Learning Constraint Satisfaction Problems: an ILP Perspective

Luc De Raedt[1], Anton Dries[1], Tias Guns[1], and Christian Bessiere[2]

[1] DTAI, KU Leuven
[2] CNRS, University of Montpellier

Abstract. We investigate the problem of learning constraint satisfaction problems from an inductive logic programming perspective. Constraint satisfaction problems are the underlying basis for constraint programming and there is a long standing interest in techniques for learning these. Constraint satisfaction problems are often described using a relational logic, so inductive logic programming is a natural candidate for learning such problems. So far, there is however only little work on the intersection between learning constraint satisfaction problems and inductive logic programming. In this note, we point out several similarities and differences between the two classes of techniques and use these to propose several interesting research challenges.

1 Introduction

Constraint programming (CP) is concerned with solving constraint satisfaction problems (CSPs). The instance of a CSP is a constraint network $p = (\mathcal{V}, D, \mathcal{C})$, defined by a finite set of variables $\mathcal{V} = \{v_1, ..., v_n\}$; a domain D, which maps every variable $v \in \mathcal{V}$ to a set of possible values $D(v)$; and a finite set of constraints $\mathcal{C} = \{c_1, ..., c_n\}$, where each constraint $c \in \mathcal{C}$ essentially corresponds to a relation. The question is to find an assignment of values to the variables so that all constraints in the constraint network are satisfied. A commonly used example of a CSP is that of Sudoku, where a 9x9 grid has to be filled up with numbers such that no number occurs more than once in the same row, column and block.

CSPs can be expressed in terms of *local* constraints. These constraints express simple relationships between a bounded number of individual variables, for example, $v_1 = v_2, v_3 \neq v_4, v_5 = v_6 + v_7 + v_8$.

However, the number of local constraints in a CSP can become very large. For example, in the case of Sudoku, we need 927 (= 36 x 27) local constraints. CSPs are therefore often expressed in terms of *global constraints*, which can represent sets of constraints. These have two main advantages: they simplify the model by reducing the number of constraints and solvers can more easily exploit the relationships between the constraints in the set. The standard example of a global constraint is the *alldifferent* constraint. For example, the constraint *alldifferent*$([v_1, v_2, v_3])$ is equivalent to the set $v_1 \neq v_2, v_1 \neq v_3, v_2 \neq v_3$. A Sudoku can be represented by 27 such constraints. Additionally, higher level languages for expressing CSPs (such as MiniZinc and Eclipse) offer constructs for expressing loops (e.g., *forall*). Using forall constructs, a Sudoku can be expressed as three statements, representing the constraints on rows, columns and blocks.

CSPs and constraint programming are being used in numerous applications, for example in domains such as time-tabling, scheduling, packing, bioinformatics, etc. However, formulating a complete CSP for an application is a non-trivial task. It can therefore be no surprise that several researchers have tackled the problem of learning the CSP from data [11,1,4,6].

In this article we argue that techniques from Inductive Logic Programming (ILP) are well suited for addressing this problem. At its core, ILP is concerned with learning a logic program (such as a CSP) given its output (such as solutions of the CSP). We show that the representational power of ILP techniques is capable of learning models with global constraints and higher level constructs such as *forall*.

Despite the apparent similarity between learning CSPs and ILP, this relationship has so far not yet received a lot of attention, but see [11,1]. This note is a first step towards alleviating this situation as it investigates the nature of the relationship between learning CSPs and ILP, and uses this to point out several interesting research directions.

2 Learning CSPs seen as an ILP problem

We will differentiate between two settings: 1) learning a single clause, and 2) learning multiple clauses. Each setting corresponds to a different approach to learning CSPs. The first corresponds to learning a constraint network, while the second corresponds to learning individual constraints that together make up a CSP.

2.1 Learning a single clause

One can see a CSP $p = (\mathcal{V}, D, \mathcal{C})$ as a single conjunctive clause of the following form:

$$\begin{aligned} p(v_1, ..., v_n) \text{ :- } & d(v_1), ..., d(v_n), \\ & c_1(v_{c_{1_1}}, v_{c_{1_2}}, ..., v_{c_{1_r}}), \\ & ..., \\ & c_m(v_{c_{m_1}}, v_{c_{m_2}}, ..., v_{c_{m_s}}). \end{aligned}$$

where $\mathcal{V} = \{v_1, ..., v_n\}$, $d(v_x)$ represents the domain of v_x and there are m constraints c_i, each involving a subset of the variables in $\mathcal{V}$.

In this setting, learning a CSP corresponds to learning a single clause for which $vars(head) = vars(body)$. This is the standard learning task in ILP. The definition of the c_i ould be part of the background knowledge (for example, *eq(), nq(), lq(), ...*). The goal is then to learn the definition of $p(v_1, ..., v_n)$ given this background knowledge and positive and negative examples.

Several observations can be made:

– CSPs are *conjunctive* descriptions and CP is heavily focussed on dealing with conjunctions as these impose strong constraints that – unlike disjunctive descriptions – propagate well in the search.

- $vars(head) = vars(body)$ assumes that all variables are explicit, and no new (auxiliary) variables are introduced by the constraints.
- The *number* of constraints in such CSPs can be quite large; it is typically much larger than the typical clauses learned in ILP; cf. the simple Sudoku example which already has 927 local constraints.
- Standard ILP systems often start from a large set of positive and negative examples. The number of solutions to a CSP problem is often small and it can already be hard to generate a single positive one. Therefore, several researchers are learning CSPs from queries [6,4] and from small sets of examples ([11,3]).

Within the existing approaches to learning CSPs, the Conacq [6] and Quacq [4] systems are state-of-the-art approaches that take this perspective.

Conacq employs a version-space like approach (Mitchell's FIND-S algorithm [12]). The version space is the space of all possible constraint networks that can be built on a given set of variables with constraints belonging to a given language. Conacq iterates over the examples to reduce the version space. The active version of Conacq [5] asks membership queries until the version space has converged on a single hypothesis. However, interdependencies between constraints make the test of convergence co-NP-complete. An efficient combination of redundancy rules given as background knowledge and of backbone tests on a clausal formulation of the version space make the convergence test efficient in practice.

Quacq is an extension of the active version of Conacq that is able to ask *partial* queries to reduce the number of queries required for convergence. Partial queries are queries that involve only a subset of the variables of the network. Thanks to this feature, for each example classified as negative, Quacq uses a dichotomic search to elucidate one constraint of the constraint network with a number of queries logarithmic in the size of the negative example.

The active version of Conacq and Quacq are particularly interesting in that they generate queries to the user. These queries basically ask whether a substitution for a set or subset of the variables in $\mathcal{V}$ violate the CSP or not. This allows it to converge more rapidly. From an ILP perspective, the setting is somewhat reminiscent of the interactive setting in Logan-H [10].

The existing approaches for learning CSPs, however, have several limitations for which ILP techniques might help:

- To reduce the number of local constraints, one could employ *global* constraints such as *alldifferent*. However, the number of literals for global constraints that could potentially belong to the body of a clause is exponential in the number of variables. Furthermore, there is vast number of global constraints that could be used (cf. the global constraint catalogue [2] which lists >400 constraints). This leads to a prohibitive large number of candidate literals. ILP can help by structuring the search over these constraints through the use of specialization/generalization operators.
- Typically, there are a vast number of syntactic variants in the hypotheses space when inducing CSPs, and this is an unsolved problem in several

CSP learning techniques. For instance, when working with an = constraint, symmetry and transitivity should be taken into account. In Conacq it was shown that explicitly adding redundant rules to the background knowledge can greatly improve performance. Possible automated solutions studied in ILP would be to work with *semantically closed* rules cf. [7,9].

2.2 Learning multiple clauses

In the above setting, an important part of the problem is to find the subsets of variables over which the constraints are active; in some cases, this also involves an order on the variables, for example the $lq()$ constraint. One solution is to consider all possible combinations of variables (e.g. all pairs of variables), and search for constraints over them. This is the typical approach taken when using version spaces.

However, a different approach is to learn the structure imposed on the set of variables. To see this, it is convenient to reconsider the Sudoku example in which there are 81 variables. By organizing them into the 9x9 matrix we have already solved a large part of the problem as the structure (a matrix) and potential global relationships (rows/columns etc) between the variables have been identified.

Consider the following clause:

$$\begin{aligned}&matrixdimension(X,Y), between(I,1,X),\\&\qquad selectrowvalues(I,X,Y,Values) \rightarrow alldiff(Values)\end{aligned}$$

where $matrixdimension(X,Y)$ succeeds if $X \times Y = n$, $between(I,1,X)$ if $I \in \{1,...,X\}$ and *selectrowvalues* would select the values corresponding to the positions in the list. All variables in this clause are universally quantified, which means that for range-restricted clauses, this is equivalent to learning a *forall* construct.

This clause identifies both the structure (the left-hand side) as well as the constraint (the right-hand side). It represents a subset of the constraints in the CSP (one for each row in the example above). Learning a CSP in this setting then corresponds to learning all of the clauses that together correspond to the CSP. The setting closely corresponds to that pursued by the clausal discovery systems Claudien [8]. Given positive and negative examples and background predicates, it learns the clauses that hold on the examples. The bodies of these clausal constraints correspond to the structure on the variables and learning such clauses can be viewed as a form of predicate invention or a change of representation.

Currently, two approaches exist that take this view on learning CSPs: ModelSeeker [3] and Lallouet et. al [11].

ModelSeeker searches for global constraints starting from an unstructured list of variables. For example, given the 81 Sudoku variables, it generates different structures on the set of 81 variables and for each of these examines which global constraints are satisfied. For instance, with 81 variables it can generate a 9 x 9 matrix, a 3 x 27 matrix, a 3 x 3 x 9 tensor, etc. The global constraints considered are those available in the global constraint catalog [2].

Formalizing the ModelSeeker approach in a clausal discovery setting could be realized by first storing each of the $v_1, ..., v_n$ variables with corresponding values $a_1, ..., a_n$ into facts $val(i, a_i)$, introducing $numberofvars(n)$ and then defining possible generators in the background theory.

Lallouet et. al propose an ILP approach to learning CPS rules, Constraint Problem Specification rules. They are logical rules of the form $head \rightarrow body$. A CPS is not a CSP $\mathcal{P} = (\mathcal{V}, D, \mathcal{C})$, it has to be matched to a partial solution (ground facts), to obtain a CSP. Because its input are ground facts, it can learn from examples of different sizes, for example n-queens for different values of n. The data is preprocessed in such a way that learning a set of CPS rules corresponds to learning a DNF on the negative examples. By negating the DNF, they can convert it into a conjunction of rules (clauses). Many ILP methods can be used to learn a DNF, however they discover that the search space is often too large for existing techniques and hence develop a bi-directional search method. It would be interesting to see whether techniques for directly learning CNFs (such as Claudien [8]) can be used to learn the CPS directly.

However, despite the correspondence between clausal discovery and learning generators and constraints, there also some significant differences:

– Unlike ILP systems, ModelSeeker does not structure the search space through a notion of generality, but rather pursues a highly optimized and tailored generate-and-test approach. There seems potential for approaches that combine the best of both worlds.
– Modelseeker can learn from a single example.
– A vast number of global constraints is considered and generated in a specific way that is tailored to the problem. A preference function is built-in to return the most promising one; this is usually the most specific one that holds for the example.
– Within the existing approaches to learning CSP, it assumed that the number of variables that is given is fixed, which in a sense turns the CSP learning problem into a propositional learning one. Using ILP techniques, like clausal discovery, this restriction could easily be lifted.
– When learning CSPs, one typically considers the noise-free case. It would however also be interesting to deal with noise. This could be related to learning soft constraints (cf. [14]).

3 Conclusion and Future Work

We see the learning of CSPs as a modern challenge in which many of the techniques and insights from ILP can play an important role. Lallouet and co [11] have shown how standard ILP techniques can be used to solve this problem, while Rigotti and co [1] have shown how clausal discovery methods can be used. However, none of the proposed approach matches up to the expert driven ModelSeeker [3] method.

Nevertheless, we believe that ILP-inspired approaches can extend the state-of-the-art in CSP learning, and we have sketched approaches for doing so. We believe that ILP can overcome some of the limitations of ModelSeeker, for example that the generator and constraint both have to be one literal, and that a conjunction of generators or constraints is not possible. Additionally, knowledge about the generalisation/specialisation of generators and constraints is currently not used during search, but only in post-processing. The ModelSeeker method also cannot take negative examples into account, nor is able to learn from examples of different sizes (although work has been done to overcome the latter restriction [13]). Additional issues such as implications between constraints (predicates) and redundancy have been tackled in the ILP community as well.

Many challenges remain though. The number of possible generators and constraints is large, leading to a huge search space. The number of examples of a certain problem can range from one to thousands. Additionally, the constraints can interact in many ways, including constraints that are equivalent dependent on the parameters they have.

Acknowledgements This work was supported by the European Commission under the project Inductive Constraint Programming (FP7- 284715).

References

1. S. Abdennadher and C. Rigotti. Automatic generation of rule-based solvers for intensionally defined constraints. *IJAIT*, 11(2):283–302, 2002.
2. N. Beldiceanu, M. Carlsson, and J.-X. Rampon. Global constraint catalog. `http://www.emn.fr/z-info/sdemasse/gccat/`.
3. N. Beldiceanu and H. Simonis. A model seeker: Extracting global constraint models from positive examples. In *CP*, pages 141–157. Springer, 2012.
4. C. Bessiere, R. Coletta, E. Hebrard, G. Katsirelos, N. Lazaar, N. Narodytska, C.-G. Quimper, and T. Walsh. Constraint acquisition via partial queries. In *IJCAI*, pages 475–481. AAAI Press, 2013.
5. C. Bessiere, R. Coletta, B. O'Sullivan, and M. Paulin. Query-driven constraint acquisition. In *IJCAI*, pages 50–55, 2007.
6. R. Coletta, C. Bessiere, B. O'Sullivan, E. C. Freuder, S. O'Connell, and J. Quinqueton. Semi-automatic modeling by constraint acquisition. In *CP*, pages 812–816. Springer, 2003.
7. L. De Raedt. *Logical and Relational Learning*. Springer, 2008.
8. L. De Raedt and L. Dehaspe. Clausal discovery. *ML*, 26(2-3):99–146, 1997.
9. L. De Raedt and J. Ramon. Condensed representations for inductive logic programming. *KR*, 4:438–446, 2004.
10. R. Khardon. Learning horn expressions with LogAn-H. In *ICML*, pages 471–478. Morgan Kaufmann Publishers Inc., 2000.
11. A. Lallouet, M. Lopez, L. Martin, and C. Vrain. On learning constraint problems. In *ICTAI*, pages 45–52, 2010.
12. T. M. Mitchell. Version spaces: A candidate elimination approach to rule learning. In *IJCAI*, pages 305–310. Morgan Kaufmann Publishers Inc., 1977.
13. N. Razakarison, M. Carlsson, N. Beldiceanu, and H. Simonis. GAC for a linear inequality and an atleast constraint with an application to learning simple polynomials. In *SOCS*. AAAI Press, AAAI Press, 2013.
14. F. Rossi and A. Sperduti. Solving and learning a tractable class of soft temporal problems: theoretical and experimental results. *JETAI*, 10, 1998.