Alternative Machine Learning Methods for Constraint Acquisition

Steve Prestwich UCC

This material is based upon works supported by the Science Foundation Ireland CONFIRM Centre for Smart Manufacturing, Research Code 16/RC/3918. We would also like to acknowledge the support of the Science Foundation Ireland under Grant No. 12/RC/2289-P2 which is co-funded under the European Regional Development Fund.

Constraint programming

CP is the area of AI concerned with modelling and solving combinatorial problems, with rich modelling languages.

A CSP has a set of problem variables, each with a domain of possible values, and a network of constraints imposed on subsets of the variables. CSPs are NP-complete problems.

A constraint is a relationship that must be satisfied by any solution, though it can be violated by non-solutions.

Constraint acquisition

Modelling a CSP requires knowledge and experience, and can be difficult even for experts: the *modelling bottleneck*.

CA attempts to avoid the bottleneck by automating modelling.

It's a possible path to the Holy Grail.

In CA we have a set of labelled *instances*, and constraint *candidates* (the *bias*). The labels are positive for solutions and (often) negative for non-solutions.

We wish to learn a subset S of the bias such that positive instances satisfy all S while negative instances (if we have any) do not.

(Some approaches also require that every non-solution violates at least 1 learned constraint.)

CA is usually based on some form of Machine Learning (ML), typically Inductive Logic Programming or Version Space Learning.

I've been exploring the use of alternative ML methods for CA, and Gene asked me to summarise the results.

Running example

A Latin square is similar to a Sudoku as a CSP, but without boxes.

It's an $N \times N$ square of integers in the range $\{1, \ldots, N\}$ s.t. no integer occurs more than once in any row or column.

Used for CA in several papers with $N = 10$. Bias has 14,850 $\{\leq, \geq, \neq\}$ candidates, with 900 of the disequalities to be learned.

I'll skip **ConAcq**, **QuAcq** etc to look at alternative ML approaches...

Supervised learning

This was presented at PTHG'19 and IJCAI-DSO'19 and published here:

S. D. Prestwich, E. C. Freuder,
B. O'Sullivan, D. Browne.
Classifier-Based Constraint Acquisition.
Annals of Mathematics and Artificial Intelligence
89:655–674, 2021.

For training data containing positive and negative instances, we proposed a 2-step process called **ClassAcq**:

- train a classifier to discriminate between solutions & failures

- transform the trained classifier to a constraint model

Any CA method based on a classifier should inherit its characteristics.

Classifiers exist for mixed data types, and large or small / imbalanced / noisy / low- or high-dimensional datasets. So **ClassAcq** provides a diverse toolbox for CA.

Not a completely new idea! Several researchers have transformed decision trees, random forests & neural networks into CP or MIP models [Lombardi, Milano & Bartolini; Pawlak & Krawiec] — but not expressly for CA, and this work is rarely referenced in the CA literature.

BayesAcq

As a **ClassAcq** experiment we used a Bernoulli naive Bayes classifier, which is fast and can handle noisy data (with errors): both nice properties for CA.

Given a vector $\vec{x} = \langle x_1, \ldots, x_N \rangle$ of values x_i to be classified, Naive Bayes classifiers usually select a class using the *maximum a posteriori* rule to choose the most likely class:

$$\text{argmax}_k \left(\Pr(C_k) \prod_{i=1}^{N} \Pr(x_i|C_k) \right) \qquad (1)$$

This rule selects the class k that is the mode of the posterior distribution.

To train the classifier we estimate the prior class probabilities $\Pr(C_k)$, and the conditional probabilities $\Pr(x_i|C_k)$ of observing x_i in class C_k: we simply count values in the dataset, so training is fast and scalable.

An assumption is that the x_i are values of independent variables or features. Although this is often unrealistic, Naive Bayes classifiers often give surprisingly good results, are provably optimal for some cases, and are a standard tool for some applications. They are also robust under noise and errors, because corrupted data can be neutralised by sufficient correct data.

The NB assumption of independence between variables seems to make them unsuitable for learning constraints between variables, but we can combine tuples of variables into single features: "constraints as features" (a standard trick).

For CA the classes are $k \in \{+, -\}$, and an instance is in $+$ (a solution) iff:

$$\prod_j \frac{\Pr(c_j = 1|C_-)}{\Pr(c_j = 1|C_+)} < \frac{\Pr(C_+)}{\Pr(C_-)}$$

We assume an *uninformed prior* $\Pr(C_+) = \Pr(C_-)$ so that an instance is classed as a solution iff:

$$\prod_j \frac{\Pr(c_j = 1|C_-)}{\Pr(c_j = 1|C_+)} < 1 \quad \text{or} \quad \sum_j \ln\left(\frac{\Pr(c_j = 1|C_-)}{\Pr(c_j = 1|C_+)}\right) < 0$$

From this we can derive a linear constraint

$$\sum_j c_j \ln\left(\frac{\Pr(c_j = 1|C_-)}{\Pr(c_j = 1|C_+)}\right) < 0$$

that mimics a Naive Bayes classifier given c_j values.

Given any previously unseen instance, we can compute the c_j then test the linear constraint; if it is satisfied then the instance is classified as a solution; if it is violated the instance is classified as a non-solution.

But a single linear constraint on binary variables is not the constraint model we desire. Instead we want to learn which candidates j (expressed on the original problem variables) should be in the model.

Fortunately, in practice the coefficients of c_j for actual constraints are quite large positive values, while those for non-target candidates have positive or negative values close to 0. So we force $c_j = 0$ for candidates j with large coefficients, and ignore all other candidates j because there is insufficient evidence that they are constraints: a heuristic that seems to work well.

We can now discard Naive Bayes and the c_j leaving a simple CA method: for each candidate j compute $K_j = \Pr(\text{viol}(j)|C_-)/\Pr(\text{viol}(j)|C_+)$ where viol(j) denotes violation of candidate j. If K_j is greater than some threshold κ then learn candidate j as a constraint, otherwise ignore it.

To prevent zero probabilities and avoid infinities we use *additive smoothing*:

$$K_j = \frac{\mathbf{n}(\text{viol}(j), C_-) + \alpha}{\mathbf{n}(\text{viol}(j), C_+) + \alpha}$$

where $\mathbf{n}(\text{viol}(j), C)$ denotes the number of instances in class C that violate candidate j, and $\alpha > 0$ is a smoothing parameter.

Choosing hyperparameters

We choose $\alpha = 0.01$: a common choice in additive smoothing. The K_j can be viewed as Bayes factors and we measured them in Turing's *decibans*:

K_j	decibans	weight of evidence
$< 10^0$	0	negative
10^0–$10^{1/2}$	0–5	barely worth mentioning
$10^{1/2}$–10^1	5–10	substantial
10^1–$10^{3/2}$	10–15	strong
$10^{3/2}$–10^2	15–20	very strong
$> 10^2$	>20	decisive

In summary, **BayesAcq** computes K_j for each candidate j, and accept j as a constraint iff $K_j > \kappa$ for some chosen κ (and α).

On our running example, we used 5000 solutions and 5000 failures. Disequalities that should be learned had *decisive* factors, while all other candidates were less than *substantial*, and it learned the correct candidates in 0.54 seconds.

Similar results for Sudoku, Golomb rulers and random 3-SAT.

Despite its approximations and NB's incorrect independence assumption, **BayesAcq** gives accurate results on common benchmarks, and inherits NB's properties:

Speed On a Golomb ruler of length 12 it took 0.07 seconds, while other methods took minutes or hours.

Scalability Most problems in the literature have a bias with tens of thousands of candidates. **BayesAcq** learned the 50 clauses for a random 3-SAT problem with a bias of 1.3B candidates, in 16,259 seconds.

Robustness In experiments **BayesAcq** gave correct results with up to 10% misclassified instances (though this required some hyperparameter tuning).

A sequential approach

Still under the heading of supervised learning, another CA method published here:

S. D. Prestwich.
Robust Constraint Acquisition by
Sequential Analysis, ECAI 2020.

BayesAcq does something like Bayesian hypothesis testing, so why not use a faster method?

Sequential analysis is a form of hypothesis testing in which a *stopping rule* is used to stop sampling as soon as the accumulated evidence is sufficient to accept or reject the hypothesis.

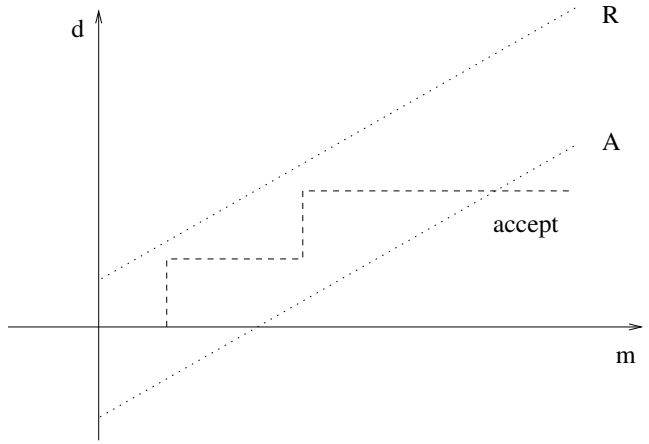
This has obvious benefits for patients in clinical trial, which can be halted as soon as it becomes obvious that an experimental treatment is harmful, or that one treatment is much more successful than another.

Another application is in manufacturing, where product lots are tested for defects: lots should be accepted or rejected after as few tests as possible, to save time and costs.

A similar approach called *Banburismus* was developed independently by Turing to speed up decryption. There are many more applications in the literature.

In CA the stopping rule might enable us to avoid testing a candidate against every training instance.

We use Wald's Sequential Probability Ratio Test (SPRT). Eg products are sampled and tested one by one ($m = 1, 2, \ldots$), counting the number d_m of defects found so far. If at any point $d_m < A_m$ the lot is accepted and the algorithm halts, where A_m is an *acceptance number*. On the other hand, if at any point $d_m > R_m$ the lot is rejected and the algorithm halts, where R_m is a *rejection number*. Otherwise the algorithm continues indefinitely.

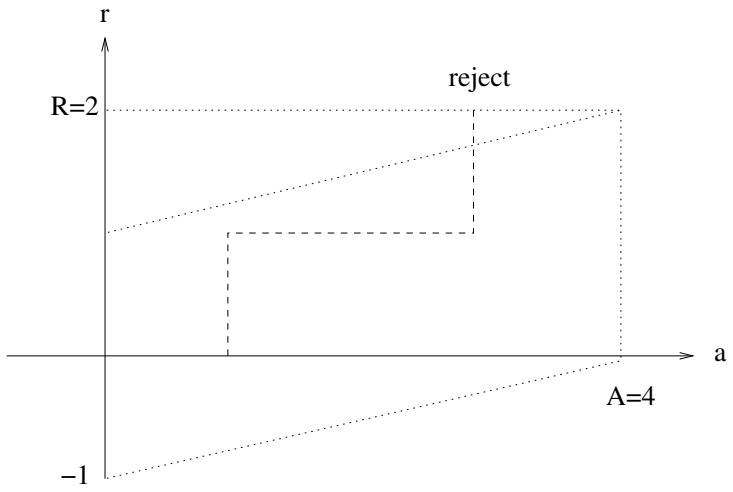


4 parameters specify the sampling plan: p_0 is the defect probability below which we prefer acceptance, p_1 is the defect probability above which we prefer rejection, α controls the type I and β the type II error rate.

The **SeqAcq** method has only 2 parameters (A, R), one of which (R) can be set to 1 if we expect no errors in the data:

SeqAcq(R,A)
 for each candidate c in the bias
  $r \leftarrow 0 \quad a \leftarrow 0$
  repeat
   randomly choose an instance e without replacement
    (if impossible then reject c as inconclusive)
   if c is violated by e
    if the instance is a solution
     $r \leftarrow r + 1$
     if $r \geq R$ reject c as a constraint
    $a \leftarrow a + 1$
    if $a \geq A$ accept c as a constraint

Here's an example where a candidate is violated by 2 sampled solutions. These cause two vertical moves (horizontal moves correspond to non-solution violations) which lead to rejection because $R = 2$. But if A non-solution violations were observed first then c would be accepted as a constraint:



Truncated SPRT

If there aren't enough violations of a candidate in the dataset (eg `all-equal`) then it will be rejected as inconclusive. To handle these cases we modify **SeqAcq** slightly: accept inconclusive candidates for which $r = 0$ and $a > 0$.

SPRT is often modified to handle inconclusive cases, yielding a Truncated SPRT that accepts or rejects them on the basis of a limited number of samples.

Results

SeqAcq is also robust and scalable.

It also gives correct results on Latin squares but 10× faster than **BayesAcq**. Similarly for Sudoku.

On the large 3-SAT example it's about 200 times faster, processing the bias of 1.3B candidates in 78 seconds.

Semi-supervised learning

Continuing the tour of ML areas, *semi-supervised learning* might be useful for CA. Suppose we have a lot of training instances but only a few are labelled as positive or negative. This might be the case in datasets that were scraped from the Web. Can we use the labelled instances and also benefit from the unlabelled instances?

A possibility is to improve CA by using *transduction* to label the unlabelled instances, giving a large labelled dataset almost for free. (Presented at PTHG'20.)

I experimented with this idea using a semi-supervised naive Bayes classifier based on the Expectation Maximisation (EM) algorithm.

Unfortunately the results were poor: each EM iteration increased the classification error.

I also tried using NB to generate binary labels where this could be done with high confidence, leaving other instances unlabeled; but even when transduction was quite accurate, CA on the large relabelled data was usually less accurate than on the small labelled data.

Other researchers have also found EM to make learning less accurate, so the result is perhaps unsurprising. A possible explanation is that our CA data violated the assumptions on which transduction relies.

In fact *it is impossible to guarantee that the introduction of unlabelled data will not degrade performance*:

J. E. van Engelen, H. H. Hoos.
A Survey on Semi-Supervised Learning.
Machine Learning 109:373–440, 2020.

Perhaps someone else can get this idea to work?

But semi-supervised learning might be unnecessary...

Unsupervised learning

What if we have *no* labelled instances? Can we learn a constraint model from a dataset of *unlabelled* instances (a set of total variable assignments that may or may not be solutions)?

Perhaps surprisingly: yes!

S. D. Prestwich.
Unsupervised Constraint Acquisition.
ICTAI 2021.

Motivation

In many ML applications, unsupervised learning is the key to scalability. Some experts believe that it's the future of AI:

> K. Hao.
> The AI Technique That Could Imbue Machines With the Ability to Reason.
> MIT Technology Review, 2019.

Finding unsupervised CA methods was proposed as a research challenge by Barry O'Sullivan over 10 years ago.

All current CA methods use *supervised learning* and need data prepared by humans. Meeting this challenge could lead to automated data collection and hence automated CA.

So unsupervised CA could be very useful.

Though CA can remove the modelling bottleneck, it introduces what is sometimes called a *data collection bottleneck*.

This is a serious drawback. Eg to learn a standard Sudoku puzzle, most methods require labelled datasets of about 10,000 instances. Providing these requires significant human effort.

For some applications there might be no known solutions, or they might be difficult or expensive to obtain, or we might not know what a non-solution looks like.

Intuition

Suppose we have training data for 10×10 Latin squares, containing some solutions and some non-solutions — but we don't know which are which, nor how many there are.

Any random variable assignment is unlikely to be a solution, because a Latin square is quite a tightly-constrained CSP. So any solutions must have been artificially added, with all Latin square constraints satisfied.

Intuitively, shouldn't target constraints be more often satisfied than non-targets in such a dataset?

Also consider 9×9 Sudoku. There are $\approx 6.7 \times 10^{21}$ solutions and 9^{81} possible matrices, giving the vanishingly small probability 3×10^{-56} of finding a solution in a purely random dataset.

So any dataset for Sudoku that contains solutions satisfies the constraints more often than a random set of instances.

Target constraints should be satisfied more often than would be expected, given the distribution of values. This might be used to distinguish them from other candidates that should not be learned.

But are real-world datasets like this? How often should we expect constraints to be satisfied? For inspiration we turn to Data Mining...

Association rule mining

I proposed a method called **MineAcq** based on ARM, a form of unsupervised learning and one of the main areas of data mining.

ARM extracts interesting correlations, patterns and associations among items in databases.

Eg for a set of items $I = \{\text{milk}, \text{bread}, \text{butter}, \text{beer}, \text{diapers}\}$ a rule is

$$\{\text{butter}, \text{bread}\} \Rightarrow \{\text{milk}\}$$

ARM has many application areas: marketing, customer relationship management, medical diagnosis, census data, protein sequences.

To learn rules we can test every possible rule (but there are shortcuts) and learn those that are interesting under some measure, eg *support* of $X \Rightarrow Y$

$$p(XY)$$

interest (or *lift*)

$$\frac{p(XY)}{p(X)\,p(Y)}$$

conviction

$$\frac{p(X)\,p(\bar{Y})}{p(X\bar{Y})}$$

MineAcq tests each candidate using a measure of interestingness, and learn those that pass the test.

But constraints are not association rules: they (usually) have no antecedent or consequent. So how to do this?

A measure that doesn't rely on these is

$$\frac{1 - \mathbb{E}[p(c(\vec{x}))]}{1 - p(c(\vec{x}))}$$

ie the ratio of the expected and observed *violation probabilities*. This should be larger than expected for target constraints.

Applied to a rule $X \Rightarrow Y$ it is equivalent to

$$\frac{p(X)\,p(\bar{Y})}{p(X\bar{Y})}$$

which is the *conviction* V. We learn candidate c if $V(c) \geq \tau$.

But conviction has a drawback: a candidate that's almost never violated has low V.

So we also learn any candidates that are almost never violated, using the measure:

$$1 - p(c(\vec{x}))$$

ie violation probability. In ARM this is:

$$p(X\bar{Y})$$

ie the *Ralambondrainy measure* R. We learn c if $R(c) \leq \rho$ where ρ is another user-defined threshold close to 0 (I use $\rho = 0.01$).

Results

On standard benchmarks (Sudoku, Latin square, Golomb rulers) and a new one (learning partial orders) MineAcq learned correct models more quickly than most methods, beaten only by SeqAcq and BayesAcq.

It often needs more instances than supervised methods. But this is quite common for unsupervised learning, and it's more than compensated for by the lack of reliance on labels.

Related approaches

Similar approaches to **MineAcq** have been applied to generating constraints for specific applications: *constraint mining*.

In *process mining* event logs are mined for useful information, and this has been used to learn constraints for scheduling problems.

A version of the Apriori algorithm has been designed to learn certain constraints for enterprise security management.

But these are not general approaches to unsupervised CA.

Some CA methods use positive-only data (Model Seeker, Tacle, Valiant's method), some can also use negative-only (QuAcq), most use mixed (Conacq, Matchmaker, SeqAcq), a few can use noisy data (SeqAcq, BayesAcq). MineAcq can use all these, and without labels.

Integrating CP and Data Mining has been an active area of research for several years, but most work has concentrated on the use of constraints in mining — MineAcq applies mining to CP.

Symbolic ML

Inductive logic programming & version space learning can be seen as a symbolic form of ML. Are there other forms that can be used for CA?

I found an application of symbolic classifiers. This work was presented at the IJCAI-DSO'22 workshop ("Extrapolating Constraint Networks by Symbolic Classification").

Most CA methods learn CSPs of a certain size from instances of that size.

A few can learn a CSP from solutions of other sizes, or learn a generalised CSP for all sizes: harder but more useful.

Model Seeker can extrapolate from input solutions of more than one size and learn a CSP of a different size. ILP can learn a generalised constraint model. (Problem size is not the only kind of parameter that can be generalised.)

A new approach to learning a CSP: "extrapolation" from learned (or hand-coded) CSPs of other sizes.

The training CSPs can be learned by any convenient CA method, so we can potentially handle *positive-only*, *negative-only*, *positive-negative*, *noisy* or *unlabelled* data.

Extrapolation by classification

Recall the $N \times N$ Latin square, a CSP with variables $v_{i,j} \in \{1, \ldots, N\}$ $(i, j = 1, \ldots, N)$ and disequality constraints $v_{i,j} \neq v_{i',j'}$ where $i = i'$ or $j = j'$.

Suppose we already have CSPs for $N = 2, 3, 5$. Can we learn a generator that, given a new value of N, outputs the correct constraints?

Bias for a fixed value of N: all disequalities $v_{i,j} \neq v_{i',j'}$ whose variable indices satisfy $(i,j) < (i',j')$.

The subset with either $i = i'$ or $j = j'$ are to be learned: call these class 1 and the rest class 0.

Why not train a classifier to learn how to distinguish between the two classes, using the classified bias as a training dataset?

We expect to be given similar information for several different problem sizes. So we include problem size, and any other relevant parameters that we know of, in the training instances.

Eg for Latin squares the training data are vectors of values $\langle N, i, j, i', j' \rangle$ where N is size and $v_{i,j} \neq v_{i',j'}$ is the candidate.

It will contain some class 1 examples such as $\langle 3, 0, 0, 1, 2 \rangle$ (because $i = i'$) and $\langle 3, 0, 2, 1, 1 \rangle$ (because $j = j'$), and class 0 examples such as $\langle 3, 0, 1, 1, 2 \rangle$ (which satisfy neither) — see paper for classified bias example.

We have 342 labelled instances for $N = 2, 3, 5$. Train a binary classifier to discriminate between classes 0 and 1. If no overfitting occurs then the trained classifier can be used to select the correct constraints from the bias of any Latin square size.

(For a constraint problem with more than one family of constraints — eg disequalities and inequalities — we treat the families separately, each with its own classified bias.)

This "bias classification" approach has the advantage that it transforms a problem with several different-sized constraint models into fixed-size classification problems, one per constraint type.

It works purely on constants and variable indices and performs no reasoning on constraint properties, so it can also be applied to SAT & ILP...

Degenerate cases

There are two special cases:

- class 1 empty, eg if we have a CA system containing a library of constraints (most will be irrelevant)

- class 0 empty, eg N-queens contains all possible disequalities

Both cases cause some classifiers to return an error message.

We test for these cases before applying a classifier: if all training instances are in class 1, or all in class 0, then we assume that the same will be true for all unseen instances.

So for any family of candidates, the result of training is either a memo that class 1 is empty, or a memo that class 0 is empty, or a trained classifier.

Which classifier?

Classification is a fundamental ML task that has recieved a great deal of attention, and a variety of methods is available. We trained several on Latin square data with sizes $N = 2, 3, 5$, then tested it on size 4 (interpolation) and all prime sizes up to 47 (extrapolation).

Results: surprisingly poor! As N increases, KNN and RF start to misclassify most candidates: by $N = 47$ they respectively misclassify 81% and 93% of all candidates. LIN, LR, SVM and MLP misclassify 4%–12%. No classifier even managed to interpolate to $N = 4$.

Only RF managed to fit the training data correctly — but it was the worst extrapolator so it overfitted.

Attempts at tuning numerical classifier hyperparameters, the SVM kernel (polynomial, sigmoid, radial basis function), the MLP architecture (number and size of hidden layers) and activation functions (sigmoid and RelU) yielded no significant improvement.

LIN and LR were the best extrapolators, perhaps because their simplicity caused less overfitting.

Why do such diverse and successful classifiers all fail hopelessly at this small binary classification task?

Conjecture: because the class 1 instances do not occupy a connected region of the feature space, or even one with a few clusters.

This looks bad for the bias classification approach. Fortunately, there is a solution...

Symbolic classification

We seem to need a classifier that can learn to separate classes 0 and 1 via simple mathematical relationships on variable indices etc. An interesting possibility is a *symbolic classifier* which learns a mathematical function to separate classes.

Symbolic regression is a well-known offshoot of GP and can be used to explain datasets, but symbolic classification has been relatively neglected. I use gplearn (Scikit-Learn, Google Colaboratory).

gplearn uses GP to evolve a function that fits the training data: class 1 is indicated by a positive output and class 0 by a negative.

gplearn worked perfectly on Latin squares! It even worked given only 1 training example ($N = 5$).

I call this method for CA by eXtrapolation *XAcq* (only a research prototype at present).

Testing on other examples...

Latin squares with global constraints

To be useful, XAcq should be able to handle global constraints such as alldifferent: eg for Latin squares, an alldifferent constraint on each row and column.

Handling high-arity global constraints is difficult for most CA methods because there are too many possibilities, and the bias becomes exponentially large. Model Seeker solves this by assuming that there is a lot of regularity. We similarly assume that alldifferent constraints will only be applied to rows and columns.

Latin square extrapolation then becomes degenerate because *all* row and columns are constrained.

N-queens

We use a CSP with variables $v_i \in \{1, \ldots, N\}$ $(i = 1, \ldots, N)$. Row constraints are implicit in the model, and the column constraints (disequalities between variables) are a degenerate case with empty class 0. So we need only use the symbolic classifier to extrapolate the diagonal constraints $|v_i - v_j| \neq |i - j|$.

XAcq successfully learns ($N = 2, 3, 5$ again) and extrapolates to bigger N (up to 47).

Golomb rulers

Suppose that specific CSPs have been learned for several sizes, containing disequalities and quaternary constraints of the form $|x_i - x_j| \neq |x_{i'} - x_{j'}|$ ($i < j$, $i' < j'$, $i < i'$, $j' \neq k$, $k \neq k'$).

All disequalities and *all* quaternary constraints are learned in each example. So both are degenerate cases with empty class 0.

Learning fixed-size Golomb ruler CSPs has been found quite challenging, but their extrapolation is easy.

Magic squares

A magic square is a square of integers in the range $1, \ldots, N$ with no two integers the same, and with each row, column and main diagonal summing to the same number $M = N(N^2 + 1)/2$.

CSP for order N has variables $v_{i,j} \in \{1, \ldots, N^2\}$ ($1 \leq i, j \leq N$). They all take different values, and each row, column and diagonal sums to the *magic constant* M which is a function of the square size.

Suppose we have magic squares for $N = 3, 4, 5$ each with a learned CSP. Regularity assumption: only sums on rows, columns and diagonals are constrained.

I applied XAcq to rowsum constraints: colsum extrapolation problem is identical, and diagsum extrapolation is degenerate. It extrapolated correctly.

BIBDs

5 values V, B, R, K, λ define the problem, but they are not independent and it is common to specify only V, K, λ. We can then calculate $R = \lambda(V-1)/(K-1)$ and $B = VR/K$.

Suppose we have learned CSPs for several instances, but that we know nothing about BIBDs: we know V, B because they are observable, but have not considered R, K, λ and do not know their importance.

Then extrapolation will fail because the problem is underspecified: no function exists that can predict R, K, λ from V, B alone.

But if the ZAcq user notices that all row sum constraints use the same value R, that the column constraints all use the same value K, and that all dot product constraints use the same value λ, s/he can add these parameters as training data columns. If they are not useful, we hope that gplearn will ignore them.

After training on parameters (6,10,5,3,2), (7,7,3,3,1), (8,14,7,4,3), (9,12,4,3,1) and (10,15,6,4,2), XAcq extrapolates correctly to (7,14,6,3,2), (9,24,8,3,2), (10,15,6,4,2), (13,26,6,3,1) and (15,35,77,3).

Note

The extrapolated BIBD models tell us nothing about the relationships between the 5 parameters: they allow us to generate a CSP for any combination of values, even (1,1,1,10,10) representing a nonsensical case with more 1s per row than there are numbers per row.

But XAcq is not a theorem prover so this seems reasonable.

An advantage: combining MineAcq with XAcq yields the first known approach to learning CSPs of an unseen size from unlabelled data.

A disadvantage: each specific CSP requires sufficient instances for CA, whereas Model Seeker can learn from a very small number of instances.

(Model Seeker aims to generate efficient and non-redundant CSPs while XAcq and most other methods do not.)

Conclusion & future work

ML is a rich research area with many ideas that can be applied to CA.

Can reinforcement learning be used for CA? Gene suggested this but I can't see a way.

Transfer learning? Model Seeker already does something like this, but using human expertise instead of ML.

I'm working with Gregory Provan & Samaksh Chandra on CA as seq2seq learning via transformers.

Notes on Model Seeker

In some ways it's still the SOTA!

- Often requires only a few solutions (or just 1).
- It's been tested on hundreds of problems.
- It can handle problems that are larger than usual in the literature, some with thousands of variables.

- It can learn high-level models containing a wide range of global constraints. These can not be handled by most methods: exponentially large bias.

- It's fast, often taking only seconds and at most a few minutes.

But it has limitations:

- It assumes that the desired constraint model has a regular structure that can be represented as a conjunctions of sets of identical constraints.

- It applies heuristics to exclude atypical usages of global constraints, to use stronger and more popular constraints, to eliminate redundant and dominated constraints, and to avoid trivial and uninteresting constraints. This might break down on less structured or on atypical applications.

- It requires human expertise to provide necessary information, which might turn out to be difficult on new applications.

It might be viewed as an expert system approach, with similar *brittleness*.

If ML-based approaches can catch up with Model Seeker, the results could be impressive!