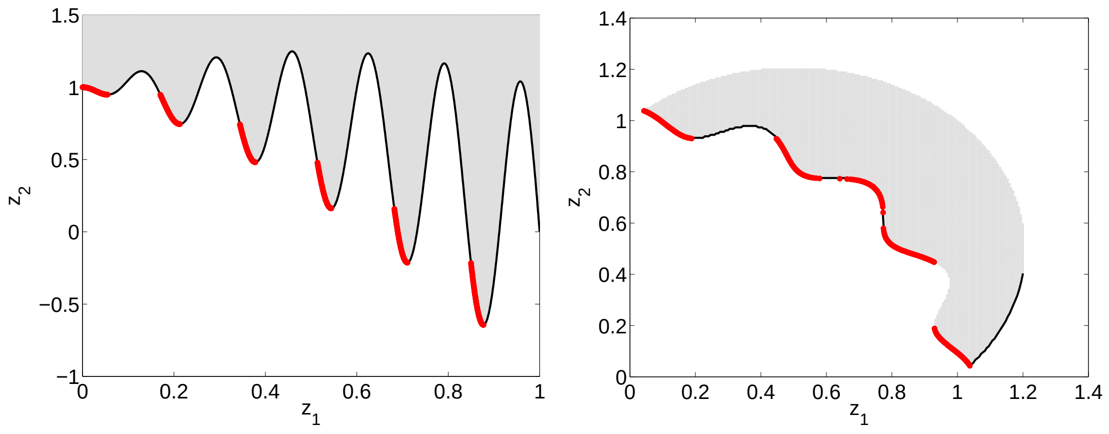

Fig. 15. D99 (left) and TNK (right) problems. The feasible objective space is the gray-shaded area, truncated at the value $z_2 = 1.5$ in the case of the D99 benchmark. The black thin line represents the lower boundary of the space, where the disconnected PF is formed by the bold-marked curves.

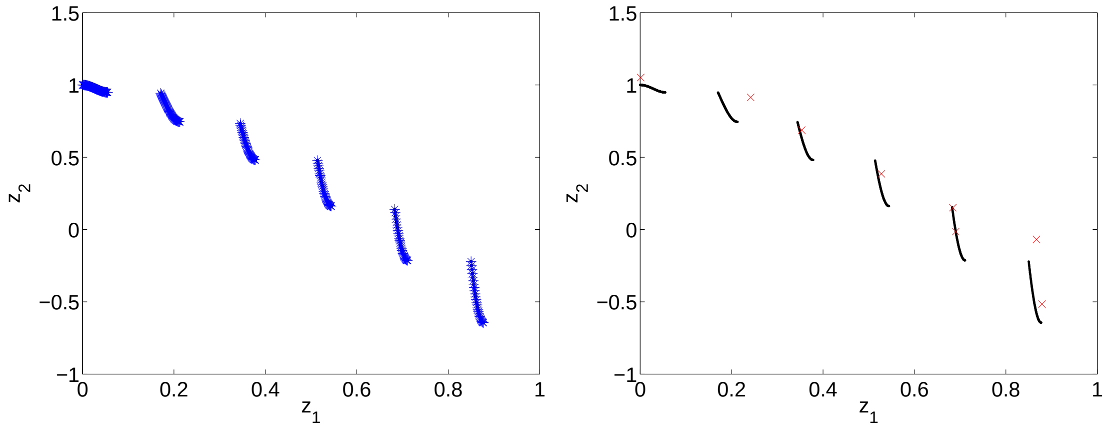

Fig. 16. (Left) Sample model of the D99 PF returned by ALP with 500 function evaluations. (Right) Sample discrete PF approximation obtained when running NSGA-II with the same computational budget. The non-dominated individuals in the current NSGA-II generation are shown by the cross-marked points.

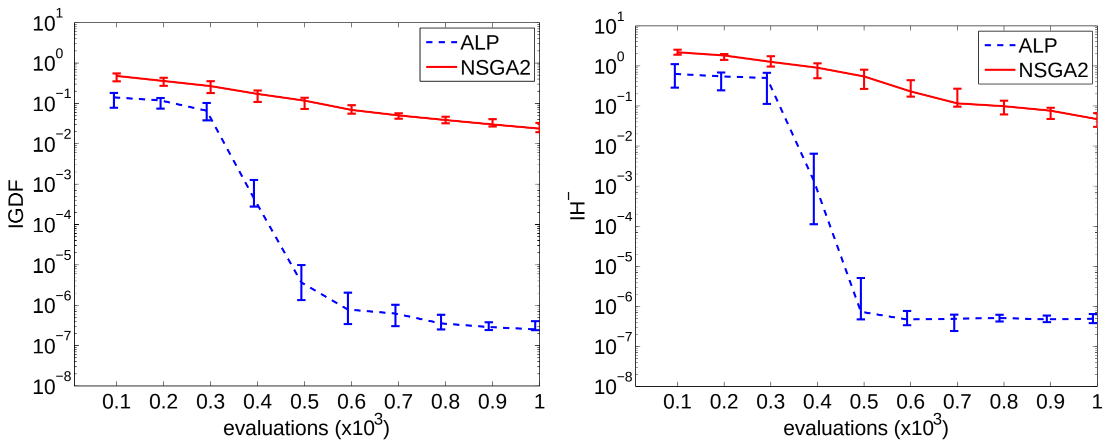

Fig. 17. Performance metrics (median values) evolution over the number of function evaluations observed in the case of D99. Error bars represent the IQR of data distribution. The dashed line plots the ALP results, while the solid line shows the NSGA-II performance. In the left graph, the performance is measured by the IGDF metric, while the right graph plots the IH^{-} metric evolution.

in Fig. 15 (left).

C. Detailed results over the TNK MOP

Tanaka's constrained MOP [36] (TNK) has two objectives and a couple of non-linear constraints. The decision space coincides with the objective space (thus the Pareto set is equal to the Pareto front). The formulation of the problems is as follows:

$$\begin{aligned} \min_{\mathbf{x}} \quad & f_1(\mathbf{x}) = x_1 \\ \min_{\mathbf{x}} \quad & f_2(\mathbf{x}) = x_2 \\ \text{s.t.} \quad & (x_1)^2 + (x_2)^2 \geq 1 + 0.1 * \cos(16 \arctan(\tfrac{x_1}{x_2})) \\ & (x_1 - \tfrac{1}{2})^2 + (x_2 - \tfrac{1}{2})^2 \leq \tfrac{1}{2} \\ & 0 < x_1, x_2 \leq \pi \end{aligned}$$

Each Pareto-optimal solution lies on the first constraint boundary. The non-convex (partial convex and concave shape) and disconnected PF is depicted in Fig. 15 (right).

In our experiments, we employ the specific setting of the NSGA-II parameters for the TNK problem available with the original NSGA-II code. The distribution indexes for the SBX crossover and the polynomial mutation operators are $\eta_c = 5$ and $\eta_m = 5$, respectively. The initial population of size 200 is allowed to evolve for 300 generations, with crossover probability set to 0.9 and mutation probability equal to 0.5.

Fig. 18 reports the performance of ALP and NSGA-II over this problem. The quality of the PF approximation returned by ALP converges to the IGDF value of $4e^{-3}$ within 6000 function evaluations. When considering a bounded computational budget of 600 up to 6000 function evaluations, the results by NSGA-II are from two up to five times worse than ALP performance. However, after 12000 function evaluations, the rank of the algorithms changes, with NSGA-II asymptotically dominating ALP. As a matter of fact, ALP learns a reasonable approximation within much fewer function evaluations than NSGA-II, but it misses a small portion of the PF located in the interval $[0.9, 1]$ on the x-axis (see Fig. 19 (middle left)). This weakness is due to the poor quality of the training examples generated in the interval considered. On the other hand, NSGA-II reaches a uniform spread of the individuals of the population over the whole PF, but at a computational cost sensibly larger that the number of function evaluations performed by ALP at convergence. For example, Fig. 19 (top left) shows a sample PF model recovered by ALP employing 600 function evaluations. The top right graph reports the approximation of the PF generated by NSGA-II with the same computational budget. The middle graphs show the PF generated by ALP (left) and NSGA-II (right) when ALP converges. The NSGA-II has not recovered portions of the second, third and fourth disconnected PF components (numbering from left to right) which ALP has accurately modeled. Finally, the graph at the bottom reports an example of the accurate approximation obtained by NSGA-II when 6×10^4 function evaluations are performed, improving the sub-optimal approximation showed in the middle right graph.

D. Detailed results over the KUR MOP

The Pareto set of Kursawe's MOP [37] (KUR) consists of several disconnected and asymmetric areas in the decision space. The PF is formed by three non-convex disconnected curves (Fig. 20 (left)). The formulation of the problem consists of the following equations:

$$\begin{aligned} \min_{\mathbf{x}} \quad & f_1(\mathbf{x}) = \textstyle\sum_{i=1}^{n-1} (-10\mathrm{e}^{(-0.2\sqrt{x_i^2 + x_{i+1}^2})}) \\ \min_{\mathbf{x}} \quad & f_2(\mathbf{x}) = \textstyle\sum_{i=1}^{n} |x_i|^{0.8} + 5\sin(x_i^3) \\ \text{s.t.} \quad & 5 \leq x_i \leq 5, i = 1, 2, 3 \end{aligned}$$

For this problem, the sequential quadratic programming algorithm [30], [31] used by ALP to generate training examples (Eq. (3)) returns extremely poor-quality approximations of the Pareto optimal solutions. Based on this very noisy training information, an accurate PF model cannot be recovered.

Figure 20 (right) shows the training examples (squares) at the 30-*th* refinement iteration during a single run of ALP. Note the large amount of noise affecting training examples for $z_1 > -17$. In this scenario, ALP cannot provide an accurate approximation (solid line in figure) of the PF (dotted line). However, ALP can easily realize the poor quality of the learned model by measuring its predictive uncertainty, which is exceptionally large also in regions densely populated of training examples. The ALP framework is not bound to a specific optimization algorithm for recovering supervision on training examples (Eq. (3)). Alternative approaches can be explored when the current choice does not provide accurate results, e.g., Gaussian Processes for global optimization [40]. An advantage of using GP would be a substantial reduction in the number of function evaluations for solving Eq. (3), thus further reducing the overall number of function evaluations of ALP (see Discussion). Another possible countermeasure consists of inverting dependent and independent objectives, i.e., considering the objective z_1 as a function of z_2 ($z_1 = g(z_2)$), rather than the other way around, in case this results in a simpler optimization task.

IX. Efficiency of uncertainty sampling

In order to learn a model of the PF, ALP generates training examples in the regression domain. However, generating supervised information is expensive, as it involves the evaluation of the objective functions of the MOP. In order to decrease the computational cost of the learning process, the next query instance selected by ALP is the point $\hat{z}_1$ in the regression domain maximizing the predictive uncertainty of the learnt PF model (uncertainty sampling). The adoption of the uncertainty sampling principle motivates the choice of the GPR method, which can quantify the uncertainty of the predictive model.

The following experiments test the efficiency of the uncertainty sampling (UNC) principle, by comparing it with an alternative query selection strategy. The competing method consists of picking the query points uniformly at random within the regression domain. The random selection principle (RND) is less informed than the UNC method, as it does not exploit the learnt PF model. However, the RND principle

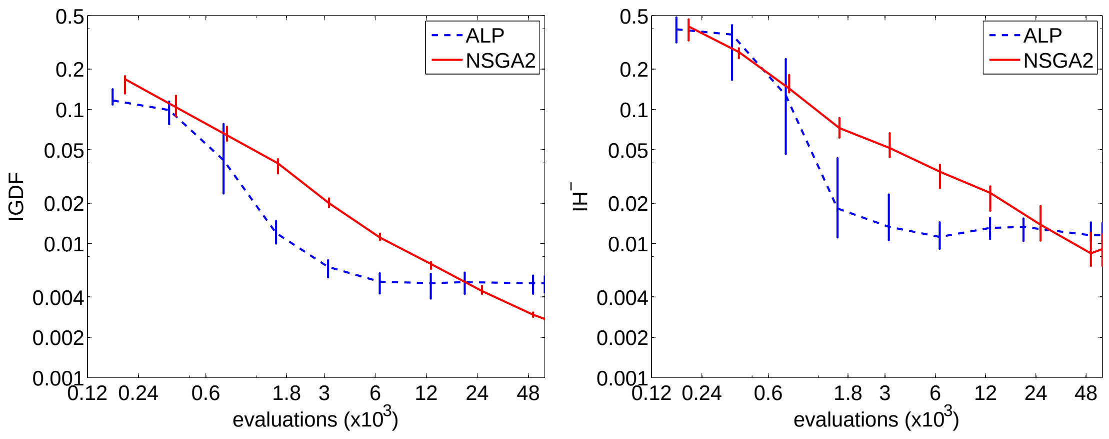

Fig. 18. Evolution of the IGDF and IH$^-$ performance metrics over the number of function evaluations when solving TNK. The data are presented analogously to that in Fig. 17, with the exception of the logarithmic x-scale used to represent both the early ALP convergence within 6000 evaluations and the crossover point of the performance located after 12000 evaluations.

provides a robust approach w.r.t. the general shape of the PF, thus representing a nontrivial competitor in our settings.

Fig. 21 reports the performance of both the UNC and RND sampling principles when increasing the number of query points to solve ZDT3 (top left graph), D99 (top right) and TNK (bottom graph). The dashed and solid lines refer to the UNC and the RND strategies, respectively. For both strategies, one hundred runs of ALP are performed, with different random seeds. Median values of the IGDF metric over the different runs are plotted, together with error bars denoting the 25-th and the 75-th percentiles. Since different stochastic components are involved in the ALP framework, the results of our comparative study are provided with statistical confidence. In particular, multiple pairwise comparisons among the UNC and RND samples are performed, one comparison for each of the different number of query points considered. The Kolmogorov-Smirnov (KS) test for two independent samples is used, with a Bonferroni-corrected significance level α/c, where $\alpha = 0.05$ and $c = 9$ is the number of comparisons performed. The Bonferroni procedure is known to be conservative, especially for highly correlated test statistics. That is, it tends to over-protect from type-I errors at the cost of a lower prevention of type-II errors. However, in our settings, type-I errors (i.e., erroneously stating a different performance among the two sampling strategies) are considered worse than type-II errors (i.e., falsely considering equivalent the performance of the strategies).

For ZDT3 (top left graph), with ten (or less) training examples, the UNC and RND strategies are statistically equivalent under a Bonferroni-corrected KS test. With 15 query points, there is statistical evidence for better results by the RND method (the p-value is 0.003). However, when increasing the number of training examples, the UNC strategy keeps improving faster than the RND strategy and significantly outperforms it if 25 or more query points are generated. With more than 25 examples, the order of magnitude of the observed p-values is lower than $1e^{-14}$, providing strong evidence against the null hypothesis of equivalent performance.

The observed behavior is consistent with intuition. The limited amount of supervised information generated by few training examples, either selected passively (i.e., at random) or actively, does not enable the learning of the PF. However, when a larger number of training instances is considered, the UNC strategy yields better results than using the less informed RND method. Finally, the UNC results are more stable than the RND results, as clearly showed by the error bars.

The top right graph refers to the D99 MOP. The previously observed behavior is confirmed. In particular, there is no statistically significant difference among the RND and UNC performance with ten (or less) training examples. When 15 or 20 training examples are used, the RND performance is statistically better than the UNC strategy results, with observed p-values equal to $1e^{-4}$ and $1e^{-7}$, respectively. However, a superior asymptotic performance of the UNC method over the RND strategy is observed with large statistical confidence: the order of magnitude of the p-values is lower than or equal to $1e^{-20}$ when the number of query points is larger than 30 (the p-value $1e^{-8}$ is observed for 25 examples). This result is consistent with the qualitative insights provided by the graphs: with more than 25 examples the RND and UNC locations are clearly different and the error bars do not overlap. The asymptotic difference among the performance of the RND and UNC strategies is even more pronounced that that observed for ZDT3. The more complicated shape of the D99 PF, which consists of six non-convex disconnected curves w.r.t. the five convex components of the ZDT3 PF, is a likely explanation for this behaviour.

The experiments over TNK confirm the better performance of the UNC strategy, whose curve dominates the curve corresponding to the RND technique. As for D99, with 10 and 15 training examples there is no significant difference between the two strategies. However, with more than 15 training examples UNC significantly outperforms RND: the p-values lower than $1e^{-6}$ provide strong support for the superiority of the UNC sampling strategy.

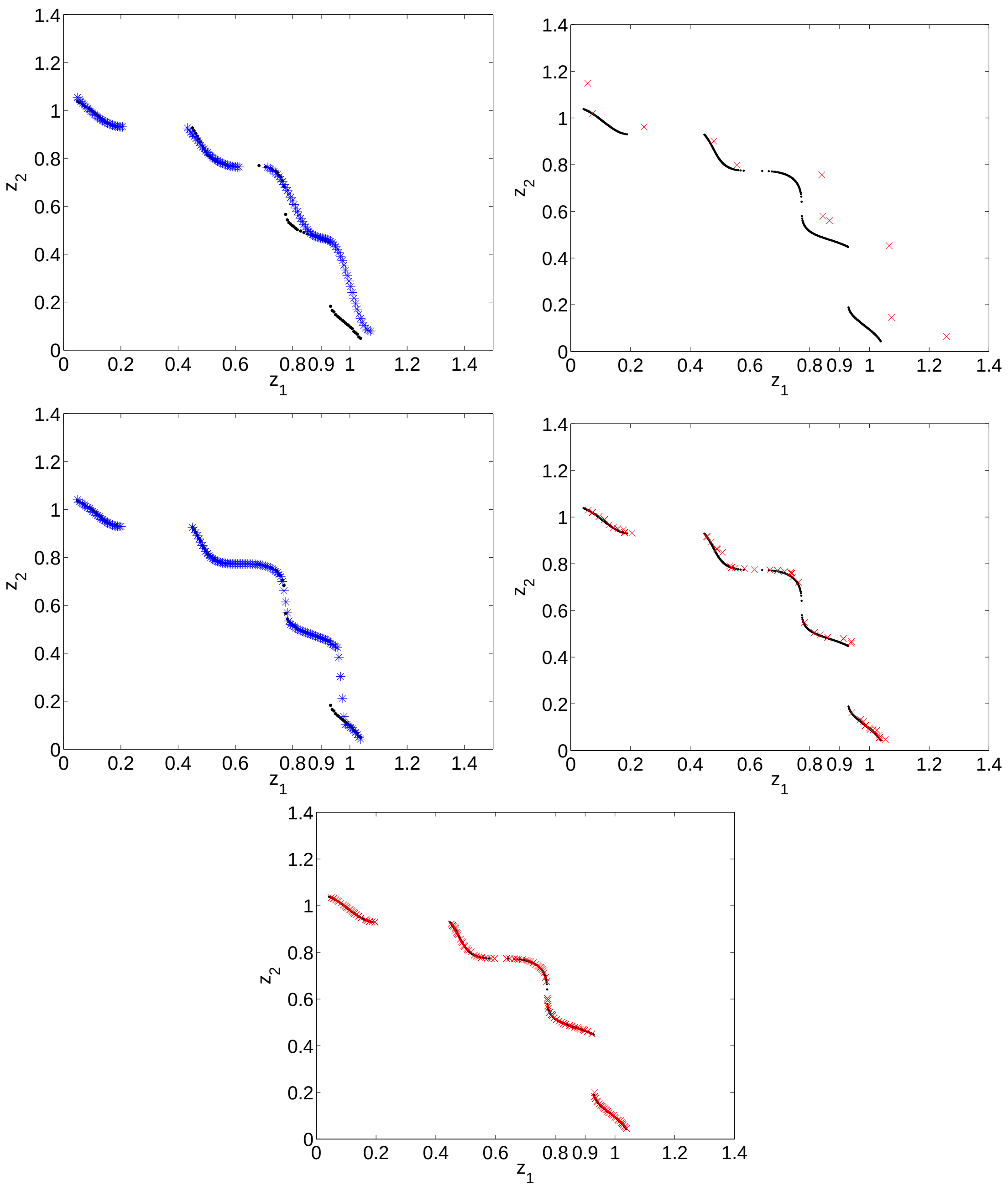

Fig. 19. (Top left) Sample PF model (asterisk-marked curve) returned by ALP after 600 function evaluations (Top right) Sample discrete PF approximation obtained when running NSGA-II with the same computational budget. The non-dominated individuals in the current NSGA-II generation are shown by the cross-marked points. (Middle left) Sample PF model returned by ALP at convergence, missing a portion of PF disconnected component located in the interval $[0.9, 1]$. (Middle right) Sample discrete PF approximation by NSGA-II obtained when ALP converges. (Bottom) Sample PF approximation by NSGA-II requiring 6×10^4 function evaluations.

X. Discussion

This work introduces the Active Leaning of Pareto fronts (ALP) algorithm. While current state of the art algorithms for MOPs are developed within the Evolutionary framework, ALP adopts a different strategy. Pareto-optimal objective vectors are generated by combining the active learning paradigm with the solution of a scalarized optimization problem. The Pareto-optimal objective vectors recovered are used as training examples to learn a model of the PF. The model is iteratively refined until the information gain obtained by the new candidate training examples becomes negligible. The information gain is estimated by the maximum predictive uncertainty of the learnt model in the regression domain.

ALP enables an *analytical* representation for the PF, which simplifies the decision making process. When the analytical PF representation is available, the decision maker (DM) is free to select and compare *any* Pareto-optimal solution, in particular within her preferred region. Once the favourite Pareto-optimal vector is selected, an associated Pareto-optimal solution is generated by solving a single instance of the

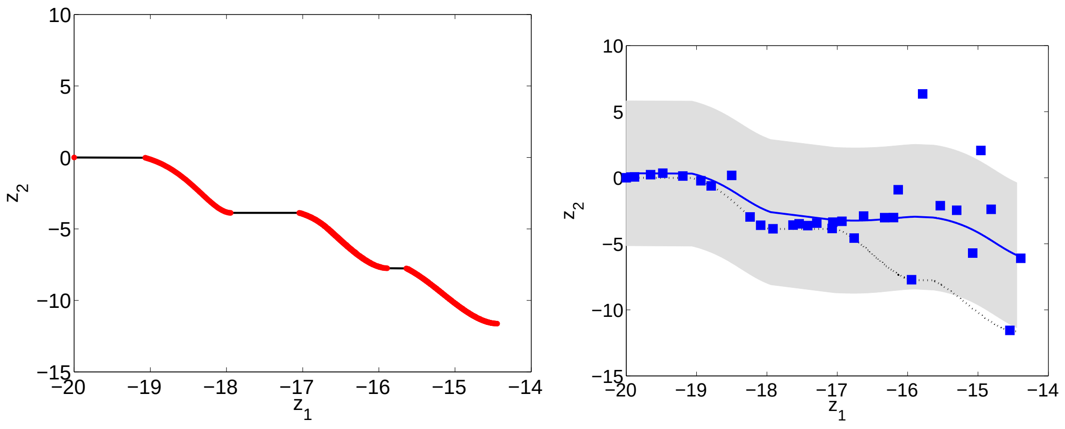

Fig. 20. (Left) The objective space of the Kursawe's (KUR) MOP. The black thin line represents the boundary of the feasible objective space, with the three bold-marked curves and the isolated point $(-20, 0)$ forming the disconnected PF. (Right) The noisy training examples (squares) generated by ALP and its inaccurate prediction (solid line). The large shaded area, denoting the 95% confidence region of the predictive model, clearly shows that ALP cannot interpret the (noisy) training data. The predictive uncertainty is exceptionally high also in regions containing several training examples (e.g., the interval [-18,-17] in the figure).

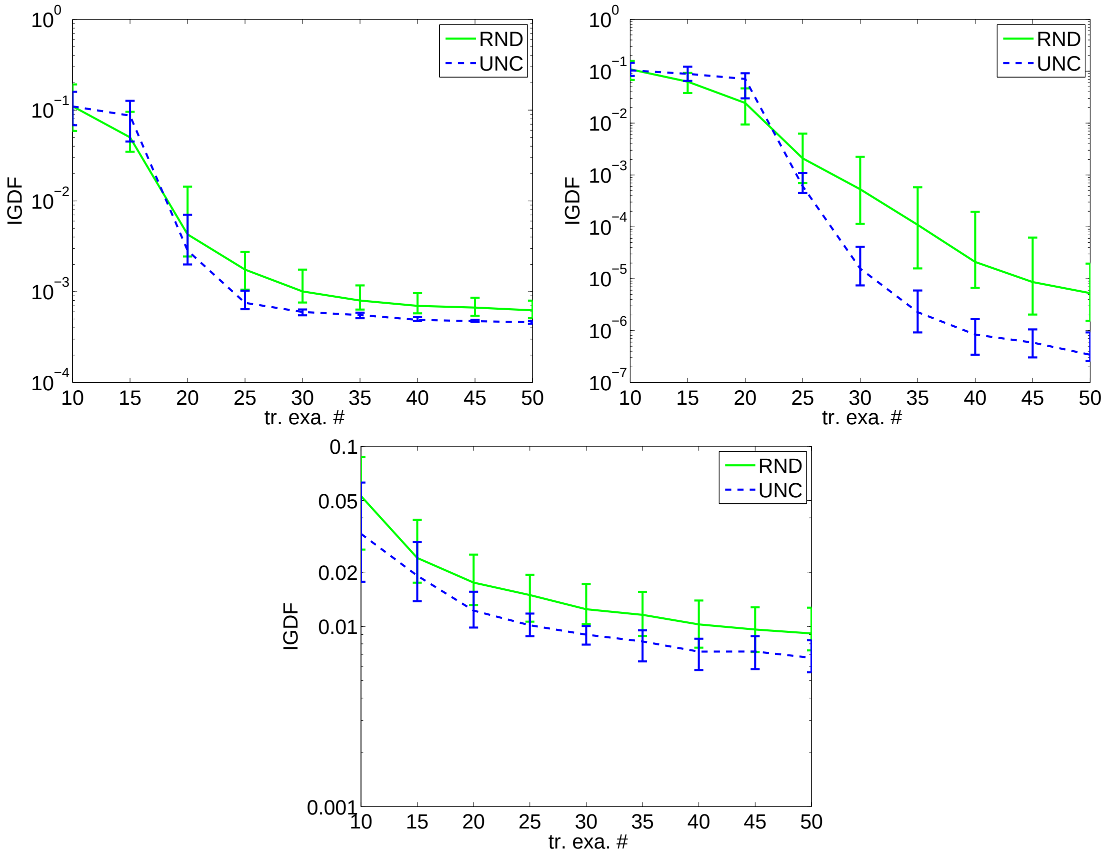

Fig. 21. Comparison between active selection of training examples based on model uncertainty and passive selection based on random sampling. The dashed lines plots the uncertainty sampling results (i.e., the performance of the original ALP algorithm), while the solid lines show the performance of the ALP variant with uncertainty sampling replaced by random sampling. Median IGDF values over 100 runs are plotted for an increasing number of training examples, with error bars representing the IQR of data distribution. The top left and the top right graphs refer to ZDT3 and D99 respectively, while the bottom graph reports the results obtained over TNK.

scalarized optimization problem (3). Furthermore, depending on the optimization algorithm adopted for problem (3), ALP may not require any derivative information. The experimental results on the RM-MEDA benchmark show that ALP on average generates the *analytical* PF representation within a lower number of function evaluations than that required by the state-of-the-art MMEA algorithm to provide a discrete PF approximation. On the ZDT3 and D99 MOPs with disconnected PF, better results are obtained when applying ALP rather than the well-known NSGA-II algorithm (e.g., ALP IGDF value is asymptotically one and five orders of magnitude better than NSGA-II performance on the above mentioned problems). For the TNK MOP, NSGA-II needs more than 12000 function evaluations to outperform ALP. With a lower number of function evaluations, ALP performance is from two up to five times better. ALP cannot learn an accurate model of the KUR PF, due to the high noise affecting the training set. However, ALP can detect the poor quality of the learnt model, by simply observing the predictive variance, in this case exceptionally high, also in regions densely populated by training examples.

We plan to extend this work along different directions. An incremental approach can be adopted to tackle MOPs with more than two objectives. First, the Pareto-optimal solution with minimum distance d from the ideal point is identified. Then m regression tasks are generated, with m being the number of objectives. The i-*th* task considers the objective z_i as the dependent variable, while the regression domain is formed by the points within distance d from the ideal point. By solving each regression task independently, m sets of approximated Pareto-optimal vectors are obtained. They are then merged into a single training set, used to learn the analytical PF representation.

We will also consider the inclusion of prior *monotonicity information* [41] in the Gaussian process regression, as any connected Pareto front can be represented by a monotone function. The investigation of strategies to reduce the computational effort, in terms of MOP objectives evaluation, required to solve the problem (3) is another interesting direction for future research. Approaches tackling *expensive optimization* by the generation of surrogate models are suitable for this purpose. Consider, for example, Gaussian processes designed for global optimization of unknown functions [40]. Our future efforts will also be devoted to model simultaneously the Pareto front in the objective space *and* the Pareto set (PS) in the decision space, as in many real-world tasks an accurate PS approximation may improve the human decision making process [24]. Furthermore, by equipping ALP with a mechanism for learning the DM preferences, the search may focus on the *most relevant* areas of the PF, guided by the user feedback [42], [43]. The incorporation of the DM preferences reduces the waste of computational resources occurring when modeling PF regions which are not of any interest for the user. Finally, possible extensions of ALP to recover the PF under *changing conditions* (dynamic multi-objective optimization [44], [45]) will be explored.

Acknowledgments

This research was partially supported by grant PRIN 2009LNP494 (Statistical Relational Learning: Algorithms and Applications) from Italian Ministry of University and Research.

References

[1] K. Miettinen, *Nonlinear Multiobjective Optimization*, ser. International Series in Operations Research and Management Science. Kluwer Academic Publishers, Dordrecht, 1999, vol. 12.

[2] K. Miettinen, F. Ruiz, and A. Wierzbicki, "Introduction to Multiobjective Optimization: Interactive Approaches," in *Multiobjective Optimization: Interactive and Evolutionary Approaches*. Springer Verlag Berlin, Heidelberg, 2008, pp. 27–57.

[3] C. C. Coello, G. Lamont, and D. van Veldhuizen, *Evolutionary Algorithms for Solving Multi-Objective Problems*, 2nd ed., ser. Genetic and Evolutionary Computation. Berlin, Heidelberg: Springer, 2007.

[4] A. Zhou, B.-Y. Qu, H. Li, S.-Z. Zhao, P. N. Suganthan, and Q. Zhang, "Multiobjective evolutionary algorithms: A survey of the state of the art," *Swarm and Evolutionary Computation*, vol. 1, no. 1, pp. 32 – 49, 2011.

[5] J. Venna, J. Peltonen, K. Nybo, H. Aidos, and S. Kaski, "Information Retrieval Perspective to Nonlinear Dimensionality Reduction for Data Visualization," *Journal of Machine Learning Research*, vol. 11, pp. 451–490, 2010.

[6] C. Rasmussen and C. Williams, *Gaussian processes for machine learning*, ser. Adaptive computation and machine learning. MIT Press, 2006.

[7] D. Cohn, L. Atlas, and R. Ladner, "Improving generalization with active learning," *Machine Learning*, vol. 15, no. 2, pp. 201–221, 1994.

[8] D. J. Lizotte, M. H. Bowling, and S. A. Murphy, "Efficient Reinforcement Learning with Multiple Reward Functions for Randomized Controlled Trial Analysis," in *Proceedings of the 27th International Conference on Machine Learning (ICML-10), June 21-24, 2010, Haifa, Israel*, J. Frnkranz and T. Joachims, Eds. Omnipress, 2010, pp. 695–702.

[9] P. Vamplew, R. Dazeley, A. Berry, R. Issabekov, and E. Dekker, "Empirical evaluation methods for multiobjective reinforcement learning algorithms," *Machine Learning*, vol. 84, no. 1-2, pp. 51–80, Jul. 2011.

[10] A. Lovison, "Singular Continuation: Generating Piecewise Linear Approximations to Pareto Sets via Global Analysis," *SIAM Journal on Optimization*, vol. 21, no. 2, pp. 463–490, 2011.

[11] Q. Zhang, A. Zhou, and Y. Jin, "RM-MEDA: A Regularity Model-Based Multiobjective Estimation of Distribution Algorithm," *IEEE Transactions on Evolutionary Computation*, vol. 12, no. 1, pp. 41–63, 2008.

[12] E. Snelson, "Flexible and efficient Gaussian process models for machine learning," Ph.D. dissertation, Gatsby Computational Neuroscience Unit, University College London, 2007.

[13] D. J. C. MacKay, *Information theory, inference, and learning algorithms*. Cambridge University Press, 2003.

[14] M. Gibbs, "Bayesian Gaussian Processes for Classification and Regression," Ph.D. dissertation, University of Cambridge, Cambridge, U.K., 1997.

[15] D. J. C. MacKay, "Comparison of Approximate Methods for Handling Hyperparameters," *Neural Computation*, vol. 11, no. 5, pp. 1035–1068, 1999.

[16] B. Settles, "Active Learning Literature Survey," University of Wisconsin-Madison, Tech. Rep. 1648, 2009.

[17] Y. Jin, "Surrogate-assisted evolutionary computation: Recent advances and future challenges." *Swarm and Evolutionary Computation*, vol. 1, no. 2, pp. 61–70, 2011.

[18] M. Pilát and R. Neruda, "Aggregate meta-models for evolutionary multiobjective and many-objective optimization," *Neurocomputing*, 2013, in press. Digital version available on-line, as of Dec 21 2012.

[19] L. Marti, J. Garcia, A. Berlanga, C. A. Coello Coello, and J. M. Molina, "On Current Model-Building Methods for Multi-Objective Estimation of Distribution Algorithms: Shortcomings and Directions for Improvement," Grupo de Inteligencia Artificial Aplicada, Universidad Carlos III de Madrid, Colmenarejo, Spain, Tech. Rep. GIAA2010E001, 2010.

[20] P. A. Bosman and D. Thierens, "Multi-objective Optimization with the Naive $\mathbb{MIDE}$A," in *Towards a New Evolutionary Computation*, ser. Studies in Fuzziness and Soft Computing, J. Lozano, P. Larraaga, I. Inza, and E. Bengoetxea, Eds. Springer Berlin Heidelberg, 2006, vol. 192, pp. 123–157.

[21] M. Laumanns and J. Ocenasek, "Bayesian Optimization Algorithms for Multi-objective Optimization," in *Seventh International Confenrence on Parallel Problem Solving From Nature (PPSN VII)*, ser. Lecture Notes in Computer Science. Springer, 2002, pp. 298–307.

[22] M. Costa and E. Minisci, "MOPED: A Multi-objective Parzen-Based Estimation of Distribution Algorithm for Continuous Problems," in *Evolutionary Multi-Criterion Optimization*, ser. Lecture Notes in Computer Science, C. Fonseca, P. Fleming, E. Zitzler, L. Thiele, and K. Deb, Eds. Springer Berlin Heidelberg, 2003, vol. 2632, pp. 282–294.

[23] H. Karshenas, R. Santana, C. Bielza, and P. Larraaga, "Multi-objective Optimization with Joint Probabilistic Modeling of Objectives and Variables," in *Evolutionary Multi-Criterion Optimization*, ser. Lecture Notes in Computer Science, R. Takahashi, K. Deb, E. Wanner, and S. Greco, Eds. Springer Berlin Heidelberg, 2011, vol. 6576, pp. 298–312.

[24] A. Zhou, Q. Zhang, and Y. Jin, "Approximating the Set of Pareto-Optimal Solutions in Both the Decision and Objective Spaces by an Estimation of Distribution Algorithm," *IEEE Transactions on Evolutionary Computation*, vol. 13, no. 5, pp. 1167–1189, 2009.

[25] Q. Zhang, W. Liu, E. P. K. Tsang, and B. Virginas, "Expensive Multiobjective Optimization by MOEA/D With Gaussian Process Model," *IEEE Transactions on Evolutionary Computation*, vol. 14, no. 3, pp. 456–474, 2010.

[26] Q. Zhang and H. Li, "MOEA/D: A Multiobjective Evolutionary Algorithm Based on Decomposition," *IEEE Transactions on Evolutionary Computation*, vol. 11, no. 6, pp. 712–731, 2007.

[27] K. Deb, "Multi-objective evolutionary optimization: Past, present and future," in *Evolutionary Design and Manufacture*, I. C. Parmee, Ed. Springer, London, 2000, pp. 225–236.

[28] E. Zitzler, K. Deb, and L. Thiele, "Comparison of Multiobjective Evolutionary Algorithms: Empirical Results," *IEEE Transactions on Evolutionary Computation*, vol. 8, no. 2, pp. 173–195, 2000.

[29] K. Deb, A. Sinha, and S. Kukkonen, "Multi-objective test problems, linkages, and evolutionary methodologies," in *Proceedings of the 8th conference on Genetic and evolutionary computation (GECCO 06), Seattle, Washington, USA*, ser. GECCO '06. New York, NY, USA: ACM, 2006, pp. 1141–1148.

[30] R. Fletcher, *Practical Methods of Optimization, (2nd ed.)*. New York, NY, USA: John Wiley and Sons, 1987.

[31] M. Powell, "Variable Metric Methods for Constrained Optimization," in *Mathematical Programming: The State of the Art*, A. Bachem, M. Grotschel, and B. Korte, Eds. Springer Verlag, 1983, pp. 288–311.

[32] R. A. Waltz, J. L. Morales, J. Nocedal, and D. Orban, "An interior algorithm for nonlinear optimization that combines line search and trust region steps," *Mathematical Programming*, vol. 107, no. 3, pp. 391–408, 2006.

[33] E. Zitzler and L. Thiele, "Multiobjective Optimization Using Evolutionary Algorithms - A Comparative Case Study," in *5th International Conference on Parallel Problem Solving from Nature (PPSN V)*, ser. Lecture Notes in Computer Science, vol. 1498. Springer, 1998, pp. 292–301.

[34] K. Deb, K. Miettinen, and S. Chaudhuri, "Toward an Estimation of Nadir Objective Vector Using a Hybrid of Evolutionary and Local Search Approaches," *IEEE Transactions on Evolutionary Computation*, vol. 14, no. 6, pp. 821–841, 2010.

[35] K. Deb, "Multi-objective Genetic Algorithms: Problem Difficulties and Construction of Test Problems." *Evolutionary Computation*, vol. 7, no. 3, pp. 205–230, 1999.

[36] M. Tanaka, "GA-based decision support system for multi-criteria optimization," in *IEEE International Conference on Systems, Man and Cybernetics*, vol. 2. IEEE, 1995, pp. 1556–61.

[37] F. Kursawe, "A Variant of Evolution Strategies for Vector Optimization," in *Proceedings of the First Workshop on Parallel Problem Solving from Nature*, ser. Lecture Notes in Computer Science (LNCS), vol. 496. Springer, 1991, pp. 193–197.

[38] K. Deb, S. Agrawal, A. Pratap, and T. Meyarivan, "A fast and elitist multiobjective genetic algorithm: NSGA-II," *IEEE Transactions on Evolutionary Computation*, vol. 6, no. 2, pp. 182–197, 2002.

[39] K. Deb, A. Pratap, S. Agarwal, and T. Meyarivan, "A fast and elitist multi-objective genetic algorithm: NSGA-II," *IEEE Transactions on Evolutionary Computation*, vol. 6, pp. 182–197, 2002.

[40] M. A. Osborne, R. Garnett, and S. J. Roberts, "Gaussian Processes for Global Optimization," in *3rd International Conference on Learning and Intelligent Optimization (LION 3)*, ser. Lecture Notes in Computer Science. Springer, 2009, pp. 1–15.

[41] J. Riihimäki and A. Vehtari, "Gaussian processes with monotonicity information," *Journal of Machine Learning Research - Proceedings Track*, vol. 9, pp. 645–652, 2010.

[42] R. Battiti and A. Passerini, "Brain-Computer Evolutionary Multi-Objective Optimization (BC-EMO): a genetic algorithm adapting to the decision maker," *IEEE Transactions on Evolutionary Computation*, vol. 14, no. 5, pp. 671–687, 2010.

[43] P. Campigotto, A. Passerini, and R. Battiti, "Handling concept drift in preference learning for interactive decision making," in *Online proc. of the International Workshop on Handling Concept Drift in Adaptive Information Systems (HaCDAIS 2010)*, 2010.

[44] M. Farina, K. Deb, and P. Amato, "Dynamic multiobjective optimization problems: test cases, approximations, and applications," *IEEE Transactions on Evolutionary Computation*, vol. 8, no. 5, pp. 425–442, 2004.

[45] E. Tantar, A.-A. Tantar, and P. Bouvry, "On dynamic multi-objective optimization, classification and performance measures," in *Proceedings of the IEEE Congress on Evolutionary Computation (CEC 2011), June 5-8, 2011, New Orleans, LA, USA*. IEEE, 2011, pp. 2759–2766.