Journal of Artificial Intelligence Research () Submitted ; published

Interactive sparse optimization of unknown combinatorial utility functions through machine learning

Paolo Campigotto CAMPIGOTTO@DISI.UNITN.IT
Andrea Passerini PASSERINI@DISI.UNITN.IT
Roberto Battiti BATTITI@DISI.UNITN.IT
DISI - Dipartimento di Ingegneria e Scienza dell'Informazione,
Università degli Studi di Trento,
Via Sommarive 14, I-38123 Povo, TN, Italy

Abstract

This work considers the problem of automatically discovering the solution preferred by a decision maker (DM). Her preferences are formalized as a combinatorial utility function, but they are not fully defined at the beginning and need to be learnt during the search for the satisficing solution. The initial information is limited to a set of catalog features from which the decisional variables of the DM are to be selected.

An interactive optimization procedure is introduced, which iteratively learns an approximation of the utility function modeling the quality of candidate solutions and uses it to generate novel candidates for the following refinement. The source of learning signals is the decision maker, who is fine-tuning her preferences based on the learning process triggered by the presentation of tentative solutions.

The proposed approach focuses on combinatorial utility functions consisting of a weighted sum of conjunctions of Boolean features. 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 of the catalog features up to a certain degree. The optimization stage solves the weighted MAX-SAT problem to generate the novel candidates. The adoption of a Satisfiability Modulo Theory rather than a Satisfiability solver during the optimization stage generalizes our approach to integer and real-valued features. Experiments on realistic problems demonstrate the effectiveness of the approach in focusing towards the optimal solution and its ability to recover from suboptimal initial choices.

1. Introduction

In real world optimization tasks, a significant portion of the problem-solving effort is usually devoted to specifying in a computable manner the function to be optimized. This modeling work consists of modifying and refining the problem definition on the basis of information elicited from the decision maker (DM). Typically, asking a user to quantify her real objectives *a priori*, without seeing any optimization results, is extremely difficult. *Interactive* decision making approaches handle this initial lack of complete knowledge by keeping the user in the loop of the optimization process. They use the information from the DM during the optimization task to guide the search towards the solution preferred by the user.

A paradigmatic case of incomplete problem definition is provided by multi-objective optimization problems, which consider the simultaneous optimal attainment of a set of conflicting objectives. While the DM usually can define the set of desirable objectives, he cannot define their relative importance, the tradeoffs and the proper combination of them

into an overall *utility function*. Several interactive multi-objective algorithms have been proposed to learn the utility function modeling the preferences of the DM (see, e.g., (Branke, Deb, Miettinen, & Słowiński, 2008) for a recent review and (Battiti & Passerini, 2010) for a recent example). They iteratively alternate preference elicitation (decision stage) and solution generation (optimization phase). At each iteration, the DM evaluates the proposed candidate solutions. The preference information obtained is used to refine a model of the DM preferences; new solutions are generated based on the learnt model. By adopting this approach, the DM preferences drive the search process and only a subset of the Pareto optimal solutions needs to be generated and evaluated.

In general, formalizing the user preferences into a mathematical model is not trivial: a model should capture the qualitative notion of preference and represent it as a quantitative function. Let us assume that the candidate solutions of the problem are described by a set of n features $x_1, \ldots, x_n$. The simplest and most used utility model is the *additive* function, where the preference of the DM for the candidate solution $\mathbf{x}$ is given by the sum of sub-utility functions:

$$U(\mathbf{x}) = \sum_{k=1}^{n} u_k(x_k)$$

with each sub-utility function u_k defined on a single feature x_k. Additive utility models are appropriate under the assumption of preferential independence among the set of features (Bacchus & Grove, 1995). Preferential independence exists when the DM preference for the values of a feature x_i does not depend on the fixed values of other features $x_j, j \neq i$. Consider, for example, a costumer of a real estate company articulating her preferences by using only two attributes: the number of bedrooms (x_1) and the distance from the city center (x_2). Her preference over the values of x_1 is the same regardless of the values of x_2, and viceversa: she *always* prefers houses with two bedrooms over houses with one bedroom and she *always* favors the location nearest to the city center.

Additive models fail to capture complex DM preferences including non-linear relationships among the features of the candidate solutions. For example, a customer willing to have a gym in the neighborhood in the case of candidate houses near the city center. When considering houses in the suburbs, the presence of free parking makes houses without a garage attractive (to keep price low) and, similarly, the proximity of green areas providing the opportunity for outdoor sport activities decreases her interest for a gym in the neighborhood. *Generalized additive independence* (GAI) (Bacchus & Grove, 1995) models overcome the limitation of simple additive models by encoding utilities as a sum of sub-utility "overlapping" functions:

$$U(\mathbf{x}) = \sum_{k=1}^{p} u_k(X_k)$$

where $X_k, k = 1 \ldots p$, are subsets of the n features that may be *non*-disjoint.

Recent work in the field of constraint programming (Gelain, Pini, Rossi, Venable, & Walsh, 2010b) formalizes the user preferences in terms of *soft constraints*. 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. Assignments to the variables of a constraint are referred as *partial* assignments and their desirability levels are termed *local* preference

values. *Global* preference scores computed by applying a combination operator to the local preference values define the desirability of a complete assignment (i.e., an assignment to the whole set of the problem variables). 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 global preference levels and by comparing them in the preference order. The work in (Gelain et al., 2010b) introduces an elicitation strategy for soft constraint problems with missing preferences, to find the solution preferred by the decision maker by asking the final user to reveal as few preferences as possible.

In this paper, soft constraints are cast into *weighted Boolean terms*. The DM preferences are represented by combinatorial utility functions which are the weighted combination of the Boolean terms. The optimization task is translated into a *weighted Maximum Satisfiability* (MAX-SAT) problem where the objective function is unknown and has to be interactively learnt. Consider again a real estate company 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 naturally formalized as a weighted MAX-SAT problem, where the constraints are encoded by Boolean terms, each term being the combination of Boolean features. Similarly to the GAI models, this representation of the DM preferences can model the combined effects of multiple non-linearly related decisional features.

This Boolean model for the DM preferences is generalized to more complex utility functions which are combinations of *predicates* in a certain theory of interest. The generalization enables to consider, e.g., integer or real-valued features. It consists of replacing Satisfiability with *Satisfiability Modulo Theory* (Barrett, Sebastiani, Seshia, & Tinelli, 2009) (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. For example, consider the case of flight selection. The predicate $x_1 + x_2 \leq 5$ *hours* defines the preference for a travel duration, calculated as flight duration (x_1) plus transfer time to the departure airport (x_2), smaller than five hours. The predicate $x_3 < 2$ states the desirability for a flight with number of stopovers (x_3) smaller than two. SMT has received increasing attention in recent years, thanks to a number of successful applications in areas like verification systems, planning and model checking. *Optimization Modulo Theory*, also known as "Satisfiability Modulo the Theory of Costs" (Cimatti, Franzn, Griggio, Sebastiani, & Stenico, 2010), extends SMT by considering optimization problems. Rather than checking for the existence of a satisfying assignment as in SMT, the target is a satisfying assignment that minimizes a given cost function.

This paper introduces a method for the joint learning and optimization of the DM preferences which handle the representations based on Satisfiability and Satisfiability Modulo Theory. It 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.

Unlike the work (Gelain et al., 2010b), our method does not assume to know in advance the decisional features of the user and their detailed combination. In the paper (Gelain et al.,

2010b) soft constraint topology and structure is assumed to be known and the incomplete information consists of missing local preference values only. The initial amount of knowledge required by our approach is limited to a set of "catalog" features from which the decisional variables of the DM are selected. The limited initial knowledge translates in a *sparse* unknown utility function: only a fraction of the catalog features represent the decisional items of the DM and only a subset of the possible terms constructed from them defines the DM utility function. Furthermore, our method can handle uncertain, inconsistent and contradictory preference information from the final user, which characterizes many human decision processes.

This paper introduces the first approach combining learning, interactive optimization and SMT. A preliminary version of our technique is described in (Campigotto, Passerini, & Battiti, 2011). This work considers a different form of feedback from the DM, uses a complete rather than a local search Satisfiability solver, and extends the experimental studies. The acquisition of the preference information from a human DM, characterized by limited patience and bounded rationality, is indeed a crucial issue. Although providing explicit weights and mathematical formulas is in general prohibitive for the DM, he can definitely evaluate the returned solutions. In (Campigotto et al., 2011) quantitative judgments are asked to the DM. However, asking the final user for 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 $\mathbf{x}'$ to solution $\mathbf{x}''$ ", rather than in specifying how much they prefer $\mathbf{x}'$ over $\mathbf{x}''$. In order to reduce the embarrassment of the decision maker when specifying precise preference scores, in this paper the evaluation by the user consists of just comparing and ranking candidate solutions. Furthermore, the experiments include a realistic problem whose MAX-SAT formulation results in an significant blow-up in the formula size, while it can be efficiently encoded into a MAX-SMT optimization task.

The organization of the paper is as follows. Section 2 introduces the algorithm for the SAT case. Section 3 introduces SMT and Optimization Modulo Theory and shows how to adapt our algorithm to this setting. Related work is discussed in Section 4. Experimental results in Section 5 on both weighted MAX-SAT and MAX-SMT problems demonstrate the effectiveness of our approach in focusing towards the optimal solutions, its robustness and its ability to recover from suboptimal initial choices. A discussion including potential research directions concludes the paper.

2. Overview of our approach

Candidate configurations are n dimensional Boolean vectors $\mathbf{x}$ consisting of *catalog* features. *A priori* knowledge of the problem is limited to the set of catalog features. The unknown combinatorial utility function expressing the DM preferences is the weighted combination of Boolean terms generated from the catalog features and has to be jointly and interactively learned during the optimization process. Furthermore, the optimal utility function is complex enough to prevent exhaustive enumeration of possible solutions. 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.

```
1.  procedure interactive_sparse_optimization
2.      input: set of the catalog variables
3.      output: learnt utility function and configuration optimizing it
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.          /* Utility function learning phase */
9.          Based on D, select terms and relative weights for current
10.         weighted MAX-SAT formulation (Eq. 2);
11.         /* Optimization phase */
12.         Get s/2 configurations by optimizing current weighted MAX-SAT
13.         formulation;
14.         Get evaluation of new configurations and add them to D;
15.     return configuration optimizing the learnt weighted MAX-SAT formulation
```

Figure 1: Pseudocode for the interactive optimization algorithm. The parameter s defines the number of examples to be evaluated by the DM at the different iterations.

Our method consists of an iterative procedure alternating a utility function learning phase with a search phase. At each step, the current approximation of the utility function, represented as a weighted MAX-SAT problem, is used to guide the search for optimal configurations. A subset of candidate configurations is obtained by solving the weighted MAX-SAT problem (*search phase*). Preference information is required for these candidates, and the utility model is refined according to the feedback received (*learning phase*). A set of randomly generated examples is employed to initialize the utility model. The pseudocode of our algorithm is in Fig. 1.

The number of training iterations does not need to be fixed at the beginning. The DM may ask for an additional iteration by comparing the recovered candidates with her own preferences. Furthermore, the number of candidates to be evaluated at each iteration is arbitrary. In the settings used here, we use s training examples at the first iteration and $s/2$ examples at each following iteration. A larger number of training instances is suggested at the first iteration to stimulate the preference expression of the DM, as discussed later in this Section, and to generate a good initial model. The following iterations generate training examples distributed in the promising regions of the search space. The goal of our approach is indeed the identification of the solution preferred by the user (*learning to optimize*) rather than an accurate global approximation of the DM utility function (*learning per se*). Therefore, only the shape of the utility function locally guiding the search to the correct direction is actually needed. This requires a shift of paradigm with respect to standard machine learning strategies, in order to model the relevant areas of the optimization fitness surface rather than reconstruct it entirely. The initialization, and the learning and search phases of our approach are explained below.

2.1 Initializing the algorithm

A set of random examples is generated to approximate the DM utility function at the first iteration. Each Boolean feature is assigned a truth value independently and uniformly at random. The evaluation of diverse examples stimulates the preference expression, especially when the user is still uncertain about her final preference (Pu & Chen, 2008). 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 about the preference for an intermediate airport until one solution suggests an airplane change in a place she dislikes (Pu & Chen, 2008).

2.2 Learning an approximation of the utility function

The refinement of the utility model consists of learning the weights of the terms, discarding the terms with zero weight. It includes both the selection of the relevant features from the catalog set and the learning of their detailed combination from the space of all possible conjunction up to certain degree d. To identify sparse solutions, we adopt 1-norm rather than 2-norm regularization. 1-norm regularization indeed provides an embedded feature selection capability, favouring sparse solutions (Tibshirani, 1996). Features selection is crucial to maximize the learning accuracy with data sets characterized by redundant and irrelevant features (Friedman, Hastie, Rosset, & Tibshirani, 2004). The "*Least absolute shrinkage and selection operator*" (Lasso) (Tibshirani, 1996) is a popular method for regression tasks which uses 1-norm regularization to achieve a sparse solution. Let $\mathcal{D} = (\mathbf{x}_i, y_i)_{i=1...m}$ be the set of m training examples, where $\mathbf{x}_i$ contains the feature values of the i-*th* example and y_i represents its score value. The loss+penalty formulation of Lasso regression is defined by the following optimization problem:

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

where $\mathbf{w}$ is the weight vector and the mapping function Φ projects the input vector to a higher dimensional space. The square loss function $\sum_{i=1}^{m}(y_i - \mathbf{w}^T \cdot \Phi(\mathbf{x}_i))^2$ measures the empirical risk as the sum of the squared training errors. The penalty function consists of 1-norm regularization, favouring automated feature selection. The regularization parameter λ trade-offs the penalty and loss terms.

In our context, the score value y_i represents the quantitative evaluation of the decision maker for the solution $\mathbf{x}_i$. The preference scores are taken from a predefined ordered set expressing the desirability levels for the candidate solutions. The mapping function Φ projects sample vectors to the space of all possible conjunctions up to d Boolean variables. Note that dealing with the explicit projection Φ in Eq. 1 is tractable only for a rather limited number of catalog features and size of conjunctions d. However, this will typically be the case when interacting with a human DM. The bounded rationality of humans indeed allows them to handle non-linear interactions just among a small number of features.

Rather than resorting to the above regression problem as in (Campigotto et al., 2011), here the learning of the utility function is cast into a ranking problem. This approach enables a much more affordable task for the decision maker, who is now just required to compare solutions rather than to assign preference scores.

Given a set of m candidate solutions $(\mathbf{x}_1, \ldots, \mathbf{x}_m)$ and an order relation $\succ$ such that $\mathbf{x}_i \succ \mathbf{x}_j$ if $\mathbf{x}_i$ should be preferred to $\mathbf{x}_j$, a *ranking* $(y_1, \ldots, y_m)$ of the m solutions can be specified as a permutation of the first m natural numbers such that $y_i < y_j$ if $\mathbf{x}_i \succ \mathbf{x}_j$. Learning to rank consists of learning a ranking function r from a datasets $\mathcal{D} = \{(\mathbf{x}_1^{(i)}, \ldots, \mathbf{x}_{m_i}^{(i)}), (y_1^{(i)}, \ldots, y_{m_i}^{(i)})\}_{i=1}^{s}$, of sets with their desired rankings. The ranking function r associates each candidate instance with a real number, with the aim that $r(\mathbf{x}_i) > r(\mathbf{x}_j) \iff \mathbf{x}_i \succ \mathbf{x}_j$. In this way, it provides an ordering that agrees with the observed training examples. Given the ranking dataset $\mathcal{D}$, the 1-norm regularization formulation can be adapted to learn the ranking function r by imposing constraints on the correct pairwise ordering of solutions within a set: $r(\mathbf{x}_h^{(i)}) > r(\mathbf{x}_k^{(i)}) \iff y_h^{(i)} < y_k^{(i)}$. In the case of support vector machines, $r(x) = \mathbf{w} \cdot \Phi(x)$ and $r(\mathbf{x}_i) > r(\mathbf{x}_j) \iff \mathbf{w} \cdot (\Phi(\mathbf{x}_i) - \Phi(\mathbf{x}_j)) > 0$. The resulting loss+penalty formulation for 1-norm support vector machine can be written as:

$$\min_{\mathbf{w} \in \mathbb{R}} \sum_{i=1}^{s} \sum_{h_i, k_i, y_{h_i}^{(i)} < y_{k_i}^{(i)}} [1 - \mathbf{w} \cdot (\Phi(\mathbf{x}_{h_i}^{(i)}) - \Phi(\mathbf{x}_{k_i}^{(i)}))]_+ + \lambda ||\mathbf{w}||_1 \tag{2}$$

where where subscript "+" indicates the positive part. The first term is the so-called empirical Hinge Loss, which measures the empirical risk by observing the inconsistencies in the ranking, i.e., the cases where a less preferred solution is ranked higher, and the cases when the pairwise ranking is correct but the difference between the ranking values $r(\mathbf{x}_i)$ and $r(\mathbf{x}_j)$ is smaller than the support vector machine margin, i.e., $0 < \mathbf{w} \cdot (\Phi(\mathbf{x}_i) - \Phi(\mathbf{x}_j)) < 1$. Note that the formulation in Eq. 2 allows to naturally account for ties and partial rankings, as constraints are only included whenever two examples should be ranked differently.

The learnt function $\hat{f}(\mathbf{x}) = \mathbf{w}^* \cdot \Phi(\mathbf{x})$, where $\mathbf{w}^*$ is the solution of the ranking problem, will be used as the novel approximation of the utility function f of the DM.

2.3 Optimizing the learnt utility function

The learnt utility function $\hat{f}$ expressing the approximation of the DM preferences is represented as a weighted MAX-SAT problem. The set of novel candidate solutions to be ranked by the DM is obtained by applying a complete solver over the weighted MAX-SAT problem. The size of the weighted MAX-SAT problem is indeed bounded by the limited cognitive capabilities of the human DM. The adoption of local search techniques for large scale problems will be discussed in Sec. 6. The MAX-SAT solver returns an optimizer for the input problem, i.e., the configuration $\mathbf{x}^*$ maximizing the weighted sum of the terms representing $\hat{f}$. In order to obtain multiple candidate solutions, we optimize again $\hat{f}$ with the additional *hard* constraint generated by the disjunction of all the terms of $\hat{f}$ unsatisfied by $\mathbf{x}^*$. Unlike the weighted terms, which may or may not be satisfied, *hard* constraints do not have a weight value and have to be satisfied. For example, let t_1 and t_5 be the terms of $\hat{f}$ unsatisfied by $\mathbf{x}^*$, then the hard constraint becomes:

$$(t_1 \vee t_5)$$

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

$$(\neg x_1^* \vee \neg x_2^* \ldots \vee \neg x_n^*)$$

which excludes $\mathbf{x}^*$ from the feasible solutions set of $\hat{f}$. The generation of the candidate solutions is iterated until the desired number of configurations have been generated or the hard constraints generated made the MAX-SAT problem unsatisfiable.

Finally, the DM will rank the new candidate solutions based on her preferences and the ranking information will be included in the training examples set for the following refinements of $\hat{f}$. The mechanism creating the training examples is motivated by the trade-off between the selection of good solutions (w.r.t. the current approximation $\hat{f}$ of utility function f) and the diversification of the search process.

3. Satisfiability Modulo Theory

In the previous section, we assumed our optimization task could be cast into a *propositional* Satisfiability problem. However, the formalism of plain propositional logic is not suitable or expressive enough to represent many real-world applications, arising, for example, in the fields of real-time control system design and formal software verification. For example, software verification applications need reasoning about equalities, arithmetic operations and data structures. These problems require or are more naturally described in more expressive theories as first-order logic (FOL), involving quantifiers, functions and predicates. Consider the toy example represented by the following formula from the theory of arithmetic over integers:

$$x + y + z \leq 4,\ x, y, z \in \{1, 2, 3\}$$

We are interested in deciding whether there is an assignment of integer values to the variables x, y and z satisfying the formula. A possible encoding into an equisatisfiable SAT proposition is given by:

$$(x_1 \wedge y_1 \wedge z_1) \vee (x_1 \wedge y_1 \wedge z_2) \vee (x_1 \wedge y_2 \wedge z_1) \vee \ldots$$

where x_i, y_i, z_i, with $i = 1, 2, 3$, is a Boolean variable which is true when the integer-valued variable x, y, z is assigned the value i, respectively. Note the blow-up in the translation affecting the SAT problem size.

Satisfiability Modulo Theory (SMT) problems generalize SAT problems by considering the satisfiability of a FOL formula with respect to a certain *background theory* T fixing the interpretation of (some of the) predicate and function symbols. Any procedure designed to solve a SMT problem is called SMT solver. 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. 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. Other theories exist for data structures such as lists and strings.

Different approaches have been developed to solve SMT problems. When deciding the satisfiability of a first-order formula φ in a given theory T, 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, e.g., the standard setting in inductive logic programming when verifying whether a certain hypothesis covers an example given the available background knowledge. Note that, whenever the cost of including such additional background theory is affordable, our optimization algorithm can be applied rather straightforwardly. Unfortunately, adding all axioms of T is not viable for many theories of interest: consider for instance the theory of *arithmetic*, which defines the interpretation of symbols such as $+, \geq, 0, 5$.

An alternative approach is represented by *eager* SMT solvers. They translate the original formula φ taken from the input theory T into an *equisatisfiable* propositional formula *in a single step*. In this way, any off-the-shelf SAT solver can be used to check the satisfiability of the generated propositional formula. The SAT solver is called once. However, a specific translator has to be developed for each theory of interest. Furthermore, the translation of *all* the theory-specific information is required at the *beginning* of the search process (hence the name "eager"), possibly resulting in large SAT formulas. Although optimizations in the translation to reduce the size of the SAT problem are possible, there is a trade-off between the degree of optimization and the time required by the SAT encoding. Note the analogy with compilers, optimizing the low-level object code (the SAT formula in our context) generated from a high-level program (the SMT problem formulation) (Barrett et al., 2009).

A more efficient approach is based on incremental translations and calls to the SAT solver. This is the case of *lazy* SMT solvers, where the theory-specific information is incrementally encoded in the SAT formulation of the problem. In particular, the lazy approach exploits *specialized* reasoning methods for the background theory of interest. When integrated as submodules in a SMT solver, these theory-specific reasoning methods are often referred as *T-solvers*. T-solvers are efficient decision procedures typically developed to check the satisfiability of *conjunctions* of literals (i.e., atomic formulas and their negations) over the given theory T. The generalization to arbitrary propositional structures is handled in conjunction with the SAT solver integrated in the SMT solver.

In this work we focus on lazy SMT solvers and their optimization variant, which we integrated in our optimization algorithm. The rest of this Section provides details about lazy SMT solvers search process and introduces Optimization Modulo Theory (OMT) solvers handling weighted Maximum Satisfiability Modulo Theory (MAX-SMT) problems. Finally, the integration of MAX-SMT solvers in our optimization approach is discussed.

3.1 Lazy Satisfiability Modulo Theory solvers

The search process of a lazy SMT solver alternates calls to the Satisfiability and the theory solver respectively, 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 the arithmetic theory over integers:

$$x + y + z \leq 3 \wedge (x \leq y \vee z = 2) \wedge (x \geq 2 \vee x \neq z) \tag{3}$$

where x, y, z are integer-valued variables. Then, $p_1 = \alpha(x + y + z \leq 3)$, $p_2 = \alpha(x \leq y)$, $p_3 = \alpha(z = 2)$, $p_4 = \alpha(x \geq 2)$ and $p_5 = \alpha(x \neq z)$. The resulting propositional formula $\alpha(\varphi)$

is:

$$p_1 \wedge (p_2 \vee p_3) \wedge (p_4 \vee p_5)$$

Note that the truth assignment

$$p_1 = \top, p_2 = \bot, p_3 = \top, p_4 = \top, p_5 = \bot$$

where $\top, \bot$ symbols encode true and false truth values, respectively, is equivalent to the statement

$$x + y + z \leq 3 \wedge x > y \wedge z = 2 \wedge x \geq 2 \wedge x = z$$

in the theory T.

The SAT solver integrated in the SMT solver searches a solution of the propositional formula $\alpha(\varphi)$. If the propositional formula in unsatisfiable, the original formula φ is also unsatisfiable and the whole SMT solver stops. Otherwise, the SAT solver provides a truth assignment satisfying $\alpha(\varphi)$. Considering the above example, it may be:

$$p_1 = \top, p_2 = \top, p_3 = \top, p_4 = \bot, p_5 = \top \tag{4}$$

The T-sover is used to validate the assignment (i.e., the conjunction of truth values) produced by the SAT solver. The predicates are evaluated using the rules of the theory T. If the validation is successful, the SMT solver stops returning the assignment of values to the variables in T satisfying φ. Otherwise, when the T-solver detects unsatisfiability, an additional constraint explaining (i.e, justifying) the unsatisfiability is included in $\alpha(\varphi)$ and the SAT solver is asked for a new assignment. For example, the assignment in Eq. 4 is not valid in the arithmetic theory over integers. Applying the inverse mapping β, we obtain:

$$x + y + z \leq 3 \wedge x \leq y \wedge z \neq 2 \wedge x \geq 2 \wedge x \neq z$$

which is clearly unsatisfiable. A possible justification explaining the unsatisfiability is given by the following constraint:

$$\neg(p_1 \wedge p_2 \wedge p_3 \wedge p_5)$$

which will be included in the propositional formula $\alpha(\varphi)$ for the following calls to the SAT solver. Continuing with the toy example, assume that the second call to the SAT solver returns the following truth assignment:

$$p_1 = \top, p_2 = \bot, p_3 = \top, p_4 = \bot, p_5 = \top \tag{5}$$

satisfying the above justification. Restoring the interpretation of the propositional variables p_i, we obtain:

$$x + y + z \leq 3 \wedge x > y \wedge z = 2 \wedge x < 2 \wedge x \neq z$$

which is satisfied by posing, e.g., $x = 1, y = 0, z = 2$.

The term *lazy* denoting this approach is due to the incremental strategy generating constraints on demand. On the contrary, eager methods produce all the constraints in a single step before the execution of the SAT solver.

Figure 2 reports the basic form (De Moura & Bjorner, 2009) of a SMT algorithm. `SAT`(φ) calls the SAT solver on the φ instance, returning a pair (r, M), where r is `sat` if

```
1. procedure SMT solver(φ)
2.     φ' = α(φ)
3.     while (true)
4.         (r,M) ← SAT(φ')
5.         if r = unsat then return unsat
6.         (r,J) ← T-Solver(β(M))
7.         if r = sat then return sat
8.         C ← ⋁_{l∈J} ¬α(l)
9.         φ' ← φ' ∧ C
```

Figure 2: Pseudocode for a basic lazy SMT solver.

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.

Off-the-shelf solvers introduce a number of refinements to this basic strategy, by pursuing a tighter integration between the SAT and the theory solvers. A common approach consists of pruning 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 (Nelson & Oppen, 1979) for a basic procedure.

3.2 Weighted MAX-SMT and Optimization Modulo Theory solvers

Weighted MAX-SMT generalizes SMT problem much like weighted MAX-SAT does with SAT ones. Given a cost function c, an assignment $\mathbf{s}$ in the input theory T is sought with minimum $c(\mathbf{s})$. The simplest formulation for the MAX-SMT problem consists of assigning a weight to each part (i.e., constraint) of the formula to be jointly satisfied. Weights represent penalties or costs for violating the constraints and are expressed by positive natural or real numbers. The cost function $c(\mathbf{s})$ is defined by the sum of the weights of the constraints unsatisfied under the assignment $\mathbf{s}$.

While a body of works exist addressing weighted MAX-SAT problems, MAX-SMT task has been tackled only recently and very few solvers have been developed (Dutertre & de Moura, 2006; Nieuwenhuis & Oliveras, 2006; Cimatti et al., 2010). Let $\{(C_1, w_1), \ldots, (C_m, w_m)\}$ the set of the weighted constraints, where C_i and w_i identify the i-*th* constraint and its associated weight, respectively. Clearly, the cost function c take values from the interval $[0, W]$, with $W = w_1 + \cdots + w_m$. Identifying a solution $\mathbf{s}$ with cost $c(\mathbf{s}) = W$ is trivial.

A popular approach (Nieuwenhuis & Oliveras, 2006) to obtain solutions with cost at most $\bar{W}$, with $\bar{W} < W$, consists of generating m additional constraints $(C_i \vee p_i)$, $i = 1 \ldots m$, with p_i a fresh propositional variable. The initial background theory T is augmented with

the integers and with the following assertions:

$$p_i \rightarrow (k_i = w_i),\ i = 1 \dots m$$
$$\neg p_i \rightarrow (k_i = 0),\ i = 1 \dots m$$
$$k_1 + \dots + k_m \leq \bar{W}$$

In this way, the SAT solver identifies a solution with cost at most $\bar{W}$ (if any). The T-solver is then called to check whether the truth assignment returned by the SAT solver is valid in the input theory T. By considering decreasing upper bound values $\bar{W}$ the solution with minimum cost can be recovered.

3.3 Integration of MAX-SMT solver in our framework

The extension of our optimization algorithm to handle MAX-SMT problems is straightforward. It consists of replacing the MAX-SAT with the MAX-SMT solver. Our framework does not need to be changed, as the effort required to handle non-Boolean encodings is completely performed by the MAX-SMT solver.

When representing user preferences in the SMT setting, the DM utility function f is 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* 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 $\mathbf{p}^* = (p_1^*, \dots, p_n^*)$ of Boolean values ($p_i^* = \{\mathbf{true}, \mathbf{false}\}$) to the predicates in S.

The diversification strategy to obtain multiple candidates solutions is the same as described in Sec. 2.3. The sequential optimization of $\hat{f}$ is performed, with the additional *hard* constraint generated by the disjunction of all the terms of $\hat{f}$ unsatisfied by $\mathbf{p}^*$. If $\mathbf{p}^*$ satisfies all the terms of $\hat{f}$, the additional *hard* constraint consists of:

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

which excludes $\mathbf{p}^*$ from the feasible solutions set of $\hat{f}$.

4. Related work

Recent work in the field of constraint programming (Gelain et al., 2010b) shares with our technique the combinatorial approach to model user preferences. It defines the user preferences in terms of *soft* constraints and introduces constraint optimization problems where the DM preferences are not completely known before the solving process starts. In this section, we first define the c-semiring formalism modeling soft constraints (Gelain et al., 2010b) and then show how a semiring-based soft constraint satisfaction problem can be translated into a weighted MAX-SAT problem (Leenen, Anbulagan, Meyer, & Ghose, 2007). Finally, we compare the preference elicitation task considered in the paper (Gelain et al., 2010b) with our problem settings.

4.1 Semiring-based soft constraint satisfaction problem

In soft constraints, a generalization of hard constraints, each assignment to the variables of one constraint is associated with a preference value taken from a preference set. The preference value represents the level of desirability of the assignment to the variables of the constraint. As the preference score is associated to a partial assignment to the problem variables, it represents a *local* preference value. The desirability of a complete assignment is defined by a *global* preference score, 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 global preference levels. Soft constraints are 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 $\mathbf{1}$ as absorbing element, the operation $\times$ must be commutative. The relation $\leq_A$ over A, $a_2 \leq_A a_1$ iff $a_2 + a_1 = a_1$, 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 $a_2 \leq_A a_1$ meaning that a_1 is "better" than a_2; $\mathbf{0}$ and $\mathbf{1}$ represent the worst and the best preference levels, respectively, and the operations $+$ and $\times$ are monotone on $\leq_A$. Consider, e.g., the following instance of c-semiring:

$$(\{5, 10, 15, \ldots, 50\}, \max, \min, 5, 50)$$

with preference values from the set $\{5, 10, 15, \ldots, 50\}$ and elements $\mathbf{0}$ and $\mathbf{1}$ represented by the values 5 and 50, respectively. The desirability of a complete assignment is obtained by taking its minimum local preference value. A complete assignment c_1 with preference score a_1 is preferred to a complete assignment c_2 with lower preference score a_2. That is, $a_2 \leq_A a_1$ iff $\max(a_2, a_1) = a_1$.

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 permits 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 (Bistarelli, Pini, Rossi, & Venable, 2010).

The work in (Leenen et al., 2007) encodes a semiring-based soft constraint satisfaction problem (SCSP) into a weighted MAX-SAT problem, ensuring that each solution of the latter problem corresponds to a solution of the former one. With no loss of generality, assume a soft constraint problem with n variables $v_1, \dots v_n$ having domain $D_1, \dots D_n$, and m constraints $c_1, \dots c_m$. Each instantiation of the variables of a constraint c_j, $j = 1 \dots m$, is associated with a value from the c-semiring $(A, +, \times, \mathbf{0}, \mathbf{1})$. For each variable v_i, $i = 1 \dots n$, and each value $d \in D_i$, a Boolean variable $b_{i,d}$ is introduced. When $b_{i,d}$ is set to true then v_i is assigned the value $d \in D_i$. The variables $b_{i,d}$, $i = 1 \dots n$, $d \in D_i$, represent the Boolean variables of the weighted MAX-SAT problem.

The set of Boolean constraints of the MAX-SAT problem consists of clauses ensuring that each variable v_i, $i = 1 \dots n$, is assigned exactly one value $d \in D_i$, and of terms representing the soft constraints of the original SCSP. In the former case, for each variable $v_i, i = 1 \dots n$, the *at-least-one-value* hard clause:

$$(b_{i,d_1} \vee b_{i,d_2} \vee \dots \vee b_{i,d_{|D_i|}})$$

and the set of $(|D_i|\dot{(}D_i| - 1))/2$ binary *at-max-one-value* hard clauses:

$$(\neg b_{i,d_j} \vee \neg b_{i,d_k}) \text{ for every pair } (d_j, d_k) \text{ with } d_j, d_k \in D_i \text{ and } 1 \leq j < k \leq |D_i|$$

are generated. They ensure that for each $i \in \{1 \dots n\}$ exactly one variable $b_{i,j}$, $j \in \{1, 2, \dots, |D_i|\}$ is set to true.

Each soft constraint of the original SCSP is represented by a set of weighted Boolean terms encoding all the possible assignments of values (i.e., configurations) to its variables. The weight of a term is set to the c-semiring value associated to the encoded configuration. For example, consider a binary soft constraint over variables v_1 and v_2 both with discrete domain $D = \{1, 2, 3\}$ and with preference scores defined by the semiring $(\{5, 10, 15, \dots, 50\}, \max, \min, 5, 50)$. The possible configurations are specified in Table 1 (left). Each row shows an assignment of values to v_1 and v_2 and the c-semiring value associated to the assignment. Given the six Boolean variables $b_{1,d}$ and $b_{2,d}$ with $d = 1, 2, 3$ defined as above, the soft constraint in Table 1 (left) is encoded into the set of Boolean terms in Table 1 (right).

A structured MAX-SAT formulation can be obtained by considering generalized Boolean clauses which are the disjunction of the terms encoding for a given soft constraint the assignments with the same preference value. For example, the terms defined at rows number $1, 5, 9$ in Table 1 (right) can be merged into a single generalized weighted clause:

$$(b_{1,1} \wedge b_{2,1}) \vee (b_{1,2} \wedge b_{2,2}) \vee (b_{1,3} \wedge b_{2,3})$$

with weight equal to 10. Furthermore, each *at-least-one-value* and *at-max-one-value* hard clause h can be cast into a soft clause represented by its negation $\neg h$ and with associated the semiring value $\mathbf{0}$ (Leenen et al., 2007). The value $\mathbf{0}$ is indeed both the minimum value in the partial order defined by the relation $\leq_A$ and the absorbing element for the operator $\times$ combining the semiring values. Therefore, a candidate solution $\mathbf{b}$ of the generated MAX-SAT problem that does not satisfy one of these soft clauses receives the minimum semiring value $\mathbf{0}$. However, this implementation of the hard clauses does not allow to discern infeasible solutions from feasible ones with lowest possible preference, i.e., feasible solutions getting the lowest semiring value.

v_1	v_2	preference value
1	1	10
1	2	40
1	3	50
2	1	5
2	2	10
2	3	30
3	1	5
3	2	5
3	3	10

num	term	weight
1	$(b_{1,1} \wedge b_{2,1})$	10
2	$(b_{1,1} \wedge b_{2,2})$	40
3	$(b_{1,1} \wedge b_{2,3})$	50
4	$(b_{1,2} \wedge b_{2,1})$	5
5	$(b_{1,2} \wedge b_{2,2})$	10
6	$(b_{1,2} \wedge b_{2,3})$	30
7	$(b_{1,3} \wedge b_{2,1})$	5
8	$(b_{1,3} \wedge b_{2,2})$	5
9	$(b_{1,3} \wedge b_{2,3})$	10

Table 1: (left) example of soft constraint. The DM prefers assignments with $v_1 < v_2$. (Right) weighted Boolean terms encoding the soft constraint defined in the left Table. When the Boolean variable $b_{i,d}$, $d \in D_i$, is set to true then v_i is assigned the value d.

Given the generated MAX-SAT formulation, the optimization task consists of finding the assignment $\mathbf{b}^*$ to the Boolean variables $b_{i,d}$, $i = 1 \dots n$, $d \in D_i$, maximizing $f(\mathbf{b})$, with $f(\mathbf{b})$ the semiring value obtained by combining by the operator $\times$ the weights of the solution components satisfied by $\mathbf{b}$. Each candidate solution $(\mathbf{b}, f(\mathbf{b}))$ of the generated MAX-SAT problem identifies an assignment of values to the variables v_i, $i = 1 \dots n$, of the original SCSP with associated semiring value $f(\mathbf{b})$.

4.2 Comparing preference models based on soft constraints with our problem settings

The work in (Gelain et al., 2010b) introduces an elicitation strategy for soft constraint problems with missing preferences to find the solution preferred by the DM while asking her to reveal as few preferences as possible. Despite the common purpose, this approach is different from ours. A major difference is the preference elicitation problem considered. In (Gelain et al., 2010b), the DM preferences are expressed using the c-semiring formalism. Decision variables, soft constraint topology and structure are assumed to be known in advance and the incomplete initial information consists only of missing local preference values. On the other hand, our investigation considers a much more limited amount of initial knowledge. We assume complete ignorance about the structure of the constraints over the decisional variables of the user. The initial problem knowledge is limited to a set of catalog features. Our algorithm extracts the decisional items of the DM from the set of catalog features and learns the weighted terms constructed from them modeling the DM preferences. Consider the translation of a SCSP into the weighted MAX-SAT formulation defined in the previous Section. The translation of the SCSP with missing preferences introduced in (Gelain et al., 2010b) results in a Boolean formula where some of the weights of the terms are not known, while our preference model consists of a MAX-SAT problem,

where both the terms and their associated weights are initially unknown and are learnt by interacting with the DM.

Furthermore, the technique in (Gelain et al., 2010b) 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: it asks the user to compare complete solutions and learns local utilities (i.e., the weights of the terms of the logic formula) 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. Note that the DM is asked for quantitative evaluations of partial solutions: she does not just rank couples of activities, she provides score values quantifying her preference for the partial activity rankings, a much more demanding task.

In order to reduce the embarrassment of the decision maker when specifying precise preference scores, interval-valued constraints (Gelain, Pini, Rossi, Venable, & Wilson, 2010a) 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 (Gelain et al., 2010a) 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 (Gelain et al., 2010a) have been introduced to handle uncertainty in the evaluations of the DM, the experiments in (Gelain et al., 2010b) do not consider the case of inconsistent preference information. Our technique is robust to imprecise information from the DM, modeled in terms of inaccurate ranking of the candidate solutions. On the other hand, the optimality of the solutions produced by the technique in (Gelain et al., 2010b) is guaranteed and the empirical studies show that the algorithm, when working on fuzzy constraint satisfaction problems with missing preferences, 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 a good approximation of the optimal solution.

Furthermore, while the work in (Gelain et al., 2010b) 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 provide 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 (Bistarelli et al., 2010) extends the soft constraint formalism to account for bipolar preference problems.

Finally, while the technique in paper (Gelain et al., 2010b) combines branch and bound search with preference elicitation, the paper (Gelain, Pini, Rossi, Venable, & Walsh, 2011) introduces an approach based on stochastic local search and test it on a subset of the benchmark problems considered in (Gelain et al., 2010b). Our approach also works with both incomplete and complete search techniques (Campigotto et al., 2011).

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 (Dutertre & de Moura, 2006). Each point of the curves depicting our results is the median value over 400 runs with different random seeds, unless otherwise stated.

5.1 Weighted MAX-SAT

Our algorithm was 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 DM unknown utility function (henceforth the *target* utility function).

The number of catalog features is 40. The maximum size of terms is assumed to be known. Furthermore, the probability of inaccurate ranking examples swapping the correct positions of the solutions is fixed to the value 0.1.

We run a set of experiments for $10, 20, \ldots 100$ initial training examples. Fig. 3 reports 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 target utility function. Its quality is measured as the approximation error w.r.t. the gold solution. Each point of the curves in the Fig. 3 is the median values over 400 runs with different random seeds. Considering the simplest problems with three and four terms, our algorithm can identify the solution preferred by the DM at the first iteration, provided that at least 90 initial training examples are available. With more than four terms in the target utility function, the optimal solution cannot be recovered at the first iteration. However, our algorithm succeeds in exploiting its active learning strategy and converges to the optimal solution when enough iterations are provided. For instance, in the case of eight terms in the target utility function, the optimal solution is discovered at the second and third iterations with 50 and 90 initial training examples, respectively.

Fig. 4 and Fig. 5 show the learning curves for our approach at the first and third iterations in the case of target utility functions with three and nine terms, respectively. Error bars indicate the range between the 25th and 75th percentiles of the underlying data distributions. As expected, in both cases the sample percentiles demonstrate a more stable behavior of our technique at the third iteration. In particular, at the third iteration the stability of our algorithm increases with additional training examples, while the variability of performance observed at the first iteration does not decrease to the same extent. Considering

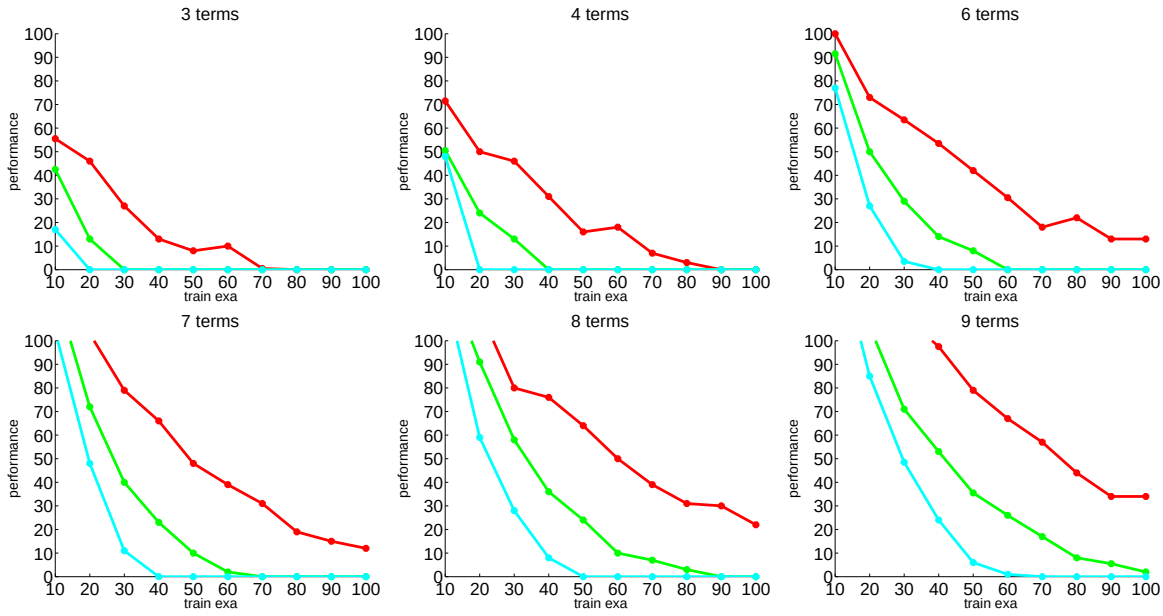

Figure 3: Learning curves for an increasing number of training examples observed for our algorithm at different iterations. The y-axis reports the solution quality, while the x-axis contains the number of training examples. Red, green and cyan colors show the performance of the algorithms at the first, the second and the third iteration, respectively. See text for details.

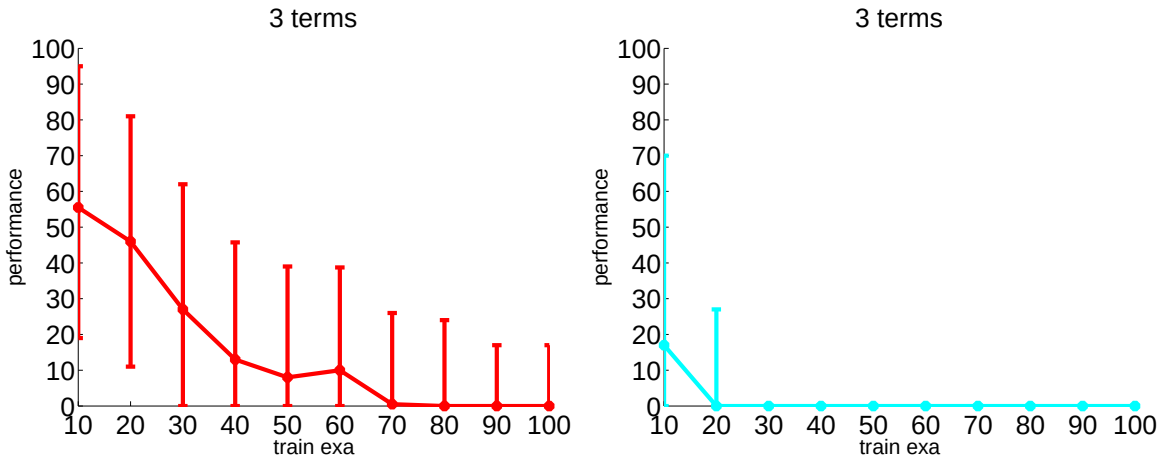

Figure 4: Learning curves at the first (left Figure) and the third (right Figure) iterations in the case of target utility functions with three terms. Error bars denote the range among the 25th and the 75th percentiles of the measurements.

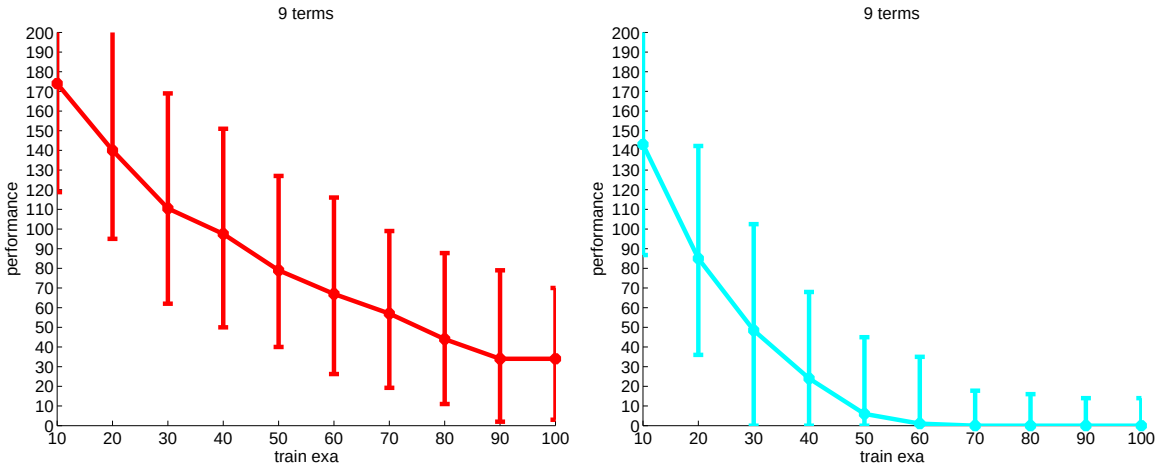

Figure 5: Learning curves at the first (left Figure) and the third (right Figure) iterations in the case of target utility functions with nine terms. Error bars denote the range among the 25th and the 75th percentiles of the measurements.

the more challenging case represented by target utility functions with nine terms, at the third iteration our algorithm consistently finds the gold solution with at least 70 examples (the interquartile range value is within the 20 units). An unstable behavior is still observed at the first iteration even in the case of 100 training examples.

5.2 Weighted MAX-SMT

MAX-SMT is a recent research area. Even if existing results (Nieuwenhuis & Oliveras, 2006) indicate that MAX-SMT solvers can efficiently address real-world problems, to the best of our knowledge no well established publicly available MAX-SMT benchmarks exist and preference elicitation tasks have not been encoded into MAX-SMT problems yet.

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 time. Each job has a fixed known duration, the constraints define the overlap of two jobs or their non-concurrent execution. The target 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 target utility function. The temporal constraints are expressed by using the difference arithmetic theory. In detail, let s_i and d_i, with $i = 1 \dots 5$, 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 target 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 probability of inaccurate ranking examples swapping the correct positions of the solutions is fixed to the value 0.1.

Fig. 6 depicts the performance of our algorithm for the cases of 3, 4, 6, 7, 8, 9 terms in the target 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. 2).

As expected, the learning problem becomes more challenging for increasing number of terms. However, the results for the scheduling problem are promising: our approach identifies the gold standard solution in all cases. In detail, less than 60 examples are required to identify the gold solution at the second iteration. At the third iteration our algorithm needs at most 40 training examples for convergence to the gold solution.

The plots in Fig. 7 and Fig. 8 show that at the third iteration our approach finds the gold solution consistently in the case of three terms target utility functions. Considering nine terms target utility functions, provided that at least 60 initial examples are used the value of the 75th percentile is within 20 units from the median value. 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 planning to build her own house and judging potential housing locations provided by a real estate company (henceforth the *housing* problem). There are different locations available, characterized by different housing values, prices, constraints about the design of the building (e.g., usually 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

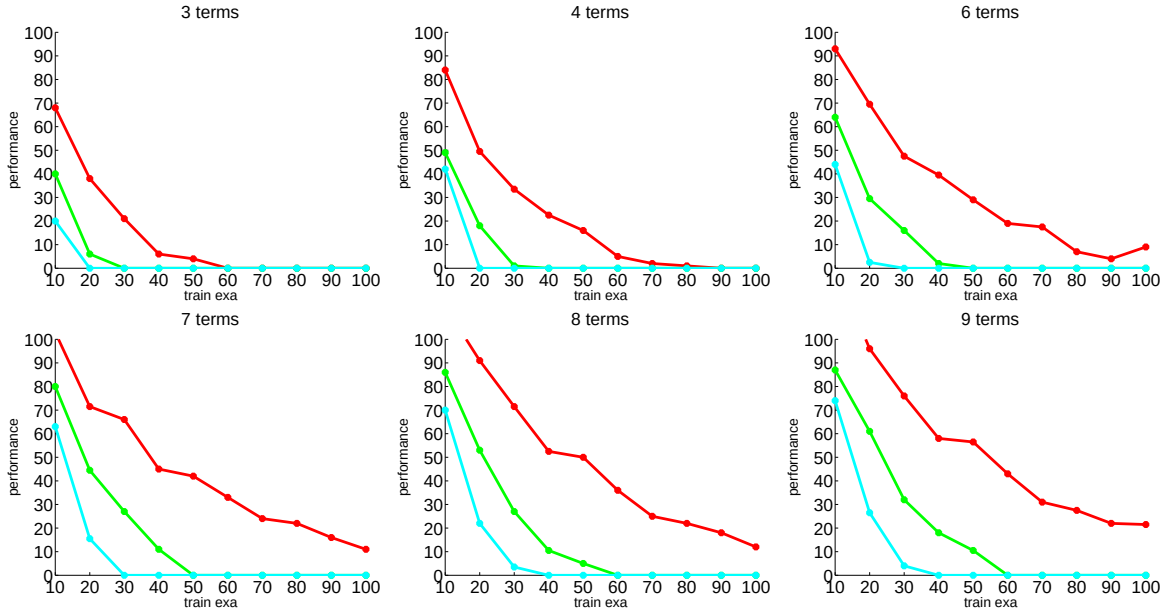

Figure 6: Learning curves observed at different iterations of our algorithm while solving the scheduling problem. The y-axis reports the solution quality, while the x-axis contains the number of training examples. Red, green and cyan colors show the performance of the algorithm at the first, the second and the third iteration, respectively. See text for details.

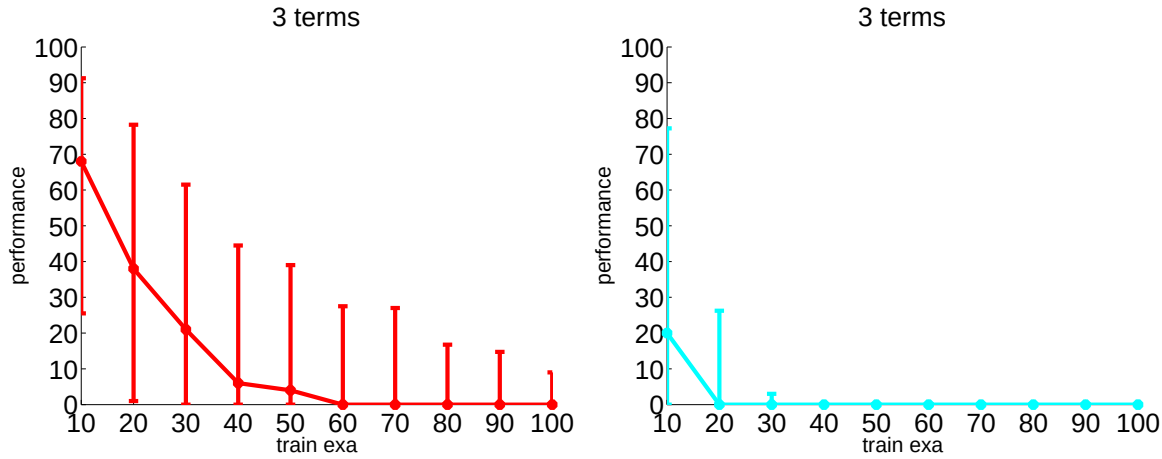

Figure 7: Learning curves at the first (left Figure) and the third (right Figure) iterations observed while solving the scheduling problem with three terms in the target utility function. Error bars denote the range among the 25th and the 75th percentiles of the measurements.

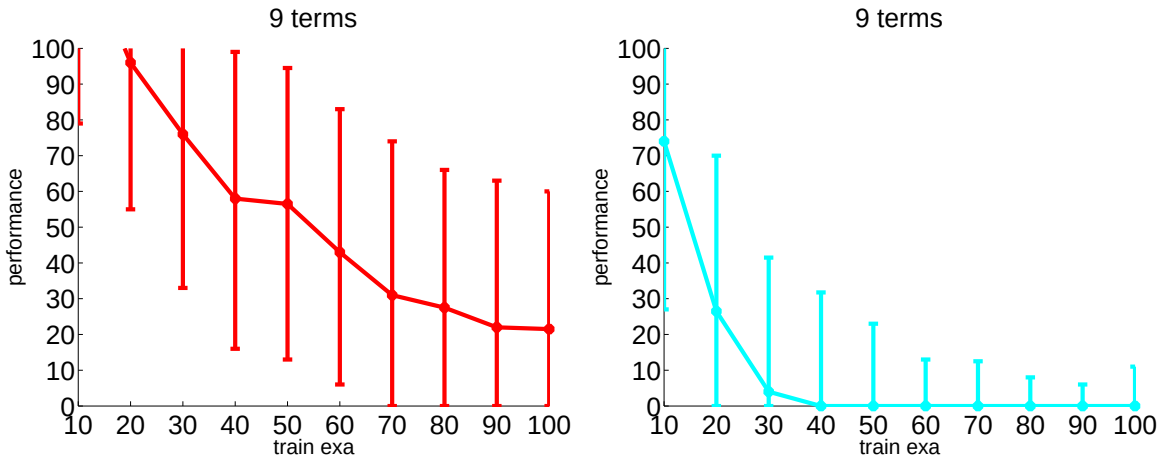

Figure 8: Learning curves at the first (left Figure) and the third (right Figure) iterations for scheduling problems with target utility functions of nine terms. Error bars denote the range among the 25th and the 75th percentiles of the measurements.

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 to the definition of the user preferences cannot be quantified in advance. Many of them may be redundant, as they do not represent any decisional criterion 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 satisfactory solutions 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 cases, a complete and precise formulation of the user preferences cannot be elicited 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 compli-

cate 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 solutions and optimize them for the customer. On the other side, the application of the preference elicitation technique introduced in (Gelain et al., 2010b) 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 Table 2. A set of 10 hard constraints (Table 3) defining feasible

Table 2: Decisional features for the housing problem.

num	feature	type
1	house type	ordinal
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

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 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 number $2, 3, \ldots, 6$ in Table 2. 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 target utility function is composed of terms with two or three predicates, with at least one term with three predicates. Term weights

Table 3: hard constraints for the housing problem. Parameters ts_i, $i = 1 \dots 13$, are threshold values specified by the user or by the sales personnel.

num	hard constraint
1	price $\leq ts_1$
2	location-based taxes and fees $\leq ts_2$ => *not* public green ares in the neighborhood *and not* public transit service quality index $\leq ts_3$
3	commercial facilities in the neighborhood => *not* (garden *and* garage)
4	crime rate $\leq ts_4$ => distance from downtown $\geq ts_5$
5	distance from working place + distance from parents house $\geq ts_6$
6	distance from working place + distance from high schools $\geq ts_7$
7	distance from parents house + distance from high schools $\geq ts_8$
8	distance from nearest free parking $\leq ts_9$ => *not* public green areas in the neighborhood
9	distance from parents house $\leq ts_{10}$ => distance from downtown $\geq ts_{11}$ *and* crime rate $\geq ts_{12}$
10	garden => house type $\geq ts_{13}$

are integer values selected uniformly at random in the range [1, 100]. Inaccurate preference information can be due to occasional inattention of the DM which with probability 0.1 swaps the correct positions of the solutions in the ranking examples.

Fig. 9 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. 10 shows the ability of the algorithm to converge to a stable result while solving Housing problems with target utility functions of three terms. The more challenging elicitation task represented by target utility functions with nine terms (Fig. 11) indicates that running three iterations of the algorithm results in less variability than unstable performance observed at the first iteration, confirming the effectiveness of our incremental approach.

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. We introduced a generic framework, enabling the adoption of off-the-shelf learning methods and MAX-SMT solvers. 1-norm regularization is employed to enforce sparsity of the learned function. The DM is asked to rank the solutions optimizing the generated weighted MAX-SMT problem. Thanks to the

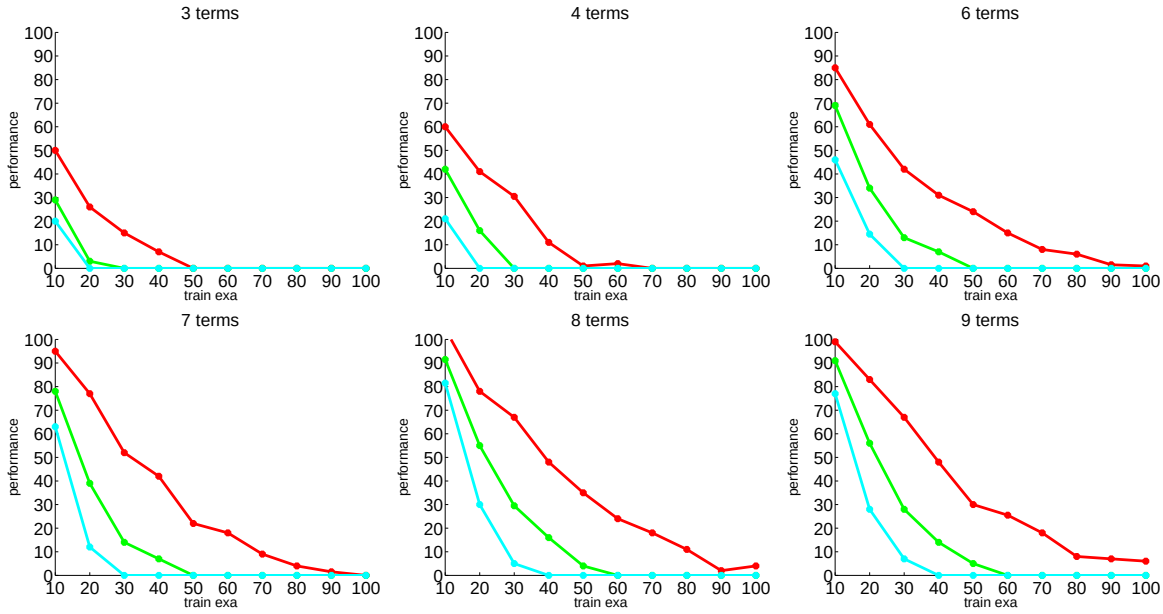

Figure 9: Learning curves observed at different iterations of our algorithm while solving the Housing problem. The data are presented analogously to that in Fig. 6.

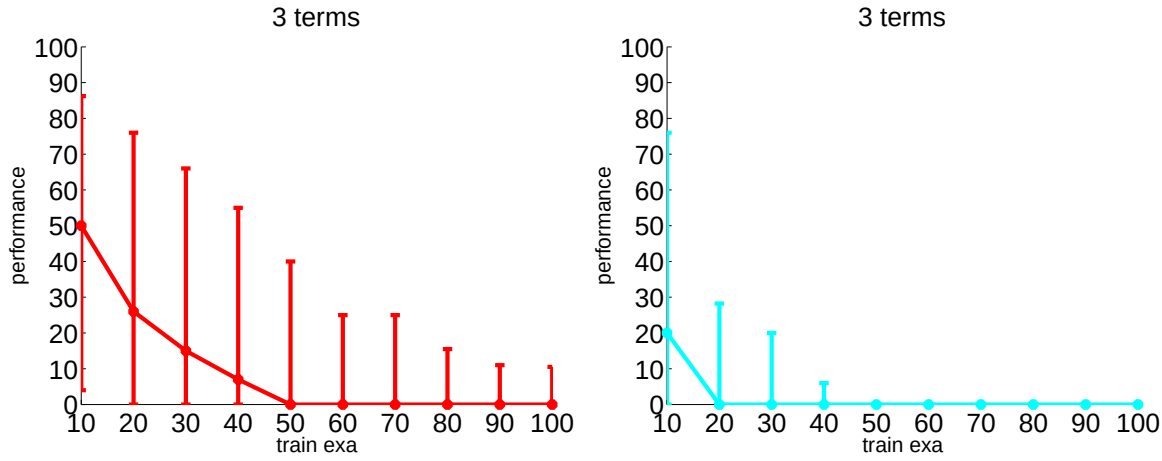

Figure 10: Learning curves at the first (left Figure) and the third (right Figure) iterations obtained while solving the Housing problem with target utility function of three terms. Error bars represents the range among the 25th and the 75th percentiles of the measurements.

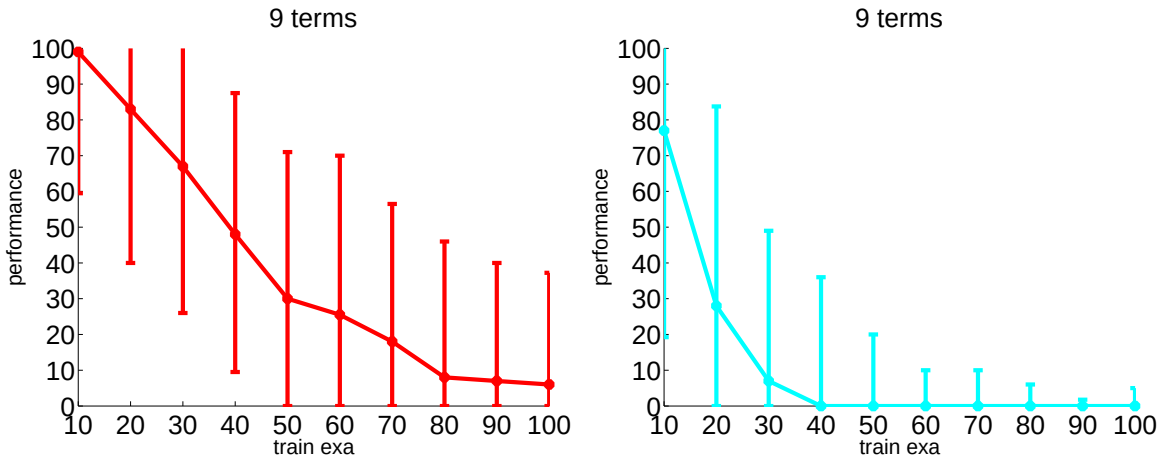

Figure 11: Learning curves at the first (left Figure) and the third (right Figure) iterations obtained while solving the Housing problem with target utility function of nine terms. Error bars represents the range among the 25th and the 75th percentiles of the measurements.

MAX-SMT formalism, our approach can handle a large class of relevant optimization tasks. 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 suboptimal initial choices. Our tests include a preference elicitation task with both known hard constraints limiting the set of feasible solutions and unknown user preferences. In the housing problem the hard constraints define the available house locations and the preferences of the DM drive the search within the set of feasible solutions.

The adoption of 1-norm regularization for the formulation of the learning problem requires that the input catalog features are explicitly projected in the space of all possible Boolean terms which can be generated by their combination. A possible alternative consisting of directly learning a non-linear function of the features, without explicitly projecting them to the resulting higher dimensional space, has been considered in (Campigotto et al., 2011). The alternative approach is based kernel ridge regression (Saunders, Gammerman, & Vovk, 1998), where 2-norm rather than 1-norm regularization is used. As expected, the experimental results show the superior performance of 1-norm regularization over 2-norm regularization in a setting with many irrelevant noisy features, due to the sparsity-inducing property of 1-norm regularization.

The algorithm introduced in this work can be generalized in a number of directions. The learning stage is based on support vector ranking, with the ranking loss function formulated as correctly ordering each pair of instances. More complex ranking losses have been proposed in the literature (see for instance (Chakrabarti, Khanna, Sawant, & Bhattacharyya, 2008)), especially to increase the importance of correctly ranking the best solutions, and could be combined with 1-norm regularization.

Active learning is a hot research area and a broad range of different approaches has been proposed (see (Settles, 2009) for a review). The simplest and most 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 (Branke et al., 2008) and information retrieval (Radlinski & Joachims, 2007). The need to trade off multiple requirements in this active learning setting is addressed in (Xu, Akella, & Zhang, 2007) where the authors consider relevance, diversity and density in selecting candidates. Our future research will consider the application of these active learning techniques. The performance of our method indeed depends on the tradeoff between the identification of candidates solutions satisfying the DM (i.e., solutions optimizing the current learnt preference model) and the generation of *informative* training examples for the following refinement of the learnt model.

In this paper the 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. 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. An option consists of resorting to an implicit representation of the function to be optimized, like the one we used in the version of our algorithm based on kernel ridge regression (Campigotto et al., 2011). Kernelized versions of zero-norm regularization (Weston, Elisseeff, Schölkopf, & Tipping, 2003) 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.

References

Bacchus, F., & Grove, A. (1995). Graphical models for preference and utility. In *Proceedings of the Eleventh Conference on Uncertainty in Artificial Intelligence (UAI-95)*, pp. 3–10. Morgan Kaufmann.

Barrett, C., Sebastiani, R., Seshia, S. A., & Tinelli, C. (2009). Satisfiability modulo theories. In *Handbook of Satisfiability*, chap. 26, pp. 825–885. IOS Press.

Battiti, R., & Passerini, A. (2010). Brain-computer evolutionary multiobjective optimization: A genetic algorithm adapting to the decision maker. *IEEE Transactions on Evolutionary Computation*, *14*(5), 671–687.

Bistarelli, S., Pini, M. S., Rossi, F., & Venable, K. B. (2010). From soft constraints to bipolar preferences: modelling framework and solving issues. *Journal of Experimental*

and Theoretical Artificial Intelligence, *22*(2), 135–158.

Branke, J., Deb, K., Miettinen, K., & Słowiński, R. (Eds.). (2008). *Multiobjective Optimization: Interactive and Evolutionary Approaches*. Springer Verlag.

Campigotto, P., Passerini, A., & Battiti, R. (2011). Active learning of combinatorial features for interactive optimization. In *LION V: Learning and Intelligent OptimizatioN Conference, Rome, Italy, Jan 17-21, 2011*, LNCS. Springer Verlag.

Chakrabarti, S., Khanna, R., Sawant, U., & Bhattacharyya, C. (2008). Structured learning for non-smooth ranking losses. In *14th ACM SIGKDD international conference on Knowledge discovery and data mining*, KDD '08, pp. 88–96. ACM.

Cimatti, A., Franzn, A., Griggio, A., Sebastiani, R., & Stenico, C. (2010). Satisfiability modulo the theory of costs: Foundations and applications. In Esparza, J., & Majumdar, R. (Eds.), *Tools and Algorithms for the Construction and Analysis of Systems*, Vol. 6015 of *LNCS*, pp. 99–113. Springer.

De Moura, L., & Bjorner, N. (2009). Satisfiability modulo theories: An appetizer. In *Formal Methods: Foundations and Applications*, LNCS, pp. 23–36. Springer.

Dutertre, B., & de Moura, L. (2006). A Fast Linear-Arithmetic Solver for DPLL(T). In *18th Computer-Aided Verification conference* , LNCS, pp. 81–94. Springer.

Friedman, J., Hastie, T., Rosset, S., & Tibshirani, R. (2004). Discussion of boosting papers. *Annals of Statistics*, *32*, 102–107.

Gelain, M., Pini, M., Rossi, F., Venable, K., & Wilson, N. (2010a). Interval-valued soft constraint problems. *Annals of Mathematics and Artificial Intelligence*, *58*, 261–298.

Gelain, M., Pini, M. S., Rossi, F., Venable, K. B., & Walsh, T. (2010b). Elicitation Strategies for Soft Constraint Problems with Missing Preferences: Properties, Algorithms and Experimental Studies. *Artificial Intelligence Journal*, *174*(3-4), 270–294.

Gelain, M., Pini, M. S., Rossi, F., Venable, K. B., & Walsh, T. (2011). A local search approach to solve incomplete fuzzy csps. In *Proceedings the 3rd International Conference on Agents and Artificial Intelligence (ICAART 2011)*.

Leenen, L., Anbulagan, Meyer, T., & Ghose, A. K. (2007). Modeling and solving semiring constraint satisfaction problems by transformation to weighted semiring max-sat. In *Proc. of the 20th Australian Joint Conference on Artificial Intelligence*, pp. 202–212.

Nelson, G., & Oppen, D. C. (1979). Simplification by cooperating decision procedures. *ACM Transactions on Programming Languages and Systems*, *1*(2), 245–257.

Nieuwenhuis, R., & Oliveras, A. (2006). On sat modulo theories and optimization problems. In *In Theory and Applications of Satisfiability Testing*, LNCS, pp. 156–169. Springer.

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

Radlinski, F., & Joachims, T. (2007). Active exploration for learning rankings from clickthrough data. In *13th ACM SIGKDD international conference on Knowledge discovery and data mining (KDD '07)*, pp. 570–579. ACM Press.

Saunders, C., Gammerman, A., & Vovk, V. (1998). Ridge regression learning algorithm in dual variables. In *In Proceedings of the 15th International Conference on Machine Learning*, pp. 515–521. Morgan Kaufmann.

Settles, B. (2009). Active learning literature survey. Tech. rep. Computer Sciences Technical Report 1648, University of Wisconsin-Madison.

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

Weston, J., Elisseeff, A., Schölkopf, B., & Tipping, M. (2003). Use of the zero norm with linear models and kernel methods. *Journal of Machine Learning Research*, *3*, 1439–1461.

Xu, Z., Akella, R., & Zhang, Y. (2007). Incorporating diversity and density in active learning for relevance feedback. In Amati, G., Carpineto, C., & Romano, G. (Eds.), *Advances in Information Retrieval*, Vol. 4425 of *LNCS*, pp. 246–257. Springer.