Brain-Computer Evolutionary Multi-Objective Optimization (BC-EMO): a genetic algorithm adapting to the decision maker

Roberto Battiti, *Fellow, IEEE,* and Andrea Passerini

***Abstract*—The centrality of the decision maker (DM) is widely recognized in the Multiple Criteria Decision Making community. This translates into emphasis on seamless human-computer interaction, and adaptation of the solution technique to the knowledge which is progressively acquired from the DM.**

This paper adopts the methodology of Reactive Search Optimization (RSO) for evolutionary interactive multi-objective optimization. RSO follows to the paradigm of "learning while optimizing", through the use of online machine learning techniques as an integral part of a self-tuning optimization scheme.

User judgments of couples of solutions are used to build robust incremental models of the user utility function, with the objective to reduce the cognitive burden required from the DM to identify a *satisficing* solution. The technique of support vector ranking is used together with a k-fold cross-validation procedure to select the best kernel for the problem at hand, during the utility function training procedure. Experimental results are presented for a series of benchmark problems.

I. Introduction

MANY evolutionary algorithms have been developed in the last years, starting at least from the eighties [1], to solve multiobjective optimization problem (MOOPs). A MOOP can be stated as:

$$\text{maximize} \quad \boldsymbol{f}(\boldsymbol{x}) = \{f_1(\boldsymbol{x}), \ldots, f_m(\boldsymbol{x})\} \tag{1}$$

$$\text{subject to} \quad \boldsymbol{x} \in \Omega \tag{2}$$

where $\boldsymbol{x} \in \mathbb{R}^n$ 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; $\boldsymbol{f} : \Omega \rightarrow \mathbb{R}^m$ is made of m objective functions which need to be jointly maximized. Objective vectors are images of decision vectors and can be written as $\boldsymbol{z} = \boldsymbol{f}(\boldsymbol{x}) = \{f_1(\boldsymbol{x}), \ldots, f_m(\boldsymbol{x})\}$. 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 $z_k \geq z'_k$ for all k and there exists at least one h such that $z_h > z'_h$. A point $\hat{\mathbf{x}}$ is Pareto optimal if there is no other $\mathbf{x} \in \Omega$ such that $\boldsymbol{f}(\mathbf{x})$ dominates $\boldsymbol{f}(\hat{\mathbf{x}})$. The set of Pareto optimal points is called *Pareto set* (PS). The corresponding set of Pareto optimal objective vectors is called *Pareto front* (PF).

R. Battiti and A. Passerini are with Dipartimento di Ingegneria e Scienza dell'Informazione, Università di Trento, Via Sommarive, 14 - 38050 Povo (Trento) - Italy (e-mail: {battiti,passerini}@disi.unitn.it).

Evolutionary algorithms (EAs) work with a population of tentative solutions and they are therefore ideally suited to search for a set of Pareto-optimal solutions to be presented to the decision maker (DM). In this paradigm, evolutionary multiobjective optimization algorithms (EMOAs) aim at *building a representative set of points near the Pareto front*.

Some of the most successful EMOAs [2], [3], [4] rely on Pareto dominance classification as a fitness measure to guide the selection of the new population. To adequately cover the PF, two important criteria are the proximity of the points to the PF and the diversity and spread of solutions, aiming at a uniform distribution on the PF. The work in [5] highlights a tradeoff between proximity and diversity that cannot be solved *a priori* without considering the specific demands of the decision maker. Furthermore, the definition of "uniform distribution" hides the fact that uniformity depends on the metric, on the scaling of objective values, in other words on the user preferences which cannot be fixed *a priori* [6].

The work [7] indicates that resorting to Pareto dominance classification to assign fitness becomes ineffective for an increasing number of objectives and proposes a refined preference ordering based on the notion of order of efficiency [8]. The reason is that the proportion of Pareto-optimal elements in a set grows very rapidly with the dimension m, so that most solutions become in practice indistinguishable unless other criteria are added.

When the interaction with a human user is considered, the assumption that a typical DM likes the idea of being presented with hundreds or thousands of representative solutions on a multi-dimensional Pareto front is very far from reality. Although creating this representative set is appealing from a mathematical and research-oriented point of view, the reality of applied decision making must consider the following crucial issues:

- Bounded rationality and information bottleneck. The typical DM (being a typical human person) cannot deal with more than a very limited number of information items at a time. After Simon's seminal papers in the fifties, aspects of bounded rationality permeate theories of choice, see for example the review in [9]. Human beings develop *satisficing* decision procedures, which are sensible given their constraints regarding human memory and information processing capabilities.
- Learning on the job. Most DMs cannot or do not like to explicitly formulate their objectives at the beginning. This is already recognized in the MOO formulation, where

a combination of the individual objectives into a single preference function is not executed: knowledge elicitation stops at the building blocks given by the individual z_k's. Interactive MOO can be used to iteratively build an approximation of the DM's utility function after specifying a suitable form, for example a linearly weighted combination of the objectives. In interactive MOO, learning by the DM happens in coordination with a computer-supported solution processes. According to [10], through interactive MOO the DM is building a conviction of what is possible and confronting this knowledge with her preferences, that also evolve.

- Simple questions, qualitative judgments. The number of questions that have to be asked to the DM before a satisfactory solution is identified is a crucial performance indicator of interactive methods. This demands for selecting appropriate questions, for extracting as much information as possible from the answers, for building approximated models which may reduce the need to bother the DM. The complexity of the questions is also an issue. All DMs can be assumed to produce qualitative judgments, like "I prefer solution A to solution B." Asking for quantitative evaluations of the kind "By how much do you prefer A over B?" is in some cases inappropriate, artificial, or even impossible. Asking for *explanations* of choices is even more difficult: most DM are typically more confident in judging and comparing than in explaining.
- Uncertainty and inconsistency. The assumption that a fixed, deterministic and error-free preference structure of the DM is available is often not realistic. Imprecisions, contradictions, changes of judgment over time are the characteristics of most human decision processes.

The above issues demand a shift of paradigm, from building a set of solutions which is representative of the true PF, to the interactive construction of a sequence of solutions where the DM is a crucial learning component in the optimization loop, a component characterized by limited rationality and scarce question-answering capabilities.

The objective of our long-term investigation is the systematic use of machine learning techniques for online learning schemes in optimization processes. In particular, the objective of Reactive Search Optimization (RSO) [11] is to design problem-solving schemes with an internal online feedback loop for the self-tuning of critical parameters. Internal parameters and algorithmic choices are adapted to the particular instance being solved. Our approach is somehow related to the Brain Computer Interfaces (BCI), proposed for example in [12], based on direct connection with brain signals like EEG. However, we assume here that the "whole brain" is in the loop with both sub-symbolic and symbolic reasoning capabilities.

In the context considered in this work, the objective of the learning process is the approximated construction of a utility function U to be optimized by the DM, who is also the source of learning signals. As in the traditional MOO context, the function to be optimized is not completely unknown like in a black-box context (see for example the response surface methodology and the related kriging and design of experiments approaches [13]), but is to be modeled after starting from the building blocks z_k characterizing positive features of a generic solution. Preference models are built from the DM input by using the support vector ranking method. The functional form of the preference function is not fixed *a priori*, like it is in the weighted sum or Tchebycheff approaches (see for example [14] for a clear explanation of scalarization methods) but it is itself learnt during the process in a reactive fashion.

The rest of the work is organized as follows. The problem of modeling user preferences for interactive MOO and our proposed machine learning approach are discussed in Section II. The support vector ranking method for learning user preferences is presented in Section III. The overall algorithm is detailed in Section IV and the related works are discussed in Section V. An extensive experimental evaluation is reported in Section VI. Finally, possible directions for future research are highlighted in Section VII and conclusions are drawn in Section VIII.

II. Brain-Computer Optimization: Learning user preferences in EMO approaches

Solving a MOO problem typically means providing a human decision maker with the solution she believes optimal according to a certain utility criterion allowing her to choose among competing Pareto-optimal alternatives. This utility criterion can of course be partially inconsistent, difficult to formalize and subject to revision according to the solutions provided by the optimization algorithm. Approaches to MOO can be roughly divided into the two broad categories of non-interactive and interactive ones [15]. The former can be further divided into *a priori* approaches, where the decision maker is required to formulate her preferences in advance, and *a posteriori* approaches, where the algorithm recovers a representative subset of the Pareto-optimal set from which the user selects the preferred solution. *A priori* methods have the drawback of requiring the user to pre-specify her preferences, for instance as a set of weights on different objectives, which is typically rather hard for a human decision maker. *A posteriori* methods, on the other hand, imply a laborious selection among a large set of candidate solutions. Interactive approaches try to overcome some of these difficulties by keeping the user in the loop of the optimization process and progressively focusing on the most relevant areas of the Pareto front, guided by the user feedback. The focus on fusing the capabilities of evolutionary computation (EC) with human evaluations reaches an extreme point with the interactive EC proposed in [16], where the fitness function is *replaced* by a human user. Our investigation follows an intermediate point, where knowledge of the objectives z_k is assumed *a priori*, and the DM is queried in order to build an explicit and robust nonlinear model of her preferences, to be used as an integral component of the problem-solving process.

Formalizing user preferences into a mathematical model is a non-trivial task. A model should be able to capture the qualitative notion of preference and represent it as a quantitative function, while retaining Pareto-optimality properties. The simplest possible model is a linear function, in which

(positive) weights encode the relative importance of different objectives:

$$U(\mathbf{z}) = \sum_{k=1}^{m} w_k z_k$$

However, its appropriateness is rather questionable [17]. Indeed, whenever objective functions correlate with each other, the most intuitive approach of giving the highest weight to the most important criterion can lead to completely unsatisfactory solutions [18], [19]. Furthermore, assuming that satisfaction increases linearly when objective function values increase is not straightforward. Relying on the concept of decreasing limiting performance from mathematical economics, Podinovskii [20] argues that improvements on poorly satisfied objectives should be considered more relevant than equally sized improvements on better satisfied ones. Most approaches generalize the linear utility function to a weighted-L_p metric of the following form:

$$U(\mathbf{z}) = -\left(\sum_{k=1}^{m}(w_k|z_k^* - z_k|)^p\right)^{1/p} \tag{3}$$

where $\mathbf{z}^*$ is a reference ideal objective vector obtained by separately maximizing each objective function subject to the feasible region, i.e., $z_k^* = \max_{\boldsymbol{x}\in\Omega} f_k(\boldsymbol{x})$. Let's note that the weights in equation (3) are raised to the power p as well, so that it will transform into equation (4) below. A popular choice in this setting is that of the Tchebycheff (or L_∞) metric, leading to the following augmented weighted Tchebycheff program (AWTP):

$$\begin{aligned} \min_{\mathbf{x},\alpha} \quad & \alpha + \rho\sum_{k=1}^{m}(z_k^{**} - z_k) \\ \text{subject to:} \quad & \\ & w_k(z_k^{**} - z_k) \le \alpha \\ & \mathbf{w} \in \mathbb{R}^m,\ w_k \ge 0,\ \sum_{k=1}^{m} w_k = 1 \\ & f_k(\mathbf{x}) = z_k,\ \mathbf{x} \in \Omega \\ & k = 1, \ldots, m \end{aligned} \tag{4}$$

where $\mathbf{z}^{**}$ is the utopian objective vector obtained adding a small positive scalar to the ideal vector $\mathbf{z}^*$. Let's comment: if ρ is set to zero, minimizing α amounts to minimizing the maximum weighted distance $w_k(z_k^{**} - z_k)$ between each individual objective and the utopian target. The augmentation by the term $\rho\sum_{k=1}^{m}(z_k^{**} - z_k)$, with ρ being a small positive scalar, generates *properly* Pareto-optimal solutions, a more robust subset of Pareto points (a finite improvement in one objective is possible only at the expense of a *reasonable* worsening in other objectives).

The interactive weighted Tchebycheff procedure (IWTP) [21] consists of solving multiple AWTP, one for each choice of weight vectors $\mathbf{w}$, in order to present the DM with a small set of well-spaced, representative sample solutions. The DM is asked to select the most preferred one among the proposed solutions, a new set of weights is produced which is consistent with the DM choice. Then a new set of solutions is generated and the procedure is repeated until the DM is satisfied with the results. A more recent paper using a Tchebycheff utility function [22] focuses on minimizing the number of questions asked to the DM. Pairwise comparisons of solutions are used to generate constraints on the Tchebycheff weights and to identify multiple disjoint regions in weight space. Representative weights from the different regions are then generated to search for new *challenger* solutions to present to the DM.

Given the critique for linear weighting schemes, related also to the nonlinear preference of "compromise" solutions which is characteristic of many human decision activities, we explicitly consider nonlinear dependencies in this work. For instance, a generic polynomial utility function such as:

$$\begin{aligned} U(\mathbf{z}) \quad = \quad & \sum_{i=1}^{m} w_i z_i + \sum_{i=1}^{m}\sum_{j\ge i} w_{ij} z_i z_j \\ & + \sum_{i=1}^{m}\sum_{j\ge i}\sum_{h\ge j} w_{ijh} z_i z_j z_h + \cdots \\ & + \sum_{i=1}^{m}\cdots\sum_{k\ge m-1} w_{i\ldots km} z_i \cdots z_k z_m \end{aligned} \tag{5}$$

would account for strong non-linear relations between objectives. However, we do not specify *a priori* a certain form for the utility function, but rather employ a Reactive Search Optimization scheme to determine the appropriate model to be used *while* the algorithm runs on a specific instance. The selection of a linear versus nonlinear model as well as the type of nonlinear model is decided in an automated way, depending on the user interaction and on the preference for simple models explaining the data which is the bread and butter of modern machine learning techniques. Our solution aims at:

1) being able to learn an *arbitrary* utility function from examples of preference information interactively provided by the DM

2) requiring DM intervention only through *holistic judgments* (comparisons of complete solutions instead of specification of detailed parameters like weights of trade-offs), by ranking sets of competing instances or specifying pairwise preferences between candidate solutions

3) naturally accounting for incomplete, imprecise and contradictory feedback from the DM

4) directly employing the learned utility function in order to guide the search for refined solutions

We rely on an adaptation of the well-known support vector machines [23] classification algorithm to preference learning in order to learn the utility function from user preference information. An EMO algorithm alternates a search phase, guided by a fitness measure based on the learned utility function, and a refinement phase where the DM is queried for feedback on candidate solutions and the utility function is updated according to such feedback.

III. Learning to rank

Given a sequence of competing instances $(\mathbf{z}_1, \ldots, \mathbf{z}_\ell) \in \mathcal{Z}^\ell$ and an order relation $\succ$ such that $\mathbf{z}_i \succ \mathbf{z}_j$ if $\mathbf{z}_i$ should be preferred to $\mathbf{z}_j$, a *ranking* $(y_1, \ldots, y_\ell)$ of the ℓ instances can be specified as a permutation of the first ℓ natural numbers such that $y_i < y_j$ if $\mathbf{z}_i \succ \mathbf{z}_j$. Learning to rank consists of learning a ranking function f from a dataset $\mathcal{D} = \{(\mathbf{z}_1^{(i)}, \ldots, \mathbf{z}_{\ell_i}^{(i)}), (y_1^{(i)}, \ldots, y_{\ell_i}^{(i)})\}_{i=1}^s$, of s sequences with their desired rankings. Different approaches have been proposed in the literature to deal with ranking tasks, see [24] for a review. A common approach consists of learning a *utility* function $U : \mathcal{Z} \to \mathbb{R}$ measuring the importance of the instance, with the aim that $U(\mathbf{z}_i) > U(\mathbf{z}_j) \iff \mathbf{z}_i \succ \mathbf{z}_j$. In the following we focus on an effective solution for learning a utility function based on suitable adaptations of the support vector machine algorithm [25], [26], [27]. The solution naturally accounts for situations in which only partial information on pairwise ordering is available for training. We start with a brief discussion on support vector machines for classification [23] and extend the formulation to handle ranking tasks.

A. Support vector classification

Let $\mathcal{D} = (\mathbf{z}_i, y_i)_{i=1}^s$ be a training set of s binary labelled examples, $y_i = 1$ or $y_i = -1$. Standard Support Vector Machines [23] for classification learn a decision function $h(\mathbf{z}) = \text{sign}\,(\langle \mathbf{w}, \mathbf{z} \rangle + b)$ trading-off fitting of the training data with *large margin* separation of classes, by solving the following optimization problem:

$$\begin{aligned} \min_{\mathbf{w} \in \mathcal{Z}, b \in \mathbb{R}, \boldsymbol{\xi} \in \mathbb{R}^s} \quad & \frac{1}{2}||\mathbf{w}||^2 + C \sum_{i=1}^{s} \xi_i \\ \text{subject to:} \quad & \\ & y_i(\langle \mathbf{w}, \mathbf{z}_i \rangle + \ b\ ) \geq 1 - \xi_i \\ & \xi_i \geq 0 \\ & i = 1, \ldots, s \end{aligned} \tag{6}$$

The first term in the minimization encourages a large margin separation, with margin given by $2/||w||$, while the second term contains penalties ξ_i for examples i not separated with the required margin (see Figure 1 for an example). The tradeoff factor C has to be adapted to the data, for example a noisy measurement process for $\mathbf{z}$ implies that margin errors ξ_i be given a smaller weight (a smaller C value).

Whenever non-linear separation surfaces are needed in order to obtain a satisfactory separation, examples are projected onto a suitable higher dimensional feature space via a mapping function Φ (see Figure 2) and the same optimization problem is solved, simply replacing $\mathbf{z}_i$ with $\Phi(\mathbf{z}_i)$. In the dual formulation of the optimization problem, projected examples always occur within dot products $\langle \Phi(\mathbf{z}_i), \Phi(\mathbf{z}_j) \rangle$. These can be replaced by an equivalent *kernel function* $k(\mathbf{z}_i, \mathbf{z}_j) = \langle \Phi(\mathbf{z}_i), \Phi(\mathbf{z}_j) \rangle$ on input examples which efficiently computes the dot product without explicitly doing the projection, which can even be infinite dimensional. The resulting decision function can be represented as a linear combination of kernel functions on support vectors: $h(\mathbf{z}) = \text{sign} \sum_{\mathbf{z}_i \in SV} \alpha_i y_i K(\mathbf{z}_i, \mathbf{z})$. Kernel

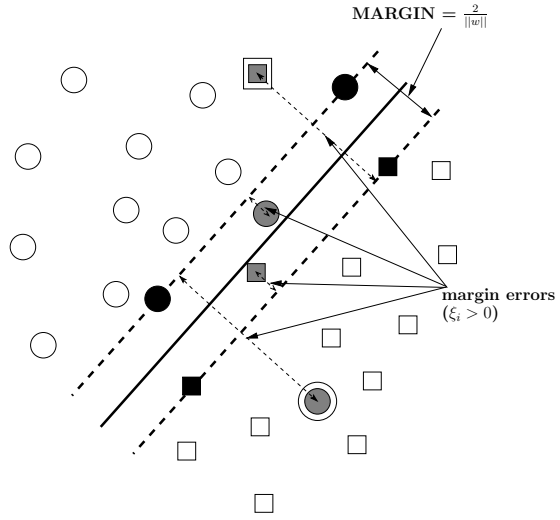

Fig. 1. Classification problem solved by support vector machines. The solid line represents the separating hyperplane, while dotted lines are hyperplanes with confidence margin equal to one. Black points are unbound SVs, grey points are bound SVs and extra borders indicate bound SVs which are also training errors. All other points do not contribute to the function to be minimized. Dotted lines indicate the margin error ξ_i for bound SVs.

functions can be seen as similarity measures generalizing dot products to arbitrary domains and allowing to decouple learning algorithms from example representation. The above decision function can be rationalized as follows: the final decision is obtained from a linearly weighted combination of the outputs y_i for selected training examples, the weights being influenced by the similarity between the current example $\mathbf{z}$ and the training example $\mathbf{z}_i$. In this sense the use of kernels is related to case-based or instance-based learning. *Universal* kernels [28] are powerful classes of kernel functions which can uniformly approximate any arbitrary continuous target function. A popular class of universal kernels is the Gaussian kernel:

$$K(\mathbf{z}, \mathbf{z}') = \exp(-\gamma ||\mathbf{z} - \mathbf{z}'||^2) \tag{7}$$

where γ is a positive constant. Kernel functions for structured data such as string, trees and graphs have been developed in the literature. In some cases the "kernel trick" transforms the computational complexity in a radical way, from practically unaffordable scalar products to efficient computations. Details on kernel machines can be found in several textbooks, e.g., see [29].

B. Support vector ranking

The SVM formulation can be easily adapted to learning the utility function for ranking: $U(\mathbf{z}) = \langle \mathbf{w}, \Phi(\mathbf{z}) \rangle$. The constant term b can be omitted because only differences of U values matter for the ranking. Given a ranking dataset $\mathcal{D} = \{(\mathbf{z}_1^{(i)}, \ldots, \mathbf{z}_{\ell_i}^{(i)}), (y_1^{(i)}, \ldots, y_{\ell_i}^{(i)})\}_{i=1}^s$, it suffices to impose constraints on the correct pairwise ordering of instances within a sequence: $U(\mathbf{z}_h^{(i)}) > U(\mathbf{z}_k^{(i)}) \iff y_h^{(i)} < y_k^{(i)}$. The resulting minimization problem can be written as:

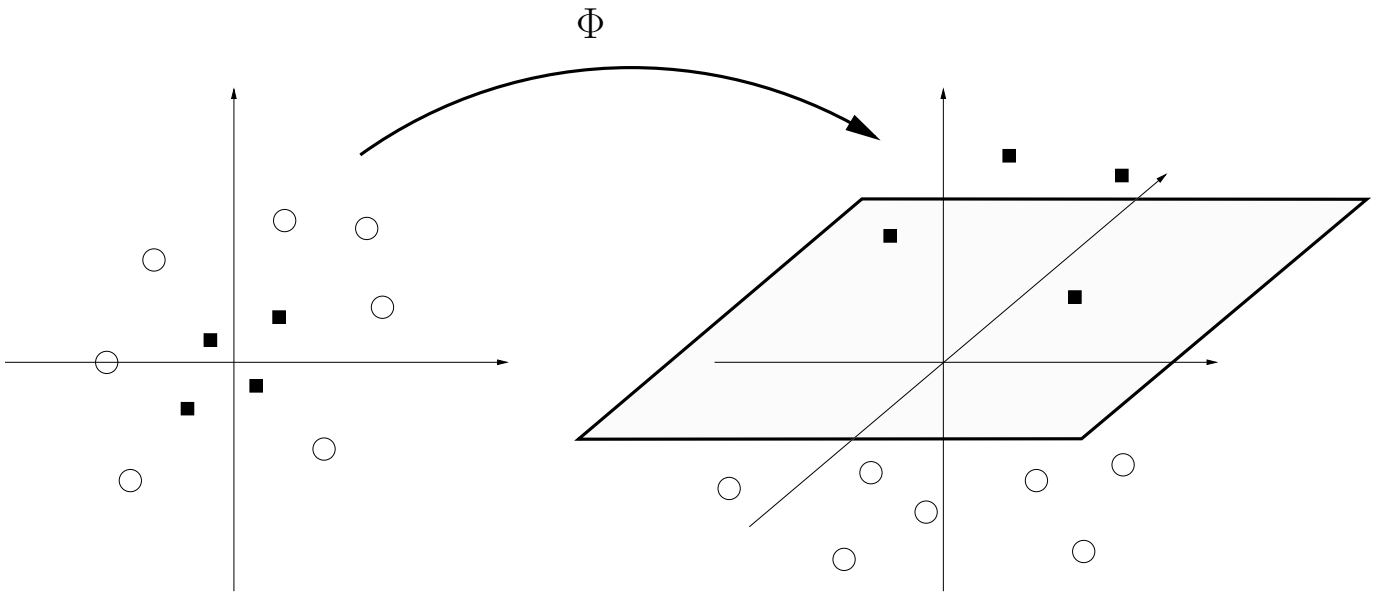

Fig. 2. Higher dimensional feature space projection via mapping function Φ for non-linearly separable problem.

$$\min_{\mathbf{w}\in\mathcal{Z},\boldsymbol{\xi}\in\mathbb{R}^*} \quad \frac{1}{2}||\mathbf{w}||^2 + C\sum_{i=1}^{s}\sum_{h_i,k_i}\xi_{i,h_i,k_i} \tag{8}$$

subject to:

$$\langle \mathbf{w}, \Phi(\mathbf{z}_{h_i}^{(i)})\rangle - \langle \mathbf{w}, \Phi(\mathbf{z}_{k_i}^{(i)})\rangle \geq 1 - \xi_{i,h_i,k_i}$$
$$\xi_{i,h_i,k_i} \geq 0$$
$$h_i, k_i : y_{h_i}^{(i)} < y_{k_i}^{(i)}$$
$$i = 1, \ldots, s$$

Note that the formulation allows to naturally account for ties and partial rankings, as constraints are only included whenever two examples should be ranked differently.

The problem is equivalent to the standard SVM formulation in (6) if we consider examples as pairs $\delta\Phi(\mathbf{z}_{h_i}^{(i)}, \mathbf{z}_{k_i}^{(i)}) = \Phi(\mathbf{z}_{h_i}^{(i)}) - \Phi(\mathbf{z}_{k_i}^{(i)})$, all pairs labels as positive and no bias term as it cancels out in the difference. In the dual formulation, the utility function thus becomes a linear combination of kernel functions on example pairs:

$$U(\mathbf{z}) = \sum_{(\mathbf{z}_{h_i}^{(i)}, \mathbf{z}_{k_i}^{(i)})\in SV} \alpha_{i,h_i,k_i}\left(K(\mathbf{z}_{h_i}^{(i)}, \mathbf{z}) - K(\mathbf{z}_{k_i}^{(i)}, \mathbf{z})\right)$$

By noting that each example $\mathbf{z}_j^{(i)}$ can appear multiple times in the summation, depending on the number of constraints it is involved in, we can rewrite the utility function more compactly as a linear combination of kernel functions on examples:

$$U(\mathbf{z}) = \sum_{\mathbf{z}_j^{(i)}\in SV^*} \alpha_{i,j} K(\mathbf{z}_j^{(i)}, \mathbf{z})$$

where:

$$\alpha_{i,j} = \sum_{\substack{(\mathbf{z}_{h_i}^{(i)}, \mathbf{z}_{k_i}^{(i)})\in SV \\ : \mathbf{z}_j^{(i)} = \mathbf{z}_{h_i}^{(i)}}} \alpha_{i,h_i,k_i} - \sum_{\substack{(\mathbf{z}_{h_i}^{(i)}, \mathbf{z}_{k_i}^{(i)})\in SV \\ : \mathbf{z}_j^{(i)} = \mathbf{z}_{k_i}^{(i)}}} \alpha_{i,h_i,k_i}$$

therefore reducing the computational cost of calculating the utility function.

IV. The BC-EMO algorithm

Support vector ranking has a number of desirable properties making it a suitable candidate for learning user preferences. First, it accepts supervision in terms of pairwise preferences, a much more affordable request for a human decision maker than a quantitative quality score. Second, it is well grounded on learning theory: its trading off data fitting and complexity of the learned hypothesis allows to effectively deal with noisy observations, a situation which is quite likely to occur when receiving feedback from a human user with only partial knowledge on the domain at hand. Third, the ability to implicitly project data onto a higher dimensional feature space via the kernel trick provides the needed flexibility in order to best approximate the underlying preference model of the specific user. The polynomial value function in eq. (5), for instance, can be easily implemented by an m-th degree inhomogeneous polynomial kernel: $K(\mathbf{z}, \mathbf{z}') = (1 + \langle \mathbf{z}, \mathbf{z}'\rangle)^m$.

The problem boils down to learning the utility function from examples of user preferences, and employing the learned function to guide the search for the globally optimal solution. The most natural approach would be to turn the MOO problem into a single objective optimization problem by directly optimizing the learned utility function. However, this requires the learned function to retain Pareto optimality, which is guaranteed if the function has non-negative weights. This non-negativity constraint cannot be easily incorporated into the general SVM formulation while retaining the advantages of the kernel

trick. A simple and effective alternative can be conceived when the preference function is employed in conjunction with evolutionary optimization algorithms. EMOAs [2], [3] typically rely on Pareto dominance classification as a fitness measure to guide the selection of the new population. Previous works [7] already highlighted that this choice can be rather suboptimal for increasing number of objectives and proposed a refined preference ordering based on the notion of order of efficiency [8]. Here we exploit the same idea by replacing order of efficiency with value of the learned utility function. An unordered population combining parents and offsprings of the current generation is sorted in steps by:

1) collecting the subset of non-dominated individuals in the population

2) sorting them according to the learned utility function

3) appending to the sorted set the result of repeating the procedure on the remaining dominated individuals.

Algorithm 1 describes this ranking procedure which clearly retains Pareto optimality. The procedure is terminated as soon as the desired set of s individuals is obtained. Any EMOA can be equipped with such a preference ranking procedure in order to guide the selection of the next population. Note that the combined ranking according to dominance and utility function preference is employed whenever comparisons between candidate individuals have to be made in creating the next generation. Algorithm 2 describes the procedure of a generic training iteration, which combines training or refinement of the utility function and preference ranking according to the learned function. Training consists of:

1) selecting a set of exa training individuals as the best ones according to the current criterion, where the utility function is replaced by random selection (within non-dominated individuals) at the first training iteration;

2) collecting pairwise preferences for these individuals from the decision maker and adding them to the set of training instances (empty at the first training iteration);

3) performing a kernel selection phase by a k-fold cross validation procedure in order to choose the best kernel for the problem at hand;

4) training the utility function on the overall set of examples with the chosen kernel.

The learned utility function is then used to sort the remaining set of candidate individuals.

The overall procedure, which we name BC-EMO as an acronym for *Brain-Computer Evolutionary Multi-Objective Optimization*, is described in Algorithm 3 for a generic EMO algorithm. The algorithm parameters are: the number of allowed training iterations ($maxit$), the number of training individuals for iteration (exa), the number of generations before the first training iteration (gen_1) and between two successive training iterations (gen_i), a minimal performance requirement for prematurely stopping the training phase ($thres$), the size of population for the EMO algorithm (s). Additional parameters can be required depending on the specific EMO algorithm employed, see the experimental section. The algorithm alternates a training phase where the current utility function is updated according to the DM feedback, and a search phase guided by the preference ordering procedure. When either the maximum number of training iterations or the desired accuracy level are reached, an additional search phase is conducted producing the final ordered population.

Figure 3 shows an illustrative example of BC-EMO application to a simple multiobjective 0/1 knapsack problem [30]. The setting consists of a set of items each one having a certain weight and profit value, and a set of knapsacks with limited

Algorithm 1 Preference ordering based on utility function

procedure PREFORDER(P_i, U, s)

Input:
P_i unordered population
U utility function
s size of output population
Output:
P_o ordered population

identify Pareto non-dominated individuals P_i^* in P_i
build P_o ordering individuals in P_i^* according to U
if $len(P_o) \geq s$ **then**
  return first s elements of P_o
else
  collect Pareto dominated individuals $P_i^d \leftarrow P_i \setminus P_i^*$
  if $P_i^d \neq \emptyset$ **then**
    $P_o^d \leftarrow$PREFORDER(P_i^d, U, $s - len(P_o)$)
    concatenate the two lists $P_o \leftarrow P_o + P_o^d$
  end if
  return P_o
end if
end procedure

Algorithm 2 Training procedure at a generic EMO iteration

procedure TRAIN(P_i, U_i, s, exa)

Input:
P_i unordered population
U_i current version of the utility function
s size of output population
exa number of training individuals for iteration
Output:
P_o ordered population
U_o refined version of the utility function
res_o estimated performance of refined utility function

$P_{tr} \leftarrow$ PREFORDER(P_i,U_i,exa)
obtain pairwise preferences for P_{tr} from the DM
sort P_{tr} according to user preferences
add P_{tr} to the current list of training instances
Choose best kernel K by k-fold cross validation
$U_o \leftarrow$ function trained on full training set with K
$res_o \leftarrow$ k-fold cv estimate of function performance
$P_o^* \leftarrow$ PREFORDER($P_i \setminus P_{tr}$, U_o, $s - len(P_{tr})$)
$P_o \leftarrow P_{tr} + P_o^*$
return P_o, U_o, res_o
end procedure

Figure 3 shows an illustrative example of BC-EMO application to a simple multiobjective 0/1 knapsack problem [30]. The setting consists of a set of items each one having a certain weight and profit value, and a set of knapsacks with limited

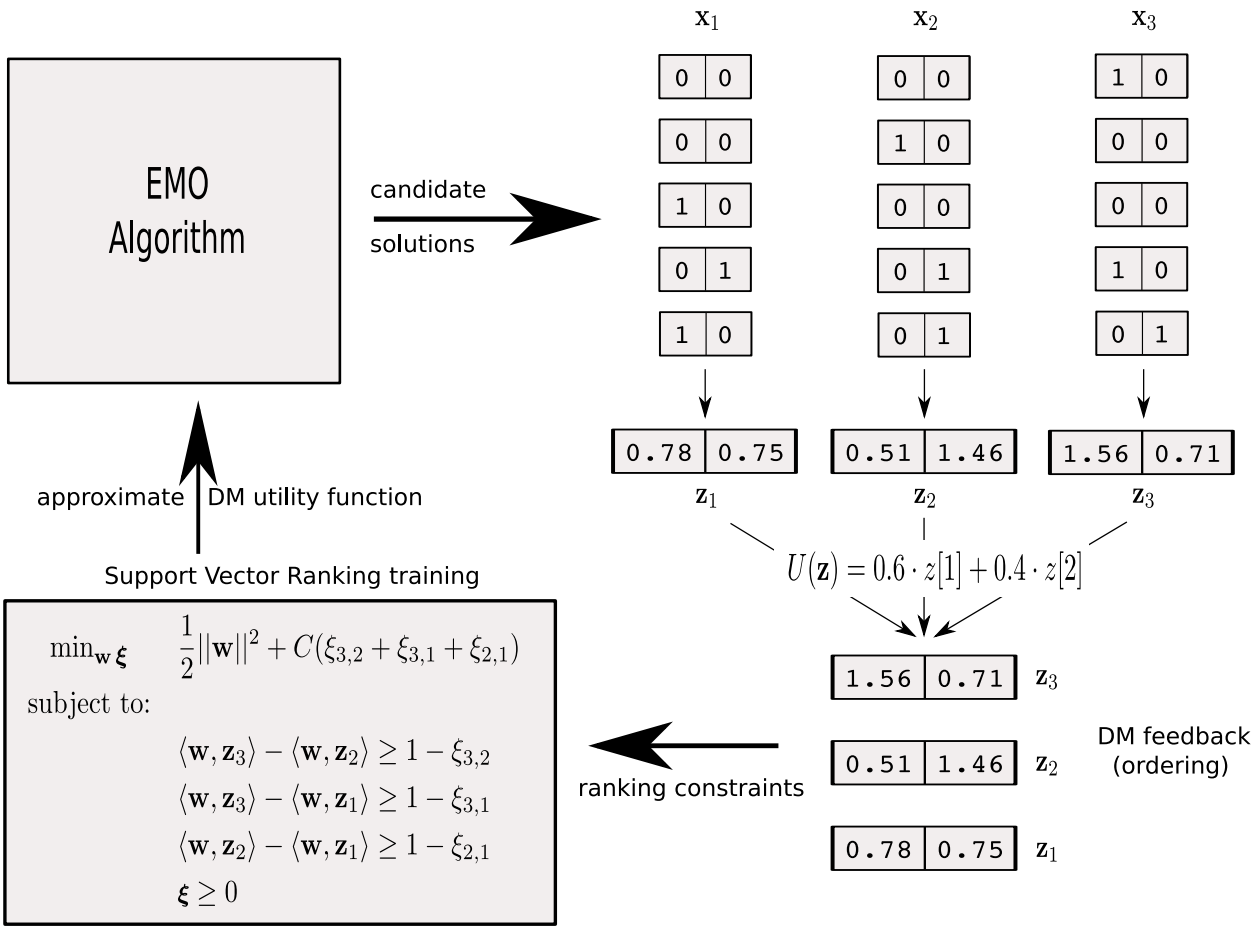

Fig. 3. Example of BC-EMO application of multiobjective 0/1 knapsack problem with five items and two knapsacks, assuming a linear utility function $U(\mathbf{z})$. Items profits are: $[0.81, 0.51, 0.07, 0.75, 0.71]$

capacities. Items have to be assigned to knapsacks trying to maximize the overall profit without exceeding knapsacks capacities and without assigning the same item to multiple knapsacks. This well-known combinatorial optimization problem is NP-hard in general; we use it here for illustrative purposes and defer its rigorous mathematic formulation to the experimental section. First, the selected EMO algorithm generates a set of candidate solutions as item assignments: the figure shows three candidate solutions $\mathbf{x}_j$, each consisting of a binary assignment for two knapsacks for each of the five available items. Each solution is converted into an objective vector $\mathbf{z}_j$ as the sum of the profits of the selected items for each knapsack. Objective vectors are passed to the DM who ranks them according to her (unknown) utility function and returns the ordered list as feedback. This feedback is converted into pairwise constraints for the support vector ranking training procedure (eq. (8)). After training, the predicted utility function is employed to guide the search for novel candidate solutions. For simplicity of presentation we dropped the dependency from the training iteration (i in equation (8)) and considered a linear utility function and a linear kernel (i.e. no feature mapping Φ).

Our BC-EMO algorithm is a generic formulation which can be implemented on top of any EMO algorithm. In this work we employed the NSGA-II [31] EMOA[1]. NSGA-II runs in its original formulation, including the crowded-comparison operator for guaranteeing a sufficiently diversified population, for gen_1 generations. Our preference model is then trained according to the DM feedback and the modified preference ordering procedure in Algorithm 1 replaces both the ordering of the new population and the selection criterion of the binary tournament selection operator. The crowding distance mechanism is switched off at this point as we are interested in directing the generation of new individuals in the most interesting regions according to the DM preferences. The preference learning module is based on the support vector machine package SVMlight[2].

Computational complexity of SVM [32] is between $O(N_{SV}^2)$ and $O(N_{SV}^3)$ depending on the value of the regularization parameter C, where N_{SV} is the number of support vectors. In the worst case, N_{SV} is quadratic in the number of instances for support vector ranking, as examples are pairs of instances. However, as supervision is provided interactively by the DM, the number of instances must be extremely limited, making support vector ranking learning negligible with respect

[1]available at http://www.iitk.ac.in/kangal/codes.shtml
[2]available at http://svmlight.joachims.org/

Algorithm 3 The BC-EMO algorithm

procedure BC-EMO($maxit$, exa, gen_1, gen_i, $thres$, s)

Input:
$maxit$ maximum number of allowed training iterations
exa number of training individuals for iteration
gen_1 generations before first training iteration
gen_i generations between two training iterations
$thres$ performance requirement to stop training
s size of population
Output:
P final ordered population

$res \leftarrow 0$, $it \leftarrow 0$, $U \leftarrow$ RAND
run the plain EMO for gen_1 generations
collect last population P
while $it \leq maxit \wedge res < thres$ **do**
$P, U, res \leftarrow$ TRAIN(P, U, s, exa)
run the EMO for gen_i generations
guided by PREFORDER with U
collect last population P
end while
run the EMO for remaining number of generations
guided by PREFORDER with U
return the final population P
end procedure

to collection of DM feedback. For instance, the whole set of 4,800 experiments for the 10 0/1 Knapsacks problem (for different random seed, number of training instances and training iterations, see Section VI) run in 12 hours on an Intel(R) Xeon(R) CPU at 2.80GHz, amounting to 9 seconds per experiment on average.

Concerning the choice of algorithm parameters, $maxit$, exa and $thres$ can be selected by the DM depending on the effort she is willing to put in providing feedback as well as depending on the desired solution quality. The number of training iterations does not need to be fixed at the beginning: the DM can choose whether to require a further training iteration by comparing the ranking provided by the algorithm with her own preferences. As a general guideline, a larger number of instances on the first training iteration tends to improve quality more than multiple training iterations. However, in a real-world setting an adaptation of the preference of the DM herself should be also accounted for, as will be discussed in Section VII, and more complex active learning strategies can be pursued [33]. The number of generations before the first training iteration (gen_1) should allow a reasonable coverage of the Pareto front in order not to miss portions possibly containing the DM preferred solutions; it should thus be of the same order of the number of generations for a plain MOO run on the same problem. The number of generations between two training iterations (gen_i) should be small enough to avoid a premature focus on a suboptimal search area, but sufficient to collect a different set of solutions: it could actually be automatically set by measuring the difference w.r.t the previous training set. Parameters of the EMO algorithm depend on the specific implementation employed: we used default values for all parameters of NSGA-II. Finally, critical SVM parameters should be adapted to the problem at hand: the choice of the kernel for non-linear utility functions is already automated by an internal cross validation strategy. The same strategy can be employed to choose the regularization parameter C, possibly including a decaying behaviour to progressively forget past feedback (see Section VII). A full optimization of all algorithm parameters is beyond the scope of this paper, but automated approaches for critical parameter tuning of generic algorithms were shown to be extremely effective [34] in practice.

V. RELATED WORK

Interactive multiobjective optimization has been an active research area in the last decades. In the field of EMO [4], a survey of preference-handling methods is presented in [35], and a recent survey of interactive EMO algorithms is given in [36]. The objective of this section is not to give an exhaustive review of the different papers but to compare our approach with some significant and related approaches dealing with modeling user preferences and interacting with the DM.

Artificial neural networks have been used in a number of works [37], [38], [39] to learn user preferences in an interactive fashion. In the interactive FFANN procedure [37], a feed-forward artificial neural network is trained to approximate the user utility function using a sample of user evaluated non-dominated solutions, and employed as the objective function of a non-linear optimization problem. However, there is no guarantee that the improved solutions found by the non-linear optimization problem are non-dominated. FFANN were later [38] used within an IWTP to replace the diversity selection criterion with a user preference criterion as predicted by the FFANN. While the resulting procedure is guaranteed to always produce non-dominated solutions, the role of FFANN is limited to halving the set of solutions to be presented to the user at each iteration, and they cannot directly guide the search procedure. Huang et al [39] developed an alternative approach where a FFANN is directly employed to produce refined weight vectors for the next AWTP. The network is trained by using as inputs weight vectors and as outputs the utility of the solution obtained by the AWTP with the corresponding weight vector. An optimization problem is then solved in order to determine the set of weights maximizing the output of the trained network, and the obtained weights are used in the next iteration of the IWTP.

While retaining the guarantee to produce non-dominated solutions, our approach has two main advantages. First, the IWTP based methods assume to be able to constraint the search to the correct subspace by relying on a linear set of weights, one for each objective. Our approach directly employs the learned utility function to guide the search and can be applied to highly non-linear combinations of objectives such as those in eq. (5). By automatically tuning the kernel to the problem at hand one can effectively approximate utility functions of varying complexity. Second, our method can be trained with pairwise preference supervision instead of quantitative

scores. As previous approaches pointed out [37], [38], pairwise preferences can be converted into scores following the method of the Analytical Hierarchical Process (AHP) [40]: a reciprocal comparison matrix is constructed from the pairwise comparisons and the normalized principal eigenvector of the matrix is computed. Each component of the normalized eigenvector, which is also named priority vector in this context, can be viewed as the score of the corresponding solution and used as the desired target. However, while the procedure perfectly fits the need to order a set of instances according to their pairwise comparisons, a function trained to fit such scores on a certain training set is not guaranteed to correctly generalize to unseen instances. The function is required to match scores which depend on the training set generating the matrix, while the ranking constraints in problem (8) simply require the learned function to correctly sort the examples.

A number of approaches exist combining interactive EMO with pairwise preference information from the DM. Quan et al [41] developed an approach based on the notion of imprecise value function [42]. Pairwise preferences are translated into a set of constraints on the weights of a linear combination of normalized objectives, which determines a constrained subspace W. Two arbitrary solutions can be compared by solving two linear programming problems, minimizing $\sum_k w_k \left[v_k(z_k) - v_k(z'_k)\right]$ and its inverse, where $v_k()$ is a normalization function. If either of the two problems has a positive minimum, preference between the two solutions can be uniquely determined, and in turn used to assign fitness to the individuals. Figueira et al. [43] also developed methods relying on the set of all *additive* utility functions consistent with user preferences:

$$U(\mathbf{z}) = \sum_{k=1}^{m} u_k(z_k)$$

where single-objective functions $u_k()$ are general non-decreasing functions. These utility functions are not used to guide the search of the evolutionary algorithm, but rather to recover desired solutions from the whole Pareto front returned by the search. Necessary (resp. possible) pairwise rankings are produced for pairs of solutions $\mathbf{z}$ and $\mathbf{z}'$ such that $U(\mathbf{z}) \geq U(\mathbf{z}')$ for all (resp. at least one of) the utility functions consistent with user preferences. These rankings are employed to generate a new sample of reference solutions together to their user preferences by interacting with the DM, and the process is repeated until the DM is satisfied with the obtained solutions. The method also accounts for intensity of preference, both global and at the level of single objectives. Phelps and Köksalan [44] employ the weighted L_p utility function in eq. (3) trained from pairwise preferences according to the middlemost weights technique [45]:

$$\begin{aligned} \max \quad & \epsilon \\ \text{subject to:} \quad & \\ & \sum_{k=1}^{m} w_k = 1,\ w_k \geq \epsilon,\ \forall k = 1, \ldots, m \\ & \sum_{k=1}^{m} w_k (z^*_k - z_{i,k})^t \leq \sum_{k=1}^{m} w_k (z^*_k - z_{j,k})^t - \epsilon \\ & \forall \mathbf{z}_i \succ \mathbf{z}_j \end{aligned} \tag{9}$$

where ϵ is a preference margin and $z_{i,k}$ indicates the k^{th} component of objective vector $\mathbf{z}_i$. Constraints are iteratively removed in chronological order in case of infeasibility. The method shares a number of similarities with our approach: the use of pairwise similarities as constraints in learning the utility function, the maximization of a preference margin and the use of the learned utility function as a fitness measure for the evolutionary algorithm. All these methods have the advantage of directly learning a function which retains Pareto optimality, with the drawback of constraining in different ways the set of allowed utility functions. On the other hand, our method aims at exploiting the ability of kernel machines to learn arbitrary functions, together with their noise robustness implied by the trade-off between data fitting and complexity of the learned hypothesis, which naturally accounts for imprecise and contradictory user preferences. These features closely correspond to real-world situations characterized by strongly non-linear and DM-dependent utility functions, and by the complex evolution of a decision process with possible imprecisions and even partial contradictions.

VI. Experimental evaluation

The experimental evaluation aims at demonstrating the effectiveness of the BC-EMO algorithm in approximating user preferences and correctly guiding the search towards the most preferred solution. Given this focus, we did not attempt to fine-tune non-critical parameters which were fixed to reasonable values for all experiments. We chose a population size of 100, 500 generations, probability of crossover equal to 1 and probability of mutation equal to the inverse of the number of decision variables. The number of initial generations (gen_1) was set to 200, while the number of generations between two training iterations (gen_i) was set to 20. A perfect fit ($thres = 1$) was required in order to prematurely stop training. The number of training iterations and examples per iteration were varied in the experiments as detailed later on in the section. Concerning the learning algorithm, we fixed the C regularization parameter in eq. (8) to 100. A preliminary analysis showed it to be a reasonable value considering the cost of the resulting optimization problem and an assumption of no noise in the DM evaluation. In general, this parameter can be tuned to the problem at hand as it depends on the kernel employed, the amount of noise and the number of available training instances.

We evaluated our approach on both discrete combinatorial and continuous problems. Each problem class challenges different aspects of MOO and can be scaled to any number of decision variables and objectives. Table I describes the test problem classes we employed. For combinatorial optimization we focused on multiobjective 0/1 knapsacks problems [30]. We considered problems with 100 items and a number of knapsacks varying from 2 to 10. Profit and weight values for each item were randomly chosen in the [0,1] range, while knapsacks capacities were randomly chosen ranging from the

TABLE I
TEST PROBLEM CLASSES EMPLOYED IN THE STUDY (FROM [30], [46])

Name	Description
0/1 KNAPSACKS	$\max_{\mathbf{x}\in\Omega} \boldsymbol{f}(\mathbf{x})$ $\Omega = \{\mathbf{x} \mid x_{i,j} \in \{0,1\}\ \forall i = 1,\dots,n\ j = 1,\dots,m\}$ $f_j(\mathbf{x}) = \sum_{i=1}^{n} p_i x_{i,j} \quad \forall j = 1,\dots,m$ $\sum_{i=1}^{n} w_i x_{i,j} \leq c_j \quad \forall j = 1,\dots,m$ $\sum_{j=1}^{m} x_{i,j} \leq 1 \quad \forall i = 1,\dots,n$
DTLZ1	$\min_{\mathbf{x}\in\Omega} \boldsymbol{f}(\mathbf{x})$ $\Omega = \{\mathbf{x} \mid 0.25 \leq x_i \leq 0.75\ \forall i = 1,\dots,n\}$ $f_1(\mathbf{x}) = 1/2 \cdot (1 + g(\mathbf{x}_m)) \cdot x_1 x_2 \cdots x_{m-1}$ $f_2(\mathbf{x}) = 1/2 \cdot (1 + g(\mathbf{x}_m)) \cdot x_1 x_2 \cdots (1 - x_{m-1})$ $\vdots$ $f_m(\mathbf{x}) = 1/2 \cdot (1 + g(\mathbf{x}_m))(1 - x_1)$ $g(\mathbf{x}_m) = 100 \cdot \left[\lvert\mathbf{x}_m\rvert + \sum_{x_i \in \mathbf{x}_m} (x_i - 0.5)^2 - \cos(20\pi(x_i - 0.5)) \right]$
DTLZ6	$\min_{\mathbf{x}\in\Omega} \boldsymbol{f}(\mathbf{x})$ $\Omega = \{\mathbf{x} \mid 0 \leq x_i \leq 1\ \forall i = 1,\dots,n\}$ $f_1(\mathbf{x}) = x_1$ $\vdots$ $f_{m-1}(\mathbf{x}) = x_{m-1}$ $f_m(\mathbf{x}) = (1 + g(\mathbf{x}_m)) h(f_1, f_2, \dots, f_{m-1}, g)$ $g(\mathbf{x}_m) = 1 + \frac{9}{\lvert\mathbf{x}_m\rvert} \sum_{x_i \in \mathbf{x}_m} x_i$ $h = m - \sum_{i=1}^{m-1} \left[\frac{f_i}{1+g} (1 + \sin(3\pi f_i)) \right]$
DTLZ7	$\min_{\mathbf{x}\in\Omega} \boldsymbol{f}(\mathbf{x})$ $\Omega = \{\mathbf{x} \mid 0 \leq x_i \leq 1\ \forall i = 1,\dots,n\}$ $f_j(\mathbf{x}) = \frac{1}{\lfloor \frac{n}{m} \rfloor} \sum_{i=\lfloor (j-1)\frac{n}{m} \rfloor + 1}^{\lfloor j \frac{n}{m} \rfloor} x_i \ \forall j = 1,\dots,m$ $g_j(\mathbf{x}) = f_m(\mathbf{x}) + 4 f_j(\mathbf{x}) - 1 \geq 0\ \forall j = 1,\dots,m-1$ $g_m(\mathbf{x}) = 2 f_m(\mathbf{x}) + \min_{i,j=1, i\neq j}^{m-1} [f_i(\mathbf{x}) - f_j(\mathbf{x})] - 1 \geq 0$

minimum item weight to the sum of all weights as suggested in [30]. For continuous optimization we selected a set of MOO test problem classes from the popular DTLZ [46] suite. We retained the original formulation of these problems as minimization tasks. Note that $\mathbf{x}_m$ indicates the components of $\mathbf{x}$ from m up to n, according to the notation used in [46]. The choice of this subset of problems is motivated by the form of their Pareto fronts which allowed us to design non-linear tasks, as will be detailed later on in the section. For test problem DTLZ1, we slightly restricted the feasible set for decision variables with respect to the original formulation (from [0,1] to [0.25,0.75]) in order to rule out sparse objective vectors (i.e. those with one or few non-zero entries) from the Pareto front and make the preference learning task more challenging when increasing the number of objectives.

The first set of experiments aims at showing the effectiveness of the method in early focusing on the correct search area with very few queries to the DM, for different test problems

and number of objectives, in the setting of linear utility functions. For each problem class, we generated a number of test instances by varying the number of objectives m from 2 to 10 and by setting the size of the input decision vector to $n = 100$ for 0/1 KNAPSACKS, $n = 2m$ for DTLZ1 and DTLZ6 and $n = 10m$, as suggested in [46], for DTLZ7. A total of 36 test instances was thus considered. Furthermore, we generated linear utility functions by randomly choosing weights in the range (0,1].

Figure 4 reports a representative set of results for test problems with four, eight and ten objectives for the four problem classes. The evaluation measure is the approximation error in percentage with respect to a *gold standard* final solution, the one which is obtained by guiding the algorithm with the true utility function. Note that the ranking function does not need to correctly estimate the value of the utility function, and it does not in general, but only to rank good solutions higher than bad ones in order to correctly guide the search. Each graph reports three learning curves for an increasing number of training examples per iteration (exa), with one, two and three iterations ($maxit$) respectively. Results are the medians over 100 runs with different random seeds for the search of the evolutionary algorithm.

The problem classes present quite different characteristics. For combinatorial optimization problems, an approximation error of less than 2% is reached with between 10 and 20 training examples depending on the number of iterations. Multiple training iterations seem beneficial only when few training examples are available, possibly correcting a poor initial estimate of the DM preference. Quite surprisingly, the algorithm even improves over the gold standard when increasing the number of objectives, as shown by a negative value of the approximation error. We conjecture this can be due to a diversification effect produced by an imperfect estimate of the DM preference, but this positive result deserves further investigation. Concerning continuous optimization, problems in class DTLZ1 (second row) were easily solved with just five training examples for up to four objectives, but were the most difficult to solve for a high number of objectives. Problems in class DTLZ6 (third row) were the easiest to solve: an approximation error of less that 1% could be achieved with at most 15 examples and two training iterations even in the case of ten objectives. Problems in class DTLZ7 (fourth row) required more examples than those in DTLZ1 in general but lead to better approximations with a high number of objectives. If one considers the number of training iterations, a second iteration was beneficial in many cases to possibly correct an early suboptimal model, especially with few training examples. Further training iterations did not provide substantial advantages.

Note that the DM is only required to compare examples within each training iteration, as we assume that the last generation will represent the most relevant solutions to her. The learning algorithm (see Section III-B) combines all these rankings into a single optimization problem: each training iteration i provides l_i ordered examples and a total of $s = maxit$ iterations are available. Given the algorithm formulation in eq. 8, there is no need for a full ranking of solutions, but the DM can provide partial information as sets of pairwise comparisons. In the presented experiments, for simplicity, we assumed a complete ranking.

In the next set of experiments we aimed at testing the ability of the method to automatically adapt to non-linear user preferences. Our aim is simulating many real world problems where the most preferred solution is a compromise among the different objectives. We thus generated test cases making sure that the utility function projected the most preferred solutions in a central area of the Pareto front, while retaining the Pareto dominance property. Designing such utility functions is not trivial. We heuristically generated them for each test problem by the following process: 1) running the plain NSGA-II algorithm to reach an approximation of the Pareto front 2) computing a reference point as the average of the final population 3) computing its distance to all other points 4) randomly generating weights of a second order polynomial 5) computing the utility value for all points according to the generated polynomial 6) keeping the polynomial if the rank correlation between the utility values and the distances to the reference point was higher than 0.4. This should guarantee that solutions distant from the center are less preferred.

BC-EMO was allowed to choose by internal 3-fold cross validation among linear and second degree polynomial kernel as well as Gaussian kernel with γ selected in the set $\{e^{-3}, e^{-2}, e^{-1}, 1, e^{1}, e^{2}, e^{3}\}$.

Figure 5 (left) shows an example of Pareto front reached by the plain NSGA-II algorithm for the DTLZ1 problem with two objectives. Figure 5 (right) shows the value of Pareto front according to the following polynomial utility function:

$$0.28 \cdot z_1^2 + 0.29 \cdot z_2 z_1 + 0.38 \cdot z_2^2 + 0.05 \cdot z_1 \qquad (10)$$

Note that the solution which is most preferred by the DM ($z_1 = 0.249, z_2 = 0.251$) lies in the middle of the Pareto front. We ran our BC-EMO algorithm on this test problem and compared to two alternative versions: using a fixed linear kernel versus automatically tuning the kernel by internal k-fold cross validation. Figure 6 shows the approximation results obtained for an increasing number of training examples. Results are medians over 10 runs. The linear kernel is completely unable to improve the performance regardless of the amount of training examples, stopping at an error of approximately 8%. On the contrary, tuning the kernel allows to reproduce the user preferences accurately enough to drive the search towards the desired solution (see Figure 5 (right)). Let's note that in the initial point, corresponding to three examples, a linear kernel is chosen by default (insufficient data for a model selection by cross validation).

Problem DTLZ6 presents a highly disconnected Pareto front, as shown in Figure 7 (left) where plain NSGA-II was again employed to generate the sample. Figure 7 (right) shows the value of Pareto front according to the following polynomial utility function:

$$0.05 \cdot z_2 z_1 + 0.6 \cdot z_1^2 + 0.38 \cdot z_2 + 0.23 \cdot z_1 \qquad (11)$$

This utility function generates two separate non-linear regions, with the global minimum located in the left region

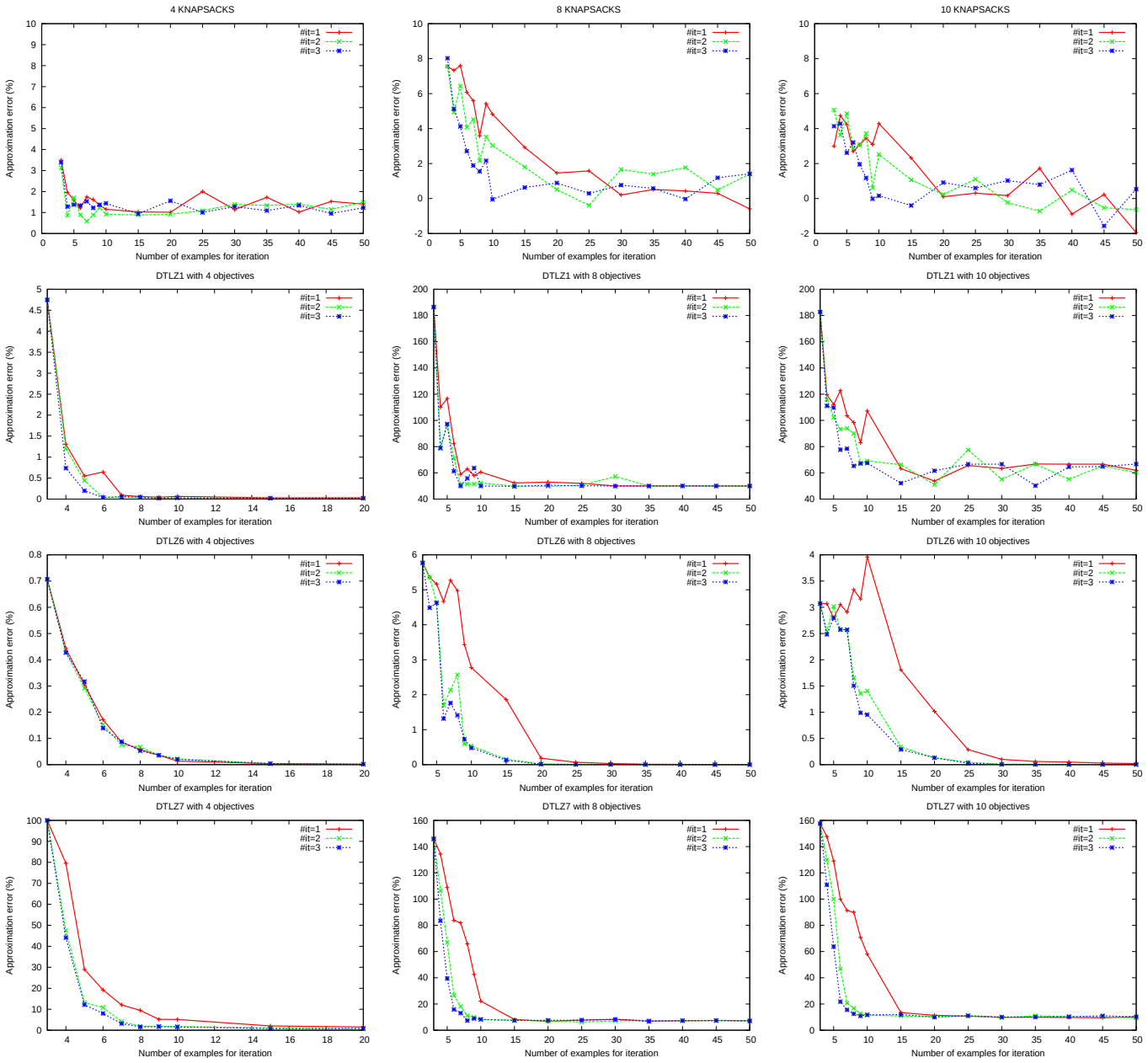

Fig. 4. Learning curves for an increasing number of training examples per iteration with one (red/solid), two (green/dashed) and three (blue/dotted) training iterations. Each row reports results for a different problem class: 0/1 KNAPSACKS, DTLZ1, DTLZ6 and DTLZ7 respectively. Each column reports results for a different number of objectives, respectively four, eight and ten. Results are medians over 100 runs. Note that y-axes have different ranges.

($z_1 = 0.192, z_2 = 3.622$). Learning curves for the resulting problem are shown in Figure 8. The linear utility function converges to a suboptimal solution, which is actually located at the left edge of the first region ($z_1 = 0, z_2 = 4$), while tuning the kernel allows to converge to the global optimum.

Finally, problem DTLZ7 presents a Pareto front similar to the one of DTLZ1, as shown in Figure 9 (left). Figure 9 (right) shows the non-linear surface obtained by using the following polynomial utility function:

$$0.44 \cdot z_1^2 + 0.33 \cdot z_1 + 0.09 \cdot z_2 z_1 + 0.14 \cdot z_2^2 \qquad (12)$$

Figure 10 reports learning curves for the problem, again showing that a linear kernel is totally unable to locate the correct region[3] stopping at approximately 33% error, while the tuned kernel version converges to the solution ($z_1 = 0.148, z_2 = 0.407$) with less than ten training examples.

Increasing the number of iterations did not change results significantly. In particular, using a linear kernel leads to a wrong direction when generating new individuals and further training iterations are useless to correct this initial behaviour.

It is interesting to note that all non-linear cases show a similar behaviour in terms of form of the approximating model. In detail, seriously suboptimal solutions are found in the few cases in which a linear kernel is selected. This happens

[3]Note that the solution found using the linear kernel lies in an area of the Pareto front which was not returned by the initial plain NSGA-II run, see Fig. 9 (right)

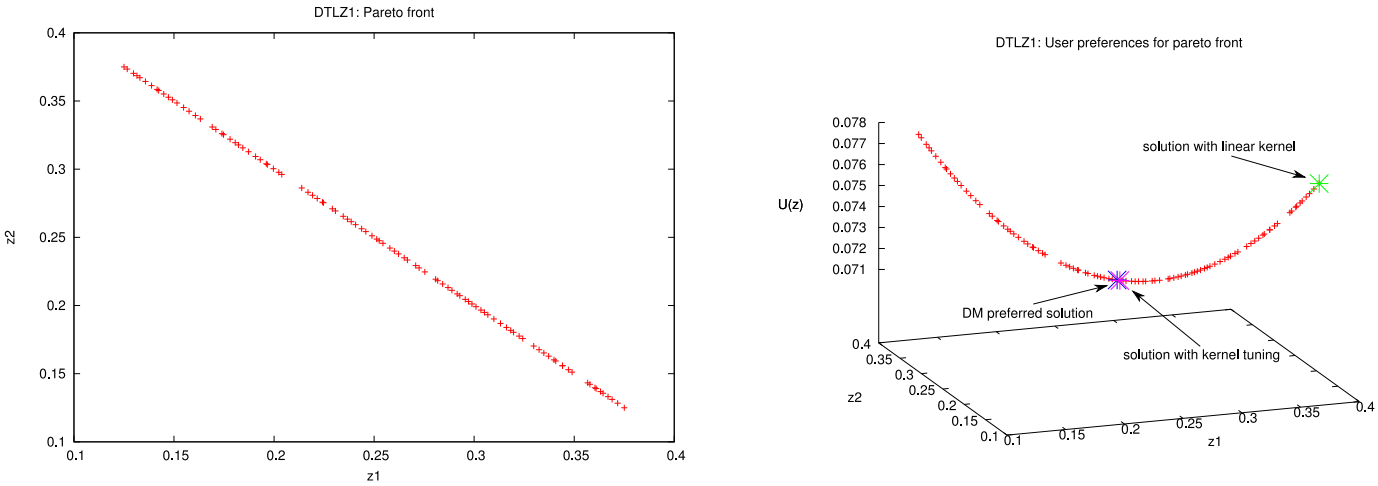

Fig. 5. Problem DTLZ1 with two objectives: (left) Pareto front for a sample run of plain NSGA-II without user preference; (right) preference values of the Pareto front according to the non-linear utility function in eq. (10).

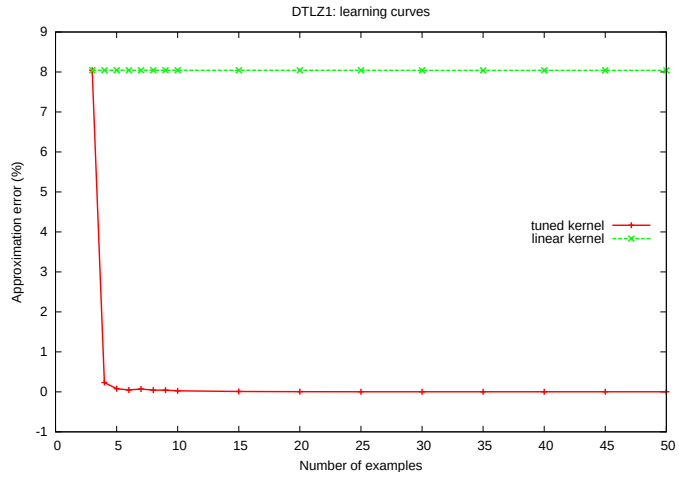

Fig. 6. Approximation error for the non-linear DTLZ1 problem as a function of the number of training instances: tuned kernel vs linear kernel. Results are medians over 10 runs.

by default with only three training examples and in rare cases with a very small numbers of examples: see for instance the spike for six examples in Figure 10. On the other hand, excellent fits are obtained when a second degree polynomial kernel is correctly chosen in the tuning phase. However, very good approximations are also achieved most of the times with Gaussian kernels, especially when enough training examples are available.

VII. Future directions

We showed that support vector machines, especially in combination with the kernel trick, provide an effective solution to model the fitness of solutions for an evolutionary algorithm, in the context of interactive multi-objective optimization (MOO).

This research work leaves many avenues for future explorations and generalizations.

MOO can be considered as a paradigmatic case of absence of complete information or of *information scarcity*: the decision maker specifies only a set of objectives but not their detailed combination. Future extensions and generalizations may consider applications to other poorly specified problems. It is widely acknowledged that most of the work in applied problem solving and optimization is related to formulating the problem by *eliciting knowledge*, by designing better and better models of the application domain through an iterative refinement process with different actors: domain experts, software experts, decision makers. Information is scarce and costly to obtain: problem solving should address this issue in an explicit and principled manner, by studying efficient and flexible modelling techniques and by quantifying the cost of obtaining improved models, e.g., the number and difficulty of questions raised during the modelling process and the computational costs for updating the model. Extensions can consider the following contexts:

- Dealing with a DM who is changing preferences. The current formulation of the approach puts on the same level feedback coming from different iterations of the search procedure. This behaviour can be suboptimal if the DM is herself learning her own preferences during the search, or if her utility function is too complex to be effectively approximated on the whole space, while only the shape locally guiding the search to the correct direction would be actually needed. It is rather straightforward to include a decay mechanism for feedback coming from previous iterations, by replacing the global regularization parameter C in equation (8) by a set of iteration-dependent parameters C_i. An exponential decay, for instance, would be obtained by setting $C_i = C \exp(i)$, allowing to early forget about past feedback and to focus on the current portion of the search space. By setting C to a small value, we force a behaviour which initially prefers simple approximations not trusting DM's feedback much. The relative importance of the feedback would then progressively increase on the rationale that the DM is becoming more confident of her evaluations.
- Objective (or *feature*) elicitation. The amount of information available is in many cases *less* that in the MOO

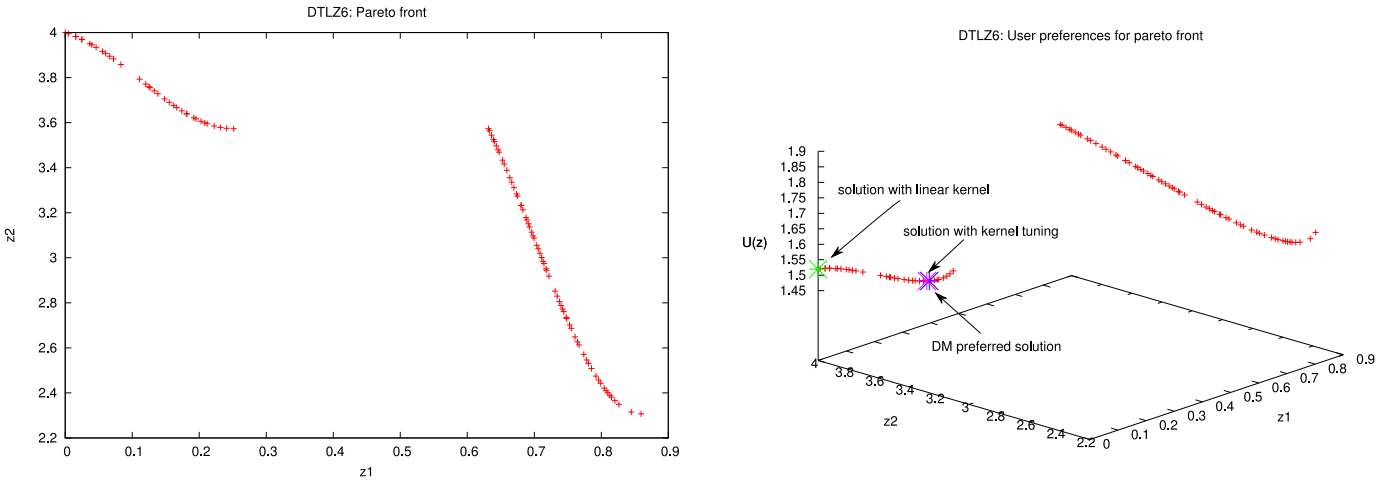

Fig. 7. Problem DTLZ6 with two objectives: (left) Pareto front for a sample run of plain NSGA-II without user preference; (right) preference values of the Pareto front according to the non-linear utility function in eq. (11).

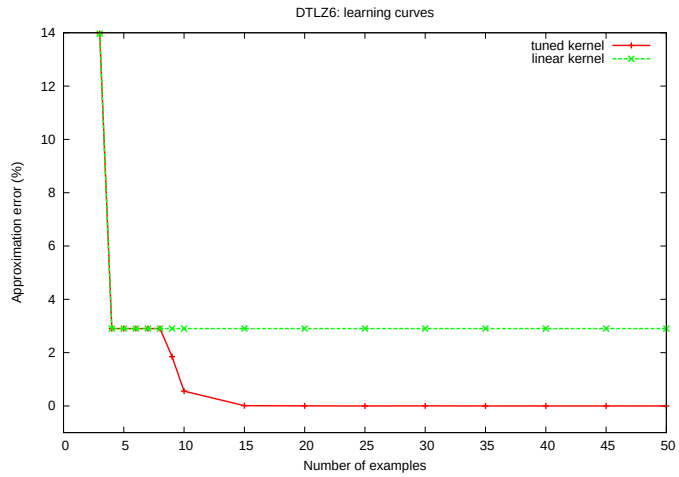

Fig. 8. Approximation error for the non-linear DTLZ6 problem as a function of the number of training instances: tuned kernel vs linear kernel. Results are medians over 10 runs.

context. In particular, objectives may be vague, fuzzy, not already formulated as clean mathematical functions. As challenging cases, let's consider the fashion business, where defining a formula for an elegant attire is far from trivial, or the industrial design area, the advertisement field, the medical sector. Machine learning can help in learning the objectives, the relevant and informative features, before learning how to combine them. Concept learning [47], *curriculum learning* [48] or *deep learning* [49] techniques can be tested here.

- Learning from hints. In some cases the available information may be formulated as constraints, or as hints about the form of the appropriate model [50]. For example, hints may indicate that the appropriate modelling functions are symmetric by exchanging some variables, or non-negative, or increasing as a function of specific inputs. Constraints can be rigid or *soft*, implying penalties for violating them.
- Adversarial models versus statistical models. While the SVM tools arise in a statistical machine learning context, and aim at designing systems which are approximately correct with high probability, other contexts assume an adversarial model, see for example [51]. An adversarial entity (real or theoretical) works to beat the system and one aims at minimizing the maximum *regret*, the worst-case loss under all possible realizations of a user's utility function. Results in this contexts are often very costly to obtain and not always strongly motivated by real-world applications (after all most users are interested in the actual obtained results and not so much in regret), but the study of worst-case results better defines the area where statistical and approximated results are the only possibility.
- Intelligent interaction strategies. *Active or query learning* methods are a viable candidate to further reduce the number of questions asked to the DM. In a world where computational costs are of much less significance than cognitive limitations, a less efficient method that relies on fewer or simpler questions might be more valuable than a faster one with higher cognitive requirements. The reduction in the number of questions can occur through the strategic choice of future examples to be presented for evaluations, depending on the past feedback received, and therefore on the current preference model. For example, if pairwise comparison queries are used, examples should be sufficiently different to permit a clear preference by the DM. In addition, the optimization context demands a shift of emphasis from standard active learning strategies: the aim is not to reconstruct an unknown model in all areas, but to concentrate on the areas which are most promising for optimization. An example in the context of intelligently-guided simulations is [52].
- Learning from related problems. *Transfer learning* techniques could provide added benefits when solving a series of related problems: we plan to evaluate the effectiveness of transferring the previous experience accumulated in two different contexts: new and related problems solved by the same DM (for example problems sharing a subset

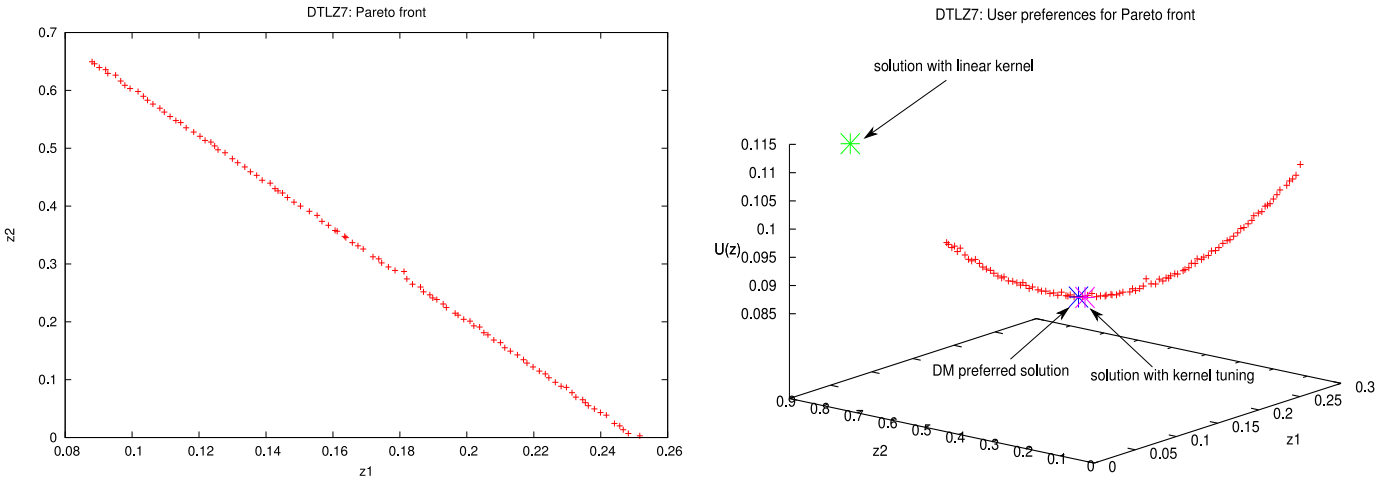

Fig. 9. Problem DTLZ7 with two objectives: (left) Pareto front for a sample run of plain NSGA-II without user preference; (right) preference values of the Pareto front according to the non-linear utility function in eq. (12).

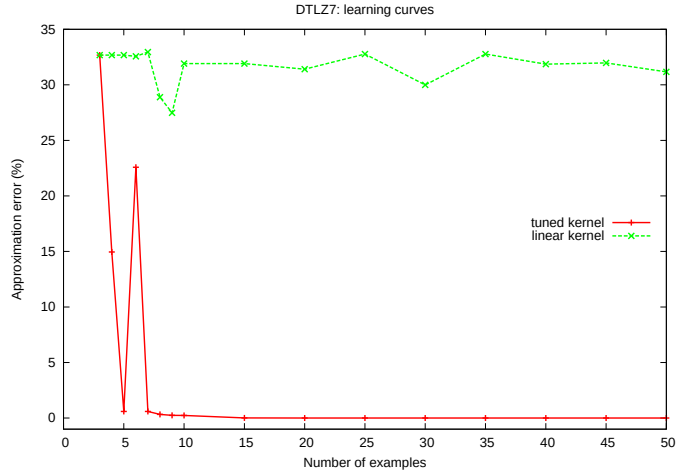

Fig. 10. Approximation error for the non-linear DTLZ7 problem as a function of the number of training instances: tuned kernel vs linear kernel. Results are medians over 10 runs.

of the objectives), or the same problem solved by DM's with different preferences.

- Learning in a discrete setting. An extension of this work to discrete multi-objective optimization tasks is of interest, by using the effective kernel functions recently introduced for structured discrete entities like graphs and trees [53].
- Group decision making. A situation in which multiple DMs need to be jointly considered could be addressed by pursuing a *multitask* approach [54]: each DM is modelled by a different utility function, but their parameters are connected to each other so that information can flow among related tasks. In Artificial Neural Networks, this is typically [54] obtained by sharing a common hidden layer connected to the per-task output ones. Kernel machines have been adapted [55] to multitask learning by acting on the weight regularizer (see eq.(6)(8)): task-dependent weights can be forced close to a common (regularized) weight vector, or task similarity can be included via the graph Laplacian of a task-similarity graph [55], to cite two useful examples. While these approaches assume that separate per-task predictions should be made on each example, a common aggregate decision on the utility of candidate solutions could be forced by averaging preferences of the DMs, taking the minimum between them, or any other sensible combination. The aggregation chosen should be reflected on the loss component of the SVM involving soft constraints, by penalizing incorrect average ranking in the first case, or incorrect worst ranking in the second.

Needless to say, all above methods can be tested in the traditional evolutionary context, not only in the multi-objective case, as ways to efficiently assign fitness values to population members where only partial information exists about the true value of solutions.

While the above points refer to potential extensions of this work by considering machine learning tools, future extensions of this investigation can consider a detailed analysis of the *robustness* of the proposed technique in different noisy scenarios. Noise can be intrinsic in the problem domain (for example when optimizing a system with stochastic outputs), or can be related to a noisy evaluation by the DM. For example, the DM may fail to consistently rank solutions, in particular if solutions tend to be close, or may make mistakes from time to time. A preliminary investigation dealing with robustness of the BC-EMO technique is presented in [56].

VIII. Conclusions

We presented a preference-based EMO algorithm (BC-EMO) characterized by its ability to learn an arbitrary utility function from a decision maker (DM) who expresses preferences between couples of selected solutions. The method to build a flexible preference model, possibly highly nonlinear, is based on Support Vector Machines and derived tools from the machine learning community. From the multi-objective decision making perspective, the main contribution of the method is its ability to function without any *a priori* assumptions on the shape on the shape of the DM's utility function.

The optimization methodology of Reactive Search Optimization (RSO) based on the paradigm of *learning while optimizing* is adopted in two directions: the progressive tuning of a preference model following a DM interactive evaluation and personal learning path, and the automated adaptation of the model form to one which is most appropriate, in a cross-validated manner, to the data collected during the interaction.

The method is robust as it can potentially withstand incomplete, imprecise and even contradictory feedback by the DM. The alternation of evolutionary optimization and DM ranking can be organized according to a flexible schedule, depending on the willingness by the DM to interact more times during the solution process and by intrinsic characteristics of the problem to be solved and the complexity of the user preferences.

The presented experimental results demonstrate the feasibility and effectiveness of the BC-EMO algorithm on a variety of benchmark tasks, with both linear and non-linear user preferences. We hope this work will motivate future extensions to integrate machine learning techniques into optimization and problem-solving techniques.

ACKNOWLEDGMENT

We acknowledge K. Deb, A. Pratap, S. Agarwal and T. Meyarivan for making available the software NSGA-II used as a starting point to develop our experiments, and T. Joachims for sharing the code of SVMlight.

REFERENCES

[1] J. D. Schaffer, "Multiple objective optimization with vector evaluated genetic algorithms," in *Proceedings of the 1st International Conference on Genetic Algorithms*. Hillsdale, NJ, USA: L. Erlbaum Associates Inc., 1985, pp. 93–100.

[2] N. Srinivas and K. Deb, "Multiobjective optimization using nondominated sorting in genetic algorithms," *Evolutionary Computation*, vol. 2, pp. 221–248, 1994.

[3] E. Zitzler and L. Thiele, "Multiobjective evolutionary algorithms: a comparative case study and the strength pareto approach," *Evolutionary Computation, IEEE Transactions on*, vol. 3, no. 4, pp. 257–271, Nov 1999.

[4] K. Deb, *Multi-objective optimization using evolutionary algorithms*. Wiley, 2001.

[5] P. Bosman and D. Thierens, "The balance between proximity and diversity in multiobjective evolutionary algorithms," *IEEE Transactions on Evolutionary Computation*, vol. 7, no. 2, pp. 174–188, 2003.

[6] J. Branke, "Consideration of Partial User Preferences in Evolutionary Multiobjective Optimization," in *Multiobjective Optimization: Interactive and Evolutionary Approaches*. Springer-Verlag Berlin, Heidelberg, 2008, pp. 157–178.

[7] F. di Pierro, K. Soon-Thiam, and D. Savic, "An investigation on preference order ranking scheme for multiobjective evolutionary optimization," *Evolutionary Computation, IEEE Transactions on*, vol. 11, no. 1, pp. 17–45, Feb. 2007.

[8] I. Das, "A preference ordering among various pareto optimal alternatives," *Structural and Multidisciplinary Optimization*, vol. 18, no. 1, pp. 30–35, 1999.

[9] J. March, "Bounded rationality, ambiguity, and the engineering of choice," *The Bell Journal of Economics*, pp. 587–608, 1978.

[10] 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.

[11] R. Battiti, M. Brunato, and F. Mascia, *Reactive Search and Intelligent Optimization*, ser. Operations research/Computer Science Interfaces. Springer Verlag, 2008, vol. 45.

[12] L. Citi, R. Poli, C. Cinel, and F. Sepulveda, "P300-based BCI mouse with genetically-optimized analogue control," *IEEE transactions on neural systems and rehabilitation engineering*, vol. 16, no. 1, pp. 51–61, 2008.

[13] D. Jones, "A Taxonomy of Global Optimization Methods Based on Response Surfaces," *Journal of Global Optimization*, vol. 21, no. 4, pp. 345–383, 2001.

[14] 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.

[15] J. Branke, K. Deb, K. Miettinen, and R. Słowiński, Eds., *Multiobjective Optimization: Interactive and Evolutionary Approaches*. Berlin, Heidelberg: Springer-Verlag, 2008.

[16] H. Takagi *et al.*, "Interactive evolutionary computation: Fusion of the capabilities of EC optimization and human evaluation," *Proceedings of the IEEE*, vol. 89, no. 9, pp. 1275–1296, 2001.

[17] B. Roy and V. Mousseau, "a theoretical framework for analysing the notion of relative importance," *Journal of Multi Criteria Decision Analysis*, 1996.

[18] R. Steuer, *Multiple Criteria Optimization: Theory, Computation, and Application*. Wiley, New York, 1986.

[19] L. Tanner, "Selecting a text-processing system as a qualitative multiple criteria problem," *European Journal of Operational Research*, vol. 50, no. 2, pp. 179–187, January 1991.

[20] V. V. Podinovskii, "The quantitative importance of criteria with discrete first-order metric scale," *Autom. Remote Control*, vol. 65, no. 8, pp. 1348–1354, 2004.

[21] R. Steuer and E. Choo, "An interactive weighted Tchebycheff procedure for multiple objective programming," *Mathematical Programming*, vol. 26, no. 3, pp. 326–344, 1983.

[22] R. Dell and M. Karwan, "An interactive MCDM weight space reduction method utilizing a Tchebycheff utility function," *Naval Research Logistics*, vol. 37, no. 2, 1990.

[23] C. Cortes and V. N. Vapnik, "Support vector networks," *Machine Learning*, vol. 20, no. 3, pp. 1–25, Sep 1995.

[24] S. Menchetti, "Learning preference and structured data: Theory and applications," Ph.D. dissertation, Universit degli Studi di Firenze, 2005.

[25] W. W. Cohen, R. E. Schapire, and Y. Singer, "Learning to order things," *Journal of Artificial Intelligence Research*, vol. 10, pp. 243–270, 1999.

[26] M. Collins and N. Duffy, "Convolution kernels for natural language," in *Advances in Neural Information Processing Systems 14*. MIT Press, 2001, pp. 625–632.

[27] M. Collins, N. Duffy, and F. Park, "New ranking algorithms for parsing and tagging: Kernels over discrete structures, and the voted perceptron," in *In Proceedings of ACL 2002*, 2002, pp. 263–270.

[28] C. A. Micchelli, Y. Xu, and H. Zhang, "Universal kernels," *J. Mach. Learn. Res.*, vol. 7, pp. 2651–2667, 2006.

[29] J. Shawe-Taylor and N. Cristianini, *Kernel Methods for Pattern Analysis*. New York, NY, USA: Cambridge University Press, 2004.

[30] S. Martello and P. Toth, *Knapsack problems: algorithms and computer implementations*. New York, NY, USA: John Wiley & Sons, Inc., 1990.

[31] K. Deb, A. Pratap, S. Agarwal, and T. Meyarivan, "A fast elitist multiobjective genetic algorithm: Nsga-ii," *IEEE Transactions on Evolutionary Computation*, vol. 6, pp. 182–197, 2002.

[32] C. Burges, "A tutorial on support vector machines for pattern recognition," in *Data Mining and Knowledge Discovery*. Boston: Kluwer Academic Publishers, 1998, (Volume 2).

[33] B. Settles, "Active learning literature survey," University of Wisconsin-Madison, Tech. Rep. 1648, 2009.

[34] F. Hutter, H. H. Hoos, K. Leyton-Brown, and T. Stützle, "ParamILS: an automatic algorithm configuration framework," *Journal of Artificial Intelligence Research*, vol. 36, pp. 267–306, October 2009.

[35] C. Coello, "Handling preferences in evolutionary multiobjective optimization: a survey," in *Proceedings of the 2000 Congress on Evolutionary Computation*, vol. 1, 2000.

[36] A. Jaszkiewicz and J. Branke, "Interactive Multiobjective Evolutionary Algorithms," in *Multiobjective Optimization: Interactive and Evolutionary Approaches*. Springer-Verlag Berlin, Heidelberg, 2008, pp. 179–193.

[37] M. Sun, A. Stam, and R. Steuer, "Solving multiple objective programming problems using feed-forward artificial neural networks: the interactive ffann procedure," *Manage. Sci.*, vol. 42, no. 6, pp. 835–849, 1996.

[38] ——, "Interactive multiple objective programming using tchebycheff programs and artificial neural networks," *Comput. Oper. Res.*, vol. 27, no. 7-8, pp. 601–620, 2000.

[39] H. Z. Huang, Z. G. Tian, and M. J. Zuo, "Intelligent interactive multiobjective optimization method and its application to reliability optimization," *IIE Transactions*, vol. 37, no. 11, pp. 983–993, 2005.

[40] T. Saaty, "Axiomatic foundation of the analytic hierarchy process," *Manage. Sci.*, vol. 32, no. 7, pp. 841–855, 1986.

[41] G. Quan, G. Greenwood, D. Liu, and S. Hu, "Searching for multiobjective preventive maintenance schedules: Combining preferences with evolutionary algorithms," *European Journal of Operational Research*, vol. 177, no. 3, pp. 1969–1984, 2007.
[42] C. White, A. Sage, and A. Dozono, "A model of multiattribute decisionmaking and trade-off weight determination under uncertainty," *IEEE Transactions on Systems, Man, and Cybernetics*, vol. 14, no. 2, pp. 223–229, 1984.
[43] J. Figueira, S. Greco, V. Mousseau, and R. Słowiński, "Interactive Multiobjective Optimization using a Set of Additive Value Functions," *Lecture Notes In Computer Science*, pp. 97–119, 2008.
[44] S. Phelps and M. Koksalan, "An interactive evolutionary metaheuristic for multiobjective combinatorial optimization," *Management Science*, pp. 1726–1738, 2003.
[45] M. Koksalan, M. Karwan, and S. Zionts, "An improved method for solving multiple criteria problems involving discrete alternatives," *IEEE Transactions on Systems, Man, and Cybernetics*, vol. 14, no. 1, pp. 24–34, 1984.
[46] K. Deb, L. Thiele, M. Laumanns, and E. Zitzler, "Scalable multi-objective optimization test problems," in *in Congress on Evolutionary Computation (CEC2002*, 2002, pp. 825–830.
[47] D. Angluin, "Queries and concept learning," *Machine learning*, vol. 2, no. 4, pp. 319–342, 1988.
[48] Y. Bengio, J. Louradour, R. Collobert, and J. Weston, "Curriculum learning," in *Proceedings of the 26th International Conference on Machine Learning*. ACM New York, NY, USA, 2009.
[49] Y. Bengio and Y. LeCun, "Scaling learning algorithms towards AI," *Large-Scale Kernel Machines*, 2007.
[50] Y. Abu-Mostafa, "Learning from hints," *Journal of Complexity*, vol. 10, pp. 165–165, 1994.
[51] D. Braziunas and C. Boutilier, "Minimax regret based elicitation of generalized additive utilities," in *Proceedings of the 23rd Conference on Uncertainty in Artificial Intelligence (UAI-07)*, 2007.
[52] M. Burl and E. Wang, "Active learning for directed exploration of complex systems," in *Proceedings of the 26th International Conference on Machine Learning*. ACM, 2009, pp. 89–96.
[53] T. Gärtner, "Kernels for structured data," Ph.D. dissertation, Universität Bonn, 2005.
[54] R. Caruana, "Multitask learning," *Machine Learning*, vol. 28, no. 1, pp. 41–75, 1997.
[55] T. Evgeniou, C. A. Micchelli, and M. Pontil, "Learning multiple tasks with kernel methods," *J. Mach. Learn. Res.*, vol. 6, pp. 615–637, 2005.
[56] P. Campigotto and A. Passerini, "Adapting to a realistic decision maker: experiments towards a reactive multi-objective optimizer." in *Proc.4th Learning and Intelligent Optimization Conference, LION 4, Venice, Italy, January 2010*, ser. LNCS, R. Battiti and C. Blum, Eds. Springer Verlag, 2010, in press.

Roberto Battiti Prof. Roberto Battiti received the Laurea degree in Physics from the University of Trento, Italy, in 1985 and the Ph.D. degree from the California Institute of Technology (Caltech), USA, in 1990. He is now full professor of Computer Science at Trento university, deputy director of the DISI Department (Electrical Engineering and Computer Science) and director of the LION lab at (machine Learning and Intelligent OptimizatioN). His main research interests are heuristic algorithms for optimization problems, in particular Reactive Search Optimization (RSO) for discrete and continuous optimization problems, neural networks and machine learning. He is now investigating new ways to integrate RSO into data mining, visual analytics and business intelligence. R. Battiti is a fellow of the IEEE. Full details about interests, research activities and scientific production can be found in the web: http://lion.disi.unitn.it/∼battiti/, http://reactive-search.org/ .

Andrea Passerini Andrea Passerini graduated in Computer Science at the University of Florence in 2000 and received his Ph.D. at the same University in 2004. He is currently Assistant Professor at the Department of Information Engineering and Computer Science of the University of Trento. His main research interests are in the area of machine learning, with a special enphasis on bioinformatics applications. In recent years he developed techniques aimed at combining statistical and symbolic approaches to learning, via the integration of inductive logic programming and kernel machines. He is also pursuing a deeper integration of machine learning approaches and complex optimization techniques. He coauthored more than forty scientific publications.