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

CP’19 PTHG Workshop

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.
constraint acquisition/learning

CA methods can acquire a model from examples of solutions & failures.

CA comes from CP & is part of an effort to bridge the gap between the current state-of-the-art & human-level Artificial Intelligence, ie the long-standing “Holy Grail” challenge of computer science:

The user simply states a problem & the computer proceeds to solve it, without further programming.
CP is well-suited to address this challenge: 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.

Here we take a very broad view & allow CP, SAT, MIP, MINLP...
We are given: a space $X$ of $\vec{x}$ solutions & failures (positive & negative examples); a space of possible constraints $C$; an unknown target constraint theory $T \subseteq C$; & a set of training instances $E$, in which positive instances satisfy $T$ while negative instances do not.

The aim is to learn a constraint model that represents them: a constraint theory $H \subseteq C$ such that all positive instances in $E$ are satisfied, & none of the negative instances.

Active methods are guided by interaction with a user, but we consider only passive methods that learn automatically.
motivations

• automated modelling

• 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

• to find instances that optimise some objective.
some CA methods

Valiant’s algorithm learns k-SAT clauses by testing whether they hold on the training data.

ModelSeeker requires only a few positive instances, & finds high-level descriptions in terms of global constraints.

Conacq is based on version spaces, and has passive & active versions.

Tacle learns functions & constraints from spreadsheets.
Lallouet et al. use inductive logic programming.

The framework of [Vu & O’Sullivan 2008] learns several types of CP model by expressing CA as a constraint problem.

QuAcq and other methods are active, ie they learn by interaction with a user.

Other methods learn soft or weighted constraints.
practical issues

We think a more diverse toolbox is needed for CA:

- To handle mixed data types.
- To handle large, small, imbalanced or noisy datasets.
- To generate compact layered models.
the DSO connection!

We propose a 2-step process (**ClassAcq**):

- train a classifier to discriminate between solutions & failures
- transform the trained classifier to a constraint model

There are many classifiers designed to tackle the above issues, so ClassAcq opens up new approaches & application areas.
some related work

New connections are being made between ML and optimisation, & ideas related to ClassAcq are already known.

But proposing them for CA seems to be new, & the vast range of classifiers has not been exploited:

- ML models are represented as optimisation problems by [Lombardi & Milano 2018] to speed up combinatorial problem modeling. They view CA as an extreme case in which an ML model completely replaces an optimisation model.
Several researchers in AI & DL have transformed DTs, RFs & ANNs into CP or MIP models, but not for CA. This work is rarely referenced in the CA literature.

Others map ANNs to MIP to find inputs that optimise some objective, eg finding optimal adversarial examples or proving that none exist.

[Pawlak & Krawiec] learn MP models from noisy training instances, containing linear, quadratic & trigonometric constraints: constraint synthesis. They discuss the possibility of an approach similar to ClassAcq but mention 2 drawbacks...
A curse of dimensionality: the number of required examples grows exponentially with the number of variables in the instances. But some classifiers are explicitly designed for very small datasets.

The resulting models are not very transparent. We argue that models need not be transparent for all applications, eg for testing whether a partial assignment can be extended to a positive example, for finding optimal adversarial examples, or for verifying classifier properties.
We believe ClassAcq can address several practical issues that have barely (or not at all) been tackled in the CA literature.

Following are some issues & classifiers that might be used to address them...
Mixed data types

CA systems typically generate discrete constraint models. But many classifiers work on continuous variables, eg SVMs & DL classifiers.

They can also be applied to categorical variables via one-hot encoding [binarisation, reification].

DTs & RFs handle combinations of categorical, discrete & continuous variables in a natural way.
There are at least three known ways of transforming a DT or RF to a CP or MIP model: a rule-based method using Boolean meta-constraints, table constraints, & MDD-based global constraints.

So this is already known, but it’s not generally thought of as CA: the usual aim is to speed up solution methods by learning part of a model.
Large datasets

Not all CA systems scale up to large datasets, but considerable effort has been put into classifier scalability:

- DL classifiers can handle class sizes in the millions, are often implemented on highly parallel architectures, & can classify images, sounds, text...

- DTs & RFs have fast greedy training algorithms based on entropy measures.
• SVMs have a fast training algorithm based on QP.

• Naive Bayes classifiers have a very simple training algorithm that simply counts occurrences to estimate probabilities, & can easily handle very large datasets. Other Bayesian classifiers can be more accurate.

More on NB later...
Small datasets

Some classifiers are explicitly designed for very small datasets.

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 are *Prototypical Networks* & *Matching Networks*, based on nearest-neighbour algorithms.
NN classifiers have been applied to problems with both large & small datasets: image classification, recommender systems, document classification, medical diagnosis, facial recognition, theft prevention...

We can derive constraint models from such classifiers, eg basic NN: if we have just one solution & 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 *global distance constraint*.

Can be generalised to multiple examples & weighted k-NN.
Imbalanced datasets

In some CA applications it is impractical to obtain a large dataset of negative examples. Eg we might collect solutions automatically, but have no idea what failures look like.

We can adapt one-class classifiers which 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.
Some applications: detection of abnormal machine behaviour, automatic medical diagnosis, authorship verification.

Proposed approaches include SVM, ANNs, DTs, nearest neighbours, Bayesian methods.

A very simple approach: find the convex hull of the training data (or a computationally cheaper approximation based on random projections) [P. Casale, O. Pujol, P. Radeva 2011]. A convex hull is a convex polytope that can be modelled exactly using an LP.
The ModelSeeker CA method also requires only positive instances, & has successfully found global constraint models for several applications.

But applications such as those above might not have a deep constraint structure to be discovered.

For these a model based on a one-class classifier is an interesting alternative.
Overfitting

This is a major problem in supervised learning: a learner interprets errors or noise as data, or places too much emphasis on outliers.

Most CA methods are not robust in this sense, eg the PAC learning algorithm of Valiant for SAT is highly vulnerable to outliers.
In contrast, many classifiers are designed to resist overfitting:

- **Soft-margin SVM** explicitly allows a small number of exceptions.

- **DL classifiers** use *dropout* to reduce overfitting by introducing noise, & often use a validation dataset to detect its occurrence.

- **Bayesian classifiers** are particularly robust as they are probabilistic in nature.
Eg take SVMs which are state-of-the-art for a vast range of applications, including medical diagnosis, fault detection & satellite data.

The simplest version learns a maximum-margin hyper-plane, & we can impose a single constraint stating that an example lies on its positive side. This can be generalised to soft margins, & adding a kernel leads to a nonlinear constraint.

This possibility is mentioned by [Pawlak & Krawiec] (but they say that the resulting models are hard for humans to understand).
“deep” constraint models

In the CA literature the learned model is usually a set of constraints on the given variables.

But 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 & given variables are connected to each other by adding channeling constraints to the model, & there might be multiple “layers” of variables.
Improved filtering is not the only motivation for creating auxiliary variables: for some problems they greatly reduce the size of the constraint model, eg covering arrays.

A similar result holds in SAT, where auxiliary variables can be used to obtain Tseitin encodings that are exponentially smaller than “flat” encodings.

We might call these “deep models”. Their automatic discovery has not been addressed in the CA literature.
Models from DTs actually contain auxiliary variables, but they do not resemble the models generated by human experts & are not introduced explicitly to reduce model size.

To learn them we propose DL classifiers. These have recently swept the field in many areas, eg image & video analysis, bioinformatics & 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.
Compiling ANNs to optimisation models is well-known but its connection to CA & compact layered models has not been previously pointed out. There are also network compression techniques that could reduce model size.

We have started DL experiments but found a negative result. We created training data whose examples were vectors of bits, randomly set but each with at least as many 1s as 0s, & trained deep Heaviside networks. The aim was to learn a compact SAT-encoding of the cardinality constraint...
\[ \sum_{i=1}^{2N} v_i \geq N \]

which is known to exist.

But we were unable to find an encoding using a state-of-the-art system (TensorFlow on a GPU).

A likely explanation is the non-differentiability of Heaviside functions: evolutionary search might yield better results on this problem.
Bayesian CA

We had more success with NB:

• Map the training data to features: 1 for constraint violation, 0 for satisfaction, for a set of candidate constraints.

• Train NB to distinguish solutions from non-solutions based on these features.

• Derive a constraint model from the trained NB.

(Details in future work.)
So far this approach has successfully learned constraints for Latin squares, Sudoku, Golomb rulers & clauses for random 3-SAT instances.

It is not confused by redundant constraints, which can confound version space methods.

Can be viewed as *Bayesian hypothesis testing*. 
As a CA method it is:

- **fast**: orders of magnitude faster than version spaces
- **scalable**: needs no more memory than the size of the training data
- **robust**: errors are swamped by enough correct data
- **general purpose**: in principle it can learn any hard constraint
The ClassAcq idea has been hinted at in the ML literature, but criticised as impractical & opaque, & has not been pursued in the CA literature.

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 an asset for CA: different classifiers & models have different advantages & applications.

We could also take advantage of work on automating classifier selection in classifier portfolios.
THE END

An earlier version of this work was presented at the IJ-CAI’19 workshop “Data Science Meets Optimisation”.