Handling concept drift in preference learning for interactive decision making

Paolo Campigotto, Andrea Passerini, and Roberto Battiti

DISI - Dipartimento di Ingegneria e Scienza dell'Informazione,
Università di Trento, Italy
{campigotto, passerini, battiti}@disi.unitn.it

Abstract. Interactive decision making methods use preference information from the decision maker during the optimization task to guide the search towards favourite solutions. In real-life applications, unforeseen changes in the preferences of the decision maker have to be considered. To the best of our knowledge, no interactive decision making technique has been explicitly designed to recognize and handle preference drift. This paper aims at covering this gap, by extending the Brain-Computer Evolutionary Multi-Objective Optimization (BC-EMO) algorithm to handle preference drift. BC-EMO is a recent multi-objective genetic algorithm. It exploits user judgments of couples of solutions to build incremental models of the user value function. The learnt model is used to refine the genetic population, generating the new individuals in the region of the Pareto front surrounding the favourite solution of the decision maker. The proposed extension of BC-EMO detects the changes of the user preferences by observing the decrease of prediction accuracy of the learnt model. The preference drift is jointly tackled by the BC-EMO learning phase, by a discounting policy for outdated training examples, and by the BC-EMO search phase, by encouraging diversification in the genetic population. Experimental results for a representative preference drift scenario are presented.

1 Introduction

Modeling real-world problems often generates optimization tasks involving multiple and conflicting objectives. Because the objectives are in conflict, a solution simultaneously optimizing all of them does not exist. The typical approach to multi-objective optimization problems (MOOPs) consists of searching for a set of trade-off solutions, called *Pareto-optimal* set, for which any single objective cannot be improved without compromising at least one of the other objectives.

Usually, the size of the Pareto-optimal set is large or infinite and the decision maker (DM) cannot tackle the overflow of information generated when analyzing it entirely. In this scenario, *interactive* decision making (IDM) techniques come to the rescue. They assume that the optimization expert (or the optimization software) cooperates with the DM. Through the interaction, the search process can be directed towards the DM preferred Pareto-optimal solutions and only a fraction of the Pareto-optimal set needs to be generated.

To the best of our knowledge, current IDM techniques consider a static preference model for the DM. This is rather unrealistic in many applications, where the DM has limited initial knowledge of the problem at hand. Only when the DM see the actual tentative solutions, she becomes aware of "what is possible". Confronted with this new knowledge, her preferences may evolve. Typical scenarios involve a DM introducing new objectives in her preference model during the search, changing the relations between the different objectives or adjusting her preference model according to the observed limitations of the feasible set. Furthermore, the DM may not be aware of her preference changes and may not explicitly alert the optimization component. From a learning perspective, interactive multi-objective optimization should thus be seen as a joint learning process involving the model and the DM herself [2].

In the machine learning (ML) community, the problem of learning in these changing conditions is known as learning under *concept drift* [13]. The problem has received increasing attention in last years, and a number of solutions have been proposed to tackle it. For a review of the recent approaches in this area, see [16]. In this work we consider concept drift in the specific setting of interactive optimization. We call *preference drift* the tendency of the decision maker to change her preferences during the interactive optimization stage.

To the best of our knowledge, no IDM technique has been explicitly designed to handle preference drift. Among the plethora of IDM algorithms, reference point methods [10, 12], which iteratively minimize the distance to ideal reference points provided by the DM, could in principle naturally handle preference drifts. However, the cognitive demands required to the DM can easily become prohibitive, especially when dealing with non-linear preference models and an increasing number of objective.

Machine learning techniques [14, 15, 8] have been employed in IDM by learning the user preferences in an interactive fashion, and can be easily adapted to deal with preference drifts. Most existing approaches are limited either by not guaranteeing the generation of Pareto optimal solutions, or by assuming a linear set of weights, one for each objective. We recently developed the Brain-Computer Evolutionary Multi-Objective Optimization (BC-EMO) algorithm [1] in order to overcome these limitations.

BC-EMO is a genetic algorithm that learns the preference information of the decision maker (formalized as a value function) by the feedback received when the DM evaluates tentative solutions. Based on this feedback, the predicted value function is refined, and it is used to modify the fitness measure of the genetic algorithm. The algorithm was shown [1] to early converge to the desired solution on both combinatorial and continuous problems with linear and non-linear value functions. The learning stage is based on a support vector ranking algorithm which provides robustness to inaccurate and contradictory DM feedback [3]. We thus selected BC-EMO as a natural candidate to be extended for managing preference drift.

The extension of BC-EMO for preference drift recovery is based on the approach of instance weighting [9], a popular strategy in the concept drift literature.

The instance weighting technique consists of reweighting the examples according to their predicted relevance for the current concept. We include this reweighting scheme in the learning component of the BC-EMO algorithm. A change detection monitor is responsible for activating the mechanism. In order to deal with concept drift in the specific setting of interactive optimization, we also introduce a diversification strategy aimed at escaping from minima which could become suboptimal for the changed preference of the DM.

The remainder of the paper is organized as follows. Section 2 introduces IDM and discusses the limitations of current techniques in regard to preference drift handling. Section 3 briefly reviews the BC-EMO algorithm, while Section 4 extends it to automatically handle preference drift. An experimental evaluation of the proposed extension is reported in Section 5. Section 6 draws some conclusions and proposes possible directions for future research.

2 Interactive decision making techniques

A MOOP can be stated as:

$$\text{minimize } \mathbf{f}(\mathbf{x}) = \{f_1(\mathbf{x}), \ldots, f_m(\mathbf{x})\} \tag{1}$$
$$\text{subject to } \quad \mathbf{x} \in \Omega$$

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; $\mathbf{f} : \Omega \to \mathbb{R}^m$ 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 exist 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 $\mathbf{f}(\mathbf{x})$ dominates $\mathbf{f}(\hat{\mathbf{x}})$. The set of Pareto-optimal points is called *Pareto set* (PS). The corresponding set of Pareto-optimal objective vectors is called *Pareto front* (PF).

Several IDM approaches have been developed to aid the DM in identifying her preferred solution [11], including evolutionary multi-objective algorithms (see for example [5] and contained references). IDM procedures exploit the preference feedback from the DM to refine a preference model, usually expressed as a value function.

In the popular family of reference point methods [10, 12] the value function is interpreted as an achievement scalarizing function, which measures the distance from a selected objective vector $\bar{\mathbf{z}}$, called *reference point*. The reference point specifies the desirable values of the objectives and it is usually provided by the DM. The distance from the reference point has a preferential meaning: the

tentative solution $\mathbf{x}^* \in \Omega$ showed to the DM is the solution minimizing the deviation from the reference point. In detail, the solution $\mathbf{x}^*$ is obtained by solving the following program:

$$\mathbf{x}^* = \min_{k=1\ldots m} \max_{k} [w_k(f_k(\mathbf{x}) - \bar{z}_k)] \qquad (2)$$
$$\text{subject to } \mathbf{x} \in \Omega$$

with weight $w_k > 0, k = 1\ldots m$. The achievement scalarizing function to minimize in Eq. 2 is the weighted Tchebychev distance from the reference point. The DM can express her bias for the k-*th* objective by assigning a value to weight w_k. After the DM has specified her desirable solution as a reference point, she can see what was feasible (the solution $\mathbf{x}^*$) and in case provide a new reference point. Many refinements and extensions of this approach exist [12]. They consider different ways of interaction with the DM (e.g., by showing a set of solution in the neighborhood of $\mathbf{x}^*$) and different refinements of the achievement scalarizing function, designed to obtain Pareto-optimal solutions with particular properties.

In principle, reference points approaches could be considered a natural way of accounting for preference drift: the DM is free to modify the reference point, exploring new regions of the Pareto front in response to a change in her preferences. However, the effort of the decision maker to modify the reference point when her preference model includes non-linear relations between the objectives may be prohibitive. The cognitive demands become unrealistic when the dimensionality of the problem increases, providing a large set of candidate directions to shift the reference point.

In the past, a number of works [14, 15, 8] introduced ML-based approaches to learn the user preferences in an interactive fashion. However, they have several limitations [1]. The approach in [14] does not guarantee the generation of Pareto optimal solutions, while the strategies developed in [15, 8] generate a linear local approximation of the user preferences and do not use directly the learned preference model to drive the search. Furthermore, in all these works the feedback from the DM is expressed in terms of quantitative scores.

The BC-EMO algorithm [1] overcomes these limitations, by learning the preference model with pairwise preference supervision, a much more affordable task for the DM, and by directly using the preference model to drive the search over the Pareto front. The algorithm does not make any assumption about the preference structure of the DM, possibly accounting for highly non-linear relations between the different objectives. This work extends BC-EMO to handle preference drift.

3 The BC-EMO algorithm

The goal of the BC-EMO algorithm consists of identifying the non-dominated solution preferred by the decision maker. To fulfill this scope, BC-EMO learns a value function from the preference information provided by the DM by using the support vector ranking [4], a supervised machine learning technique that learns

Algorithm 1 Training procedure at the generic i-*th* EMO iteration

---
1: **procedure** TRAIN(P_i, U_{i-1}, exa)
2:     $P_{tr} \leftarrow$ PREFORDER(P_i,U_{i-1},exa)
3:     obtain pairwise preferences for P_{tr} from the DM
4:     sort P_{tr} according to user preferences and add it to training instances
5:     Choose best kernel K and regularization C by k-fold cross validation
6:     $U_i \leftarrow$ function trained on full training set with K and C
7:     $res_i \leftarrow$ k-fold cv estimate of function performance
8:     **return** U_i, res_i
9: **end procedure**

---

to rank the input data. Training examples consist of pairwise comparisons of non-dominated solutions which are turned into ranking constraints for the learning algorithm. No specific assumptions are made about the form of the DM value function: BC-EMO has a tuning phase selecting the most appropriate kernel (i.e., similarity measure) in order to best approximate the targets, allowing it to learn an *arbitrary* value function provided enough data are available. Furthermore, support vector ranking allows to effectively deal with noisy training observations thanks to a regularization parameter C trading-off data fitting with complexity of the learned model.

The learned value function is used to rank the current population during the *selection* phase of the BC-EMO algorithm, where a sub-population is selected for reproduction on the basis of fitness (i.e., quality of the solutions). In particular, the BC-EMO selection procedure, which we will refer to as PREFORDER, consists of:

1. collecting the subset of non-dominated individuals in the population;
2. sorting them according to the learned value function;
3. appending to the sorted set the result of repeating the procedure on the remaining dominated individuals, until the desired number of individuals is reached.

The procedure is guaranteed to retain Pareto-optimality regardless of the form of the learned value function. Any evolutionary multi-objective algorithm (EMOA) that needs comparisons between candidate individuals can be equipped with the BC-EMO selection procedure (replacing or integrating the original selection procedure). Algorithm 1 describes the procedure of the generic i-*th* training iteration, in which: 1) the *exa* best individuals from the current population P_i are selected according to PREFORDER with current value function U_{i-1}; 2) DM feedback is collected for these examples; 3) parameter selection, training and evaluation are conducted on the training data enriched with P_{tr}. This procedure will be modified in the next section in order to account for preference drifts.

The overall BC-EMO approach consists of three steps:

1. *initial search phase*: the *plain* EMOA selected is run for a given number of generations and produces a final population P_1;

2. *training phase*: using P_1 as initial population, a specific number of training iterations are executed to learn the value function V by interacting with the DM. The final population obtained (P_2) is collected;
3. *final search phase*: the selected EMOA equipped with the BC-EMO selection procedure is run for a given number of generations, using P_2 as initial population and producing the final ordered population.

Each training iteration alternates a refinement phase, where the DM is queried for feedback on candidate solutions and the value function is updated according to such feedback, with a search phase, where the EMOA equipped with the BC-EMO selection procedure is run for a given number of iterations. The training phase is executed until the maximum number of training iterations or the desired accuracy level are reached.

The parameters of the BC-EMO algorithm 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_s). Algorithm 2 contains the pseudocode of the BC-EMO approach applied on top of a generic EMO algorithm. Further details on the algorithm can be found in [1].

Algorithm 2 The BC-EMO algorithm

1: **procedure** BC-EMO($maxit$, exa, gen_1, gen_s)
2:     $res \leftarrow 0$, $it \leftarrow 0$, $U \leftarrow$ RAND
3:     run the EMOA for gen_1 generations
4:     collect last population P
5:     **while** $it \leq maxit$ **do**
6:         $U, res \leftarrow$ TRAIN(P, U, exa)
7:         run the EMOA for gen_s generations guided PREFORDER with U
8:         collect last population P
9:     **end while**
10:    run the EMOA for the remaining generations guided PREFORDER with U
11:    **return** the final population P
12: **end procedure**

---

4 Handling preference drift with BC-EMO

The effect of preference drift is a decrease of the accuracy of the learnt model over time. In the original version of BC-EMO, training data arrives in batches over time and the model is re-trained every gen_s generations, when a new batch of training examples is available. The extension to handle preference drift consists of a mechanism for drift detection and of a reweighting of the past training examples inversely proportional to the observed decrease in the performance accuracy.

First, a cost in the range $[0, 1]$ is associated with each training example, initialized to the value one and defining the relevance of the example for the concept to predict. The detection of a drift in the preferences of the decision maker is based on the prediction accuracy of the learnt model. Let b_i and U_{i-1} the new batch of observable data and the current model at the generic i-*th* training iteration, respectively. The performance of the current model is the prediction accuracy $0 \leq res_{i-1} \leq 1$ over batch b_{i-1}. Furthermore, let $0 \leq res'_{i-1} \leq 1$ the prediction accuracy of the current model over batch b_i. If the difference between res_{i-1} and res'_{i-1} is bigger than a fixed threshold t_d, with $t_d > 0$, a drift in the preferences of the decision maker is assumed. In this case, the cost of the training examples collected so far (i.e., the training examples of batches $b_1, b_2, \ldots b_{i-1}$) is decreased as a function of the value $res_{i-1} - res'_{i-1}$. In detail, the cost is updated by a multiplicative factor $d = c(res_{i-1} - res'_{i-1})$, where the function c is defined as follows:

$$c(x) = \begin{cases} 1 & \text{if } x \leq t_d \\ 1 - x & \text{if } t_d < x < 0.5 \\ 0 & \text{if } x \geq 0.5 \end{cases} \quad (3)$$

Let us comment. If the value $res_{i-1} - res'_{i-1}$ is bigger than the threshold and smaller than 0.5, the cost of the training examples is decreased by the normalized value of the difference $res'_{i-1} - res_{i-1}$. When the decrease of the performance accuracy over the last batch of observable data is bigger than value 0.5, the training examples of the previous batches are discarded (i.e., their cost becomes zero). The rationale for this choice is that a large decrease in the accuracy of the learnt model is seen as symptom of a radical change in the preferences of the DM, outdating training examples collected in previous iterations.

If a drift in the preferences of the user has been detected, the model selection phase is executed using only the data in the i-*th* batch rather than using all the collected examples, as in the original version of BC-EMO (algorithm 1, line 6). Furthermore, the genetic population of the EMOA is reinitialized.

Let res_i the prediction accuracy of the selected model over batch b_i. If res_i is smaller than threshold t_r, the selected model is discarded as it does not satisfy the minimal performance requirement, and all training examples of of batches $b_1 \ldots b_{i-1}$ are discarded as well. The plain EMOA underlying BC-EMO will then be executed starting with a random population, until the next training iteration is reached. The rationale for this choice is the assumption that the poor performance of the selected model is caused by the collected training examples, localized in a region of the Pareto front that does not provide informative examples to learn the drift of the user preferences. The plain EMOA algorithm is executed to generate a population representing the whole Pareto front (without considering the preferences of the decision maker), in order to create more informative training examples at the next training iteration.

Algorithm 3 describes the modification of the training procedure at the generic i-*th* training iteration of BC-EMO to handle preference drift.

Algorithm 3 Training procedure to handle preference drift

1: **procedure** TRAIN(P_i, U_{i-1}, exa, res_{i-1}, t_d, t_r)
2:     $P_{tr} \leftarrow$ PREFORDER(P_i,U_{i-1},exa)
3:     obtain pairwise preferences for P_{tr} from the DM
4:     $b_i \leftarrow$ sort P_{tr} according to user preferences
5:     Add b_i to training instances
6:     $res'_{i-1} \leftarrow$ test U_{i-1} using b_i
7:     **if** $res_{i-1} - res'_{i-1} > t_d$ **then**
8:         Decrease costs of examples in $b_1 \ldots b_{i-1}$ according to (3) using t_d
9:         Re-initialize P_i randomly
10:     **end if**
11:     Choose best kernel K and regularization C by k-fold cross validation
12:     $res_i \leftarrow$ k-fold cv estimate of function performance
13:     **if** $res_i \geq t_r$ **then**
14:         $U_i \leftarrow$ function trained on full training set with K and C
15:     **else**
16:         $res_i \leftarrow 0, U_i \leftarrow$ RAND
17:     **end if**
18:     **return** U_i, res_i
19: **end procedure**

---

5   Experimental results

The experimental evaluation is focused on demonstrating the effectiveness of the extension of BC-EMO to handle decision maker preference drift for a selected case study. Given this focus, we did not attempt to fine-tune non-critical parameters which were fixed for the experiment.

Following [1], BC-EMO has been applied on top of NSGA-II [6] EMOA. We chose a population size of 100, 2000 generations, probability of crossover equal to one and probability of mutation equal to the inverse of the number of decision variables. Concerning the learning task of BC-EMO, the number of initial generations (gen_1) was set to 200, while the number of generations between two training iterations (gen_s) was set to 100. Both 5 and 10 examples per training iteration are tested. The minimum performance requirement threshold t_r was set to 0.5, while a decrease of the performance greater than 10% ($t_d = 0.1$) triggers the procedure handling the preference drift.

The case study consists of the bi-objective version of DTLZ6 problem, taken from popular DTLZ suite [7]:

$$\min_{\mathsf{x} \in \Omega} (\mathsf{x})$$
$$\Omega = \{\mathsf{x} \,|\, 0 \leq x_i \leq 1 \,\forall\, i = 1, \ldots, n\}$$
$$f_1(\mathsf{x}) = x_1, \ldots, f_{m-1}(\mathsf{x}) = x_{m-1},$$
$$f_m(\mathsf{x}) = (1 + g(\mathsf{x}_m))h(f_1, f_2, \ldots, f_{m-1}, g)$$
$$g(\mathsf{x}_m) = 1 + (9/|\mathsf{x}_m|) \sum_{x_i \in \mathsf{x}_m} x_i$$
$$h = m - \sum_{i=1}^{m-1} [(f_i/(1+g))(1 + \sin(3\pi f_i))]$$

This problem is characterized by an highly disconnected Pareto front, with both convex and concave regions (Fig. 1 (left)).
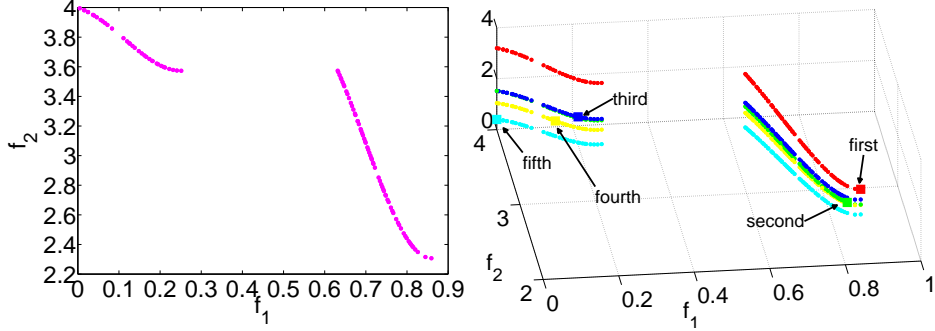

Fig. 1: Problem DTLZ6 with two objectives: (left) Pareto front for a sample run of plain NSGA-II without user preference; (right) preference values of the Pareto front according to the different values functions simulating the preference drift.

The drift in the preferences of the user is simulated by a sequence of five value functions:

1. $0.2 * f_1 + 0.8 * f_2$
2. $0.05 * f_2 * f_1 + 0.6 * f_1^2 + 0.38 * f_2$
3. $0.05 * f_2 * f_1 + 0.6 * f_1^2 + 0.38 * f_2 + 0.23 * f_1$
4. $0.05 * f_2 * f_1 + 0.68 * f_1^2 + 0.26 * f_2 + 0.23 * f_1$
5. $0.05 * f_2 * f_1 + 0.68 * f_1^2 + 0.1 * f_2 + 0.23 * f_1$

The sequence is generated by increasing the importance of the first objective (f_1) w.r.t. the second objective (f_2), assuming a non-linear formulation of the user preferences. This experimental setting simulates a decision maker that gradually becomes aware of the relation between her objectives to be optimized. Note that designing value functions which are non-monotonic in the Pareto front while retaining Pareto dominance properties is a non-trivial task. See [1] for a description of the generation process. Fig.1 (right) shows the different value functions considered, and their global minima over the Pareto front (square marked points). Tracking the shift of the global minimum between disconnected regions of the Pareto front is a challenging task for the optimization algorithm. The changes between the different value functions were fixed at generations 300, 600, 900 and 1200.

Fig. 2 and 3 report the results for the plain BC-EMO algorithm, for a baseline algorithm and for our BC-EMO extension, respectively, over the considered case study. The performance of the algorithms is measured in terms of percent approximation error w.r.t. the *gold standard* solution (y-axis) in function of the generation of the genetic population (x-axis). The gold standard solution is obtained by guiding the algorithm with the true value function. Each graph reports
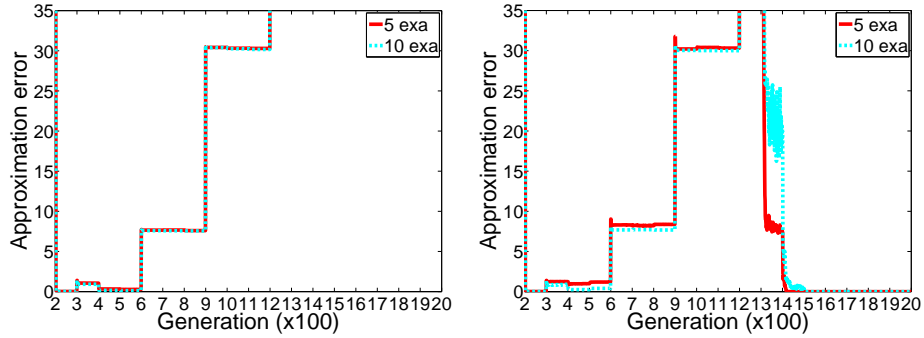
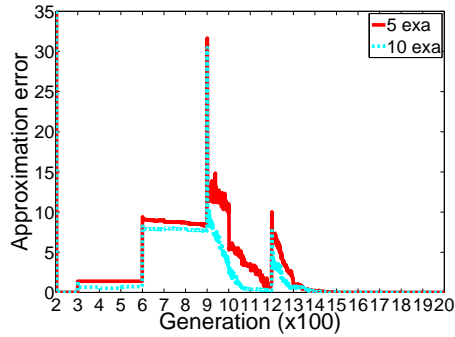Fig. 2: Performance of the BC-EMO algorithm (left) and of the baseline algorithm (right).

Fig. 3: Performance of the extension of BC-EMO to handle preference drift.

two learning curves for an increasing number of training examples per iteration (*exa*). Results are the medians over 500 runs with different random seeds for the search of the evolutionary algorithm.

At the generic i-*th* training iteration, the baseline algorithm retrains the learnt model using only the i-*th* batch of observable data. This is the only difference with the plain BC-EMO. Experimental results not reported here due to space limitations show the better performance obtained by discarding the previous training examples rather than discounting their cost by a fixed multiplicative factor in $(0, 1)$.

Fig. 2 (left) shows that the original version of BC-EMO cannot handle preference drift. The algorithm cannot track the changes of the user preferences: with the exception of the first drift, the performance of the algorithm keeps degrading each time the value function changes. After the last drift of the user preferences, the percent approximation error exceeds value 35%. This sub-optimal performance is caused by the lack of diversification during the search phase of the algorithm: the genetic population "gets trapped" in the region surrounding the global minima of the first and the second value functions.

A better performance is showed by the baseline algorithm (Fig. 2 (right)). Like BC-EMO, with 10 examples per iteration, a successful recover from the first drift is shown. The baseline algorithm fails to detect the second and the third changes of the value function (at generation 600 and 900, respectively): between generations 900 and 1200, the curves for both 5 and 10 training examples show a constant percentage deviation from the gold solution greater than 30%. When the fourth concept drift happens, the worst performance is observed. However, after generation 1400 the approximation error rapidly becomes zero, with both 5 and 10 training examples per iteration. Three iterations are required for perfect recovery from the fourth concept drift.

As expected, the best results are observed for the extension of BC-EMO designed for handling preference drift. Even if three training iteration are not enough for the perfect recovery from the second concept drift, the favourite solution of the decision maker generated by her third preference drift is perfectly identified. In the case of 10 training examples per iteration, an approximation error smaller than 1% is obtained at generation 1100. An even faster recovery is observed from fourth concept drift: two training iteration are required to approximate the new gold solution within an 1% approximation error. Note that, with the exception of the peak at generation 900 (corresponding to the third preference drift), the results tend to remain within 10% of the gold solution when 10 examples per iteration are provided.

6 Conclusion

This work addresses the problem of handling evolving preferences in interactive decision making. We modify BC-EMO, a recent multi-objective genetic algorithm based on pairwise preferences, by adapting its learning stage to learn under a concept drift. Our solution relies on the popular approach of instance weighting, in which the relative importance of examples is adjusted according to their predicted relevance for the current concept. We integrate these modifications with a diversification strategy favouring exploration as a response to changing DM preferences. Experimental results on a benchmark MOO problem with non-linear user preferences show the ability of the approach to early adapt to concept drifts.

Our promising preliminary results leave much room for future work. First, additional benchmark problems with evolving non-linear user preferences will be generated, possibly derived from real-world applications. Both sudden and gradual preference drift will be considered. Furthermore, active learning approaches could be devised in order to reduce the number of answers to the DM. This requires a shift of paradigm with respect to standard active learning strategies, in order to model the relevant areas of the optimization surface rather than reconstruct it entirely, and early detect and adapt to a changing surface.

References

1. Battiti, R., Passerini, A.: Brain-computer evolutionary multi-objective optimization (BC-EMO): a genetic algorithm adapting to the decision maker. IEEE Transactions on Evolutionary Computation (2010, to appear)
2. Belton, V., Branke, J., Eskelinen, P., Greco, S., Molina, J., Ruiz, F., Słowiński, R.: Interactive multiobjective optimization from a learning perspective. In: Multiobjective Optimization: Interactive and Evolutionary Approaches, pp. 405–433. Springer-Verlag, Berlin, Heidelberg (2008)
3. Campigotto, P., Passerini, A.: Adapting to a realistic decision maker: experiments towards a reactive multi-objective optimizer. In: LION IV: Learning and Intelligent OptimizatioN Conference, Venice, Italy, Jan 18-22, 2010. Lecture Notes in Computer Science, Springer Verlag (2010, to appear)
4. Collins, M., Duffy, N.: Convolution kernels for natural language. In: Advances in Neural Information Processing Systems 14. pp. 625–632. MIT Press (2001)
5. Deb, K.: Multi-objective optimization using evolutionary algorithms. Wiley (2001)
6. Deb, K., Pratap, A., Agarwal, S., Meyarivan, T.: A fast elitist multi-objective genetic algorithm: NSGA-II. IEEE Transactions on Evolutionary Computation 6, 182–197 (2000)
7. Deb, K., Thiele, L., Laumanns, M., Zitzler, E.: Scalable multi-objective optimization test problems. In: Congress on Evolutionary Computation (CEC2002). pp. 825–830 (2002)
8. Huang, H.Z., Tian, Z.G., Zuo, M.J.: Intelligent interactive multiobjective optimization method and its application to reliability optimization. IIE Transactions 37(11), 983–993 (2005)
9. Klinkenberg, R., Rüping, S.: Concept drift and the importance of examples. In: Text Mining Theoretical Aspects and Applications. pp. 55–77. Physica-Verlag (2002)
10. Miettinen, K.: Nonlinear Multiobjective Optimization, International Series in Operations Research and Management Science, vol. 12. Kluwer Academic Publishers, Dordrecht (1999)
11. Miettinen, K., Ruiz, F., Wierzbicki, A.: Introduction to Multiobjective Optimization: Interactive Approaches. In: Multiobjective Optimization: Interactive and Evolutionary Approaches, pp. 27–57. Springer-Verlag Berlin, Heidelberg (2008)
12. Miettinen, K., Mäkelä, M.M.: On scalarizing functions in multiobjective optimization. OR Spectrum 24, 193–213 (2002)
13. Schlimmer, J., Granger, R.: Incremental learning from noisy data. Mach. Learn. 1(3), 317–354 (1986)
14. Sun, M., Stam, A., Steuer, R.: Solving multiple objective programming problems using feed-forward artificial neural networks: the interactive ffann procedure. Manage. Sci. 42(6), 835–849 (1996)
15. Sun, M., Stam, A., Steuer, R.: Interactive multiple objective programming using tchebycheff programs and artificial neural networks. Comput. Oper. Res. 27(7-8), 601–620 (2000)
16. Žliobaité, I.: Learning under concept drift: an overview. Tech. rep., Vilnius University, Faculty of Mathematics and Informatics (2009)