Accepted Manuscript

Structured learning modulo theories

Stefano Teso, Roberto Sebastiani, Andrea Passerini
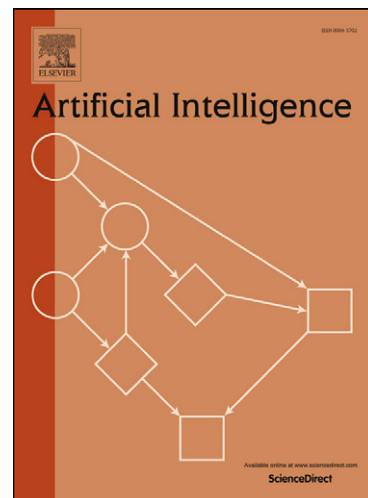
Please cite this article in press as: S. Teso et al., Structured learning modulo theories, *Artificial Intelligence* (2015), http://dx.doi.org/10.1016/j.artint.2015.04.002

Structured Learning Modulo Theories

Stefano Teso[a], Roberto Sebastiani[b], Andrea Passerini[b]

[a]*DKM - Data & Knowledge Management Unit,*
Fondazione Bruno Kessler,
Via Sommarive, 18, I38123 Povo, TN, Italy.
[b]*DISI - Dipartimento di Ingegneria e Scienza dell'Informazione,*
Università degli Studi di Trento,
Via Sommarive 5, I-38123 Povo, TN, Italy.

Abstract

Modelling problems containing a mixture of Boolean and numerical variables is a long-standing interest of Artificial Intelligence. However, performing inference and learning in hybrid domains is a particularly daunting task. The ability to model these kind of domains is crucial in "learning to design" tasks, that is, learning applications where the goal is to learn from examples how to perform automatic *de novo* design of novel objects. In this paper we present Structured Learning Modulo Theories, a max-margin approach for learning in hybrid domains based on Satisfiability Modulo Theories, which allows to combine Boolean reasoning and optimization over continuous linear arithmetical constraints. The main idea is to leverage a state-of-the-art generalized Satisfiability Modulo Theory solver for implementing the inference and separation oracles of Structured Output SVMs. We validate our method on artificial and real world scenarios.

Keywords: Satisfiability Modulo Theory, Structured-Output Support Vector Machines, Optimization Modulo Theory, Constructive Machine Learning, Learning with Constraints

1. Introduction

Research in machine learning has progressively widened its scope, from simple scalar classification and regression tasks to more complex problems

Email addresses: `steteso@fbk.eu` (Stefano Teso), `roberto.sebastiani@unitn.it` (Roberto Sebastiani), `passerini@disi.unitn.it` (Andrea Passerini)

involving multiple related variables. Methods developed in the related fields of statistical relational learning (SRL) [1] and structured-output learning [2] allow to perform learning, reason and inferences about relational entities characterized by both hard and soft constraints. Most methods rely on some form of (finite) First-Order Logic (FOL) to encode the learning problem, and define the constraints as weighted logical formulae. One issue with these approaches is that First-Order Logic is not suited for efficiently reasoning over hybrid domains, characterized by both continuous and discrete variables. The Booleanization of an n-bit integer variable requires 2^n distinct Boolean states, making naive translation impractical; for rational variables the situation is even worse. In addition, standard FOL automated reasoning techniques offer no mechanism to deal efficiently with operators among numerical variables, like comparisons (e.g. "less-than", "equal-to") and arithmetical operations (e.g. summation), limiting the range of realistically applicable constraints to those based solely on logical connectives. On the other hand, many real-world domains are inherently hybrid and require to reason over inter-related continuous and discrete variables. This is especially true in *constructive* machine learning tasks, where the focus is on the *de-novo* design of objects with certain characteristics to be learned from examples (e.g. a recipe for a dish, with ingredients, proportions, etc.).

There is relatively little previous work on *hybrid* SRL methods. A number of approaches [3, 4, 5, 6] focused on the feature representation perspective, in order to extend statistical relational learning algorithms to deal with continuous features as *inputs*. On the other hand, performing inference over joint continuous-discrete relational domains is still a challenge. The few existing attempts [7, 8, 9, 10, 11, 12] aim at extending statistical relational learning methods to the hybrid domain. All these approaches focus on modeling the probabilistic relationships between variables. While this allows to compute marginal probabilities in addition to most probable configurations, it imposes strong limitations on the type of constraints they can handle. Inference is typically run by approximate methods, based on variational approximations or sampling strategies. Exact inference, support for hard numeric (in addition to Boolean) constraints and combination of diverse theories, like linear algebra over rationals and integers, are out of the scope of these approaches. Hybrid Markov Logic networks [8] and Church [7] are the two formalisms which are closer to the scope of this paper. Hybrid Markov Logic networks [8] extend Markov Logic by including continuous variables, and allow to embed numerical comparison operators (namely $\neq$, $\geq$ and $\leq$) into the constraints

2

by defining an *ad hoc* translation of said operators to a continuous form amenable to numerical optimization. Inference relies on a stochastic local search procedure that interleaves calls to a MAX-SAT solver and to a numerical optimization procedure. This inference procedure is incapable of dealing with hard numeric constraints because of the lack of feedback from the continuous optimizer to the satisfiability module. Church [7] is a very expressive probabilistic programming language that can potentially represent arbitrary constraints on both continuous and discrete variables. Its focus is on modelling the generative process underlying the program, and inference is based on sampling techniques. This makes inference involving continuous optimization subtasks and hard constraints prohibitively expensive, as will be discussed in the experimental evaluation.

In order to overcome the limitations of existing approaches, we focused on the most recent advances in automated reasoning over hybrid domains. Researchers in automated reasoning and formal verification have developed logical languages and reasoning tools that allow for *natively* reasoning over mixtures of Boolean *and* numerical variables (or even more complex structures). These languages are grouped under the umbrella term of *Satisfiability Modulo Theories* (SMT) [13]. Each such language corresponds to a decidable fragment of First-Order Logic augmented with an additional background theory $\mathcal{T}$. There are many such background theories, including those of linear arithmetic over the rationals $\mathcal{LRA}$ or over the integers $\mathcal{LIA}$, among others [13]. In SMT, a formula can contain Boolean variables (i.e. 0-ary logical predicates) and connectives, mixed with symbols defined by the theory $\mathcal{T}$, e.g. rational variables and arithmetical operators. For instance, the SMT($\mathcal{LRA}$) syntax allows to write formulas such as:

$$\texttt{touching\_i\_j} \leftrightarrow ((x_i + dx_i = x_j) \lor (x_j + dx_j = x_i))$$

where the variables are Boolean ($\texttt{touching\_i\_j}$) or rational (x_i, x_j, dx_i,dx_j).[1] More specifically, SMT is a decision problem, which consists in finding an assignment to the variables of a quantifier-free formula, both Boolean and

---

[1]Note that SMT solvers handle also formulas on *combinations* of theories, like e.g. $(d \geq 0) \land (d < 1) \land ((f(d) = f(0)) \rightarrow (\texttt{read}(\texttt{write}(V, i, x), i + d) = x + 1))$, where d, i, x are integer variables, V is an array variable, f is an uninterpreted function symbol, $\texttt{read}$ and $\texttt{write}$ are functions of the theory of arrays [13]. However, for the scope of this paper it suffices to consider the $\mathcal{LRA}$ and $\mathcal{LIA}$ theories, and their combination.

theory-specific ones, that makes the formula true, and it can be seen as an extension of SAT.

Recently, researchers have leveraged SMT from decision to optimization. In particular, *MAX-SAT Modulo Theories* (*MAX-SMT*) [14, 15, 16] generalizes MAX-SAT [17] to SMT formulae, and consists in finding a theory-consistent truth assignment to the atoms of the input SMT formula φ which maximizes the total weight of the satisfied clauses of φ. More generally, *Optimization Modulo Theories* (*OMT*) [14, 18, 19, 20, 21, 22] consists in finding a *model* for φ which minimizes the value of some (arithmetical) term, and strictly subsumes MAX-SMT [19]. Most important for the scope of this paper is that there are high-quality OMT solvers which, at least for the $\mathcal{LRA}$ theory, can handle problems with thousands of hybrid variables.

In this paper we propose *Learning Modulo Theories* (LMT), a class of novel hybrid statistical relational learning methods. The main idea is to combine a solver able to deal with Boolean and rational variables with a structured output learning method. In particular, we rely on structured-output Support Vector Machines (SVM) [23, 24], a very flexible max-margin structured prediction method. Structured-output SVMs are a generalization of binary SVM classifiers to predict structured outputs, like sequences or trees. They generalize the max-margin principle by learning to separate correct from incorrect output structures with a large margin. Training structured-output SVMs requires a separation oracle for generating counter-examples and updating the parameters, while the prediction stage requires an inference oracle generating the highest scoring candidate structure for a certain input. In order to implement the two oracles, we leverage a state-of-the-art OMT solver. This combination enables LMT to perform learning and inference in mixed Boolean-numerical domains. Thanks to the efficiency of the underlying OMT solver, and of the cutting plane algorithm we employ for weight learning, LMT is capable of addressing constructive learning problems which cannot be efficiently tackled with existing methods. Furthermore, LMT is *generic*, and can in principle be applied to any of the existing background theories. This paper builds on a previous work in which MAX-SMT was used for interactive preference elicitation [25]. Here we focus on generating novel structures from a few prototypical examples, and cast the problem as supervised structured-output learning. Furthermore, we increase the expressive power from MAX-SMT to full OMT. This allows to model much richer cost functions, for instance by penalizing an unsatisfied constraint by a cost proportional to the distance from satisfaction.

The rest of the paper is organized as follows. In Section 2 we review the relevant related work, with an in-depth discussion on all hybrid approaches and their relationships with our proposed framework. Section 3 provides an introduction to SMT and OMT technology. Section 4 reviews structured-output SVMs and shows how to cast LMT in this learning framework. Section 5 reports an experimental evaluation showing the potential of the approach. Finally, conclusions are drawn in Section 6.

2. Related Work

There is a body of work concerning integration of relational and numerical data from a feature representation perspective, in order to effectively incorporate numerical features into statistical relational learning models. Lippi and Frasconi [3] incorporate neural networks as feature generators within Markov Logic Networks, where neural networks act as numerical functions complementing the Boolean formulae of standard MLNs. Semantic Based Regularization [6] is a framework for integrating logic constraints within kernel machines, by turning them into real-valued constraints using appropriate transformations (T-norms). The resulting optimization problem is no longer convex in general, and they suggest a stepwise approach adding constraints in an incremental fashion, in order to solve progressively more complex problems. In Probabilistic Soft Logic [4], arbitrarily complex similarity measures between objects are combined with logic constraints, again using T-norms for the continuous relaxation of Boolean operators. In Gaussian Logic [5], numeric variables are modeled with multivariate Gaussian distributions. Their parameters are tied according to logic formulae defined over these variables, and combined with weighted first order formulae modeling the discrete part of the domain (as in standard MLNs). All these approaches aim at extending statistical relational learning algorithms to deal with continuous features as *inputs*. On the other hand, our framework aims at allowing learning and inference over hybrid continuous-discrete domains, where continuous and discrete variables are the *output* of the inference process.

While a number of efficient lifted-inference algorithms have been developed for Relational Continuous Models [26, 27, 28], performing inference over joint continuous-discrete relational domains is still a challenge. The few existing attempts aim at extending statistical relational learning methods to the hybrid domain.

5

Hybrid Probabilistic Relational Models [9] extend Probabilistic Relational Models (PRM) to deal with hybrid domains by specifying templates for hybrid distributions as standard PRM specify templates for discrete distributions. A template instantiation over a database defines a Hybrid Bayesian Network [29, 30]. Inference in Hybrid BN is known to be hard, and restrictions are typically imposed on the allowed relational structure (e.g. in conditional Gaussian models, discrete nodes cannot have continuous parents). On the other hand, LMT can accomodate arbitrary combinations of predicates from the theories for which a solver is available. These currently include linear arithmetic over both rationals and integers as well as a number of other theories like strings, arrays and bit-vectors.

Relational Hybrid Models [11] (RHM) extend Relational Continuous Models to represent combinations of discrete and continuous distributions. The authors present a family of lifted variational algorithms for performing efficient inference, showing substantial improvements over their ground counterparts. As for most hybrid SRL approaches which will be discussed further on, the authors focus on efficiently computing probabilities rather than efficiently finding optimal configurations. Exact inference, hard constraints and theories like algebra over integers, which are naturally handled by our LMT framework, are all out of the scope of these approaches. Nonetheless, lifted inference is a powerful strategy to scale up inference and equipping OMT and SMT tools with lifting capabilities is a promising direction for future improvements.

The PRISM [31] system provides primitives for Gaussian distributions. However, inference is based on proof enumeration, which makes support for continuous variables very limited. Islam et al. [12] recently extended PRISM to perform inference over continuous random variables by a *symbolic* procedure which avoids the enumeration of individual proofs. The extension allows to encode models like Hybrid Bayesian Networks and Kalman Filters. Being built on top of the PRISM system, the approach assumes the exclusive explanation and independence property: no two different proofs for the same goal can be true simultaneously, and all random processes within a proof are independent (some research directions for lifting these restrictions have been suggested [32]). LMT has no assumptions on the relationships between proofs.

Hybrid Markov Logic Networks [8] extend Markov Logic Networks to deal with numeric variables. A Hybrid Markov Logic Network consists of both First Order Logic formulae and numeric terms. Most probable explanation

6

(MPE) inference is performed by a hybrid version of MAXWalkSAT, where optimization of numeric variables is performed by a general-purpose global optimization algorithm (L-BFGS). This approach is extremely flexible and allows to encode arbitrary numeric constraints, like soft equalities and inequalities with quadratic or exponential costs. A major drawback of this flexibility is the computational cost, as each single inference step on continuous variables requires to solve a global optimization problem, making the approach infeasible for addressing medium to large scale problems. Furthermore, this inference procedure is incapable of dealing with hard constraints involving numeric variables, as can be found for instance in layout problems (see e.g. the constraints on touching blocks or connected segments in the experimental evaluation). This is due to the lack of feedback from the continuous optimizer to the satisfiability module, which should inform about conflicting constraints and help guiding the search towards a more promising portion of the search space. Conversely, the OMT technology underlying LMT is built on top of SMT solvers and is hence specifically designed to tightly integrate theory-specific and SAT solvers [14, 15, 18, 19, 20]. Note that the tight interaction between theory-specific and modern CDCL SAT solvers, plus many techniques developed for maximizing their synergy, are widely recognised as one key reason of the success of SMT solvers [13]. Note also that previous attempts to substitute standard SAT solvers with Walk-SAT inside an SMT solver have failed, producing dramatic worsening of performance [33].

Hybrid ProbLog [10] is an extension of the probabilistic logic language ProbLog [34] to deal with continuous variables. A ProbLog program consists of a set of probabilistic Boolean facts, and a set of deterministic first order logic formulae representing the background knowledge. Hybrid ProbLog introduces a set of probabilistic continuous facts, containing both discrete and continuous variables. Each continuous variable is associated with a probability density function. The authors show how to compute the probability of success of a query, by partitioning the continuous space into admissible intervals, within which values are interchangeable with respect to the provability of the query. The drawback of this approach is that in order to make this computation feasible, severe limitations have to be imposed on the use of continuous variables. No algebraic operations or comparisons are allowed between continuous variables, which should remain uncoupled. Some of these limitations have been overcome in a recent approach [35] which performs inference by forward (i.e. from facts to rules) rather than backward reasoning,

7

which is the typical inference process in (probabilistic) logic programming engines (SLD-resolution and its probabilistic extensions). Forward reasoning is more amenable to be adapted to sampling strategies for performing approximate inference and dealing with continuous variables. On the other hand, inference by sampling makes it prohibitively expensive to reason with hard continuous constraints.

Church [7] is a very expressive probabilistic programming language that can easily accomodate hybrid discrete-continuous distributions and arbitrary constraints. In order to deal with the resulting complexity, inference is again performed by sampling techniques, which result in the same aforementioned limitations. Indeed, our experimental evaluation shows that Church is incapable of solving in reasonable time the simple task of generating a pair of blocks conditioned on the fact that they touch somewhere. [2]

An advantage of these probabilistic inference approaches is that they allow to return marginal probabilities in addition to most probable explanations. This is actually the main focus of these approaches, and the reason why they are less suitable for solving the latter problem when the search space becomes strongly disconnected. As with most structured-output approaches over which it builds, LMT is currently limited to the task of finding an optimal configuration, which in a probabilistic setting corresponds to generating the most probable explanation. We are planning to extend it to also perform probability computation, as discussed in the conclusions of the paper.

3. From Satisfiability to Optimization Modulo Theories

Propositional satisfiability (SAT), is the problem of deciding whether a logical formula over Boolean variables and logical connectives can be satisfied by some truth value assignment of the Boolean variables. [3] In the last two decades we have witnessed an impressive advance in the efficiency of SAT solvers, which nowadays can handle industrially derived formulae in the order

---

[2]The only publicly available version of Hybrid ProbLog is the original one by Gutmann, Jaeger and De Raedt [10] which does not support arithmetic over continuous variables. However we have no reason to expect the more recent version based on sampling should have a substantially different behaviour with respect to what we observe with Church.

[3]CDCL SAT-solving algorithms, and SMT-solving ones thereof, require the input formula to be in conjunctive normal form (CNF), i.e., a conjunction of *clauses*, each clause being a disjunction of propositions or of their negations. Since they very-effectively pre-convert input formulae into CNF [36], we assume wlog input formulae to have any form.

of up to $10^6 - 10^7$ variables. Modern SAT solvers are based on the conflict-driven clause-learning (CDCL) schema [37], and adopt a variety of very-efficient search techniques [38].

In the contexts of automated reasoning (AR) and formal verification (FV), important *decision* problems are effectively encoded into and solved as Satisfiability Modulo Theories (SMT) problems [39]. SMT is the problem of deciding the satisfiability of a (typically quantifier-free) first-order formula with respect to some decidable *background theory* $\mathcal{T}$, which can also be a combination of theories $\bigcup_i \mathcal{T}_i$. Theories of practical interest are, e.g., those of equality and uninterpreted functions ($\mathcal{EUF}$), of linear arithmetic over the rationals ($\mathcal{LRA}$) or over the integers ($\mathcal{LIA}$), of non-linear arithmetic over the reals ($\mathcal{NLA}$), of arrays ($\mathcal{AR}$), of bit-vectors ($\mathcal{BV}$), and their combinations.

In the last decade efficient SMT solvers have been developed following the so-called *lazy approach*, that combines the power of modern CDCL SAT solvers with the expressivity of dedicated decision procedures ($\mathcal{T}$-*solvers*) for several first-order theories of interest. Modern lazy SMT solvers —like e.g. CVC4[4], MathSAT5[5], Yices[6], Z3[7]— combine a variety of solving techniques coming from very heterogeneous domains. We refer the reader to [40, 13] for an overview on lazy SMT solving, and to the URLs of the above solvers for a description of their supported theories and functionalities.

More recently, SMT has also been leveraged from decision to optimization. *Optimization Modulo Theories* (*OMT*) [14, 18, 19, 20], is the problem of finding a *model* for an SMT formula φ which minimizes the value of some arithmetical cost function. References [18, 19, 20] present some general OMT procedure adding to SMT the capability of finding models minimizing cost functions in $\mathcal{LRA}$. This problem is denoted OMT($\mathcal{LRA}$) if only the $\mathcal{LRA}$ theory is involved in the SMT formula, OMT($\mathcal{LRA} \cup \mathcal{T}$) if some other theories are involved. Such procedures combine standard lazy SMT-solving with LP minimization techniques. OMT($\mathcal{LRA} \cup \mathcal{T}$) procedures have been implemented into the OptiMathSAT tool,[8] a sub-branch of MathSAT5.

---

[4] http://cvc4.cs.nyu.edu/

[5] http://mathsat.fbk.eu/

[6] http://yices.csl.sri.com/

[7] ttp://research.microsoft.com/en-us/um/redmond/projects/z3/ml/z3.html

[8] http://optimathsat.disi.unitn.it/

Example 3.1. *Consider the following toy $\mathcal{LRA}$-formula φ:*

$$(cost = x+y) \wedge (x \geq 0) \wedge (y \geq 0) \wedge (A \vee (4x+y-4 \geq 0)) \wedge (\neg A \vee (2x+3y-6 \geq 0))$$

and the OMT($\mathcal{LRA}$) problem of finding the model of φ (if any) which makes the value of cost minimum. In fact, depending on the truth value of A, there are two possible alternative sets of constraints to minimize:

$$\{A, (cost = x + y), (x \geq 0), (y \geq 0), (2x + 3y - 6 \geq 0)\}$$
$$\{\neg A, (cost = x + y), (x \geq 0), (y \geq 0), (4x + y - 4 \geq 0)\}$$

whose minimum-cost models are, respectively:

$$\{A = True, x = 0.0, y = 2.0, cost = 2.0\}$$
$$\{A = False, x = 1.0, y = 0.0, cost = 1.0\}$$

from which we can conclude that the latter is a minimum-cost model for φ.

Overall, for the scope of this paper, it is important to highlight the fact that OMT solvers are available which, thanks to the underlying SAT and SMT technologies, can handle problems with a large number of hybrid variables (in the order of thousands, at least for the $\mathcal{LRA}$ theory).

To this extent, we notice that the underlying theories and $\mathcal{T}$-*solvers* provide the meaning and the reasoning capabilities for specific predicates and function symbols (e.g., the $\mathcal{LRA}$-specific symbols "$\geq$" and "$+$", or the $\mathcal{AR}$-specific symbols "read(...)", "write(...)") that would otherwise be very difficult to describe, or to reason over, with logic-based automated reasoning tools —e.g., traditional first-order theorem provers cannot handle arithmetical reasoning efficiently— or with arithmetical ones —e.g., DLP, ILP, MILP, LGDP tools [41, 42, 43] or CLP tools [44, 45, 46] do not handle *symbolic* theory-reasoning on theories like $\mathcal{EUF}$ or $\mathcal{AR}$. Also, the underlying CDCL SAT solver allows SMT solvers to handle a large amount of Boolean reasoning very efficiently, which is typically out of the reach of both first-order theorem provers and arithmetical tools.

These facts motivate our choice of using SMT/OMT technology, and hence the tool OptiMathSAT, as workhorse engines for reasoning in hybrid domains. Hereafter in the paper we consider only plain OMT($\mathcal{LRA}$).

Another prospective advantage of SMT technology is that modern SMT solvers (e.g., MathSAT5, Z3, . . .) have an *incremental interface*, which allows for solving sequences of "similar" formulae without restarting the search

from scratch at each new formula, and instead reusing "common" parts of the search performed for previous formulae (see, e.g., [47]). This drastically improves overall performance on sequences of similar formulae. An incremental extension of OPTIMATHSAT, fully exploiting that of MATHSAT5, is currently available.

Note that a current limitation of SMT solvers is that, unlike traditional theorem provers, they typically handle efficiently only quantifier-free formulae. Attempts at extending SMT to quantified formulae have been made in the literature [48, 49, 50], and a few SMT solvers (e.g., Z3) do provide some support for quantified formulae. However, the state of the art of these extensions is still far from being satisfactory. Nonetheless, the method we present in the paper can be easily adapted to deal with these types of extensions once they reach the required level of maturity.

4. Learning Modulo Theories using Cutting Planes

4.1. An introductory example

In order to introduce the LMT framework, we start with a toy learning example. We are given a unit-length bounding box, $[0, 1] \times [0, 1]$, that contains a given, fixed block (rectangle), as in Figure 1 (a). The block is identified by the four constants (x_1, y_1, dx_1, dy_1), where x_1, y_1 indicate the bottom-left corner of the rectangle, and dx_1, dy_1 its width and height, respectively. Now, suppose that we are assigned the task of fitting another block, identified by the variables (x_2, y_2, dx_2, dy_2), in the same bounding box, so as to minimize the following *cost* function:

$$cost := w_1 \times dx_2 + w_2 \times dy_2 \tag{1}$$

with the additional requirements that (i) the two blocks "touch" either from above, below, or sideways, and (ii) the two blocks do not overlap.

It is easy to see that the weights w_1 and w_2 control the shape and location of the optimal solution. If both weights are positive, then the cost is minimized by any block of null size located along the perimeter of block 1. If both weights are negative and $w_1 \ll w_2$, then the optimal block will be placed so as to occupy as much horizontal space as possible, while if $w_1 \gg w_2$ it will prefer to occupy as much vertical space as possible, as in Figure 1 (b,c). If w_1 and w_2 are close, then the optimal solution depends on the relative amount of available vertical and horizontal space in the bounding box.
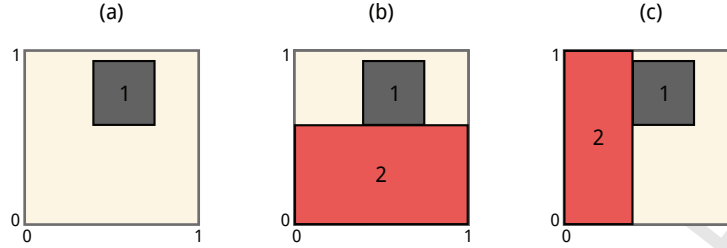
11

Figure 1: (a) Initial configuration. (b) Optimal configuration for $w_1 \ll w_2$. (c) Optimal configuration for $w_1 \gg w_2$.

This toy example illustrates two key points. First, the problem involves a mixture of *numerical variables* (coordinates, sizes of block 2) and *Boolean variables*, along with *hard rules* that control the feasible space of the optimization procedure (conditions (i) and (ii)), and *costs* — or soft rules — which control the shape of the optimization landscape. This is the kind of problem that can be solved in terms of Optimization Modulo Linear Arithmetic, OMT($\mathcal{LRA}$). Second, it is possible to estimate the weights w_1, w_2 from data in order to learn what kind of blocks are to be considered optimal. The goal of our learning procedure is precisely to find a good set of weights from examples. In the following we will describe how such a learning task can be framed within the structured output SVMs framework.

4.2. Notation

Symbol	Meaning
`above`, `right`, ...	Boolean variables
$x, y, dx, \ldots$	Rational variables
$(\boldsymbol{I}, \boldsymbol{O})$	Complete object; $\boldsymbol{I}$ is the input, $\boldsymbol{O}$ is the output
$\varphi_1, \ldots, \varphi_m$	Constraints
$\mathbb{1}_k(\boldsymbol{I}, \boldsymbol{O})$	Indicator for Boolean constraint φ_k over $(\boldsymbol{I}, \boldsymbol{O})$
$c_k(\boldsymbol{I}, \boldsymbol{O})$	Cost for arithmetical constraint φ_k over $(\boldsymbol{I}, \boldsymbol{O})$
$\boldsymbol{\psi}(\boldsymbol{I}, \boldsymbol{O})$	Feature representation of the complete object
$\psi_k(\boldsymbol{I}, \boldsymbol{O}) := \mathbb{1}_k(\boldsymbol{I}, \boldsymbol{O})$	Feature associated to Boolean constraint φ_k
$\psi_k(\boldsymbol{I}, \boldsymbol{O}) := -c_k(\boldsymbol{I}, \boldsymbol{O})$	Feature associated to arithmetical constraint φ_k
$\boldsymbol{w}$	Weights

Table 1: Explanation of the notation used throughout the text.

We consider the problem of learning from a training set of n complex objects $\{(\boldsymbol{I}_i, \boldsymbol{O}_i)\}_{i=1}^n$, where each object $(\boldsymbol{I}, \boldsymbol{O})$ is represented as a set of Boolean and rational variables:

$$(\boldsymbol{I}, \boldsymbol{O}) \in \underbrace{(\{\top, \bot\} \times \ldots \times \{\top, \bot\})}_{\text{Boolean part}} \times \underbrace{(\mathbb{Q} \times \ldots \times \mathbb{Q})}_{\text{rational part}}$$

We indicate Boolean variables using predicates such as $\mathtt{touching}(i, j)$, and write rational variables as lower-case letters, e.g. $cost$, $distance$, x, y. Please note that while we write Boolean variables using a First-Order syntax for readability, our method does require the grounding of all Boolean predicates prior to learning and inference. In the present formulation, we assume objects to be composed of two parts: $\boldsymbol{I}$ is the $input$ (or observed) part, while $\boldsymbol{O}$ is the $output$ (or query) part.[9] The learning problem is defined by a set of m $constraints$ $\{\varphi_k\}_{k=1}^m$. Each constraint φ_k is either a Boolean- or rational-valued function of the object $(\boldsymbol{I}, \boldsymbol{O})$. For each Boolean-valued constraint φ_k, we denote its $indicator$ $function$ as $\mathbb{1}_k(\boldsymbol{I}, \boldsymbol{O})$, which evaluates to 1 if the constraint is satisfied and to -1 otherwise (the choice of -1 to represent falsity is customary in the max-margin literature). Similarly, we refer to the $cost$ of a rational-valued constraint φ_k as $c_k(\boldsymbol{I}, \boldsymbol{O}) \in \mathbb{Q}$. The feature space representation of an object $(\boldsymbol{I}, \boldsymbol{O})$ is given by the $feature$ $vector$ $\boldsymbol{\psi}(\boldsymbol{I}, \boldsymbol{O})$, which is a function of the constraints. Each $soft$ constraint φ_k has an associated finite weight $w_k \in \mathbb{Q}$ (to be learned from the data), while $hard$ constraints have no associated weight. We denote the vector of learned weights as $\boldsymbol{w} := (w_1, w_2, \ldots, w_m)$, and its Euclidean norm as $\|\boldsymbol{w}\|$. Table 1 summarizes the notation used throughout the text.

4.3. A Structural SVM approach to LMT

Structured-output SVMs [23] are a very flexible framework that generalizes max-margin methods to the prediction of complex outputs such as strings, trees and graphs. In this setting the association between inputs $\boldsymbol{I}$ and outputs $\boldsymbol{O}$ is controlled by a so-called $compatibility$ $function$ $f(\boldsymbol{I}, \boldsymbol{O}) := \boldsymbol{w}^\top \boldsymbol{\psi}(\boldsymbol{I}, \boldsymbol{O})$ defined as a linear combination of the joint feature space representation $\boldsymbol{\psi}(\boldsymbol{I}, \boldsymbol{O})$ of the input-output pair. Inference amounts to

---

[9]We depart from the conventional x/y notation for indicating input/output pairs to avoid name clashes with the block coordinate variables.

13

finding the most compatible output O^* for a given input I, which equates to solving the following optimization problem:

$$O^* = \mathrm{argmax}_{O} f(I, O) = \mathrm{argmax}_{O} w^\top \psi(I, O) \qquad (2)$$

Performing inference on structured domains is non-trivial, since the maximization ranges over an exponential (and possibly unbounded) number of candidate outputs.

Learning is formulated within the regularized empirical risk minimization framework. In order to learn the weights from a training set of n examples, one needs to define a non-negative *loss function* $\Delta(I, O, O')$ that, for any given observation I, quantifies the penalty incurred when predicting O' instead of the correct output O. Learning can be expressed as the problem of finding the weights w that minimize the per-instance error ξ_i and the model complexity [23]:

$$\underset{w, \xi}{\mathrm{argmin}} \ \frac{1}{2}\|w\|^2 + \frac{C}{n}\sum_{i=1}^{n} \xi_i \qquad (3)$$

s.t. $w^\top(\psi(I_i, O_i) - \psi(I_i, O')) \geq \Delta(I_i, O_i, O') - \xi_i \ \forall \ i = 1, \ldots, n; \ O' \neq O_i$

Here the constraints require that the compatibility between any input I_i and its corresponding correct output O_i is always higher than that with all wrong outputs O' by a margin, with ξ_i playing the role of per-instance violations. This formulation is called n-slack margin rescaling and it is the original and most accessible formulation of structured-output SVMs. See [51] for an extensive exposition of alternative formulations.

Weight learning is a quadratic program, and can be solved very efficiently with a cutting-plane (CP) algorithm [23]. Since in Eq (3) there is an exponential number of constraints, it is infeasible to naively account for all of them during learning. Based on the observations that the constraints obey a subsumption relation, the CP algorithm [51] sidesteps the issue by keeping a working set of active constraints $\mathcal{W}$: at each iteration, it augments the working set with the most violated constraint, and then solves the corresponding reduced quadratic program using a standard SVM solver. This procedure is guaranteed to find an ϵ-approximate solution to the QP in a polynomial number of iterations, independently of the cardinality of the output space and of the number of examples n [23]. The n-slack margin rescaling version of the CP algorithm can be found in Algorithm 1 (adapted from [51]). Please

14

Data: Training instances $\{(\boldsymbol{I}_1, \boldsymbol{O}_1), \ldots, (\boldsymbol{I}_n, \boldsymbol{O}_n)\}$, parameters C, ϵ
Result: Learned weights $\boldsymbol{w}$

1   $\mathcal{W}_i \leftarrow \emptyset$, $\xi_i \leftarrow 0$ for all $i = 1, \ldots, n$;
2   **repeat**
3   **for** $i = 1, \ldots, n$ **do**
4          $\boldsymbol{O}'_i \leftarrow \mathrm{argmax}_{\boldsymbol{O}'} \left[\Delta(\boldsymbol{O}_i, \boldsymbol{O}') - \boldsymbol{w}^\top (\boldsymbol{\psi}(\boldsymbol{I}_i, \boldsymbol{O}_i) - \boldsymbol{\psi}(\boldsymbol{I}_i, \boldsymbol{O}')) \right]$;
5   **if** $\Delta(\boldsymbol{O}_i, \boldsymbol{O}'_i) - \boldsymbol{w}^\top (\boldsymbol{\psi}(\boldsymbol{I}_i, \boldsymbol{O}_i) - \boldsymbol{\psi}(\boldsymbol{I}_i, \boldsymbol{O}'_i)) > \boldsymbol{\xi}_i + \epsilon$ **then**
6

$$
\begin{aligned}
\mathcal{W}_i \quad &\leftarrow \quad \mathcal{W}_i \cup \{\boldsymbol{O}'_i\} \\
\boldsymbol{w}, \boldsymbol{\xi} \quad &\leftarrow \quad \underset{\boldsymbol{w}, \boldsymbol{\xi} \geq 0}{\mathrm{argmin}} \, \frac{1}{2} \boldsymbol{w}^\top \boldsymbol{w} + \frac{C}{n} \sum_{i=1}^{n} \xi_i \\
&\quad\;\; \text{s.t.} \quad \forall \boldsymbol{O}'_1 \in \mathcal{W}_1 \, : \, \boldsymbol{w}^\top \left[\boldsymbol{\psi}(\boldsymbol{I}_1, \boldsymbol{O}_1) - \boldsymbol{\psi}(\boldsymbol{I}_1, \boldsymbol{O}'_1) \right] \geq \\
&\qquad\qquad\qquad \Delta(\boldsymbol{O}_i, \boldsymbol{O}'_1) - \xi_1 \\
&\qquad\qquad \vdots \\
&\qquad\qquad \forall \boldsymbol{O}'_n \in \mathcal{W}_n \, : \, \boldsymbol{w}^\top \left[\boldsymbol{\psi}(\boldsymbol{I}_n, \boldsymbol{O}_n) - \boldsymbol{\psi}(\boldsymbol{I}_n, \boldsymbol{O}'_n) \right] \geq \\
&\qquad\qquad\qquad \Delta(\boldsymbol{O}_i, \boldsymbol{O}'_n) - \xi_n
\end{aligned}
$$

7   **end**
8   **end**
9   **until** *no $\mathcal{W}_i$ has changed during iteration*;
10   **return** $\boldsymbol{w}$

Algorithm 1: Cutting-plane algorithm for training structural SVMs, according to the *n*-slack formulation presented in [51].

note that in our experiments we make use of the faster, but otherwise equivalent, 1-slack margin rescaling variant [51]. We report the *n*-slack margin rescaling version here for ease of exposition.

The CP algorithm is generic, meaning that it can be adapted to any structured prediction problem as long as it is provided with: (i) a joint feature space representation $\boldsymbol{\psi}$ of input-output pairs (and consequently a compatibility function f); (ii) an oracle to perform inference, i.e. to solve Equation (2); and (iii) an oracle to retrieve the most violated constraint of the QP, i.e. to solve the *separation* problem:

$$
\mathrm{argmax}_{\boldsymbol{O}'} \boldsymbol{w}^T \boldsymbol{\psi}(\boldsymbol{I}_i, \boldsymbol{O}') + \Delta(\boldsymbol{I}_i, \boldsymbol{O}_i, \boldsymbol{O}') \tag{4}
$$

15

The two oracles are used as sub-routines during the optimization procedure. For a more detailed account, and in particular for the derivation of the separation oracle formulation, please refer to [23].

One key aspect of the structured output SVMs is that efficient implementations of the two oracles are fundamental for the learning task to be tractable in practice. The idea behind Learning Modulo Theories is that, when a hybrid Boolean-numerical problem can be encoded in SMT, the two oracles can be implemented using an Optimization Modulo Theory solver. This is precisely what we propose in the present paper. In the following sections we show how to define a feature space for hybrid Boolean-numerical learning problems, and how to use OMT solvers to efficiently perform inference and separation.

4.4. Learning Modulo Theories with OMT

Let us formalize the previous toy example in the language of LMT. In the following we give a step-by-step description of all the building blocks of an LMT problem: the background knowledge, the hard and soft constraints, the cost function, and the loss function.

Input, Output, and Background Knowledge. Here the input $\boldsymbol{I}$ to the problem is the observed block (x_1, y_1, dx_1, dy_1) while the output $\boldsymbol{O}$ is the generated block (x_2, y_2, dx_2, dy_2). In order to encode the set of constraints $\{\varphi_k\}$ that underlie both the learning and the inference problems, it is convenient to first introduce a background knowledge of predicates expressing facts about the relative positioning of blocks. To this end we add a fresh predicate $\mathtt{left}(i, j)$, that encodes the fact that "a generic block of index i touches a second block j from the left", defined as follows:

$$\mathtt{left}(i, j) := (x_i + dx_i = x_j) \wedge$$
$$((y_j \leq y_i \leq y_j + dy_j) \vee (y_j \leq y_i + dy_i \leq y_j + dy_j))$$

Similarly, we add analogous predicates for the other directions: $\mathtt{right}(i, j)$, $\mathtt{below}(i, j)$, $\mathtt{over}(i, j)$ (see Table 2 for the full definitions).

Hard constraints. The hard constraints represent the fact that the output $\boldsymbol{O}$ should be a valid block within the bounding box (all the constraints φ_k are implicitly conjoined):

$$0 \leq x_2, y_2, dx_2, dy_2 \leq 1 \qquad (x_2 + dx_2) \leq 1 \wedge (y_2 + dy_2) \leq 1$$

16

Then we require the output block $\boldsymbol{O}$ to "touch" the input block $\boldsymbol{I}$:

$$\texttt{left}(1,2) \vee \texttt{right}(1,2) \vee \texttt{below}(1,2) \vee \texttt{over}(1,2)$$

Note that whenever this rule is satisfied, both conditions (i) and (ii) of the toy example hold, i.e. touching blocks never overlap.

Touching blocks	
	$\texttt{left}(i,j) := x_i + dx_i = x_j \wedge$
Block i touches block j, left	$((y_j \leq y_i \leq y_j + dy_j) \vee$
	$(y_j \leq y_i + dy_i \leq y_j + dy_j))$
	$\texttt{below}(i,j) := y_i + dy_i = y_j \wedge$
Block i touches block j, below	$((x_j \leq x_i \leq x_j + dx_j) \vee$
	$(x_j \leq x_i + dx_i \leq x_j + dx_j))$
Block i touches block j, right	$\texttt{right}(i,j) := \text{Analogous to } \texttt{left}(i,j)$
Block i touches block j, over	$\texttt{over}(i,j) := \text{Analogous to } \texttt{below}(i,j)$

Figure 2: Background knowledge used in the toy block example.

Cost function. Finally, we encode the cost function $cost = w_1 dx_2 + w_2 dy_2$, completing the description of the optimization problem. In the following we will see that the definition of the cost function implicitly defines also the set of features, or equivalently the set of soft constraints, of the LMT problem.

Soft constraints and Features. Now, suppose we were given a training set of instances analogous to those pictured in Figure 1 (c), i.e. where the supervision includes output blocks that preferentially fill as much vertical space as possible. The learning algorithm should be able to learn this preference by inferring appropriate weights. This kind of learning task can be cast within the structured SVM framework, by defining an appropriate joint feature space $\boldsymbol{\psi}$ and oracles for the inference and separation problems.

Let us focus on the feature space first. Our definition is grounded on the concept of *reward* assigned to an object $(\boldsymbol{I}, \boldsymbol{O})$ with respect to the set of formulae $\{\varphi_k\}_{k=1}^m$. We construct the feature vector

$$\boldsymbol{\psi}(\boldsymbol{I}, \boldsymbol{O}) := (\psi_1(\boldsymbol{I}, \boldsymbol{O}), \ldots, \psi_m(\boldsymbol{I}, \boldsymbol{O}))^\top$$

by collating m per-formula rewards $\psi_k(\boldsymbol{I}, \boldsymbol{O})$, where:

$$\psi_k(\boldsymbol{I}, \boldsymbol{O}) := \begin{cases} \mathbb{1}_k(\boldsymbol{I}, \boldsymbol{O}) & \text{if } \varphi_k \text{ is Boolean} \\ -c_k(\boldsymbol{I}, \boldsymbol{O}) & \text{if } \varphi_k \text{ is arithmetical} \end{cases}$$

17

Here $\mathbb{1}_k$ is an indicator for the satisfaction of a Boolean constraint φ_k, while c_k denotes the cost associated to real-valued constraints, please refer to Table 1 for more details. In other words, the feature representation of a complex object $(\boldsymbol{I}, \boldsymbol{O})$ is the vector of all indicator/cost functions associated to the soft constraints. Returning to the toy example, where the cost function is

$$cost := w_1 \times dx_2 + w_2 \times dy_2 = \boldsymbol{w}^\top \boldsymbol{\psi}(\boldsymbol{I}, \boldsymbol{O})$$

the feature space of an instance $(\boldsymbol{I}, \boldsymbol{O})$ is simply $\boldsymbol{\psi}(\boldsymbol{I}, \boldsymbol{O}) = (-dx_2, -dy_2)^\top$, which reflects the size of the output block $\boldsymbol{O}$. The negative sign here is due to interpreting the features as *rewards* (to be maximized), while the corresponding soft constraints can be seen as *costs* (to be minimized); see Eq 5 where this relationship is made explicit.

According to this definition both satisfied and unsatisfied rules contribute to the total reward, and two objects $(\boldsymbol{I}, \boldsymbol{O})$, $(\boldsymbol{I}', \boldsymbol{O}')$ that satisfy/violate similar sets of constraints will be close in feature space. The compatibility function $f(\boldsymbol{I}, \boldsymbol{O}) := \boldsymbol{w}^\top \boldsymbol{\psi}(\boldsymbol{I}, \boldsymbol{O})$ computes the (weighted) total reward assigned to $(\boldsymbol{I}, \boldsymbol{O})$ with respect to the constraints. Using this definition, the maximization in the inference (Equation 2) can be seen as attempting to find the output $\boldsymbol{O}$ that maximizes the total reward with respect to the input $\boldsymbol{I}$ and the rules, or equivalently the one with minimum cost. Since $\boldsymbol{\psi}$ can be expressed in terms of Satisfiability Modulo Linear Arithmetic, the latter minimization problem can be readily cast as an OMT problem. Translating back to the example, maximizing the compatibility function f boils down to:

$$\operatorname{argmax} \boldsymbol{w}^\top \boldsymbol{\psi}(\boldsymbol{I}, \boldsymbol{O}) = \operatorname{argmax} (-dx_2, -dy_2)\boldsymbol{w} = \operatorname{argmin} (dx_2, dy_2)\boldsymbol{w} \quad (5)$$

which is exactly the cost minimization problem in Equation 1.

Loss function. The loss function determines the dissimilarity between output structures, which in our case contain a mixture of Boolean and rational variables. We observe that by picking a loss function expressible as an OMT($\mathcal{LRA}$) problem, we can readily use the same OMT solver used for inference to also solve the CP separation oracle (Equation (4)). This can be achieved by selecting a loss function such as the following Hamming loss in

18

feature space:

$$
\begin{aligned}
\Delta(\boldsymbol{I}, \boldsymbol{O}, \boldsymbol{O'}) \ &:= \sum_{k\,:\,\varphi_k \text{ is Boolean}} |\mathbb{1}_k(\boldsymbol{I}, \boldsymbol{O}) - \mathbb{1}_k(\boldsymbol{I}, \boldsymbol{O'})| + \\
&\qquad \sum_{k\,:\,\varphi_k \text{ is arithmetical}} |c_k(\boldsymbol{I}, \boldsymbol{O}) - c_k(\boldsymbol{I}, \boldsymbol{O'})| \\
&= \ \|\boldsymbol{\psi}(\boldsymbol{I}, \boldsymbol{O}) - \boldsymbol{\psi}(\boldsymbol{I}, \boldsymbol{O'})\|_1
\end{aligned}
$$

This loss function is piecewise-linear, and as such satisfies the desideratum.

Since both the inference and separation oracles required by the CP algorithm can be encoded in OMT($\mathcal{LRA}$), we can apply an OMT solver to efficiently solve the learning task. In particular, our current implementation is based on a vanilla copy of SVM$^{\text{struct}}$ [10], which acts as a cutting-plane solver, whereas inference and separation are implemented with the OPTIMATHSAT OMT solver.

To summarize, an LMT problem can be broken down into several components: a *background knowledge*, a set of soft and hard *constraints*, a *cost function* and a *loss function*. The background knowledge amounts to a set of SMT formulae and constants useful for encoding the problem constraints, which in turn determine the relation between inputs and outputs. The hard constraints define the space of candidate outputs, while the soft constraints correspond one-to-one to features. The overall cost function is a linear combination of the dissatisfaction/cost (or, equivalently, satisfaction/reward) associated to the individual soft constraints, and as such is controlled entirely by the choice of features. Finally, the loss function determines the dissimilarity between output structures. While in the present paper we focused on a Hamming loss in feature space, LMT can work with any loss function that can be encoded as an SMT formula.

5. Experimental Evaluation

In the following we evaluate LMT on two novel design problems that stress the ability of LMT to deal with rather complex mixed Boolean-numerical problems.

---

[10]http://www.cs.cornell.edu/people/tj/svm_light/svm_struct.html.

5.1. Stairway to Heaven

In this section we are interested in learning how to assemble different kinds of *stairways* from examples. For the purpose of this paper, a stairway is simply a collection of m blocks (rectangles) located within a two-dimensional, unit-sized bounding box $[0, 1] \times [0, 1]$. Clearly not all possible arrangements of blocks form a stairway; a stairway must satisfy the following conditions: (i) the first block touches either the top or the bottom corner of the left edge of the bounding box; (ii) the last block touches the opposite corner at the right edge of the bounding box; (iii) there are no gaps between consecutive blocks; (iv) consecutive blocks must actually form a step and, (v) no two blocks overlap. Note that the property of "being a stairway" is a *collective* property of the m blocks.

More formally, each block $i = 1, \ldots, m$ consists of four rational variables: the origin (x_i, y_i), which indicates the bottom-left corner of the block, a width dx_i and a height dy_i; the top-right corner of the block is $(x_i + dx_i, y_i + dy_i)$. A stairway is simply an assignment to all $4 \times m$ variables that satisfies the above conditions.

Our definition does not impose any constraint on the *orientation* of stairways: it is perfectly legitimate to have *left* stairways that start at the top-left corner of the bounding box and reach the bottom-right corner, and *right* stairways that connect the bottom-left corner to the top-right one. For instance, a left 2-stairway can be defined with the following block assignment (see Figure 3 (a)):

$$(x_1, y_1, dx_1, dy_1) = \left(0, \frac{1}{2}, \frac{1}{2}, \frac{1}{2}\right) \qquad (x_2, y_2, dx_2, dy_2) = \left(\frac{1}{2}, 0, \frac{1}{2}, \frac{1}{2}\right)$$

Similarly, a right 2-stairway is obtained with the assignment (Figure 3 (b)):

$$(x_1, y_1, dx_1, dy_1) = \left(0, 0, \frac{1}{2}, \frac{1}{2}\right) \qquad (x_2, y_2, dx_2, dy_2) = \left(\frac{1}{2}, \frac{1}{2}, \frac{1}{2}, \frac{1}{2}\right)$$

We also note that the above conditions do not impose any explicit restriction on the width and height of individual blocks (as long as consecutive ones are in contact and there is no overlap). Consequently we allow for both *ladder* stairways, where the total amount of vertical and horizontal surface of the individual blocks is minimal, as in Figure 3 (a) and (b); and for *pillar* stairways, where either the vertical or horizontal block lengths are maximized, as in Figure 3 (c). There are of course an uncountable number of intermediate stairways that do not belong to either of the above categories.
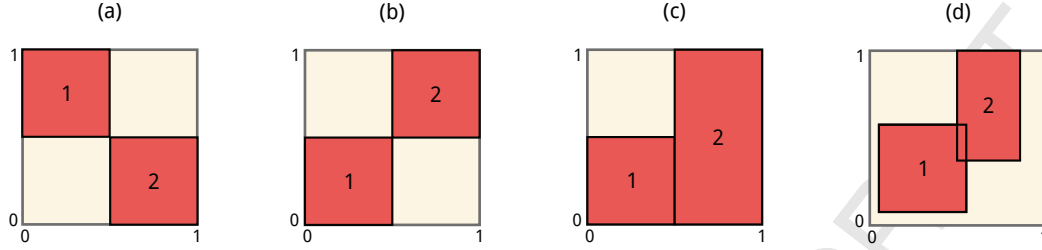
Figure 3: (a) A *left ladder* 2-stairway. (b) A *right ladder* 2-stairway. (c) A *right pillars* 2-stairway. (d) A block assignment that violates conditions (i), (ii) and (iv), and as such does not form a stairway.

Inference amounts to generating a set of variable assignments to all blocks, so that none of conditions (i)-(v) is violated, and the cost of the soft rules is minimized[11]. This can be easily encoded as an OMT($\mathcal{LRA}$) problem. As a first step, we define a background knowledge of useful predicates. We use four predicates to encode the fact that a block i may touch one of the four corners of the bounding box, namely $\texttt{bottom\_left}(i)$, $\texttt{bottom\_right}(i)$, $\texttt{top\_left}(i)$, and $\texttt{top\_right}(i)$, which can be written as, e.g.:

$$\texttt{bottom\_right}(i) := (x_i + dx_i) = 1 \wedge y_i = 0$$

We also define predicates to describe the relative positions of two blocks i and j, such as $\texttt{left}(i, j)$:

$$\begin{aligned}
\texttt{left}(i, j) := (x_i + dx_i) = x_j \wedge \\
((y_j \leq y_i \leq y_j + dy_j) \vee \\
(y_j \leq y_i + dy_i \leq y_j + dy_j))
\end{aligned}$$

encodes the fact that block i is touching block j from the left. Similarly, we also define $\texttt{below}(i, j)$ and $\texttt{over}(i, j)$. Finally, and most importantly, we combine the above predicates to define the concept of *step*, i.e. two blocks i and $i + 1$ that are both touching and positioned as to form a stair:

$$\begin{aligned}
\texttt{left\_step}(i, j) := (\texttt{left}(i, j) \wedge (y_i + dy_i) > (y_j + dy_j)) \vee \\
(\texttt{over}(i, j) \wedge (x_i + dx_i) < (x_j + dx_j))
\end{aligned}$$

---

[11] Note that as will be detailed later on, the full set of constraints representing stairs will not be available during inference. We will rather provide a super-set of candidate soft constraints, and the training stage will be responsible of learning appropriate weights for the constraints characterising the type of stairs in the training set.

21

We define `right_step`(i,j) in the same manner. For a complete description of the background knowledge, see Table 2.

Corners	
Block i at bottom-left corner	$\texttt{bottom\_left}(i) := x_i = 0 \land y_i = 0$
Block i at bottom-right corner	$\texttt{bottom\_right}(i) := (x_i + dx_i) = 1 \land y_i = 0$
Block i at top-left corner	$\texttt{top\_left}(i) := x_i = 0 \land (y_i + dy_i) = 1$
Block i at top-right corner	$\texttt{top\_right}(i) := (x_i + dx_i) = 1 \land (y_i + dy_i) = 1$

Relative block positions	
Block i touches block j, left	$\texttt{left}(i,j) := (x_i + dx_i) = x_j \land$ $((y_j \leq y_i \leq y_j + dy_j) \lor$ $(y_j \leq y_i + dy_i \leq y_j + dy_j))$
Block i touches block j, below	$\texttt{below}(i,j) := (y_i + dy_i) = y_j \land$ $((x_j \leq x_i \leq x_j + dx_j) \lor$ $(x_j \leq x_i + dx_i \leq x_j + dx_j))$
Block i touches block j, over	$\texttt{over}(i,j) := (y_j + dy_j) = y_i \land$ $((x_j \leq x_i \leq x_j + dx_j) \lor$ $(x_j \leq x_i + dx_i \leq x_j + dx_j))$

Steps	
Left step	$\texttt{left\_step}(i,j) :=$ $(\texttt{left}(i,j) \land (y_i + dy_i) > (y_j + dy_j)) \lor$ $(\texttt{over}(i,j) \land (x_i + dx_i) < (x_j + dx_j))$
Right step	$\texttt{right\_step}(i,j) :=$ $(\texttt{left}(i,j) \land (y_i + dy_i) < (y_j + dy_j)) \lor$ $(\texttt{below}(i,j) \land (x_i + dx_i) < (x_j + dx_j))$

Table 2: Background knowledge used in the stairways experiment.

The background knowledge allows to encode the property of being a *left* stairway as:

$$\texttt{top\_left}(1) \land \bigwedge_{i \in [1,m-1]} \texttt{left\_step}(i, i+1) \land \texttt{bottom\_right}(m)$$

Analogously, any *right* stairway satisfies the following condition:

$$\texttt{bottom\_left}(1) \land \bigwedge_{i \in [1,m-1]} \texttt{right\_step}(i, i+1) \land \texttt{top\_right}(m)$$

22

However, our inference procedure does not have access to this knowledge. We rather encode an appropriate set of *soft rules* (costs) which, along with the associated weights, should bias the optimization towards block assignments that form a stairway of the correct type.

We include a few hard rules to constrain the space of admissible block assignments. We require that all blocks fall within the bounding box:

$$\forall i \; 0 \leq x_i, dx_i, y_i, dy_i \leq 1$$

$$\forall i \; 0 \leq (x_i + dx_i) \leq 1 \;\wedge\; 0 \leq (y_i + dy_i) \leq 1$$

We also require that blocks do not overlap:

$$\forall i \neq j \quad (x_i + dx_i \leq x_j) \vee (x_j + dx_j \leq x_i) \vee$$
$$(y_i + dy_i \leq y_j) \vee (y_j + dy_j \leq y_i)$$

Finally, we require (without loss of generality) blocks to be ordered from left to right, $\forall i \; x_i \leq x_{i+1}$.

Note that only condition (v) is modelled as a hard constraint. The others are implicitly part of the problem *cost*. Our cost model is based on the observation that it is possible to discriminate between the different stairway types using only four factors: minimum and maximum step size, and amount of horizontal and vertical *material*. These four factors are useful features in discriminating between the different stairway types without having to resort to quadratic terms, e.g. the areas of the individual blocks. For instance, in the cost we account for both the maximum step height of all left steps (a good stairway should not have too high steps):

$$maxshl = m \times \max_{i \in [1, m-1]} \begin{cases} (y_i + dy_i) - (y_{i+1} + dy_{i+1}) & \text{if } i, i+1 \text{ form a left step} \\ 1 & \text{otherwise} \end{cases}$$

and the minimum step width of all right steps (good stairways should have sufficiently large steps):

$$minswr = m \times \min_{i \in [1, m-1]} \begin{cases} (x_{i+1} + dx_{i+1}) - (x_i + dx_i) & \text{if } i, i+1 \text{ form a right step} \\ 0 & \text{otherwise} \end{cases}$$

The value of these costs depends on whether a pair of blocks actually forms a left step, a right step, or no step at all. Note that these costs are multiplied

23

by the number of blocks m. This allows to renormalize costs according to the number of steps; e.g. the step height of a stairway with m uniform steps is half that of a stairway with $m/2$ steps. Finally, we write the average amount of vertical material as $vmat = \frac{1}{m} \sum_i dy_i$. All the other costs can be written similarly; see Table 3 for the complete list. As we will see, the normalization of individual costs allows to learn weights which generalize to stairways with a larger number of blocks with respect to those seen during training.

Putting all the pieces together, the complete cost is:

$$
\begin{aligned}
cost \ := \ (& maxshl, minshl, maxshr, minshr, \\
& maxswl, minswl, maxswr, minswr, \\
& vmat, hmat) \ \boldsymbol{w}
\end{aligned}
$$

Minimizing the weighted cost implicitly requires the inference engine to decide whether it is preferable to generate a *left* or a *right* stairway, thanks to the $minshl, \dots, minswr$ components, and whether the stairway should be a *ladder* or *pillar*, due to $vmat$ and $hmat$. The actual weights are learned, allowing the learnt model to reproduce whichever stairway type is present in the training data.

(a) Hard constraints	
Bounding box	$\forall i \; 0 \leq x_i + dx_i \leq 1 \; \wedge$ $\quad 0 \leq y_i + dy_i \leq 1$
No overlap	$\forall i \neq j \; (x_i + dx_i \leq x_j) \vee (x_j + dx_j \leq x_i) \vee$ $\quad (y_i + dy_i \leq y_j) \vee (y_j + dy_j \leq y_i)$
Blocks left to right	$\forall i \; x_i \leq x_{i+1}$

(b) Soft constraints (features)	
Max step height left	$maxshl = m \times \max_i (y_i + dy_i) - (y_{i+1} + dy_{i+1})$
Min step height left	$minshl = m \times \min_i (y_i + dy_i) - (y_{i+1} + dy_{i+1})$
Max step height right	$maxshr = m \times \max_i (y_{i+1} + dy_{i+1}) - (y_i + dy_i)$
Min step height right	$minshr = m \times \min_i (y_{i+1} + dy_{i+1}) - (y_i + dy_i)$
Max step width left	$maxswl = m \times \max_i (x_{i+1} + dx_{i+1}) - (x_i + dx_i)$
Min step width left	$minswl = m \times \min_i (x_{i+1} + dx_{i+1}) - (x_i + dx_i)$
Max step width right	$maxswr = m \times \max_i (x_i + dx_i) - (x_{i+1} + dx_{i+1})$
Min step width right	$minswr = m \times \min_i (x_i + dx_i) - (x_{i+1} + dx_{i+1})$
Vertical material	$vmat = \frac{1}{m} \sum_i dy_i$
Horizontal material	$hmat = \frac{1}{m} \sum_i dx_i$

(c) Cost	
$cost := (maxshl, minshl, maxshr, minshr,$	
$maxswl, minswl, maxswr, minswr,$	
$vmat, hmat) \; \boldsymbol{w}$	

Table 3: List of all rules used in the stairway problem. Top, hard rules affect both the inference and separation procedures. Middle, soft rules (costs), whose weight is learned from data. Bottom, the total cost of a stairway instance is the weighted sum of all individual soft constraints.

To test the stairway scenario, we focused on learning one model for each of six kinds of stairway: *left ladder*, *right ladder*, *left pillar* and *right pillar* with a preference for horizontal blocks, and *left pillar* and *right pillar* with vertical blocks. In this setting, the input $\boldsymbol{I}$ is empty, and the model should generate all m blocks as output $\boldsymbol{O}$ during test.

We generated "perfect" stairways of 2 to 6 blocks for each stairway type

25

to be used as training instances. We then learned a model using all training instances up to a fixed number of blocks: a model using examples with up to 3 blocks, another with examples of up to 4, *etc.*, for a total of 4 models per stairway type. Then we analyzed the generalization ability of the learnt models by generating stairways with a *larger* number of blocks (up to 10) than those in the training set. The results can be found in Figure 4.
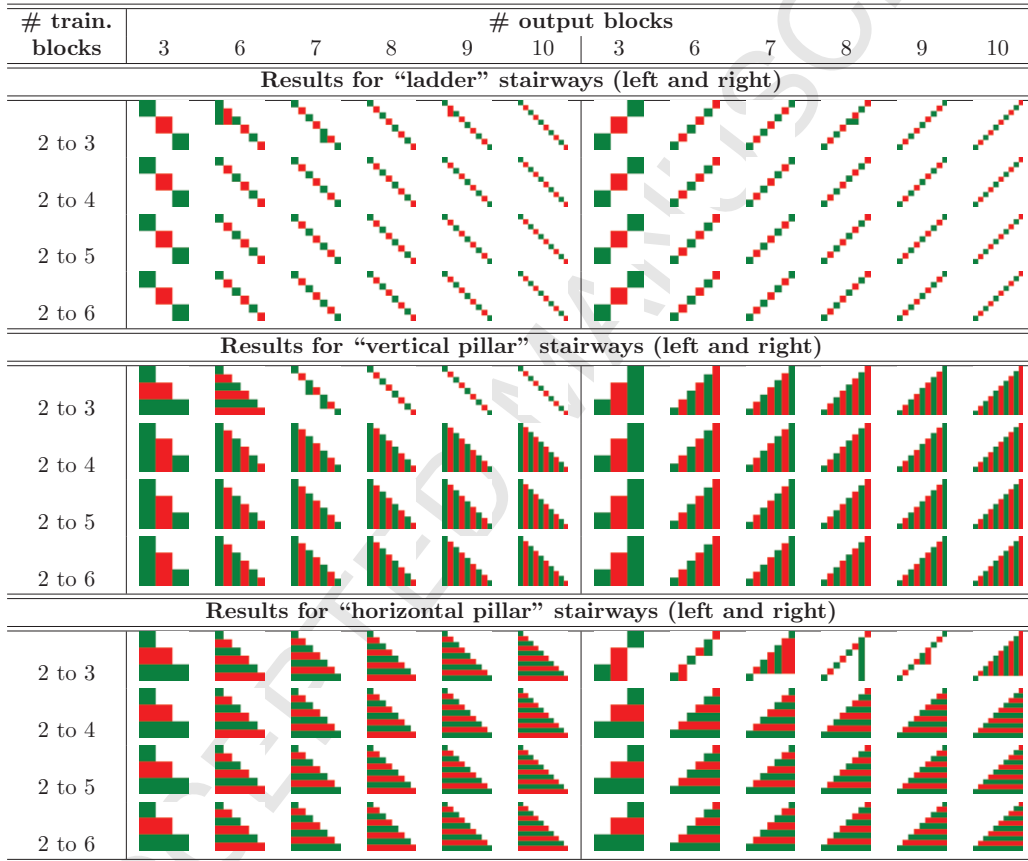
# train.	# output blocks											
blocks	3	6	7	8	9	10	3	6	7	8	9	10
Results for "ladder" stairways (left and right)												
2 to 3												
2 to 4												
2 to 5												
2 to 6												
Results for "vertical pillar" stairways (left and right)												
2 to 3												
2 to 4												
2 to 5												
2 to 6												
Results for "horizontal pillar" stairways (left and right)												
2 to 3												
2 to 4												
2 to 5												
2 to 6												

Figure 4: Results for the stairway construction problem. From top to bottom: results for the *ladder*, *horizontal pillar*, and *vertical pillar* cases. Images in row labeled with N picture the stairways generated by a model learnt on a training set of perfect stairways made of $2, 3, \ldots, N$ steps, with a varying number of generated blocks.

The experiment shows that LMT is able to solve the stairway construction problem, and can learn appropriate models for all stairway types, as expected. As can be seen in Figure 4, the generated stairways can present some imper-

fections when the training set is too small (e.g., only two training examples; first row of each table), especially in the 10 output blocks case. However, the situation quickly improves when the training set increases: models learned with four training examples are always able to produce perfect 10-block stairways of the same kind. Note again that the learner has no explicit notion of what a stairway is, but just the values of step width, height and material for some training examples of stairways.

All in all, albeit simple, this experiment showcases the ability of LMT to handle learning in hybrid Boolean-numerical domains, whereas other formalisms are not particularly suited for the task. As previously mentioned, the Church [7] language allows to encode arbitrary constraints over both numeric and Boolean variables. The stairway problem can indeed be encoded in Chuch in a rather straightforward way. However, the sampling strategies used for inference are not conceived for performing optimization with hard continuous constraints. Even the simple task of generating two blocks, conditioned on the fact that they form a step, is prohibitively expensive. [12]

5.2. Learning to Draw Characters

In this section we are concerned with automatic character drawing, a novel structured-output learning problem that consists in learning to translate any input noisy hand-drawn character into its symbolic representation. More specifically, given an unlabeled black-and-white image of a handwritten letter or digit, the goal is to construct an equivalent *vectorial* representation of the same character.

In this paper, we assume the character to be representable by a polyline made of a given number m of *directed* segments, i.e. segments identified by a starting point (x^b, y^b) and an ending point (x^e, y^e). The input image $\boldsymbol{I}$ is seen as the set P of coordinates of the pixels belonging to the character, while the output $\boldsymbol{O}$ is a set of m directed segments $\{(x_i^b, y_i^b, x_i^e, y_i^e)\}_{i=1}^m$. Just like in the previous section, we assume all coordinates to fall within the unit bounding box.

Intuitively, any good output $\boldsymbol{O}$ should satisfy the following requirements: (i) it should be as similar as possible to the noisy input character; and (ii) it should actually "look like" the corresponding vectorial character. Here we interpret the first condition as implying that the generated segments should

---

[12]We interrupted the inference process after 24 hours of computation.

cover as many pixels of the input image as possible (although alternative interpretations are possible). Under this interpretation, we can informally write the inference problem as follows:

$$\text{argmax}_{\boldsymbol{O}} \ (coverage(\boldsymbol{I}, \boldsymbol{O}), orientation(\boldsymbol{O})) \ \boldsymbol{w}$$

where the *orientation* term encodes information on the orientation of the segments which should be useful for computing the "looking like" condition. In the following, we will detail how to formulate and compute these two quantities.
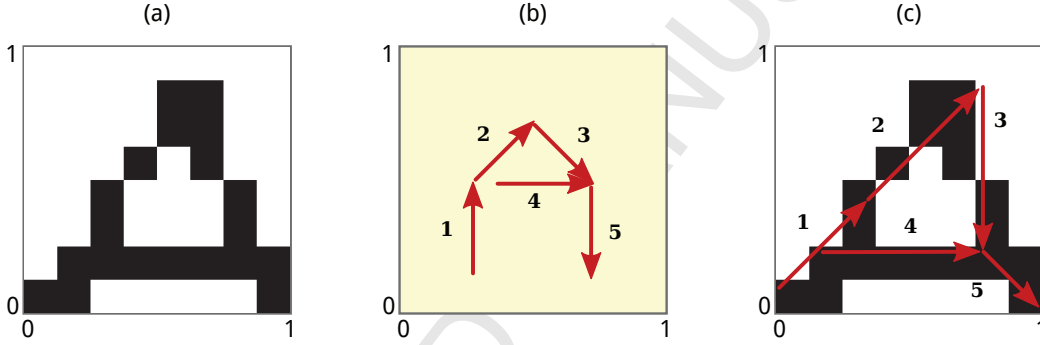
Figure 5:   Left, example 8×8 bitmap image of an "A". Middle, a set of 5 segments satisfying the "looking like an A" rules in the text. Right, 5 segments satisfying both the rules for character "A" and fitting the underlying image.

Since the output is supposed to be a polyline, we constrain consecutive segments to be connected, i.e. to share an endpoint:

$$\forall i \ \texttt{connected}(i, i+1)$$

We also want the segments to be no larger than the image, nor smaller than a pixel: $\forall i \ min\_length \leq length(i) \leq 1$. Additionally, we constrain (without loss of generality) each segment to be ordered from left to right, i.e. $x_i^b \leq x_i^e$. Finally, we restrict the segments to be either horizontal, vertical or 45° diagonal, that is:

$$\forall i \ \texttt{horizontal}(i) \lor \texttt{vertical}(i) \lor \texttt{diagonal}(i)$$

This restriction allows us express all numerical constraints in linear terms. All the predicates used above are defined as in Table 4.

Under these assumptions, we can encode the *coverage* reward as:

$$coverage(\boldsymbol{I}, \boldsymbol{O}) := \frac{1}{|P|} \sum_{p \in P} \mathbb{1}(\texttt{covered}(p))$$

where $\texttt{covered}(p)$ is true if pixel p is covered by at least one of the m segments:

$$\texttt{covered}(p) := \bigvee_{i \in [1,m]} \texttt{covered}(p, i)$$

The fact that a segment $i = (x_i^b, y_i^b, x_i^e, y_i^e)$ covers pixel $p = (x, y)$ implicitly depends on the orientation of the segment, and is computed using constructs like:

$$\text{If} \quad \texttt{horizontal}(i) \quad \text{then} \quad \texttt{covered}(p, i) := x_i^b \leq x \leq x_i^e \wedge y = y_i^b$$

The coverage formulae for the other segment types can be found in Table 4.

As for the *orientation* term, it should contain features related to the vectorial representation of characters. These include both the direction of the individual segments and the connections between pairs of segments. As an example, consider this possible description of "looking like an A":

$$
\begin{aligned}
\texttt{increasing}(1) &\quad \wedge \quad \texttt{h2t}(1, 2) \wedge \\
\texttt{increasing}(2) &\quad \wedge \quad \texttt{h2t}(2, 3) \wedge \\
\texttt{decreasing}(3) &\quad \wedge \quad \texttt{h2h}(3, 4) \wedge \\
\texttt{horizontal}(4) &\quad \wedge \quad \texttt{h2t}(4, 5) \wedge \\
\texttt{decreasing}(5) &
\end{aligned}
$$

Here $\texttt{increasing}(i)$ and $\texttt{decreasing}(i)$ indicate the direction of segment i, and can be written as:

$$
\begin{aligned}
\texttt{increasing}(i) &:= y_i^e > y_i^b \\
\texttt{decreasing}(i) &:= y_i^e < y_i^b
\end{aligned}
$$

Then we have connections between pairs of segments. We encode connection types following the convention used for Bayesian Networks, where the head of a directed segment is the edge containing the arrow (represented by the ending point (x^e, y^e)) and the tail is the opposite edge (the starting point

(x^b, y^b)). For instance, $\texttt{h2t}(i,j)$ indicates that i is head-to-tail with respect to j, $\texttt{h2h}(i,j)$ that they are head-to-head:

$$\texttt{h2t}(i,j) := (x_i^e = x_j^b) \wedge (y_i^e = y_j^b)$$
$$\texttt{h2h}(i,j) := (x_i^e = x_j^e) \wedge (y_i^e = y_j^e)$$

For a pictorial representation of the "looking like an A" constraint, see Figure 5 (b). We include a number of other, similar predicates in the background knowledge; for a full list, see Table 4.

For example, suppose we have a 8×8 image of an upper-case "A", as in Figure 5 (a). A character drawing algorithm should decide how to overlay 5 segments on top of the input image according to the previous two criteria. A good output would look like the one pictured in Figure 5 (c).

However, the formula for the "looking like an A" constraint is not available at test time and should be learned from the data. In order to do so, the *orientation* term includes possible directions ($\texttt{increasing}$, $\texttt{decreasing}$, $\texttt{right}$) for all m segments and all possible connection types between *all* segments ($\texttt{h2t}$, $\texttt{h2h}$, $\texttt{t2t}$, $\texttt{t2h}$). Note that we do not include detailed segment orientation (i.e., $\texttt{horizontal}$, $\texttt{vertical}$, $\texttt{diagonal}$) in the feature space to accommodate for alternative vectorial representations of the same letter. For instance, the first segment in an "A", which due to the left-to-right rule necessarily fits the lower left portion of the character, is bound to be $\texttt{increasing}$, but may be equally likely $\texttt{vertical}$ or $\texttt{diagonal}$ (see e.g. Figures 5 (b) and (c)).

Summing up, the *orientation* term can be written as:

$$
\begin{aligned}
(\quad & increasing(1), decreasing(1), right(1), \\
& \cdots \\
& increasing(m), decreasing(m), right(m), \\
& h2t(1,2), t2h(1,2), h2h(1,2), t2t(1,2), \\
& \cdots \\
& h2t(1,m), t2h(1,m), h2h(1,m), t2t(1,m), \\
& \cdots \\
& h2t(m-1,m), t2h(m-1,m), h2h(m-1,m), t2t(m-1,m) \quad)
\end{aligned}
$$

where each feature is the indicator function of the corresponding Boolean variable, e.g. $increasing(1) := \mathbb{1}(\texttt{increasing}(1))$ (see Table 5).

30

Segment types	
Segment i is horizontal	$\texttt{horizontal}(i) := (x_i^b \neq x_i^e) \wedge (y = y_i^b)$
Segment i is vertical	$\texttt{vertical}(i) := (x_i^b = x_i^e) \wedge (y_i^e \neq y_i^b)$
Segment i is diagonal	$\texttt{diagonal}(i) := |x_i^e - x_i^b| = |y_i^e - y_i^b|$
Segment i is increasing	$\texttt{increasing}(i) := y_i^e > y_i^b$
Segment i is decreasing	$\texttt{decreasing}(i) := y_i^e < y_i^b$
Segment i is left-to-right	$\texttt{right}(i) := x_i^e > x_i^b$
Segment i is incr. vert.	$\texttt{incr\_vert}(i) := \texttt{increasing}(i) \wedge \texttt{vertical}(i)$
Segment i is decr. vert.	$\texttt{decr\_vert}(i) := \texttt{decreasing}(i) \wedge \texttt{vertical}(i)$
Segment i is incr. diag.	$\texttt{incr\_diag}(i) := \texttt{increasing}(i) \wedge \texttt{diagonal}(i)$
Segment i is decr. diag.	$\texttt{decr\_diag}(i) := \texttt{decreasing}(i) \wedge \texttt{diagonal}(i)$
Segment length	
Lenght of horiz. segment i	$\texttt{horizontal}(i) \rightarrow length(i) = |x_i^e - x_i^b|$
Length of vert. segment i	$\texttt{vertical}(i) \rightarrow length(i) = |y_i^e - y_i^b|$
Lenght of diag. segment i	$\texttt{diagonal}(i) \rightarrow length(i) = \sqrt{2}\,|y_i^e - y_i^b|$
Connections between segments	
Segments i,j are head-to-tail	$\texttt{h2t}(i,j) := (x_i^e = x_j^b) \wedge (y_i^e = y_j^b)$
Segments i,j are head-to-head	$\texttt{h2h}(i,j) := (x_i^e = x_j^e) \wedge (y_i^e = y_j^e)$
Segments i,j are tail-to-tail	$\texttt{t2t}(i,j) := (x_i^b = x_j^b) \wedge (y_i^b = y_j^b)$
Segments i,j are tail-to-head	$\texttt{t2h}(i,j) := (x_i^b = x_j^e) \wedge (y_i^b = y_j^e)$
Segments i,j are connected	$\texttt{connected}(i,j) := \texttt{h2h}(i,j) \vee \texttt{h2t}(i,j) \vee$ $\texttt{t2h}(i,j) \vee \texttt{t2t}(i,j)$
Whether segment $i = (x^b, y^b, x^e, y^e)$ covers pixel $p = (x,y)$	
Coverage of pixel p	$\texttt{covered}(p) := \bigvee_i \texttt{covered}(p,i)$
Coverage of pixel p by seg. i	

$\texttt{incr\_vert}(i) \rightarrow \texttt{covered}(p,i) := y_i^b \leq y \leq y_i^e \wedge x = x_i^b$

$\texttt{decr\_vert}(i) \rightarrow \texttt{covered}(p,i) := y_i^e \leq y \leq y_i^b \wedge x = x_i^b$

$\texttt{horizontal}(i) \rightarrow \texttt{covered}(p,i) := x_i^b \leq x \leq x_i^e \wedge y = y_i^b$

$\texttt{incr\_diag}(i) \rightarrow \texttt{covered}(p,i) := y_i^b \leq y \leq y_i^e \wedge x_i^b \leq x \leq x_i^e \wedge x_i^b - y_i^b = x - y$

$\texttt{decr\_diag}(i) \rightarrow \texttt{covered}(p,i) := y_i^e \leq y \leq y_i^b \wedge x_i^b \leq x \leq x_i^e \wedge x_i^b + y_i^b = x + y$

Table 4: Background knowledge used in the character writing experiment.

(a) Hard constraints	
Left-to-right ordering	$x_i^b \leq x_i^e$
Allowed segment types	$\texttt{vertical}(i) \vee \texttt{horizontal}(i) \vee \texttt{diagonal}(i)$
Consecutive segments are connected	$\texttt{connected}(i, i+1)$
Minimum segment size	$min\_length \leq length(i) \leq 1$
(b) Soft constraints (features)	
Non-zero pixel coverage	$coverage := \frac{1}{|P|} \sum_{p \in P} \mathbb{1}(\texttt{covered}(p))$
Indicator of increasing segment i	$increasing(i) := \mathbb{1}(\texttt{increasing}(i))$
Indicator of decreasing segment i	$decreasing(i) := \mathbb{1}(\texttt{decreasing}(i))$
Indicator of right segment i	$right(i) := \mathbb{1}(\texttt{right}(i))$
Indicator of head-to-tail i, j	$h2t(i,j) := \mathbb{1}(\texttt{h2t}(i,j))$
Indicator of tail-to-head i, j	$t2h(i,j) := \mathbb{1}(\texttt{t2h}(i,j))$
Indicator of head-to-head i, j	$h2h(i,j) := \mathbb{1}(\texttt{h2h}(i,j))$
Indicator of tail-to-tail i, j	$t2t(i,j) := \mathbb{1}(\texttt{t2t}(i,j))$

(c) Cost

$$cost := \boldsymbol{w}^\top (\underbrace{increasing(i), decreasing(i), right(i),}_{\text{for all segments } i}$$

$$\underbrace{h2t(i,i+1), t2h(i,i+1), h2h(i,i+1), t2t(i,i+1),}_{\text{for all segments } i}$$

$$coverage)$$

Table 5: List of all rules used in the character writing problem. Top, hard rules. Middle, soft rules (costs). Bottom, total cost of a segment assignment.

We evaluated LMT on the character drawing problem by carrying out an extensive experiment using a set of noisy B&W 16×20 character images[13]. The dataset includes 39 instances of handwritten images of each alphanumerical character. We downscaled the images to 12×12 for speeding up the experiments. Learning to draw characters is a very challenging constructive problem, made even more difficult by the low quality of the noisy images in the dataset (see, e.g. Figure 7.) In this experiment we learn a model for each of the first five letters of the alphabet (A to E), and assess the ability of LMT to generalize over unseen handwritten images of the same character.

We selected for each letter five images at random out of the 39 available to be employed as training instances. For each of these, we used OP-TIMATHSAT to generate a "perfect" vectorial representation according to a human-provided letter template (similar to the "looking like an A" rule

---

[13]Dataset taken from http://cs.nyu.edu/~roweis/data.html

above), obtaining a training set of five fully supervised images. Please note that the training outputs generated by this process may not be optimal from a "perceptual" perspective, but only with respect to the "looking like an A" rule. The resulting supervision obtained with this procedure—which can be seen in the first rows of Figures 6 to 10—is, in some cases, very noisy, and depends crucially on the quality of the character image. This is particularly relevant for the "B", which is the most geometrically complex of the characters and thus more difficult to capture with bitmap images.

For each letter, we randomly sampled a test set of 10 instances out of the 33 non-training images. Then we learned a model for each letter, and used it to infer the vectorial representation of the test images. In order to assess the robustness of the learning method with respect to the amount of available supervision, we repeated this procedure four times, each time adding a training instance to the training set: the number of instances in the training set grows from 2 (the first two training images) to 5 (all of the training images). We indicate the predictions obtained by models learned with k examples as *pred@k*. The number of segments m was known during both training and inference. In particular, we used 4 segments for the "D", 5 segments for the "C" and "E", 7 segments for the "A", and 9 segments for the "B". The output for all letters can be found in Figures 6 to 10, from the second to the fifth rows of each figure.

We sped up the computations in two ways. First, during learning we imposed a 2 minute timeout on the separation oracle used for training. Consequently most invocations to the separation routine did return an approximate solution. Analyzing the weights, we found that this change had little effect on the learned model (data not shown). This can be explained by observing that the cutting-plane algorithm does not actually *require* the separation oracle to be perfect: as long as the approximation is consistent with the hard rules (which is necessarily the case even in sub-optimal solutions), the constraint added to the working set $\mathcal{W}$ at each iteration still restricts the quadratic program in a sound manner (see Algorithm 1). As a consequence, the learning procedure is still guaranteed to find an ϵ-approximate solution, but it may take more iterations to converge. This trick allows training to terminate very quickly (in the order of minutes for a set of five training examples). Furthermore, it enables the user to fine tune the trade-off between separation complexity and number of QP sub-problems to be solved.

Second, prior to inference we convert the learned weights associated to the segment and connection features into *hard* constraints and add them

33

to the learned model. This way we constrain OptiMathSAT to search for solutions that do respect the learned weights, while still allowing for some flexibility in the choice of the actual solution. In practice, for each per-segment feature (i.e. those associated to `increasing`, `decreasing` and `right` soft constraints) and connection features (i.e. `h2t`, `t2t`, *etc.*) with a positive weight, we add the corresponding hard rule. If more than one weight is positive for any given segment/connection, we add the disjuction of the hard rules to the model.

As a quantitative measure of the quality of the predictions, we also report the distance between the generated vectorial representation O for each letter and a corresponding human-made gold standard O'. Here the error is computed by first aligning the segments using an optimal translation, and then summing the distances between all corresponding segment endpoints. The human generated images respect the same "looking like an X" rule used for generating the training set, i.e. they have the same number of segments, drawn in the same order and within the same set of allowed orientations. One minor difference is they do not follow the requirement that segment endpoints match, for simplicity; this has little impact on the resulting values. The values in Figure 11 are the average over all instances in the test set, when varying the training set size.
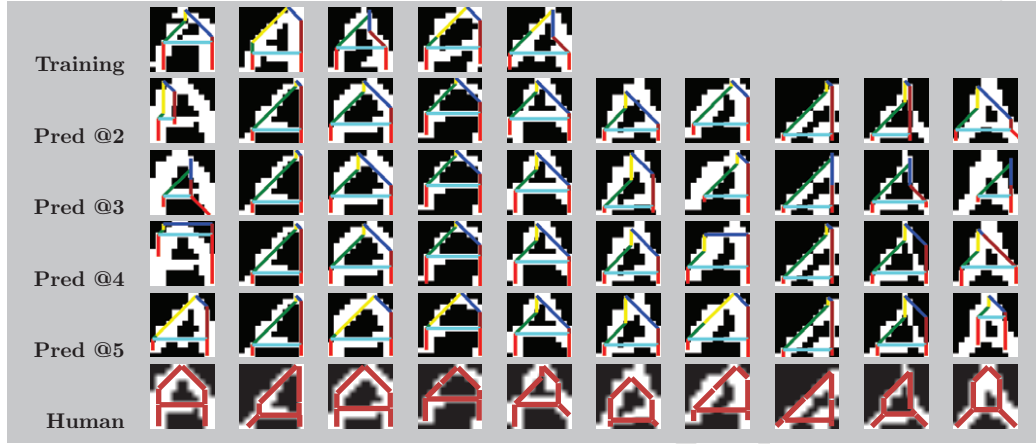
Figure 6: Results for the "A" 12×12 character drawing task. The training instances are lined up in the first row. The second to fifth row are the segmentations generated by models learned with the first two training instances, the first three instances, *etc.*, respectively. The last row are the human-made segmentations used in the comparison. The generated vectorial representations are shown overlayed over the corresponding bitmap image. Segments are colored for readability.
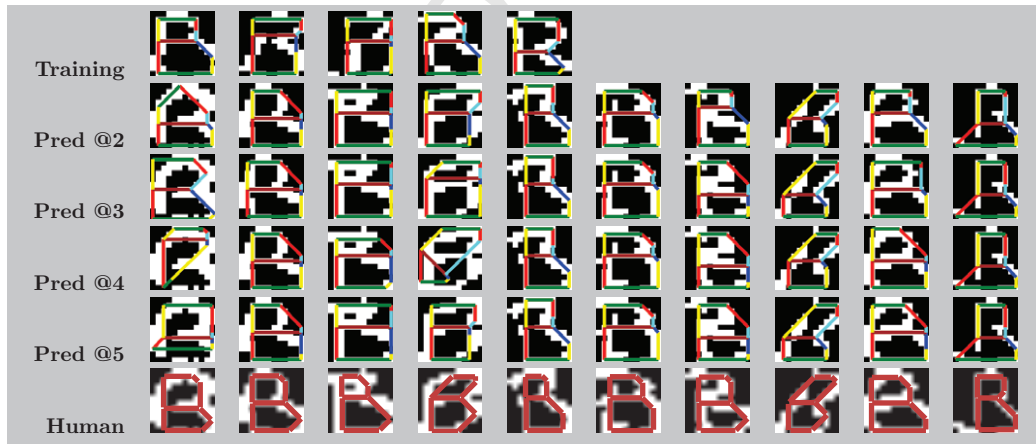

Figure 7: Results for the "B" 12×12 character drawing task.
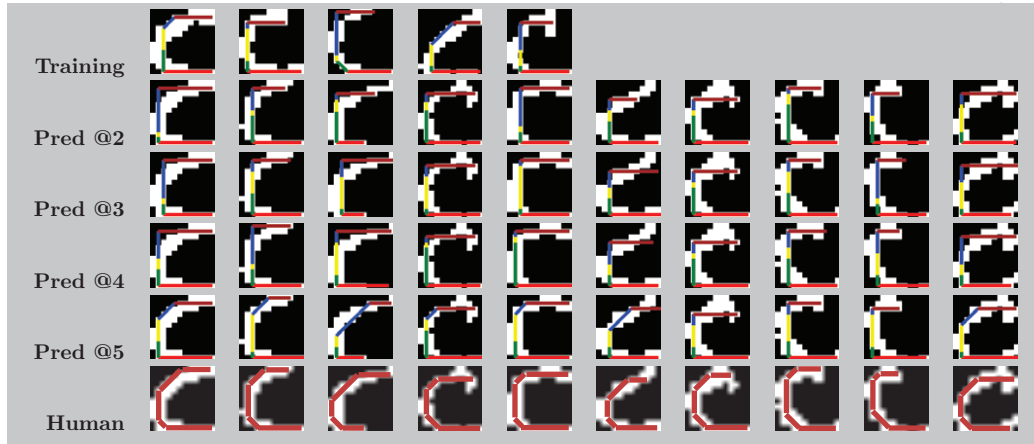
Figure 8: Results for the "C" 12×12 character drawing task.
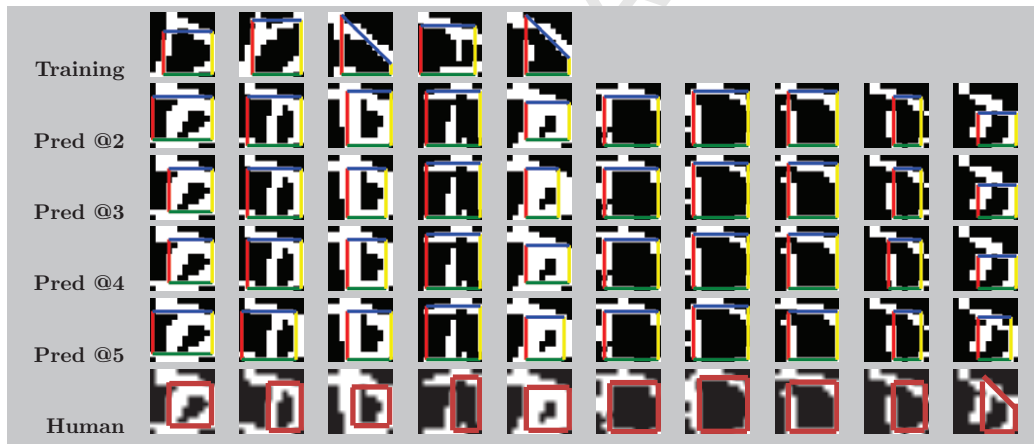
Figure 9: Results for the "D" 12×12 character drawing task.
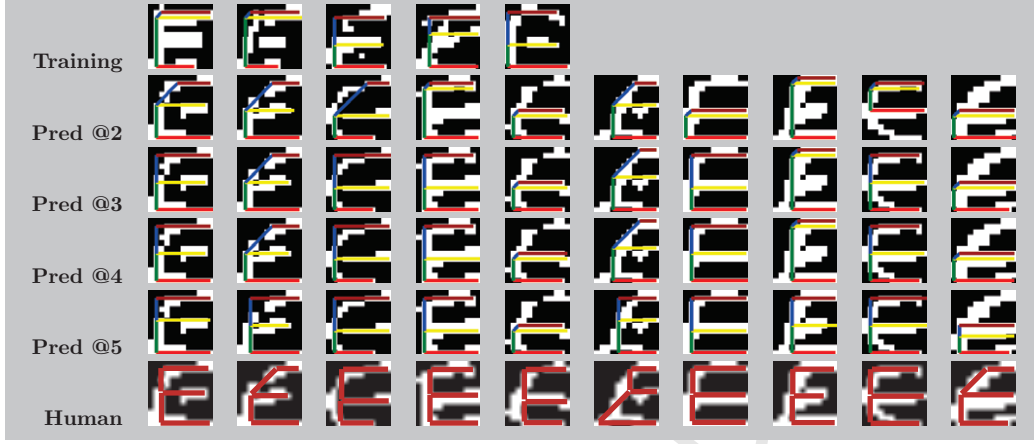
Figure 10:   Results for the "E" 12×12 character drawing task.
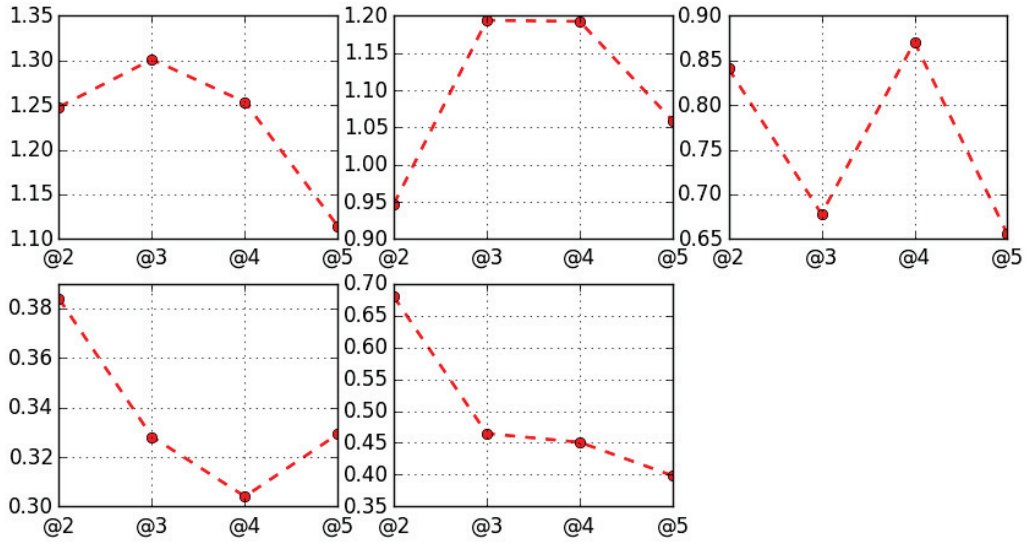

Figure 11:   Average distance between the predicted vectorial images and the human-made ones, while increasing the number of training instances. The **@k** ticks on the x-axis indicate the size of the training set. The y-axis represents the sum of per-segment distances averaged over all images in the test set. From left to right, top to bottom: results for "A", "B", "C", "D", and "E".

The results show that LMT is able to address the character drawing problem and produce reasonable outputs for all the target letters. Please

37

note that both the coordinates and the number of character pixels can vary widely between test instances, and our results highlight the generalization ability of our method. Furthermore, the predictions tend to get closer to the human-provided segmentations as the number of training instances increases.

For the simplest cases (i.e. "C", "D", and "E", drawn using four to five segments), the outcome is unambiguous: there are only a handful of cases for "E" where two horizontal segments are too close, and this is only due to the fact that we do not encode (soft) constraints on the segment lengths. The only real issue is with the "D", where the fourth (blue) segment is never predicted as diagonal, despite there being two examples with that property in the training set. The main reason is that in none of the test images it is possible to draw a 45° diagonal without sacrificing pixel coverage. None of the predictions looks perceptually "wrong".

More complex letters like the "A" and "B", with a higher number of segments, also show a similar behavior: the generated letters indeed generalize the given examples. However, there are a handful of predictions that are suboptimal with respect to coverage of the bitmap character (e.g. see the first column in Figure 6) or do not represent a "perceptually" correct character, a behaviour which is less frequent for the larger training sets (e.g. second-to-last row of Figure 6). This can be explained by (i) the complexity of the 7- and (especially) the 9-segment inference problems, (ii) the fact that more segments imply more features, and consequently may require more examples to be learned correctly, and (iii) the fact that our cost function does not fully discriminate between perceptually different solutions.

The distance-to-human results in Figure 11 also show how the algorithm produces more perceptually reasonable predictions as the training set increases: while the values do fluctuate, in all cases the distance at *pred@5* is lower than that at *pred@2*. Summarizing, excluding cases of pathologically bad inputs—such as the third "B" training examples leading to the bad performance in the *pred@3* and *pred@4* experiments—LMT is able to learn an appropriate model for each letter and generalize the learned template over unseen inputs.

In this paper we only consider the case where the training data is labeled with a fully observed vectorial letter. Our method, however, can in principle be extended to work with partially observed supervision, e.g. with training instances labeled only with the character type, in order to discover the vectorial representation. More on this point can be found in the next section.

6. Conclusions

In this work we presented a novel class of methods for structured learning problems with mixed Boolean and numerical variables. These methods, termed Learning Modulo Theories, are based on a combination of structured-output SVMs and Satisfiability Modulo Theories. In contrast to classical First-Order Logic, SMT allows to natively describe, and reason over, numerical variables and mixed logical/arithmetical constraints. By leveraging the existing state-of-the-art OMT solvers, LMT is well-suited for dealing with hybrid constructive learning problems, avoiding the combinatorial explosion of the state space that would affect an equivalent FOL formulation.

Experimental results on both artificial and real world datasets show the potential of this approach. The stairway application is a simple instance of a layout problem, where the task is to find an optimal layout subject to a set of constraints. Automated or interactive layout synthesis has a broad range of potential applications, including urban pattern layout [52], decorative mosaics [53] and furniture arrangement [54, 55, 56]. Note that many spatial constraints can be encoded in terms of relationships between blocks [54]. Existing approaches typically design an energy function to be minimized by stochastic search. Our approach suggests how to automatically identify the relevant constraints and their respective weights, and can accomodate hard constraints and exact search. This is especially relevant for *water-tight* layouts [57], where the whole space needs to be filled (i.e. no gaps or overlaps) by deforming basic elements from a predetermined set of templates (as in residential building layout [58]). The character drawing application shows how to learn symbolic representations from a handful of very noisy training instances. Deep generative networks have been previously used for similar tasks, see for instance the work by Hinton [59] on generating images of digits with Deep Boltzmann Machines. However, these methods do not learn symbolic representations for characters and generate bitmaps rather than vectorial representations. Furthermore, they require thousands of training examples to be learned. Recent extensions have been developed addressing the problem of learning from few [60] or even single [61] examples, but they focus on clean images of the target symbols. Generally speaking, the LMT framework allows to introduce a learning stage in all application domains where SMT and OMT approaches have shown their potential, ranging, e.g., from hardware and software verification [62, 63, 39], to engineering of chemical reactions [64] and synthetic biology [65].

39

This work can be extended in a number of directions. First, the current formulation assumes knowledge of the desired output for training examples. This requirement can be loosened by introducting latent variables for the unobserved part of the output, to be maximized over during training [66]. Second, OMT is currently limited to quantifier free formulae and linear algebra for what concerns numeric theories. The former requires to ground all predicates before performing inference, while the latter prevents the formulation of non-linear constraints, e.g. on areas and Euclidean distances. Some attempts to extend SMT solvers to quantified formulae [48, 49, 50] and to non-linear arithmetic [67, 68, 69] have been presented in the literature; although the state of the art of these extensions is not yet satisfactory in terms of efficiency, since they can currently handle problems which are much smaller –in terms of size, number of variables and of arithmetical operations– than the quantifier-free problems with linear constraints at the reach of current SMT solvers, these techniques are evolving rapidly and promise rapid improvements in terms of performances in the next few years; hence, we can rather easily extend our framework in these directions as soon as the underlying SMT technology is mature enough. Finally, LMT is currently focused on the task of finding the maximal configuration, and cannot compute marginal probabilities. We are planning to introduce support for probability computation by leveraging ideas from weighted model counting [70].

Acknowledgements

References

[1] L. Getoor, B. Taskar, Introduction to statistical relational learning, The MIT press, 2007.

[2] G. H. Bakir, T. Hofmann, B. Schölkopf, A. J. Smola, B. Taskar, S. V. N. Vishwanathan, Predicting Structured Data (Neural Information Processing), The MIT Press, 2007.

[3] M. Lippi, P. Frasconi, Prediction of protein -residue contacts by markov logic networks with grounding-specific weights, Bioinformatics 25 (2009) 2326–2333.

[4] M. Broecheler, L. Mihalkova, L. Getoor, Probabilistic similarity logic, in: Uncertainty in Artificial Intelligence (UAI), 2010, pp. 73–82.

[5] O. Kuželka, A. Szabóvá, M. Holec, F. Železný, Gaussian logic for predictive classification, in: D. Gunopulos, T. Hofmann, D. Malerba, M. Vazirgiannis (Eds.), Machine Learning and Knowledge Discovery in Databases, volume 6912 of *Lecture Notes in Computer Science*, Springer Berlin Heidelberg, 2011, pp. 277–292.

[6] M. Diligenti, M. Gori, M. Maggini, L. Rigutini, Bridging logic and kernel machines, Machine Learning 86 (2012) 57–88.

[7] N. D. Goodman, V. K. Mansinghka, D. M. Roy, K. Bonawitz, J. B. Tenenbaum, Church: a language for generative models, in: D. A. McAllester, P. Myllymäki (Eds.), UAI, AUAI Press, 2008, pp. 220–229.

[8] J. Wang, P. Domingos, Hybrid markov logic networks, in: Proceedings of the 23rd National Conference on Artificial Intelligence - Volume 2, AAAI'08, AAAI Press, 2008, pp. 1106–1111.

[9] P. Nrman, M. Buschle, J. Knig, P. Johnson, Hybrid probabilistic relational models for system quality analysis., in: EDOC, IEEE Computer Society, 2010, pp. 57–66.

[10] B. Gutmann, M. Jaeger, L. De Raedt, Extending problog with continuous distributions, in: P. Frasconi, F. Lisi (Eds.), Inductive Logic Programming, volume 6489 of *Lecture Notes in Computer Science*, Springer Berlin Heidelberg, 2011, pp. 76–91.

[11] J. Choi, E. Amir, Lifted relational variational inference, in: N. de Freitas, K. P. Murphy (Eds.), UAI'12: Proceedings of the Twenty-Eigth Conference on Uncertainty in Artificial Intelligence, AUAI Press, 2012, pp. 196–206.

[12] M. a. Islam, C. r. Ramakrishnan, I. v. Ramakrishnan, Inference in probabilistic logic programs with continuous random variables, Theory Pract. Log. Program. 12 (2012) 505–523.

[13] C. Barrett, R. Sebastiani, S. A. Seshia, C. Tinelli, Satisfiability Modulo Theories, in: [38], pp. 825–885.

[14] R. Nieuwenhuis, A. Oliveras, On SAT Modulo Theories and Optimization Problems, in: Proc. Theory and Applications of Satisfiability Testing - SAT 2006, volume 4121 of *Lecture Notes in Computer Science*, Springer, 2006.

[15] A. Cimatti, A. Franzén, A. Griggio, R. Sebastiani, C. Stenico, Satisfiability modulo the theory of costs: Foundations and applications, in: Proc. Tools and Algorithms for the Construction and Analysis of Systems, TACAS, volume 6015 of *Lecture Notes in Computer Science*, Springer, 2010, pp. 99–113.

[16] A. Cimatti, A. Griggio, B. J. Schaafsma, R. Sebastiani, A Modular Approach to MaxSAT Modulo Theories, in: International Conference on Theory and Applications of Satisfiability Testing, SAT, volume 7962 of *Lecture Notes in Computer Science*, Springer, 2013.

[17] C. M. Li, F. Manyà, MaxSAT, Hard and Soft Constraints, in: [38], pp. 613–631.

[18] R. Sebastiani, S. Tomasi, Optimization in SMT with LA(Q) Cost Functions, in: proc. International Joint Conference on Automated Reasoning, IJCAR, volume 7364 of *LNAI*, Springer, 2012, pp. 484–498.

[19] R. Sebastiani, S. Tomasi, Optimization Modulo Theories with Linear Rational Costs, ACM Transactions on Computational Logics (2015). In print.

[20] Y. Li, A. Albarghouthi, Z. Kincad, A. Gurfinkel, M. Chechik, Symbolic Optimization with SMT Solvers, in: Proceedings of the 41st ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages, POPL '14, ACM, New York, NY, USA, 2014, pp. 607–618.

[21] R. Sebastiani, P. Trentin, Pushing the Envelope of Optimization Modulo Theories with Linear-Arithmetic Cost Functions, in: Proc. Int.

Conference on Tools and Algorithms for the Construction and Analysis of Systems, TACAS'15, volume 9035 of *Lecture Notes in Computer Science*, Springer, 2015.

[22] N. Bjorner, A.-D. Phan, νZ - Maximal Satisfaction with Z3, in: Proc International Symposium on Symbolic Computation in Software Science, EasyChair Proceedings in Computing (EPiC), Gammart, Tunisia, 2014. `http://www.easychair.org/publications/?page=862275542`.

[23] I. Tsochantaridis, T. Joachims, T. Hofmann, Y. Altun, Large margin methods for structured and interdependent output variables, J. Mach. Learn. Res. 6 (2005) 1453–1484.

[24] T. Joachims, T. Hofmann, Y. Yue, C.-N. Yu, Predicting structured objects with support vector machines, Communications of the ACM 52 (2009) 97–104.

[25] P. Campigotto, A. Passerini, R. Battiti, Active Learning of Combinatorial Features for Interactive Optimization, in: Proceedings of the 5th Learning and Intelligent OptimizatioN Conference (LION V), Rome, Italy, Jan 17-21, 2011, Lecture Notes in Computer Science, Springer Verlag, 2011.

[26] J. Choi, D. Hill, E. Amir, Lifted inference for relational continuous models, in: UAI'10: Proceedings of the Twenty-Sixth Conference on Uncertainty in Artificial Intelligence.

[27] J. Choi, A. Guzmn-Rivera, E. Amir, Lifted relational kalman filtering., in: Proceedings of IJCAI'11, pp. 2092–2099.

[28] B. Ahmadi, K. Kersting, S. Sanner, Multi-evidence lifted message passing, with application to pagerank and the kalman filter, in: Proceedings of IJCAI'11, pp. 1152–1158.

[29] K. P. Murphy, Inference and Learning in Hybrid Bayesian Networks, Technical Report, University of California, Berkeley, Computer Science Division, 1998.

[30] S. L. Lauritzen, Propagation of probabilities, means, and variances in mixed graphical association models, Journal of the American Statistical Association 87 (1992) 1098–1108.

[31] T. Sato, A statistical learning method for logic programs with distribution semantics., in: L. Sterling (Ed.), ICLP, MIT Press, 1995, pp. 715–729.

[32] M. A. Islam, Inference and Learning in Probabilistic Logic Programs with Continuous Random Variables, Ph.D. thesis, Stony Brook University, 2012.

[33] A. Griggio, Q. S. Phan, R. Sebastiani, S. Tomasi, Stochastic Local Search for SMT: Combining Theory Solvers with WalkSAT, in: Frontiers of Combining Systems, FroCoS'11, volume 6989 of *LNAI*, Springer, 2011.

[34] L. De Raedt, A. Kimmig, H. Toivonen, Problog: A probabilistic prolog and its application in link discovery., in: M. M. Veloso (Ed.), IJCAI, 2007, pp. 2462–2467.

[35] B. Gutmann, I. Thon, A. Kimmig, M. Bruynooghe, L. De Raedt, The magic of logical inference in probabilistic programming., TPLP 11 (2011) 663–680.

[36] S. Prestwich, CNF Encodings, in: [38], pp. 75–97.

[37] J. P. Marques-Silva, I. Lynce, S. Malik, Conflict-Driven Clause Learning SAT Solvers, in: [38], pp. 131–153.

[38] A. Biere, M. J. H. Heule, H. van Maaren, T. Walsh (Eds.), Handbook of Satisfiability, Frontiers in Artificial Intelligence and Applications, IOS Press, 2009.

[39] L. de Moura, N. Bjørner, Satisfiability modulo theories: introduction and applications, Commun. ACM 54 (2011) 69–77.

[40] R. Sebastiani, Lazy Satisfiability Modulo Theories, Journal on Satisfiability, Boolean Modeling and Computation, JSAT 3 (2007) 141–224.

[41] E. Balas, Disjunctive programming, in: M. Junger, T. M. Liebling, D. Naddef, G. L. Nemhauser, W. R. Pulleyblank, G. Reinelt, G. Rinaldi, L. A. Wolsey (Eds.), 50 Years of Integer Programming 1958-2008, Springer Berlin Heidelberg, 2010, pp. 283–340.

[42] A. Lodi, Mixed Integer Programming Computation, in: 50 Years of Integer Programming 1958-2008, Springer-Verlag, 2009, pp. 619–645.

[43] N. W. Sawaya, I. E. Grossmann, A hierarchy of relaxations for linear generalized disjunctive programming, European Journal of Operational Research 216 (2012) 70–82.

[44] J. Jaffar, M. J. Maher, Constraint Logic Programming: A Survey, Journal of Logic Programming 19/20 (1994) 503–581.

[45] P. Codognet, D. Diaz, Compiling constraints in clp(fd), The Journal of Logic Programming 27 (1996) 185 – 226.

[46] J. Jaffar, S. Michaylov, P. J. Stuckey, R. H. C. Yap, The clp(r) language and system, ACM Trans. Program. Lang. Syst. 14 (1992) 339–395.

[47] A. Cimatti, A. Griggio, B. J. Schaafsma, R. Sebastiani, The MathSAT 5 SMT Solver, in: Tools and Algorithms for the Construction and Analysis of Systems, TACAS'13., volume 7795 of *Lecture Notes in Computer Science*, Springer, 2013, pp. 95–109.

[48] P. Rümmer, A constraint sequent calculus for first-order logic with linear integer arithmetic, in: I. Cervesato, H. Veith, A. Voronkov (Eds.), Logic for Programming, Artificial Intelligence, and Reasoning, volume 5330 of *Lecture Notes in Computer Science*, Springer Berlin Heidelberg, 2008, pp. 274–289.

[49] P. Baumgartner, C. Tinelli, Model evolution with equality modulo built-in theories, in: N. Bjørner, V. Sofronie-Stokkermans (Eds.), Automated Deduction CADE-23, volume 6803 of *Lecture Notes in Computer Science*, Springer Berlin Heidelberg, 2011, pp. 85–100.

[50] E. Kruglov, Superposition modulo theory, Ph.D. thesis, Universität des Saarlandes, Postfach 151141, 66041 Saarbrücken, 2013.

[51] T. Joachims, T. Finley, C.-N. J. Yu, Cutting-plane training of structural svms, Machine Learning 77 (2009) 27–59.

[52] Y.-L. Yang, J. Wang, E. Vouga, P. Wonka, Urban pattern: Layout design by hierarchical domain splitting, ACM Trans. Graph. 32 (2013) 181:1–181:12.

45

[53] A. Hausner, Simulating decorative mosaics, in: Proceedings of the 28th Annual Conference on Computer Graphics and Interactive Techniques, SIGGRAPH '01, ACM, New York, NY, USA, 2001, pp. 573–580.

[54] L.-F. Yu, S.-K. Yeung, C.-K. Tang, D. Terzopoulos, T. F. Chan, S. J. Osher, Make it home: Automatic optimization of furniture arrangement, ACM Trans. Graph. 30 (2011) 86:1–86:12.

[55] P. Merrell, E. Schkufza, Z. Li, M. Agrawala, V. Koltun, Interactive furniture layout using interior design guidelines, ACM Trans. Graph. 30 (2011) 87:1–87:10.

[56] Y.-T. Yeh, L. Yang, M. Watson, N. D. Goodman, P. Hanrahan, Synthesizing open worlds with constraints using locally annealed reversible jump mcmc, ACM Trans. Graph. 31 (2012) 56:1–56:11.

[57] C.-H. Peng, Y.-L. Yang, P. Wonka, Computing layouts with deformable templates, ACM Trans. Graph. 33 (2014) 99:1–99:11.

[58] P. Merrell, E. Schkufza, V. Koltun, Computer-generated residential building layouts, ACM Trans. Graph. 29 (2010) 181:1–181:12.

[59] G. E. Hinton, To recognize shapes first learn to generate images, in: Computational Neuroscience: Theoretical Insights into Brain Function, Elsevier, 2007.

[60] R. Salakhutdinov, J. B. Tenenbaum, A. Torralba, Learning with hierarchical-deep models., IEEE Trans. Pattern Anal. Mach. Intell. 35 (2013) 1958–1971.

[61] B. M. Lake, R. Salakhutdinov, J. B. Tenenbaum, One-shot learning by inverting a compositional causal process, in: NIPS'13, pp. 2526–2534.

[62] D. Beyer, A. Cimatti, A. Griggio, M. E. Keremoglu, R. Sebastiani, Software model checking via large-block encoding, in: FMCAD, IEEE, 2009, pp. 25–32.

[63] A. Franzen, A. Cimatti, A. Nadel, R. Sebastiani, J. Shalev, Applying SMT in Symbolic Execution of Microcode, in: Proc. Int. Conference on Formal Methods in Computer Aided Design (FMCAD'10), IEEE, 2010.

[64] R. Fagerberg, C. Flamm, D. Merkle, P. Peters, Exploring chemistry using smt, in: Proceedings of the 18th International Conference on Principles and Practice of Constraint Programming, CP'12, Springer-Verlag, Berlin, Heidelberg, 2012, pp. 900–915.

[65] B. Yordanov, C. M. Wintersteiger, Y. Hamadi, H. Kugler, Smt-based analysis of biological computation, in: NASA Formal Methods Symposium 2013, volume 7871 of *Lecture Notes in Computer Science*, Springer Verlag, 2013, pp. 78–92.

[66] C.-N. J. Yu, T. Joachims, Learning structural svms with latent variables, in: Proceedings of the 26th Annual International Conference on Machine Learning, ICML '09, ACM, New York, NY, USA, 2009, pp. 1169–1176.

[67] M. Fränzle, C. Herde, T. Teige, S. Ratschan, T. Schubert, Efficient solving of large non-linear arithmetic constraint systems with complex boolean structure, JSAT 1 (2007) 209–236.

[68] D. Jovanovic, L. M. de Moura, Solving non-linear arithmetic, in: B. Gramlich, D. Miller, U. Sattler (Eds.), IJCAR, volume 7364 of *Lecture Notes in Computer Science*, Springer, 2012, pp. 339–354.

[69] D. Larraz, A. Oliveras, E. Rodriguez-Carbonell, A. Rubio, Minimal-Model-Guided Approaches to Solving Polynomial Constraints and Extensions, in: C. Sinz, U. Egly (Eds.), Theory and Applications of Satisfiability Testing - SAT'14 - 17th International Conference, volume 8561 of *Lecture Notes in Computer Science*, Springer, 2014, pp. 333–350.

[70] M. Chavira, A. Darwiche, On probabilistic inference by weighted model counting, Artif. Intell. 172 (2008) 772–799.

47