Learning Compact Representations of Constraint Networks

Christian Bessiere, Clément Carbonnel and Areski Himeur

University of Montpellier, CNRS, LIRMM, Montpellier, France

Abstract. Passive constraint acquisition aims to learn constraint networks from examples of solutions and non-solutions. There typically exist many constraint networks that are consistent with a given set of examples, so the performance of an acquisition system is critically dependent on its ability to determine which network will generalize the best to unseen data. We introduce a framework for representing constraint networks in compressed form and present a novel method for constraint acquisition. Our method learns a constraint network that achieves a high compression ratio, with the idea that such networks are highly structured and therefore less prone to overfitting. Experiments demonstrate that this approach significantly reduces the number of examples needed for training and achieves a high accuracy on unseen data.

1 Introduction

Constraint programming (CP) is a powerful paradigm for solving combinatorial problems. A significant bottleneck in the use of CP is the modeling process itself: translating a problem into a constraint network typically requires expertise in both the problem domain and CP. Constraint acquisition aims to alleviate this limitation by automatically learning a model. This paper focuses on passive constraint acquisition, where the input of the learning process is a set of solutions and non-solutions to the problem.

Several approaches exist for passive acquisition. Given examples (solutions and non-solutions) and a set of candidate constraints, CONACQ.1 computes a SAT formula representing all the constraint models consistent with the examples [2, 3]. BAYESACQ takes as input a set of examples and a set of candidate constraints and follows a statistical approach to identify for each candidate constraint whether or not to include it in the output model [11]. LANGUAGE-FREE ACQ learns from examples without a set of candidate constraints [4]. It computes a suitable constraint language and a consistent network as part of the learning process. MODELSEEKER identifies relations from the global constraints catalog and tries to apply them to common structures while being consistent with the given examples [1]. The method presented in [9] produces a set of linear, quadratic and trigonometric constraints that minimizes the number of terms involved and is consistent with given examples.

A fundamental challenge is that given a set of examples, there often exist many possible networks that are consistent with them. Then an important question is: which network will generalize best to unseen data, avoiding overfitting to the specific training examples? This paper proposes that learning compact representations of constraint networks, rather than unstructured lists of constraints, is key to better generalization. We hypothesize that compact models can capture the underlying structure of the problem more effectively than simply fitting the provided examples with a classic constraint network, making them less prone to overfitting.

We first introduce a novel representation that can capture some structured constraint networks, which we call templates. A template associates each variable with a set of attributes and contains a set of rules, which can be understood as mechanisms for generating constraints based on these attributes. Importantly, each individual rule has the potential to generate many constraints at once. We then describe a framework that learns a template directly from a set of examples (rather than an explicit constraint network) with a two-step acquisition pipeline. First, a baseline acquisition method is used to learn an initial network that is consistent with the training set but may contain a large number of constraints. In the second step, this network is refined using a heuristic of our design that computes a *small* template (i.e. with few rules and attributes) capable of generating a *large* sub-network that is consistent with the training set. We run experiments on a wide array of benchmarks and show that combining the passive acquisition algorithm LANGUAGE-FREE ACQ with our framework yields constraint networks that are more interpretable and achieve significantly higher accuracy on unseen examples.

The notion of templates is conceptually similar to several intermediate representations used in the literature, such as the rule-based language learned using Inductive Logic Programming in [8], first-order constraints in COUNT-CP [7], and constraint specifications in GENCON [13]. However, a key distinction is that templates encode all variable attributes and constraint-generating rules in the same mathematical object. This property is critical in our setting because we learn attributes and rules simultaneously from raw examples, using the specific structure of templates to formulate this joint learning problem as a series of CP optimization models.

The rest of the paper is organized as follows. Section 2 provides background definitions. Section 3 introduces templates as a means to represent constraint networks concisely. Section 4 describes our algorithm for learning templates. Section 5 details the CP models used within our learning algorithm. Section 6 presents our experimental evaluation. Finally, Section 7 concludes and discusses perspectives.

2 Background

Constraint programming consists in expressing a problem as a constraint network and finding solutions, that is, assignments of values to all the variables so that no constraint is violated. A *vocabulary* is a pair (X, D), where X is a finite set of variables and D is a finite domain. Given a vocabulary (X, D), a *constraint* is a pair (R, S), where R is a relation of arity r over D (that is, R is a subset of

D^r) and S is a sequence of r variables (called the *scope* of the constraint). An assignment $A : X \to D$ *satisfies* a constraint (R, S) if $A[S] \in R$; otherwise, the assignment *violates* the constraint.

Definition 1 (Constraint network)**.** *A constraint network is a tuple $N = (X, D, C)$, where (X, D) is a vocabulary and C is a set of constraints. An assignment $A : X \to D$ satisfies N if A satisfies all constraints in C.*

Given two constraint networks $N_1 = (X, D, C_1)$ and $N_2 = (X, D, C_2)$ over the same vocabulary, we write $N_1 \subseteq N_2$ if and only if $C_1 \subseteq C_2$. A *constraint language* Γ is a set of relations over a finite domain. The arity of Γ is the maximum arity over all relations in Γ. A constraint network N is over a constraint language Γ if the relation R of each constraint of N is such that $R \in \Gamma$.

Given a vocabulary (X, D), an *example* on this vocabulary is a pair $e = (a(e), b(e))$, where $a(e)$ is an assignment, and $b(e)$ is a Boolean. We say that e is a *positive example* if $b(e)$ is $true$; otherwise e is a *negative example*. We say that a constraint network N *accepts* (resp. *rejects*) an example e if $a(e)$ satisfies (resp. does not satisfy) N. A constraint network N is *consistent* with a positive (resp. negative) example e if and only if N accepts (resp. rejects) e. A *training set* E is a set of examples over a given vocabulary. A constraint network N is *consistent* with a training set E if and only if N is consistent with every example in E. Given a training set E, E^+ (resp. E^-) denotes the subset of all positive (resp. negative) examples of E. The constraint acquisition task is to find a constraint network that is consistent with a given training set.

3 Compact representations of constraint networks

In this section, we introduce a new compact representation of constraint networks and generalize the constraint acquisition task with this new representation. Our representation is based on the observation that many constraint networks can be efficiently represented using a rule-based formalism, where variables are labeled with attributes and constraints are applied to sequences of variables if and only if their attributes follow certain rules. For example, attributes may correspond to coordinates in a matrix (as in the usual constraint programming model of Sudoku) or specify the type of a variable (day, teacher, room, etc.). In our framework, an attribute is simply a mapping that assigns natural numbers to variables.

Definition 2 (Attribute)**.** *Given a set of variables X, an* attribute *ϕ over X is a function $\phi : X \to \mathbb{N}$.*

The *width* of an attribute ϕ is the maximum value of its range and is denoted by $w(\phi)$. Given a sequence of attributes $\Phi = (\phi_1, \dots, \phi_m)$ over X, we denote $w(\Phi) = (w(\phi_1), \dots, w(\phi_m))$. Given a constant $w \in \mathbb{N}$, we denote w^X the attribute ϕ such that $\forall x \in X, \phi(x) = w$.

Definition 3 (Rule)**.** *Given a set of variables X and a sequence of attributes $\Phi = (\phi_1, \dots, \phi_m)$ over X, a* rule *over Φ is a triple (R, J, f) where R is a relation of arity r, J is a sequence of pairs (i, k) from $\{1, \dots, m\} \times \{1, \dots, r\}$ called the* selector*, and f is a function $f : \mathbb{N}^{|J|} \to \{0, 1\}$ called the* trigger*. Given a scope $S = (x_1, \dots, x_r) \in X^r$ without repetition, we say that the constraint (R, S) is* produced *by the rule iff $f((\phi_i(S[k]))_{(i,k)\in J}) = 1$.*

For clarity, we will slightly abuse notation by writing selectors for a given scope $(x_1, x_2, \dots)$ as (ϕ_i, x_k) instead of (i, k) to make explicit which attribute and which variable are being referenced. Intuitively, a rule (R, J, f) acts as a conditional generator of constraints: it specifies when a relation R should be applied to a scope of variables. The sequence J selects which attribute-variable pairs to examine, and the trigger function f determines whether the constraint should be produced based on the selected attributes and variables.

Definition 4 (Template)**.** *A template T is a quadruple (X, D, Φ, P) where X is a set of variables over the domain D, Φ is a sequence of attributes and P is a set of rules.*

We denote $P(T)$ the set of rules of a template T. The *interpretation* of a template $T = (X, D, \Phi, P)$ denoted $N(T)$ is the constraint network (X, D, C) such that $(R, S) \in C$ iff (R, S) is produced by a rule in P. The interpretation of a specific rule (R, J, f) within a template T is denoted $N(T, (R, J, f))$ and is defined as $N((X, D, \Phi, \{(R, J, f)\}))$. We denote $T + (R, J, f)$ the template obtained by adding the rule (R, J, f) to T, and we denote $T + \phi$ the template obtained by adding the attribute ϕ at the end of the sequence of attributes of T.

Example 1. Let us consider a basic constraint network for the Sudoku problem with 81 variables $X = \{x_{i,j} \mid i, j \in [1, 9]\}$ of domain $D = [1, 9]$. The constraints impose that every pair of variables in the same row (that is, sharing the same index i), same column (same index j) or 3×3 square must be different. This network has a very concise representation as a template.

First, we define three attributes for each cell variable x: its row index ϕ_1, its column index ϕ_2, and its square index ϕ_3. All three attributes range from 1 to 9. Let $\neq$ be the not-equal relation. We can generate all constraints with three simple rules, each applying to distinct pairs of variables (x_u, x_v).

Row Rule: $(\neq, ((\phi_1, x_u), (\phi_1, x_v)), f)$ with $f(a, b) = 1 \Leftrightarrow a = b$. The selector $((\phi_1, x_u), (\phi_1, x_v))$ indicates we compare the first attribute (row) of both variables (i.e. $\phi_1(x_u)$ and $\phi_1(x_v)$). The trigger function f returns $true$ iff the two observed attributes are equal. This rule produces the constraint $(\neq, (x_u, x_v))$ iff x_u and x_v belong to the same row.

Column Rule: $(\neq, ((\phi_2, x_u), (\phi_2, x_v)), f)$, where the trigger function f is the same as the row rule. This rule produces the constraint $(\neq, (x_u, x_v))$ iff x_u and x_v belong to the same column.

Square Rule: $(\neq, ((\phi_3, x_u), (\phi_3, x_v)), f)$, again with the same trigger function f. This rule produces the constraint $(\neq, (x_u, x_v))$ iff x_u and x_v belong to the same 3×3 square.

Thus, a template with just three attributes and three rules is sufficient to generate the entire Sudoku constraint network. This representation of the Sudoku model as a template is not unique. For example, an alternative template would only contain the first two attributes (row and column) but use a more intricate trigger for the third rule: $f_3(r_1, r_2, c_1, c_2) = 1 \Leftrightarrow (\lfloor (r_1 - 1)/3 \rfloor = \lfloor (r_2 - 1)/3 \rfloor) \wedge (\lfloor (c_1 - 1)/3 \rfloor = \lfloor (c_2 - 1)/3 \rfloor)$ with selector $J_3 = ((\phi_1, x_u), (\phi_1, x_v), (\phi_2, x_u)(\phi_2, x_v))$.

As Example 1 illustrates, multiple templates can represent the same constraint network. If only the attribute values change, we say that the templates are *equivalent*.

Definition 5 (Template equivalence)**.** *Two templates $T_1 = (X, D, \Phi_1, P_1)$ and $T_2 = (X, D, \Phi_2, P_2)$ are* equivalent*, denoted $T_1 \equiv T_2$, iff:*
- *$P_1 = P_2$ (identical rules);*
- *$w(\Phi_1) = w(\Phi_2)$ (identical attribute widths);*
- *$\forall (R, J, f) \in P_1, N(T_1, (R, J, f)) = N(T_2, (R, J, f))$ (each rule produces identical constraints).*

For instance, permuting the row indices in the first template of Example 1 will always result in an equivalent template.

We now extend constraint acquisition to template-based representations. A template T is *consistent* with a set of examples E if its interpretation $N(T)$ is consistent with E. Given a training set E over a vocabulary (X, D), the task of constraint acquisition using templates is to learn a template that is consistent with E. Note that learning a template requires learning not only rules, but also attributes.

In general, there always exists a template consistent with any given training set. However, some of these templates are clearly unsatisfactory from a practical point of view; for example, the template might contain as many rules as there are constraints in its interpretation. Instead, we will try to learn templates that are fairly compact (in particular, we impose that each rule produces a significant fraction of the constraints) and that rely on trigger functions with interpretable semantics (e.g. based on integer comparisons and other basic arithmetical relations). We present our approach in the next section.

4 Learning templates

Given a training set E over a vocabulary (X, D), our goal is to find a template T that is consistent with E. We propose a two-step approach. First, we use a constraint acquisition method that tends to learn very dense networks to learn an initial constraint network N consistent with E. Second, we learn a template consistent with E whose interpretation is a subset of the network N learned in the first step. This section details the algorithm for this second step.

4.1 Overview

Our algorithm takes three inputs: a training set E, an initial constraint network N consistent with E and a trigger language Λ. The network N must be consistent with the training set E and provides a superset of constraints from which the template will be constructed. The trigger language Λ is a set of functions that can be used as triggers in the rules. Algorithm 1 describes our algorithm for learning a template. It starts with an empty template (line 1) and greedily learns new attributes and rules (lines 3-10) until the interpretation of the template is consistent with the training set. The process to add new rules ($SaturateWithNewRules$) to the template is described in Section 4.2 and the process to guess a suitable width for a new attribute ($GuessAttributeWidth$) is described in Section 4.3.

Bear in mind that we aim to learn highly compact templates, i.e., templates that produce many constraints using few attributes and rules. For this reason, we do not allow the addition of arbitrary rules or attributes. Instead, we define a set of admissible new rules (Section 4.2) and a heuristic to determine a suitable width for new attributes (Section 4.3). We use a parameter α that controls the minimum number of new constraints that must be produced by a new rule and update it dynamically to allow the search to explore more complex templates as needed. In each iteration of the loop, the algorithm adds a new attribute if a suitable one exists (line 6). Whenever a new attribute is added, the algorithm updates the template with new rules greedily until it is no longer possible to find a new admissible rule (line 7). The algorithm then checks if the interpretation of the template is consistent with the training set E (line 8). If the template is not consistent with E, the parameter α is decreased (line 9). The algorithm then attempts to update the template with admissible rules again (line 10). If T is still not consistent with E, the algorithm goes into the next iteration of the main loop; otherwise, the termination condition is met and the algorithm returns T (line 11).

Algorithm 1: Learning a template

```
Input: A training set $E$; a constraint network
       $N = (X, D, C)$ consistent with $E$; a set of triggers $\Lambda$.
Output: A template $T$ such that $N(T) \subseteq C$ and $N(T)$ is
        consistent with $E$.
1  $T \leftarrow (X, D, \emptyset, \emptyset)$;
2  $\alpha \leftarrow 0.3$;
3  while $N(T)$ is not consistent with $E$ do
4      $w \leftarrow GuessAttributeWidth(T, N, \Lambda, \alpha)$;
5      if $w \geq 0$ then
6          $T \leftarrow T + (w^*)^X$;
7          $T \leftarrow SaturateWithNewRules(T, N, \Lambda, \alpha)$;
8      if $N(T)$ is not consistent with $E$ then
9          $\alpha \leftarrow \alpha \times 0.9$;
10         $T \leftarrow SaturateWithNewRules(T, N, \Lambda, \alpha)$;
11 return $T$;
```

4.2 The procedure $SaturateWithNewRules$

To promote compactness and ensure correctness, we only accept *admissible rules* that produce more than a given number of new constraints.

Definition 6 (Admissible rules)**.** *Given a template T, a constraint network $N = (X, D, C)$, a set of triggers Λ, and a real lb, the set $Adm(T, N, \Lambda, lb)$ consists of all rules (R, J, f) such that:*

- *$f \in \Lambda$: the trigger f is a member of the set of triggers Λ;*
- *$N(T, (R, J, f)) \subseteq C$: constraints produced by the rule are in C;*
- *$|N(T, (R, J, f)) \setminus N(T)| > lb$: the rule produces more than lb new constraints.*

If we restrict the algorithm to add rules to the current template without flexibility, this prevents the discovery of certain rules that could become admissible with different attribute value assignments. To illustrate this limitation, consider a timetabling problem where an attribute representing days is learned first. Initially, days might be assigned arbitrary numerical values (e.g., $Day1 = 3$, $Day2 = 1$, $Day3 = 2$) based on a rule that only requires variables to be scheduled on the same day. If we later need to add a rule requiring constraints between consecutive days ($Day1$ before $Day2$, $Day2$ before $Day3$), the arbitrary initial numbering would make this rule inadmissible. To overcome this, we consider all admissible rules that can be added to any equivalent template (Definition 5) and formalize it through the concept of admissible rules modulo equivalence.

Definition 7 (Admissible rules modulo equivalence)**.** *Given a template T, a constraint network N, a set of triggers Λ, and a threshold lb, $AdmEq(T, N, \Lambda, lb)$ is the set of tuples (T', R, J, f) such that $T' \equiv T$ and $(R, J, f) \in Adm(T', N, \Lambda, lb)$*

Algorithm 2 describes how we add rules to a template T. It iteratively adds admissible rules modulo equivalence that produce the maximum number of new constraints, replacing at each iteration the template with an equivalent one if needed. The minimum number of new constraints lb required for a rule to be admissible is defined as $lb = \alpha \times |N(T)|/|P(T)|$ (with $|N(T)|$ the number of constraints in the interpretation of T and $|P(T)|$ the number of rules in T) when $|P(T)| > 0$, and $lb = 0$ otherwise. This threshold ensures that the new rule achieves a compression ratio comparable to the average of existing rules in the template. We recall that α is updated dynamically in the main algorithm (Algorithm 1) to relax the threshold for admissibility when the template is not consistent with the training set. Algorithm $SaturateWithNewRules$ terminates when no admissible rule can be added to the template, i.e., when $AdmEq(T, N, \Lambda, lb) = \emptyset$.

Algorithm 2: $SaturateWithNewRules(T, N, \Lambda, \alpha)$:

```
Input: A template $T$; A constraint network $N = (X, D, C)$;
       a set of triggers $\Lambda$; a real $\alpha$.
Output: An updated template with new rules added.
1 $lb \leftarrow \alpha \times \frac{|N(T)|}{\max(|P(T)|,1)}$;
2 while $AdmEq(T, N, \Lambda, lb) \neq \emptyset$ do
3     $(T', R, J, f) \leftarrow \operatorname*{argmax}_{(T',R,J,f) \in AdmEq(T,N,\Lambda,lb)} \Big(|N(T' + (R, J, f))| - |N(T)|\Big)$;
4     $T \leftarrow T' + (R, J, f)$;
5     $lb \leftarrow \alpha \times \frac{|N(T)|}{|P(T)|}$;
6 return $T$;
```

4.3 The procedure $GuessAttributeWidth$

In each iteration of the main loop, Algorithm 1 attempts to add a new attribute to the template. When adding this attribute, a key difficulty is to find a suitable width w^* that balances the need for expressiveness with the goal of keeping the template compact. In our algorithm, we set the possible widths to be $0 \leq w < |X|$. Let $cov(w, T, N, \Lambda)$ denote the maximum number of constraints that can be newly produced by a new rule when adding an attribute of width at most w:

$$\max_{\substack{w' \leq w \\ (T',R,J,f) \in AdmEq((T+(w')^X),N,\Lambda,0)}} \Big(|N(T'+(R, J, f))|-|N(T)|\Big)$$

For notational clarity, in the following discussion we omit the parameters T, N, and Λ from the function notation of $cov(w, T, N, \Lambda)$ as they are fixed during the process of adding an attribute, i.e., $cov(w) = cov(w, T, N, \Lambda)$. As w grows from 0 to $|X|-1$, the function $cov(w)$ is non-decreasing but will typically grow in a non-linear fashion. Assuming the data contains a hidden feature, we may expect that $cov(w)$ grows quickly as w approaches the width of that feature and slowly afterwards. We propose the *maximum cover above expectation* (MCAE) as a heuristic criterion for determining when that happens. For $0 \leq w < |X|$, let $cov_{lin}(w)$ denote the approximation of $cov(w)$ obtained by linear interpolation between 0 and $|X|-1$, i.e. $cov_{lin}(w) = cov(0)+w\cdot(cov(|X|-1)-cov(0))/(|X|-1)$. If cov grows unexpectedly fast when approaching a value w' and slowly between $w'+1$ and $|X|-1$, then the gap $cov(w')-cov_{lin}(w')$ will be large. Therefore, if we return $w^* = \arg\max(cov(w) - cov_{lin}(w))$, we have an increased chance of returning the domain size of a hidden feature.

In order to make progress, we impose that a new attribute makes it possible to add at least one new admissible rule. This translates into a constraint $cov(w^*) > lb$, with lb being the lower bound on the number of new constraints produced by a rule described in the previous section. Computing the optimum value w^* for the above criterion requires solving three optimization problems: two for computing $cov(0)$ and $cov(|X| - 1)$ and then one for computing w^*. In order to avoid computing $cov(0)$ we crudely approximate it with 0. The complete procedure to compute the optimum width w^* is described in Algorithm 3.

Algorithm 3: $GuessAttributeWidth(T, N, \Lambda, \alpha)$

```
Input: A template $T$; a constraint network $N = (X, D, C)$; a
       set of triggers $\Lambda$; a real $\alpha$.
Output: The optimal width $w^*$ for a new attribute, or $-1$ if
        no admissible attribute exists.
1 $cov_{max} \leftarrow cov(|X| - 1, T, N, \Lambda)$;
2 $lb \leftarrow \alpha \times \frac{|N(T)|}{\max(|P(T)|,1)}$;
3 if $cov_{max} > lb$ then
4     $w^* \leftarrow \operatorname*{argmax}_{0 \leq w < |X|,\ cov(w,T,N,\Lambda) > lb} \Big(cov(w, T, N, \Lambda) - \frac{w}{|X|-1} \cdot cov_{max}\Big)$;
5     return $w^*$;
6 return $-1$;
```

4.4 Termination and correctness

For any integer $r > 1$, let $f^r_{suc} : \mathbb{N}^r \to \{0, 1\}$ be the function given by $f^r_{suc}(a_1, a_2, \ldots, a_r) = 1 \Leftrightarrow (a_1 + 1 = a_2) \wedge (a_2 + 1 = a_3) \wedge \cdots \wedge (a_{r-1} + 1 = a_r)$. We call f^r_{suc} the *successor function* of arity r.

Proposition 1. *Algorithm 1 is guaranteed to terminate and return a template consistent with E if f^r_{suc} is in Λ for all r such that N contains a constraint of arity r, and N contains at least $r_{max} + 2$ variables with r_{max} the maximum arity of a constraint in N.*

Proof. Correctness is immediate as Algorithm 1 can only exit (or skip) the main loop if $N(T)$ is consistent with E.

For termination, the loop in Algorithm 2 can only iterate at most $|C|$ times in total (where C is the set of constraints in N) because an admissible rule must produce at least one new constraint in N. In addition, the parameter α strictly decreases at each iteration of the main loop. Eventually, any rule producing at least one constraint from $N \setminus N(T)$ becomes admissible.

Consider a constraint $(R, (x_1, \ldots, x_r))$ of arity r in N but not already in $N(T)$. We show that a rule producing only this constraint always exists provided Λ contains the successor function of arity r and a new attribute ϕ_i of width $r+1$ can be introduced at line 6 (which is guaranteed by our assumption that $|X| \geq r+2$). We set $\phi_i(x_1) = 0$, $\phi_i(x_2) = 1, \ldots, \phi_i(x_r) = r-1$, and $\phi_i(y) = r+1$ for all other variables y. The rule $(R, ((\phi_i, x_1), (\phi_i, x_2), \ldots, (\phi_i, x_r)), f^r_{suc})$ produces $(R, (x_1, \ldots, x_r))$ and no other constraint.

It follows from the argument above that each iteration of the main loop of Algorithm 1 will add at least one new constraint to $N(T)$ once the threshold becomes strictly below 1. Together with the invariant $N(T) \subseteq N$ and the fact that N is consistent with E, this implies that $N(T)$ will eventually be consistent with E as well. At this point, the algorithm will exit the main loop and return T. □

5 Model

This section details the CP models used to solve the optimization subproblems of our learning algorithm: finding an optimal new attribute width (line 4 of Algorithm 3) and the most impactful new rule (line 3 of Algorithm 2).

Our models assume that the trigger language Λ is composed of all functions expressible as the conjunction of two elementary binary Boolean functions taken from a base set Λ'. (This is the setting we chose for our experiments, see Section 6 for more details.)

We represent a rule over Λ as a tuple (R, J_1, f_1, J_2, f_2), where $f_1, f_2 \in \Lambda'$ and J_1, J_2 are the attribute selectors for f_1 and f_2, respectively. The rule produces a constraint (R, S) if both trigger functions evaluate to true, i.e., iff $f_1((\phi_i(S[k]))_{(i,k)\in J_1}) = 1$ and $f_2((\phi_i(S[k]))_{(i,k)\in J_2}) = 1$.

The CP model searches for a rule of this kind and (optionally) a new attribute ϕ_{new}. For the remainder of the section, let:

- $T = (X, D, \Phi, F)$ be the current template with Φ the set of existing attributes and F the set of existing rules;
- $N = (X, D, C)$ be the initial constraint network (over a constraint language Γ) learned by the baseline constraint acquisition method;
- Λ' be the language allowed for the trigger functions f_1, f_2 in a rule;
- $\mathcal{J}$ be the set of all possible attribute selectors $J = ((i_1, k_1), (i_2, k_2))$ using attributes from $\Phi \cup \{\phi_{\text{new}}\}$;
- C_+ be the set of constraints from the initial network N that are not yet produced by the current template T;
- C_- be the set of constraints (R', S) (where $S \in X^r$, $R' \in \Gamma$ of arity r) such that $(R', S) \notin C$. This is the set of constraints that must not be produced by any rule.

The variables of the model are:

- *Attribute values*: For each variable $x \in X$:
  - For each existing attribute $\phi_i \in \Phi$, an integer variable $v_{x,i}$ representing $\phi_i(x)$ of domain $[0, w(\phi_i)]$.
  - For the new attribute ϕ_{new}, an integer variable $v_{x,\text{new}}$ representing $\phi_{\text{new}}(x)$ of domain $[0, n-1]$.
- *New attribute width*: An integer variable d representing the width of the new attribute ϕ_{new}. We set $d \in [0, n-1]$ with n the total number of variables in the network N. We add constraints to ensure $v_{x,\text{new}} \leq d$ for each variable $x \in X$.
- *New rule*: Boolean variables to define the new rule (R, J_1, f_1, J_2, f_2):
  - $X_R(R')$ for each $R' \in \Gamma$;
  - $X_{f1}(f), X_{f2}(f)$ for each $f \in \Lambda'$;
  - $X_{J1}(J), X_{J2}(J)$ for each $J \in \mathcal{J}$.

  We ensure that exactly one variable is true in each group (e.g., $\sum_{R'\in\Gamma} X_R(R') = 1$). We denote $\mathit{selected}_{J1}(R', J_1, f_1) \equiv X_R(R') \wedge X_{f1}(f_1) \wedge X_{J1}(J_1)$ and $\mathit{selected}_{J2}(R', J_2, f_2) \equiv X_R(R') \wedge X_{f2}(f_2) \wedge X_{J2}(J_2)$ that indicate which rule is currently selected for each possible $(R', J_1, f_1) \in \Gamma \times \Lambda' \times \mathcal{J}$ and $(R', J_2, f_2) \in \Gamma \times \Lambda' \times \mathcal{J}$.
- *Constraints produced*: For each target constraint $(R, S) \in C_+$, a Boolean variable $c_{(R,S)}$ indicating whether the new rule produces this constraint.

We also define a helper predicate $\mathit{trigger}(S, J)$ which evaluates to true iff $t(\phi_{i_1}(S[k_1]), \ldots, \phi_{i_t}(S[k_t]))$, where $J = ((i_1, k_1), \ldots, (i_t, k_t))$ and the attribute values $\phi_{i_x}(S[k_x])$ correspond to the value of $v_{S[k_x], i_x}$.

New constraints produced For each $(R', S) \in C_+$, we force that $c_{(R',S)}$ is true iff the new rule produces the constraint (R', S). For each rule part $(J, f) \in \Lambda' \times \mathcal{J}$:

$$c_{(R',S)} \Rightarrow \neg\mathit{selected}_{J1}(R', J, f) \vee \mathit{trigger}(S, J, f)$$
$$c_{(R',S)} \Rightarrow \neg\mathit{selected}_{J2}(R', J, f) \vee \mathit{trigger}(S, J, f)$$

We do not have to ensure the opposite direction of the implication because it will be implicitly enforced by the maximization objectives (described below).

Forbidden constraints For each forbidden constraint $(R', S) \in C_-$ we introduce a corresponding boolean variable $t_{(R',S)}$. Then, for each rule part $(J, f) \in \mathcal{J} \times \Lambda'$ we ensure that:

$$\neg\mathit{selected}_{J1}(R', J, f) \vee \neg\mathit{trigger}(S, J, f) \vee t_{(R',S)}$$
$$\neg\mathit{selected}_{J2}(R', J, f) \vee \neg\mathit{trigger}(S, J, f) \vee \neg t_{(R',S)}$$

Existing rules For each existing rule $(R, J_1, f_1, J_2, f_2) \in F$, and for every scope S with arity matching R we ensure that $\mathit{trigger}(S, J_1, f_1) \wedge \mathit{trigger}(S, J_2, f_2)$ is true if (R, S) is produced by the rule in the initial T and false otherwise.

Threshold and objective function As described in Section 4.2, we only seek solutions in which the new rule produces a minimum number of new constraints. Given the current value of α, we add:

$$\sum_{(R',S)\in C_+} c_{(R',S)} > \alpha \times |N(T)|/|F|$$

The MCAE heuristic involves two optimization phases using this CP model. During the first phase, we compute the maximum number $\mathit{cov}(n-1)$ of new constraints that can be produced if the width of the new attribute is at most $n-1$. This is done by maximizing

$$\sum_{(R',S)\in C_+} c_{(R',S)}$$

without additional constraints. During the second phase, we compute the width w^* for the new attribute that maximizes the MCAE criterion. For this, we use the value $\mathit{cov}(n-1)$ calculated in the first phase and maximize

$$\left(\sum_{(R',S)\in C_+} c_{(R',S)}\right) - \frac{d}{n-1} \cdot \mathit{cov}(n-1).$$

The solution to this second CP model yields the optimal width $d = w^*$, the variable-value assignments for the new attribute ϕ_{new} (and potentially updated values for $\phi_i \in \Phi$), and the description of an admissible rule that produces that maximum number of new constraints.

The CP model can be readily adapted for the task of adding a new rule without introducing a new attribute (line 3 of Algorithm 2). This is achieved by removing all variables and constraints related to ϕ_{new} and using the objective function of the first phase (maximization of newly produced constraints).

6 Experimental evaluation

In this section, we evaluate our method that we call TACQ (which includes both Algorithm 1 and the baseline acquisition method used to learn the initial network N) experimentally on several benchmark problems. For each benchmark, we will compare the classification accuracy of the interpretation of the template learned by TACQ and that of the network learned by the baseline acquisition method. We will then dive deeper into the details, using the nurse rostering problem as an example, to assess the effectiveness of MCAE for determining attribute widths and examine the structure of the template learned by TACQ.

As the method used to generate the initial network N, we could use any constraint acquisition method, such as CONACQ.1 [2, 3] (with the most specific network it suggests) or BAYESACQ [11]. We chose to use LANGUAGE-FREE ACQ (LFA) [4] because it only needs a training set as input, whereas both CONACQ.1 and BAYESACQ require background knowledge in the form of a constraint language. This allows our full framework to only need a training set and a trigger language as input. Since our template acquisition algorithm acts as a refinement step over the output of LFA, we will use LFA as baseline for comparison. We fix the trigger language Λ to be the set of all trigger functions that can be expressed as conjunctions of two binary Boolean functions taken from the set $\{f_1, f_=, f_{\neq}, f_<, f_{\leq}, f^2_{suc}\}$, where f_1 is the constant binary function that always returns 1, $f_R(x, y) = 1$ iff xRy for $R \in \{=, \neq, <, \leq\}$, and f^2_{suc} is the successor function defined in Section 4.4.

We have implemented the framework described in Section 4 as well as the LFA algorithm in the Python programming language. The underlying CP solver is Google OR-Tools [10]. All experiments were conducted on an AMD Epyc 9554 processor (utilizing 8 cores per run) and 16GB of RAM.

6.1 Benchmark problems

For each benchmark instance, unless otherwise mentioned, we generated independently a training and a test set. The solutions are generated by finding solutions to a constraint network representing the target concept using a CP solver with a randomized value selection strategy. Negative examples are generated from solutions, with half of the negative examples created by randomly permuting the values assigned to two variables in a solution. The other half was created by randomly altering the value assigned to a single variable in a solution. All the benchmarks used in the experimental evaluation of the baseline method LFA [4] (Sudoku, Jigsaw Sudoku, Schur's Lemma, Subgraph Isomorphism, N-Queens and Golomb Ruler) are included in our benchmarks, with identical parameters.

Sudoku The problem involves filling a 9×9 grid with digits from 1 to 9 such that each row, column, and 3×3 square contains all the digits exactly once. The variables are the 81 cells of the grid with domain the digits from 1 to 9. The constraints are that each pair of variables in the same row, column, and 3×3 square must be different.

Jigsaw Sudoku This problem is the same as Sudoku, but instead of the standard 3×3 squares, the grid is divided into irregular shapes, called jigsaw pieces. The variables are the cells of the grid with domain the digits from 1 to 9. The constraints are that each pair of variables in the same row, column, and jigsaw piece must be different. There exists significant variance in experimental results depending on the shape of the jigsaw pieces. To reflect this, we used the three different layouts (**[#1]**, **[#2]** and **[#3]**) from [4].

Schur's Lemma The Schur's Lemma problem consists of putting n balls labeled from 1 to n into 3 boxes such that for any triple of balls (x, y, z) such that $x + y = z$, not all three balls are in the same box. The variables $\{x_1, \dots, x_n\}$ are the n balls with domain $\{1, 2, 3\}$. The constraints are $NotAllEqual(x_i, x_j, x_k)$ for all i, j, k such that $i + j = k$. We ran experiments on this problem with $n = 9$ which is the parameter with the highest number of solutions (546).

Subgraph Isomorphism Given two graphs G and H, the subgraph isomorphism problem involves determining whether G contains a subgraph that is isomorphic to H. For this problem, the target constraint network consists of $|H|$ variables $x_1, \dots, x_n$ with domains of size $|G|$. Binary constraints $x_i \neq x_j$ for all i, j ensure that the mapping between the vertices of H and G is a one-to-one function. Additionally, another binary constraint ensures that for any edge (a, b) in H, the pair (x_a, x_b) is an edge in G. In our experiments, H is a cycle of size 5 and G is a random graph with 20 vertices and 100 edges. For this benchmark, negative examples are generated as paths and closed walks of G computed using a randomized value selection.

N-Queens (coordinate-based) The N-Queens problem involves placing N queens on an $N \times N$ chessboard such that no two queens threaten each other according to chess rules. We employ the standard coordinate-based representation with one variable per column. The constraint network consists of N variables $x_1, \dots, x_n$, where x_i represents the row position of the queen in the ith column. Each variable has domain $\{1, \dots, N\}$. For all i, j where $i \neq j$, the constraints are $x_i \neq x_j$ (ensuring that no two queens share the same row) and $|x_i - x_j| \neq |i - j|$ (ensuring no diagonal attacks). This formulation yields a binary constraint language of size N, corresponding to the possible diagonal distance values. Our experiments used $N = 8$, which produces a problem with 92 distinct solutions. For training data, positive examples were generated by computing random solutions to the constraint network out of the 92 possible solutions.

Golomb Ruler The Golomb Ruler problem involves finding a set of marks on a ruler such that the difference between any two marks is unique. The target constraint network consists of n variables, each representing the position of a mark on the ruler, with domains of fixed size $\{0, \dots, m\}$. The network includes a quaternary constraint $|x_i - x_j| \neq |x_k - x_l|$ for all $i < j$ and $k < l$. For our experiments, we use $n = 10$ and $m = 60$. Positive examples are generated by computing a random solution of the target constraint network with the symmetry-breaking constraint $x_i < x_j$ for all $i < j$, followed by randomly permuting the values of this solution. For this benchmark, non-solutions are generated by randomly altering one value in a solution.

Exam Timetabling This problem (used as a benchmark in [14, 15, 12, 13]) involves scheduling exams for a set of courses across multiple semesters within a specified period, with the goal of assigning each course to a unique timeslot while adhering to specific constraints. An instance of the problem is defined by five parameters: s the number of semesters, n the courses per semester, t the timeslots per day, d the number of days and r the number of rooms. The variables are the $s \times n$ courses with domain the $t \times d \times r$ timeslots. The constraints are that each course must be assigned to a unique timeslot and that courses from the same semester must be scheduled on different days to avoid conflicts.

[#1] 3 semesters and 2 courses with 3 days, 2 slots and 1 room;
[#2] 4 semesters and 3 courses with 3 days, 2 slots and 2 rooms;
[#3] 5 semesters and 4 courses with 5 days, 2 slots and 2 rooms.

Nurse Rostering This problem is used in [12, 13] and also present in [7, 6] with slight differences. It involves scheduling nurses for a set of shifts over a specified period, with the goal of assigning each shift to a nurse while adhering to specific constraints. An instance of the problem is defined by three parameters: n the number of nurses, s the number of shifts per day, k the number of slots per shift and d the number of days. The variables are the $k \times s \times d$ slots with domain the n nurses. The constraints are that no two slots of the same day are assigned the same nurse and slots in the last shift of a day and the first shift of the next day cannot be assigned the same nurse. For this problem, we have selected three instances with different parameter configurations.

[#1] 4 days, 2 shifts and 4 nurses per shift with 12 nurses;
[#2] 5 days, 3 shifts and 5 nurses per shift with 18 nurses;
[#3] 7 days, 3 shifts and 3 nurses per shift with 15 nurses.

All benchmark instances can be modeled with constraints of arity at most 3 (even Golomb Ruler with $n = 10$, as shown in [4]).

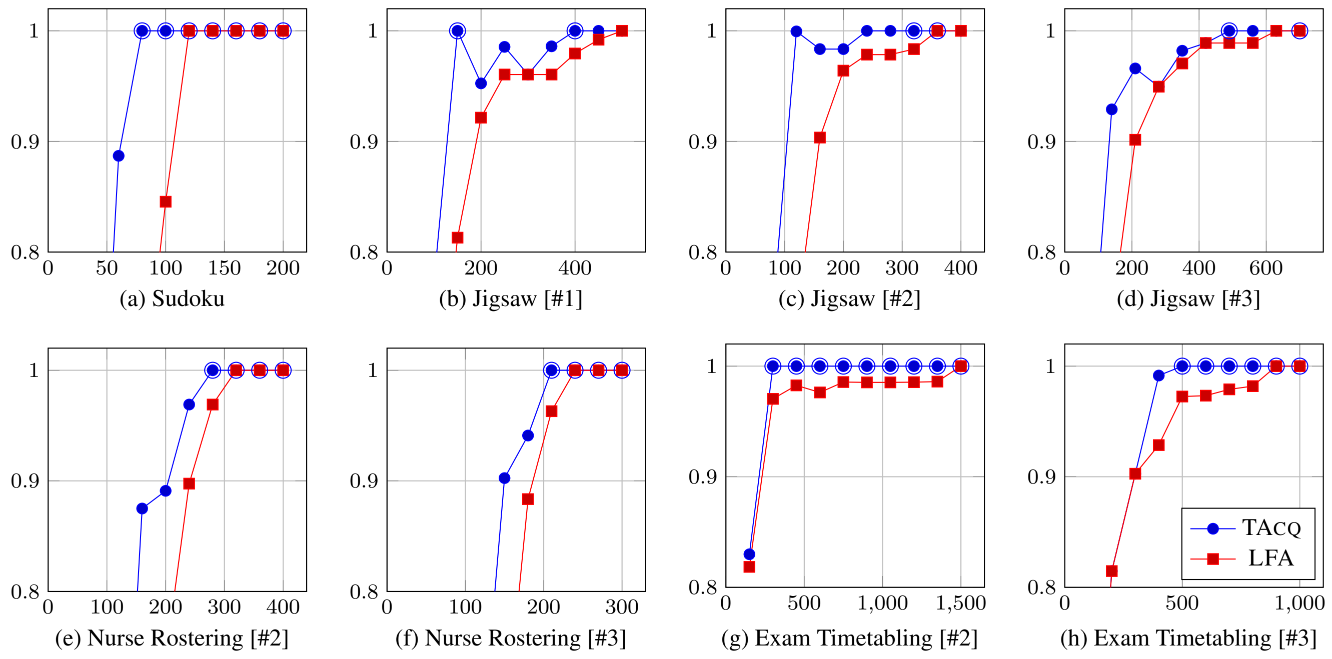

Figure 1: Accuracy over independently generated examples of the network learned by LANGUAGE-FREE ACQ [LFA] and with our method [TACQ] as a function of the number of examples in the training set. We circle the points where the template found is equivalent to the target.

The LFA algorithm, which generates the initial network N, prioritizes learning lower arity constraints so N also meets this maximum arity. Our trigger language Λ includes the successor function f^2_{succ}, and f^3_{succ} can be expressed by a conjunction of two binary successor functions. As a result, the conditions of Proposition 1 are satisfied, guaranteeing termination in our experiments.

6.2 *Accuracy and equivalence*

Protocol For each benchmark instance, we executed both LFA and our method TACQ. The performance of a model is measured in terms of its accuracy, which is computed on a separate set of 2000 examples generated independently. All training and test sets contain an equal number of positive and negative examples. We conduct a series of experiments with increasing numbers of examples in the training sets, systematically selected within an interval such that the upper bound allows LFA to achieve 100% accuracy or run out of solutions for the training set. We also record for each instance whether the learned network is equivalent to the model used for data generation. We set a timeout of 3 hours for each call to the CP solver.[1]

Accuracy A summary comparing the learning curves of TACQ and LFA for representative benchmarks is shown in Figure 1. We omit the smallest instances of Exam Timetabling and Nurse Rostering from the figure due to space constraints. We also omit Schur's Lemma, Subgraph Isomorphism, N-Queens and Golomb Ruler as the accuracy curves of both LFA and TACQ are exactly the same.

We observe that our method consistently yields accuracy results that are either equivalent or superior to those of LFA across all experiments. For Sudoku, TACQ achieves 100% accuracy for all runs with at least 80 examples in the training set, whereas LFA requires 120 examples. Across the Jigsaw Sudoku instances, TACQ required on average 25% fewer examples than LFA to consistently reach 100% accuracy. For Nurse Rostering, TACQ reduced the required number of examples by 21% on average. For the Exam Timetabling benchmark, the average reduction was 41%, peaking at 80% for the instance [#2].

Several benchmarks (Schur's Lemma, Subgraph Isomorphism, N-Queens, Golomb Ruler) have identical accuracy progression for both TACQ and LFA. Concerning the Subgraph Isomorphism, N-Queens and Golomb Ruler, this occurs primarily because LFA fails to learn a constraint network over the same language as the target model. As TACQ learns a template whose interpretation is a subset of the network N provided by LFA, no fundamental improvement is possible in this case. For Schur's Lemma, the natural template representation of the target model (a single rule that produces $NotAllEqual(x_i, y_j, z_k)$ iff $i + j = k$) is not expressible with our trigger language. This causes TACQ to learn many rules that overfit the initial network N returned by LFA. To summarize, if we ignore the 8-Queens problem where 100% accuracy is never reached by neither LFA nor TACQ, we need on average over all benchmark instances 22% fewer examples than LFA to consistently learn a network with 100% accuracy.

Equivalence and runtimes The learned model is equivalent to the target model for all experiments where TACQ reached 100% accuracy, with the exception of the Jigsaw Sudoku benchmark. In these three instances, TACQ fails to consistently learn an equivalent model. Only 6 out of the 13 templates achieving 100% accuracy had their interpretation equivalent to the target network. We believe this behavior is caused by two distinct factors. First, we use a biased training set, as row and column constraints are sufficient to reject all negative examples. This bias occasionally causes the main loop of Algorithm 1 to exit early, with 100% accuracy achieved but not equivalence. Second, the model learned by LFA in the first step contains a large number of redundant constraints. This makes the constraint optimization models for new rules and attributes more difficult to solve, with OR-Tools frequently reaching the timeout and failing to consistently return an optimal solution.

[1] Complete results, code and data of the experiments are available online [5]

Benchmark	$\lvert E\rvert$	LFA	TACQ
Sudoku	80	1m	15h 32m 9s
Jigsaw [#1]	400	39s	31h 36m 27s
Jigsaw [#2]	240	46s	27h 20m 9s
Jigsaw [#3]	490	41s	33h 54m 58s
Schur's Lemma	560	4s	1h 45m 4s
Subgraph Isomorphism	640	12s	19s
8-Queens	184	10s	1m 22s
Golomb ruler	2100	3m 59s	2h 42m 20s
Exam Timetabling [#1]	119	1s	2s
Exam Timetabling [#2]	300	22s	27s
Exam Timetabling [#3]	500	4m 24s	4m 43s
Nurse Rostering [#1]	100	37s	21m 11s
Nurse Rostering [#2]	280	5m 12s	9h 51m 51s
Nurse Rostering [#3]	210	33s	2h 11m 21s

Table 1: Comparison of runtimes for LFA and TACQ. Runtimes for TACQ include the time taken by LFA to learn the initial network N.

More generally, we noted that TACQ is significantly slower than LFA on all benchmarks except Exam Timetabling, sometimes by orders of magnitude as illustrated in Table 1. This is not surprising because TACQ solves multiple difficult optimization problems as part of the learning process. This makes TACQ most suited for applications where examples are scarce (or costly to obtain) and learning can be done off-line.

6.3 *Learned attributes*

A key component of our template learning algorithm is the MCAE heuristic, which we use to determine the width of a new attribute. This heuristic aims to find a width that balances maximizing the potential for new rules to produce constraints against the risk of overfitting introduced by a large attribute domain. To illustrate the behavior and effectiveness of MCAE, we analyze its application during the learning process for the instance [#3] of Nurse Rostering (7 days, 3 shifts and 3 nurses per shift with 15 nurses) with 210 examples in the training set. In this setting, our algorithm learns two attributes ϕ_1, ϕ_2 and two rules. For each attribute, Figure 2 shows the value of $cov(w)$ (defined as the maximum number of new constraints that can be produced by a rule when the new attribute has width w) and the value of the MCAE objective function for each potential width w. Our algorithm selects the width w^* that maximizes the MCAE objective.

Attribute ϕ_1 The function cov increases stepwise, with minor gains at widths 3 and 5, followed by a very sharp increase at width $w = 6$, where 252 new constraints can be produced. For $w > 6$, cov plateaus completely until $w = 62$ which corresponds to the maximum width possible (the graph stops at 20 for brevity). The MCAE objective function reaches a global maximum at $w = 6$, a width that matches a hidden feature in the data (the number of days). This width corresponds to the 7 days in the problem data, as the attribute values are indexed from 0 to 6.

Attribute ϕ_2 For the second attribute, cov increases rapidly between $w = 0$ and $w = 3$, reaching 54 new constraints produced. Beyond $w = 3$, the coverage enters a long plateau, remaining at 54 until $w = 6$. A very small increase occurs at $w = 7$, where the maximum observed coverage reaches 55 constraints, after which it

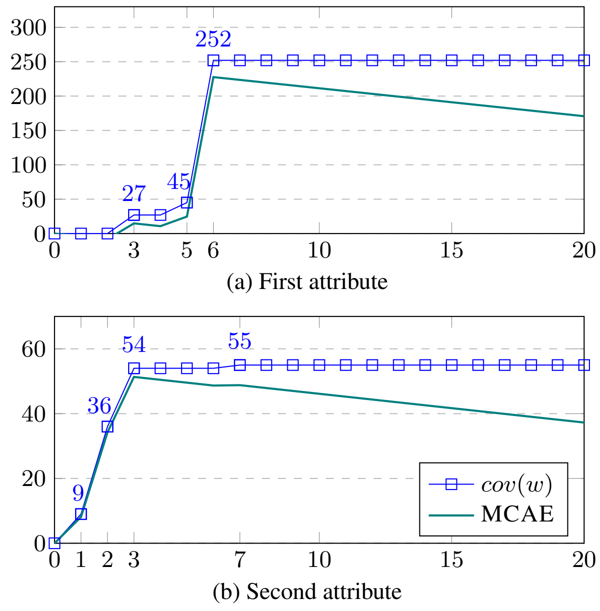

Figure 2: Evolution of $cov(w)$ and value of the MCAE objective function depending on the width of the first and second attributes for the instance [#3] of Nurse Rostering.

plateaus again until $w = 62$ (i.e. $n - 1$). The MCAE heuristic correctly identifies the smallest attribute width $w = 3$ that enables the creation of a rule producing all the constraints in the target model missing from the interpretation of the template.

We could observe from the learned template that the first attribute ϕ_1 correctly partitions the slots into seven days (numbered from 6 to 0, hence corresponding to a width of 6). Similarly, the second attribute ϕ_2 groups the slots within each day into numbered shifts such that the last shift of a day is equal to the first shift of the next day plus one. The two rules learned on these attributes correspond respectively to "no nurse can be assigned to two slots on the same day" and "no nurse can be assigned to the last shift of a day and the first shift of the next day". These interpretations can be directly recovered from the trigger functions of each rule.

7 Conclusion

We introduced templates, a compact representation of constraint networks, to capture recurring structures within CP models. We developed a novel framework for constraint acquisition that learns templates, optimizing for a high compression ratio. Our experiments validate this approach. On various structured benchmarks (Sudoku, Jigsaw Sudoku, Nurse Rostering, and Exam Timetabling), our method significantly reduced the number of training examples needed to achieve high accuracy compared to the baseline LANGUAGE-FREE ACQ (LFA) algorithm. Furthermore, as seen in the Nurse Rostering example, the learned templates can be highly interpretable.

A promising research direction is to investigate how these interpretable rules can be leveraged to learn parameterized CP models, which can generalize to different instances of the same problem. Experiments also highlight that our method does not improve upon LFA for certain benchmarks; we believe that this limitation can be alleviated by integrating background knowledge (e.g. known variable attributes or information on the target constraint language).

Acknowledgements

This work was partially supported by the TAILOR project, funded by EU Horizon 2020 research and innovation programme under GA No 952215, by the AI Interdisciplinary Institute ANITI, funded by the French program "Investing for the Future – PIA3" under grant agreement no. ANR-19-PI3A-0004, and by the ANR AXIAUM project ANR-20-THIA-0005-01 (Data Science Institute of the University of Montpellier). Experiments were performed with the support of MESO@LR-Platform at the University of Montpellier.

References

[1] N. Beldiceanu and H. Simonis. A model seeker: Extracting global constraint models from positive examples. In M. Milano, editor, *Principles and Practice of Constraint Programming - 18th International Conference, CP 2012, Québec City, QC, Canada, October 8-12, 2012. Proceedings*, volume 7514 of *Lecture Notes in Computer Science*, pages 141–157. Springer, 2012. doi: 10.1007/978-3-642-33558-7_13. URL https://doi.org/10.1007/978-3-642-33558-7_13.

[2] C. Bessiere, R. Coletta, F. Koriche, and B. O'Sullivan. A sat-based version space algorithm for acquiring constraint satisfaction problems. In J. Gama, R. Camacho, P. Brazdil, A. Jorge, and L. Torgo, editors, *Machine Learning: ECML 2005, 16th European Conference on Machine Learning, Porto, Portugal, October 3-7, 2005, Proceedings*, volume 3720 of *Lecture Notes in Computer Science*, pages 23–34. Springer, 2005. doi: 10.1007/11564096_8. URL https://doi.org/10.1007/11564096_8.

[3] C. Bessiere, F. Koriche, N. Lazaar, and B. O'Sullivan. Constraint acquisition. *Artif. Intell.*, 244:315–342, 2017. doi: 10.1016/j.artint.2015.08.001. URL https://doi.org/10.1016/j.artint.2015.08.001.

[4] C. Bessiere, C. Carbonnel, and A. Himeur. Learning constraint networks over unknown constraint languages. In *Proceedings of the Thirty-Second International Joint Conference on Artificial Intelligence, IJCAI 2023, 19th-25th August 2023, Macao, SAR, China*, pages 1876–1883. ijcai.org, 2023. doi: 10.24963/IJCAI.2023/208. URL https://doi.org/10.24963/ijcai.2023/208.

[5] A. Himeur. Code and data for "learning compact representations of constraint networks". Available at https://gite.lirmm.fr/coconut/tacq, 2025.

[6] M. Kumar, S. Teso, and L. D. Raedt. Acquiring integer programs from data. In S. Kraus, editor, *Proceedings of the Twenty-Eighth International Joint Conference on Artificial Intelligence, IJCAI 2019, Macao, China, August 10-16, 2019*, pages 1130–1136. ijcai.org, 2019. doi: 10.24963/IJCAI.2019/158. URL https://doi.org/10.24963/ijcai.2019/158.

[7] M. Kumar, S. Kolb, and T. Guns. Learning constraint programming models from data using generate-and-aggregate. In C. Solnon, editor, *28th International Conference on Principles and Practice of Constraint Programming, CP 2022, July 31 to August 8, 2022, Haifa, Israel*, volume 235 of *LIPIcs*, pages 29:1–29:16. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2022. doi: 10.4230/LIPICS.CP.2022.29. URL https://doi.org/10.4230/LIPIcs.CP.2022.29.

[8] A. Lallouet, M. Lopez, L. Martin, and C. Vrain. On learning constraint problems. In *22nd IEEE International Conference on Tools with Artificial Intelligence, ICTAI 2010, Arras, France, 27-29 October 2010 - Volume 1*, pages 45–52. IEEE Computer Society, 2010. doi: 10.1109/ICTAI.2010.16. URL https://doi.org/10.1109/ICTAI.2010.16.

[9] T. P. Pawlak and K. Krawiec. Automatic synthesis of constraints from examples using mixed integer linear programming. *Eur. J. Oper. Res.*, 261(3):1141–1157, 2017. doi: 10.1016/j.ejor.2017.02.034. URL https://doi.org/10.1016/j.ejor.2017.02.034.

[10] L. Perron and V. Furnon. Or-tools (v9.11), 2025. URL https://developers.google.com/optimization/.

[11] S. D. Prestwich, E. C. Freuder, B. O'Sullivan, and D. Browne. Classifier-based constraint acquisition. *Ann. Math. Artif. Intell.*, 89 (7):655–674, 2021. doi: 10.1007/s10472-021-09736-4. URL https://doi.org/10.1007/s10472-021-09736-4.

[12] D. Tsouros, S. Berden, and T. Guns. Learning to learn in interactive constraint acquisition. *Proceedings of the AAAI Conference on Artificial Intelligence*, 38(8):8154–8162, Mar. 2024. doi: 10.1609/aaai.v38i8.28655. URL https://ojs.aaai.org/index.php/AAAI/article/view/28655.

[13] D. Tsouros, S. Berden, S. Prestwich, and T. Guns. Generalizing constraint models in constraint acquisition. In T. Walsh, J. Shah, and Z. Kolter, editors, *AAAI-25, Sponsored by the Association for the Advancement of Artificial Intelligence, February 25 - March 4, 2025, Philadelphia, PA, USA*, pages 11362–11371. AAAI Press, 2025. doi: 10.1609/AAAI.V39I11.33236. URL https://doi.org/10.1609/aaai.v39i11.33236.

[14] D. C. Tsouros and K. Stergiou. Efficient multiple constraint acquisition. *Constraints An Int. J.*, 25(3-4):180–225, 2020. doi: 10.1007/S10601-020-09311-4. URL https://doi.org/10.1007/s10601-020-09311-4.

[15] D. C. Tsouros and K. Stergiou. Learning max-csps via active constraint acquisition. In L. D. Michel, editor, *27th International Conference on Principles and Practice of Constraint Programming, CP 2021, Montpellier, France (Virtual Conference), October 25-29, 2021*, volume 210 of *LIPIcs*, pages 54:1–54:18. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2021. doi: 10.4230/LIPICS.CP.2021.54. URL https://doi.org/10.4230/LIPIcs.CP.2021.54.