Active Learning of Combinatorial Features for Interactive Optimization

Paolo Campigotto, Andrea Passerini, and Roberto Battiti

DISI - Dipartimento di Ingegneria e Scienza dell'Informazione
Università degli Studi di Trento
{campigotto,passerini,battiti}@disi.unitn.it
http://www.disi.unitn.it

Abstract. We address the problem of automated discovery of preferred solutions by an interactive optimization procedure. The algorithm iteratively learns a utility function modeling the quality of candidate solutions and uses it to generate novel candidates for the following refinement. We focus on combinatorial utility functions made of weighted conjunctions of Boolean variables. The learning stage exploits the sparsity-inducing property of 1-norm regularization to learn a combinatorial function from the power set of all possible conjunctions up to a certain degree. The optimization stage uses a stochastic local search method to solve a weighted MAX-SAT problem. We show how the proposed approach generalizes to a large class of optimization problems dealing with satisfiability modulo theories. Experimental results demonstrate the effectiveness of the approach in focusing towards the optimal solution and its ability to recover from suboptimal initial choices.

1 Introduction

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

Unfortunately the above picture is not realistic in many application scenarios, where *learning* about the problem definition goes hand in hand with delivering a set of solutions of improving quality, as judged by a decision maker (DM) responsible for selecting the final solution. In particular, this holds in the context of multi-objective optimization, where one aims at maximizing at the same time a set of functions f_1, ..., f_n. Multi-objective optimization, when cast in the language of machine learning, is a paradigmatic case of lack of information, where only some relevant building blocks (*features*) are initially given as the individual function f_i's, but their combination into a *utility function* modeling the

C.A. Coello Coello (Ed.): LION 5, LNCS 6683, pp. 336–350, 2011.

preferences of the DM is not given and has to be learnt by interacting with the DM [1]. Dealing with human DM, characterized by limited patience and bounded rationality, demands for some form of strategic production of candidates to be evaluated (query learning), and requires to account for the possible mistakes and dynamical evolution of her preferences (learning about concrete possibilities may lead somebody to change his/her initial objectives and evaluations). A further complication is related to the difficulty of delivering quantitative judgments by the DM, who is often better off in ranking possibilities more than in delivering utility values. The interplay of optimization and machine learning has been advocated in the past for example in the Reactive Search Optimization (RSO) context, see [2,3] also for an updated bibliography and [4] for an application of RSO in the context of multi-objective optimization.

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

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

We show how to generalize the proposed method to more complex utility functions which are combinations of *predicates* in a certain theory of interest. A standard setting is that of scheduling, where solutions could be starting times for each job, predicates define time constraints for related jobs, and weights specify costs paid for not satisfying a certain set of constraints. The generalization basically consists of replacing satisfiability with satisfiability modulo theory [5] (SMT). SMT is a powerful formalism combining first-order logic formulas and theories providing interpretations for the symbols involved, like the theory of arithmetic for dealing with integer or real numbers. It has received consistently increasing attention in recent years, thanks to a number of successful applications in areas like verification systems, planning and model checking.

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

This manuscript is organized as follows: Section 2 introduces the algorithm for the SAT case. Section 3 introduces SMT and its weighted generalization and shows how to adapt our algorithm to this setting. Related works are discussed in Section 4. Section 5 reports the experimental evaluation for both SAT and SMT problems. 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. The only assumption we make on the utility function is its sparsity, both in the number of features (from the whole set of catalog ones) and in the number of terms constructed from them. We rely on this assumption in designing our optimization algorithm.

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

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

The refinement of the utility model consists of learning the weights of the terms, discarding the terms with zero weight. In the following, we assume that the available feedback consists of a quantitative score. We thus learn the utility function by performing regression over the set of the Boolean vectors. Adapting the method to other forms of feedback, such as ranking of sets of solutions, is straightforward as will be discussed in Section 6. We address the regression task by the Lasso [6]. The Lasso is an appropriate choice on problem domains with many irrelevant features, as its 1-norm regularization can automatically select input features by assigning zero weights to the irrelevant ones. Feature selection is crucial for achieving accurate prediction if the underlying model is sparse [7].

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

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

Fig. 1. Pseudocode for the interactive optimization algorithm

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

where the mapping function Φ projects sample vectors to the space of all possible conjunctions of up to d Boolean variables. The learnt function $f(\mathbf{x}) = \mathbf{w}^T \cdot \Phi(\mathbf{x})$ will be used as the novel approximation of the utility function. A new iteration of our algorithm can now take place. The pseudocode of our algorithm is in Fig. 1.

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. This will typically be the case when interacting with a human DM. A possible alternative consists of directly learning a non-linear function of the features, without explicitly projecting them to the resulting higher dimensional space. We do this by kernel ridge regression [8] (Krr), where 2-norm regularization is used in place of 1-norm. The resulting dual formulation can be kernelized into:

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

where K and I are the kernel and identity matrices respectively and λ is again the regularization parameter. The learnt function is a linear combination of kernel values between the example and each of the training instances: $f(\mathbf{x}) = \sum_{i=1}^{m} \alpha_i K(\mathbf{x}, \mathbf{x}_i)$. We employ a Boolean kernel [9] which implicitly considers all conjunctions of up to d features:

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

With the lasso, the function $\Phi(\cdot)$ maps the Boolean variables to all possible terms of size up to d. This allows for an explicit representation of the learnt

utility function f as a weighted combination of the selected Boolean terms. On the other hand, in the kernel ridge regression case terms are only implicitly represented via the Boolean kernel K_B. In both cases, the value of the learnt function f is used to guide the search of the SLS algorithm. In the following, the two proposed approaches are referred as the *Lasso* and the *Krr* algorithms. As will be shown in the experimental section, the sparsity-inducing property of the *Lasso* allows it to consistently outperform *Krr*. The problem of addressing more complex scenarios, possibly involving non-human DM, where we can not afford an explicit projection, will be discussed in Section 6.

3 Satisfiability Modulo Theory

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

Unfortunately, adding all axioms of T is not viable for many theories of interest: consider for instance the theory of *arithmetic*, which restricts the interpretation of symbols such as $+, \geq, 0, 5$. A more efficient alternative consists of using *specialized* reasoning methods for the background theory of interest. The resulting problem is known as *satisfiability modulo theory* (SMT)[5] and has drawn a lot of attention in recent years, guided by its applicability to a wide range of real-world problems. Among them, consider, for example, problems arising in formal hardware/software verification or in real-time embedded systems design. Popular examples of useful theories include various theories of arithmetic over reals or integers such as linear or difference ones. Linear arithmetic considers $+$ and $-$ functions alone, applied to either numerical constants or variables, plus multiplication by a numerical constant. Difference arithmetic is a fragment of linear arithmetic limiting legal predicates to the form $x - y \leq c$, where x, y are variables and c is a numerical constant. Very efficient procedures exists for checking satisfiability of difference logic formulas [10]. A number of theories have been studied apart from standard arithmetic ones (e.g., the theory of bit-vector arithmetic to model machine arithmetic).

```
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
```

Fig. 2. Pseudocode for a basic lazy SMT-solver

3.1 Satisfiability Modulo Theory Solvers

The most successful SMT solvers can be grouped into the two main approaches named *eager* and *lazy*. The eager approach consists of developing theory-specific and efficient translators which translate a query formula into an equisatisfiable propositional one, much like compilers do when optimizing the code generated from a high-level program. Lazy approaches, on the other hand, work by building efficient *theory solvers*, inference systems specialized on a theory of interest. These solvers are integrated as submodules into a generic SAT solver. In the rest of the paper we will focus on this latter class of SMT solvers, which we integrated in our optimization algorithm. The simplest approach for building a lazy SMT-solver consists of alternating calls to the satisfiability and the theory solver respectively, until a solution satisfying both solvers is retrieved or the problem is found to be unsatisfiable. Let φ be a formula in a certain theory T, made of a set of n predicates $A = \{a_1, \ldots, a_n\}$. A mapping α maps φ into a propositional formula $\alpha(\varphi)$ by replacing its predicates with propositional variables $p_i = \alpha(a_i)$. The inverse mapping β replaces propositional variables with their corresponding predicates, i.e., $\beta(p_i) = a_i$. For example, consider the following formula in a non-linear theory T:

$$(\cos(x) = 3 + \sin(y)) \wedge (z \leq 8) \tag{2}$$

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

Figure 2 reports the basic form [11] of an SMT algorithm. `SAT`(φ) calls the SAT solver on the φ instance, returning a pair (r, M), where r is `sat` if the instance is satisfiable, `unsat` otherwise. In the former case, M is a truth assignment satisfying φ. `T-Solver`(S) calls the theory solver on the formula S and returns a pair (r, J), where r indicates if the formula is satisfiable. If $r =$`unsat`, J is a *justification* for S, i.e any unsatisfiable subset $J \subset S$. The next iteration calls the SAT solver on an extended instance accounting for this justification.

State-of-the-art solvers introduce a number of refinements to this basic strategy, by pursuing a tighter integration between the two solvers. A common underlying idea is to prune the search space for the SAT solver by calling the theory

solver on partial assignments and propagating its results. Finally, combination methods exist to jointly employ different theories, see [12] for a basic procedure.

3.2 Weighted MAX-SMT

Weighted MAX-SMT generalizes SMT problems much like weighted MAX-SAT does with SAT ones. While a body of works exist addressing weighted MAX-SAT problems, the former generalization has been tackled only recently and very few solvers have been developed [13,14,15]. The simplest formulation consists of adding a cost to each or part of the formulas to be jointly satisfied, and returning the assignment of variables minimizing the sum of the costs of the unsatisfied clauses, or a satisfying assignment if it exists. The following is a "weighted version" of Eq. 2:

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

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

Generalizing, consider a true utility function f expressed as a weighted sum of terms, where a term is the conjunction of up to d predicates defined over the variables in the theory T. The set of *all* n possible predicates represents the search space S of the MAX-SAT solver integrated in the MAX-SMT solver. Our approach learns an approximation $\hat{f}$ of f and gets one of its optimizers $\mathbf{v}$ from the MAX-SMT solver. The optimizer (and in general each candidate solution in the theory T) identifies an assignment $\mathbf{p}^* = (p_1^*, \ldots, p_n^*)$ of Boolean values ($p_i^* = \{\mathbf{true}, \mathbf{false}\}$) to the predicates in S. The DM is asked for a feedback on the candidate solution $\mathbf{v}$ and returns a possibly noisy quantitative score $s \approx f(\mathbf{v})$. The pair $(\mathbf{p}^*, s)$ represents a new training example for our approach. In order to obtain multiple training examples, we optimize again $\hat{f}$ with the additional *hard*[1] constraint generated by the disjunction of all the terms of $\hat{f}$ unsatisfied by $\mathbf{p}^*$. For example, let t_1 and t_5 be the terms of $\hat{f}$ unsatisfied by $\mathbf{p}^*$, then the hard constraint becomes:

$$(t_1 \vee t_5)$$

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

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

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

The learning component of our algorithm is then re-trained, including in the training set the new collected examples and the approximation of the true utility function is refined. A new optimization phase can now take place (see Fig. 1).

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

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

4 Related Works

Active learning is a hot research area and a broad range of different approaches has been proposed (see [16] 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 [1] and information retrieval [17]. The need to trade off multiple requirements in this active learning setting is addressed in [18] where the authors consider relevance, diversity and density in selecting candidates. Note that our approach relies on query *synthesis* rather than selection, as *de-novo* candidate solutions are generated by the SLS algorithm. Nonetheless, our diversification strategies are very simple and could be significantly improved by taking advantage of the aforementioned literature.

Choosing relevant features according to their weight within the learnt model is a common selection strategy (see e.g. [19]). When dealing with implicit feature spaces as in kernel machines, the problem can be addressed by introducing a hyper-parameter for each input feature, like a feature-dependent variance for Gaussian kernels [20]. Parameters and hyper-parameters (or their relaxed real-valued version) are jointly optimized trying to identify a small number of relevant features. One-norm regularization [6] has the advantage of naturally inducing sparsity in the set of selected features. Approaches also exist [21] which directly address the combinatorial problem of zero-norm optimization.

A large body of recent work exists for developing interactive approaches [1] to multiobjective optimization. A common approach consists of modeling the utility function as a linear combination of objectives, and iteratively updating its weights trying to match the DM requirements. Our algorithm allows to deal with complex non-linear interactions between (Boolean) objectives and, thanks to the SMT extension, can be applied to a wide range of optimization problems.

Very recent works in the field of constraint programming [22] define the user preferences in terms of *soft* constraints and introduce constraint optimization problems where the data are not completely known before the solving process starts. In particular, the work in [22] introduces an elicitation strategy for soft constraint problems with missing preferences, with the purpose of finding the solution preferred by the DM asking to reveal as few preferences as possible. Despite the common purpose, this approach is different from ours. A major difference regards the preference elicitation problem considered. In [22] decision variables and soft constraints are assumed to be known in advance and the information uncertainty consists only of missing preference values. On the other

hand, our settings assume sparsity of the utility function, both in the number of features (from the whole set of catalog features) and in the selection of the terms constructed from them. Furthermore, our technique is robust to imprecise information from the DM, modeled in terms of inaccurate preference scores for the candidate solutions. Even if interval-valued constraints [23] have been introduced to handle uncertainty in the evaluations of the DM, the experiments in [22] do not consider the case of inconsistent preference information. Finally, while the technique in [22] combines branch and bound search with preference elicitation and the adoption of local search algorithms is matter of research, our approach works straightforwardly with both incomplete and complete search techniques.

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 [13].

5.1 Weighted MAX-SAT

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

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

We run a set of experiments for $10, 20, \ldots 100$ initial training examples, for the *Lasso* and the *Krr* versions of the algorithm. Results are expressed in terms of the quality of the learnt utility function (Fig. 3) and of the approximation of the *gold* solution (Fig. 4). Each point of the curves in the Fig. 3 and 4 is the mean and the median values, respectively, over 400 runs with different random seeds.

Fig. 3 shows the quality of the learnt utility function, in terms of the root mean squared error (rmse) between the true and the predicted values for a benchmark of 1000 test examples. A better approximation is generated by the *Lasso* algorithm for all the considered true utility functions. Furthermore, while increasing the number of training examples, a faster improvement is observed for the *Lasso* w.r.t. the *Krr* algorithm. Consider, for example, the case of nine terms. With 40 training examples, the performance of *Krr* is within 10 units from the value observed for the *Lasso* method. When 100 examples are employed, the

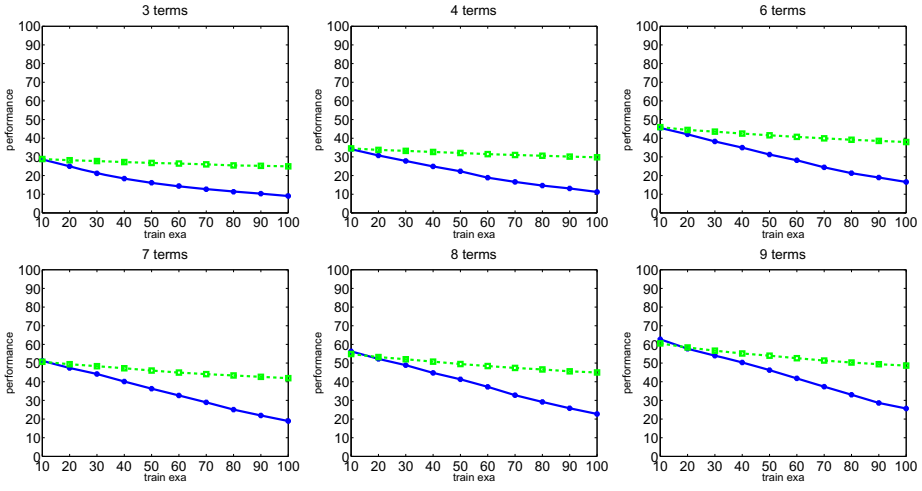

Fig. 3. Quality of the learnt utility function for an increasing number of training examples observed for the algorithms at the first iteration. The y-axis reports the root mean squared error between the true and the predicted values for a benchmark of 1000 test examples. The x-axis contains the number of training examples. The solid blue and the dashed green lines show the performance of the *Lasso* and the *Krr* algorithms, respectively. See text for details.

mean rmse of the *Lasso* algorithm is less than value 30, while the performance of the *Krr* method does not increase beyond value 50.

The superior performance of the *Lasso* algorithm is confirmed by the experiments in Fig. 4, reporting the quality of the *best* configuration at the different iterations for an increasing number of initial training examples. The *best* configuration is the configuration optimizing the current approximation of the true utility function. Its quality is measured in terms of the approximation error w.r.t. the gold solution.

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

However, the *Lasso* approach outperforms the *Krr* results when the true utility function includes at least six terms. First, note that the *Lasso* algorithm succeeds in exploiting its active learning strategy, and converges rather quickly to the optimal solution when enough iterations are provided. At the first iteration its approximation error is above 40 even when 30 training examples are used. At the third iteration, the *Lasso* algorithm identifies the gold standard solution, when at least 60 training examples are available. On the other hand, for true utility functions with more than seven terms *Krr* fails to improve over its suboptimal solution when increasing the number of examples and iterations. As a consequence, the *Krr* algorithm does not identify the gold solution, even

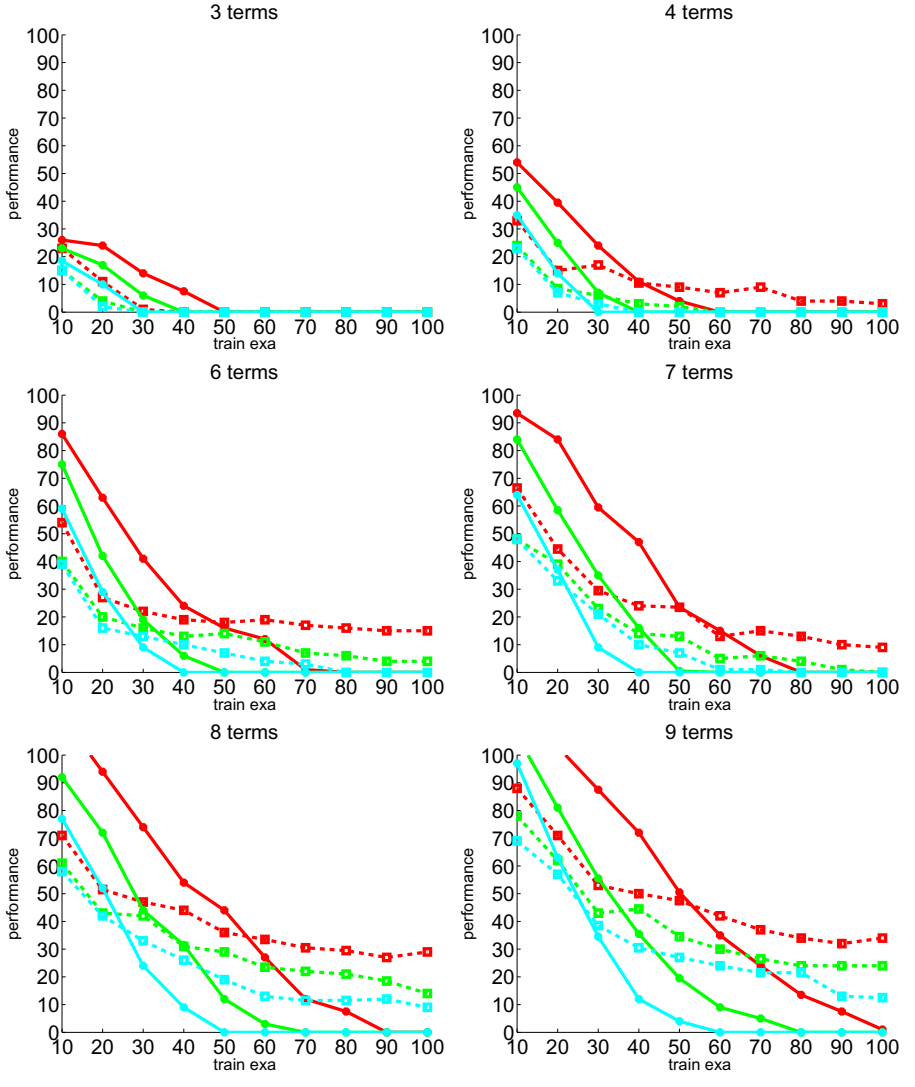

Fig. 4. Learning curves for an increasing number of training examples observed for the two algorithms at different iterations. The y-axis reports the solution quality, while the x-axis contains the number of training examples. The dashed lines refer to the *Krr* algorithm, while the solid lines are for the *Lasso* algorithm. Furthermore, red, green and cyan colors show the performance of the algorithms at the first, the second and the third iteration, respectively. See text for details.

in the case of 100 training examples. However, when very few training examples are available, the *Krr* algorithm reaches a better approximation than *Lasso*.

5.2 Weighted MAX-SMT

SMT is a hot research area [11]. However, MAX-SMT techniques are very recent and there are no well established publicly available benchmarks for weighted MAX-SMT problems. Existing results [14] indicate that MAX-SMT solvers can efficiently address real-world problems.

In this work, we modeled a scheduling problem as a MAX-SMT problem. In detail, a set of five jobs must be scheduled over a given period of time. Each job has a fixed known duration, the constraints define the overlap of two jobs or their non-concurrent execution. The true utility function is generated by selecting uniformly at random weighed terms over the constraints. The solution of the problem is a schedule assigning a starting date to each job and minimizing the cost, where the cost of the schedule is the sum of the weights of the violated terms of the true utility function. The temporal constraints are expressed by using the difference arithmetic theory. In detail, let s_i and d_i, with $i = 1 \dots 5$, 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 true utility function is three and it is assumed to be known. Their weights are distributed uniformly at random in the range $[1, 100]$. Similarly to the MAX-SAT case, the experimental setting includes Gaussian noise (with mean 0 and standard deviation 10) affecting the cost values of the training examples.

Fig. 5 depicts the performance of the Lasso algorithm for the cases of 3, 4, 6, 7, 8, 9 terms in the true utility function. The y-axis reports the solution quality measured in terms of deviation from the gold solution, while the x-axis contains the number n of training examples at the first iteration. At the following iterations, $n/2$ examples are added to the training set (see Sec. 2). Each point of the curves is the median value over 400 runs with different random seeds.

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

Finally, note that the approach based on Krr does not maintain an explicit representation of the learnt utility function, and therefore a direct extension to SMT problems is not possible for the current MAX-SMT solvers which tightly integrate SAT and theory solvers as discussed in Section 3.

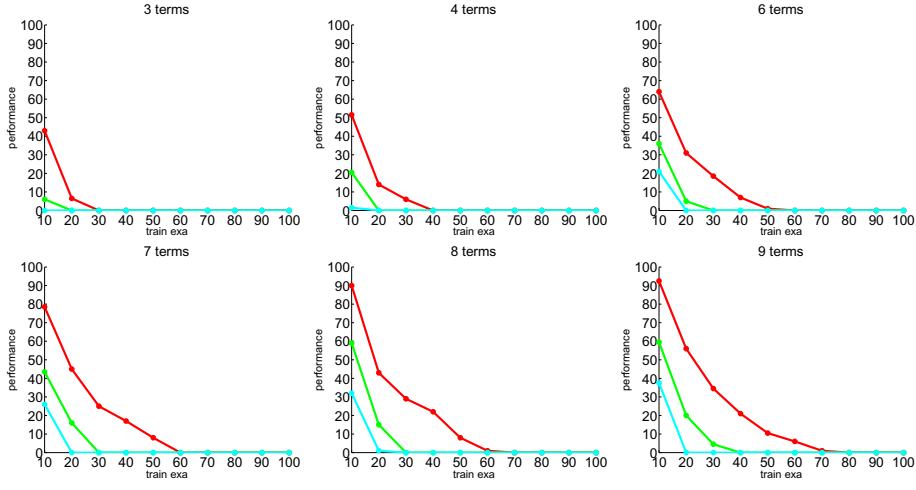

Fig. 5. Learning curves observed at different iterations of the Lasso algorithm while solving the scheduling problem. The y-axis reports the solution quality, while the x-axis contains the number of training examples. Red, green and cyan colors show the performance of the algorithm at the first, the second and the third iteration, respectively. See text for details.

6 Discussion

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

The algorithm can be generalized in a number of directions. The availability of a quantitative feedback is not necessarily straightforward, especially when a human DM is involved in the loop. A more affordable request is often that of ranking sets of candidates according to preference. Our setting can be easily adapted to this setting by replacing the squared error loss in the learning stage with appropriate ranking losses. The simplest solution consists of formulating it as correctly ordering each pair of instances as done in support vector ranking, and applying 1-norm SVM [24]. More complex ranking losses have been proposed

in the literature (see for instance [25]), especially to increase the importance of correctly ranking the best solutions, and could be combined with 1-norm regularization.

Our experimental evaluation is focused on small-scale problems, typical of an interaction with a human DM. In principle, when combined with appropriate SMT solvers, our approach could be applied to larger real-world optimization problems, whose formulation is only partially available. In this case, a local search algorithm rather than a complete solver will be used during the optimization stage, as showed in the experiments on the weighted MAX-SAT instances. However, the cost of requiring an explicit representation of all possible conjunction of predicates (even if limited to the unknown part) would rapidly produce an explosion of computational and memory requirements. One option is that of resorting to an implicit representation of the function to be optimized, like the one we used in the *Krr* algorithm. Kernelized versions of zero-norm regularization [26] 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. We are currently investigating both research directions.

Finally, we are also considering larger preference elicitation problems, with both known hard constraints limiting the set of feasible solutions and unknown user preferences. This setting allows us to address many real-world scenarios. In the house sale system, for instance, the hard constraints could define the available house types or locations, and the preferences of the DM would drive the search within the set of feasible solutions.

References

1. Branke, J., Deb, K., Miettinen, K., Słowiński, R. (eds.): Multiobjective Optimization: Interactive and Evolutionary Approaches. Springer, Heidelberg (2008)
2. Battiti, R., Brunato, M., Mascia, F.: Reactive search and intelligent optimization. Springer, Heidelberg (2008)
3. Battiti, R., Brunato, M.: Reactive search optimization: Learning while optimizing. In: Gendreau, M., Potvin, J.Y. (eds.) Handbook of Metaheuristics, 2nd edn. Int. Series in Op. Res. & Man. Sci., vol. 146, pp. 543–571. Springer Science, Heidelberg (2010)
4. Battiti, R., Campigotto, P.: Reactive Search Optimization: Learning While Optimizing. An Experiment in Interactive Multi-Objective Optimization. In: VIII Metaheur. Int. Conf. (MIC 2009), Germany. LNCS, Springer, Heidelberg (2009)
5. Barrett, C., Sebastiani, R., Seshia, S.A., Tinelli, C.: Satisfiability modulo theories. In: Handbook of Satisfiability, pp. 825–885. IOS Press, Amsterdam (2009)
6. Tibshirani, R.: Regression shrinkage and selection via the lasso. Journal of the Royal Statistical Society, Series B 58, 267–288 (1996)
7. Friedman, J., Hastie, T., Rosset, S., Tibshirani, R.: Discussion of boosting papers. Annals of Statistics 32, 102–107 (2004)

8. Suanders, C., Gammerman, A., Vovk, V.: Ridge regression learning algorithm in dual variables. In: ICML 1998 (1998)
9. Khardon, R., Roth, D., Servedio, R.: Efficiency versus convergence of boolean kernels for on-line learning algorithms. Journal of Artif. Int. Res. 24(1), 341–356 (2005)
10. Nieuwenhuis, R., Oliveras, A.: DPLL(T) with exhaustive theory propagation and its application to difference logic. In: Etessami, K., Rajamani, S.K. (eds.) CAV 2005. LNCS, vol. 3576, pp. 321–334. Springer, Heidelberg (2005)
11. de Moura, L., Bjorner, N.: Satisfiability modulo theories: An appetizer. In: Oliveira, M.V.M., Woodcock, J. (eds.) SBMF 2009. LNCS, vol. 5902, pp. 23–36. Springer, Heidelberg (2009)
12. Nelson, G., Oppen, D.C.: Simplification by cooperating decision procedures. ACM Trans. Program. Lang. Syst. 1(2), 245–257 (1979)
13. Dutertre, B., de Moura, L.: A Fast Linear-Arithmetic Solver for DPLL(T). In: Ball, T., Jones, R.B. (eds.) CAV 2006. LNCS, vol. 4144, pp. 81–94. Springer, Heidelberg (2006)
14. Nieuwenhuis, R., Oliveras, A.: On sat modulo theories and optimization problems. In: In Theory and App. of Sat. Testing. LNCS, pp. 156–169. Springer, Heidelberg (2006)
15. Cimatti, A., Franzén, A., Griggio, A., Sebastiani, R., Stenico, C.: Satisfiability modulo the theory of costs: Foundations and applications. In: Esparza, J., Majumdar, R. (eds.) TACAS 2010. LNCS, vol. 6015, pp. 99–113. Springer, Heidelberg (2010)
16. Settles, B.: Active learning literature survey. Technical Report Computer Sciences Technical Report 1648, University of Wisconsin-Madison (2009)
17. Radlinski, F., Joachims, T.: Active exploration for learning rankings from clickthrough data. In: 13th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD 2007), pp. 570–579. ACM Press, New York (2007)
18. Xu, Z., Akella, R., Zhang, Y.: Incorporating diversity and density in active learning for relevance feedback. In: Amati, G., Carpineto, C., Romano, G. (eds.) ECIR 2007. LNCS, vol. 4425, pp. 246–257. Springer, Heidelberg (2007)
19. Guyon, I., Weston, J., Barnhill, S., Vapnik, V.: Gene selection for cancer classification using support vector machines. Machine Learning 46(1-3), 389–422 (2002)
20. Chapelle, O., Vapnik, V., Bousquet, O., Mukherjee, S.: Choosing multiple parameters for support vector machines. Machine Learning 46(1-3), 131–159 (2002)
21. Kaizhu, H., Irwin, K., Michael, R.: Direct Zero-Norm Optimization for Feature Selection. In: IEEE International Conference on Data Mining, pp. 845–850 (2008)
22. Gelain, M., Pini, M.S., Rossi, F., Venable, K.B., Walsh, T.: Elicitation strategies for soft constraint problems with missing preferences: Properties, algorithms and experimental studies. Artif. Intell. 174(3-4), 270–294 (2010)
23. Gelain, M., Pini, M.S., Rossi, F., Venable, K.B., Wilson, N.: Interval-valued soft constraint problems. Annals of Mat. and Art. Int. 58, 261–298 (2010)
24. Zhu, J., Rosset, S., Hastie, T., Tibshirani, R.: 1-norm Support Vector Machines. In: Neural Information Processing Systems. MIT Press, Cambridge (2003)
25. Chakrabarti, S., Khanna, R., Sawant, U., Bhattacharyya, C.: Structured learning for non-smooth ranking losses. In: 14th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, KDD 2008, pp. 88–96. ACM, New York (2008)
26. Weston, J., Elisseeff, A., Schölkopf, B., Tipping, M.: Use of the zero norm with linear models and kernel methods. Journal of Mach. Learn. Res. 3, 1439–1461 (2003)