Learning Constraints from Examples

Luc De Raedt
Dept. of Computer Science
KU Leuven, box 2402
BE-3001 Heverlee, Belgium
luc.deraedt@cs.kuleuven.be

Andrea Passerini
University of Trento
Via Sommarive 9
Povo Trento, Italy
andrea.passerini@unitn.it

Stefano Teso
Dept. of Computer Science
KU Leuven, box 2402
BE-3001 Heverlee, Belgium
stefano.teso@cs.kuleuven.be

Abstract

While constraints are ubiquitous in artificial intelligence and constraints are also commonly used in machine learning and data mining, the problem of learning constraints from examples has received less attention. In this paper, we discuss the problem of constraint learning in detail, indicate some subtle differences with standard machine learning problems, sketch some applications and summarize the state-of-the-art.

Introduction

Constraints are ubiquitous in artificial intelligence and operations research, they appear in purely logical problems such as propositional satisfiability, constraint satisfaction problems, and full-fledged constraint optimization.

The term *constraint learning*, as used in this paper, refers to the problem of finding a set of constraints, a constraint theory, that satisfies a given dataset. This use of the term constraint learning differs from that used in solver technology, where clause learning or constraint learning refers to the process of adding clauses to improve the efficiency of the solver when inconsistencies are found. This is actually a deductive procedure. In contrast, we refer to inductive learning, that is, to learning a constraint theory from examples.

In recent years there has been a significant interest in the relationship between machine learning and data mining on the one hand, and constraint satisfaction and programming on the other hand (Passerini, Tack, and Guns 2017; Bessiere et al. 2016b; 2017a). The use of constraints in data mining and machine learning is now a mainstream topic in constraint programming and artificial intelligence (cf. (Davidson, Guns, and Nijssen 2017)). While constraints are extremely popular from a modeling and solving perspective, also in machine learning and data mining, there are today only relatively few approaches to learning constraints from data. This paper wants to not only survey these developments but also to position constraint learning in a broader machine learning context. We will argue that there are some subtle differences with standard "function" learning in machine learning, and we will also point out some challenges and possible applications.

Copyright © 2018, Association for the Advancement of Artificial Intelligence (www.aaai.org). All rights reserved.

Constraint Learning

From one perspective, constraint learning can be formalized as a concept-learning problem (Mitchell 1997).

Given

- a space of possible instances X; typically instances are (partial or complete) assignments to a set of variables V belonging to the unknown constraint theory, i.e., logical interpretations;
- a space of possible constraints C;
- an unknown target constraint theory $T \subseteq C$;
- a set of training instances E; positive instances satisfy T, while negatives do not;

Find a constraint theory H ($H \subseteq C$) such that all positive instances in E are satisfied (or satisfiable in the case of partial assignments) in H; and none of the negatives.

As usual, several variations on this definition can be considered such as finding the best constraint theory w.r.t. a loss function as well as learning a soft constraint theory rather than a hard one, cf. below.

To illustrate this definition, let us consider the prototypical example of a constraint theory: a boolean theory. The space of possible constraints could then be the set of all clauses over the set of variables V and the instances could then be complete assignments to the variables. This formulation actually corresponds to the setting introduced by Leslie Valiant in his seminal paper on PAC-learning (probably approximately correct learning) (Valiant 1984).

From a machine learning perspective, the above formulation of learning constraint theories is in line with those of concept learning, where the aim is to learn a classifier that allows one to distinguish members of the concept from those that are not. However, while in concept-learning one induces a binary function, the learned theories in constraint learning typically allow also for other uses. A constraint theory cannot only be used to distinguish satisfiable from unsatisfiable instances but it can also be used to complete partial assignments. After all, constraint theories are typically used in this way. Given particular values for some of the variables $Ev \subseteq V$, one can infer which values may still be assigned to the other variables $V \backslash Ev$. When considering constraint optimization, one wants to infer what is the best such assignment, possibly accounting for weights associated with soft

constraints. This use of constraint theory shares some similarities with structured output prediction, albeit that there the variables Ev are typically fixed and the focus is on efficient techniques for computing the best possible output ($V \backslash Ev$) given the input (Ev). In this respect, constraint learning is closer to learning probabilistic models as also these models can be used to find the best (or most likely) possible world in the light of the evidence. Furthermore, probabilistic models can also be trained on complete or partial examples.

Applications

Constraint learning is useful in every context where constraints or constraint optimization is being used. This includes but is not limited to learning logical theories and formulas (as in computational learning theory (Kearns and Vazirani 1994) and inductive logic programming (Muggleton and De Raedt 1994)), in constraint programming and operations research, in a database and spreadsheet context.

To illustrate this, (Kolb et al. 2017) have recently shown that it is possible to learn the formulas and constraints that hold in a particular spreadsheet. Many users of Excel and other spreadsheets have difficulties in producing the formulas that compute the averages or totals of particular rows or columns, certainly if the operations to be used span multiple tables in the spreadsheet. Now (Kolb et al. 2017) have designed the Tacle system that given a spreadsheet showing all of the desired values, can reverse engineer the formula and constraints (such as functional dependencies). A formula computing the sum of three values acts as a constraint, because the formula also allows for the computation of one of the values given the sum and the two other values. Spreadsheets are often used as constraint solvers.

Another interesting example is ModelSeeker (Beldiceanu and Simonis 2012). It has successfully induced global models for problem such as sports scheduling and it has dealt with problems containing 1000s of variables. For instance, ModelSeeker has identified the constraints underlying the scheduling of the Bundesliga (the German Football Liga) from a single example schedule.

Configuration problems such as customizing a car or a laptop on an e-commerce website, can be viewed as prototypical constraint satisfaction and optimization problems. Here the hard constraints define the space of products and the soft (parameterized) ones represent the customer's preferences over feasible assignments. Methods such as SetMargin (Teso, Passerini, and Viappiani 2016) and Coactive Learning (Teso, Dragone, and Passerini 2017) tackle interactive learning of these parameters. More generally, many wide-spread timetabling (Smet et al. 2013) and design tasks (Mitchell, Steadman, and Liggett 1976; Harada, Witkin, and Baraff 1995) with an incomplete or ambiguous specification can in principle be adapted or completed through constraint learning.

The induction of constraints is also attracting attention from outside the field of artificial intelligence. (Smith, Ferns, and Albarghouthi 2017) describe the Bach system that induces relational specifications from data describing the input-output behavior of a set of functions in a program or library for use in a software engineering context, and (Pawlak and Krawiec 2017) induce constraints of mixed integer linear programs in an operations research context.

Finally, constraint learning can substantially speed up the design of complex architectures such as cyber-physical systems (Lee 2008). A substantial amount of human effort is typically required in order to design appropriate architectures, simultaneously satisfying known constraints and meeting engineering preferences developed through experience. A constraint learning component could help formalize these preferences by extracting them from existing architectures, in order to use them in the design of novel ones.

Dimensions of Constraint Learning

While successful formalisms for solving constraint satisfaction and optimization problems have been developed (Rossi, Van Beek, and Walsh 2006), and constraint learning has been identified as an important topic (O'Sullivan 2010), a general framework for constraint learning is still lacking. We believe it is useful to distinguish different settings for constraint learning; cf. (De Raedt et al. 2016).

Existing approaches to constraint learning often follow diverse recipes and aim at different goals. At one end of the spectrum there are methods that aim at learning of *hard* satisfaction problems, e.g. learning Boolean concepts and constraint satisfaction problems, based on symbolic and logical learning frameworks. At the other end there are methods that target *soft* constraint optimization problems, e.g. weighted maximum satisfiability, constrained optimization tasks with preferences over the constraints, and graphical models, often backed by non-symbolic machine learning algorithms. Representations can be based on *boolean* or on *first order logic* (as in inductive logic programming and statistical relational learning). Furthermore, existing methods span both offline, *batch* learning settings where the data is assumed as given, and online, *interactive* ones where the data is queried from an external oracle, often a human decision maker. A final dimension is concerned with the nature of the instances, are they *partial* or *complete* assignments to the variables?

Basic Algorithms

We first focus on the basic algorithm of (Valiant 1984) for learning a boolean k-CNF theory from examples; it is one of the simplest possible settings for learning a constraint theory and it corresponds to learning a "hard" boolean theory from complete instances in batch. In the next step, we will expand this basic approach along the different representations, that is, we shall discuss the necessary changes to apply it to first order logical theories, to constraint programs and to the learning of "soft" theories. Finally, we shall discuss user-interaction, preferences and decomposition.

Valiant's algorithm

Valiant's algorithm is learning a k-CNF theory, that is, a theory consisting of a set of clauses containing at most k literals. A clause is a disjunction of literals of the form v or $\neg v$ with the variables $v \in V$. Valiant's algorithm is very simple, it enumerates the set of all clauses H_0 containing at most k literals over the variables in V, and then repeatedly prunes

the current set of clauses H_{i+1} w.r.t. the next positive example p_i. Those clauses in H_i that do not satisfy the next positive example are simply deleted. This process continues until all positive examples have been processed. No negative examples are required or used. The algorithm assumes a noise-free setting.

Valiant's algorithm is essentially a generate-and-test algorithm – it generates all the constraints and then tests whether they hold on the dataset. It can also be examined from a learning as search perspective. From this perspective it always keeps track of the most specific hypothesis (i.e. H_i) covering the already processed examples, which is in line with Mitchell's Find-S algorithm (Mitchell 1997).

Valiant's algorithm has nice properties – it is a PAC-learning algorithm and it will also converge to the correct solution (assuming a noise-free setting) in the limit.

Inductive Logic Programming

Valiant's algorithm for k-CNF has been extended towards first order logic using principles of inductive logic programming (De Raedt and Dehaspe 1997; De Raedt and Džeroski 1994). This involves turning the k-CNF clauses into first order logic clauses by turning every literal of the form l (or $\neg l$) into a first order literal of the form $p(t_1, ..., t_n)$ where the t_i are logical terms (either constants or logical variables), and universally quantifying the resulting clauses. Consider for instance the clause $\forall X, Y : X \neq Y \vee \neg p(X, Y)$ which states that whenever $p(X, Y)$ holds, X and Y are different. In addition to changing the formulas, also the interpretations must be changed. One typically uses Herbrand interpretations then, these are sets of ground facts (facts that do not contain any variable). All facts in the interpretation are assumed to be true, all other ones are false. For instance, the interpretation $p(a, a)$ would not satisfy the above listed clause. The other elements of Valiant's algorithm remain the same, although optimizations are possible (and a constraint on the size of atoms is applied to obtain PAC-learning results). One such optimization is to prune the search according to the generality relation between clauses. A clause c_1 is more general than a clause c_2 if all examples covered by c_2 are also covered by c_1, that is, when $c_1 \models c_2$ (c_1 entails c_2). For instance, when a clause $a \vee b \vee c$ is not satisfied by a particular example, none of the proper subsets of the clause can satisfy the example as each such subset imposes stronger restrictions. Thus these subsets (specializations) can be pruned.

The resulting framework for clausal discovery has been used to learn constraints and regularities in relational data; similar principles (focusing on specific types of rules such as functional and multi-valued dependencies) have been used in a database setting, e.g. (Flach and Savnik 1999).

Alternative inductive logic programming techniques for learning constraint satisfaction problems have also been considered, cf. (Lallouet et al. 2010) who learn definite clauses from examples in the more traditional inductive logic programming setting of learning from entailment. This setting corresponds to a kind of rule learning, which can also be related to constraint learning. In rule learning, one typically learns a disjunction of conjuctions, so a formula in DNF form. In contrast, the constraint theories targeted in constraint learning are typically conjunctive, cf. Valiant's algorithm for learning k-CNF. These two forms of logical expression are each others duals. For instance, adding a literal to one of the conjunctive rules in a DNF yields a specialisation, while adding a literal to a clause or clausal theory yields a generalisation. One interesting property of these logical approaches to learning that carries over to constraint learning is that one obtains interpretable models, models that can readily be interpreted by the end-user. Furthermore, they enable the encoding and use of background knowledge in the learning process.

Constraint programming

The approaches to concept-learning that have been developed in a series of papers by Bessiere et al. (Bessiere et al. 2005; 2013; 2016a; 2017b) combine elements of the above two approaches. First, they are based on the learning as search paradigm of (Mitchell 1997), but rather than using the Find-S algorithm (that keeps track only of the most specific hypothesis), they use the more general versionspace algorithm (which keeps track of the sets of the most general and of the most specific hypotheses). Secondly, constraint satisfaction problems (CSPs) as studied in the constraint programming community are usually conjunctive. Indeed, in a CSP one is given a set of variables V, corresponding domains $D(v)$ for each of the variables $v \in V$ and a set of constraints C, where each constraint $c(v_1, ..., v_n)$ specifies a relation over $D(v_1) \times ... \times D(v_n)$. The problem is then to find an assignment θ to the variables V so that all constraints in C are satisfied. From a logical perspective, the constraint theory (or constraint network as it is often called) corresponds to a first order logical expression $l_1 \wedge ... \wedge l_m$, where the l_i are first order literals as in the work on inductive logic programming described above. In many of the works by Bessiere et al. (Bessiere et al. 2013; 2017b) the space of literals is determined by setting a language bias. To specify the language bias, it is assumed that both the variables V and the allowed basic constraints (such as $=, \neq, \geq$) are given. The set C of possible constraints is then the set of all possible literals that can be constructed with V and the basic constraints. This effectively allows one to propositionalize the learning problem. For instance, assume we have the variables A, B and C, and as only basic predicate $=$. Then each instance of (A, B, C) can be turned into a boolean feature vector. For instance, the assignment $(7, 9, 9)$ would have the values $(false, true, false)$ for the features $(A = B, B = C, A = B)$. In this way, the constraint learning problem reduces to that of learning a monomial (a conjunctive boolean expression) over the resulting features and the corresponding algorithms from computational learning theory apply.

Nevertheless, there are some special challenges that arise in the context of learning CSPs. One of these is concerned with the relationships that hold amongst the different constraint primitives. To continue the above example, whenever $A = B$ and $B = C$ hold it follows that also $A = C$ holds. This introduces redundancy in the hypotheses and search space and needs special techniques to cope with these (Abdennadher and Rigotti 2002). Another one is that CSPs typ-

ically have only a few solutions and thus there are only very few positive examples. Some works therefore combine the search for a solution to the learning task with that of finding a solution to the CSP itself (Bessiere et al. 2013).

Learning Soft Constraints

Numerous variants on the basic SAT problem have been considered in artificial intelligence. While the basic SAT problem (as used by Valiant) is looking for satisfying assignments, variations exist that look, for instance, for the number of satisfying assignments (model counting) or for the best satisfying assignment (maxSAT problem). The latter is an example of constraint optimization problem as one is looking for the assignment θ that scores best w.r.t. a particular objective function s that takes into account the degree to which the assignment θ satisfies the constraints (or clauses) C. For maxSAT the score is simply the number of clauses that are satisfied, and for weighted maxSAT, each clause obtains a weight and the score is the sum of the weights of the satisfied clauses.

Given that such soft constraint satisfaction problems are also prominent in the artificial intelligence literature (e.g., for probabilistic reasoning and decision theoretic planning), it can be no surprise that the learning of such soft constraint theories has also been studied. One typically distinguishes two settings. In the first setting, parameter learning, the constraints (or clauses) C are given and fixed and the goal is to learn the scoring function s, that is, the weights of the clauses. In the second setting, structure learning, both the constraints and the weights are being learned.

Parameter learning of soft constraints was pioneered by (Rossi and Sperduti 2004). Here the scoring function is estimated interactively by interleaving two steps. In the first step the algorithm chooses a few high-scoring assignments and presents them to the user, who ranks them according to their preferences. In a second step the parameters are fine-tuned based on the obtained supervision by employing max-margin techniques (Joachims 2002). Similar ideas were developed independently in the preference learning community (Domshlak et al. 2011) and recently employed for learning in customization and design problems (Teso, Passerini, and Viappiani 2016).

CLEO (Campigotto, Battiti, and Passerini 2015) adopts a similar approach for parameter learning of maxSAT and maxSMT (satisfiability modulo theories (Barrett et al. 2009)) theories in preference learning settings. It does so by exhaustively enumerating all clauses up to a given length, employing a sparsifying prior to set most learned weights to zero. This amounts to a form of structure learning – all constraints deemed irrelevant for fitting the supervision are associated to a null parameter, and effectively eliminated from the learned theory.

Learning weighted sets of clauses is closely related to probabilistic reasoning. Indeed, many inference problems in probabilistic graphical models (Darwiche 2009) can be reduced to weighted model counting problems. WMC solvers provide state of the art results in probabilistic inference. In addition, weighted max SAT solving is closely related to MAP and MPE inference in graphical models. Markov logic (Domingos and Lowd 2009) uses weighted first order logic clauses as its representation and WMC and weighted MAX SAT for probabilistic inference. It is a kind of soft extension of the first order clausal theories listed in the inductive logic programming subsection. Many papers have been written on learning the structure and the parameters of Markov Logic (Domingos and Lowd 2009).

Whereas Markov logic targets first order logic clauses, the frameworks of (Teso, Sebastiani, and Passerini 2017) and (Pawlak and Krawiec 2017) go one step further, targeting the learning of satisfiability modulo theories and mixed-integer linear problems. The induced models do not have a probabilistic interpretation.

User interaction through queries

Especially when learning hard constraint theories the use of active learning and queries has been popular. Rather than assuming that the examples are given and fixed, the learner interacts with a knowledgeable end-user in order to speed up the learning process. This is not surprising as learning from queries has proven to be extremely useful ever since Angluin's seminal paper (Angluin 1987).

Although many types of queries can in principle be used, let us illustrate this idea using membership queries, partial queries, and preference queries.

Membership queries simply ask whether a particular assignment satisfies the unknown target theory or not. As an illustration, consider the learning of CSPs using monomials. Given that our current hypothesis would be the monomial $c_1 \wedge ... \wedge c_n$ and we follow the Find-S algorithm one can in principle generate an example that satisfies $c_1 \wedge ... \wedge c_{i-1} \wedge \neg c_i \wedge c_{i+1} \wedge ... \wedge c_n$ and ask the user to classify it. In case it is positive, we can safely delete c_i from our current hypothesis and repeat the process. In case it is negative, we do know that c_i is a necessary condition (Mitchell 1997). Although the idea is simple, its application to CSPs is more tricky (Bessiere et al. 2017b) as there might not exist an assignment that satisfies $c_1 \wedge ... \wedge c_{i-1} \wedge \neg c_i \wedge c_{i+1} \wedge ... \wedge c_n$. This may be due to redundancies (such as the $A = C$ in the search space mentioned earlier).

In the same context, a partial query could ask whether a particular partial assignment Ev can be extended into a complete satisfying assignment for the unknown target theory. If the answer is positive, one can generalize the hypotheses in the current versionspace; if it is negative, one has obtained a very strong constraint on the possible solutions to the constraint learning problem. In that case, any constraint theory C for which $C \wedge Ev$ is satisfiable cannot be a solution the learning problem. Partial queries have been used in (Bessiere et al. 2016a).

Finally, when learning soft constraint theories, it is interesting to employ preference learning. For instance, (Rossi and Sperduti 2004) ask which of two solutions is to be preferred. This allows for tuning the scoring function by learning clause weights that correctly rank the stated preferences. An alternative especially suited for manipulative or design tasks are improvement queries (Shivaswamy and Joachims 2015), whereby the user constructs an assignment with higher score (than the learner-provided one).

When interacting with the user, the learner should of course focus on generating the most informative questions in order to minimize the burden on the user and to speed up the learning process.

Similar learning mechanisms were developed in the context of incremental preference elicitation (Pigozzi, Tsoukias, and Viappiani 2016), where the goal is to estimate the preferences of a customer so to suggest high-quality, personalized recommendations. The learned constraints can be either hard and convey qualitative preferences among alternatives, as in CP-nets (Boutilier et al. 1997), or weighted, in which case the weight of the satisfied constraints determines the utility of the recommendation (Braziunas and Boutilier 2007).

Despite the similarities, their focus is on producing high-quality recommendations as quickly as possible by exploring only the most promising models, rather than on learning a faithful constraint theory.

Decomposition and Structuring

When tackling constraint learning problems with many variables, it is often necessary to identify and exploit possible structures into the problem.

This is especially important when interacting with human decision makers, as they may have difficulties in parsing or understanding (potentially complex) complete instances. In this case decomposing or simplifying the interaction protocol, similarly to what is done with partial assignments in (Bessiere et al. 2013), can help to avoid overloading the user and consequently improve the reliability of the obtained supervision.

One impressive example from the constraint programming community is the work on ModelSeeker (Beldiceanu and Simonis 2012). ModelSeeker starts from a flat assignment, that is, a list of variables together with the values they have been assigned; ModelSeeker also employs a vast number of constraints from the global constraint library as primitive constraints. These global constraints (such as `alldiff`) are then used in the "bias" of ModelSeeker (cf. the subsection on constraint programming).

An innovative component of ModelSeeker is actually searching for structure in the flat list, which is a vector (matrix) of dimension 1 x n. ModelSeeker then considers numerous variations of this list, such as a matrices of dimension $m \times n/m$ (with m a divisor of n). For each of these resulting matrices it then looks for global constraints that hold row-, column- or diagonal-wise.

Challenges

There are many different challenges on constraint learning. A first one is concerned with the different representations of the constraint theory. So far, only the main representations have been considered. It would be interesting to look more into the many variations of the SAT problem, and into mixed Integer Linear Programing. A second one is that especially when learning soft constraints, the inference complexity must be controlled. Improving inference will lead to better learning techniques.

Finally, for what concerns interaction, most existing approaches expect the user to be able to parse and understand any given full assignment. Of course, this expectation is unrealistic in any sufficiently complex (i.e. real-world) constraint satisfaction problem. A promising direction involves studying interaction involving simplified instances or summaries. For instance, QuAcq (Bessiere et al. 2016a) learns hard constraint theories by exchanging partial assignments with the user. The caveat is that it is not immediately applicable to soft constraint problems, as the scoring function may not be defined over partial assignments. Further, in this setting the acquired labels refer to partial (and potentially small) assignments, "dumbing down" the amount of information conveyed by individual queries. Effective learning in this scenario might require one to increase the amount of information conveyed by individual queries, while keeping their scope sufficiently small: such "fat" queries might convey labels, explanations, arguments, or other kinds of data.

Acknowledgements

This work was supported by the European Research Council (ERC) Advanced Grant 694980 "SYNTH: Synthesising Inductive Data Models". The first author is grateful to Christian Bessiere, Anton Dries, Tias Guns and Helmut Simonis for many interesting discussions on learning constraints.

References

Abdennadher, S., and Rigotti, C. 2002. Automatic generation of rule-based solvers for intensionally defined constraints. *International Journal on Artificial Intelligence Tools* 11(2):283–302.

Angluin, D. 1987. Queries and concept-learning. *Machine Learning* 2:319–342.

Barrett, C. W.; Sebastiani, R.; Seshia, S. A.; and Tinelli, C. 2009. Satisfiability modulo theories. *Handbook of satisfiability* 825–885.

Beldiceanu, N., and Simonis, H. 2012. A model seeker: Extracting global constraint models from positive examples. In *Proc. Principles and Practice of Constraint Programming*, volume 7154 of *Lecture Notes in Computer Science*, 141–157.

Bessiere, C.; Coletta, R.; Koriche, F.; and O'Sullivan, B. 2005. A SAT-based version space algorithm for acquiring constraint satisfaction problems. In *Proc. European Conference on Machine Learning*, volume 3720 of *Lecture Notes in Computer Science*, 23–34. Springer.

Bessiere, C.; Coletta, R.; Hebrard, E.; Katsirelos, G.; Lazaar, N.; Narodytska, N.; Quimper, C.-G.; Walsh, T.; et al. 2013. Constraint acquisition via partial queries. In *Proc. 23rd International Joint Conference on Artificial Intelligence*, 475–481.

Bessiere, C.; Daoudi, A.; Hebrard, E.; Katsirelos, G.; Lazaar, N.; Mechqrane, Y.; Narodytska, N.; Quimper, C.-G.; and Walsh, T. 2016a. New approaches to constraint acquisition. In *Data Mining and Constraint Programming - Foundations of a Cross-Disciplinary Approach*, volume 10101 of *Lecture Notes in Computer Science*. Springer. 51–76.

Bessiere, C.; De Raedt, L.; Kotthoff, L.; Nijssen, S.; O'Sullivan, B.; and Pedreschi, D., eds. 2016b. *Data Min-*

ing and Constraint Programming - Foundations of a Cross-Disciplinary Approach, volume 10101 of *Lecture Notes in Computer Science*. Springer.

Bessiere, C.; De Raedt, L.; Guns, T.; Kotthoff, L.; Nanni, M.; Nijssen, S.; O'Sullivan, B.; Paparrizou, A.; Pedreschi, D.; and Simonis, H. 2017a. The inductive constraint programming loop. *IEEE Expert* 32(5):44–52.

Bessiere, C.; Koriche, F.; Lazaar, N.; and O'Sullivan, B. 2017b. Constraint acquisition. *Artificial Intelligence* 244:315–342.

Boutilier, C.; Brafman, R.; Geib, C.; and Poole, D. 1997. A constraint-based approach to preference elicitation and decision making. In *AAAI Spring Symposium on Qualitative Decision Theory*, 19–28.

Braziunas, D., and Boutilier, C. 2007. Minimax regret based elicitation of generalized additive utilities. In *Proc. of the Uncertainty in Artificial Intelligence Conference*, 25–32.

Campigotto, P.; Battiti, R.; and Passerini, A. 2015. Learning modulo theories for preference elicitation in hybrid domains. *CoRR abs:1508.04261*.

Darwiche, A. 2009. *Modeling and reasoning with Bayesian networks*. Cambridge University Press.

Davidson, I.; Guns, T.; and Nijssen, S. 2017. Data mining and machine learning using constraint programming languages. Tutorial presented at IJCAI.

De Raedt, L., and Dehaspe, L. 1997. Clausal discovery. *Machine Learning* 26:99–146.

De Raedt, L., and Džeroski, S. 1994. First order jk-clausal theories are PAC-learnable. *Artificial Intelligence* 70:375–392.

De Raedt, L.; Dries, A.; Guns, T.; and Bessiere, C. 2016. Learning constraint satisfaction problems: An ilp perspective. In *Data Mining and Constraint Programming - Foundations of a Cross-Disciplinary Approach*, volume 10101 of *Lecture Notes in Computer Science*. Springer. 96–112.

Domingos, P., and Lowd, D. 2009. *Markov Logic: An Interface Layer for Artificial Intelligence*. Morgan & Claypool Publishers.

Domshlak, C.; Hüllermeier, E.; Kaci, S.; and Prade, H. 2011. Preferences in artifical intelligence: An overview. *Artificial Intelligence* 175(7-8):1037–1052.

Flach, P., and Savnik, I. 1999. Database dependency discovery: a machine learning approach. *AI Communications* 12(3):139–160.

Harada, M.; Witkin, A.; and Baraff, D. 1995. Interactive physically-based manipulation of discrete/continuous models. In *Proc. 22nd Annual Conference on Computer Graphics and Interactive Techniques*, 199–208. ACM.

Joachims, T. 2002. Optimizing search engines using clickthrough data. In *Proc. ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, 133–142.

Kearns, M., and Vazirani, U. 1994. *An Introduction to Computational Learning Theory*. The MIT Press.

Kolb, S.; Paramonov, S.; Guns, T.; and De Raedt, L. 2017. Learning constraints in spreadsheets and tabular data. *Machine Learning* 106:1441–1468.

Lallouet, A.; Lopez, M.; Martin, L.; and Vrain, C. 2010. On learning constraint problems. In *Proc. IEEE International Conference on Tools with Artificial Intelligence*, 45–52.

Lee, E. A. 2008. Cyber physical systems: Design challenges. In *Proc. IEEE Symposium on Object Oriented Real-Time Distributed Computing*, 363–369.

Mitchell, W. J.; Steadman, J. P.; and Liggett, R. S. 1976. Synthesis and optimization of small rectangular floor plans. *Environment and Planning B: Planning and Design* 3(1):37–70.

Mitchell, T. M. 1997. *Machine Learning*. McGraw-Hill.

Muggleton, S., and De Raedt, L. 1994. Inductive logic programming: Theory and methods. *Journal of Logic Programming* 19/20:629–679.

O'Sullivan, B. 2010. Automated modelling and solving in constraint programming. In *Proc. 24th AAAI Conference on Artificial Intelligence*, 1493–1497.

Passerini, A.; Tack, G.; and Guns, T. 2017. Introduction to the special issue on combining constraint solving with mining and learning. *Artificial Intelligence* 244:1–5.

Pawlak, T. P., and Krawiec, K. 2017. Automatic synthesis of constraints from examples using mixed integer linear programming. *European Journal of Operational Research* 261(3):1141–1157.

Pigozzi, G.; Tsoukias, A.; and Viappiani, P. 2016. Preferences in artificial intelligence. *Annals of Mathematics and Artificial Intelligence* 77(3-4):361–401.

Rossi, F., and Sperduti, A. 2004. Acquiring both constraint and solution preferences in interactive constraint systems. *Constraints* 9(4):311–332.

Rossi, F.; Van Beek, P.; and Walsh, T. 2006. *Handbook of constraint programming*. Elsevier.

Shivaswamy, P., and Joachims, T. 2015. Coactive learning. *Journal of Artificial Intelligence Research* 53:1–40.

Smet, P.; De Causmaecker, P.; Bilgin, B.; and Vanden Berghe, G. 2013. Nurse rostering: a complex example of personnel scheduling with perspectives. In *Automated Scheduling and Planning*. Springer. 129–153.

Smith, C.; Ferns, G.; and Albarghouthi, A. 2017. Discovering relational specifications. In *Proc. 11th Joint Meeting on Foundations of Software Engineering*, 616–626. ACM.

Teso, S.; Dragone, P.; and Passerini, A. 2017. Coactive critiquing: Elicitation of preferences and features. In *Proc. 31st AAAI Conference on Artificial Intelligence*, 2639–2645.

Teso, S.; Passerini, A.; and Viappiani, P. 2016. Constructive preference elicitation by setwise max-margin learning. In *Proc. 25th International Joint Conference on Artificial Intelligence*, 2067–2073.

Teso, S.; Sebastiani, R.; and Passerini, A. 2017. Structured learning modulo theories. *Artificial Intelligence* 244:166–187.

Valiant, L. 1984. A theory of the learnable. *Communications of the ACM* 27:1134–1142.