Constraint Acquisition Via Classification

S. D. Prestwich, E. C. Freuder, B. O'Sullivan, D. Browne

Insight Centre for Data Analytics
School of Computer Science & Information Technology
University College Cork, Ireland
{steven.prestwich,eugene.freuder,david.browne,
barry.osullivan}@insight-centre.org

Abstract. In constraint-based reasoning, the effort to bridge the gap between the current state-of-the-art and human-level Artificial Intelligence has been embodied in a long-standing challenge to address the "Holy Grail" of computer science: the user simply states a problem and the computer proceeds to solve it, without further programming. This in turn presents the challenge of automating the acquisition of a model of the problem in a form suitable for solving. Constraint acquisition methods can be used to acquire a model from examples of solutions and failures. However, in general they do not handle mixed data types, or very large, small or imbalanced datasets; they are susceptible to overfitting; and they are not designed to generate compact models. A two-step process — training a classifier to discriminate between solutions and failures, then transforming the trained classifier to a constraint model — can tackle these issues and opens up new application areas.

1 Introduction

A *constraint satisfaction problem* (CSP) involves a set of problem variables and sets of values that can be assigned to each variable. An instantiation of the variables assigns a value to every variable. Some of these instantiations will be solutions to the problem and some will be failures. A correct model of the problem specifies a set of restrictions (constraints) on which combinations of values are permitted, such that all and only the instantiations that satisfy all the constraints are solutions. In this paper the term *constraint satisfaction* is used very broadly, and the derived models might be formulated as constraint networks, SAT formulas or mathematical programs.

In constraint-based reasoning the effort to bridge the gap between the current state of the art and human-level Artificial Intelligence (AI) has been embodied in a long-standing challenge to address the "Holy Grail" of computer science: the user simply states a problem and the computer proceeds to solve it, without further programming [11]. The premise has been that Constraint Programming (CP) is well-suited to address this challenge, as once a problem is expressed as a constraint satisfaction (or optimization) problem there are general purpose algorithms that can, in principle, proceed to solve it. Furthermore, as such problems are ubiquitous within AI and have many practical applications [10], progress in this direction will have broad implications.

The field of *Constraint Acquisition* (CA) addresses the initial challenge of automating the expression of the problem as a constraint satisfaction problem. In CA we are

given examples of solutions and failures (*positive* and *negative* examples respectively) and the aim is to learn a constraint model that represents them. Beside the general goal of automated problem modelling, the model might be used as an explanation of the problem, to classify partial assignments, to show that a partial assignment cannot be placed in a class, to speed up the solution of future problems, or to find instances that optimise some objective.

The CA problem is defined in [20] as follows. We are given: a space X of $\boldsymbol{x}$ *instances* (assignments to variables V); a space of possible constraints C; an unknown target constraint theory $T \subseteq C$; and a set of training instances E, in which *positive* instances satisfy T while *negative* instances do not. The task is to find a constraint theory $H \subseteq C$ such that all positive instances in E are satisfied, and none of the negative instances. A more detailed formal definition and theoretical results are given in [3]. *Active* methods are guided by interaction with a user, but we shall consider only *passive* CA methods which learn automatically.

Several CA systems have been devised (see Section 3), based on machine learning, inductive logic programming and other methods, but current methods generally do not address practical difficulties such as mixed data types, scalability, overfitting, small or imbalanced datasets, and finding compact layered models. We propose a two-step approach: (i) train a classifier to discriminate between solutions and failures; (ii) transform the trained classifier to a constraint model. We call this approach *classifier-based constraint acquisition* (CLASSACQ) and it greatly extends the CA toolkit, offering new ways of tackling these issues.

The contributions of this paper are: we identify the connection between classification and CA; we outline several new classifier-based constraint models; we indicate which CA issues these might solve; and we discuss future directions for this fruitful line of research. Section 2 maps several classifiers to constraint models, Section 3 discusses related work, and Section 4 concludes the paper and discusses future work.

2 Classifiers and constraint acquisition

New connections between machine learning (ML) and optimisation are being established with increasing frequency, and ideas related to CLASSACQ are already known. A strand of work mentioned in a recent survey [17] is the representation of ML models as optimisation problems. For example trained decision trees (DTs) and some artificial neural networks (ANNs) can be compiled into optimisation models (see below and Section 3). However, this work was not motivated by CA and is rarely referenced in the CA literature.

In this section we show that CLASSACQ can address several practical issues that have not been tackled in the CA literature, and that it opens up new application areas. Except for the cases of DTs and ANNs we have not seen these constraint models defined elsewhere, and none seem to have been used for CA.

2.1 Mixed data types

CA systems typically generate discrete constraint models. However, many classifiers work on continuous variables, for example support vector machines (SVMs) and deep

learning classifiers. They can also be applied to categorical variables via *one-hot encoding* (also called *binarisation* or *reification*).

DTs and random forests (RFs) handle combinations of categorical, discrete and continuous variables in a natural way. There are at least three known ways of transforming a DT or RF to a CP: a rule-based method using Boolean meta-constraints, table constraints, and MDD-based global constraints [4]. [18, 26] used a method similar to the rule-based model for CP and MIP models. However, these are not generally thought of as CA methods, and their usual aim is to speed up solution methods by learning part of a model.

2.2 Large datasets

Not all CA systems scale up to large datasets but considerable effort has been put into classifier scalability. Deep learning classifiers can handle class sizes in the millions, and are often implemented on highly parallel architectures for speed. DTs and RFs have fast greedy training algorithms based on entropy measures. SVMs have a fast training algorithm based on quadratic programming. Naive Bayes classifiers have a very simple training algorithm that simply counts occurrences to estimate probabilities, and can easily handle very large datasets.

Taking the Bernoulli form of Naive Bayes as an example, if the data is binary then the classifier reduces to a single linear constraint stating that an example must lie on the positive side of the decision boundary. (If the data is nonbinary then we add binary auxiliary variables and channeling constraints, using standard MIP or SAT techniques. We omit the details for space reasons.) This might seem too trivial a model to be a useful for CA, yet Naive Bayes famously works surprisingly well for many problems, and the constraint model should be useful for the same applications, which include recommender systems, sentiment analysis and credit card fraud detection.

2.3 Small datasets

[19] discuss the possibility of using classifiers to generate constraint models, but criticise it on the grounds that the number of required examples grows exponentially with the number of variables in the instances. However, some classifiers are explicitly designed for small datasets [22]. In *few-shot learning* the training dataset has only a small number of examples from each class, or just one in the case of *one-shot learning*. Recent examples of such classifiers are *Prototypical Networks* [22] and *Matching Networks* [27] which are based on nearest-neighbour algorithms. Nearest-neighbour classifiers have been applied to problems with both large and small datasets, including image classification, recommender systems, document classification, medical diagnosis, facial recognition and theft prevention.

We can derive constraint models for such classifiers. A particularly simple example is the basic nearest-neighbour algorithm. If we have just one solution and one failure, the classifier reduces to a single constraint stating that an example is closer to the solution than to the failure. Depending on the distance metric, this might be expressed using a CP global distance constraint. It can be generalised to multiple examples and weighted k-nearest neighbours.

2.4 Imbalanced datasets

In some CA applications it is impractical to obtain a large dataset of negative examples. For example we might collect solutions automatically, but have no idea what failures look like. For such problems we can adapt *one-class classifiers* which are surveyed in [14] and can be used when the negative class is absent, poorly sampled or ill-defined. The aim of one-class classification is to recognise examples from a class, rather than to discriminate between classes. Its many applications include the detection of abnormal machine behaviour, automatic medical diagnosis and authorship verification [14]. Proposed approaches include a form of SVM, ANNs, DTs, nearest neighbours, genetic algorithms and Bayesian methods.

A simple example is the method of [6], which computes the convex hull of the training data (actually a computationally cheaper approximation based on random projections). A convex hull is a convex polytope which can be modelled exactly using a linear program. If the data is integral then integrality constraints can be added to obtain a MIP or finite-domain CSP.

The ModelSeeker CA system [2] also requires only positive instances, and has successfully found global constraint models for several applications. However, applications such as those above might not have a deep constraint structure to be discovered, and for these a model based on a one-class classifier is an interesting alternative. Both are ideal for historical data, but ModelSeeker requires a heuristic step to choose between alternative models.

2.5 Overfitting

Overfitting is a major problem in supervised learning, and occurs when a learner interprets errors or noise as data, or places too much emphasis on outliers. Current CA methods are not robust in this sense, and the seminal PAC learning algorithm of Valiant [25] for SAT is highly vulnerable to outliers. In contrast, many classifiers are designed to resist overfitting. For example soft-margin SVM explicitly allows a small number of exceptions. Bayesian classifiers are particularly robust as they are probabilistic in nature. Deep learning classifiers use the *dropout* technique to reduce overfitting by introducing noise, and often use a validation dataset to detect its occurrence.

As an example we consider SVMs which are state-of-the-art for a vast range of applications, including medical diagnosis, fault detection and satellite data. The simplest version learns a maximum-margin hyperplane, and we can impose a single constraint stating that an example lies on its positive side. This can be generalised to soft margins, and adding a kernel leads to a nonlinear constraint. This possibility is mentioned in [19] but criticised on the grounds that the resulting models are hard for humans to understand. We discuss this point in Section 3.

2.6 Layered models

In the CA literature the learned model is usually a set of constraints on the given variables. However, it is well known in CP that better models are sometimes obtained by

defining extra *auxiliary variables*, on which it might be easier or more powerful to express certain constraints. Auxiliary and given variables are connected to each other by adding *channeling constraints* to the model [7]. However, improved filtering is not the only motivation for creating auxiliary variables: for some problems they greatly reduce the size of the constraint model, for example the problem of finding *covering arrays* [13] (see [12] chapter 11.8 for a discussion of similar techniques for other problems). A similar result holds in SAT, where auxiliary variables can be used to obtain *Tseitin encodings* that are exponentially smaller than "flat" encodings [24].

Although auxiliary variables are an important modeling technique, their automatic discovery has not been addressed in the CA literature. Constraint models derived from DTs contain auxiliary variables, but they do not resemble the models generated by human experts and are not introduced explicitly to reduce model size. Ideally we require a method for discovering *useful* auxiliary variables, arranged in layers and connected by channeling constraints, which lead to compact constraint models.

For this we propose Deep Learning (DL) classifiers which have recently swept the field in many areas, including image and video analysis, bioinformatics and malware detection. It is known in DL that, although feedforward networks with a single hidden layer are *universal approximators* that can model any function with arbitrary accuracy, deep networks can be much more compact. DL also has techniques for reducing network size [5, 9] which in CLASSACQ leads directly to smaller constraint models. We therefore conjecture that CA will be particularly powerful when based on deep learning to learn what we might call *deep constraint models*. Compiling ANNs to optimisation models is well-known (see Section 3) but its connection to CA and compact layered models has not been previously pointed out, nor the use of network compression techniques to reduce model size.

3 Related work

Examples of CA systems include ModelSeeker [2] which requires only a few positive instances, and finds high-level descriptions in terms of global constraints; Tacle [15] which learns functions and constraints from spreadsheets; CONACQ [3] which is based on version spaces, and has passive and active versions; and the framework of [28] which learns several types of CP model by expressing CA as a constraint problem. Other systems for CA and related areas are cited in [3, 11, 20].

It is known that DTs can be transformed into constraint models [4, 18, 26], as can ANNs using *neuron constraints* [1, 16, 18]. ANNs can also be transformed to MIP models [8, 21, 23]. However, these methods are not generally used for CA. [1] model hard-to-describe parts of problems via neuron global constraints, embedded in a larger model designed by an expert. [16] transform DTs and ANNs into solvers called "consistency checking classifiers" which build propagators and answer partial queries. CLASSACQ does not build propagators but it uses a wider range of classifiers. [4, 18] embed DTs and ANNs into CP, MIP and other optimisation models. [8, 23] map ANNs to MIP to find inputs that optimise some objective, such as finding optimal adversarial examples or proving that none exist. [21] map ANNs to MIP, to solve planning problems with continuous action spaces. Thus DTs and ANNs have been mapped several times to var-

ious optimisation models, but usually not exactly for a CA application. Moreover, other classifiers have been neglected in this context, and we believe that several models we outlined are new. [17] survey the use of ML methods to boost combinatorial problem modelling, including the representation of ML methods within optimisation problems. They view CA as an extreme case in which an ML model completely replaces an optimisation model. In our view, as the aim of CA is to learn a constraint model that is compatible with given data, using classifiers to learn a complete constraint model is a form of passive CA.

[19] learn mathematical programming models from noisy training instances, containing linear, quadratic and trigonometric constraints. Their *constraint synthesis* method models and solves the CA problem as a MIP, using parameters to control features such as the number of allowed constraints, and their models are human-readable. They discuss the possibility of an approach similar to CLASSACQ, and mention that converting a trained ANN to a mathematical programming model requires the introduction of auxiliary variables and additional constraints, but they criticise this idea in two ways. Firstly, they mention a curse of dimensionality associated with CA: that the number of required examples grows exponentially with the number of variables in the instances. However, some classifiers are explicitly designed for very small datasets (as discussed in Section 2.3). Secondly, they mention the possibility of using classifiers such as SVMs and Naive Bayes to generate constraints representing decision boundaries between positive and negative instances, but criticise this approach on the grounds of transparency. We argue that constraint models need not be transparent for all applications, for example for testing whether a partial assignment can be extended to a positive example, for finding optimal adversarial examples, or for verifying classifier properties.

4 Conclusion

In this paper we have argued that CA can be achieved by finding a constraint model for a trained classifier. This idea, which we call CLASSACQ, has been briefly mentioned in the Machine Learning literature but criticised as impractical, and has not been pursued in the CA literature. We outlined several new classifier-based constraint models, and results from the classification literature indicate that these can alleviate drawbacks that have barely been addressed in CA: mixed-type data; large, small and imbalanced datasets; overfitting; and discovering layered compact models with auxiliary variables and channeling constraints (considered an advanced modelling technique in CP).

We conjecture that any trained classifier can, at least in principle, be used to derive a constraint model of some form. The diversity of known classifiers is therefore an important asset for CA: different classifiers and models have different advantages and suit different applications. Because classification is a mature field with many highly accurate and efficient algorithms, CLASSACQ greatly enriches the available tools and application areas for CA.

We believe that this connection is a useful research direction for CA, with many classifiers waiting to be used for special applications, and more efficient constraint models waiting to be discovered. We could also take advantage of work on automating classifier selection in classifier portfolios, and extend it to take into account the

suitability of the derived constraint models. This is a work in progress and we aim to present computational results in the near future.

Acknowledgments This publication has emanated from research supported in part by a research grant from Science Foundation Ireland (SFI) under Grant Number SFI/12/RC/2289 which is co-funded under the European Regional Development Fund. A version of this work was also presented at the IJCAI'19 workshop "Data Science Meets Optimisation".

References

1. A. Bartolini, M. Lombardi, M. Milano, L. Benini. Neuron Constraints to Model Complex Real-World Problems. *Proceedings of the 17th International Conference on Principles and Practice of Constraint Programming, Lecture Notes in Computer Science* vol. 6876, 2011, pp. 115–129.
2. N. Beldiceanu, H. Simonis. ModelSeeker: Extracting Global Constraint Models from Positive Examples. *Data Mining and Constraint Programming, Lecture Notes in Computer Science* vol. 10101, Springer 2016, pp. 77–95.
3. C. Bessiere, F. Koriche, N. Lazaara, B. O'Sullivan. Constraint Acquisition. *Artificial Intelligence* **244**:315–342, 2017.
4. A. Bonfietti, M. Lombardi, M. Milano. Embedding Decision Trees and Random Forests in Constraint Programming. *Proceedings of the International Conference on AI and OR Techniques in Constraint Programming for Combinatorial Optimization Problems, Lecture Notes in Computer Science*, Springer 2015, pp. 74–90.
5. D. Browne, M. Giering, S. D. Prestwich. Pulse-Net: Dynamic Compression of Convolutional Neural Networks. *Proceedings of the IEEE 5th World Forum on Internet of Things*, 2019.
6. P. Casale, O. Pujol, P. Radeva. Approximate Convex Hulls Family for One-Class Classification. *Proceedings of the International Workshop on Multiple Classifier Systems, Lecture Notes in Computer Science* vol. 6713, 2011, pp. 106–115.
7. B. M. W. Cheng, K. M. F. Choi, H. H. M. Lee, J. C. K. Wu. Increasing Constraint Propagation by Redundant Modeling: an Experience Report. *Constraints* **4**:167–192, 1999.
8. M. Fischetti, J. Jo. Deep Neural Networks as 0-1 Mixed Integer Linear Programs: A Feasibility Study. *Constraints* **23**(3):296–309, 2018.
9. J. Frankle, M. Carbin. The Lottery Ticket Hypothesis: Finding Sparse, Trainable Neural Networks. *Proceedings of the International Conference on Learning Representations*, 2019 (to appear).
10. E. C. Freuder. Constraints: The Ties that Bind. *Proceedings of the 21st National Conference on Artificial Intelligence,* AAAI Press 2006, pp. 1520–1523.
11. E. C. Freuder. Progress Towards the Holy Grail. *Constraints* **23**:158–171, 2018.
12. I. P. Gent, K. E. Petrie, J.-F. Puget. Handbook of Constraint Programming, Elsevier, 2006.
13. B. Hnich, S. D. Prestwich, E. Selensky, B. M. Smith. Constraint Models for the Covering Test Problem. *Constraints* **11**(3):199–219, 2006.
14. S. S Khan, M. Madden. One-Class Classification: Taxonomy of Study and Review of Techniques. *The Knowledge Engineering Review* **29**(3):345–374, 2014.
15. S. Kolb, S. Paramonov, T. Guns, L. De Raedt. Learning Constraints in Spreadsheets and Tabular Data. *Machine Learning* **106**:1441–1468, 2017.
16. A. Lallouet, A. Legtchenko. Two Contributions of Constraint Programming to Machine Learning. *Proceedings of the European Conference on Machine Learning, Lecture Notes in Artificial Intelligence* vol. 3720, Springer 2005, pp. 617–624.

17. M. Lombardi, M. Milano. Boosting Combinatorial Problem Modeling with Machine Learning. *Proceedings of the 27th International Joint Conference on Artificial Intelligence*, 2018, pp. 5472–5478.
18. M. Lombardi, M. Milano, A. Bartolini. Empirical Decision Model Learning. *Artificial Intelligence* **244**(Supplement C):343–367, 2017.
19. T. P. Pawlak, K. Krawiec. Automatic Synthesis of Constraints from Examples Using Mixed Integer Linear Programming. *European Journal of Operational Research* **261**(3):1141–1157, 2017.
20. L. De Raedt, A. Passerini, S. Reso. Learning Constraints from Examples. *Proceedings of the 32nd AAAI Conference on Artificial Intelligence*, 2018, pp. 7965–7970.
21. B. Say, G. Wu, Y. Q. Zhou, S. Sanner. Nonlinear Hybrid Planning With Deep Net Learned Transition Models and Mixed-Integer Linear Programs. *Proceedings of the 26th International Joint Conference on Artificial Intelligence*, 2017, pp. 750–756.
22. J. Snell, K. Swersky, R. Zemel. Prototypical Networks for Few-shot Learning. *Proceedings of the 31st Conference on Neural Information Processing Systems*, 2017.
23. V. Tjeng, R. Tedrake. Verifying Neural Networks with Mixed Integer Programming. CoRR, 2017.
24. G. Tseitin. On The Complexity of Derivation in Propositional Calculus. *Automation of Reasoning: Classical Papers in Computational Logic* **2**:466–483, J. Siekmann and G. Wrightson (eds.), Springer-Verlag, 1983.
25. L. G. Valiant. A Theory of the Learnable. *Communications of the ACM* **27**(11):1134–1142, 1984.
26. S. Verwer, Y. Zhang, Q. C. Ye. Auction Optimization Using Regression Trees and Linear Models as Integer Programs. *Artificial Intelligence* **244**:368–395, 2017.
27. O. Vinyals, C. Blundell, T. Lillicrap, K. Kavukcuoglu, D. Wierstra. Matching Networks for One Shot Learning. *Proceedings of the 30th Conference on Neural Information Processing Systems*, 2016, pp. 3637–3645.
28. X.-H. Vu, B. O'Sullivan. A Unifying Framework for Generalized Constraint Acquisition. *International Journal on Artificial Intelligence Tools* **17**(5):803–833, 2008.