PhD Dissertation

International Doctorate School in Information and Communication Technologies

DISI - University of Trento

A Reactive Search Optimization approach to interactive decision making

Paolo Campigotto

Advisor:

Prof. Roberto Battiti

Università degli Studi di Trento

December 2010

Abstract

Reactive Search Optimization (RSO) advocates the integration of learning techniques into search heuristics for solving complex optimization problems. In the last few years, RSO has been mostly employed in self-adapting a local search method in a manner depending on the previous history of the search. The learning signals consisted of data about the structural characteristics of the instance collected while the algorithm is running. For example, data about sizes of basins of attraction, entrapment of trajectories, repetitions of previously visited configurations. In this context, the algorithm learns by interacting from a previously unknown environment given by an existing (and fixed) problem definition.

This thesis considers a second interesting online learning loop, where the source of learning signals is the decision maker, who is fine-tuning her preferences (formalized as an utility function) based on a learning process triggered by the presentation of tentative solutions. The objective function and, more in general, the problem definition is not fully stated at the beginning and needs to be refined during the search for a satisfying solution. In practice, this lack of complete knowledge may occur for different reasons: insufficient or costly knowledge elicitation, soft constraints which are in the mind of the decision maker, revision of preferences after becoming aware of some possible solutions, etc.

The work developed in the thesis can be classified within the well known *paradigm of Interactive Decision Making (IDM). In particular, it considers interactive optimization from a machine learning perspective, where IDM is seen as a joint learning process involving the optimization component and the DM herself. During the interactive process, on one hand, the decision maker improves her knowledge about the problem in question and, on the other hand, the preference model learnt by the optimization component evolves in response to the additional information provided by the user. We believe that understanding the interplay between these two learning processes is essential to improve the design of interactive decision making systems. This thesis goes in this direction, 1) by considering a final user that may change her preferences as a result of the deeper knowledge of the problem and that may occasionally provide inconsistent feedback during the interactive process, 2) by introducing a couple of IDM techniques that can learn an arbitrary preference model in these changing and noisy conditions. The investigation is performed within two different problems settings, the traditional multi-objective optimization and a constraint-based formulation for the DM preferences.*

In both cases, the ultimate goal of the IDM algorithm developed is the identification of the solution preferred by the final user. This task is accomplished by alternating a learning phase generating an approximated model of the user preferences with an optimization stage identifying the optimizers of the current model. Current tentative solutions will be evaluated by the final user, in order to provide additional training data. However, the cognitive limitations of the user while analyzing the tentative solutions demands to minimize the amount of elicited information. This requires a shift of paradigm with respect to standard machine learning strategies, in order to model the relevant areas of the optimization surface rather than reconstruct it entirely. In our approach the shift is obtained both by the application of well known active learning principles during the learning phase *and by the suitable trade-off among diversification and intensification of the search during the optimization stage.*

Keywords
Interactive decision making, reactive search optimization, multi-objective optimization, preference elicitation, machine learning, concept drift, satisfiability modulo theory.

Contents

Acknowledgements

Come inizio della tesi, permettetemi di ringraziare tutti Coloro che hanno contribuito a renderla possibile. Per primo, il mio advisor Roberto Battiti che mi ha dato l'opportunita' di intraprendere un Dottorato nel suo gruppo di Machine Learning and Intelligent Optimization e ha corretto tutto quanto scritto in queste pagine (tranne questi ringraziamenti!). Oltre a Roberto, i risultati presentati in questa tesi sono da condividere con Andrea Passerini, che non ha lesinato consigli, correzioni e proposte determinanti per realizzare questo lavoro. Non potrei non citare Franco Mascia ed Elisa Cilia, che hanno condiviso con me gli anni del Dottorato. Un ringraziamento particolare anche a Mauro Brunato, per i suoi "consigli scientifici" e non solo (tuttora custodisco gelosamente l'unico esemplare mondiale di bottiglia di grappa "SAT-customized"). Scrivendo queste pagine, mi piace ricordare tutte le persone che hanno gravitato attorno al laboratorio LION in questi anni. Infine, condivido la felicita' per il traguardo raggiunto anche con la mia famiglia, la mia ragazza ed i miei amici.

Chapter 1

Introduction

In many decision making problems, the crucial issue is not that of delivering a single solution, but that of critically analyzing a mass of tentative solutions, which can easily grow up to thousands or millions, to identify the solution preferred by the final user. Delivering to the decision maker (DM) the entire set of the tentative solutions so that the user can pick her most preferred solution is impractical, due to the prohibitive effort required to the DM.

In principle, this laborious selection among the large set of candidate solutions could be avoided by including in the initial formulation of the problem the specification of the utility criterion of the DM. However, requiring a human DM to pre-specify her preferences, without seeing any actual optimization result, is extremely difficult. In typical decision making problems the preferences of the DM cannot be fully defined at the beginning and needs to be learnt and refined during the search for a satisfying solution. This lack of complete knowledge can occur for different reasons: insufficient or costly knowledge elicitation, soft constraints which are in the mind of the decision maker, revision of the preferences after becoming aware of some possible solutions, etc.

Interactive decision making methods (IDM) overcome these difficulties by keeping the user in the loop of the optimization process. They use preference information from the decision maker during the optimization task to guide the search towards her favourite solution. This thesis introduces two new interactive decision making techniques. With the extent of identifying the favourite solution of the decision maker, they learn and optimize an approximation of the preference model of the final user.

One technique is developed within the context of the traditional multi-objective optimization problem, where the user searches for her favourite solution within the Pareto-optimal set. Our approach considers the limited and bounded rationality of the humans when making decisions: it accounts for noisy and inconsistent feedback from the user and can handle a DM preference model that changes over time. The incomplete knowledge about the problem to be solved is formalized by assuming the knowledge of a set of desirable objectives, and ignorance of their detailed combination. To the best of our knowledge, in the context of multi-objective optimization no interactive decision making technique has been explicitly designed to handle a preference model that evolves over time. This work aims at covering this gap, by introducing also a representative multi-objective problem with evolving user preferences.

The alternative problem setting considered in this thesis consists of a constraint-based formulation of the DM preference model. The preferences are expressed in terms of soft constraints, with each constraint the conjunction of decisional features of the DM. The utility function of the user is modeled by the weighted sum of the constraints. This formulation is a natural way to express preferences in many real world applications. Consider, e.g., a user selecting a house among a set of candidates, a customer judging the value of a car or assessing her interests to a movie. This thesis introduces an interactive optimization procedure alternating a learning phase with an optimization stage. In detail, it iteratively learns an utility function modeling the quality of the candidate solutions and uses it to generate novel candidates for the following refinement. The learning stage exploits the sparsity-inducing property of 1-norm regularization to learn a combinatorial function from the power set of all possible conjunctions up to a certain degree. The optimization stage uses a satisfiability modulo theories solver, which enables the definition of a general approach for a large class of optimization problems.

1.1 Multi-objective optimization formulation

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 terms *multiple criteria decision making* or *multi-objective optimization* refers to solving these problems. A multi-objective optimization problem (MOOP) can be stated as:

$$\begin{aligned} \text{minimize} \quad & \mathbf{f}(\mathbf{x}) = \{f_1(\mathbf{x}), \dots, f_m(\mathbf{x})\} \\ \text{subject to} \quad & \mathbf{x} \in \Omega \end{aligned} \tag{1.1}$$

where $\mathbf{x} \in \mathbb{R}^n$ is a vector of n decision variables; $\Omega \subset \mathbb{R}^n$ is the *feasible region* and is typically specified as a set of constraints on the decision variables; $\mathbf{f} : \Omega \to \mathbb{R}^m$ is a vector of m objective functions which need to be jointly minimized. Objective vectors are images of decision vectors and can be written as $\mathbf{z} = \mathbf{f}(\mathbf{x}) = \{f_1(\mathbf{x}), \dots, f_m(\mathbf{x})\}$. Problem 1.1 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[1] is called *Pareto set* (PS). The corresponding set of Pareto-optimal objective vectors is called *Pareto front* (PF).

1.2 Motivation of the thesis

The centrality of the decision maker is widely recognized in the multiple criteria decision making community. However, in the experiments considered in IDM literature the user preferences are usually formalized into a mathematical model, with the extent of representing the qualitative notion of preference as a quantitative function, while retaining the Pareto-optimality properties. This mathematical model emulates the decision maker in the interactive optimization process. The preference model is usually represented by the linear combination of the objectives or it is expressed as a function of the distance from the ideal point. In the first case, the model is formalized into a function $U(\mathbf{z})$ as follows:

$$U(\mathbf{z}) = \sum_{k=1}^{m} w_k z_k \tag{1.2}$$

where the (positive) weights encode the relative importance of the different objectives. In the second case, the utility function is defined by a weighted-L_p metric of the following form:

$$U(\mathbf{z}) = -\left(\sum_{k=1}^{m} w_k |z_k^* - z_k|^p\right)^{1/p} \tag{1.3}$$

[1]In the multi-objective optimization literature, these trade-off solutions are known by means of different synonyms: efficient, nondominated, noninferior or Pareto-optimal solutions.

in which $\mathbf{z}^*$ is a reference ideal objective vector obtained by separately maximizing each objective function subject to the feasible region, i.e., $z_k^* = \max_{\mathbf{x}\in\Omega} f_k(\mathbf{x})$.

The utility functions in Eq. 1.2 and 1.3 are used to simulate the feedback of a real user during the interactive process. That is, the preference of the user for the solution $\mathbf{z}'$ is expressed by the value $U(\mathbf{z}')$. This approach has several limitations:

1. the linear weighting scheme in Eq. 1.2 cannot model the typical human decision making process occurring in many real life situations. When a strong non-linear relation correlates the different objectives, the most intuitive approach of giving highest weight to the most important criterion can lead to completely unsatisfactory solutions [41, 45]. In many decision making situations, assuming that satisfaction increases linearly with the decrease of the objective functions is inappropriate.

   Even the generalization in Eq. 1.3 of the linear utility function cannot model the nonlinear preference of "compromise" solutions, which characterizes many human decision activities [4];

2. an *error-free* preference structure of the DM is assumed. However, imprecisions and contradictions characterize most human decision processes. As result, uncertain and inconsistent feedback is often observed in real decision making problems;

3. a *static* preference model for the DM is considered. The preference structure is modeled by a function *specified a priori* (Eq.1.2 and 1.3), which remain fixed during the interactive optimization process. This is rather unrealistic in many concrete applications, where the DM has only limited initial knowledge of the problem at hand. In many cases, only when the DM sees the actual tentative solutions, she becomes

aware of "what is possible". Confronted with this new knowledge, her preferences may evolve over time. 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. As a results, her judgment changes over time.

According to [25], the number of real world applications of the optimization techniques developed by the multi-criteria decision making community is modest. The reason for this failure is the "high complexity of the methods as perceived by real decision makers" [25]. As matter of fact, the typical decision maker is not necessarily an expert in algorithmic and mathematical details, but she is a user who needs a fast and simple way of navigating among the set of the Pareto-optimal solutions, guided by her preferences. These observations motivate the development of a robust preference elicitation phase in IDM techniques, enabling the user to express her preferences in a simple way, to change them over time and accounting for inconsistent feedback from the DM.

1.2.1 Preferences as soft constraints

The importance of learning the preference of the DM is not limited to the multi-objective optimization research community. In the last few years, the preference elicitation problem has been investigated in the context of different disciplines, including machine learning and constraint programming. In the machine learning (ML) community, the task of learning and predicting preferences in an automatic way is known as *preference learning* [20]. As notable applications, consider, e.g., Web search engines and recommender systems. Very recent research in the field of constraint programming [21] defines the preferences of the DM in terms of *soft* constraints and introduce constraint optimization problems where the data are not completely known before the solving process starts. In soft constraints, a generalization of hard constraints, each assignment to the variables of the constraint is associated to a preference value taken from a preference set. The preference value represents the level of desirability of the assignment. The desirability of a complete assignment is computed by applying a combination operator to the local preference values. Thus, a set of soft constraints generates an order (partial or total) over the complete assignments of the variables of the problem. Given two solutions of the problem, the preferred one is selected by computing their preference levels and by comparing them in the preference order. The work in [21] introduces an elicitation strategy for soft constraint problems with missing preferences, to find the solution preferred by the decision maker asking the final user to reveal as few preferences as possible. Soft constraints are modeled by a general framework that can unify previous extensions of the constraint satisfaction formalism (e.g., weighted or fuzzy constraint satisfaction problems). The optimality of the solutions produced is guaranteed and the empirical studies in [21] show that on fuzzy constraint satisfaction problems with missing preferences the algorithm can also provide a solution at any point in time, whose quality increases with the computation time (anytime property).

However, the work in in [21] has several limitations and open issues:

- it does not consider the inconsistent and imprecise preference information from the DM characterizing many human decision processes;
- it assumes initial complete knowledge of both the decisional features of the DM and their detailed combination (represented in terms of soft constraints). The elicitation process focuses exclusively on assessing the weights of the soft constraints;
- it expresses the preference information by quantitative judgments about

assignments to the variables of a specific constraint. However, asking to the final user precise scores is in many cases inappropriate or even impossible. Most of the users are typically more confident in comparing solutions, providing qualitative judgments like "I prefer solution A to solution B", rather than in specifying how much they prefer A over B;

- it models just negative preferences, i.e., the final user can express just different degrees of unsatisfaction for the solutions. In many real life problems, the interaction with the final users is naturally modeled by specifying what she likes and what she dislikes, reflecting the typical human behavior, where the degree of preference for a solution is defined by comparing its advantages with its disadvantages;
- it combines branch and bound search with preference elicitation, the adoption of local search algorithms is a matter of current research, as pointed out by the authors themselves.

In this thesis, we introduce a technique that can solve the above issues, testing its performance over a couple of realistic decision problems.

1.3 Contribution of the thesis

This thesis tackles the problem of learning the user preferences in the context of interactive decision making. In particular, we formalize the preference learning problem within two settings:

1. the traditional multi-objective optimization formulation;
2. a constraint-based formulation modeling the DM preferences.

In both cases, we introduce a novel technique based on machine learning. The adoption of machine learning enables a robust approach, handling contradictory and inconsistent feedback from the decision maker as well as a dynamic preference model of the final user.

1.3.1 Contribution in interactive multi-objective optimization

Concerning the traditional MOOP, the contribution of this thesis consists of a new technique that can handle unforeseen changes in the preferences of the decision maker. Real world optimization tasks are often characterized by noisy and changing conditions. The problem of learning in changing conditions is known in the machine learning community as learning under *concept drift* [39]. The problem has received increasing attention in past few years, and a number of solutions have been proposed to tackle it. For a review of the recent approaches in this area, see [51]. In this thesis 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, the current IMO techniques usually consider a static preference model for the DM: no IMO technique has been explicitly designed to handle preference drift. Among the plethora of IMO algorithms, reference point methods [29, 30], 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 objectives.

Machine learning techniques [43, 44, 23] have been employed in IMO 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. The recent Brain-Computer Evolutionary Multi-Objective Optimization (BC-EMO) algorithm [4] overcomes 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. Fast convergence of the algorithm to the desired solution was shown [4] 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 [8]. We thus selected BC-EMO as a natural candidate to be extended for managing preference drift [9].

The extension of BC-EMO for preference drift recovery is based on the approach of instance weighting [27], 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 utility function of the DM.

An additional contribution of this thesis consists of a benchmark problem simulating the noisy and changing conditions occurring during real world optimization tasks. A real user cannot be emulated via the static error-free utility function defined in Eq. 1.2 and 1.3. The limited rationality of people when making decisions motivates the need for a more realistic simulation of the seamless human-computer interaction characterizing IMO techniques. Furthermore, solving real MOOPs requires the ability to search over complex Pareto fronts. These observations translate into the definition of benchmark problems characterized by:

1. highly complex Pareto front, including concave, convex and disconnected regions;
2. arbitrary DM preference model;
3. uncertain, inconsistent and contradictory feedback from the final user;
4. unforeseen changes in the preferences of the decision maker.

Uncertain preference information is modeled by considering occasional inattention of the DM or her embarrassment when required to compare too similar solutions. Evaluating the ability of the proposed preference model to simulate the human decision process is outside the scope of this work. The generation of benchmark problems with the above characteristics provides test cases for the extended version of BC-EMO, and, more in general, may help to identify the MOO algorithms that are more promising for future testing on real-life scenarios.

1.3.2 Constraint-based formulation

In many real-life problems, preferences can be naturally expressed as soft constraints. Given the set of soft constraints, the aim consists of finding a solution optimizing them. This means that there is an utility function measuring the quality of the candidate solutions in terms of preferences.

Consider, for example, a house sale system suggesting candidate houses according to their characteristics, such as "the kitchen is roomy", "the house has a garden","the neighbourhood is quiet". The task can be formalized as a weighted MAX-SAT problem, where the constraints are encoded by Boolean terms, with each term the combination of Boolean features. The preferences of the final user are expressed by the weighted sum of the constraints, with the weights defining the relative importance of the constraints.

In the setting we consider here the combinatorial utility function expressing the DM preference model is the weighted combination of Boolean terms. However, it is unknown and has to be jointly and interactively learned during the optimization process. Note that the optimal utility function is *complex* enough to prevent exhaustive enumeration of possible solutions.

Our method consists of an iterative procedure alternating a search phase with a model refinement phase. At each step, the current approximation of the utility function is used to guide the search for optimal configurations; preference information is required for a subset of the recovered candidates, and the utility model is refined according to the feedback received. A set of randomly generated examples is employed to initialize the utility model at the first iteration.

We show how to generalize the proposed method to more complex utility functions which are combinations of *predicates* in a certain theory of interest. A standard setting is for example that of scheduling, where solutions could be starting times for each job, predicates define time constraints for related jobs, and weights specify costs paid for not satisfying a certain set of constraints. The generalization basically consists of replacing satisfiability with satisfiability modulo theories [1] (SMT). SMT is a powerful formalism combining first-order logic formulas and theories providing interpretations for the symbols involved, like the theory of arithmetic for dealing with integer or real numbers. It is receiving increasing attention in recent years, thanks to a number of successful applications in areas like verification systems, planning and model checking. Optimization modulo theories, also known as "Satisfiability modulo the theory of costs" [13], extends SMT by considering optimization problems. Rather than checking the existence of a satisfying assignment as in SMT, the target is a satisfying assignment that minimizes a given cost function. Optimization modulo theories is a very recent research field; this is the first work combining learning, interactive optimization and SMT. Therefore, to test our IDM technique we encoded different optimization tasks as weighted MAX-SMT optimization problems.

Considering our critique to the soft constraint-based approach in [21], our technique offers the following advantages:

- it does not assume to know in advance the decisional features of the user and their detailed combination. It can select the variables of the learning problem from a set of "catalog" features;
- it can handle noisy and inconsistent feedback from the user;
- it may adapt to both qualitative and quantitative evaluations from the DM, by asking the comparison of solutions rather than the assignment of preference degrees in terms of scores from a predefined range;
- in the case of quantitative evaluations from the DM, it allows the user to state both negative or positive judgments for the provided solutions, including the possibility to express "indifference";
- both complete and local search techniques may be adopted to optimize the learnt preference model.

On the other hand, our approach cannot guarantee the optimality of the retrieved solutions. However, the experimental results on both weighted MAX-SAT and MAX-SMT problems demonstrate the effectiveness of our technique in focusing towards the optimal solutions, its robustness as well as its ability to recover from suboptimal initial choices.

1.4 Outline of the thesis

The remainder of the thesis is organized as follows. Chapter 2 presents our approach in the context of multi-objective optimization. First, the limitations of current IDM techniques in regard to preference drift handling are discussed. Then, the BC-EMO algorithm is reviewed and extended to automatically handle preference drift. The constraint-based formulation of the user preferences is tackled in Chapter 3. The preference structure of the DM is expressed as a combinatorial optimization function and the elicitation task is solved by combining active learning of combinatorial features with the optimization of learnt utility function models. The comparison of our method with recent preference elicitation approaches developed in the context of constraint programming [21, 28, 6] is discussed. As no established real world benchmarks are available at the time of this writing, a couple of realistic problems is defined and included in experimental evaluation of the proposed technique. Chapter 4 draws some conclusions, by summarizing the results and the contribution of the thesis and by proposing possible directions for future research.

Chapter 2

Handling preference drift in interactive decision making

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. The work in [9] 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.

2.1 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 [37, 36]. Only when the DM sees 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 [5].

In the machine learning (ML) community, the problem of learning in these changing conditions is known as learning under *concept drift* [39]. The work in [51] reviews the recent approaches for *concept drift* recovery. In this thesis we consider *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 [29, 30], which iteratively minimize the distance to ideal reference points provided by the DM, could in principle naturally handle preference drifts. However, the cognitive effort of the DM can easily become prohibitive, especially when dealing with non-linear preference models and an increasing number of objective.

Different IDM approaches [43, 44, 23] employ machine learning techniques to learn the user preferences and can be easily adapted to tackle preference drifts. However, these approaches cannot guarantee the generation of Pareto optimal solutions or assume a linear set of weights, one for each objective. A recent genetic technique, the Brain-Computer Evolutionary Multi-Objective Optimization (BC-EMO) algorithm [4], overcomes these limitations.

BC-EMO represents the preference information of the decision maker in terms of a value function. Based on the DM evaluation of current tentative solutions, the predicted value function is refined, and it is used to modify the fitness measure of the genetic algorithm. The convergence of the algorithm to the desired solution on both combinatorial and continuous problems with linear and non-linear value functions has been tested [4]. Robustness to inaccurate and contradictory DM feedback [8] is obtained by the adoption of a support vector ranking technique in the learning stage of BC-EMO. 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 exploits the instance weighting scheme [27] from the concept drift literature, which reweights the examples according to their predicted relevance for the current concept. The activation of the reweighting mechanism is triggered by a change detection monitor. Furthermore, in order to deal with concept drift in the specific setting of interactive optimization, a diversification strategy aimed at escaping from minima which could become suboptimal for the changed preference of the DM is also introduced.

The remainder of the chapter is organized as follows. Section 2.2 introduces IDM and discusses the limitations of current techniques in regard to preference drift handling. Section 2.3 briefly reviews the BC-EMO algorithm, while Section 2.4 extends it to automatically handle preference drift. An experimental evaluation of the proposed extension is reported in Section 2.5. Section 2.6 draws some conclusions and proposes possible directions for future research.

2.2 Interactive decision making techniques

Several IDM approaches have been developed to aid the DM in identifying her preferred solution [31], including evolutionary multi-objective algorithms (see for example [15] 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 [29, 30] 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:

$$\begin{aligned} \mathbf{x}^* = \quad & \min \max_{k=1\ldots m} [w_k(f_k(\mathbf{x}) - \bar{z}_k)] \\ & \text{subject to } \mathbf{x} \in \Omega \end{aligned} \tag{2.1}$$

with weight $w_k > 0, k = 1 \ldots m$. The achievement scalarizing function to minimize in Eq. 2.1 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. Let us emphasize the rationale of this approach:

1. the location of the reference point causes the procedure to focus on a certain region in the Pareto front;
2. a local approximation of the preference model is expressed by the distance function from the reference point. Using the weighted Tchebychev distance metric, every solution of the Pareto front can be obtained by altering the reference point only [48].

Many refinements and extensions of this approach exist [30]. 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 [43, 44, 23] introduced ML-based approaches to learn the user preferences in an interactive fashion. However, they have several limitations [4]. The approach in [43] does not guarantee the generation of Pareto optimal solutions, while the strategies developed in [44, 23] 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 [4] 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.

2.3 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 [14], a supervised machine learning technique that learns to rank the input data. Training examples consist of pairwise comparisons of non-dominated solutions which are turned into

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
```

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 [4].

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
```

2.4 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:

$$c(x) = \begin{cases} 1 & \text{if } x \leq t_d \\ 1 - x & \text{if } t_d < x < 0.5 \\ 0 & \text{if } x \geq 0.5 \end{cases} \tag{2.2}$$

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 $b_1 \dots b_{i-1}$ 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.

2.5 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 [4], BC-EMO has been applied on top of NSGA-II [16] 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

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 $b_1 \dots b_{i-1}$ according to (2.2) 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
```

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 [17]:

$$\begin{aligned}
&\min\nolimits_{\times\in\Omega}(\times)\\
&\Omega = \{\times \,|\, 0 \leq x_i \leq 1 \,\forall\, i = 1, \dots, n\}\\
&f_1(\times) = x_1, \dots, f_{m-1}(\times) = x_{m-1},\\
&f_m(\times) = (1 + g(\times_m))h(f_1, f_2, \dots, f_{m-1}, g)\\
&g(\times_m) = 1 + (9/|\times_m|) \textstyle\sum_{x_i\in\times_m} x_i\\
&h = m - \textstyle\sum_{i=1}^{m-1} [(f_i/(1+g))(1 + \sin(3\pi f_i))]
\end{aligned}$$

This problem is characterized by a highly disconnected Pareto front, with both convex and concave regions (Fig. 2.1 (left)).

The drift in the preferences of the user is simulated by a sequence of

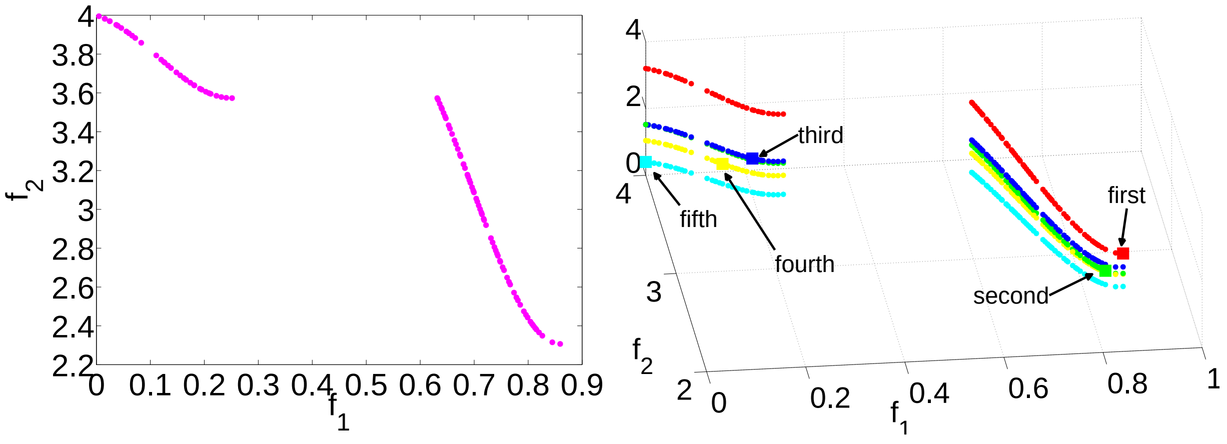

Figure 2.1: Problem DTLZ6 with two objectives: (left) Pareto front for a sample run of plain NSGA-II without user preference; (right) preference values of the Pareto front according to the different values functions simulating the preference drift.

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 [4] for a description of the generation process. Fig.2.1 (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. 2.2 and 2.3 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

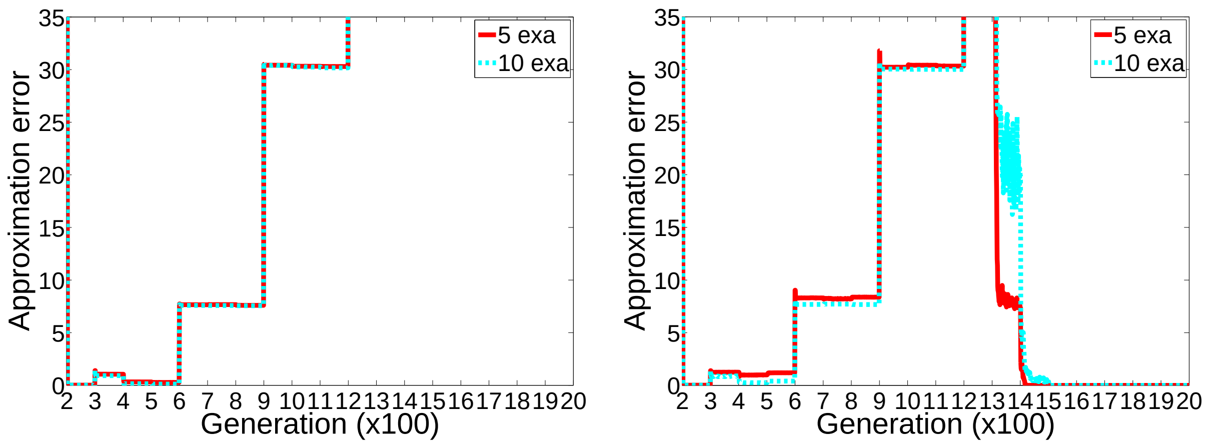

Figure 2.2: Performance of the BC-EMO algorithm (left) and of the baseline algorithm (right).

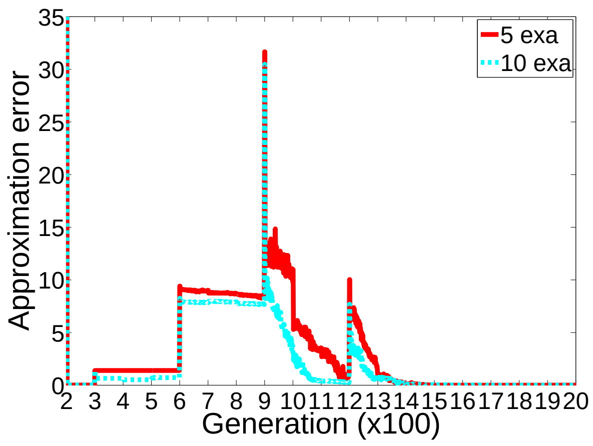

Figure 2.3: Performance of the extension of BC-EMO to handle preference drift.

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 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 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. 2.2 (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. 2.2 (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 the 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.

2.6 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,

more sophisticated active learning approaches could be devised in order to reduce the number of queries 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.

Chapter 3

Active Learning of Combinatorial Features for Interactive Optimization

In this chapter, based on the work in [10], we address again the problem of automated discovery of preferred solutions by an interactive optimization procedure. However, rather than considering the traditional multi-objective optimization paradigm, we focus on combinatorial utility functions made of weighted conjunctions of Boolean variables. Our approach resorts again to the “learning to optimize” framework, where a utility function modeling the quality of candidate solutions is iteratively learnt and used to generate novel candidates for the following refinement. In the proposed algorithm, the learning stage exploits the sparsity-inducing property of 1-norm regularization to learn a combinatorial function from the power set of all possible conjunctions up to a certain degree. The optimization stage uses a stochastic local search method to solve a weighted MAX-SAT problem. We show how the proposed approach generalizes to a large class of optimization problems dealing with satisfiability modulo theories. Experimental results demonstrate the effectiveness of the approach in focusing towards the optimal solution and its ability to recover from suboptimal initial choices.

3.1 Introduction

The field of combinatorial optimization focussed in the past mostly on solving well defined problems, where the function $f(x)$ to optimize is given, either in a closed form, or as a simulator which can be interrogated to deliver f values corresponding to inputs, possibly with some noise leading to stochastic optimization. One therefore distinguishes two separated phases, a first one related to defining the problem through appropriate consulting, knowledge elicitation, modeling steps, and a second one dedicated to solving the problem either optimally, in the few cases when this is possible, or approximately, in most real-world cases leading to NP-hard problems.

Unfortunately the above picture is not realistic in many application scenarios, where *learning* about the problem definition goes hand in hand with delivering a set of solutions of improving quality, as judged by a decision maker (DM) responsible for selecting the final solution. In particular, this holds in the context of multi-objective optimization, where one aims at maximizing at the same time a set of functions f_1, ..., f_n. Multi-objective optimization, when cast in the language of machine learning, is a paradigmatic case of lack of information, where only some relevant building blocks (*features*) are initially given as the individual function f_i's, but their combination into a *utility function* modeling the preferences of the DM is not given and has to be learnt by interacting with the DM [7]. Dealing with human DM, characterized by limited patience and bounded rationality, demands for some form of strategic production of candidates to be evaluated (query learning), and requires to account for the possible mistakes and dynamical evolution of her preferences (learning about concrete possibilities may lead somebody to change his/her initial objectives and evaluations). A further complication is related to the difficulty of delivering quantitative judgments by the DM, who is often better off in ranking possibilities more than in delivering utility values. The interplay of optimization and machine learning has been advocated in the past for example in the Reactive Search Optimization (RSO) context, see [2] also for an updated bibliography and [3] for an application of RSO in the context of multi-objective optimization.

In this work, we focus on a setting in which the optimal utility function is both *unknown* and *complex* enough to prevent exhaustive enumeration of possible solutions. We start by considering combinatorial utility functions expressed as weighted combinations of terms, each term being a conjunction of Boolean features. A typical scenario would be a house sale system suggesting candidate houses according to their characteristics, such as "the kitchen is roomy", "the house has a garden","the neighbourhood is quiet". The task can be formalized as a weighted MAX-SAT problem, a well-known formalization which allows to model a large number of real-world optimization problems. However, in the setting we consider here the underlying utility function is unknown and has to be jointly and interactively learned during the optimization process.

Our method consists of an iterative procedure alternating a search phase and a model refinement phase. At each step, the current approximation of the utility function is used to guide the search for optimal configurations; preference information is required for a subset of the recovered candidates, and the utility model is refined according to the feedback received. A set of randomly generated examples is employed to initialize the utility model at the first iteration.

We show how to generalize the proposed method to more complex utility functions which are combinations of *predicates* in a certain theory of interest. A standard setting is that of scheduling, where solutions could be starting times for each job, predicates define time constraints for related jobs, and weights specify costs paid for not satisfying a certain set of constraints. The generalization basically consists of replacing satisfiability with satisfiability modulo theory [1] (SMT). SMT is a powerful formalism combining first-order logic formulas and theories providing interpretations for the symbols involved, like the theory of arithmetic for dealing with integer or real numbers. It has received consistently increasing attention in recent years, thanks to a number of successful applications in areas like verification systems, planning and model checking.

Experimental results on both weighted MAX-SAT and MAX-SMT problems demonstrate the effectiveness of our approach in focusing towards the optimal solutions, its robustness as well as its ability to recover from sub-optimal initial choices.

The chapter is organized as follows: Section 3.2 introduces the algorithm for the SAT case. Section 3.3 introduces SMT and its weighted generalization and shows how to adapt our algorithm to this setting. Related works are discussed in Section 3.4. Section 3.5 reports the experimental evaluation for both SAT and SMT problems. A discussion including potential research directions concludes the chapter.

3.2 Overview of our approach

Candidate configurations are n dimensional Boolean vectors $\mathbf{x}$ consisting of *catalog* features. The only assumption we make on the utility function is its sparsity, both in the number of features (from the whole set of catalog ones) and in the number of terms constructed from them. We rely on this assumption in designing our optimization algorithm.

The candidate solutions are obtained by applying a stochastic local search (SLS) algorithm that searches the Boolean vectors maximizing the weighted sum of the terms of the learnt utility model. At each iteration, the algorithm chooses between a random and a greedy move with probability

wp and $1 - wp$, respectively. A greedy move consists of flipping one of the variables leading to the maximum increase in the sum of the weights of the satisfied terms (if improving moves are not available, the least worsening move is accepted). The main difference w.r.t the "standard" weighted SLS algorithms consists of the DNF rather than CNF representation, which we believe to be a more natural choice when modeling combined effects of multiple non-linearly related features. Since switching from disjunctive to conjunctive normal form representations may involve an exponential increase in the size of the Boolean formula, we implemented a method that operates on formulae represented as a weighted linear sum of terms.

The candidate solutions generated by the optimizer during the search phase are first sorted by their predicted score values and then shuffled uniformly at random. The first $s/2$ configurations are selected, where s is the number of the random training examples generated at the initialization phase. The evaluation of the selected configurations completes the generation of the new training examples.

The refinement of the utility model consists of learning the weights of the terms, discarding the terms with zero weight. In the following, we assume that the available feedback consists of a quantitative score. We thus learn the utility function by performing regression over the set of the Boolean vectors. Adapting the method to other forms of feedback, such as ranking of sets of solutions, is straightforward as will be discussed in Section 3.6. We address the regression task by the Lasso [46]. The Lasso is an appropriate choice on problem domains with many irrelevant features, as its 1-norm regularization can automatically select input features by assigning zero weights to the irrelevant ones. Feature selection is crucial for achieving accurate prediction if the underlying model is sparse [19].

Let $D = (\mathbf{x}_i, y_i)_{i=1\ldots m}$ the set of m training examples, where $\mathbf{x}_i$ is the Boolean vector and y_i its preference score. The learning task is accomplished by solving the following lasso problem:

$$\min_{\mathbf{w}} \sum_{i=1}^{m} (y_i - \mathbf{w}^T \cdot \Phi(\mathbf{x}_i))^2 + \lambda ||\mathbf{w}||_1 \tag{3.1}$$

where the mapping function Φ projects sample vectors to the space of all possible conjunctions of up to d Boolean variables. The learnt function $f(\mathbf{x}) = \mathbf{w}^T \cdot \Phi(\mathbf{x})$ will be used as the novel approximation of the utility function. A new iteration of our algorithm can now take place. The pseudocode of our algorithm is in Fig. 3.1.

In principle, one may argue that showing random examples to the user during the initialization phase (lines 5-6 in Fig. 3.1) is not an appropriate choice and may result a little bit artificial. However, the evaluation of diverse examples stimulates the preference expression, especially when the user is still uncertain about her final preference [37]. In particular, the diversity of the examples helps the user to reveal the hidden preferences: in many cases the decision maker is not aware of all preferences until she sees them violated. For example, a user does not usually think of stating the preference for an intermediate airport until one solution proposes an airplane change in a place she dislikes [37].

Note that dealing with the explicit projection Φ in Eq. 3.1 is tractable only for a rather limited number of catalog features and size of conjunctions d. This will typically be the case when interacting with a human DM. A possible alternative consists of directly learning a non-linear function of the features, without explicitly projecting them to the resulting higher dimensional space. We do this by kernel ridge regression [42] (Krr), where 2-norm regularization is used in place of 1-norm. The resulting dual formulation can be kernelized into:

$$\boldsymbol{\alpha} = (K + \lambda I)^{-1} \mathbf{y}$$

```
1.  procedure interactive_optimization
2.      input: set of the catalog variables
3.      output: configuration optimizing the learnt utility function
4.      /* Initialization phase */
5.      initialize training set $D$ by selecting $s$ configurations uniformly at random;
6.      get the evaluation of the configurations in $D$;
7.      while (termination_criterion)
8.          /* Learning phase */
9.          Based on $D$, select terms and relative weights for current
10.         weighted MAX-SAT formulation (Eq. 3.1);
11.         /* Optimization phase */
12.         Get new configurations by optimizing current weighted MAX-SAT
13.         formulation;
14.         /* Training examples selection phase */
15.         Select $s/2$ configurations, get their evaluation and add them to $D$;
16.     return configuration optimizing the learnt weighted MAX-SAT formulation
```

Figure 3.1: Pseudocode for the interactive optimization algorithm.

where K and I are the kernel and identity matrices respectively and λ is again the regularization parameter. The learnt function is a linear combination of kernel values between the example and each of the training instances: $f(\mathbf{x}) = \sum_{i=1}^{m} \alpha_i K(\mathbf{x}, \mathbf{x}_i)$. We employ a Boolean kernel [26] which implicitly considers all conjunctions of up to d features:

$$K_B(\mathbf{x}, \mathbf{x}') = \sum_{l=1}^{d} \binom{\mathbf{x}^T \cdot \mathbf{x}'}{l}$$

With the lasso, the function $\Phi(\cdot)$ maps the Boolean variables to all possible terms of size up to d. This allows for an explicit representation of the learnt utility function f as a weighted combination of the selected Boolean terms. On the other hand, in the kernel ridge regression case terms are only implicitly represented via the Boolean kernel K_B. In both cases, the value of the learnt function f is used to guide the search of the SLS algorithm. In the following, the two proposed approaches are referred as the *Lasso* and the *Krr* algorithms. As will be shown in the experimental section, the sparsity-inducing property of the *Lasso* allows it to consistently outperform *Krr*. The problem of addressing more complex scenarios, possibly involving non-human DM, where we can not afford an explicit projection, will be discussed in Section 3.6.

3.3 Satisfiability Modulo Theory

In the previous section, we assumed our optimization task could be cast into a *propositional* satisfiability problem. However, many applications of interest require or are more naturally described in more expressive logics as first-order logic (FOL), involving quantifiers, functions and predicates. In these cases, one is usually interested in validity of a FOL formula with respect to a certain *background theory* T fixing the interpretation of (some of the) predicate and function symbols. A general purpose FOL reasoning system such as Prolog, based on the resolution calculus, needs to add to the formula a conjunction of all the axioms in T. This is, for instance, the standard setting we consider in inductive logic programming when verifying whether a certain hypothesis covers an example given the available background knowledge. Whenever the cost of including such additional background theory is affordable, our algorithm can be applied rather straighforwardly.

Unfortunately, adding all axioms of T is not viable for many theories of interest: consider for instance the theory of *arithmetic*, which restricts the interpretation of symbols such as $+, \geq, 0, 5$. A more efficient alternative consists of using *specialized* reasoning methods for the background theory of interest. The resulting problem is known as *satisfiability modulo theory* (SMT)[1] and has drawn a lot of attention in recent years, guided by its applicability to a wide range of real-world problems. Among them, consider, for example, problems arising in formal hardware/software verification or in real-time embedded systems design. Popular examples of useful theories include various theories of arithmetic over reals or integers such as linear or difference ones. Linear arithmetic considers $+$ and $-$ functions alone, applied to either numerical constants or variables, plus multiplication by a numerical constant. Difference arithmetic is a fragment of linear arithmetic limiting legal predicates to the form $x - y \leq c$, where x, y are variables and c is a numerical constant. Very efficient procedures exists for checking satisfiability of difference logic formulas [34].

A number of theories have been studied apart from standard arithmetic ones. Machine arithmetic, for instance, is more naturally modeled by the theory of bit-vector arithmetic, which includes bit-wise operations. The theory of arrays includes two functions `read(a,i)` and `write(a,i,v)`. The former returns the value of array `a` at index `i`, the latter an array identical to `a` but for position `i` having value changed to `v`. This theory is extensively used to model arrays in programs as well as an abstraction of memory. Other theories exists for data structures such as lists and strings.

3.3.1 Satisfiability Modulo Theory solvers

The most successful SMT solvers can be grouped into the two main approaches named *eager* and *lazy*. The eager approach consists of developing theory-specific and efficient translators which translate a query formula into an equisatisfiable propositional one, much like compilers do when optimizing the code generated from a high-level program. Lazy approaches, on the other hand, work by building efficient *theory solvers*, inference systems specialized on a theory of interest. These solvers are integrated as submodules into a generic SAT solver. In the rest of the chapter we will focus on this latter class of SMT solvers, which we integrated in our optimization algorithm. The simplest approach for building a lazy SMT-solver consists of alternating calls to the satisfiability and the theory solver respectively,

```
procedure SMT-solver($\varphi$)
    $\varphi' = \alpha(\varphi)$
    while (true)
        (r,M) ← SAT($\varphi'$)
        if r = unsat then return unsat
        (r,J) ← T-Solver($\beta(M)$)
        if r = sat then return sat
        C ← $\bigvee_{l \in J} \neg\alpha(l)$
        $\varphi' \leftarrow \varphi' \wedge C$
```

Figure 3.2: Pseudocode for a basic lazy SMT-solver.

until a solution satisfying both solvers is retrieved or the problem is found to be unsatisfiable. Let φ be a formula in a certain theory T, made of a set of n predicates $A = \{a_1, \ldots, a_n\}$. A mapping α maps φ into a propositional formula $\alpha(\varphi)$ by replacing its predicates with propositional variables $p_i = \alpha(a_i)$. The inverse mapping β replaces propositional variables with their corresponding predicates, i.e., $\beta(p_i) = a_i$. For example, consider the following formula in a non-linear theory T:

$$(\cos(x) = 3 + \sin(y)) \wedge (z \leq 8) \tag{3.2}$$

Then, $p_1 = \alpha(\cos(x) = 3 + \sin(y))$ and $p_2 = \alpha(a_i \leq 8)$. Note that the truth assignment $p_1 = true, p_2 = false$ is equivalent to the statement $(\cos(x) = 3 + \sin(y)) \wedge (z > 8)$ in the theory T.

Figure 3.2 reports the basic form [32] of an SMT algorithm. `SAT`(φ) calls the SAT solver on the φ instance, returning a pair (r, M), where r is `sat` if the instance is satisfiable, `unsat` otherwise. In the former case, M is a truth assignment satisfying φ. `T-Solver`(S) calls the theory solver on the formula S and returns a pair (r, J), where r indicates if the formula is satisfiable. If $r =$`unsat`, J is a *justification* for S, i.e any unsatisfiable subset $J \subset S$. The next iteration calls the SAT solver on an extended instance accounting for this justification.

State-of-the-art solvers introduce a number of refinements to this basic strategy, by pursuing a tighter integration between the two solvers. A common underlying idea is to prune the search space for the SAT solver by calling the theory solver on partial assignments and propagating its results. Finally, combination methods exist to jointly employ different theories, see [33] for a basic procedure.

3.3.2 Weighted MAX-SMT

Weighted MAX-SMT generalizes SMT problems much like weighted MAX-SAT does with SAT ones. While a body of works exist addressing weighted MAX-SAT problems, the former generalization has been tackled only recently and very few solvers have been developed [18, 35, 13]. The simplest formulation consists of adding a cost to each or part of the formulas to be jointly satisfied, and returning the assignment of variables minimizing the sum of the costs of the unsatisfied clauses, or a satisfying assignment if it exists. The following is a "weighted version" of Eq. 3.2:

$$5 \cdot (\cos(x) = 3 + \sin(y)) + 12 \cdot (z > 8) \tag{3.3}$$

where 5 and 12 are the cost of the violation of the first and the second predicate, respectively.

Generalizing, consider a true utility function f expressed as a weighted sum of terms, where a term is the conjunction of up to d predicates defined over the variables in the theory T. The set of *all* n possible predicates represents the search space S of the MAX-SAT solver integrated in the MAX-SMT solver. Our approach learns an approximation $\hat{f}$ of f and gets one of its optimizers $\mathbf{v}$ from the MAX-SMT solver. The optimizer (and in general each candidate solution in the theory T) identifies an assignment $\mathbf{p}^* = (p_1^*, \ldots, p_n^*)$ of Boolean values ($p_i^* = \{\mathbf{true}, \mathbf{false}\}$) to the predicates in S. The DM is asked for a feedback on the candidate solution $\mathbf{v}$ and returns a possibly noisy quantitative score $s \approx f(\mathbf{v})$. The pair $(\mathbf{p}^*, s)$

represents a new training example for our approach. In order to obtain multiple training examples, we optimize again $\hat{f}$ with the additional *hard*[1] constraint generated by the disjunction of all the terms of $\hat{f}$ unsatisfied by $\mathbf{p}^*$. For example, let t_1 and t_5 be the terms of $\hat{f}$ unsatisfied by $\mathbf{p}^*$, then the hard constraint becomes:

$$(t_1 \vee t_5)$$

If $\mathbf{p}^*$ satisfies all the terms of $\hat{f}$, i.e., $\hat{f}(\mathbf{p}^*) = 0$, the additional *hard* constraint generated is

$$(\neg p_1^* \vee \neg p_2^* \ldots \vee \neg p_n^*)$$

which excludes $\mathbf{p}^*$ from the feasible solutions set of $\hat{f}$. The generation of the training examples is iterated till the desired number of examples have been created or the hard constraints generated made the MAX-SMT problem unsatisfiable.

The learning component of our algorithm is then re-trained, including in the training set the new collected examples and the approximation of the true utility function is refined. A new optimization phase can now take place (see Fig. 3.1).

The mechanism creating the training examples is motivated by the tradeoff between the selection of good solutions (w.r.t. the current approximation of the true utility function) and the diversification of the search process.

3.4 Related works

Active learning is a hot research area and a broad range of different approaches has been proposed (see [40] for a review). The simplest and most

[1] *Hard* constraints do not have a cost, and they have to be satisfied. On the contrary, the terms with a cost, which may or may not be satisfied, are called *soft* constraints.

common framework is that of *uncertainty sampling*: the learner queries the instances on which it is least certain. However, the ultimate goal of a recommendation or optimization system is selecting the best instance(s) rather than correctly modeling the underlying utility function. The query strategy should thus tend to suggest good candidate solutions and still learn as much as possible from the feedback received. Typical areas where research on this issue is quite popular are single- and multi-objective interactive optimization [7] and information retrieval [38]. The need to trade off multiple requirements in this active learning setting is addressed in [49] where the authors consider relevance, diversity and density in selecting candidates. Note that our approach relies on query *synthesis* rather than selection, as *de-novo* candidate solutions are generated by the SLS algorithm. Nonetheless, our diversification strategies are very simple and could be significantly improved by taking advantage of the aforementioned literature.

Choosing relevant features according to their weight within the learnt model is a common selection strategy (see e.g. [22]). When dealing with implicit feature spaces as in kernel machines, the problem can be addressed by introducing a hyper-parameter for each input feature, like a feature-dependent variance for Gaussian kernels [12]. Parameters and hyper-parameters (or their relaxed real-valued version) are jointly optimized trying to identify a small number of relevant features. One-norm regularization [46] has the advantage of naturally inducing sparsity in the set of selected features. Approaches also exist [47, 24] which directly address the combinatorial problem of zero-norm optimization.

A large body of recent work exists for developing interactive approaches [7] to multiobjective optimization. A common approach consists of modeling the utility function as a linear combination of objectives, and iteratively updating its weights trying to match the DM requirements. Our algorithm allows to deal with complex non-linear interactions between (Boolean) objectives and, thanks to the SMT extension, can be applied to a wide range of optimization problems.

3.4.1 Constraint programming approaches for preference elicitation

Very recent works in the field of constraint programming [21] define the user preferences in terms of *soft* constraints and introduce constraint optimization problems where the data are not completely known before the solving process starts.

In soft constraints, a generalization of hard constraints, each assignment to the variables of one constraint is associated to a preference value taken from a preference set. The preference value represents the level of desirability of the assignment. The desirability of a complete assignment is computed by applying a combination operator to the local preference values. A set of soft constraints generates an order (partial or total) over the complete assignments of the variables of the problem. Given two solutions of the problem, the preferred one is selected by computing their preference levels and by comparing them in the preference order. Soft constraints are therefore represented by an algebraic structure, called *c-semiring* (where letter "c" stays for "constraint"), providing two operations for combining ($\times$) and comparing ($+$) preference values. In detail, the c-semiring is a tuple $(A, +, \times, \mathbf{0}, \mathbf{1})$ where:

- A is a set and $\mathbf{0}, \mathbf{1} \in A$;
- $+$ is commutative, associative and idempotent; $\mathbf{0}$ is its unit element and $\mathbf{1}$ is its absorbing element;
- $\times$ is commutative, associative, distributes over $+$; $\mathbf{1}$ is its unit element and $\mathbf{0}$ is its absorbing element.

Note that a c-semiring is a semiring with additional properties for the two operations: the operation $+$ must be idempotent and with $\mathbf{0}$ as absorbing element, the operation $\times$ must be commutative. The relation $\leq_A$ over A, $a \leq_A b$ iff $a + b = b$, is a partial order, with $\mathbf{0}$ and $\mathbf{1}$ its minimum and maximum elements, respectively. The relation $\leq_A$ allows to compare (some of) the desirability levels, with $a \leq_A b$ meaning that b is "better" than a; $\mathbf{0}$ and $\mathbf{1}$ represent the worst and the best preference levels, respectively, and the operations $+$ and $\times$ are monotone on $\leq_A$.

The c-semiring formalism can model just *negative* preferences. First, the best element in the ordering induced by $\leq_A$, denoted by $\mathbf{1}$, behaves as indifference, since $\forall a \in A, 1 \times a = a$. This result is consistent with intuition: when using only negative preferences, indifference is the best level of desirability that can be expressed. Furthermore, the combination of desirability levels returns a lower overall preference, since $a \times b \leq_A a, b$. This result reflects the desired property of negative preferences: the combination of desirability levels returns lower preferences.

The generality of the semiring-based soft constraint formalism allows to express several kinds of preferences, including partially ordered ones. For example, different instances of c-semirings encode weighted or probabilistic soft constraint satisfaction problems [6].

The work in [21] introduces an elicitation strategy for soft constraint problems with missing preferences, with the purpose of finding the solution preferred by the DM asking to reveal as few preferences as possible. Despite the common purpose, this approach is different from ours. A major difference regards the preference elicitation problem considered. In [21] decision variables and soft constraints are assumed to be known in advance and the information uncertainty consists only of missing preference values. On the other hand, our settings assume sparsity of the utility function, both in the number of features (from the whole set of catalog features) and in the selection of the terms constructed from them.

The technique in [21] is based on *local* elicitation queries, with the final user asked to reveal her preferences about assignments for specific soft constraints. *Global* preferences or bounds for global preferences associated to complete solutions of the problem are derived from the local preference information. Our technique goes in the opposite direction, inducing local utilities from global preference values. In many cases, recognizing appealing or unsatisfactory global solutions may be much easier than defining local utility functions, associated to partial solutions. For example, while scheduling a set of activities, the evaluation of complete schedules may be more affordable than assessing how specific ordering choices between couples of activities contribute to the global preference value.

In order to reduce the embarrass of the decision maker when specifying precise preference scores, interval-valued constraints [28] allow users to state an interval of utility values for each instantiation of the variables of a constraint. The adoption of interval-valued soft constraints is appealing when the user may have a vague idea of the preference scores or when she may not be willing to reveal her preference, for example for privacy reasons. Furthermore, note that informal definitions of degrees of preference such as “quite high”, “more or less”, “low” or “undesirable” cannot be naturally mapped to precise preference scores. However, the technique described in [28] requires the user to provide all the information she has about the problem (in terms of preference intervals) *before* the solving phase, without seeing any optimization result.

Even if interval-valued constraints [28] have been introduced to handle uncertainty in the evaluations of the DM, the experiments in [21] do not consider the case of inconsistent preference information. Our technique is robust to imprecise information from the DM, modeled in terms of inaccurate preference scores for the candidate solutions. On the other side, the optimality of the solutions produced by the technique in [21] is guaranteed and the empirical studies show that on fuzzy constraint satisfaction problems with missing preferences the algorithm can also provide a solution at any point in time, whose quality increases with the computation time (anytime property). While our iterative approach satisfies the anytime property, it cannot guarantee the optimality of the retrieved solution. However the promising experimental results show the ability of our heuristic to find the optimal solution.

When asking to the DM quantitative evaluations of the solutions results inappropriate or even impossible, rather than resorting to preference intervals, our technique can be straightforwardly extended to express preference information in terms of qualitative judgments, based on the comparison of complete solutions. The extension basically consists of the replacement of support vector regression with support vector ranking, as discussed below in Sec. 3.6.

Furthermore, while the works in [21] considers *unipolar* preference problems, modeling just negative preferences, our approach naturally accounts for bipolar preference problems, with the final user specifying what she likes and what she dislikes. Bipolar preference problems provides a better representation of the typical human decision process, where the degree of preference for a solution reflects the compensation value obtained by comparing its advantages with the disadvantages. Note that the work in [6] extends the soft constraint formalism to account for bipolar preference problems. For further details, see Appendix A.

Finally, while the technique in [21] combines branch and bound search with preference elicitation and the adoption of local search algorithms is matter of research, our approach works straightforwardly with both incomplete and complete search techniques.

3.5 Experimental results

The following empirical evaluation demonstrates the versatility and the efficiency of our approach for the weighted MAX-SAT and the weighted MAX-SMT problems. The MAX-SMT tool used for the experiments is the "Yices" solver [18]. Each point of the curves depicting our results is the median value over 400 runs with different random seeds, unless otherwise stated.

3.5.1 Weighted MAX-SAT

The *Lasso* and the *Krr* algorithms were tested over a benchmark of randomly generated utility functions according to the triplet (*number of features, number of terms, max term size*), where *max term size* is the maximum allowed number of Boolean variables per term. We generate functions for: $\{(5,3,3),(6,4,3),(7,6,3),(8,7,3),(9,8,3),(10,9,3)\}$. Each utility function has two terms with maximum size. Terms weights are integers selected uniformly at random in the interval $[-100,0) \cup (0,100]$. We consider as *gold* standard solution the configuration obtained by optimizing the true utility function.

The number of catalog features is 40. The maximum size of terms is assumed to be known. The walk probability parameter of the SLS algorithm *wp* is set to 0.2. Furthermore, the score values of the training examples are affected by Gaussian noise, with mean 0 and standard deviation 10.

We run a set of experiments for $10, 20, \ldots 100$ initial training examples, for the *Lasso* and the *Krr* versions of the algorithm. Results are expressed in terms of the quality of the learnt utility function (Fig. 3.3) and of the approximation of the *gold* solution (Fig. 3.4). Each point of the curves in the Fig. 3.3 and 3.4 is the mean and the median values, respectively, over 400 runs with different random seeds.

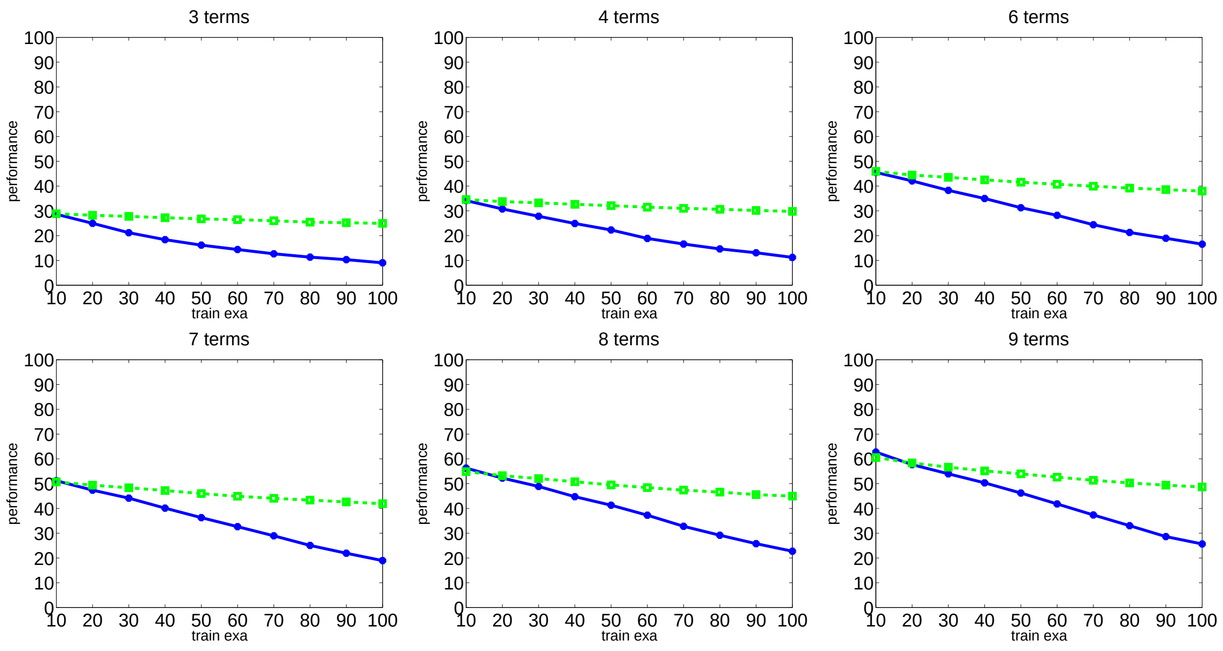

Figure 3.3: Quality of the learnt utility function for an increasing number of training examples observed for the algorithms at the first iteration. The y-axis reports the root mean squared error between the true and the predicted values for a benchmark of 1000 test examples. The x-axis contains the number of training examples. The solid blue and the dashed green lines show the performance of the *Lasso* and the *Krr* algorithms, respectively. See text for details.

Fig. 3.3 shows the quality of the learnt utility function, in terms of the root mean squared error (rmse) between the true and the predicted values for a benchmark of 1000 test examples. A better approximation is generated by the *Lasso* algorithm for all the considered true utility functions. Furthermore, while increasing the number of training examples, a faster improvement is observed for the *Lasso* w.r.t. the *Krr* algorithm. Consider, for example, the case of nine terms. With 40 training examples, the performance of *Krr* is within 10 units from the value observed for the *Lasso* method. When 100 examples are employed, the mean rmse of the *Lasso* algorithm is less than value 30, while the performance of the *Krr* method does not increase beyond value 50.

The superior performance of the *Lasso* algorithm is confirmed by the experiments in Fig. 3.4, reporting the quality of the *best* configuration at the different iterations for an increasing number of initial training examples. The *best* configuration is the configuration optimizing the current approximation of the true utility function. Its quality is measured in terms of the approximation error w.r.t. the gold solution.

Considering the simplest problems with three and four terms, the performance of *Krr* is comparable with the results obtained by *Lasso*, except at the first iteration of *Krr* in the case of four terms true utility functions, where the gold solution is not identified even with 100 initial training examples.

However, the *Lasso* approach outperforms the *Krr* results when the true utility function includes at least six terms. First, note that the *Lasso* algorithm succeeds in exploiting its active learning strategy, and converges rather quickly to the optimal solution when enough iterations are provided. At the first iteration its approximation error is above 40 even when 30 training examples are used. At the third iteration, the *Lasso* algorithm identifies the gold standard solution, when at least 60 training examples

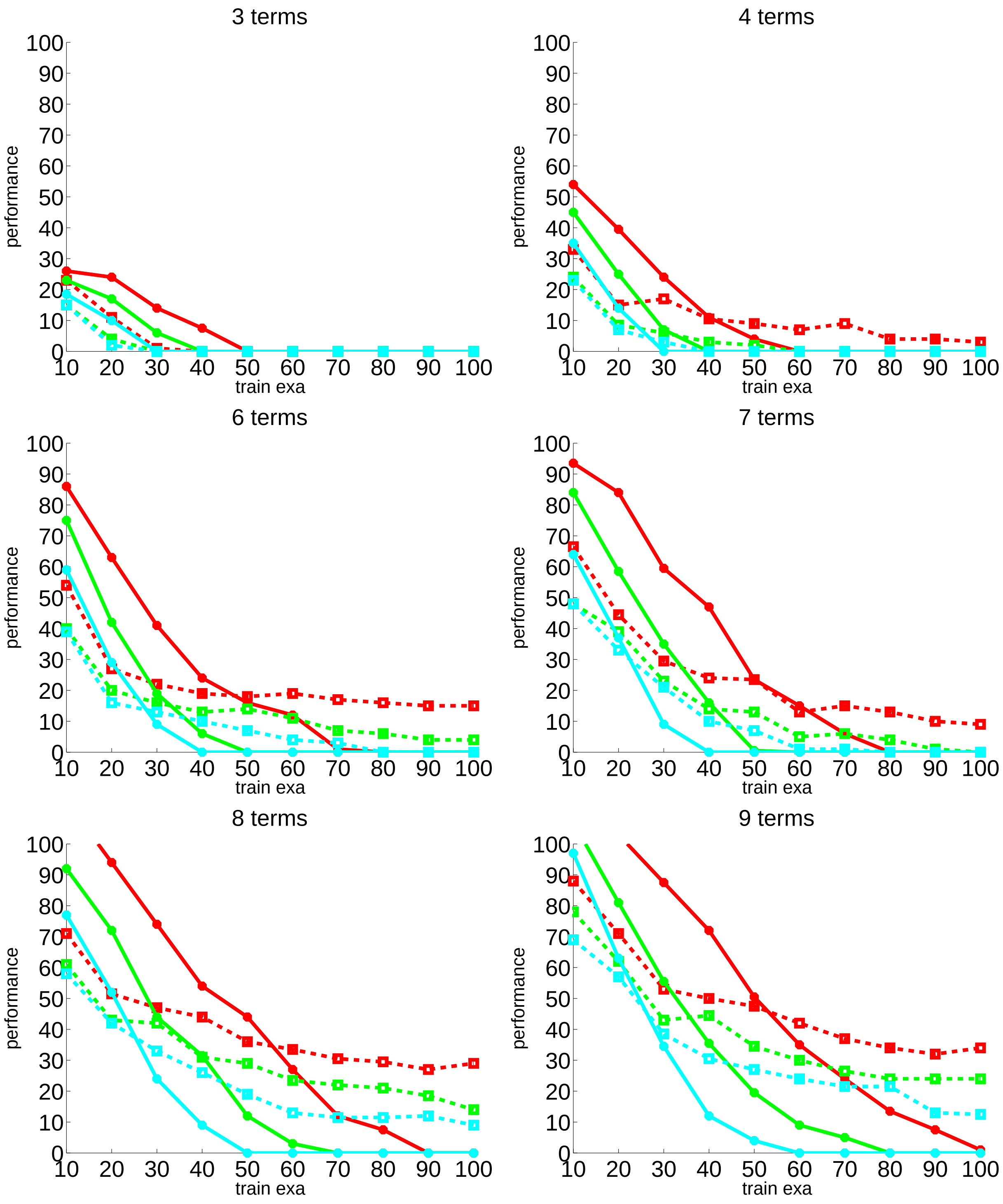

Figure 3.4: Learning curves for an increasing number of training examples observed for the two algorithms at different iterations. The y-axis reports the solution quality, while the x-axis contains the number of training examples. The dashed lines refer to the *Krr* algorithm, while the solid lines are for the *Lasso* algorithm. Furthermore, red, green and cyan colors show the performance of the algorithms at the first, the second and the third iteration, respectively. See text for details.

are available. On the other hand, for true utility functions with more than seven terms *Krr* fails to improve over its suboptimal solution when increasing the number of examples and iterations. As a consequence, the *Krr* algorithm does not identify the gold solution, even in the case of 100 training examples. However, when very few training examples are available, the *Krr* algorithm reaches a better approximation than *Lasso*.

Further results for the *Lasso* algorithm are reported in Appendix B. In particular, they refer to a second implementation of the technique, with the SLS algorithm in the optimization stage replaced by a complete solver and with the Boolean training examples generation described in Sec. 3.3.2 for the MAX-SMT version of our approach.

Fig. 3.5 and Fig. 3.6 show the learning curves for both the *Lasso* and *Krr* approaches at the first and third iteration, respectively, in the case of true utility functions with nine terms. Error bars indicates the range between the 25th and 75th percentiles of the underlying data distributions. In both cases, the sample percentiles demonstrate a more stable behavior of the *Lasso* technique. In particular, at the first iteration the stability of *Lasso* increases with additional training examples, while the variability of *Krr* does not decrease of the same extent. At the third iteration, with at least 80 examples the *Lasso* algorithm consistently finds the gold solution, while an unstable behavior is still observed for *Krr*.

3.5.2 Weighted MAX-SMT

SMT is a hot research area [32]. However, MAX-SMT techniques are very recent and there are no well established publicly available benchmarks for weighted MAX-SMT problems. Existing results [35] indicate that MAX-SMT solvers can efficiently address real-world problems.

In this work, we modeled a scheduling problem as a MAX-SMT problem. In detail, a set of five jobs must be scheduled over a given period of

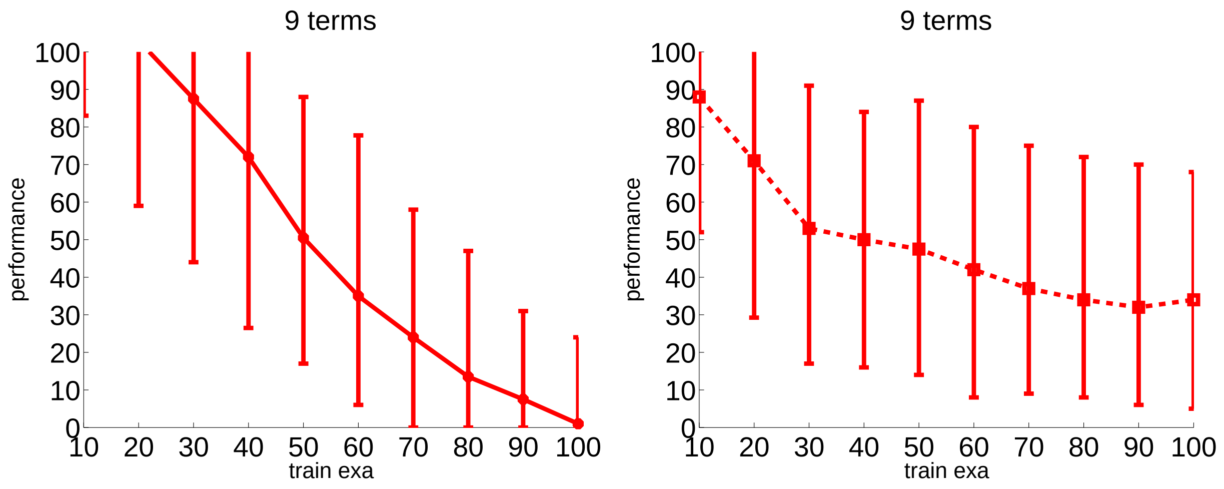

Figure 3.5: Learning curves observed for the *Lasso* (right) and *Krr* (left) algorithms at the first iteration in the case of true utility functions with nine terms. Error bars represents the range among the 25th and the 75th percentiles of the measurements.

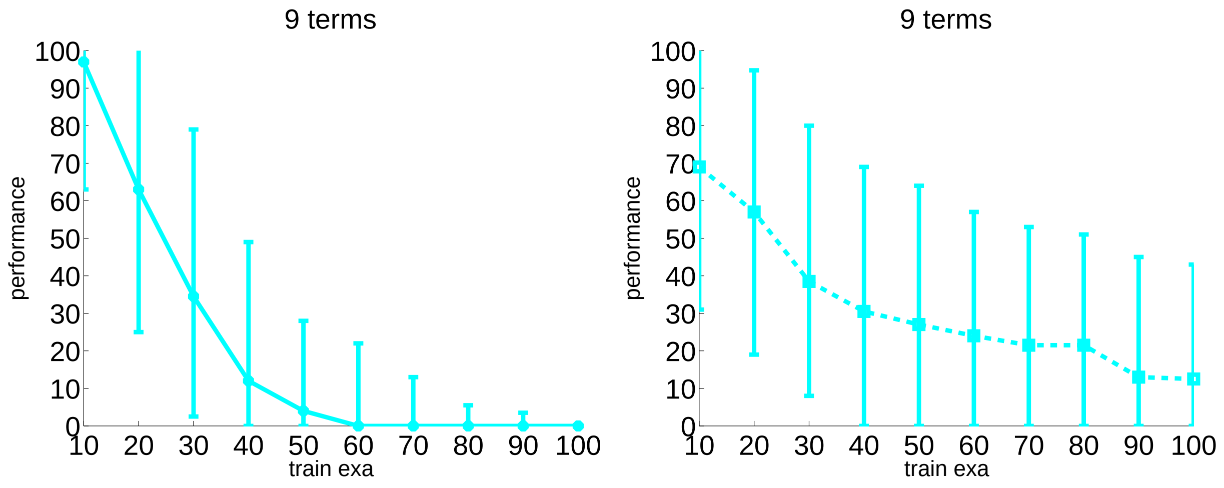

Figure 3.6: Learning curves observed for the *Lasso* (right) and *Krr* (left) algorithms at the third iteration in the case of true utility functions with nine terms. Error bars represents the range among the 25th and the 75th percentiles of the measurements.

time. Each job has a fixed known duration, the constraints define the overlap of two jobs or their non-concurrent execution. The true utility function is generated by selecting uniformly at random weighed terms over the constraints. The solution of the problem is a schedule assigning a starting date to each job and minimizing the cost, where the cost of the schedule is the sum of the weights of the violated terms of the true utility function. The temporal constraints are expressed by using the difference arithmetic theory. In detail, let s_i and d_i, with $i = 1 \ldots 5$, be the starting date and the duration of the i-*th* job, respectively. If s_i is scheduled before s_j, the constraint expressing the overlap of the two jobs is $s_j - s_i < d_i$, while their non-concurrent execution is encoded by $s_j - s_i \geq d_i$ Note that there are 40 possible constraints for a set of 5 jobs. The maximum size of the terms of the true utility function is three and it is assumed to be known. Their weights are distributed uniformly at random in the range $[1, 100]$. Similarly to the MAX-SAT case, the experimental setting includes Gaussian noise (with mean 0 and standard deviation 10) affecting the cost values of the training examples.

Fig. 3.7 depicts the performance of the Lasso algorithm for the cases of 3, 4, 6, 7, 8, 9 terms in the true utility function. The y-axis reports the solution quality measured in terms of deviation from the gold solution, while the x-axis contains the number n of training examples at the first iteration. At the following iterations, $n/2$ examples are added to the training set (see Sec. 3.2).

As expected, the learning problem becomes more challenging while increasing the number of terms. However, the results for the scheduling problem are promising: our approach identifies the gold standard solution in all the cases. In detail, less than 40 examples are required to identify the gold solution at the second iteration. At the third iteration our algorithm needs only 20 training examples for convergence to the gold solution.

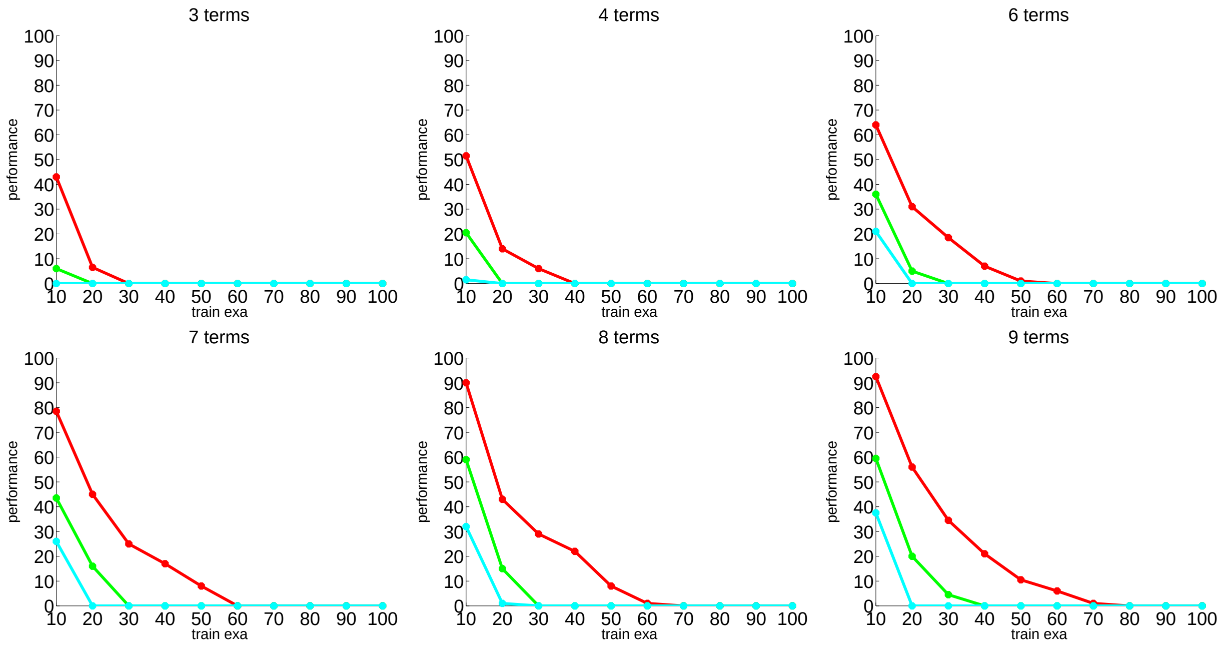

Figure 3.7: Learning curves observed at different iterations of the Lasso algorithm while solving the scheduling problem. The y-axis reports the solution quality, while the x-axis contains the number of training examples. Red, green and cyan colors show the performance of the algorithm at the first, the second and the third iteration, respectively. See text for details.

Note that the approach based on Krr does not maintain an explicit representation of the learnt utility function, and therefore a direct extension to SMT problems is not possible for the current MAX-SMT solvers which tightly integrate SAT and theory solvers as discussed in Section 3.3.

The plots in Fig. 3.8 and Fig. 3.9 show that at the third iteration our approach finds the gold solution consistently, provided that at least 50 initial examples are used in the case of nine terms true utility functions. As expected, a more unstable behavior is observed at the first iteration for both three and nine terms cases.

For a second realistic application of our preference elicitation technique, consider a customer judging potential housing locations provided by a real estate company (henceforth the Housing problem). There are different

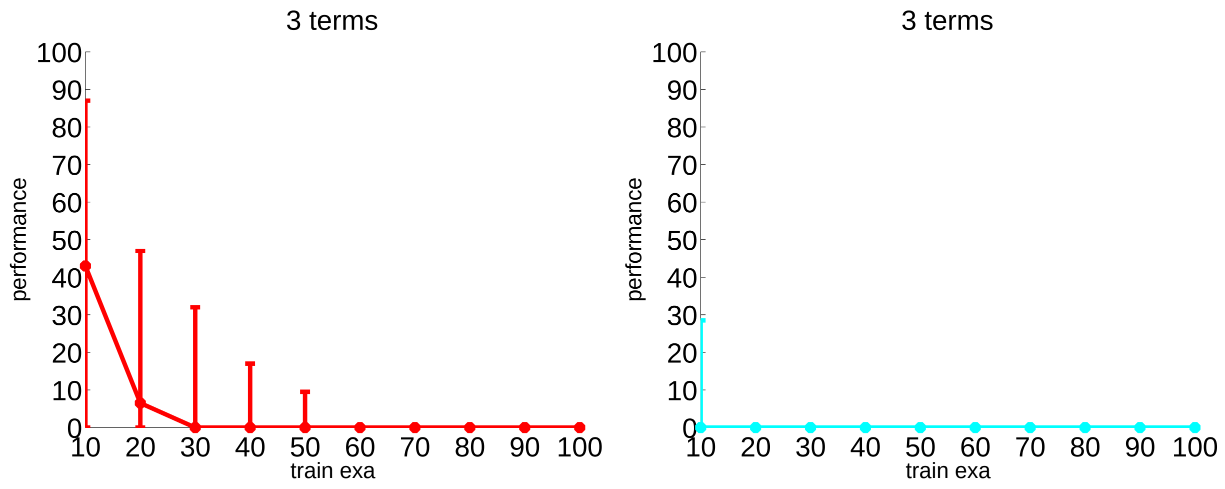

Figure 3.8: Learning curves at the first (left Figure) and the third (right Figure) iterations observed while solving the scheduling problem with three terms in the true utility function. Error bars denote the range among the 25th and the 75th percentiles of the measurements.

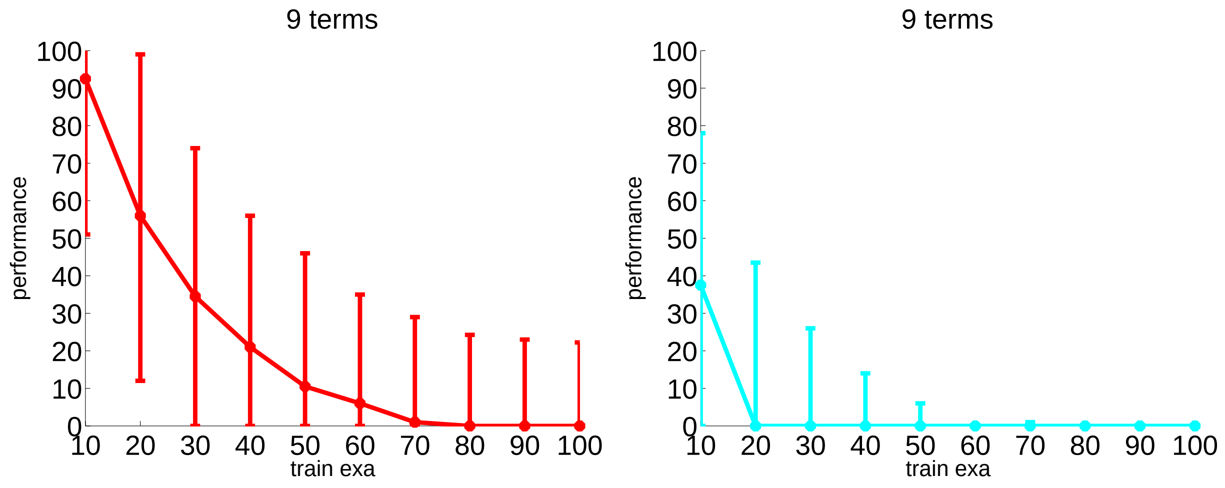

Figure 3.9: Learning curves at the first (left Figure) and the third (right Figure) iterations for scheduling problems with true utility functions of nine terms. Error bars denote the range among the 25th and the 75th percentiles of the measurements.

locations available, characterized by different housing values, prices, constraints about the design of the building (e.g., in the city center you cannot have a family house with a huge garden and pool), etc. The customer may formulate her judgments by considering a description of the housing locations based on a predefined set of parameters, including, e.g., crime rate, distance from downtown, location-based taxes and fees, public transit service quality, cultural resources accessibility, walking and cycling facilities, etc. In addition, she is free to express her own requirements, consisting of financial issues, working opportunities, personal interests (e.g., the proximity to commercial facilities or green areas), etc. As a result, this problem is characterized by a plethora of decisional features whose contribution in the definition of the user preferences cannot be quantified in advance. Many of them may be redundant, as they do not represent any decisional criteria for the customer. Furthermore, while specifying in advance hard constraints for the locations may be straightforward (consider, e.g., cost bounds stated by the user or building design requirements asserted by the company), assessing the user preferences in terms of the combination of this redundant set of decisional features may demand a prohibitive effort. In the real world, the elicitation process is usually driven by the sales personnel of the company in collaboration with the customer. Their joint effort identifies the customer decisional features from the catalog set and defines the (nonlinear) relationships among the selected features. For example, consider the following preference information from the decision maker: "I like family houses with a big garden and I'm not interested in living near the place where I work. On the other hand, I would like a location near the school of my children. However, in the case of good price, I could accept a flat in the downtown, provided that commercial facilities are reachable on foot and there are free parkings in the neighborhood". Finally, in order to provide satisfying locations to the customer, the sales personnel has to assess a rank for the (possibly conflicting) stated preferences. Considering the previous preference information statement, the sales personnel should quantify, e.g., how much a family house with big garden is preferred to a location near the children's school (or viceversa).

However, this process may often produce poor results, which do not fulfill the expectations of the user. In most of the cases, a complete and precise formulation of the user preferences cannot be accomplished before the customer becomes aware of some possible solutions. As a result, soft constraints remain in the mind of the decision maker, and revisions of the stated preferences after seeing the actual optimization results are an inescapable fact. To complicate things, misunderstanding between the persons may arise and possibly imprecise and inconsistent answers of the user to the elicitation queries have to be considered. In this context, our preference elicitation technique provides a robust housing location recommendation system that can evaluate the suitability of the locations and optimize them for the customer. On the other side, the application of the preference elicitation technique introduced in [21] is difficult, as it assumes to know in advance both the decisional features of the user and their detailed combination (represented in terms of soft constraints), while the elicitation process focuses exclusively on assessing the preferences for the different instantiations of the variables of the constraints.

In our experiments, the formulation of the housing problem is as follows. The set of catalog features is listed in Tab. 3.1. A set of 10 hard constraints (Tab. 3.2) defining feasible housing locations and known in advance is considered. The hard constraints are stated by the costumer (e.g., cost bounds) or by the company (e.g, constraints about the distance of the available locations from user-defined points of interest). Note that constraints 5,6,7 define a linear bi-objective problem among distances from user-defined points of interest. Prices of potential housing locations are

Table 3.1: Decisional features for the housing problem.

num	feature	type
1	house type	categorical
2	garden	Boolean
3	garage	Boolean
4	commercial facilities in the neighborhood	Boolean
5	public green areas in the neighborhood	Boolean
6	cycling and walking facilities in the neighborhood	Boolean
7	price	numerical
8	distance from downtown	numerical
9	crime rate	numerical
10	location-based taxes and fees	numerical
11	public transit service quality index	numerical
12	distance from high schools	numerical
13	distance from nearest free parking	numerical
14	distance from working place	numerical
15	distance from parents house	numerical

defined as a function of the other features. For example, price increases if a semi-detached house rather than a flat is selected or in the case of green areas in the neighborhood. On the other side, e.g., when crime index of potential locations increases, price decreases. Soft constraints are represented by weighted terms including predicates in the linear arithmetic theory or Boolean variables, in the case of features $2, 3, \ldots, 6$ in Tab. 3.1. For example, one predicate may model the preference for a location with distance from nearest free parking smaller than a given threshold, while, a Boolean variable encodes, e.g., the aspiration for houses with garage.

We generated a set of 40 predicates. The true utility function is composed of terms with two up to three predicates, with at least one term with maximum size. Term weights are integer values selected uniformly at random in the range [1, 100]. The experimental setting includes Gaussian noise (with mean 0 and standard deviation 10) affecting the cost values of

Table 3.2: Hard constraints for the housing problem. Parameter *ts* is a threshold value specified by the user or by the sales personnel.

num	hard constraint
1	price $\leq$ *ts*
2	location-based taxes and fees $\leq$ *ts* => *not* public green ares in the neighborhood *and not* public transit service quality index $\leq$ *ts*
3	commercial facilities in the neighborhood => *not* (garden *and* garage)
4	crime rate $\leq$ *ts* => distance from downtown $\geq$ *ts*
5	distance from working place + distance from parents house $\geq$ *ts*
6	distance from working place + distance from high schools $\geq$ *ts*
7	distance from parents house + distance from high schools $\geq$ *ts*
8	distance from nearest free parking $\leq$ *ts* => *not* public green areas in the neighborhood
9	distance from parents house $\leq$ *ts* => distance from downtown $\geq$ *ts and* crime rate $\geq$ *ts*
10	garden => house type $\geq$ *ts*

the training examples.

Fig. 3.10 reports the results over a benchmark of 400 randomly generated utility functions for each of the following instantiation of the triplet (*number of features, number of terms, max term size*): $\{(5,3,3),(6,4,3),(7,6,3),(8,7,3),(9,8,3),(10,9,3)\}$. The promising results observed for the scheduling problem are confirmed. A stable behavior is observed for our approach at the third iteration: the quality of the solution rapidly improves with a larger number of examples and the algorithm succeeds in exploiting its active learning strategy. As a consequence, the gold solution is quickly identified.

Fig. 3.11 shows the ability of the algorithm to converge to a stable result while solving Housing problems with true utility functions of three terms. The more challenging elicitation task represented by true utility functions with nine terms (Fig. 3.12) indicates that running three iterations of the

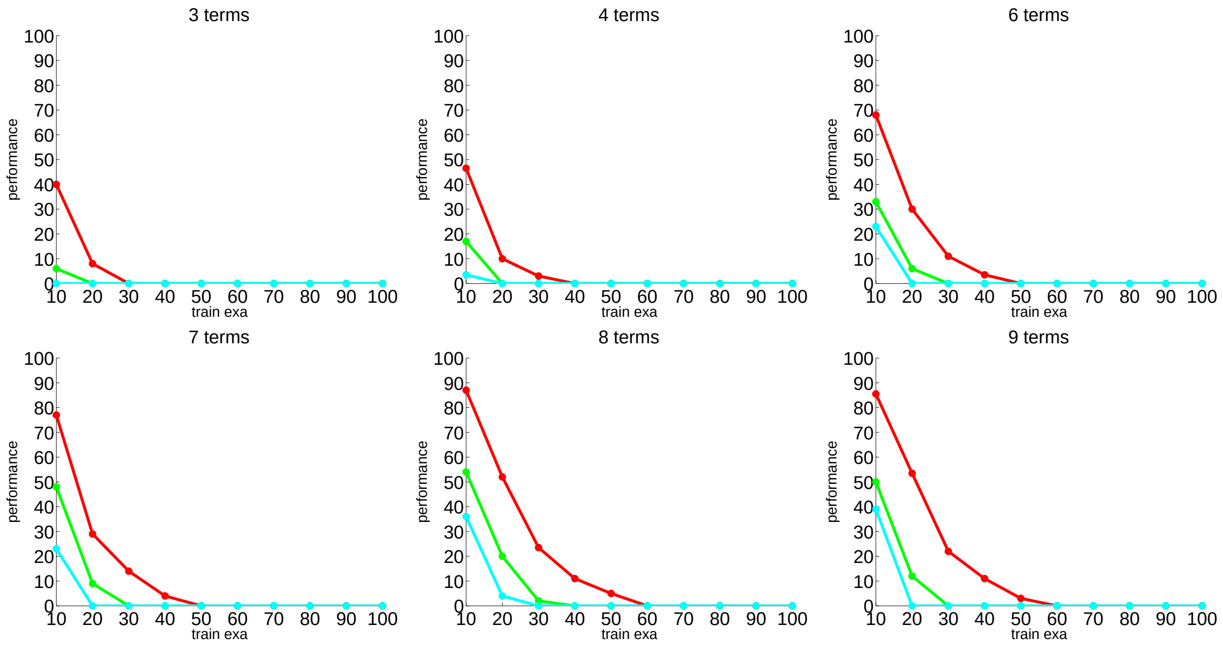

Figure 3.10: Learning curves observed at different iterations of the Lasso algorithm while solving the Housing problem. The data are presented analogously to that in Fig. 3.7.

algorithm results in less variability than unstable performance observed at the first iteration, confirming the efficiency of our incremental approach.

3.6 Discussion

We presented an interactive optimization strategy for combinatorial problems over an unknown utility function. The algorithm alternates a search phase using the current approximation of the utility function to generate candidate solutions, and a refinement phase exploiting feedback received to improve the approximation. One-norm regularization is employed to enforce sparsity of the learned function. An SLS algorithm addresses the weighted MAX-SAT problem resulting from the search phase. We show how to adapt the approach to a large class of relevant optimization problems dealing with satisfiability modulo theories. Experimental results on both weighted MAX-SAT and MAX-SMT problems demonstrate the ef-

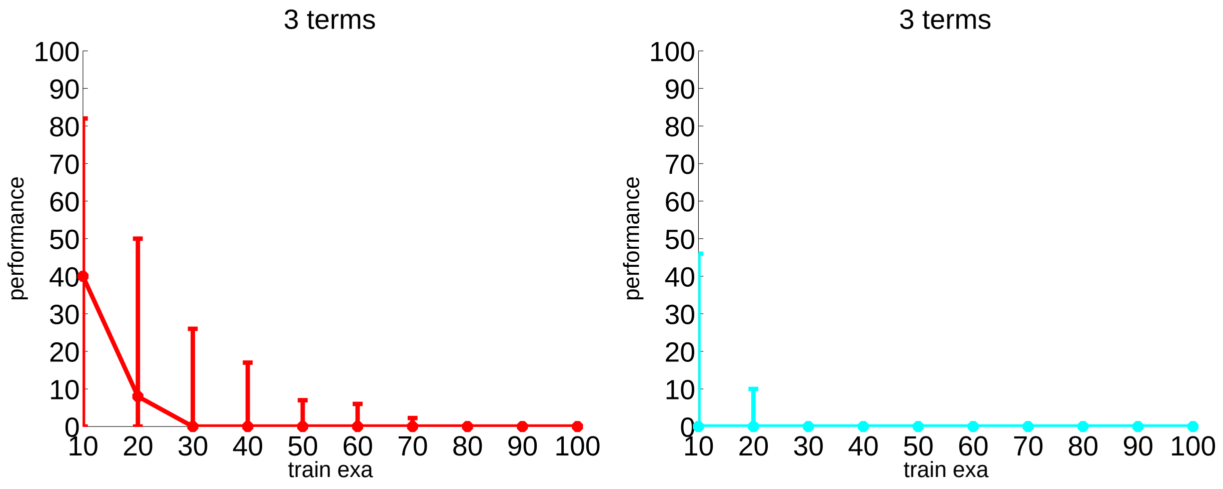

Figure 3.11: Learning curves at the first (left Figure) and the third (right Figure) iterations obtained while solving the Housing problem with true utility function of three terms. Error bars represents the range among the 25th and the 75th percentiles of the measurements.

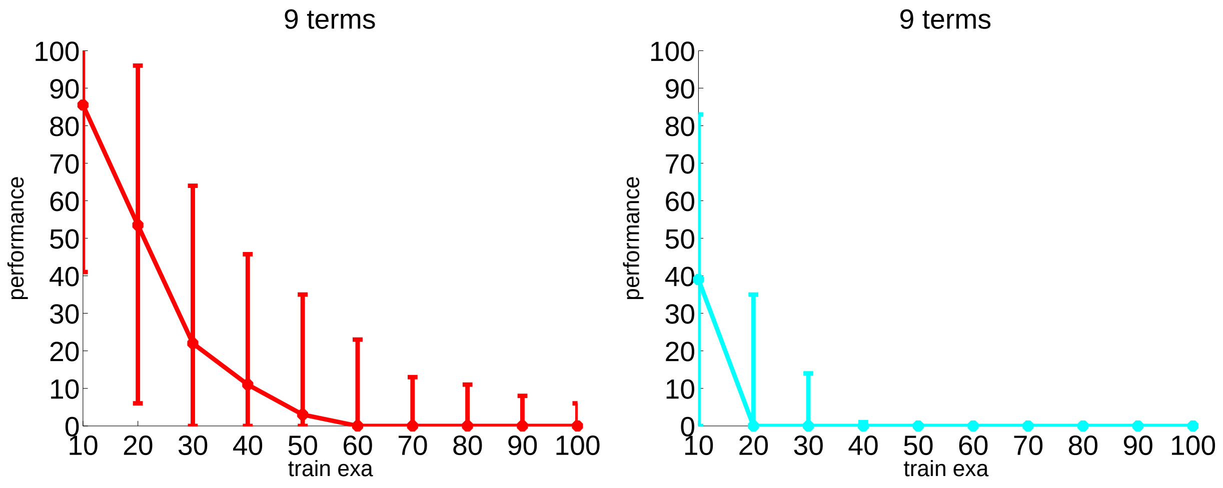

Figure 3.12: Learning curves at the first (left Figure) and the third (right Figure) iterations obtained while solving the Housing problem with true utility function of nine terms. Error bars represents the range among the 25th and the 75th percentiles of the measurements.

fectiveness of our approach in focusing towards the optimal solutions, its robustness as well as its ability to recover from suboptimal initial choices.

The algorithm can be generalized in a number of directions. The availability of a quantitative feedback is not necessarily straightforward, especially when a human DM is involved in the loop. A more affordable request is often that of ranking sets of candidates according to preference. Our setting can be easily adapted to this setting by replacing the squared error loss in the learning stage with appropriate ranking losses. The simplest solution consists of formulating it as correctly ordering each pair of instances as done in support vector ranking, and applying 1-norm SVM [50]. More complex ranking losses have been proposed in the literature (see for instance [11]), especially to increase the importance of correctly ranking the best solutions, and could be combined with 1-norm regularization.

Our experimental evaluation is focused on small-scale problems, typical of an interaction with a human DM. In principle, when combined with appropriate SMT solvers, our approach could be applied to larger real-world optimization problems, whose formulation is only partially available. In this case, a local search algorithm rather than a complete solver will be used during the optimization stage, as showed in the experiments on the weighted MAX-SAT instances. However, the cost of requiring an explicit representation of all possible conjunction of predicates (even if limited to the unknown part) would rapidly produce an explosion of computational and memory requirements. One option is that of resorting to an implicit representation of the function to be optimized, like the one we used in the *Krr* algorithm. Kernelized versions of zero-norm regularization [47] could be tried in order to enforce sparsity in the projected space. However, the lack of an explicit formula would prevent the use of all the efficient refinements of SMT solvers, based on a tight integration between SAT and theory solvers. A possible alternative is that of pursuing an incremental feature selection strategy and iteratively solving increasingly complex approximations of the underlying problem.

Finally, we are currently investigating larger preference elicitation problems, with both known hard constraints limiting the set of feasible solutions and unknown user preferences. This setting allows us to address many real-world scenarios. The promising results for the Housing problem, where the hard constraints define the available house types and locations and the preferences of the DM drive the search within the set of feasible solutions, constitute the first step along this research direction.

Chapter 4

Conclusions and perspectives

This thesis introduces two novel approaches for interactive decision making. The first one is developed within the context of traditional interactive multi-objective optimization, while the second one considers a constraint-based formulation for the user preferences. Both techniques are an instantiation of the “learning to optimize” paradigm coming from Reactive Search Optimization [2], successfully applied in the past for the *online* configuration of relevant parameters for optimization algorithms. The presented approaches implement an incremental algorithm, alternating a search phase using the current approximation of the utility function to generate candidate solutions, and a refinement phase exploiting feedback received to improve the approximation.

In particular, the first technique addresses the problem of handling evolving preferences in interactive multi-objective optimization. To the best of our knowledge, in the context of multi-objective optimization no interactive decision making technique has been explicitly designed to handle a preference model that evolves over time. With the extent of covering this gap, 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 representative 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. In detail, we plan to extend the experimental evaluation including additional benchmark problems with evolving non-linear user preferences, possibly derived from real-world applications. Both sudden and gradual preference drift will be considered. We are currently investigating more complex models for uncertain user preference information, considering the DM fatigue during the interactive process (the probability to obtain an incorrect answer increases with the number of questions) and the level of satisfaction achieved by the DM (inaccurate feedback may increase when the final user is required to evaluate solutions very different from her favourite one).

Our second approach encodes the user preferences in terms of stochastic logic utility functions. It consists of an incremental algorithm employing 1-norm regularization to enforce sparsity of the learned function and a SLS algorithm addressing the weighted MAX-SAT problem resulting from the search phase. Furthermore, we show how to adapt the approach to a large class of relevant optimization problems dealing with satisfiability modulo theories. Experimental results on both weighted MAX-SAT and MAX-SMT problems demonstrate the effectiveness of our technique in focusing towards the optimal solutions, its robustness as well as its ability to recover from suboptimal initial choices. In particular, our realistic problems consider a redundant set of decisional features to choose from, a set of known hard constraints defining feasible configurations and a strong non-linear relationship between the features representing the preference structure of the user. Furthermore, some of the decisional features may be defined as a function of the others, like the price variable in the Housing problem formulation.

The presented algorithm can be generalized in a number of directions. First, the preference drift concept defined in the context of multi-objective optimization can be extended to the constraint-based formulation of user preferences. Second, the availability of a quantitative feedback is not necessarily straightforward, especially when a human DM is involved in the loop. A more affordable request is often that of ranking sets of candidates according to preference. Our algorithm can be easily adapted to this setting by replacing the squared error loss in the learning stage with appropriate ranking losses. Furthermore, while our experimental evaluation is focused on small-scale problems, typical of an interaction with a human DM, we plan to extend our approach to deal with larger real-world optimization problems, whose formulation is only partially available. A typical case would be the automatic configuration of a complex system, which is too costly to be queried for each candidate solution during its optimization process. Finally, we will consider for both our IDM techniques more sophisticated active learning approaches, for example taken from information retrieval, in order to reduce the number of queries to the DM.

Bibliography

[1] C. Barrett, R. Sebastiani, S. A. Seshia, and C. Tinelli. Satisfiability modulo theories. In *Handbook of Satisfiability*, chapter 26, pages 825–885. IOS Press, 2009.

[2] R. Battiti, M. Brunato, and F. Mascia. *Reactive search and intelligent optimization*. Springer Verlag, 2008.

[3] R. Battiti and P. Campigotto. Reactive search optimization: Learning while optimizing. an experiment in interactive multi-objective optimization. In *MIC2009, VIII Metaheuristic International Conference, Hamburg, Germany, July 13-16, 2009*, Lecture Notes in Computer Science. Springer Verlag, 2009.

[4] R. Battiti and A. Passerini. Brain-computer evolutionary multi-objective optimization (BC-EMO): a genetic algorithm adapting to the decision maker. *IEEE Transactions on Evolutionary Computation*, 2010, to appear.

[5] V. Belton, J. Branke, P. Eskelinen, S. Greco, J. Molina, F. Ruiz, and R. Słowiński. Interactive multiobjective optimization from a learning perspective. In *Multiobjective Optimization: Interactive and Evolutionary Approaches*, pages 405–433. Springer-Verlag, Berlin, Heidelberg, 2008.

[6] Stefano Bistarelli, Maria Silvia Pini, Francesca Rossi, and Kristen Brent Venable. From soft constraints to bipolar preferences: modelling framework and solving issues. *J. Exp. Theor. Artif. Intell.*, 22(2):135–158, 2010.

[7] J. Branke, K. Deb, K. Miettinen, and R. Słowiński, editors. *Multiobjective Optimization: Interactive and Evolutionary Approaches*. Springer-Verlag, 2008.

[8] P. Campigotto and A. Passerini. 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.

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

[10] P. Campigotto, A. Passerini, and R. Battiti. Active learning of combinatorial features for interactive optimization. In *LION V: Learning and Intelligent OptimizatioN Conference, Rome, Italy, Jan 17-21, 2011*, Lecture Notes in Computer Science. Springer Verlag, 2011, to appear.

[11] S. Chakrabarti, R. Khanna, U. Sawant, and C. Bhattacharyya. Structured learning for non-smooth ranking losses. In *KDD '08*, pages 88–96, New York, NY, USA, 2008.

[12] O. Chapelle, V.N. Vapnik, O. Bousquet, and S. Mukherjee. Choosing multiple parameters for support vector machines. *Machine Learning*, 46(1–3):131–159, 2002.

[13] A. Cimatti, A. Franzén, A. Griggio, R. Sebastiani, and C. Stenico. Satisfiability modulo the theory of costs: Foundations and applications. In Javier Esparza and Rupak Majumdar, editors, *TACAS*, volume 6015 of *LNCS*, pages 99–113. Springer, 2010.

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

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

[16] K. Deb, A. Pratap, S. Agarwal, and T. Meyarivan. A fast elitist multi-objective genetic algorithm: NSGA-II. *IEEE Transactions on Evolutionary Computation*, 6:182–197, 2000.

[17] K. Deb, L. Thiele, M. Laumanns, and E. Zitzler. Scalable multi-objective optimization test problems. In *Congress on Evolutionary Computation (CEC2002)*, pages 825–830, 2002.

[18] B. Dutertre and L. de Moura. A Fast Linear-Arithmetic Solver for DPLL(T). In *Proceedings of the 18th Computer-Aided Verification conference*, volume 4144 of *LNCS*, pages 81–94. Springer-Verlag, 2006.

[19] J. Friedman, T. Hastie, S. Rosset, and R. Tibshirani. Discussion of boosting papers. *Annals of Statistics*, 32:102–107, 2004.

[20] J. Fürnkranz and E. Hüllermeier. *Preference Learning: An Introduction.* Springer-Verlag, 2010.

[21] Mirco Gelain, Maria Silvia Pini, Francesca Rossi, Kristen Brent Venable, and Toby Walsh. Elicitation strategies for soft constraint problems with missing preferences: Properties, algorithms and experimental studies. *Artif. Intell.*, 174(3-4):270–294, 2010.

[22] I. Guyon, J. Weston, S. Barnhill, and V.N. Vapnik. Gene selection for cancer classification using support vector machines. *Machine Learning*, 46(1–3):389–422, 2002.

[23] H. Z. Huang, Z. G. Tian, and M. J. Zuo. Intelligent interactive multiobjective optimization method and its application to reliability optimization. *IIE Transactions*, 37(11):983–993, 2005.

[24] H. Kaizhu, K. Irwin, and R.L. Michael. Direct zero-norm optimization for feature selection. In *ICDM*, pages 845–850, Los Alamitos, CA, USA, 2008.

[25] I. Kaliszewski. Out of the mist–towards decision-maker-friendly multiple criteria decision making support. *European Journal of Operational Research*, 158(2):293–307, October 2004.

[26] R. Khardon, D. Roth, and R.A. Servedio. Efficiency versus convergence of boolean kernels for on-line learning algorithms. *J. Artif. Int. Res.*, 24(1):341–356, 2005.

[27] R. Klinkenberg and S. Rüping. Concept drift and the importance of examples. In *Text Mining Theoretical Aspects and Applications*, pages 55–77. Physica-Verlag, 2002.

[28] F. Rossi K. B. Venable M. Gelain, M. S. Pini and N. Wilson. Interval-valued soft constraint problems. *Accepted by AMAI - Annals of Mathematics and Artificial Intelligence - Special Issue for ISAIM 2008, Springer, 14 June 2010.*

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

[30] K. Miettinen and M. M. Mäkelä. On scalarizing functions in multiobjective optimization. *OR Spectrum*, 24:193–213, 2002.

[31] K. Miettinen, F. Ruiz, and A.P. Wierzbicki. Introduction to Multiobjective Optimization: Interactive Approaches. In *Multiobjective Optimization: Interactive and Evolutionary Approaches*, pages 27–57. Springer-Verlag Berlin, Heidelberg, 2008.

[32] Leonardo Moura and Nikolaj Bj**o**rner. Satisfiability modulo theories: An appetizer. pages 23–36, 2009.

[33] Greg Nelson and Derek C. Oppen. Simplification by cooperating decision procedures. *ACM Trans. Program. Lang. Syst.*, 1(2):245–257, 1979.

[34] R. Nieuwenhuis and A. Oliveras. Dpll(t) with exhaustive theory propagation and its application to difference logic. In *In CAV05 LNCS 3576*, pages 321–334. Springer, 2005.

[35] R. Nieuwenhuis and A. Oliveras. On sat modulo theories and optimization problems. In *In Theory and Applications of Satisfiability Testing (SAT), LNCS 4121*, pages 156–169. Springer, 2006.

[36] P. Pu and L. Chen. Integrating tradeoff support in product search tools for e-commerce sites. In *Proceedings of the 6th ACM conference on Electronic commerce (EC '05)*, pages 269–278, New York, NY, USA, 2005. ACM Press.

[37] P. Pu and L. Chen. User-involved preference elicitation for product search and recommender systems. *AI magazine*, 29(4):93–103, Winter 2008.

[38] F. Radlinski and T. Joachims. Active exploration for learning rankings from clickthrough data. In *KDD '07: Proceedings of the 13th ACM*

SIGKDD international conference on Knowledge discovery and data mining, pages 570–579, New York, NY, USA, 2007. ACM Press.

[39] J. Schlimmer and R. Granger. Incremental learning from noisy data. *Mach. Learn.*, 1(3):317–354, 1986.

[40] B. Settles. Active learning literature survey. Technical Report Computer Sciences Technical Report 1648, University of Wisconsin-Madison, 2009.

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

[42] C. Suanders, A. Gammerman, and V. Vovk. Ridge regression learning algorithm in dual variables. In *ICML'98*, 1998.

[43] M. Sun, A. Stam, and R.E. Steuer. Solving multiple objective programming problems using feed-forward artificial neural networks: the interactive ffann procedure. *Manage. Sci.*, 42(6):835–849, 1996.

[44] M. Sun, A. Stam, and R.E. Steuer. Interactive multiple objective programming using tchebycheff programs and artificial neural networks. *Comput. Oper. Res.*, 27(7-8):601–620, 2000.

[45] L. Tanner. Selecting a text-processing system as a qualitative multiple criteria problem. *European Journal of Operational Research*, 50(2):179–187, January 1991.

[46] R. Tibshirani. Regression shrinkage and selection via the lasso. *Journal of the Royal Statistical Society, Series B*, 58:267–288, 1996.

[47] Jason Weston, André Elisseeff, Bernhard Schölkopf, and Mike Tipping. Use of the zero norm with linear models and kernel methods. *J. Mach. Learn. Res.*, 3:1439–1461, 2003.

[48] A.P. Wierzbicki. On the completeness and constructiveness of parametric characterizations to vector optimization problems. *OR Spektrum*, 8:73–87, 1986.

[49] Zuobing Xu, Ram Akella, and Yi Zhang. Incorporating diversity and density in active learning for relevance feedback. In *ECIR*, pages 246–257, 2007.

[50] Ji Zhu, Saharon Rosset, Trevor Hastie, and Rob Tibshirani. 1-norm Support Vector Machines. In *Neural Information Processing Systems.* MIT Press, 2003.

[51] I. Žliobaitė. Learning under concept drift: an overview. Technical report, Vilnius University, Faculty of Mathematics and Informatics, 2009.

Appendix A

Expressing bipolar preferences within the soft constraints formalism

The work in [6] extends the soft constraint formalism to handle bipolar preference problems, by introducing a structure with the ability of compensating positive and negative preferences. Compensation among heterogeneous preferences includes also the possibility of expressing "indifference" (i.e., neither positive nor negative preference) for a given solution. In principle, one may think to represent bipolar preferences in terms of a bi-objective optimization problem with positive preferences to be maximized and negative ones to be minimized. In this case, the heterogeneous preferences are considered separately, and a solution is associated with a pair of scores expressing its negative and positive preference levels, respectively. The solutions are ordered by the Pareto optimality criterion, thus two solutions defeating each other on one preference level are incomparable. The lack of compensation between negative and positive preferences and the incomparability among the (large) set of non-dominated solutions reduce the appeal of this approach.

To overcomes these drawbacks, the extension in [6] represents bipolar preferences with two separate algebraic structures. In particular, in addition to a c-semiring $(N, +_n, \times_n, \mathbf{0}_n, \mathbf{1}_n)$ for the negative preferences, the algebraic structure $(P, +_p, \times_p, \mathbf{0}_p, \mathbf{1}_p)$ is used for positive preferences. The additive operator $+_p$ has the same properties as the corresponding one $+_n$ in c-semirings. Therefore, a partial order over P is defined by the relation $\leq_P$ over P, $a \leq_P b$ iff $a{+}b = b$. Opposite to the operator $\times_n$, the absorbing and the unit elements for the operator $\times_p$ are the best $(\mathbf{1}_p)$ and the worst $(\mathbf{0}_p)$ elements in the set, respectively. Furthermore, for each $a, b \in P$ we have $a, b \leq_P a \times_p b$. Therefore, this structure models the desired properties with positive preferences: the combination of positive preferences generate better preference and the value encoding indifference is smaller than all the positive preference levels. The structures for the positive and the negative preferences are then combined in a *bipolar preference structure*. It is a tuple $(N, P, +, \times, \perp, \Box, \perp)$ where:

- $(N, +_n, \times_n, \perp, \Box)$ is a negative preference structure;
- $(P, +_p, \times_p, \Box, \perp)$ is a positive preference structure;
- $+ : (N \cup P)^2 \rightarrow (N \cup P)$ is an operator s.t. $a_n + a_p = a_p$ for any $a_n \in N$ and $a_p \in P$; it induces a partial ordering on $N \cup P$: $\forall a, b \in N \cup P$, $a \leq b$ iff $a + b = b$;
- $\times : (N \cup P)^2 \rightarrow (N \cup P)$ is the compensation operator, commutative and monotone ($\forall a, b, c \in N \cup P$, if $a \leq b$ then $a \times c \leq b \times c$).

Given the order induced by $+$ on $N \cup P$, we have that $\perp \leq \Box \leq \perp$, with $\perp$ and $\perp$ the unique minimum and maximum elements of the bipolar structure. The element $\Box$ is used to model indifference; it is smaller than any positive preference and greater than any negative one. The combination of a positive and a negative preference is a preference which is higher than, or equal to, the negative one and lower than, or equal to, the positive one. The above bipolar structure generalizes both c-semirings and positive preference structures. Furthermore, a different number of levels may be used to express positive and negative preferences and the operator $\times_n$ is not constrained to be equal to $\times_p$.

Appendix B

Additional MAX-SAT experiments

Fig. B.1 and Fig. B.2 depict the performance of the *Lasso* approach with the SLS algorithm in the optimization stage replaced by a complete solver and with the Boolean training examples generation described in Sec. 3.3.2 for the MAX-SMT version of our technique. Henceforth this MAX-SAT variant of the *Lasso* algorithm will be referred as $Lasso_c$. Each point in the graphs is the median value over 400 runs with different random seeds.

In detail, Fig. B.1 contains the learning curves obtained over the same MAX-SAT benchmark considered in the tests of Fig. 3.4. As expected, due to the limited size of the search space, there is not a clear superiority of one approach over the other. With three, four, six and seven terms in the true utility function, $Lasso_c$ converges more rapidly to the gold solution at the third iteration. Considering the cases of eight and nine terms, *Lasso* outperforms $Lasso_c$ over all the observed iterations.

Fig. B.2 contains the learning curves observed for $Lasso_c$ in the case of weighted MAX-SAT problems with 10 known hard constraints. The hard constraints involve both features appearing in the true utility function and features that do not represent any decisional criteria for the user. In particular, the hard constraints include both randomly generated clauses, terms and implications. Each clause and term contains up to three features, while

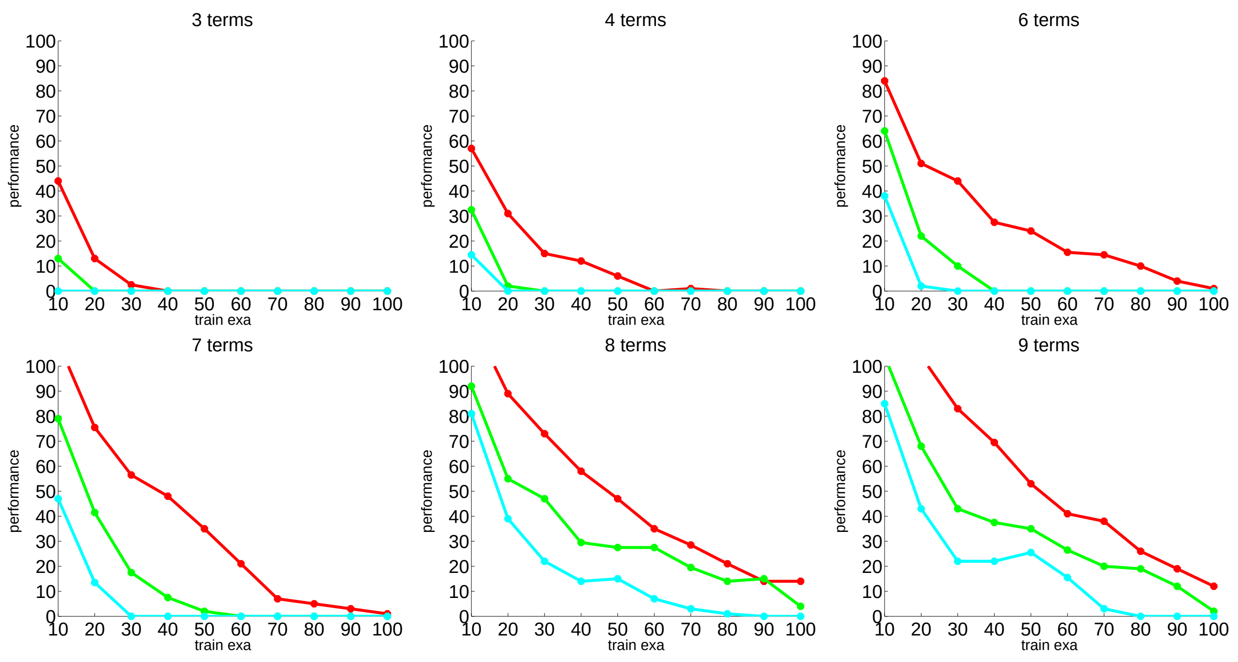

Figure B.1: Learning curves for an increasing number of training examples observed for the $Lasso_c$ approach. The y-axis reports solution quality, while the x-axis contains the number of training examples. Red, green and cyan colors show the performance of the algorithms at the first, second and third iteration, respectively.

the maximum size of the antecedent and the consequent of an implication is four. The comparison of Fig. B.1 with Fig. B.2 shows the faster convergence of our approach when including a known set of hard constraints in the problem definition. For example, with ten hard constraints the cases of three and four terms become a trivial elicitation task for our technique. We argue that the better performance in terms of convergence speed is caused by the reduction of the search space generated by the hard constraints.

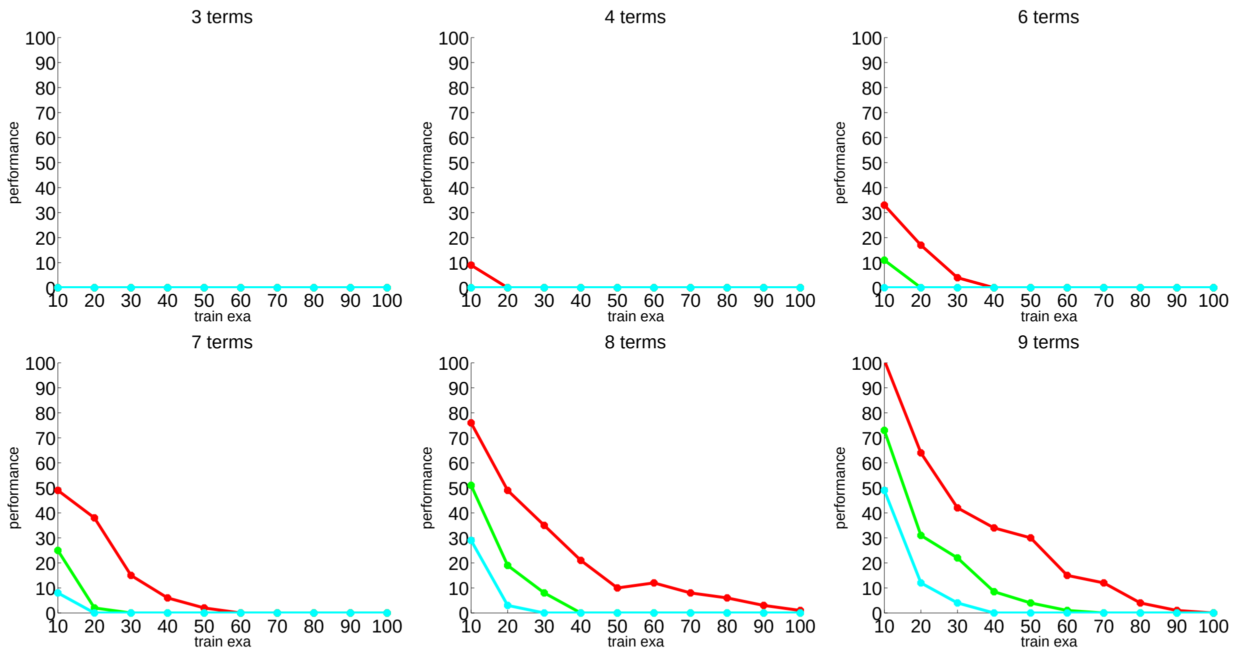

Figure B.2: Learning curves for the $Lasso_c$ approach in the case of weighted MAX-SAT problems with 10 known hard constraints. The data are presented analogously to that in Fig. B.1