CORA Cork Open Research Archive
Cartlann Taighde Oscailte Chorcaí

Title	Extrapolating constraint networks by symbolic classification
Authors	Prestwich, Steven D.
Publication date	2022-07
Original Citation	Prestwich, S. D. (2022) 'Extrapolating Constraint Networks by Symbolic Classification ', IJCAI 2022 DSO Workshop: Data science meets optimisation, Messe Wien, Vienna, Austria, 23-29 July, forthcoming publication.
Type of publication	Conference item
Rights	© 2022 IJCAI
Download date	2023-09-26 01:19:56
Item downloaded from	https://hdl.handle.net/10468/13288

UCC
University College Cork, Ireland
Coláiste na hOllscoile Corcaigh

Extrapolating Constraint Networks by Symbolic Classification

1st Steven Prestwich
Dept of Computer Science, University College Cork, Ireland
Email: s.prestwich@cs.ucc.ie

***Abstract*—Constraint acquisition (CA) methods aim to learn constraint satisfaction problems (CSPs) from data, thus automating the difficult and error-prone task of constraint modelling. Most CA methods learn CSPs of a certain size from examples of that size, for example an 8-queens CSP from 8-queens solutions. A few methods can learn a CSP from solutions of other sizes, or learn a generalised CSP for all sizes, which is more challenging but also more useful in practice. This paper describes a new approach to learning a CSP: extrapolation from learned CSPs of other sizes. This has the advantage that the training CSPs can be learned by any convenient CA method, thus potentially handling positive-only, negative-only, positive-negative, noisy or unlabelled data. We model the problem as classification, and show that a symbolic classifier based on genetic programming outperforms alternatives such as random forests and deep learning.**

I. Introduction

Constraint Programming provides powerful modeling languages and solvers for constrained optimisation and decision making problems, and has many applications. A constraint satisfaction problem (CSP) consists of a set of variables, each with a domain of possible values, and a network of constraints each defined on a subset of the variables. Modelling a new application as a CSP requires expertise, which can impede the uptake of Constraint Programming. This has motivated the development of Constraint Acquisition (CA) in which constraints are learned from known solutions and (often) non-solutions.

CA has been developed over approximately 20 years and a variety of approaches have been explored, several based on some form of machine learning. From a set of possible constraints (*candidates*) called the *bias* we must select which are to be learned, based on training data containing examples of solutions and (usually) non-solutions. CA has been identified as an important topic by [14] and as progress toward the "Holy Grail" of computer science: the user presents a problem to the computer in a manner natural, and the computer solves it [8], [9]. A recent survey of the CA field is given in [22].

Most CA methods only learn CSPs of a fixed size from instances of that size [1], [2], [4], [5], [7], [10], [13]–[15], [17]–[21], [25], [27], [28]. For example they learn constraints for the 8-queens problem from a dataset of 8-queens solutions (and possibly non-solutions depending on the method). To the best of our knowledge only the pioneering Model Seeker [3] method can take as input solutions of more than one size (for examples 3-, 4- and 5-queens) and learn a CSP of a different size (8-queens). Another approach is to use Inductive Logic Programming to learn a generalised constraint model [12] (for example for N-queens). This generalised form CA has wider applicability than the usual fixed-size methods but is also a more challenging problem. (Problem size is not the only kind of parameter that can be generalised, as discussed in Sections III-D and III-E.)

We propose a new approach to this generalised form of CA: learning how to generate CSPs of any size from learned CSPs of other sizes. Typically we would provide constraints for small instances and extrapolate to larger instances, so we shall refer to this approach as *extrapolation*. This is distinct from the Model Seeker approach: for example, we might learn an 8-queens CSP from sets of disequalities for 3-, 4- and 5-queens. We describe a machine learning-based approach to the extrapolation problem, and show that a symbolic approach is superior to the more common subsymbolic methods.

The paper is organised as follows. Section II describes the method, Section III tests it on several benchmarks, and Section IV summarises the contribution and discusses future work.

II. CSP extrapolation by bias classification

We propose a classification-based approach. Classifiers can be used as the basis of fixed-size CA methods [19] and we shall show that they can also be applied to extrapolation.

A. Running example: Latin squares

As an example consider the Latin square, which has been used before as a CA benchmark [1], [17]–[19], [25]. An $N \times N$ Latin square is a square of integers in which each row and column is a permutation of the integers 1–N. As a CSP it can be modelled by 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 sizes $N = 2, 3, 5$. We aim to learn a generator that, given a new value of N, can output the correct constraints. A bias for a fixed value of N contains the set of all disequalities $v_{i,j} \neq v_{i',j'}$ whose variable indices satisfy $(i, j) < (i', j')$ (that is $i < i'$ or $i = i'$ and $j < j'$ to avoid redundancy). Of these candidates, the subset with either $i = i'$ or $j = j'$ are to be learned: we shall call these class 1 and the remainder class 0. In principle we can train a classifier to learn how to distinguish between the two classes, using the classified bias as a training dataset.

However, we expect to be given similar information for several different problem sizes, and for each size the problem will look different (though hopefully with a common pattern

to be learned). We therefore include problem size, and any other relevant parameters that we know of, in the training examples. For Latin squares the training data are vectors of values $\langle N, i, j, i', j' \rangle$ where N is the Latin square size and $v_{i,j} \neq v_{i',j'}$ is the candidate. The training dataset 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 condition).

In our Latin square example, the classified bias for N=2 is as follows. Class 1 (the candidates to be learned) contains:

$\langle 2,1,1,1,2 \rangle$	$\langle 2,1,2,2,2 \rangle$	$\langle 2,1,1,2,1 \rangle$	$\langle 2,1,2,2,2 \rangle$

while class 0 (not to be learned) contains:

$\langle 2,1,1,2,2 \rangle$	$\langle 2,1,2,2,1 \rangle$

Similarly for N=3. Class 1 contains:

$\langle 3,1,1,1,2 \rangle$	$\langle 3,2,1,2,2 \rangle$	$\langle 3,3,1,3,2 \rangle$
$\langle 3,1,1,1,3 \rangle$	$\langle 3,2,1,2,3 \rangle$	$\langle 3,3,1,3,3 \rangle$
$\langle 3,1,2,1,3 \rangle$	$\langle 3,2,2,2,3 \rangle$	$\langle 3,3,2,3,3 \rangle$
$\langle 3,1,1,2,1 \rangle$	$\langle 3,1,2,2,2 \rangle$	$\langle 3,1,3,2,3 \rangle$
$\langle 3,1,1,3,1 \rangle$	$\langle 3,1,2,3,2 \rangle$	$\langle 3,1,3,3,3 \rangle$
$\langle 3,2,1,3,1 \rangle$	$\langle 3,2,2,3,2 \rangle$	$\langle 3,2,3,3,3 \rangle$

while class 0 contains:

$\langle 3,1,1,2,2 \rangle$	$\langle 3,1,2,2,1 \rangle$	$\langle 3,1,3,2,1 \rangle$
$\langle 3,1,1,2,3 \rangle$	$\langle 3,1,2,2,3 \rangle$	$\langle 3,1,3,2,2 \rangle$
$\langle 3,1,1,3,2 \rangle$	$\langle 3,1,2,3,1 \rangle$	$\langle 3,1,3,3,1 \rangle$
$\langle 3,1,1,3,3 \rangle$	$\langle 3,1,2,3,3 \rangle$	$\langle 3,1,3,3,2 \rangle$
$\langle 3,2,1,3,2 \rangle$	$\langle 3,2,2,3,1 \rangle$	$\langle 3,2,3,3,1 \rangle$
$\langle 3,2,1,3,3 \rangle$	$\langle 3,2,2,3,3 \rangle$	$\langle 3,2,3,3,2 \rangle$

We combine these classified biases to obtain $4 + 18 = 22$ class 1 examples and $2 + 18 = 20$ class 0 examples, plus 300 examples for $N = 5$ (not shown) giving 342 examples in total. We then 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 size of Latin square. For example it should be able to recognise that $\langle 8, 2, 3, 5, 3 \rangle$ is in class 1 while $\langle 8, 2, 3, 5, 4 \rangle$ is in class 0.

For a constraint problem with more than one family of constraints — for example 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. As it works purely on constants (N) and variable indices (i, j, i', j') and performs no reasoning on constraint properties, it can also be applied to other formalisms such as SAT and ILP.

B. Degenerate cases

There are two special cases that we shall refer to as *degenerate*. Firstly, the training data class 1 might be empty. This can occur if we have a CA system containing a library of constraint types, all of which are tested: for any given dataset, most types of constraint will probably be irrelevant. Secondly, the training data class 0 might be empty. An example is shown below, where a constraint model for N-queens contains all possible disequalities so all $\neq$ candidates are in class 1.

Both degenerate cases will cause some classifiers to return an error message, as classifiers typically require examples of both classes to be provided. We therefore test for these cases before applying a classifier: if all training examples are in class 1, or all in class 0, then we assume that the same will be true for all unseen examples. 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.

In the testing phase, given a new candidate (such as $\langle 8, 2, 3, 5, 3 \rangle$) in a family of constraints (such as disequalities): if the result of training for that family is a memo then we either accept the candidate (if class 0 was empty) or reject it (if class 1 was empty); otherwise we apply the trained classifier to decide.

C. Classifier experiments

Which classifier should be used for extrapolation? Classification is a fundamental machine learning task that has recieved a great deal of attention in the literature, and a variety of methods is available. We experimented with some standard classifiers: linear (LIN), a support vector machine (SVM) with polynomial kernel, a random forest (RF), K-nearest neighbours (KNN), a multi-layer perceptron (MLP) and logistic regression (LR). All classifier hyperparameters use Scikit-Learn default values. We trained each classifier on Latin squares data with sizes $N = 2, 3, 5$ as described above, then tested it on size 4 as a small interpolation experiment, and all prime sizes up to 47.

The results are shown in Table I and are 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 only misclassify 4%–12% but this is still unacceptable. We expected at least a successful interpolation for $N = 4$ but no classifier managed this. Worse, only RF managed to fit the training data correctly — but it was the worst extrapolator, so its training success was probably a case of overfitting.

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. Neural networks are universal function approximators so MLP might be successful after more tuning, but so far we have been unable to achieve this.

Why do such diverse and successful classifiers all fail hopelessly at this small binary classification task? We conjecture that the failure to fit the training data is because the class 1 examples do not occupy a connected region of the feature space, or even one with a few clusters. But even if this problem could be solved, we might find poor testing results because the

N	number of candidates	number of misclassified candidates					
		LIN	SVM	KNN	RF	MLP	LR
2	6	1	2	1	0	1	1
3	36	15	14	6	0	4	12
4	120	33	20	16	5	16	46
5	300	62	40	30	0	37	105
7	1,176	183	148	243	234	151	313
11	7,260	962	846	3,099	3,738	961	1,261
13	14,196	1,736	1,551	7,127	8,640	1850	2,103
17	41,616	4,244	4,486	25,177	30,264	5399	4,760
19	64,980	6,074	6,879	41,629	49,770	8375	6,669
23	139,656	11,062	14,569	96,778	114,570	17809	11,891
29	353,220	22,767	36,125	262,432	307,200	44442	23,954
31	461,280	28,014	46,986	348,273	406,458	57819	29,295
37	936,396	48,312	94,669	732,976	849,024	116102	49,950
41	1,412,040	66,164	142,335	1,124,493	1,297,368	174,100	68,060
43	1,708,476	76,470	172,129	1,370,409	1,578,330	210,083	78,561
47	2,438,736	100,103	245,379	1,980,341	2,273,754	298,519	102,695

TABLE I
CLASSIFIER RESULTS ON LATIN SQUARES

classifiers are not limited to integral functions, which are key in our application. This failure is discouraging for our bias classification approach. Fortunately, there is a solution.

D. Symbolic classification

We seem to need a classifier that can learn to separate classes 0 and 1 via simple mathematical relationships. An interesting possibility is a *symbolic classifier* in which a mathematical function is learned that can be used to separate classes. The related task of *symbolic regression* is a well-known offshoot of genetic programming (the use of genetic algorithms to evolve functions and programs) and can be used to explain datasets and even discover laws of physics [26], but symbolic classification has been relatively neglected [11].

Several symbolic classifiers are available, and we use one called gplearn [24] implemented in Scikit-Learn [16] via Google Colaboratory (Colab) [6]. gplearn performs symbolic classification through symbolic regression: it uses genetic programming to evolve a function that fits the training data in the sense that class 1 is indicated by a positive output and class 0 by a negative, by minimising log loss. (How a zero value is classified is unclear from the documentation.)

A symbolic classifier can in principle learn the Latin square pattern. For example it could learn the function

$$1 - 2|i - i'||j - j'|$$

which is positive if $i = i'$ or $j = j'$, otherwise negative. However, gplearn needs some tailoring to our application. It introduces real-valued constants which we do not want, as fitting a real-valued function to fundamentally integral data is likely to extrapolate poorly. We avoid this by preventing it from introducing any constants at all (setting hyperparameter const_range to None). This does not prevent it from learning the Latin square pattern because it can simulate the effect of small integers. For example it is easily verified that this function:

$$N - N|i - i'||j - j'| - N|i - i'||j - j'|$$

is also positive if and only if $i = i'$ or $j = j'$, though it contains no constants other than N which was provided in the data. We also forbid the use of operators that might introduce non-integers: division, $\sqrt{}$, log, sin, cos, tan, inverse. However, we allow the other operators provided by gplearn: $+$, $-$, $\times$, min, max, neg(ation) and abs(olute value).

Of course, the existence of a function that solves our problem does not guarantee that gplearn can discover it. Moreover, it is possible to devise functions that fit the training examples but do not extrapolate well, such as

$$N - N|i - i'||j - j'| - N|i - i'||j - j'| + N(N - \min(N, 6))$$

which gives identical results up to $N = 5$ but all-positive results for $N \geq 6$. We might hope that gplearn's bias toward simpler functions (controlled by the parsimony_coefficient hyperparameter) would avoid such a function, but there are no guarantees. However, this seems to happen rarely.

We used mainly default hyperparameter settings for gplearn, but tuned parsimony_coefficient to 0.0003, population_size to 5,000, class_weight to "balanced" (class sizes might vary greatly and this setting automates the choice of class weights) and stopping_criteria to 0.0001 (this is a metric value used to stop evolution when the results are accurate enough).

The classifier worked perfectly on our Latin square example, and even worked when given only one training example ($N = 5$): all candidates for all N are correctly classified. Despite gplearn's bias toward simpler functions it learned the rather complicated function shown in Figure 1 but did not overfit the training data. The function can be simplified and analysed if we wish to understand the generated models, which might be useful for an Explainable Artificial Intelligence approach, but we leave this for future work.

We call this method for CA by eXtrapolation *XAcq*. Currently XAcq is only a research prototype and no implemented system is available.

```
sub(add(sub(add(abs(N),abs(N)),abs(mul(mul(abs(neg(abs(neg(min(abs(sub(j,i)),
abs(sub(i',j')))))))),add(neg(add(abs(N),abs(N))),neg(abs(neg(i))))),abs(N)))),
max(abs(N),sub(j',j))),abs(mul(mul(abs(min(abs(sub(j,i)),abs(sub(i',j'))))),
add(neg(max(sub(j,i),j'),neg(abs(N)))),abs(N))))
```

Fig. 1. Symbolic classifier function learned for Latin square disequalities

III. Experiments

We already showed that XAcq works for a Latin square CSP with disequalities so we now try other problems. Runtimes are not shown because the classifier runs on Colab which uses unknown remote platforms. But as a rough guide, all runtimes are between a few seconds and five minutes.

A. Latin squares with global constraints

To be useful, XAcq should be able to handle global constraints such as alldifferent: for example a more efficient CSP for Latin squares posts 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 problem by making simplifying assumptions about the CSP: that there is a great deal of regularity because (for example) it is a matrix model. We can make similar assumptions when we construct the bias, by assuming that alldifferent constraints will only be applied to rows and columns, not to arbitrary subsets of the variables. The Latin square extrapolation problem then becomes degenerate because *all* row and columns are constrained. We discuss more interesting high-arity constraints below.

B. N-queens

We use a CSP with variables $v_i \in \{1, \ldots, N\}$ ($i = 1, \ldots, N$). The row constraints are implicit in the model (with one finite domain variable per row), 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 $|v_i - v_j| \neq |i - j|$.

The training examples have the form $\langle N, i, j, k \rangle$ where i, j denote the variable indices involved in the candidate, and k is the number on the right hand side of the candidate:

$$|v_i - v_j| \neq k$$

We include in the bias all possible k in the range $1, \ldots, N$ and it will be up to the classifier to infer which of these should be learned: those for which $k = |i - j|$. Actually, because we enumerate the bias in such a way that $j < i$ (it would be redundant to test both $|v_i - v_j| \neq k$ and $|v_j - v_i| \neq k$) this reduces to $k = i - j$.

XAcq successfully learns the training examples and extrapolates correctly to bigger N (tested on all prime N from 7 to 47).

C. Golomb rulers

A Golomb ruler is a set of N marks at integer positions along an imaginary ruler such that no two pairs of marks are the same distance apart. The smallest number is 0 and the largest is the ruler length L. The $N = 12$ case has been used several times for CA [1], [2], [5], [17]–[19].

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

It turns out that *all* disequalities and *all* quaternary constraints are learned in each example. So both are degenerate cases with empty class 0, and XAcq simply learns their entire biases. Learning the fixed-size Golomb ruler CSPs has been found quite challenging in the literature, but their extrapolation is easy.

D. 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 $M = N(N^2 + 1)/2$. A CSP for a magic square of 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. This was used as a CA benchmark in [3].

Suppose we have magic square examples $N = 3, 4, 5$ each with a learned CSP, which for $N = 3$ is of the form:

rowsum(1,15)	rowsum(2,15)	rowsum(3,15)
colsum(1,15)	colsum(2,15)	colsum(3,15)
diagsum(1,15)	diagsum(2,15)	

Note that we are making a Model Seeker-like regularity assumption: that sums on arbitrary subsets of the data need not be in the bias, only sums on rows, columns and diagonals. Note also that we do not assume any particular syntax: rowsum(A,B) represents the constraint that the sum of the row A variables is B. Similarly for colsum and diagsum. For $N = 4\,(5)$ there are 4 (5) rowsum and colsum constraints, and M is 34 (65).

For this experiment we consider only the rowsum constraints: the colsum extrapolation problem is identical while diagsum extrapolation is degenerate. The training vectors are of the form $\langle N, i, m \rangle$ ($n = 3, 4, 5$, $i = 1, \ldots, n$ and $m = 1, \ldots, 70$) so only a few of the candidates are to be learned (those with the correct M value). The upper bound of 70 on m was chosen as a sufficiently large number.

XAcq correctly extrapolated to the larger cases ($N = 6, M = 111$), ($N = 7, M = 175$), ($N = 8, M = 260$) and ($N = 9, M = 369$) given an upper bound of 400 on M.

E. BIBDs

This problem was used by [3] but is less popular as a CA benchmark, and it poses a interesting challenge to XAcq.[1] This comes from the combinatorial problem of finding *balanced incomplete block designs* (BIBDs). For our purposes we can think of the BIBD *incidence matrix*, which is a binary matrix with V rows and B columns, and some constraints: each row sums to R, each column sums to K, and the dot product of each pair of rows is λ. The 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 $B = VR/K$. Unless these relationships (and an extra inequality) hold there is no solution to the problem.

Suppose we have learned CSPs for several instances of this problem. Suppose also 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. In this case extrapolation will fail because the problem is underspecified: no function exists that can predict R, K, λ from V, B alone. This is unlike the magic square problem in which M is a function of N. Although we are aiming for a passive method (with no user interaction) we appear to need some human insight here.

However, unlike in the Model Seeker approach we have available some learned CSPs. It is not unreasonable to expect the user of our hypothetical system to notice certain regularities in these CSPs: 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. Moreover, it might be noticed that the R, K, λ values are different in the given CSPs. The user need not understand the relationships between these numbers, or the mathematics of block designs, to suspect that this might be more than a coincidence and that R, K, λ might be relevant parameters. This reasoning might even be automated: find numbers that are repeated within each specific CSP but differ between CSPs, and make them parameters.

Following this reasoning, the user would include the observed R, K, λ values in the training examples in case they are useful for extrapolation (if they are not then we hope that the classifier will learn to ignore them). So instead of $\langle V, B \rangle$ the training examples will take the form $\langle V, B, R, K, \lambda \rangle$ and extrapolation should be possible.

We performed an experiment to test this. Considering only the part of the bias involving row sum constraints, we created CSPs with 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). Again we make a simplifying assumption of regularity, by including in the bias only constraints on columns, rows and pairs of rows. From this training data XAcq extrapolates correctly to CSPs for (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). The pattern to be learned for extrapolation is simple: all row constraints sum to the input value R. The column sum and dot product constraints are equally simple to extrapolate. In summary, we can extrapolate a few specific CSPs for BIBDs and generate CSPs for any new parameter values. We needed only to observe some common values in the CSPs and use them in the training data.

Unfortunately our extrapolated 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 designed to discover such information and is not a theorem prover. An attempt to learn this kind of information was made by [23] but they found that there are many spurious solutions that hide the correct one.

IV. Conclusion

Most CA methods learn fixed-size CSPs from instances of the same size. CSP extrapolation is a new approach to learning CSPs of unseen sizes, and we described a method called XAcq based on symbolic classification by genetic programming. Some CA methods [12] use Inductive Logic Programming, which is a different form of symbolic machine learning based on first order logic. XAcq differs from the Model Seeker [3] approach, which learns CSPs of one size from solutions of other sizes. Instead it extrapolates from existing fixed-size CSPs that can be learned using any convenient CA method. Under simplifying assumptions of CSP regularity, it works on high-arity global constraints as well as simple low-arity constraints. It performs no reasoning on constraints, and simply learns patterns among constants and variable indices. For this reason it should also work for SAT, MIP and other optimisation and decision-making technologies.

The only other work we know of that tries to learn integer functions for Constraint Programming is that of [23], which uses a global constraint to learn simple polynomials. XAcq instead uses symbolic machine learning to learn more general functions, making it more flexible but also less tractable, and perhaps more prone to overfitting.

XAcq has both advantages and disadvantages with respect to the Model Seeker approach. An advantage is that XAcq can learn the training CSPs using any convenient fixed-size CA method. This provides considerable flexibility as there are methods that can learn from positive-only data (Valiant's method for SAT [27], Model Seeker [3]), negative-only data (QuAcq [4], MineAcq [18]) positive-negative data (ConAcq [5] and many others), unlabelled data (MineAcq [18]) and noisy data (BayesAcq [19], SeqAcq [17], MineAcq [18]). All these methods, or a human modeller, could be used to provide CSPs to XAcq. In particular, combining MineAcq with XAcq yields the first known approach to learning CSPs of an unseen size from unlabelled data. A disadvantage of XAcq is that each specific CSP requires sufficient instances for learning,

[1] Thanks to Helmut Simonis for proposing this problem and magic squares.

whereas Model Seeker can learn from a very small number of instances. Model Seeker also aims to generate efficient and non-redundant CSPs while XAcq does not.

ACKNOWLEDGEMENTS

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

REFERENCES

[1] H. A. Addi, C. Bessiere, R. Ezzahir, and N. Lazaar. Time-bounded query generator for constraint acquisition. In *15th International Conference on the Integration of Constraint Programming, Artificial Intelligence, and Operations Research, Lecture Notes in Computer Science vol. 10848*, pages 1–17, 2018.

[2] R. Arcangioli, C. Bessiere, and N. Lazaar. Multiple constraint acquisition. In *25th International Joint Conference on Artificial Intelligence*, 2016.

[3] N. Beldiceanu and H. Simonis. Modelseeker: Extracting global constraint models from positive examples. In *Data Mining and Constraint Programming, Lecture Notes in Computer Science vol. 10101, Springer*, pages 77–95, 2016.

[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 *Proceedings of the Twenty-Third International Joint Conference on Artificial Intelligence*, IJCAI '13, page 475–481. AAAI Press, 2013.

[5] C. Bessiere, R. Coletta, F. Koriche, and B. O'Sullivan. Constraint Acquisition. *Artificial Intelligence*, 244:315–342, 2017.

[6] E. Bisong. Google colaboratory. Apress, Berkeley, CA, 2019. In: Building Machine Learning and Deep Learning Models on Google Cloud Platform, https://doi.org/10.1007/978-1-4842-4470-8_7.

[7] A. Bonfietti, M. Lombardi, and M. Milano. Embedding decision trees and random forests in constraint programming. In *12th International Conference on AI and OR Techniques in Constraint Programming for Combinatorial Optimization Problems, Lecture Notes in Computer Science vol. 9075*, pages 74–90, 2015.

[8] E. C. Freuder. Constraints: The ties that bind. In *21st National Conference on Artificial Intelligence, AAAI Press*, pages 1520–1523, 2006.

[9] E. C. Freuder. Progress Towards the Holy Grail. *Constraints*, 23:158–171, 2018.

[10] S. Kolb, S. Paramonov, T. Guns, and L. De Raedt. Learning Constraints in Spreadsheets and Tabular Data. *Machine Learning*, 106:1441–1468, 2017.

[11] M. F. Korns. Genetic programming symbolic classification: A study. In *GPTP*, 2017.

[12] A. Lallouet, M. Lopez, L. Martin, and C. Vrain. On learning constraint problems. In *Proceedings of the IEEE International Conference on Tools With Artificial Intelligence*, page 45–52, 2010.

[13] M. Lombardi, M. Milano, and A. Bartolini. Empirical Decision Model Learning. *Artificial Intelligence*, 244(Supplement C):343–367, 2017.

[14] B. O'Sullivan. Automated modelling and solving in constraint programming. In *24th AAAI Conference on Artificial Intelligence*, pages 1493–1497, 2010.

[15] T. P. Pawlak and K. Krawiec. Automatic Synthesis of Constraints from Examples Using Mixed Integer Linear Programming. *European Journal of Operational Research*, 261(3):1141–1157, 2017.

[16] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay. Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, 12:2825–2830, 2011.

[17] S. D. Prestwich. Robust constraint acquisition by sequential analysis. In *24th European Conference on Artificial Intelligence, Frontiers in Artificial Intelligence and Applications vol 325, IOS Press*, pages 355–362, 2020.

[18] S. D. Prestwich. Unsupervised constraint acquisition. In *33rd International Conference on Tools with Artificial Intelligence*, 2021.

[19] S. D. Prestwich, E. C. Freuder, B. O'Sullivan, and D. Browne. Classifier-based constraint acquisition. *Annals of Mathematics and Artificial Intelligence*, 89:655–674, 2021.

[20] L. De Raedt and L. Dehaspe. Clausal Discovery. *Machine Learning*, 26:99–146, 1997.

[21] L. De Raedt and S. Džeroski. First Order jk-clausal Theories are PAC-Learnable. *Artificial Intelligence*, 70:375–392, 1994.

[22] L. De Raedt, A. Passerini, and S. Teso. Learning constraints from examples. In *32nd AAAI Conference on Artificial Intelligence*, pages 7965–7970, 2018.

[23] 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 *In M. Helmert and G. Röger, eds, Proceedings of the 6th Annual Symposium on Combinatorial Search, Leavenworth, Washington, USA*. AAAI Press, July 11-13, 2013.

[24] T. Stephens. gplearn.

[25] D. C. Tsouros, K. Stergiou, and P. G. Sarigiannidis. Efficient methods for constraint acquisition. In *24th International Conference on Principles and Practice of Constraint Programming, Lecture Notes in Computer Science vol. 11008*, pages 373–388, 2018.

[26] S.-M. Udrescu and M. Tegmark. Ai feynman: A physics-inspired method for symbolic regression. *Science Advances*, 6(16), 2020. eaay2631.

[27] L. G. Valiant. A Theory of the Learnable. *Communications of the ACM*, 27(11):1134–1142, 1984.

[28] X.-H. Vu and B. O'Sullivan. A Unifying Framework for Generalized Constraint Acquisition. *International Journal on Artificial Intelligence Tools*, 17(5):803–833, 2008.