Preference elicitation for interactive learning of Optimization Modulo Theory problems

Paolo Campigotto Andrea Passerini Roberto Battiti
DISI - Dipartimento di Ingegneria e Scienza dell'Informazione,
Università degli Studi di Trento,
Via Sommarive 14, I-38123 Povo, TN, Italy
campigotto,passerini,battiti@disi.unitn.it

1 Introduction

We consider the problem of automatically discovering the solution preferred by a decision maker (DM). Recent work in the field of constraint programming [8] formalizes the user preferences in terms of *soft constraints*, whose weights are assumed to be partially unknown. An elicitation strategy is introduced based on the DM queries for constraint weights. In this paper, soft constraints are cast into *weighted Boolean terms*. The DM preference is represented by a combinatorial utility function which is a weighted combination of 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.

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* [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. 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. *Optimization Modulo Theory*, also known as "Satisfiability Modulo the Theory of Costs" [7] or MAX-SMT in short, 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 work introduces a method for learning DM preferences casting the task into jointly learning and optimizing an unknown MAX-SMT problem. To the best of our knowledge, this is the first approach combining learning and SMT. A preliminary version of our technique is described in [5]. Unlike the approach in [8], our technique does not assume to know in advance the decisional features of the user and their detailed combination. 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 is translated into 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. Unlike previous works on preference elicitation [8, 3], focusing on consistent feedback and optimality guarantees, our method can handle uncertain, inconsistent and contradictory preference information from the final user, which characterizes many human decision processes.

```
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.      Select s configurations uniformly at random;
6.      Ask the DM for the ranking of the s configurations;
7.      initialize training set D by the ranking of the s configurations;
8.      /* Refinement phase */
9.      while (termination_criterion)
10.         /* Utility function learning phase */
11.         Based on D, select terms and relative weights for current
12.         weighted MAX-SAT formulation (Eq. 1);
13.         /* Optimization phase */
14.         Get s/2 new configurations by optimizing current weighted MAX-SAT
15.         formulation;
16.         Ask the DM for the ranking of the new s/2 configurations and add it to D;
17.     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 compared by the DM at the different iterations. Training set D is formed by the partial rankings of candidate solutions generated at the initialization phase and at the different iterations of the algorithm.

2 Overview of our approach

For the sake of simplicity, we introduce the method in the MAX-SAT formulation and later generalize to the MAX-SMT case. 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 learnt 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.

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 instance, is used to guide the search for optimal configurations. A subset of candidate configurations is obtained by solving the weighted MAX-SAT instance (*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 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 conjunctions up to certain degree d.

The learning of the utility function is cast into a ranking problem. To identify sparse solutions, we adopt 1-norm regularization. Given a datasets $\mathcal{D} = \{(\mathbf{x}_1^{(i)}, \ldots, \mathbf{x}_{m_i}^{(i)}), (y_1^{(i)}, \ldots, y_{m_i}^{(i)})\}_{i=1,\ldots,s}$, of s partial rankings, where $(y_1^{(i)}, \ldots, y_{m_i}^{(i)})$ defines the i-*th* ranking, the learning problem is formulatd as:

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

where where subscript "+" indicates the positive part. 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.

The learnt function $\hat{f}(\mathbf{x}) = \mathbf{w}^T \cdot \Phi(\mathbf{x})$, 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. Function $\hat{f}$ is represented as a weighted MAX-SAT instance. The set of novel candidate solutions to be ranked by the DM is obtained by applying a complete solver over the weighted MAX-SAT instance. The MAX-SAT solver returns an optimizer for the input instance, 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 $\hat{f}$, 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 instance 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 tradeoff 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 Computational complexity and user cognitive load

The cognitive capabilities of the humans when making decisions limit the number n of catalog features and the size d of the Boolean terms. Therefore, the learning phase (problem 1) is accomplished in a negligible amount of time (w.r.t. the user response time). Analogous observation holds for the computational effort required by the search phase. Proposing a query consists of generating c candidates to be ranked. Each candidate is obtained by a run of the complete MAX-SAT solver. Even if the search space size is 2^n, the bounded value of n and the efficient performance of modern SAT solvers, that can manage problems with thousands of variables and millions of clauses, enable the completion of the search phase in a negligible amount of time. Finally, the estimation of the cognitive load of the DM is $O(c \log c)$, with c being the number of candidates to be ranked. In the experiments reported here, the value of c used in the refinement phase varies from 5 to 50. Keeping c low limits the computational effort of the DM. As matter of fact, the DM typically prefers to rank small batches of good quality solutions rather than a single large batch of candidates, including also lower quality ones. Therefore, a rule of thumb for the algorithm configuration consists of keeping the value of c low and increasing the number of refinement iterations.

4 Optimization Modulo Theory

In the previous sections, we assumed our optimization task could be cast into a *propositional* Maximum-Satisfiability problem. However, the formalism of plain propositional logic is not suitable or expressive enough to represent many real-world applications. These problems require or are more naturally described in more expressive theories as first-order logic (FOL), involving quantifiers, functions and predicates. *Optimization Modulo Theory* problems generalize MAX-SAT problems by considering the maximum-satisfiability of a FOL formula with respect to a certain *background theory* T fixing the interpretation of (some of the) predicate and function symbols.

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 MAX-SAT solver integrated in the MAX-SMT one finds a Boolean assignment to the predicates maximizing the learnt utility function. The Theory-sover is then used to validate this assignment using the rules of the theory T. Validation amounts at finding values for the theory variables consistent with the truth assignment of the predicates. If the validation is successful, the SMT solver stops, returning these variable values. Otherwise, an additional constraint explaining (i.e., justifying) the unsatisfiability is included in the MAX-SMT instance and the SAT solver is asked for a new assignment. The diversification strategy to obtain multiple candidates solutions is analogous to the case of MAX-SAT.

5 Experimental results

For a 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, 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 to the definition of the user preferences cannot be quantified in advance. Many of them may be uninformative, as they do not represent any decisional criterion for the customer. The set of catalog features used in our experiments is listed in Table 1.

Table 1: 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	distance from downtown	numerical
8	crime rate	numerical
9	location-based taxes and fees	numerical
10	public transit service quality index	numerical
11	distance from high schools	numerical
12	distance from nearest free parking	numerical
13	distance from working place	numerical
14	distance from parents house	numerical
15	price	numerical

A set of 10 hard constraints (Table 2) defining feasible housing locations and known in advance is considered. The hard constraints are stated by the customer (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, depending on the variable type. 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 a house with garage.

We generated a set of 40 predicates. The unknown DM utility function is composed of terms with two or three predicates, with at least one term with three predicates. The maximum size of terms

Table 2: hard feasibility constraints for the housing problem. Parameters ρ_i, $i = 1 \dots 13$, are threshold values specified by the user or by the sales personnel.

num	hard constraint
1	price $\leq \rho_1$
2	location-based taxes and fees $\leq \rho_2 =>$ *not* public green ares in the neighborhood *and not* public transit service quality index $\leq \rho_3$
3	commercial facilities in the neighborhood $=>$ *not* (garden *and* garage)
4	crime rate $\leq \rho_4 =>$ distance from downtown $\geq \rho_5$
5	distance from working place + distance from parents house $\geq \rho_6$
6	distance from working place + distance from high schools $\geq \rho_7$
7	distance from parents house + distance from high schools $\geq \rho_8$
8	distance from nearest free parking $\leq \rho_9 =>$ *not* public green areas in the neighborhood
9	distance from parents house $\leq \rho_{10} =>$ distance from downtown $\geq \rho_{11}$ *and* crime rate $\geq \rho_{12}$
10	garden $=>$ house type $\geq \rho_{13}$

(three) is assumed to be known. Term weights are integer values selected uniformly at random in the range [1, 100]. Inaccurate preference information can be due to occasional inattention or uncertainty of the DM in comparing certain solutions. We model this by adding a 0.1 probability of swapping the correct ranking during feedback.

We run a set of experiments for $s = 10, 20, \dots 100$ initial training examples to be ranked. The performance of our algorithm is measured by considering 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 DM utility function. Its quality is expressed in terms of the approximation error w.r.t. the gold solution, where the *gold* solution is the configuration obtained by optimizing the true DM utility function. Fig. 2 reports the results over a benchmark of 400 randomly generated utility functions for each of the following instantiation of the triplet (*number of DM 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 quality of the solution rapidly improves with a larger number of examples and the algorithm succeeds in exploiting its active learning strategy.

5.1 DM utility function reconstruction

Our method does not need to learn the exact form of the DM utility function. The goal of our approach is indeed to elicit as few preference information from the DM as possible in order to identify her favourite solution (*learning to optimize*). For example, consider the toy DM utility function represented by the negation of a single ternary term: $\neg(x_1 \wedge x_2 \wedge x_3)$. The approximation of the DM utility function consisting, e.g., of the formula $\neg x_1$ is sufficient to find the favourite DM solution. More in general, only the shape of the utility function locally guiding the search to the correct direction is actually needed.

6 Discussion

This work presented an interactive optimization strategy for combinatorial problems over an unknown utility function. A generic framework is introduced, enabling the adoption of off-the-shelf learning methods and MAX-SMT solvers. 1-norm regularization is employed to enforce sparsity of the learnt function. Thanks to the MAX-SMT formalism, our approach can handle a large class of relevant optimization tasks.

Our algorithm 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 [6]), especially to increase the importance of correctly ranking the best solutions, and could be combined with 1-norm

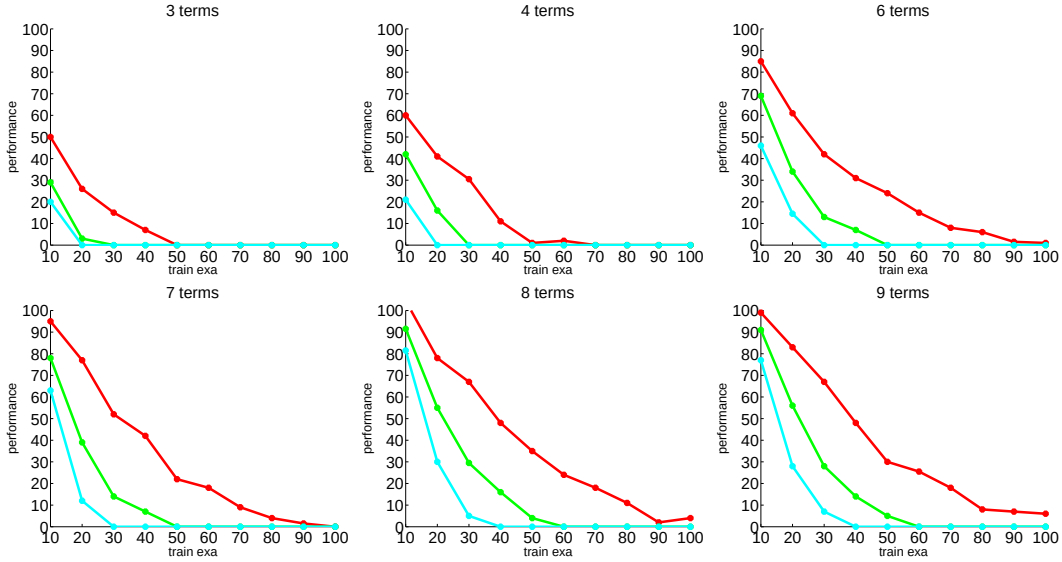

Figure 2: Learning curves observed at different iterations of our algorithm. The *y*-axis reports the approximation error, while the *x*-axis contains the number of initial training examples. Approximation error is the difference between the utility of the solution optimizing the current approximation of the DM preference function and the utility of the gold solution, i.e., the favourite DM solution (see text for details). Red, green and cyan colors show the performance of the algorithm at the first, the second and the third iteration, respectively.

regularization. Retaining the need for sparse features learning, Bayesian approaches to preference elicitation [4] could also be investigated in order to further reduce the cognitive burden for the DM.

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 can 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, as in 2-norm kernel machines. However, our preliminary results [5] showed this to produce worse results for the lack of sparsification in feature space. Kernelized versions of zero-norm regularization [9] could be tried to address this shortcoming. On the other hand, 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.

Finally, another direction for future research is the extension of our approach to handle feedback from multiple DMs [10]. In particular, an interesting case study is the exploitation of preferences of previous DMs to minimize the elicitation effort for a new user [2].

References

[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] E. Bonilla, S. Guo, and S. Sanner. Gaussian process preference elicitation. In J. Lafferty, C. K. I. Williams, J. Shawe-Taylor, R.S. Zemel, and A. Culotta, editors, *Advances in Neural Information Processing Systems 23*, pages 262–270. 2010.

[3] C. Boutilier, K. Regan, and P. Viappiani. Simultaneous elicitation of preference features and utility. In *Proceedings of the Twenty-fourth AAAI Conference on Artificial Intelligence*, pages 1160–1167, Atlanta, USA, July 2010.

[4] D. Braziunas. Computational approaches to preference elicitation. Technical report, Department of Computer Science, University of Toronto, 2006.

[5] 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*, LNCS. Springer Verlag, 2011.

[6] S. Chakrabarti, R. Khanna, U. Sawant, and C. Bhattacharyya. Structured learning for non-smooth ranking losses. In *14th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 88–96. ACM, 2008.

[7] A. Cimatti, A. Franzen, A. Griggio, R. Sebastiani, and C. Stenico. Satisfiability Modulo the Theory of Costs: Foundations and Applications. In Javier Esparza and Rupak Majumdar, editors, *Tools and Algorithms for the Construction and Analysis of Systems*, volume 6015 of *LNCS*, pages 99–113. Springer, 2010.

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

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

[10] Yan Yan, Romer Rosales, Glenn Fung, and Jennifer Dy. Active learning from crowds. In *Proceedings of the 28th International Conference on Machine Learning*.