Democratizing Constraint Satisfaction Problems through Machine Learning

Mohit Kumar, Samuel Kolb, Clement Gautrais, Luc De Raedt
KU Leuven, Belgium
mohit.kumar@cs.kuleuven.be, samuel.kolb@cs.kuleuven.be, clement.gautrais@cs.kuleuven.be, luc.deraedt@cs.kuleuven.be

Abstract

Constraint satisfaction problems (CSPs) are used widely, especially in the field of operations research, to model various real world problems like scheduling or planning. However, modelling a problem as a CSP is not trivial, it is labour intensive and requires both modelling and domain expertise. The emerging field of constraint learning deals with this problem by automatically learning constraints from a given dataset. While there are several interesting approaches for constraint learning, these works are hard to access for a non-expert user. Furthermore, different approaches have different underlying formalism and require different setups before they can be used. This demo paper combines these researches and brings it to non-expert users in the form of an interactive Excel plugin. To do this, we translate different formalism for specifying CSPs into a common language, which allows multiple constraint learners to coexist, making this plugin more powerful than individual constraint learners. Moreover, we integrate learning of CSPs from data with solving them, making it a self sufficient plugin. For the developers of different constraint learners, we provide an API that can be used to integrate their work with this plugin by implementing a handful of functions.

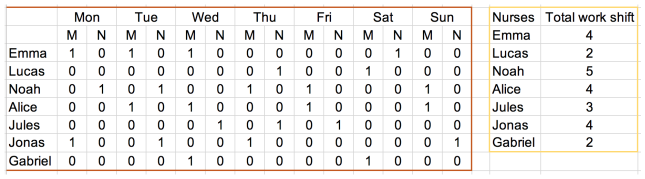
Introduction

In operations research as well as artificial intelligence, the use of constraints is ubiquitous. Many problems can be cast as a constraint satisfaction or optimisation problem. But while there is a lot of work on *using* constraints, there is only little work on learning them. This has changed in recent years, there is now a substantial interest in constraint learning from examples (De Raedt, Passerini, and Teso 2018). This is the problem of finding a set of constraints that satisfy a given dataset. In this way, constraint learning from examples wants to alleviate the model construction bottleneck in domains such as scheduling, rostering, transportation, etc.

Constraint learning techniques are emerging in many flavors, a key distinction being the underlying representation they are using. For example, Bessiere (Bessiere et al. 2016) learns arbitrary constraint models, which includes constraints on numeric variables, for example $x \leq y$ or $x \neq y$, where x and y are integer variables. ModelSeeker (Beldiceanu and Simonis 2016) takes a vector of values as input and learns global constraints like "all values should be different" or "the values should follow a sequence". INCAL (Kolb et al. 2018) learns satisfiability modulo theory from positive and negative examples, while ARNOLD (Kumar, Teso, and De Raedt 2019) acquires integer programs with polynomial inequalities using positive examples. Some constraint learners focus on more specific problems, for instance, COUNT-OR (Kumar et al. 2019) learns constraints specific to rostering problems and requires only positive examples, while TACLE (Paramonov et al. 2017) learns common spreadsheet formulae and relations.

Even though all these methods learn constraints, they differ in the type of constraints they can learn, the type of data they can use, and that determines the kind of problems they can be applied to. However, we often encounter datasets on which many of these learners can be applied simultaneously to uncover the underlying formalism. For instance, consider the example shown in Figure 1. The table highlighted with red shows the working schedule for different nurses in a week, where each day has a morning and a night shift. The other table (highlighted in yellow) shows the total number of working shifts in the week for each nurse. Here we can apply COUNT-OR to learn scheduling constraints like: "Every shift requires a Nurse", and at the same time apply TaCLe to learn that "the column in the second table is a sum of the corresponding rows in the first one". Therefore, combining the constraints learned by different approaches would yield a much more powerful model. To the best of our knowledge, this approach of combining multiple algorithms has not yet been pursued within constraint learning.

We make the following contributions. First, we introduce an interactive Microsoft Excel© plugin, named SYNTHCSP, for synthesizing and solving CSPs. As the name suggests, it combines both learning and solving of CSPs. Second, SYNTHCSP integrates different constraint learners, such as TACLE (Paramonov et al. 2017) and COUNT-OR (Kumar et al. 2019). This is made possible by encoding all learned constraints in a common language, which can then be solved using any off-the-shelf solver that supports that language. Finally, SYNTHCSP is independent of the underlying methods used to acquire constraints and supports each of them, we provide an easy to use API to integrate any constraint learner. For this demo, we integrated COUNT-OR and TACLE , but we are working on integrat-



	Mon		Tue		Wed		Thu		Fri		Sat		Sun	
	M	N	M	N	M	N	M	N	M	N	M	N	M	N
Emma	1	0	1	0	1	0	0	0	0	0	0	1	0	0
Lucas	0	0	0	0	0	0	0	1	0	0	1	0	0	0
Noah	0	1	0	1	0	0	1	0	1	0	0	0	1	0
Alice	0	0	1	0	1	0	0	0	1	0	0	0	1	0
Jules	0	0	0	0	0	1	0	1	0	1	0	0	0	0
Jonas	1	0	0	1	0	0	1	0	0	0	0	0	0	1
Gabriel	0	0	0	0	1	0	0	0	0	0	1	0	0	0

Nurses	Total work shift
Emma	4
Lucas	2
Noah	5
Alice	4
Jules	3
Jonas	4
Gabriel	2

Figure 1: Dataset with two tables, first one representing a schedule for nurses, while second one gives total number of working shifts for each nurse.

ing further approaches such as ARNOLD (Kumar, Teso, and De Raedt 2019) and INCAL (Kolb et al. 2018). We believe these features will contribute towards democratizing the use and learning of CSPs.

SynthCSP is part of VisualSynth, a project that aims at making data science available to spreadsheet users (Gautrais et al. 2020). VisualSynth interacts with spreadsheet users through visual hints, in the form of colored cells or simply by monitoring user actions such as cell editing, deletion or selection. Based on these interactions, VisualSynth can suggest data science tasks to perform, such as wrangling, model learning, prediction or constraint learning. In this paper, we focus on demonstrating the learning and solving of constraints in VisualSynth, which has never been presented before.

Learning and Solving CSP in Spreadsheet

Given a space of constraints S and data D, the problem of constraint learning from examples boils down to finding a set of constraints $C \subset S$, such that all constraints $c \in C$ are satisfied by the data D.[1] Constraint learning is studied here in the context of spreadsheets, which implies that the data D takes the form of a set of tables. Secondly, we will require that all constraints in the space S can be translated to a common language. We use Gurobi[2], which is one of the leading software to model mathematical programs, so we support all the constraints that can be represented in Gurobi, which includes linear as well as quadratic constraints. So far, the optimisation criteria in Gurobi has been not exploited, although in principle it would be easy to do this. The reason is that the current constraint learners do not simultaneously learn the optimisation criteria, but some recent work like (Kumar et al. 2020) does this and we plan to integrate it as a future work. Notice that, by changing the solver we can also integrate other approaches which cannot be modelled using Gurobi, this is also left as a future work.

Different constraint learners employ different strategies for learning. For instance, TaCLe (Paramonov et al. 2017) and Count-OR (Kumar et al. 2019) use a clever generate-and-test approach. They generate the constraints using a predefined vocabulary and test it on the examples to acquire a list of satisfied constraints, while making sure that only relevant constraints are generated. On the other hand, approaches such as ARNOLD (Kumar, Teso, and De Raedt 2019) combine a generality ordering on the constraints to guide and direct the search towards those constraints that are satisfied by the data. This approach allows for intelligent pruning very much in the spirit of concept-learning and frequent pattern mining. A third approach, used by works such as INCAL (Kolb et al. 2018), is to encode the learning problem as an optimization problem and to solve it using an off-the-shelf solver to acquire a constraint model satisfying the data. SynthCSP can accommodate all of these strategies.

Once the constraints have been learned, the focus shifts to the problem of constraint solving. Given data D with missing values and a set of constraints C, the aim is to fill out the missing values such that all constraints $c \in C$ are satisfied. The whole pipeline can be broken down into 4 major steps:

Extract relevant examples: Every constraint learner needs examples to learn the underlying constraints. The definition of example varies for different learners. For instance, TaCLe learns spreadsheet formulae and relations on tabular data and hence uses a table row as an example. Count-OR learns constraints on multi-dimensional boolean data and hence an example is a table with multiple headers and values containing 0s and 1s. As soon as the user opens a spreadsheet, SynthCSP looks for relevant examples corresponding to each constraint learner.

Learn constraints: Next, SynthCSP allows user to learn constraints using all approaches for which at least one relevant example was found in the spreadsheet.

Add constraints to a model: After learning, SynthCSP adds all learned constraints to a Gurobi model. To create the model, it first generates the list of cells affected by the learned constraints. Then it initializes a model in Gurobi with each of these cells as a variable. Finally it adds each constraint into the initialized model, making sure that the constraints are expressed in terms of the cell variables.

Solve: Finally the inference engine is called for the built model to complete any partial data or to generate new data. The complete demo can be found at https://youtu.be/7uM2Ap-Rv4Q. To add a constraint learner to SynthCSP, any system can use our API to define these steps by implementing a handful of functions.

Conclusion and Future Work

Constraint satisfaction problems are widely used to model and solve various real world problems. In this demo we bring the power of both learning and solving CSPs to spreadsheets. As spreadsheets are used widely to store data, our plugin democratizes constraint learning approaches and makes them easily accessible, also to the non-experts in this field. Moreover, as we provide a shared inference engine for multiple constraint learners, we can use constraints learned using different constraint learning algorithms to complete a dataset, this makes the plugin more powerful than individual constraint learners. As shown, our API can be used by any constraint learner to make their work easily accessible to the public with a very little effort, as long as the constraints can be represented in the common language. Going forward, we aim to integrate more constraint learners as well as solvers into SynthCSP.

[1]One could relax this using loss functions and allow for both soft and hard constraints, cf. (De Raedt, Passerini, and Teso 2018).

[2]https://www.gurobi.com/

Acknowledgments

This work has received funding from the European Research Council (ERC) under the European Union's Horizon 2020 research and innovation programme (grant agreement No. [694980] SYNTH: Synthesising Inductive Data Models). This research was also partially funded by the Data-driven logistics project (FWO-S007318N) and the Flemish government(AI research program).

References

Beldiceanu, N.; and Simonis, H. 2016. ModelSeeker: Extracting Global Constraint Models from Positive Examples. In *Data Mining and Constraint Programming - Foundations of a Cross-Disciplinary Approach*, 77–95. Springer.

Bessiere, C.; Daoudi, A.; Hebrard, E.; Katsirelos, G.; Lazaar, N.; Mechqrane, Y.; Narodytska, N.; Quimper, C.-G.; and Walsh, T. 2016. New approaches to constraint acquisition. In *Data mining and constraint programming*, 51–76. Springer.

De Raedt, L.; Passerini, A.; and Teso, S. 2018. Learning constraints from examples. In *Thirty-Second AAAI Conference on Artificial Intelligence*.

Gautrais, C.; Dauxais, Y.; Teso, S.; Kolb, S.; Verbruggen, G.; and De Raedt, L. 2020. Human-Machine Collaboration for Democratizing Data Science .

Kolb, S.; Teso, S.; Passerini, A.; and Raedt, L. D. 2018. Learning SMT(LRA) Constraints using SMT Solvers. In *Proceedings of the Twenty-Seventh International Joint Conference on Artificial Intelligence, IJCAI-18*.

Kumar, M.; Kolb, S.; Teso, S.; and Raedt, L. D. 2020. Learning MAX-SAT from Contextual Examples for Combinatorial Optimisation. In *The Thirty-Fourth AAAI Conference on Artificial Intelligence, AAAI 2020*, 4493–4500.

Kumar, M.; Teso, S.; Causmaecker, P. D.; and Raedt, L. D. 2019. Automating Personnel Rostering by Learning Constraints Using Tensors. In *31st IEEE International Conference on Tools with Artificial Intelligence, ICTAI 2019*.

Kumar, M.; Teso, S.; and De Raedt, L. 2019. Acquiring Integer Programs from Data. In *Proceedings of the Twenty-Eighth International Joint Conference on Artificial Intelligence, IJCAI-19*.

Paramonov, S.; Kolb, S.; Guns, T.; and Raedt, L. D. 2017. TaCLe: Learning Constraints in Tabular Data. In *Proceedings of the 2017 ACM on Conference on Information and Knowledge Management, CIKM 2017*, 2511–2514. ACM.