THÈSE POUR OBTENIR LE GRADE DE DOCTEUR DE L'UNIVERSITÉ DE MONTPELLIER

En Informatique

École Doctorale : Information, Structures et Systèmes (I2S)

Unité de Recherche : Laboratoire d'informatique, de Robotique et de Microélectronique de Montpellier (LIRMM)

Reducing Dependency on Prior Knowledge in Constraint Acquisition

Présentée par Areski HIMEUR
le 03 décembre 2025

Sous la direction de Christian BESSIERE
Co-encadré par Clément CARBONNEL

Devant le jury composé de

Gilles AUDEMARD, Professeur des universités, Université d'Artois - CRIL	Rapporteur
Christian BESSIERE, Directeur de recherche, CNRS	Directeur de thèse
Clément CARBONNEL, Chargé de recherche, CNRS	Co-encadrant de thèse
Florence DUPIN DE SAINT-CYR, Maîtresse de conférences, Université de Toulouse - IRIT	Examinatrice
Simon de GIVRY, Chargé de recherche, MIAT - INRAE Toulouse	Rapporteur
Marie-José HUGUET, Professeure des universités, INSA Toulouse - LAAS-CNRS	Présidente

UNIVERSITÉ DE MONTPELLIER

Remerciements

Je tiens à exprimer ma profonde gratitude à toutes les personnes qui ont contribué, de près ou de loin, à la réalisation de cette thèse. Je remercie tout d'abord mon directeur de thèse, Christian Bessiere, pour son encadrement exceptionnel. Sa rigueur scientifique et son enthousiasme pour la recherche ont été une source d'inspiration profonde. Je suis également reconnaissant envers mon second encadrant, Clément Carbonnel, pour ses précieux conseils, son soutien constant et pour m'avoir poussé à toujours donner le meilleur de moi-même.

J'adresse ma sincère reconnaissance aux membres de mon jury, Gilles Audemard, Florence Dupin de Saint-Cyr, Simon de Givry et Marie-José Huguet, d'avoir accepté d'évaluer mon travail. Je remercie tout particulièrement Gilles Audemard et Simon de Givry pour leurs rapports détaillés et pertinents qui ont contribué à l'amélioration de cette thèse. Je remercie aussi l'ensemble du jury d'avoir fait le déplacement pour la soutenance et pour les discussions enrichissantes qui ont suivi.

Je remercie chaleureusement mes collègues et amis du laboratoire et de l'université, pour les discussions stimulantes et l'ambiance conviviale qui ont rendu cette expérience encore plus enrichissante autant scientifiquement qu'humainement. Je sais que cette thèse n'aurait pas été la même sans vous.

Merci à ma mère, mon père, mes sœurs et mon frère pour leur amour, leur soutien inconditionnel et leurs encouragements. Je sais que sans vous, je n'aurais jamais pu accomplir cela. J'aimerais aussi exprimer ma reconnaissance à tous mes amis proches, pour avoir toujours cru en moi et m'avoir soutenu dans les moments difficiles. Enfin, je remercie du fond du cœur Paula, ma compagne, qui a partagé ma vie durant toute cette aventure, pour son amour, sa patience et son soutien indéfectible tout au long de ces années. Ta présence est pour moi une source de motivation inestimable pour laquelle je ne saurais jamais assez te remercier.

Abstract

Constraint programming is a powerful paradigm for solving complex combinatorial problems. However, its adoption is often hindered by the modeling challenge that requires both domain expertise and a deep understanding of the constraint programming paradigm. Constraint acquisition aims to automate this modeling process by learning constraints from data. In particular, passive constraint acquisition focuses on learning from a provided batch of examples.

Current passive constraint acquisition methods, however, face two significant limitations that restrict their practical use. First, they require the user to provide a large dataset of examples. Second, they presuppose the user has prior knowledge about the problem, such as a list of potential constraints or a grammar to express them. This dissertation addresses these fundamental limitations by proposing novel passive constraint acquisition methods that operate without any prior knowledge and with a minimum of examples, while producing models that tend to be interpretable and to generalize well to unseen data.

Résumé en français

French Summary

La *Programmation Par Contraintes* (PPC) est un paradigme puissant pour résoudre des problèmes combinatoires complexes, trouvant des applications dans des domaines variés. Un de ses atouts majeurs est la séparation entre la modélisation du problème (la définition d'un ensemble de variables et de contraintes) et sa résolution (la recherche de solutions par un solveur générique). Cependant, cette phase de modélisation constitue souvent un obstacle majeur pour l'utilisation de la PPC. Elle requiert une double expertise : une connaissance approfondie du domaine du problème à modéliser et une maîtrise des concepts de la PPC pour formuler des contraintes pertinentes. Cet obstacle limite l'adoption de la PPC par des non-experts.

Pour surmonter cette difficulté, le domaine de l'*acquisition de contraintes* a émergé. Son objectif est d'automatiser le processus de modélisation en apprenant un réseau de contraintes directement à partir de données, souvent des exemples de solutions (assignations de valeurs aux variables qui devront satisfaire l'ensemble des contraintes cibles) et de non-solutions (assignations qui ne devront pas satisfaire l'ensemble des contraintes cibles). Néanmoins, la plupart des méthodes d'acquisition de contraintes existantes souffrent d'une limitation majeure : elles exigent des connaissances préalables sur la structure du problème à apprendre. L'utilisateur doit fournir, en plus des exemples, un ensemble de contraintes candidates potentielles ou un langage de contraintes fixe (une grammaire) dans lequel le modèle final doit être exprimé. Cette exigence réintroduit une barrière et rend ces méthodes difficilement applicables dans des scénarios où cette connaissance est absente, par exemple lorsqu'on apprend d'une base de données de solutions précédemment validées ou d'un classifieur "boîte noire" comme un réseau de neurones.

La présente thèse s'attaque à ce verrou en proposant des méthodes d'acquisition de contraintes qui réduisent progressivement la dépendance aux connaissances a

priori, avec pour objectif de rendre le processus plus autonome et plus généralisable à de nouvelles données.

La première contribution de cette thèse est une nouvelle méthode d'acquisition de contraintes, nommée LFA, qui élimine la nécessité de fournir un langage de contraintes prédéfini. L'idée centrale est d'apprendre simultanément le réseau de contraintes et un langage adéquat pour l'exprimer. Pour éviter de générer des langages arbitrairement complexes et de ne pas obtenir un modèle surajusté aux données d'entrainement, notre approche est guidée par un principe de simplicité : elle cherche le langage le plus simple capable d'expliquer les exemples fournis. La simplicité est ici définie en termes de l'arité maximale des relations et du nombre total de relations dans le langage.

Nous avons formalisé ce problème et prouvé qu'il est NP-complet. Pour le résoudre en pratique, nous avons développé un algorithme qui explore les langages par ordre de complexité croissante et utilise un modèle de satisfiabilité maximale pondérée pour trouver, pour un langage donné, un réseau de contraintes cohérent avec les exemples (un réseau où chaque exemple de solution satisfait l'ensemble des contraintes, tandis que chaque exemple de non-solution en viole au moins une). Les évaluations expérimentales sur un large éventail de problèmes de référence (Sudoku, Jigsaw Sudoku, Planning d'Infirmières, etc.) démontrent que LFA est capable d'apprendre des réseaux de contraintes qui représentent le problème à modéliser sans aucune connaissance préalable du langage cible, constituant ainsi une avancée significative vers une acquisition de contraintes entièrement automatisée.

Bien que la méthode LFA résolve le problème de la dépendance à un langage prédéfini, nous avons également identifié une piste d'amélioration. LFA apprend une simple liste de portées de contraintes (les séquences de variables sur lesquelles les contraintes s'appliquent). La méthode peine ainsi à capturer la structure sous-jacente qui régit l'application de ces contraintes. Par exemple, dans un Sudoku, les contraintes de différence s'appliquent systématiquement à toutes les paires de cases d'une même ligne. LFA doit apprendre chacune de ces contraintes individuellement, ce qui peut nécessiter un grand nombre d'exemples pour éliminer les contraintes parasites et peut nuire à la généralisation.

Pour répondre à cette limite, notre seconde contribution consiste dans un premier temps à introduire une nouvelle représentation des réseaux de contraintes, que nous appelons *templates*. Un template est une structure qui capture la relation entre les

variables et les contraintes de manière plus compacte et interprétable. Il se compose de deux éléments :

- Des attributs associés aux variables, qui capturent leurs propriétés structurelles (par exemple, un attribut “ligne” et un attribut “colonne” pour chaque case du Sudoku).
- Des règles qui génèrent des contraintes en se basant sur ces attributs (par exemple, une règle stipulant : “appliquer la contrainte de différence à toute paire de variables partageant la même valeur pour l’attribut ‘ligne’”).

Avec cette nouvelle représentation, nous proposons une méthode d’acquisition de contraintes, nommée TAcq, qui apprend des templates à partir d’exemples. L’originalité de TAcq est qu’elle apprend non seulement les règles, mais aussi les attributs pertinents directement à partir des données, sans qu’ils soient fournis a priori. L’algorithme fonctionne comme une étape de raffinement : il prend en entrée un réseau de contraintes initial (par exemple, celui appris par LFA) et cherche un template permettant de représenter un large nombre de ses contraintes avec un template ayant peu d’attributs et de règles. Ce template, en capturant la structure intrinsèque du problème, favorise une meilleure généralisation et réduit drastiquement le nombre d’exemples nécessaires pour atteindre une haute précision. Nos expériences montrent que, pour de nombreux problèmes structurés (Sudoku, Planning d’Infirmières, Emploi du Temps d’examens, etc.), TAcq surpasse LFA en termes de précision, tout en produisant des modèles plus interprétables sans nécessiter de connaissances préalables, excepté les exemples.

Ensemble, ces contributions tracent une voie vers une acquisition de contraintes plus autonome. En levant d’abord la nécessité d’un langage de contraintes prédéfini, puis en introduisant un formalisme capable de capturer la structure inhérente des problèmes, cette thèse rend l’apprentissage de modèles de contraintes plus accessible et plus robuste. Nous présentons également des perspectives sur l’utilisation de réseaux de neurones convolutifs (CNN) comme oracles pour l’acquisition de contraintes. En exploitant les techniques d’explicabilité, nous envisageons d’exploiter de nouvelles données que les CNN peuvent fournir pour guider l’extraction de contraintes. Cette approche ouvre la voie à une nouvelle direction d’apprentissage de contraintes à partir de données non structurées, en utilisant des modèles d’apprentissage automatique comme sources de données.

Contents

Chapter 1
Introduction

Constraint programming is a powerful paradigm for solving complex combinatorial problems. It separates the modeling of the problem, expressed as a set of constraints, from the solving process. This separation allows for greater flexibility and adaptability, as the same solving techniques can be applied to different problems. However, the task of modeling a problem may require significant expertise in the field of the problem, as well as a deep understanding of the constraint programming paradigm.

A user of constraint programming must be able to identify the relevant constraints, express them in a suitable formalism, and ensure that the resulting model accurately captures the requirements of the problem. This often involves a steep learning curve and can be a significant bottleneck to entry for non-experts or those unfamiliar with constraint programming techniques. As a result, many potential users of constraint programming may find it challenging to effectively model their problems, leading to underutilization of this powerful approach in practice.

To address this modeling bottleneck, the field of constraint acquisition has emerged, and it consists in automatically learning a set of constraints that represent a given problem from data. However, two significant challenges limit its practical use. First, current methods often require the user to provide a large dataset of examples. Second, they presuppose the user has prior knowledge of the problem, such as a list of potential constraints or a grammar to express them. Assuming that the user has such prior knowledge is a major limitation, as this requirement may not always be realistic. This limitation is even greater in scenarios involving non-human sources of examples, like a database or a neural network, which may not have any attached background knowledge about the problem to learn.

The central problem tackled in this thesis is how to enable a constraint acquisition method to learn the constraints of a problem *without any prior knowledge* and with a *minimum of examples*, while producing models that tend to be interpretable and to generalize well to unseen data. In particular, we focus on passive constraint acquisition, where we learn from a provided batch of examples, without interaction.

To overcome the limitations of prior work, our contributions are based on the idea of promoting simplicity and compactness to learn constraint networks that generalize well to unseen data. This finally leads us to discuss the possibility of using constraint acquisition with non-traditional data sources rather than a human. In such scenarios, we can exploit new types of data that, for instance, a convolutional neural network with the help of explainability techniques, can provide to learn a constraint network.

Outline of the Thesis

Chapter 2: Background. This chapter presents the necessary context with an overview of constraint optimization (covering key concepts in constraint programming and Boolean satisfiability) and concept learning.

Chapter 3: Constraint Acquisition. This chapter presents the constraint acquisition problems and a short survey of some constraint acquisition methods. We highlight their limitations, setting the stage for our contributions.

Chapter 4: Language-Free Constraint Acquisition. This chapter presents our first contribution, a novel acquisition method named LFA that removes the need for prior knowledge of the constraint language. Our approach computes a suitable language as part of the learning process.

Publications *Learning Constraint Networks over Unknown Constraint Languages* – IJCAI 2023 [6] and *Apprendre un CSP sans connaître son langage* (French extended abstract) – JFPC-2024 [7].

Software All the data and code used in this chapter are available at github.com/hareski/language-free-acq. The method is implemented in the Python `languageFreeAcq` package, which can be installed with `pip install languageFreeAcq`.

Chapter 5: Learning Compact Representations of the Scopes. This chapter introduces a new compact representation of constraint networks called a template, which consists of attributes for variables and rules that generate constraints based on them. We then present the TAcq framework, a two-step pipeline that uses LFA to generate an initial network and then refines it into a template.

Publication This chapter is based on the conference paper *Learning Compact Representations of Constraint Networks* accepted for publication at ECAI 2025 (to appear).

Software All the data and code used in this chapter are available at github.com/hareski/tacq. The method is implemented in the Python `TAcq` package, which can be installed via `pip install tacq`.

Chapter 6: Perspectives on CNNs as Oracles. This chapter explores the potential of using convolutional neural networks (CNNs) as a source of examples for constraint acquisition. We discuss how explainability techniques may be used to guide the extraction of constraints from CNNs.

Chapter 7: Conclusion. This chapter concludes the manuscript by summarizing the key contributions and outlining promising directions for future research.

Chapter 2
Background

This chapter provides the necessary background for understanding the methods and concepts presented in this thesis. We will cover the topics of constraint optimization with constraint programming and Boolean satisfiability in Section 2.1 and then present the domain of concept learning in Section 2.2.

2.1 Constraint Programming and Satisfiability

We present in this section two main approaches relevant to this thesis: *constraint programming*, which is a powerful paradigm for solving CSPs, and *Boolean satisfiability*, a specialized but highly efficient case. We do not present all the approaches to constrained optimization; in particular, we omit other approaches such as linear programming and integer programming.

2.1.1 Constraint Programming

Constraint programming is a programming paradigm that splits the *modeling* part, where the problem is expressed as a *constraint network*, and the *solving* part, where a solution is found using a general-purpose solver. This approach, which splits the modeling phase from the solving phase, constitutes a fundamental characteristic of constraint programming and establishes it as a declarative programming paradigm. The focus is on defining the problem in terms of constraints rather than specifying how to solve it. Then, the solver leverages sophisticated techniques such as constraint propagation, backtracking, and heuristics to efficiently explore the solution space.

The initial works on constraint networks were introduced in [30] in 1974 and focused on binary constraints, which are defined on pairs of variables only. However, we consider the general case where constraints can be defined on sets of variables of arbitrary size.

Let us consider a fixed set of variables $X = \{x_1, x_2, \ldots, x_n\}$ and a finite domain $D = \{d_1, d_2, \ldots, d_m\}$, where each variable x_i can take any value from the domain D. A *relation* R of arity r over the domain D is a subset of D^r. We say that R *accepts* a tuple $\tau \in D^r$ if $\tau \in R$, and that R *rejects* a tuple $\tau \in D^r$ if $\tau \notin R$. While this definition is formal, in practice, many relations are defined intentionally by a property or predicate, rather than being explicitly listed as a set of tuples (e.g. the binary inequality relation $R_{\neq}$ defined as the set of pairs of distinct elements for any domain D).

A *constraint* over (X, D) is defined as a pair (R, S) where R is a relation of arity r over D (i.e., $R \subseteq D^r$) and S is a sequence of r variables from X, referred to as the *scope* of the constraint. An assignment $\alpha : X \to D$ *satisfies* a constraint (R, S) if $\alpha[S] \in R$; conversely, the assignment *violates* the constraint if $\alpha[S] \notin R$.

Definition 1 (Constraint network)**.** *A constraint network is a tuple $N = (X, D, C)$, where X is a set of variables, D is a finite domain, and C is a set of constraints over (X, D).*

A constraint network $N = (X, D, C)$ *accepts* an assignment $\alpha : X \to D$ if and only if α satisfies all constraints in C; N *rejects* it otherwise. An assignment that is accepted by N is called a *solution* to the constraint network. Conversely, an assignment that violates any constraint in C is a *non-solution*. Given a constraint network $N = (X, D, C)$, we say that a relation R is *applied* to a sequence of variables S if there exists a constraint (R, S) in C. Two constraint networks are *equivalent* if they have exactly the same set of solutions.

Definition 2 (Constraint Satisfaction Problem (CSP))**.** *Given a constraint network $N = (X, D, C)$, the Constraint Satisfaction Problem (CSP) is the task of determining whether a solution to N exists.*

Solving a constraint network consists in finding an assignment that is a solution to the network or proving that no such assignment exists. To solve a constraint

network, we employ a *constraint solver*, a specialized program designed to implement general-purpose algorithms for solving a constraint network.

Example 2.1.1 (Nurse Rostering as a Constraint Network)**.** *The Nurse Rostering problem is a classic example of a combinatorial optimization problem where constraint programming is effective. The goal of the Nurse Rostering problem is to create a work schedule, assigning a set of nurses to shifts over a given period (e.g., a week). A valid schedule must respect a set of rules that depend on the specific requirements of the schedule. For example, let us consider the following simple set of rules:*

- *Rule 1: Each shift must be covered by exactly one nurse.*
- *Rule 2: A nurse cannot be assigned to two different shifts on the same day.*
- *Rule 3: A nurse cannot work the last shift of a day and the first shift of the next day.*

To translate this specific problem into a constraint network, we must define its three components: the variables, their domains, and the constraints. Let us assume an instance with d days, s shifts per day, and n nurses.

Variables (X): *A natural way to model the problem is to define a variable for each assignment to be made. We create a variable $x_{i,j}$ to represent the nurse assigned to shift j on day i. The set of all variables is $X = \{x_{i,j} \mid i \in [1, d], j \in [1, s]\}$.*

Domain (D): *The domain of each variable is the set of possible values it can take. In this case, it is the set of available nurses. We can represent them with integers: $D = \{1, 2, \ldots, n\}$.*

The fact that each shift (i, j) is represented by a unique variable $x_{i,j}$ that must take a single value from the domain D implicitly ensures that each shift will be covered by exactly one nurse (Rule 1). This highlights that some choices of representation have an impact on the constraints that need to be defined. We now translate the remaining rules into constraints.

Constraints (C):

- *Rule 2: To ensure that a nurse does not work two different shifts on the same day, we must state that for any given day i, the values of the variables $x_{i,j}$ and $x_{i,k}$ (with $j \neq k$) must be different. Using the binary disequality relation $R_{\neq}$, we can add the following set of constraints to C:*

$$\{(R_{\neq}, (x_{i,j}, x_{i,k})) \mid i \in [1, d], j, k \in [1, s], j \neq k\}$$

- *Rule 3: To ensure that a nurse cannot work the last shift of a day and the first shift of the next day, we can add the following set of constraints to C:*

$$\{(R_{\neq}, (x_{i,s}, x_{i+1,1})) \mid i \in [1, d-1]\}$$

The constraint network $N = (X, D, C)$ can be produced with fixed parameters (d, s, n) of a specific instance of the Nurse Rostering problem. An assignment $\alpha : X \to D$ that is a solution of N represents a valid schedule that adheres to all the stated rules.

There are no general limitations on the types of constraints that can be defined in a constraint network. However, some constraints are more commonly used than others, such as binary arithmetical constraints defined on pairs of variables (e.g. equality, inequality, comparison) or constraints such as AllDifferent which ensures that all variables in a set take different values, and NotAllEqual which ensures that at least two variables in a set take different values. The relations that are used to define the constraints in a constraint network can be of various types and are referred to as the *constraint language* of the network.

Definition 3 (Constraint language)**.** *A* constraint language Γ *is a set of relations over a finite domain.*

A constraint network N is said to be over the constraint language Γ if every relation R in the constraints of N belongs to Γ. The arity of a constraint language is defined as the maximum arity among all relations contained in Γ.

2.1.2 Boolean Satisfiability

Boolean satisfiability (SAT) is a fundamental problem in computer science and logic, where the goal is to determine whether there exists an assignment of truth values to a set of Boolean variables such that a given Boolean formula evaluates to `True`. Its central importance in computational complexity was established in the seminal work of Stephen Cook, who proved it was the first problem shown to be NP-complete [13]. Beyond its theoretical importance as the canonical NP-complete problem, SAT has found widespread practical applications across numerous domains.

Let us first introduce the necessary terminology to formalize the SAT problem. A *literal* is either a Boolean variable x or its negation $\neg x$. To facilitate notation during the manuscript, we extend the domain of a truth assignment α to literals. Specifically, for any variable x and its negation $\neg x$, we define $\alpha(\neg x) = \neg\alpha(x)$. The notation $\alpha(l)$ denotes the truth value assigned to literal l under the assignment α.

A *clause* is a disjunction (logical OR) of literals, typically written as $l_1 \vee l_2 \vee \cdots \vee l_k$ where each l_i is a literal. A clause is *satisfied* by a truth assignment α if at least one of its literals evaluates to `True` under that assignment. Given an assignment α and a clause C_i, we denote $\alpha(C_i)$ the truth value assigned to clause C_i under the assignment α, i.e. $\alpha(C_i) =$ `True` if and only if at least one literal in C_i is assigned `True` by α.

Definition 4 (Boolean Satisfiability Problem (SAT))**.** *Given a finite set of clauses $CL = \{C_1, C_2, \ldots, C_m\}$ over a set of Boolean variables $X = \{x_1, x_2, \ldots, x_n\}$, the SAT problem asks whether there exists a truth assignment $\alpha : X \to \{\texttt{True}, \texttt{False}\}$ such that $\alpha(C_i) =$ `True` for every clause $C_i \in CL$.*

The SAT problem can be viewed as a special case of constraint networks where all the variables are Boolean variables (i.e. the domain is restricted to $\{1, 0\}$) and the constraint language consists exclusively of Boolean clauses. However, all these limitations allow us to use specific and efficient algorithms to solve the problem, such as the DPLL algorithm [16] and its modern variants like CDCL (Conflict-Driven Clause Learning) solvers [38]. Modern SAT solvers are capable of handling instances with millions of variables and clauses. This leads us to consider SAT as an interesting way to model and solve some combinatorial problems we will encounter in this thesis.

In Chapter 4 of this thesis, we will more specifically use the WEIGHTED PARTIAL MAX-SAT problem, a generalization of SAT, where some clauses must be satisfied

(*hard clauses*) and others can be satisfied or not (*soft clauses*). The goal is to satisfy all hard clauses while maximizing the sum of the weights of satisfied soft clauses.

Definition 5 (Weighted Partial Maximum Satisfiability Problem (WEIGHTED PARTIAL MAX-SAT))**.** *Given a set of hard clauses CL_H and a set of soft clauses CL_S, where each clause c_i in CL_S has a weight w_i, the* WEIGHTED PARTIAL MAX-SAT *problem asks for a truth assignment α that satisfies all clauses in CL_H and maximizes the sum of weights of satisfied clauses in CL_S.*

2.2 Concept Learning

2.2.1 Concept Learning Task

Definition 6 (Concept)**.** *Given a set of variables X and a finite domain D, a* concept *is a function $c : D^X \to \{0, 1\}$.*

A *concept class* is a set of concepts. The output of a concept is often interpreted as a classification label, where 1 indicates that the assignment belongs to the *positive* class and 0 indicates that it belongs to the *negative* class.

One may notice that the definition of concept and concept class does not specify how the function or the set must be described or represented. There are various formalisms that can be used to represent concepts, such as decision trees, logical formulas, or neural networks. An example of a possible representation for a concept is a constraint network. The concept is defined by a constraint network $N = (X, D, C)$, where X is a set of variables, D is a finite domain, and C is a set of constraints over the variables in X. Any assignment $\alpha \in D^X$ can be evaluated by checking whether the assignment satisfies all constraints in C. If it does, we output 1 (positive); otherwise, we output 0 (negative). Following this idea, a way to describe a concept class could be to specify all the constraints that can be used in the constraint network that describes a concept in the class.

Given a set of variables X and a finite domain D, an *example* e over (X, D) is an assignment $\alpha \in D^X$ along with a label $b \in \{0, 1\}$. The label indicates whether it belongs to the positive class (1) or the negative class (0). We denote $\alpha(e)$ the assignment of the example e and $b(e)$ its label.

Definition 7 (Training Set)**.** *A training set E is a finite set of examples.*

We say that the training set E is drawn from a distribution $\mathcal{D}$ over the assignments of X if the assignments in E are drawn from the distribution $\mathcal{D}$. We say that a concept c is *consistent* with a training set E if, for all assignments $e \in E$, $c(\alpha(e)) = b(e)$. In other words, a concept is consistent with a training set if it classifies all examples in the training set as they are labeled.

The *concept learning task* [29] consists in inferring a concept from a training set. Typically, a concept learning task can be described by a concept class H called the *hypothesis space*, and a training set E with the goal of finding a hypothesis $h \in H$ consistent with E. To define a concept learning task as a formal problem, we must specify an encoding for the concept class H. For example, the concept learning problem formally defined in Chapter 1 of [28] takes as input a language to describe a concept and a procedure that matches concept descriptions to (consistent) training sets (these two first inputs define the hypothesis space) and a training set E. The goal is to find a concept description within the given language that is consistent with the provided training set (i.e. the procedure matches this description with E). An algorithm that solves a concept learning problem is called a *learning algorithm*. A concept learning task is often related to a *classification task*, where the goal is to assign labels to assignments based on the learned concept.

Example 2.2.1 (A Simple Concept Learning Task)**.** *Let $X = \{x_1, x_2, x_3\}$, $D = \{0, 1\}$ and a training set E with the following examples:*

- $e_1 : \{(x_1, 1), (x_2, 0), (x_3, 0)\}$ *with label* 1
- $e_2 : \{(x_1, 0), (x_2, 1), (x_3, 0)\}$ *with label* 1
- $e_3 : \{(x_1, 0), (x_2, 0), (x_3, 1)\}$ *with label* 0

Let the concept class H of all concepts such that each concept h has a representation as a unique binary clause over the variables X and such that $h(\alpha) = 1$ if and only if α satisfies the clause (the concept returns the truth value (0 or 1) of the clause over the assignment α).

Consider the concept learning task of finding a hypothesis $h \in H$ that is consistent with the training set E. A possible representation using a clause of a concept in H that is consistent with the training set E is the clause $x_1 \vee x_2$. But the training set E does not allow us to uniquely determine a clause. In fact,

> *there are multiple concepts in H that are consistent with the training set. For example, the concept of $x_1 \vee \neg x_3$ also represents a concept that is consistent with the training set.*

As illustrates in the example above, given a specific concept learning task, the training set may not be sufficient to uniquely determine a concept.

2.2.2 Generalization to Unseen Assignments

It is important to note that while the concept learning task is often to determine a hypothesis h that is consistent with the training set E, other criteria may also be important. Generally, the hypothesis should ideally match a *target concept* c, the underlying concept c that has generated the training set. However, this target concept is unknown. The learning algorithm only has access to the training set.

An underlying goal of a concept learning task is, therefore, often to find a concept that generalizes well to unseen assignments (i.e. that classifies new assignments as the target concept does). This is often referred to as the *generalization* ability of a learning algorithm. Since not all assignments are equally important for a given classification task, the generalization ability is measured by the *error* of the concept under a fixed distribution $\mathcal{D}$ over the assignments of X. The error of a concept h with respect to a target concept c and a distribution $\mathcal{D}$ is defined as the probability that the concept misclassifies an assignment drawn from the distribution $\mathcal{D}$:

$$\text{error}(h, c, \mathcal{D}) = \Pr_{e \sim \mathcal{D}}[h(e) \neq c(e)]$$

In contrast, the *accuracy* is defined as the fraction of assignments that are correctly classified by the concept ($\text{accuracy}(h, c, \mathcal{D}) = 1 - \text{error}(h, c, \mathcal{D})$). However, evaluating the error of a given concept is not straightforward, as it depends on an unknown target concept and an unknown distribution $\mathcal{D}$. During experimental evaluation, the error can be approximated by testing the concept on a separate test set drawn from the same distribution $\mathcal{D}$ that the training set E was drawn from. This is based on the (quite strong but classic in ML [20]) assumption that the training set is representative of the distribution $\mathcal{D}$ that the concept is expected to generalize to. The error is estimated as the fraction of assignments in the test set that are misclassified by the concept.

A phenomenon is related to the generalization ability of the learned concept and can significantly impact the performance of the learning algorithm: *overfitting*. A learning algorithm overfits to a training set E when it learns a concept that is overly specific to the particular examples in E, resulting in poor generalization to unseen assignments. In other words, the learned concept captures the training set too closely and fails to generalize to new assignments.

In practice, the concept learning process can be subject to *noise* in the training set. Noise refers to inconsistencies in the training set that deviate from the unknown target concept. The presence of noise in training sets exacerbates the overfitting problem. When training examples contain labeling errors, a perfectly consistent concept may actually represent the noise rather than the true target concept. In such cases, allowing some inconsistency with the training set in the learning task may lead to better generalization performance. A learning algorithm that is able to determine the target concept despite the noise is said to be *robust* to noise.

Chapter 3
Constraint Acquisition

The modeling of constraint networks is a crucial step in constraint programming. It involves defining all the constraints to describe a specific problem. This process is often done manually by experts in constraint programming, who have the knowledge and experience to define the constraints that capture the problem. However, this process can be difficult and time-consuming, often requiring expert knowledge both in constraint programming and the specific problem domain.

To address this challenge, the field of *Constraint Acquisition* (CA) has emerged, aiming to automate the modeling of constraint networks. Given data about a problem, CA methods learn information about how to model the problem using constraints. In this chapter, we will present the domain of constraint acquisition and its relation to concept learning. We will then introduce the main approaches to constraint acquisition and discuss the challenges and limitations of these approaches.

3.1 Oracle and Training Set

While the term "user" is frequently employed in the literature to describe the entity that supplies data or feedback for constraint acquisition, this terminology may implicitly suggest a human operator interacting with the system. However, the term user can be somewhat limiting, as it does not encompass the full range of entities that might play this role. To address this limitation, we will instead employ the term *oracle* to denote the entity that provides data or feedback within the constraint acquisition context. Additionally, "oracle" is commonly employed in machine learning to refer to an entity that provides answers or information. The term is quite general

and can refer to any system that provides information about the problem. Some examples of oracles include:

- a human expert that can provide solutions and non-solutions to the problem or answer questions about it;
- some historical data of the problem. For instance, a dataset of assignments that have been previously proven to be solutions;
- any automated system. For example, a neural network can be used as an oracle to classify new assignments if we suppose that the neural network has learned the underlying structure of the problem;
- a simulation or a model of the problem that can provide feedback on the correctness of assignments. For instance, a simulation of a physical system can be used to provide feedback on the correctness of assignments representing the state of a system.

We distinguish between two types of constraint acquisition methods. When the whole available data is provided in advance, the process is referred to as *passive constraint acquisition*. When the data is collected through interaction with the oracle, it is referred to as *active constraint acquisition*.

In active constraint acquisition, constraint acquisition methods need to actively query the oracle to obtain data. In this case, one of the main challenges is to decide on the queries to ask the oracle in order to learn the model of the target constraint problem as efficiently as possible. Meanwhile, in passive constraint acquisition, the complete data available is already given, and the focus is only on exploiting this data. In this manuscript, we choose to refer to an oracle whether the data is provided in advance or collected through interaction with it. This maintains a consistent terminology throughout the text, as the oracle is the entity that provides data or feedback, regardless of the method used for constraint acquisition.

The oracle can provide various types of data, each leading to different constraint acquisition approaches. The most common type consists of a training set of examples as presented in the previous chapter. A constraint network N is said to be *consistent* with a training set E if all the assignments of positive examples satisfy all the constraints of N, and all the assignments of negative examples violate at least one constraint of N.

3.2 Version Space

Constraint acquisition methods do not learn arbitrary constraint networks. Instead, they operate under assumptions that restrict the search space to a manageable subset of all possible constraint networks. This restriction is essential to ensure that the learned constraint network does not overfit to the training set and can generalize to unseen assignments. Without such assumptions, any training set could trivially be represented as a constraint network containing a single constraint whose scope encompasses all variables and whose relation consists solely of the positive examples in the training set. While this network would achieve perfect consistency with the training data, it should provide no generalization capability, as it would only accept the specific assignments present in the training set and reject all others.

The search space can be restricted in various ways. Mainly, one can use a fixed set of relations (i.e. a constraint language) from which the constraints can be defined. For instance, the relations can be restricted to some binary arithmetical relations (e.g. equality, inequality, ...). The search space can also be limited by fixing a set of candidate scopes (i.e. the variables involved in the constraints) from which the constraints are defined. For instance, the scopes can be restricted to pairs of variables that are adjacent in some topological structure (e.g. a grid, a graph, ...). With these restrictions, a set of candidate constraints $\mathcal{C}_{cand}$ can be defined, which is a set of constraints that can be used to build the constraint network. The candidate constraints are typically defined over a fixed set of variables X and a finite domain D.

Definition 8 (Version Space)**.** *Given a set of variables X, a domain D, a set of candidate constraints $\mathcal{C}_{cand}$ all over (X, D), and a training set E, the* version space *is the set of constraint networks (X, D, C) where $C \subseteq \mathcal{C}_{cand}$ that is consistent with all examples in E and is denoted as $VS(E, \mathcal{C}_{cand})$.*

The version space $VS(E, \mathcal{C}_{cand})$, when non-empty, is fundamentally bounded by two types of constraint networks, delineating the boundaries of consistent hypotheses within the predefined candidate constraints.

On one hand, there exists a unique *most specific network* consisting of constraints denoted as S, which represents the most restrictive constraint network that remains consistent with the given training set E. This network (X, D, S) is constructed by

taking the conjunction of all individual candidate constraints from $\mathcal{C}_{cand}$ that are satisfied by every assignment of positive example in E. Consequently, the network (X, D, S) implicitly rejects the maximum possible number of assignments within the complete variable domain D^X (including all assignments of negative examples in E), while ensuring that all positive examples from E are solutions.

Conversely, the version space also contains one or multiple *most general networks* composed of constraints denoted as $G_1, G_2, \ldots, G_k$. These networks represent the least restrictive, yet still consistent, hypotheses within the candidate constraints. A network (X, D, G_i) is characterized as most general if and only if it is consistent with the training set, and the removal of any constraint from G_i would cause the resulting network to accept the assignment of at least one negative example. In essence, these networks are minimal conjunctions of constraints, where each constraint is indispensable for ensuring consistency with the negative examples.

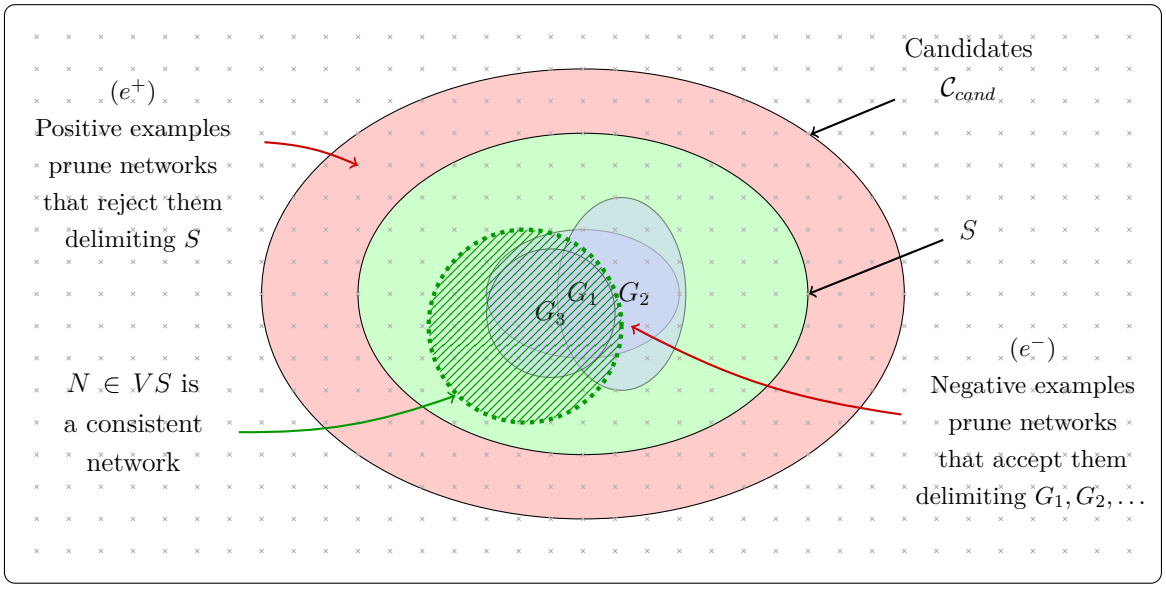


Figure 3.1: Illustration of the version space over a set of candidate constraints $\mathcal{C}_{cand}$: each gray cross represents a constraint. The arrow (e^+) indicates the constraints pruning of the version space based on information provided by the positive examples. The arrow (e^-) indicates the pruning of the version space based on negative examples (with multiple sets $G_1, G_2, \ldots$). A consistent network must contain all constraints of at least one most general networks $G_1, G_2, \ldots, G_n$ and be a subset of the most specific network S.

As illustrated in Figure 3.1, the version space can be represented using the set

of constraints in S and the most general networks $G_1, G_2, \ldots, G_k$. A consistent constraint network $N = (X, D, C) \in VS(E, \mathcal{C}_{cand})$ is a network that is a subset of the most specific (i.e. $C \subseteq S$) and contains all the constraints of at least one of the most general networks (i.e. $C \supseteq G_i$ for some i).

Each network in the version space is a constraint network that is consistent with the training set. If new examples are provided by the oracle, they are used to prune the version space. A positive example eliminates all candidate networks that contain a constraint that would reject the assignment of the positive example. Conversely, a negative example eliminates all candidate networks that would accept the assignment of the negative example. The goal of a constraint acquisition algorithm is to reduce the version space, ideally to a single network, which would then be the learned target network. If the version space becomes empty, it implies that no network formed from the candidate constraints can explain the training set, suggesting that the candidate constraints are insufficiently expressive or that the training set contains noise.

3.3 Constraint Acquisition

The passive constraint acquisition domain was initially presented in [10, 12]. Given a set of candidate constraints $\mathcal{C}_{cand}$ and a training set E, three fundamental problems can be formulated:

- *Version space problem:* give a representation of the version space $VS(E, \mathcal{C}_{cand})$;
- *Consistency problem:* determine whether there exists a constraint network that is consistent with the examples (Is $VS(E, \mathcal{C}_{cand})$ not empty?);
- *Convergence problem:* determine whether the version space converges to constraint networks all representing the same target concept (i.e. whether the version space contains only equivalent networks).

The consistency problem can be solved in polynomial time, and the convergence problem is CoNP-complete [12].

In this manuscript, we consider *passive constraint acquisition* as a specific concept learning task. The task consists in inferring a constraint network from a training set.

Typically, the candidate constraint networks are limited by a hypothesis space that is represented by a set of candidate constraints $\mathcal{C}_{cand}$ (often called the bias), and the inferred constraint network must be consistent with the training set.

When we evaluate the generalization ability of a learning algorithm for a passive constraint acquisition task, we will consider that there exists a unique and unknown (to the learning algorithm) constraint network $N^* = (X, D, C^*)$ that has been used to label the training set E. We will refer to it as the *target constraint network* of the learning task.

This presentation of the passive constraint acquisition task highlights a central challenge inherent in concept learning tasks. The training set E is typically only a small sample of the entire assignment space D^X and, as a result, is insufficient to uniquely determine a network consistent with E. This naturally poses a challenge for the learning process: what is the "best" network to choose from this space? Which network will generalize best to unseen examples?

Constraint acquisition has been applied in various domains, including automated program analysis [27] and robotics [31]. Moreover, tools like PyConA [42] have been developed to facilitate the use of various constraint acquisition methods. PyConA is a Python library based on CPMpy [19], which provides a framework for implementing and experimenting with different constraint acquisition algorithms.

3.4 Overview of Constraint Acquisition Methods

This section provides an overview of key constraint acquisition methods from the literature, highlighting their main characteristics. We will also discuss some limitations of each method.

ConAcq.1 [10, 12] ConAcq.1 learns from a set E of both positive and negative examples with a fixed set C_{cand} of candidate constraints. It produces a Sat formula that is a representation of the full version space $VS(E, C_{cand})$. Each candidate constraint is represented by a Boolean variable, and each example is encoded as a set of clauses. This allows it to determine whether the version space is empty (consistency problem) by checking the satisfiability of the Sat formula. It is also possible to output the most specific network, any most general networks $G_1, G_2, \ldots,$

or any consistent network $N \in VS(E, C_{cand})$. The paper also proposes a way to determine whether the version space converges to a single concept (convergence problem).

ConAcq.2 [12] is an active extension of ConAcq.1 that actively queries the oracle with "membership queries". A membership query asks the oracle whether a given assignment is a solution of the problem or not. The algorithm aims to maximize the information gained from each query. It presents heuristics to select the most informative assignment to query, based on the current version space. The algorithm iteratively prunes the version space by querying for new examples and updating the Sat formula until it converges to a single concept or reaches a predefined stopping criterion.

QuAcq [9] is an active constraint acquisition method that, given a fixed set of candidate constraints, actively queries the oracle for "partial queries". A partial query presents the oracle with a partial assignment (an assignment of values to a subset of the variables) and asks whether this assignment violates a constraint whose scope is fully assigned. That is, the oracle must determine if the partial assignment violates a constraint whose scope is contained within that specific subset of variables. Crucially, this query does not ask if the partial assignment can be extended to a full solution, but only about its own local validity. This type of query is more informative than a full assignment query, as it allows the algorithm to focus on the scopes of a specific candidate constraint. However, this approach assumes that the oracle possesses a deep knowledge of the problem representation as a constraint network, allowing it to evaluate the local validity.

GenAcq and Mine&Ask [8, 15] GenAcq is a technique that can be integrated into any constraint acquisition method that tries to infer new constraints over the scopes of variables of the same "type". It achieves this by querying the oracle with "generalization queries", which prompt the oracle to decide if the relation of a learned constraint can be applied to other variables of the same type as those in the original constraint. This approach requires providing the variable types in advance. G-QuAcq is the name of the GenAcq algorithm when integrated into the QuAcq method. Mine&Ask addresses the need for providing variable types in GenAcq by

learning the potential types of variables directly during the constraint acquisition process. It tries to find a community structure that is a group of variables with dense connections (i.e. many constraints between them) internally and with sparser connections with other variables. Thus, the method is efficient only if the problem has community structure.

ClassAcq and BayesAcq [35] ClassAcq is a passive constraint acquisition method. The core idea is to first train a classifier to discriminate between positive and negative examples, and then to derive a constraint model directly from the trained classifier. BayesAcq is an implementation of the ClassAcq approach that uses a naïve Bayes classifier. The algorithm relies on a fixed set of candidate constraints C_{cand} and a training set E and treats each candidate as independent. For each $c \in C_{cand}$, it calculates a score K_c, which is the ratio of the probability that c is violated by the assignment of a negative example of E in to the probability that it is violated by the assignment of a positive example of E (this ratio allows BayesAcq to be robust to noise in the training examples). This score is interpreted as the "weight of evidence" that the candidate is a constraint in the target network. A candidate c is learned as a constraint if its score K_c exceeds a predefined threshold. However, it relies on the set of candidate constraints which must be expressive enough to capture the target constraints and not too large to avoid overfitting.

Model Seeker [5] Model Seeker is a passive constraint acquisition method designed to find models for highly structured problems by learning from positive examples. The hypothesis space is defined by relations of a large "catalog" [3, 4] which contains high-level predefined relations such as AllDifferent or Lex that are applied over scopes on predefined topologies of the variables, like a matrix (e.g. applying AllDifferent over the rows and columns of a matrix). The algorithm searches through these candidate constraints to find those that are satisfied by all assignments of provided positive examples. It then presents a ranked list of candidate constraints to the user, allowing them to select the most appropriate ones for their problem. The primary limitation of the Model Seeker method is that it is effective only for problems with a common regular structure, as the scopes are limited to predefined topologies. Additionally, its success is entirely dependent on the expressiveness of the relations catalog: if a necessary relation for a constraint is not

present in the catalog, it cannot be learned. On the other hand, because the method relies on high-level relations and predefined topologies, the learned constraints can often be directly used over different instances of the same problem.

COUNT-CP [21] COUNT-CP is a passive constraint acquisition method that learns from positive examples constraints of the form $lb \leq expr \leq ub$ with $expr$ an expression over the variables in the problem and lb and ub are symbolic bounds (i.e. they can be expressions that depend on the problem instance parameters). The algorithm learns the expression over a predefined grammar. The algorithm works in two steps: first, it learns individual constraints that satisfy all positive examples, and then it groups these individual constraints to form first-order constraints over predefined sets of variables (e.g. matrix or user-provided partitions like the edges of a graph). Finally, it employs a filtering step to remove trivially true rules and rules that describe redundant constraints or overly restrictive. A limitation of COUNT-CP lies in the choice of the grammars, which must be expressive enough to capture the target constraints, and it lies in the partitions of the variables, which must be predefined.

URPILS [47] learns constraints from positive examples E using the Minimum Description Length (MDL) principle. URPILS attempts to compute a network N that minimizes the cost of describing both N and E given N, where the encoding cost of E given N is proportional to the number of solutions of N. The method is robust to noise by allowing inconsistencies with the training set E, which must also be encoded. Constraint networks are built over Boolean variables $0, 1$ and represented through "objects" (e.g. Sudoku cells and values). Predefined object "relations" (e.g. row, column) are used to express constraints as first-order logic rules over a fixed grammar. URPILS performs greedy search with a solution count approximation, iteratively adding constraints to reduce encoding cost. Its effectiveness depends drastically on the chosen grammar and on the predefined "objects" and "relations".

This review of existing methods highlights a significant reliance on a predefined set of candidate constraints and, in many cases, a need for provided structural information (e.g., variable types in GenAcq, topologies in Model Seeker, predefined relations

between variables in COUNT-CP and URPILS). This reliance on prior knowledge is a limitation, as it may not always be available.

Chapter 4
Learning over Unknown Constraint Languages

This chapter addresses a fundamental limitation in constraint acquisition: the requirement for prior knowledge of the target constraint language. Constraint acquisition methods assume that either a set of candidate constraints, a grammar for generating constraints, or a fixed constraint language is provided in advance. This assumption significantly restricts their practical applicability, especially when the oracle is a human user without expertise in constraint programming or when dealing with black-box systems that lack explicit knowledge.

This chapter is primarily based on the paper [6] published in the Proceedings of the Thirty-Second International Joint Conference on Artificial Intelligence (IJCAI-23).

Contents

4.1 Introduction

This chapter presents a general approach to constraint acquisition that requires only that the oracle provide a set of solutions and non-solutions of the target constraint network, while also addressing the overfitting problem. Instead of providing a set of candidate constraints, we compute a suitable constraint language as part of the learning process. Our method is based on the idea that the best constraint language is the simplest one, which we define in terms of the maximum arity and number of relations. This notion of simplicity allows our method to search for a constraint network over an unknown constraint language that is consistent with the training set while avoiding the pitfalls of overfitting.

We first define the decision version of the constraint acquisition problem over an unknown constraint language when the maximum number k of relations in the unknown language and the maximum arity r of its relations are given. We prove that the problem is NP-complete, even when $k = r = 1$. Given a number k of relations and an arity r, we encode the problem of finding a constraint network over an unknown language as a partial maximum satisfiability problem. We then propose a method to solve the problem of finding a constraint network over an unknown language when k and r are not given. We only need to assume a preference order on the pairs (k, r), and we search for the smallest pair (k, r) such that a constraint network over an unknown language of size k and arity r is consistent with the training set. We finally validate our method by analyzing its behavior on a set of benchmark problems.

Closely Related Work. There already exist two approaches that could be considered as partially addressing the constraint language assumption, though they still

require significant prior knowledge. ARNOLD learns integer programs from examples of feasible solutions by generating potential constraints that only include sums, products, and comparisons among terms [22]. The generation of constraints follows a general-to-specific order, and collects those that are satisfied by the assignments of example solutions in order to produce an integer program. The constraint acquisition COUNT-CP uses a grammar to generate constraints from examples of solutions and non-solutions, allowing it to learn a constraint network that is consistent with the training set. This grammar is able to express a wide range of "counting" constraints (see [11]), but is not able to express arbitrary constraints. Thus, both approaches still require substantial background knowledge about the target constraint language in the form of predefined grammars, which limits their applicability when the appropriate constraint types are unknown a priori.

The rest of this chapter is organized as follows. In Section 4.2, we present the formalization of the constraint acquisition problem over an unknown constraint language, and prove that it is NP-complete. In Section 4.3, we describe our method. In Section 4.4, we study the effectiveness of our method through several experiments. In Section 4.5, we discuss the limitations of our method, and outline future research directions. Finally, in Section 4.6, we conclude with a summary of our contribution.

4.2 Language Acquisition

We consider the constraint acquisition problem without any language of constraints or a set of candidate constraints given as input data. A straightforward approach to eliminate prior knowledge may be to apply a constraint acquisition method that needs candidate constraints, such as CONACQ, while providing all the possible constraints over the target network variables as candidate constraints. However, this approach is not practical because the number of possible constraints grows exponentially with the number of variables and their domains. Specifically, for n variables with domains of size d, the constraint space includes $n \times 2^d$ unary constraints, $\frac{n(n-1)}{2} \times 2^{d^2}$ binary constraints, and so forth. This renders this approach computationally impractical.

To partially address this issue, we can fix the maximum arity r of the candidate constraints. To keep the same expressiveness, we can consider all possible arities

iteratively and first learn unary constraints, then binary constraints, and so on, until we find a constraint network that is consistent with the training set. This approach still suffers from the exponential growth of the number of candidate constraints, which makes it impractical for large arities or domains, but could be feasible for small arities.

However, it still faces the most important practical problem: overfitting. If too many candidate constraints are provided, the method may learn a constraint network that overfits the training data, failing to generalize to unseen examples. Any approach that provides a large number of candidate constraints risks overfitting if it does not include a mechanism to prevent this issue.

Rather than working with a fixed constraint language, we propose to compute a suitable constraint language Γ alongside a constraint network over Γ that is consistent with the training set.

An important difficulty with this approach is that there may exist a large number of constraint languages that are consistent with a given training set. Some of these languages are clearly unsatisfactory from a practical point of view. For example, every training set over n variables is trivially consistent with a constraint network over a constraint language of arity n and size 1. This constraint network has a single constraint covering all variables. Its satisfying assignments are exactly the positive examples in the training set.

Our intuition is that the best constraint language is the simplest. Because "simplicity" is difficult to define formally, we instead consider (as a rough approximation) that the best language is the *smallest* in terms of its maximum arity and number of relations. This leads us to the following definition for the constraint acquisition problem without language.

Definition 9 (Language-Free Acq)**.** *Given a training set E and two natural numbers k and r, the problem Language-Free Acq asks whether there exists a constraint network over a language of size at most k and arity at most r consistent with E.*

In practice, we will solve an optimization and search variant of this problem, in which we attempt to find a constraint network with minimum (k, r) that is consistent with E. The problem is multi-objective (both the language arity and size must

be minimized), so multiple strategies are possible: for example, one could define a real-valued cost function $f(k, r)$ to be minimized or compute a Pareto front. The next theorem states that Language-Free Acq is NP-complete even when $(k, r) = (1, 1)$, so solving the optimization/search variant is likely to be difficult regardless of the chosen strategy.

Theorem 1. *Language-Free Acq is NP-complete even when $k = r = 1$.*

Proof. We first prove membership in NP. Suppose that there exists a constraint network $N = (X, D, C)$ over a language $\Gamma = (R_1, \ldots, R_k)$ of arity r and size k that is consistent with E. We can further assume that each constraint rejects at least the assignment of one negative example that is accepted by all other constraints, in which case N has at most $|E|$ constraints. We specify each relation $R_j \in \Gamma$ succinctly by listing only the tuples $t \in R_j$ such that there exists a constraint $c = (R_j, S)$ and an example $e \in E$ such that $\alpha(e)[S] = t$; the number of such tuples is at most $|C| \cdot |E| \leq |E|^2$. This succinct representation of N has polynomial size (even when r, k are part of the input) and can be checked for consistency with E in polynomial time, so Language-Free Acq $\in$ NP.

In order to prove NP-hardness, we reduce Sat to Language-Free Acq. Let $CL = \{Z_1, Z_2, \ldots, Z_m\}$ be a set of m clauses over a set $V = \{v_1, v_2, ..., v_n\}$ of n Boolean variables. We define a training set E over the set of variables $X = \{x_v \mid v \in V\} \cup \{x_{\neg v} \mid v \in V\}$ of domain $D = \{v \mid v \in V\} \cup \{\neg v \mid v \in V\} \cup \{\star\}$ such that:

- $e^+_\star \in E^+$ such that $\forall x \in X, \alpha(e^+_\star)[x] = \star$
- $\forall v \in V, e^+_v \in E^+$ such that:
$$\forall x \in X, \alpha(e^+_v)[x] = \begin{cases} \neg v & \text{if } x = x_v \\ v & \text{if } x = x_{\neg v} \\ \star & \text{otherwise} \end{cases}$$
- $\forall v \in V, e^-_v \in E^-$ such that:
$$\forall x \in X, \alpha(e^-_v)[x] = \begin{cases} v & \text{if } x = x_v \\ \neg v & \text{if } x = x_{\neg v} \\ \star & \text{otherwise} \end{cases}$$

- $\forall Z \in CL, e^-_Z \in E^-$ with $\alpha(e^-_Z)[x_l] = \begin{cases} l & \text{if } l \in Z \\ \star & \text{otherwise} \end{cases}$

We claim that there exists a constraint network $N = (X, D, C)$ over a language $\{R\}$ of size 1 and arity 1 which is consistent with E if and only if CL is satisfiable.

Let us assume that there exists such a network N consistent with E. We show that this implies that CL is satisfiable. N must accept $\alpha(e^+_\star)$, so $\star \in R$. For each $v \in V$, N must reject $\alpha(e^-_v)$. Hence, we have either $v \notin R$ and $(R, \{x_v\}) \in C$, or $\neg v \notin R$ and $(R, \{x_{\neg v}\}) \in C$. We cannot have both v and $\neg v$ forbidden by R because $\alpha(e^+_v)$ must be accepted by N. We thus have exactly one among v and $\neg v$ forbidden by R, and we can define the assignment $\mu : V \to \{\texttt{False}, \texttt{True}\}$ such that for each $v \in V$ we have $\mu(v) = \texttt{True} \Leftrightarrow v \notin R$ and $\mu(v) = \texttt{False} \Leftrightarrow \neg v \notin R$. We also know that for all $Z \in CL$, N has to reject $\alpha(e^-_Z)$. Now, $\alpha(e^-_Z)[x] = \star$ for all x except those literals composing Z. As a result, there is at least one literal l in Z such that $l \notin R$, and then $\mu(l) = \texttt{True}$. We conclude that μ satisfies CL. Let us now assume that there exists an assignment μ that satisfies CL. We show that this implies that there exists a network $N = (X, D, C)$ over a language $\{R\}$ of size 1 and arity 1 which is consistent with E. We define the relation $R = D \setminus \{l \mid \mu(l) = \texttt{True}\}$ and $C = \{(R, (x_v)) \mid \forall v \in V, \mu(v) = \texttt{True}\} \cup \{(R, (x_{\neg v})) \mid \forall v \in V, \mu(v) = \texttt{False}\}$. N accepts $\alpha(e^+_\star)$ because $\star \in R$. For all $v \in V$, either $(R, \{x_v\}) \in C$ with $\neg v \in R$ and $v \notin R$, or $(R, \{x_{\neg v}\}) \in C$ with $v \in R$ and $\neg v \notin R$, but not both. Hence, N accepts $\alpha(e^+_v)$ and rejects $\alpha(e^-_v)$. As μ satisfies CL, for each $Z \in CL$, there exists $l \in Z$ such that $\mu(l) = \texttt{True}$. Thus, $(R, \{x_l\}) \in C$ with $l \notin R$ and N rejects $\alpha(e^-_Z)$. We conclude that the network N constructed above, which is over a language $\{R\}$ of size 1 and arity 1, is consistent with E.

Both N and E are computable in polynomial time from CL. □

4.3 The Method

In this section, we present our method for constraint acquisition, which is based on repeatedly solving instances of the Language-Free Acq problem.

4.3.1 Overview

Given a training set E, our goal is to compute a constraint network consistent with E with minimum (k, r). As noted in Section 4.2, multiple strategies are possible. The most direct approach would be to output a constraint network with minimum $k + r$. We believe that increasing the arity should incur a greater penalty than increasing the number of relations, so we will minimize $k + r^2$ instead. We break ties by giving preference to lower arity (for example, six relations of arity two are preferred over one relation of arity three).

It may be the case that multiple constraint networks have the same arity and number of distinct relations. In this case, we output a network with the largest number of constraints. Our intuition behind this decision is that for sufficiently large training sets, the fact that few relations can be applied to many scopes without rejecting the assignment of any positive example is unlikely to be observed by chance. For the same reason, if multiple constraint networks have the same (k, r) and number of constraints, we output one whose constraints are the tightest (i.e. whose relations accept the fewest tuples and therefore a constraint with this relation rejects the most assignments). Given (k, r), we compute the desired constraint network (or prove that none exists) using a Weighted Partial Max-Sat model, which we describe in the next sub-section. Since our model is particularly efficient for small values of (k, r), we perform bottom-up minimization, constructing and solving a model for each (k, r) by increasing order of $k + r^2$. We output the first constraint network found.

4.3.2 The Model

Given (k, r) fixed. Our goal is to compute a constraint network $N = (X, D, C)$ over a language of size k and arity r that is consistent with E, whose number of constraints is maximum, and with the tightest constraints possible. We model this optimization problem as an instance of Weighted Partial Max-Sat. In the following, $RT(E)$ will denote the set of all pairs (t, v) such that $t \in D^r$, $v \in X^r$, and there exists an example e in E such that $t = \alpha(e)[v]$.

For each relation R_u of the target language $\{R_1, \ldots, R_k\}$, we have three kinds of Boolean variables in the Weighted Partial Max-Sat model:

- For all t in D^r, r_t^u is `True` iff $t \notin R_u$;
- For all v in X^r, s_v^u is `True` iff $(R_u, v) \in C$;
- For all (t, v) in $D^r \times X^r$, $c_{(t,v)}^u \equiv r_t^u \wedge s_v^u$.

First, for each $(u, t, v) \in \{1, ..., k\} \times D^r \times X^r$, we ensure that $c_{(t,v)}^u \equiv r_t^u \wedge s_v^u$ with the following hard clauses:

$$\begin{cases} r_t^u \vee \neg c_{(t,v)}^u \\ s_v^u \vee \neg c_{(t,v)}^u \\ \neg r_t^u \vee \neg s_v^u \vee c_{(t,v)}^u \end{cases} \tag{4.1}$$

Second, we make sure that all the assignment of positive examples are accepted by the corresponding constraint network with the following set of hard clauses:

$$\forall u \in \{1, ..., k\}, \forall (t, v) \in RT(E^+), \quad \neg c_{(t,v)}^u \tag{4.2}$$

Similarly, we make sure that the assignments of all negative examples are rejected:

$$\forall e \in E^-, \quad \bigvee_{\forall u \in \{1,...,k\}, \forall (t,v) \in RT(\{e\})} c_{(t,v)}^u \tag{4.3}$$

Finally, in order to maximize the number of constraints in the network and, in a second time, minimize the number of tuples in the relations, we add a soft clause (s_v^u) with weight 1 for each $u \in \{1, \ldots, k\}$ and $v \in X^r$, and a soft clause (r_t^u) with weight $\epsilon < 1/(k \cdot |D^r|)$ for each $u \in \{1, \ldots, k\}$ and $t \in D^r$. This completes the description of the model.

The size of this Weighted Partial Max-Sat model is defined by its number of clauses and their sizes. The number of –constant-size– clauses of type (1) and (2) is $O(|D|^r \cdot |X|^r \cdot k)$. There are $|E^-|$ clauses of type (3), each of size bounded above by $|D|^r \cdot |X|^r \cdot k$. Finally, the number of –constant-size– soft clauses (s_v^u) and (r_t^u) is bounded above by $(|X|^r + |D|^r) \cdot k$. This gives a total size of our model in $O(|E^-| \cdot |D|^r \cdot |X|^r \cdot k)$.

Observe that the exponential dependency on the maximum arity r is not a necessary feature of all models for this problem because Language-Free Acq is

in NP (even when r is part of the input). However, this model has the advantage of being flexible (it is easy, for example, to maximize the number of constraints in the learned constraint network), and very efficient for small values of r.

In addition, this upper bound is quite loose, as not all variables/clauses need to be generated. For any u, t, v such that $(t, v) \notin RT(E^-)$, we do not need to generate the variable $c^u_{(t,v)}$ and the corresponding clauses described in Equation (4.1) because it will never appear in the clauses of type Equation (4.2) or Equation (4.3). The same is true for all u, t, v such that $(t, v) \in RT(E^+)$. These refinements are particularly effective when the number of examples is small.

Finally, we note that the model contains some symmetries. For example, its solution set is invariant under permutation of the relational indices $u \in \{1, \ldots, k\}$ and permutation of the entries of (t, v) for a fixed u. These symmetries can easily be broken using standard techniques. It was unnecessary for our experiments as k, r were always fairly small.

Example 4.3.1 (Illustrative Example)**.** *Consider a toy problem with two variables $X = \{x_1, x_2\}$ and domains $D = \{0, 1\}$. We aim to learn a constraint network with parameters $k = 1$ (one relation) and $r = 2$ (binary).*

Suppose the training set E consists of one positive example $e^+ = \{x_1 = 0, x_2 = 1\}$ and one negative example $e^- = \{x_1 = 0, x_2 = 0\}$.

The model generates the following Boolean variables for the single relation R_1:

- ***Relation variables:*** $r^1_{00}, r^1_{01}, r^1_{10}, r^1_{11}$. *(e.g., $r^1_{00} =$* `True` *means tuple $(0, 0)$ is forbidden by R_1.)*
- ***Scope variables:*** $s^1_{(x_1,x_2)}$ *and* $s^1_{(x_2,x_1)}$.
- ***Connection variables:*** $c^1_{(00,(x_1,x_2))}, c^1_{(01,(x_1,x_2))}, \ldots$ *and similarly for scope (x_2, x_1).*

The clauses are generated as follows:

- ***Consistency (Eq. 4.1).*** *For every tuple t and scope (x_1, x_2), we enforce $c \equiv r \wedge s$. For instance, for tuple $(0, 0)$:*

$$r^1_{00} \vee \neg c^1_{(00,(x_1,x_2))}, \quad s^1_{(x_1,x_2)} \vee \neg c^1_{(00,(x_1,x_2))}, \quad \neg r^1_{00} \vee \neg s^1_{(x_1,x_2)} \vee c^1_{(00,(x_1,x_2))}$$

- ***Positive examples (Eq. 4.2).*** *For e^+, the network must not reject the*

assignment on any scope:

$$\neg c^1_{(01,(x_1,x_2))} \quad \textit{and} \quad \neg c^1_{(10,(x_2,x_1))}$$

- ***Negative examples (Eq. 4.3).*** *For e^-, the network must reject the assignment on at least one scope:*

$$c^1_{(00,(x_1,x_2))} \vee c^1_{(00,(x_2,x_1))}$$

4.4 Experimental Evaluation

In this section, we experimentally evaluate our method (that we will refer to as LFA) on several benchmark problems. For each benchmark, we investigate how the number of examples affects the accuracy of the learned constraint network, the similarity of the learned constraint network with the target (i.e. whether they are over the same constraint language, logically equivalent, or exactly identical), and the observed runtime. We then provide a detailed analysis for a representative benchmark of constraint acquisition, the Sudoku, examining in particular how the fraction $|E^+|/|E|$ of positive examples in the training set affects the learning process.

4.4.1 Implementation

We have implemented the strategy described in Section 4.3.1 in the Python programming language. The initial implementation used for our paper [6] was based on the UWrMaxSat solver [33], which is a state-of-the-art Max-Sat solver. However, we have since switched to the OR-Tools solver [32] for the current implementation. The switch to OR-Tools was motivated by its performance, and ease of integration with our Python code without the need to install an external solver. We have also simplified the use of our method by providing it as a Python package. The package includes a simple interface for easy access. A small example is provided in Figure 4.1, which illustrates how to use the package to learn a constraint network from a training set stored in a CSV file. The complete source code and datasets for the experiments are hosted on GitHub: `github.com/hareski/language-free-acq` under the Academic Free License (AFL-3.0), and are available on the Python Package Index (PyPI)

under the name `languageFreeAcq` (pypi.org/project/languageFreeAcq). The package can be easily installed by using the command `pip install languageFreeAcq`. All experiments presented in this section were conducted on 8 cores of an AMD Epyc 9554 processor with 16GB of RAM.

```
from languageFreeAcq import Acquisition

csp = Acquisition().learn("path/to/training_set.csv")

# Get the learned constraint network:
constraints = csp.get_constraints()

# Get the learned constraint language:
relations = csp.get_language()
```

Figure 4.1: Illustration of how to use our method in an external Python script.

4.4.2 Benchmark Problems

We assess our method on a diverse set of benchmark problems. For each benchmark, we describe a constraint network that models the problem, including its variables, domains, and constraint language. We also explain how the examples in the training set are generated for each benchmark. The test set is generated in the same way as the training set.

Sudoku

Sudoku is a logic puzzle with a 9×9 grid that must be filled with the digits 1 to 9 in such a way that all the rows, all the columns, and 9 non-overlapping 3×3 squares contain all the digits from 1 to 9. For this problem, the target constraint network has 81 variables $x_1, \ldots, x_{81}$, domains of size 9, and a binary constraint $x_i \neq x_j$ for all i, j in the same row, column or square. Positive examples are generated by computing solutions of the target constraint network with a constraint solver using a randomized domain value strategy. Non-solutions are generated by randomly altering one value in a solution.

Jigsaw Sudoku

Jigsaw Sudoku is a variant of Sudoku in which the partition in 3×3 squares is replaced by a partition into non-overlapping, irregular shapes of size 9 called jigsaw shapes. The irregularity of the jigsaw shapes makes Jigsaw Sudoku particularly difficult (or even impossible) to learn for methods that rely heavily on predefined constraint topologies, such as MODEL SEEKER. For this problem, the target constraint network has 81 variables $x_1, \ldots, x_{81}$, domains of size 9, and a binary constraint $x_i \neq x_j$ for all i, j in the same row, column or jigsaw shape. Examples are generated in the same way as for Sudoku.

We have observed significant variance in experimental results for different types of jigsaw shapes. To reflect this, we have divided this benchmark into three sub-families (#1, #2 and #3) corresponding to three different layouts.

1	1	1	1	1	2	2	2	2	1	1	2	2	2	2	2	3	3
1	3	1	1	1	2	4	2	2	1	1	1	2	2	2	2	3	3
5	3	3	3	3	4	4	2	4	1	1	4	4	5	5	5	3	3
5	5	5	3	3	3	4	2	4	1	4	4	4	5	6	3	3	3
5	6	6	6	6	3	4	4	4	1	4	4	4	5	6	6	6	7
5	6	6	6	7	7	7	7	7	8	8	8	4	5	6	6	6	7
5	5	6	8	7	9	7	7	9	8	8	5	5	5	6	6	7	7
5	8	6	8	7	9	9	9	9	8	8	9	9	9	9	7	7	7
8	8	8	8	8	8	9	9	9	8	8	9	9	9	9	9	7	7

(a) Jigsaws [#1]

1	1	2	2	2	2	3	3	3
1	1	1	1	2	2	3	3	3
1	4	1	4	2	2	2	3	3
1	4	4	4	5	5	5	5	3
4	4	6	4	7	5	7	5	5
8	8	6	4	7	7	7	5	9
8	8	6	7	7	7	7	5	9
8	8	6	6	6	6	9	9	9
8	8	8	6	6	9	9	9	9

(b) Jigsaws [#2]

(c) Jigsaws [#3]

Figure 4.2: Visualization of the three Jigsaw Sudoku layouts. Each cell represents the Jigsaw number (e.g., all cells with value 1 are in the same jigsaw region) of the variable $x_{i,j}$ where (i, j) are the coordinates of the variable on the 9×9 grid. The regions are labeled with numbers from 1 to 9, and each number corresponds to a different color. The regions are irregularly shaped, which is characteristic of Jigsaw Sudoku. The figure shows three different layouts ([#1], [#2], and [#3]).

Schur's Lemma

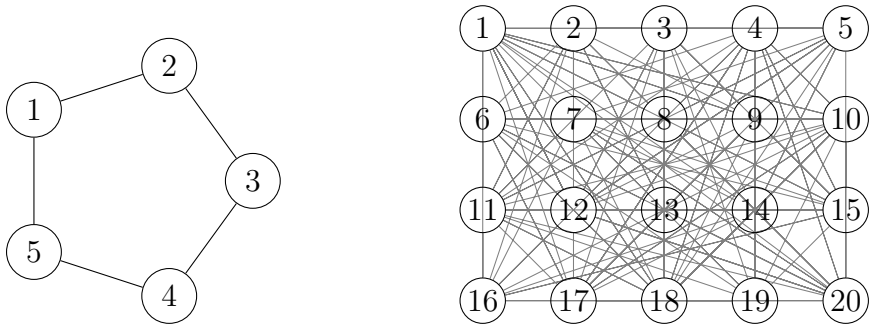
The problem is to put n balls labelled $1, \ldots, n$ into 3 boxes so that for any triple of balls (x, y, z) with $x + y = z$, not all are in the same box. For this problem,

the target constraint network has n variables $x_1, \ldots, x_n$, domains of size 3, and a ternary constraint $\texttt{NotAllEqual}(x_i, x_j, x_k)$ for all i, j, k such that $i + j = k$. We ran the experiment with $n = 9$ which is the parameter with the highest number of solutions (546). Positive examples are generated by computing solutions of the target constraint network with a constraint solver using a randomized domain value strategy. Non-solutions are generated by randomly altering one value in a solution.

Subgraph Isomorphism

Given two graphs G and H, subgraph isomorphism is the problem of determining whether G contains a subgraph that is isomorphic to H. For this problem, the target constraint network has $|H|$ variables $x_1, \ldots, x_n$ and domains of size $|G|$. A binary constraint $x_i \neq x_j$ for all i, j ensures that the mapping between the vertices of H and G is a one-to-one function and another binary constraint ensures that for any edge (a, b) in H, (x_a, x_b) is an edge of G. We ran the experiment with H a cycle of size 5 (5-cycle) and a new random graph G for each run having 20 vertices and 100 edges.

Positive examples are generated by computing solutions of the target constraint network with a constraint solver using a randomized domain value strategy. Non-solutions are paths and closed walks (not isomorphic to H) of G computed using a randomized domain value strategy.



(a) Graph H (pattern to find) (b) Example of graph G

Figure 4.3: Visualization of graphs used in the subgraph isomorphism problem. The task is to find whether graph H is isomorphic to a subgraph of G.

8-Queens

The 8-Queens problem is the problem of placing 8 queens on an 8×8 chessboard such that no two queens can attack each other. For this problem, we use the standard representation in which there is a variable per column. The target constraint network has N variables $x_1, \ldots, x_N$, where x_i represents the row in which the queen on the ith column is positioned. All domains are $\{1, \ldots, N\}$. There are binary constraints $x_i \neq x_j$ and $|x_i - x_j| \neq |i - j|$ for all i, j. This gives us a language of size N corresponding to all possible values of $|i - j|$. The constraint language is binary and has size N. We ran the experiment with $N = 8$, for which the problem has 92 solutions. With this parameterization, the target constraint language contains 8 binary relations. We generate positive examples by computing a random solution, and negative examples by randomly permuting two values or altering one value in a solution.

Golomb Ruler

The Golomb Ruler problem is the problem of finding a set of marks on a ruler such that the difference between any two marks is unique. The target constraint network has n variables, each representing the position of a mark on the ruler, domains of fixed size $\{0...m\}$, and a quaternary constraint $|x_i - x_j| \neq |x_k - x_l|$ for all i, j, k, l (we do not ensure that i, j, k, l are distinct, so the constraint is quaternary, but it can be seen as a ternary constraint in some cases where $i = j$ or $k = l$, or even binary). For the experiment, we choose to 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 $i < j$, and then randomly permuting the values of this solution. Non-solutions are generated by randomly altering one value in a solution.

Exam Timetabling This problem (used as a benchmark in [41, 43–45]) 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. We ran this benchmark problem with three instances with different parameter configurations:

[#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.

Positive examples are generated by computing solutions of the target constraint network with a constraint solver using a randomized domain value strategy. Non-solutions are generated by randomly altering one value or permuting the values of two variables in a solution.

Nurse Rostering This problem is used in [41, 43] and also present in [21, 22] 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 slots per shift with 12 nurses;
[#2] 5 days, 3 shifts and 5 slots per shift with 18 nurses;
[#3] 7 days, 3 shifts and 3 slots per shift with 15 nurses.

Positive examples are generated by computing solutions of the target constraint network with a constraint solver using a randomized domain value strategy. Non-solutions are generated by altering one value or randomly permuting the values of two variables in a solution.

4.4.3 Network and Language Acquisition

In this first experiment, we evaluate the overall performance of our method on the acquisition of our benchmark problems. We first present the experimental protocol

and then discuss important points in the results.

Protocol

We conduct a series of experiments with different numbers of examples in the training sets that depend on the instance of the problem. For each benchmark problem and number $|E|$ of examples, we run our acquisition method LFA 5 times with a new randomly sampled training set for each run. Training sets contain positive and negative examples in equal proportion. The timeout is set to 12 hours. The performance of the model is measured in terms of the average accuracy over the five runs, which is computed on a new set of 2000 examples generated independently. We also report the optimal (k, r) values found, the number of times the learned language is the target language (out of the 5 runs), the number of times the learned network is equivalent to the target network (i.e. they have exactly the same solutions) and the number of times the learned network is precisely the target network (i.e. with exactly the same constraints). We finally record the average runtime of the acquisition process, including the time required to prove that there does not exist any network consistent with E for values of (k, r) smaller than the (optimal) one returned.

Results

We provide a summary of the results in Table 4.1. The target network for the Sudoku problem is consistently learned (that is, learned for all 5 runs) with 200 examples in the training set, and even as few as 100 examples in 2 runs out of 5. Jigsaw Sudoku required significantly more examples before reaching 100% accuracy, from 500 to 1300 depending on the jigsaw shapes. For this problem, the target constraint network is never learned, regardless of the size of the training set. Equivalent networks are learned instead, which we observed to correspond to the target network with additional redundant constraints. (For the classical Sudoku, all possible redundant inequality constraints are already included in the target network. This is not true for all jigsaw shapes.) For Schur's Lemma, with only 50 examples, the target constraint language is consistently learned, and the accuracy is above 85%. This is particularly interesting because this language has arity 3, and with our search strategy (minimizing $k + r^2$ where k is the number of relations and r is the arity), the target language

Problem	$\|E\|$	Accuracy	(k, r)	Language	Equivalent	Target	Runtime(s)
Sudoku	100	83.7%	$(1, 2)$	5/5	2/5	2/5	29.5
	150	98.9%	$(1, 2)$	5/5	4/5	4/5	21.7
	200	100%	$(1, 2)$	5/5	5/5	5/5	28.6
	400	100%	$(1, 2)$	5/5	5/5	5/5	36.3
Jigsaw #1	400	98.9%	$(1, 2)$	5/5	0/5	0/5	29.1
	600	99.5%	$(1, 2)$	5/5	0/5	0/5	36.8
	800	99.8%	$(1, 2)$	5/5	3/5	0/5	34.9
	1200	99.9%	$(1, 2)$	5/5	4/5	0/5	38.3
	1300	100%	$(1, 2)$	5/5	5/5	0/5	37.0
	1400	100%	$(1, 2)$	5/5	5/5	0/5	39.1
Jigsaw #2	400	99.3%	$(1, 2)$	5/5	2/5	0/5	30.1
	600	99.9%	$(1, 2)$	5/5	4/5	0/5	39.0
	700	99.9%	$(1, 2)$	5/5	4/5	0/5	35.5
	800	100%	$(1, 2)$	5/5	5/5	0/5	35.8
Jigsaw #3	400	99.7%	$(1, 2)$	5/5	3/5	0/5	35.4
	500	100%	$(1, 2)$	5/5	5/5	0/5	32.3
	600	100%	$(1, 2)$	5/5	5/5	0/5	41.7
Schur's	10	51.9%	$(1, 2)$	0/5	0/5	0/5	0.1
Lemma	50	86.9%	$(1, 3)$	5/5	0/5	0/5	6.7
	100	96.3%	$(1, 3)$	5/5	0/5	0/5	1.4
	200	98.8%	$(1, 3)$	5/5	2/5	2/5	1.7
	400	99.7%	$(1, 3)$	5/5	3/5	3/5	2
	500	99.8%	$(1, 3)$	5/5	4/5	4/5	1.8
	600	100%	$(1, 3)$	5/5	5/5	5/5	1.9
Subgraph	100	58.7%	$(1, 2)$	0/5	0/5	0/5	0.5
Isomorphism	400	99.7%	$(2, 2)$	0/5	0/5	0/5	3.9
	600	99.9%	$(2, 2)$	0/5	4/5	0/5	7.4
	700	100%	$(2, 2)$	0/5	5/5	0/5	7.4
	800	100%	$(2, 2)$	0/5	5/5	0/5	10.1
8-Queens	100	87.2%	$(2, 2)$	0/5	0/5	0/5	2.4
	184	99%	$(3, 2)$	0/5	0/5	0/5	8.9

Problem	$\|E\|$	Accuracy	(k, r)	Language	Equivalent	Target	Runtime(s)
Golomb	400	76.6%	$(1, 2)$	0/5	0/5	0/5	167.8
ruler	800	71.5%	$(2, 2)$	0/5	0/5	0/5	21 616
	1600	-	-	-	-	-	$>$ 43 200
	3200	100%	$(1, 3)$	0/5	5/5	0/5	10 399
Nurse	100	77.8%	$(1, 2)$	2/5	0/5	0/5	29.94
Rostering #1	150	96%	$(1, 2)$	4/5	4/5	3/5	13
	200	100%	$(1, 2)$	5/5	5/5	5/5	4.55
Nurse	200	75.3%	$(1, 2)$	1/5	0/5	0/5	43 200
Rostering #2	300	99%	$(1, 2)$	5/5	3/5	3/5	157.9
	400	100%	$(1, 2)$	5/5	5/5	5/5	77.9
Nurse	400	98%	$(1, 2)$	5/5	3/5	3/5	77.9
Rostering #3	600	99.5%	$(1, 2)$	5/5	4/5	4/5	115.7
	700	99.7%	$(1, 2)$	5/5	4/5	4/5	115.7
	800	100%	$(1, 2)$	5/5	5/5	5/5	106.5
Exam	100	84%	$(2, 2)$	0/5	0/5	0/5	2
Timetabling #1	150	98%	$(2, 2)$	4/5	4/5	4/5	2
	200	100%	$(2, 2)$	5/5	5/5	5/5	4.5
Exam	400	97.2%	$(2, 2)$	0/5	0/5	0/5	9.2
Timetabling #2	800	99.6%	$(2, 2)$	3/5	3/5	3/5	9.9
	1200	99.6%	$(2, 2)$	3/5	3/5	3/5	9.6
	1400	99.9%	$(2, 2)$	4/5	4/5	4/5	9.7
	1500	100%	$(2, 2)$	5/5	5/5	5/5	10.1

Problem	$\|E\|$	Accuracy	(k, r)	Language	Equivalent	Target	Runtime(s)
Exam	600	98.3%	$(2, 2)$	2/5	2/5	2/5	55.3
Timetabling #3	700	99%	$(2, 2)$	2/5	2/5	2/5	33.7
	800	99.5%	$(2, 2)$	2/5	2/5	2/5	28.5
	900	100%	$(2, 2)$	5/5	5/5	5/5	31.8

Table 4.1: Summary of the experiment with the LFA method described in Section 4.4.3. $|E|$ is the number of examples in the training set; Accuracy is the accuracy measured on a new set of 2000 examples generated independently; (k, r) gives the optimal values computed for the size and arity of the learned constraint language; Language is the number of times the target language is learned out of 5 runs; Equivalent is the number of times the learned and target network are equivalent out of 5 runs; Target is the number of times the target network is learned out of 5 runs.

has a score of 10. Before it even attempts to find a solution for a language with arity 3 (like the target for Schur's Lemma), it first exhaustively attempts to find a network consistent with the training set over all constraint languages with at most 6 binary relations ($k + r^2 \leq 10$). Thus, this means that all constraint languages with at most 6 binary relations can be ruled out with very few examples. Learning the target network consistently requires up to 600 examples, although 2 runs out of 5 succeeded with only 200. Subgraph isomorphism is the first problem for which the target language contains two relations. With 100 examples, the learned networks are over a language with a single relation, and the accuracy is below 60%. 100% accuracy and equivalent networks are reached with 700 examples, although the target network and language can never be learned. This is because it is theoretically not possible to distinguish the graph G (whose edges correspond to the tuples of one relation in the target language) from another graph G' with identical 5-cycles using only examples. Since the examples only capture the presence or absence of a 5-cycle, any graph with identical 5-cycles may generate the same training examples. For the 8-Queens problem, the experiment is limited to at most 184 examples because the problem has only 92 solutions and we need 50% of positive examples in the training set. We observe that even training sets with 184 examples are not sufficient to reach 100% accuracy or learn the target constraint language (which is of size 8). Instead, our method outputs constraint networks over languages of size only 3 that achieve 99% accuracy.

The Golomb Ruler is particularly challenging because the target language has arity 4. Runtimes are extremely high, with the 12 hours timeout being reached for 1600 examples. In three of the five runs, no solution was found within this timeframe; thus we exclude results for 1600 examples. With 400 examples, an accuracy of 76.6% is reached with a constraint language containing a single binary relation. With 800 examples, a lower accuracy of 71.5% is reached with a constraint language containing two binary relations. Perhaps surprisingly, with 3200 examples, an equivalent constraint network over a language with a single ternary relation is obtained in all five runs. This relation is symmetric and is applied to all possible triples of distinct variables, revealing some hidden structure in the solutions of this problem. For the Nurse Rostering, the target constraint language and the target constraint network are consistently learned with a number of examples ranging from 200 to 800, depending on the instance. Running times are generally small when the number of examples is large enough to reach a good accuracy, but can be very long when the number of examples is relatively small. This is the case, for instance, with the #2 of the Nurse Rostering problem, where we reach the 12 hours timeout. However, the solver returns the best network found during the search, which cannot be verified as optimal. These (possibly suboptimal) networks achieve only 75.3% accuracy on average. For the Exam Timetabling problem, the target constraint network is consistently learned with a number of examples ranging from 200 to 1500, depending on the instance. The second instance seems to be the most difficult one in terms of the number of examples needed to reach 100% accuracy, requiring 1500 examples. However, in this specific instance, the accuracy is above 99.6% from 800 examples. Increasing the training set size from 800 to 1500 examples only serves to gain the final 0.4% of accuracy.

4.4.4 Detailed Analysis on the Sudoku Problem

In this section, we investigate how the number of examples and the positive-to-negative ratio in the training set affect the accuracy and runtime of the learned constraint network with an archetypal example of constraint acquisition: the Sudoku.

Runtime

In this experiment, we take a closer look at the runtime required by our method LFA, as a function of the number of examples. We focus on the Sudoku problem and run the experiment with the number $|E|$ of examples going from 0 to 300 by steps of 5. For each value, we run our method five times (with $|E^+|/|E| = 0.5$) and report the average runtime. For this experiment, we set the timeout to 3 hours (10 800 seconds). If any of the five runs reaches the 3 hours timeout without finding and proving the optimal network, we ignore the others and simply report a timeout for the corresponding $|E|$.

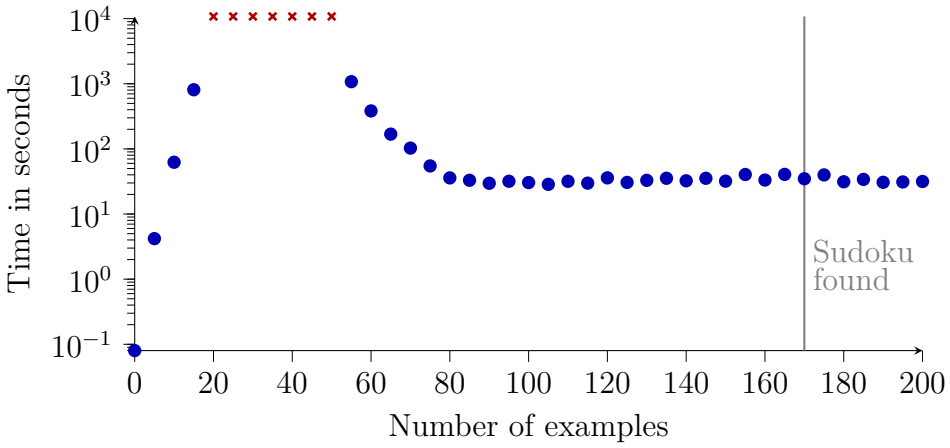


Figure 4.4: Runtime as a function of the number of examples for the Sudoku problem. The vertical line indicates the number of examples from which the output is the target Sudoku network. Notice that the y-axis is on a logarithmic scale.

Figure 4.4 shows our results. An optimal constraint network (in the sense of our objective function) is found rather quickly when the number of examples is either extremely small (5 to 15) or sufficiently large (55 or more). This is unsurprising because examples translate into hard clauses in our Weighted Partial Max-Sat model: with very few examples the model is underconstrained, and with sufficiently many examples the search space becomes small. The transition appears to occur between 15 and 55 examples, where the solver systematically reaches the timeout. As we will see in the last experiment, for such values of $|E|$ the accuracy of an optimal constraint network would be close to 0.5. This means that, at least in the case of the Sudoku problem, very long runtimes are only observed for training sets that are too

small for our method to learn the target constraint network (independently of the computational resources available). For 170 examples or more, the optimal solution is the target Sudoku network and is found in less than 30 seconds.

Ratio of Positive Examples

In this experiment, we vary the fraction of positive examples $p = |E^+|/|E|$ in the training set. For each $p \in \{0, 0.2, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9, 1\}$, we generate training sets with a fraction p of positive examples. In Table 4.2 we report the minimum number of examples needed by our method to return the target Sudoku network. Results are averaged over five runs.

p	0	0.2	0.4	0.5	0.6	0.7	0.8	0.9	1
$\|E\|$	×	343	172	137	115	106	97	105	×

Table 4.2: Number $|E|$ of examples needed to learn the target Sudoku network for a given fraction p of positive examples. A cross indicates that the target network is never returned.

We see that our method does not learn the target constraint network with only positive examples ($p = 1$). Indeed, in this case a degenerate constraint network with $(k, r) = (1, 1)$ can be learned. It applies a very loose relation to all variables, and does not enforce real constraints on the values of the variables. This network is the simplest network that is consistent with all the examples and maximizes the number of constraints in the network. Likewise, our method does not learn the target network with $p = 0$. In this case, the degenerate constraint network that can be learned applies the empty relation to all variables, meaning that it rejects all assignments of values to the variables.

Except for these two extreme cases, we observe that as the ratio of positive increases, fewer examples are needed to acquire the target network. This is not valid past a certain point; we observe an increase in the number of required examples at $p = 0.9$ compared to $p = 0.8$ because our method needs some negative examples to learn the target relation. Overall, it seems that our method performs best on the Sudoku problem when the ratio of positive examples is 0.8.

Accuracy and Number of Constraints Learned

In this experiment, we analyze the accuracy of the learned constraint networks as a function of the number of examples. We also measure the average number of constraints learned that are not in the target network. We vary the number $|E|$ of examples from 50 to 200 by steps of 5, with $|E^+|/|E| = 0.5$ for all training sets. For each value $|E|$, we run our method 5 times on samples of $|E|$ randomly generated examples. We then measure the average accuracy of the 5 learned constraint networks using 2000 additional examples and the average number of constraints learned that are not in the target network. Figure 4.5 shows the average accuracy as a function of $|E|$ and Figure 4.6 shows the average number of constraints learned that are not in the target network but are learned by our method with this specific number of examples as a function of $|E|$.

The language learned with more than 75 examples is always the target language, which is the one with only the binary disequality relation. With more than 75 examples, the number of constraints not learned that are in the target network is always 0. Indeed, our method primarily maximizes the number of constraints learned that satisfy all examples. Thus, if the language learned is the target language, our method cannot miss any constraints, it can only learn additional constraints that are not in the target network.

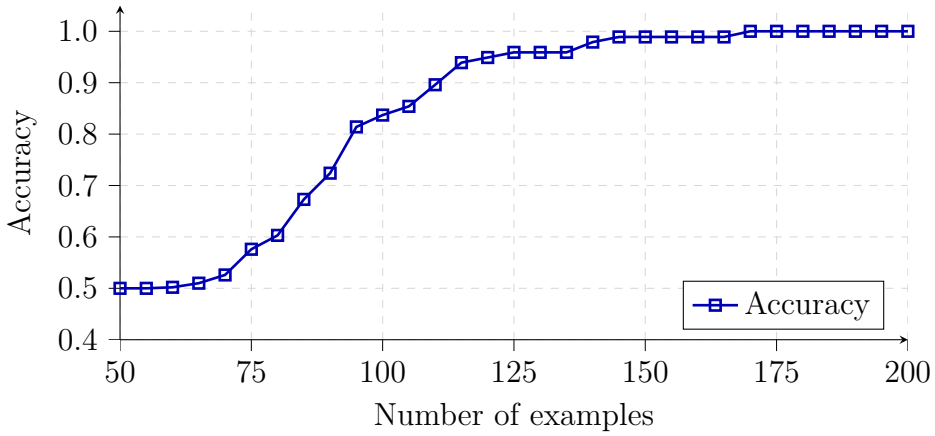


Figure 4.5: Accuracy as a function of the number of examples in the training set for the Sudoku problem.

In Figure 4.5, we observe that the accuracy increases with the number of examples. At 75 examples, the measured accuracy is close to 0.5, which means that the number

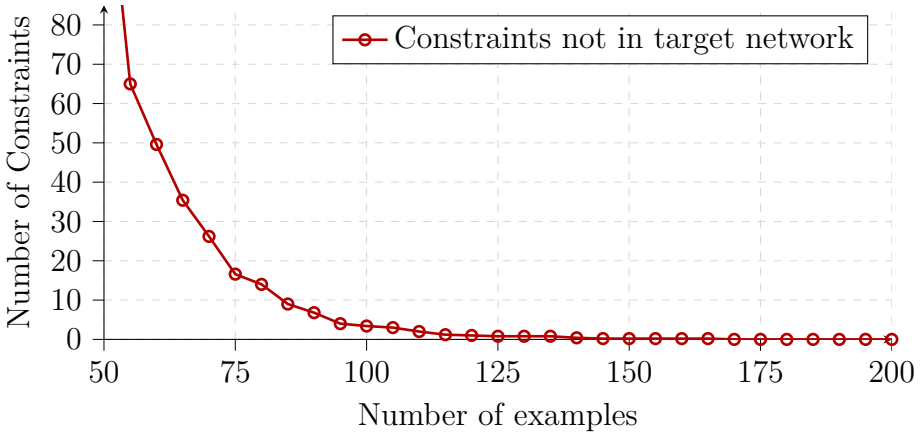


Figure 4.6: Number of constraints learned that are not in the target network as a function of the number of examples in the training set for the Sudoku problem.

of examples is too small to allow our method to learn a network capturing the problem. From 75 to 120 examples, the accuracy increases up to 0.95. From 120 to 170 examples, the accuracy slowly increases up to 1. This evolution is explained by analyzing the number of constraints learned that are not in the target network, as shown in Figure 4.6. A 9×9 grid contains 3,240 distinct binary scopes without permutation (sets of two variables), each representing a potential constraint scope. The target Sudoku constraint network consists of 810 constraints in total out of these 3,240 possibilities. There does not exist any redundant binary disequality constraint that can be added to the target network without removing a solution (in contrast to the Jigsaw Sudoku problem, where redundant constraints can be added to the target network). Thus, the target constraint network is the largest (in terms of the number of constraints) possible constraint network that can be learned for the Sudoku problem using only the binary disequality relation. Figure 4.6 shows that the number of constraints learned that are not in the target network decreases rapidly from 81 at 50 examples to only 1 at 120 examples, reaching zero at 170 examples. This indicates that the learned constraint network becomes equivalent to the target network when $|E| \geq 170$.

Notably, over 99.97% of the scopes not in the target constraint network are eliminated using only 120 examples. Then, 50 additional training examples are then needed to reach 100% accuracy, and eliminate the last remaining scopes not in the target. The slowness in gaining the last percentage points of accuracy is explained

by the fact that our method maximizes the number of constraints in the network, which leads to the inclusion of some erroneous constraints. These constraints can only be ruled out by a positive example that contradicts them. This is why the accuracy does not reach 100% until 170 examples, even though a constraint network very close to the target network in terms of accuracy and of constraints learned is learned with 120 examples.

A way to address this issue is to change the maximization criterion to a minimization one, which would lead to a network with the minimum number of constraints that is consistent with all examples. However, this approach has its own limitations. To demonstrate the shortcomings of simply changing the maximization criterion to a minimization one, we conducted an additional experiment on the Sudoku problem. The issue with the minimization approach is that it results in a network with an insufficient number of constraints compared to the target network. In our experiment, we analyzed the learned networks for various training set sizes from 10 to 200 examples, with $|E^+|/|E| = 0.5$ for all training sets. For each value $|E|$, we run our method 5 times on samples of $|E|$ randomly generated examples. For the range before 130 examples (10-130), by steps of 10, we compared how often the target language was learned using either the maximization or the minimization criterion across 5 runs for each training set size. The results are shown in Table 4.3. For values of $|E|$ greater than or equal to 130 examples, we vary the number of examples by steps of 5, and we report the average accuracy of the 5 learned constraint networks in Figure 4.7 and show the average total number of constraints learned in Figure 4.8 using the minimization criterion. The results are compared to the maximization criterion, which is shown in Figure 4.5 and Figure 4.6.

$\|E\|$	10	20	30	40	50	60	70	80	90	100	110	120	130
Maximization	0/5	0/5	0/5	0/5	3/5	4/5	4/5	5/5	5/5	5/5	5/5	5/5	5/5
Minimization	0/5	0/5	0/5	0/5	0/5	0/5	0/5	0/5	0/5	2/5	3/5	4/5	5/5

Table 4.3: Number of times out of 5 runs the target language ($\Gamma = \{\neq\}$) is learned with maximization vs. minimization for different training set sizes $|E|$.

Table 4.3 shows that the target constraint language is never learned with the minimization criterion for training sets smaller than 100 examples. The target language is learned in only 2 out of 5 runs with 100 examples and in 3 out of 5 runs

with 110 examples. It is only with 130 examples that the target language is learned in all runs. On the other hand, with the maximization criterion, the target language is learned in all runs for all training set sizes with only 80 examples and 3 out of 5 runs with 50 examples. This shows that the maximization criterion is much more effective than the minimization criterion for learning the target constraint language in the Sudoku problem.

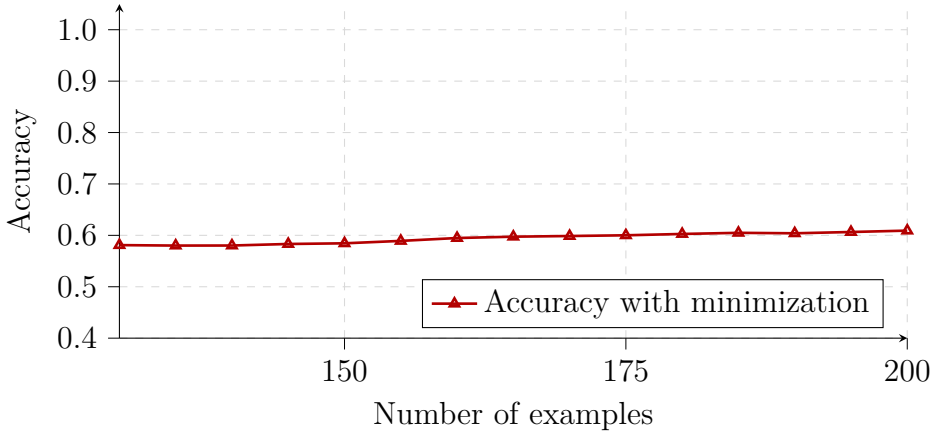


Figure 4.7: Accuracy as a function of the number of examples in the training set for the Sudoku problem when the maximization criterion is changed to a minimization criterion.

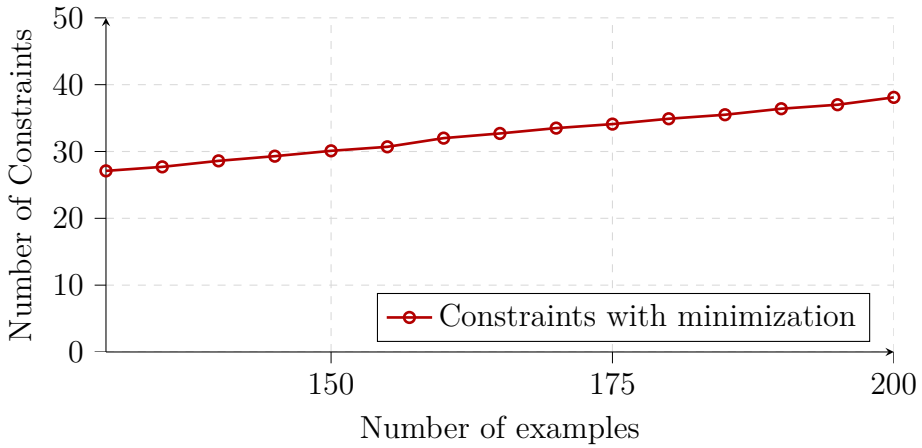


Figure 4.8: Number of constraints learned in total as a function of the number of examples in the training set for the Sudoku problem when the maximization criterion is changed to a minimization criterion. The target network has 810 constraints.

We observe in Figure 4.7 that the accuracy is very low compared to the accuracy

obtained with the maximization criterion. The accuracy is around 0.6 for all values of $|E|$ and does not reach 1 even with 200 examples. This is because the learned constraint network does not capture the problem well, as shown in Figure 4.8. The number of constraints learned is very low, and even with 200 examples, it is only 38 on average. The network learned is very loose and does not capture the problem well. This illustrates how unsatisfactory it is to use a minimization criterion. Indeed, the target constraint network has 810 constraints, and the learned network with the minimization criterion has only 38 constraints on average.

4.5 Limitations and Perspectives

The experimental results are promising, but they also reveal several limitations of the current approach. In this section, we analyze these limitations and outline potential directions for future research that aim to overcome them.

4.5.1 Lack of Structure

A fundamental limitation of our current approach lies in how it handles constraint scopes. The model is agnostic to any underlying structure in the set of scopes where constraints apply. It treats every possible tuple of variables of arity r as an independent candidate scope, simply learning a list of scopes to which the relations in the learned language should be applied. This leads to a failure to capture the high-level structure inherent in problems. We believe that the fairly large number of examples sometimes required to gain the last percentage points of accuracy is largely due to this limitation.

The model does not recognize that certain scopes of variables are more relevant than others based on their positions or relationships within the problem domain. For instance, in the Sudoku problem, the disequality constraints are not applied to random pairs of variables. They are applied systematically to all pairs of variables within the same row, column, or 3×3 block. Our model does not recognize the concept of a row; instead, it must learn each individual scope (e.g. $(x_{1,1}, x_{1,2})$ then $(x_{1,1}, x_{1,3})$, and so on) independently. This lack of a higher-level understanding of scope patterns is what may make it difficult to gain the last percentage points of accuracy.

The key insight is that we need to move beyond a simple list of scopes and instead learn a more structured representation of the arrangement of the scopes. The goal is to capture the inherent structure of the problem in a way that allows the model to generalize better and require fewer examples to achieve high accuracy. To address this, the next chapter introduces a more advanced, structured representation called a template. A template is composed of two core components. First, it includes a set of attributes that describe the variables in the problem, such as their row index or block index. Second, it contains a set of rules that define how constraints should be applied based on these attributes. For example, a rule might state that a disequality constraint should be applied to any pair of variables that share the same row index. By learning compact templates, the model can effectively capture the structure of the problem. This structured approach is expected to significantly reduce the number of examples needed to achieve high accuracy, as it allows the model to generalize from fewer instances by leveraging the inherent structure of the problem.

4.5.2 Interpretability of the Language

A core principle of our method is to search for the simplest language, which we currently define by minimizing the maximum arity and the number of relations. While effective, this definition is rudimentary as it does not account for the intrinsic complexity or interpretability of the relations themselves. For instance, a language with a single, highly irregular and randomly structured relation might be preferred over a language with two simple, highly structured relations, even if the latter is more intuitive and generalizable.

A promising direction for future research is to refine this notion of simplicity by considering the descriptive complexity of the learned language relative to a set of known, common relations. The idea is to favor new languages that can be constructed from a basis of well-understood relations using a few elementary operations. Let us consider a reference language Γ containing common and well-understood relations (e.g., equality, disequality, less than). Let Ω be a set of elementary set operations, such as union ($\cup$), intersection ($\cap$), and set difference ($\setminus$).

We define a recursive function $\#op(R, \Gamma, \Omega)$ that counts the minimum number of operations from Ω needed to express the relation in R using the relations in Γ:

$$\#op(R, \Gamma, \Omega) = \begin{cases} 0 \text{ if } R \in \Gamma \\ \min_{\omega \in \Omega, R=\omega(R'_1,R'_2)} 1 + \#op(R'_1, \Gamma, \Omega) + \#op(R'_2, \Gamma, \Omega) \end{cases}$$

To compute the "cost" of a constraint language L, we should not simply sum the number of operations needed to compute the cost of each relation in L. Instead, we compute the cost of a set of relations $L = \{R_1, \ldots, R_m\}$ by considering the minimum number of operations needed to express all relations together in L. This is because the cost of a single relation does not capture the complexity of a language as a whole, especially when relations can be interdependent or share common structures. We prefer to have a bias for languages where relations build upon each other naturally.

Therefore, we define a *cost* of expressing a set of target relations $L = \{R_1, \ldots, R_m\}$ from the reference language Γ as the minimum number of operations needed to construct the relations in L from the relations in Γ and the operations in Ω:

$$cost(L, \Gamma, \Omega) = \min_{\pi \in Sym(n), \omega \in \Omega} \sum_{i=1}^{m} \#op(R_{\pi(i)}, \Gamma \cup \{R_{\pi(1)}, \ldots, R_{\pi(i-1)}\}, \Omega)$$

This cost function encourages the reuse of intermediate relations, rewarding languages that possess internal structure. Integrating such a cost function into an objective may allow the model to find a trade-off between an approach where the language is predefined, and the fully flexible method presented in this chapter. It would guide the search towards languages that are not only compact in size and arity but also conceptually simple and interpretable.

4.6 Conclusion

This chapter introduced LFA, a new method for constraint acquisition that discards the assumption that the target constraint language is known a priori by the oracle. Our central contribution is a method that learns the language as part of learning the model by jointly searching for a compact language (in terms of arity and number of relations) and a set of scopes for each relation, which together describe a network consistent with the training examples. We formalized the decision problem underlying

LFA and proved it is NP-complete. We then proposed an algorithm based on iterative calls to a Weighted Partial Max-Sat solver. We show that our method can be applied to a wide range of problems such as Sudoku, Jigsaw Sudoku, Exam Timetabling and Nurse Rostering. Finally, we analyzed in detail how the ratio of positive to negative examples and the optimization criterion influence accuracy.

Our method demonstrates strong generalization capabilities, achieving high accuracy with relatively small training sets. However, the last percentage points of accuracy may demand disproportionately many examples; this calls for richer notions of simplicity that capture some structure of the problem. In the next chapter, we explore how to learn a constraint network considering an advanced notion of simplicity in the arrangement of the scopes of constraints, which addresses one of the main limitations of the current approach.

Chapter 5
Learning Compact Representations of the Scopes

There typically exist many constraint networks that are consistent with a given training set, so the generalization ability of a constraint 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.

This chapter is primarily based on the paper accepted for publication in the Proceedings of the 28th European Conference on Artificial Intelligence (ECAI-25).

Contents

5.1 Introduction

The previous chapter introduced LFA, a method that successfully removes the need for a predefined constraint language by learning it alongside the constraint network. However, as discussed in its limitations (Section 4.5), this approach has a fundamental weakness: it is agnostic to any underlying structure in the set of scopes where constraints are applied. LFA learns a simple, flat list of scopes for each relation of the learned language. By failing to recognize high-level patterns, LFA may learn a network that overfits the training set, requiring a large number of examples to correctly discard all spurious constraints.

This chapter addresses this challenge and proposes that learning a more compact representation of the constraint network is key to better generalization to unseen assignments. Instead of learning an explicit, flat list of constraints in extension, we aim to learn a model that captures the underlying structure of the problem. We hypothesize that a model that can express patterns like "apply a constraint to all scopes of variables in the same row" will be more robust, less prone to overfitting, and more interpretable.

To achieve this, we first introduce a novel representation, which we call templates, that can capture some structured constraint networks. A template associates each variable with a set of attributes and contains a set of rules, which produce constraints based on these attributes. Each individual rule has the potential to generate many constraints at once. We then describe a framework that learns such a template directly from 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 data but may be overly specific. Second, our core contribution is an algorithm that refines this network with heuristics devised to output a template that contains few rules and attributes, but that remains consistent with the examples. We run experiments on a wide array of benchmarks and show that combining LFA with our template-based algorithm yields constraint networks that have a higher generalization ability.

Closely Related Work. The idea of templates is conceptually similar to other intermediate representations of learning approaches in the literature. COUNT-CP [21] and URPILS introduced in Chapter 2 learn first-order logic constraints. The method presented in [23] uses a rule-based language using Inductive Logic Programming. GenCon learns parameterized constraint models (i.e. capable of modeling varying instances of the same problem) from existing instance-specific models [41]. The method learns a classifier that predicts whether a candidate constraint, parameterized by some parameters, belongs to the target constraint network or not. The method extracts interpretable decision rules from the classifier to construct constraint specifications that can generate a constraint network for any specific problem instance. This allows GenCon to learn constraints that are applicable to different instances of the same problem without needing to redefine them for each instance.

A crucial distinction, however, is that these existing approaches require the relevant properties to be provided as part of the input. For instance, they may need predefined variable partitions, features, or relations between variables. In contrast, our framework is designed to learn both the attributes and the rules that rely on them simultaneously from the raw examples. We introduce the specific structure of templates, which allows us to formulate this joint learning problem as a series of CP optimization models.

The rest of this chapter is organized as follows. Section 5.2 introduces templates as a means to represent constraint networks concisely. Section 5.3 describes our algorithm for learning templates from examples. Section 5.4 details the Constraint Programming models used as subroutines within our learning algorithm. Section 5.5 presents our experimental evaluation, comparing our approach to LFA. Then, Section 5.6 discusses future work and research directions. Finally, Section 5.7 concludes and discusses future research directions.

5.2 Compact Representations

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 10 (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 $\Phi = (\phi_1, \ldots, \phi_m)$ over X, we denote $w(\Phi) = (w(\phi_1), \ldots, w(\phi_m))$. Given a constant $w \in \mathbb{N}$ and a set of variables X, we denote w^X the attribute ϕ such that $\forall x \in X, \phi(x) = w$ (all variables have the same attribute value w).

Definition 11 (Rule)**.** *Given a set of variables X and a sequence of attributes $\Phi = (\phi_1, \ldots, \phi_m)$ 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 $\{1, \ldots, m\} \times \{1, \ldots, r\}$ called the* selector*, and f is a function $f : \mathbb{N}^{|J|} \to \{0, 1\}$ called the* trigger*. Given a scope $S = (x_1, \ldots, x_r) \in X^r$ 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 $(x_1, x_2, \ldots)$ 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 12 (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 5.2.1 (Sudoku)**.** *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. We can generate all constraints with three simple rules, each applying to distinct pairs of variables (x_u, x_v).

Row Rule: $(R_{\neq}, ((\phi_1, x_u), (\phi_1, x_v)), f)$ 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: $(R_{\neq}, ((\phi_2, x_u), (\phi_2, x_v)), f)$*, 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: $(R_{\neq}, ((\phi_3, x_u), (\phi_3, x_v)), f)$*, 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:

$$f_3(\phi_1(x_u), \phi_1(x_v), \phi_2(x_u), \phi_2(x_v)) = 1$$
$$\iff$$
$$\left(\left\lfloor \frac{\phi_1(x_u) - 1}{3} \right\rfloor = \left\lfloor \frac{\phi_1(x_v) - 1}{3} \right\rfloor\right) \wedge \left(\left\lfloor \frac{\phi_2(x_u) - 1}{3} \right\rfloor = \left\lfloor \frac{\phi_2(x_v) - 1}{3} \right\rfloor\right)$$

with selector $J_3 = ((\phi_1, x_u), (\phi_1, x_v), (\phi_2, x_u), (\phi_2, x_v))$.

Example 5.2.2 (Nurse Rostering)**.** *Let us consider a constraint network for a Nurse Rostering problem. An instance of this problem is defined by a number of days* d_{max}*, shifts per day* s_{max}*, slots per shift* k_{max}*, and number of nurses* n_{max}*. Given an instance* $(d_{max}, s_{max}, k_{max})$*, a basic network consists of* $d_{max} \times s_{max} \times k_{max}$ *variables* $X = \{x_{d,s,k} \mid d \in [1, d_{max}], s \in [1, s_{max}], k \in [1, k_{max}]\}$ *of domain* $D = [1, n_{max}]$ *each representing a slot that can be filled by a nurse. If an assignment gives the value* n *to a variable* $x_{d,s,k}$*, it means that nurse* n *is assigned the* s *shift on day* d*. The constraints are of two main types: no nurse works two slots on the same day (*intra-day*), and no nurse works the last shift of one day and the first shift of the next (*inter-day*). A template can represent this network compactly.*

First, we define two attributes for each variable $x_{d,s,k}$*:* $\phi_{day}(x_{d,s,k}) = d$*, and its shift,* $\phi_{shift}(x_{d,s,k}) = s$*. We can generate all constraints with two simple rules, each applying to distinct pairs of variables* (x_u, x_v)*.*

Intra-Day: A rule $(R_{\neq}, ((\phi_{day}, x_u), (\phi_{day}, x_v)), f_1)$ *produces a disequality relation* $(R_{\neq})$ *constraint between any two slots* (x_1, x_2) *if their day attributes are identical* $(((\phi_{day}, x_1), (\phi_{day}, x_2))$ *indicate that we only consider the first attribute of both variables, which is the day attribute). The trigger for this rule is* $f_1(d_1, d_2) = 1 \Leftrightarrow d_1 = d_2$.

Inter-Day: A second rule $(R_{\neq}, J, f_2)$ *produces a* $(R_{\neq}, (x_u, x_v))$ *constraint if* x_u *is in the last shift of one day and* x_v *the first shift of the next. The selector* J *is* $((\phi_{day}, x_u), (\phi_{day}, x_v), (\phi_{shift}, x_u), (\phi_{shift}, x_v))$ *indicates that we consider both attributes of both variables, and* $f_2(d_1, d_2, s_1, s_2) = 1 \Leftrightarrow (d_2 = d_1 + 1) \wedge (s_1 = s_{max} - 1) \wedge (s_2 = 0)$.

As Example 5.2.1 illustrates, multiple templates can represent the same constraint network. If only the attribute values change, we say that the templates are *equivalent*.

Definition 13 (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 5.2.1 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 (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.

5.3 Learning Templates

Given a training set E over (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.

5.3.1 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 it 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 5.3.2 and the process to guess a suitable width for a new attribute (GuessAttributeWidth) is described in Section 5.3.3.

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 5.3.2) and a heuristic to determine a suitable width for new attributes (Section 5.3.3). 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). *(We recall that given a constant $w \in \mathbb{N}$ and a set of variables X, w^X denotes the attribute ϕ such that $\forall x \in X, \phi(x) = w$. This attribute is added at the end of the sequence of attributes of T and is then used in the next line as a new attribute of width w.)* 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

Algorithm 1: Learning a template

Input: A training set E; a constraint network $N = (X, D, C)$ consistent with E; a set of triggers Λ.

Output: A template T such that $N(T) \subseteq C$ and $N(T)$ is consistent with E.

```
1  T ← (X, D, ∅, ∅);
2  α ← 0.3;
3  while N(T) is not consistent with E do
4      w ← GuessAttributeWidth(T, N, Λ, α);
5      if w ≥ 0 then
6          T ← T + (w*)^X;
7          T ← SaturateWithNewRules(T, N, Λ, α);
8      if N(T) is not consistent with E then
9          α ← α × 0.9;
10         T ← SaturateWithNewRules(T, N, Λ, α);
11 return T;
```

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).

5.3.2 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 14 (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 13) and formalize it through the concept of admissible rules modulo equivalence.

Definition 15 (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 $AdmEq(T, N, \Lambda, lb) = \emptyset$.

5.3.3 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

Algorithm 2: SaturateWithNewRules(T, N, Λ, α):

Input: A template T; A constraint network $N = (X, D, C)$; a set of triggers Λ; a real α.

Output: An updated template with new rules added.

$lb \leftarrow \alpha \times \frac{|N(T)|}{\max(|P(T)|,1)}$;
while $AdmEq(T, N, \Lambda, lb) \neq \emptyset$ **do**
  $(T', R, J, f) \leftarrow \underset{(T',R,J,f)\in AdmEq(T,N,\Lambda,lb)}{\operatorname{argmax}} \Big(|N(T' + (R, J, f))| - |N(T)|\Big)$;
  $T \leftarrow T' + (R, J, f)$;
  $lb \leftarrow \alpha \times \frac{|N(T)|}{|P(T)|}$;
return T;

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:

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

Algorithm 3: GuessAttributeWidth(T, N, Λ, α)

Input: A template T; a constraint network $N = (X, D, C)$; a set of triggers Λ; a real α.

Output: The optimal width w^* for a new attribute, or -1 if no admissible attribute exists.

$cov_{max} \leftarrow cov(|X| - 1, T, N, \Lambda)$;
$lb \leftarrow \alpha \times \frac{|N(T)|}{\max(|P(T)|,1)}$;
if $cov_{max} > lb$ **then**
  $w^* \leftarrow \underset{\substack{0\leq w<|X| \\ cov(w,T,N,\Lambda)>lb}}{\operatorname{argmax}} \Big(cov(w, T, N, \Lambda) - \frac{w}{|X|-1} \cdot cov_{max}\Big)$;
  return w^*;
return -1;

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.

5.3.4 Termination, Correctness and Complexity

We begin by establishing that Algorithm 1 is guaranteed to terminate and produce a template that is consistent with the training set E under certain mild conditions. 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 2. *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.*

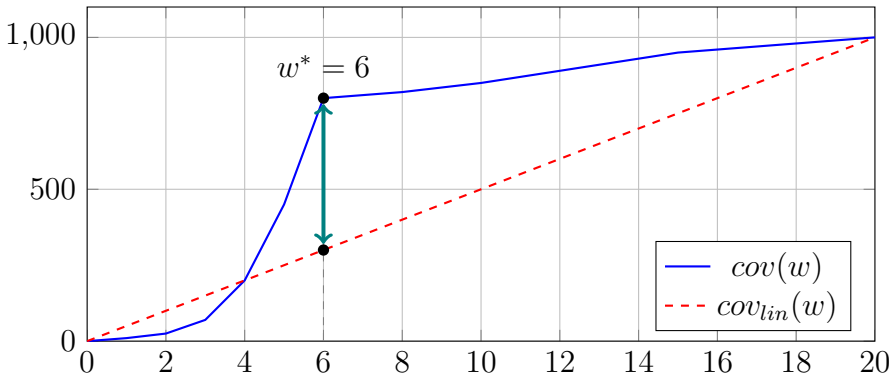


Figure 5.1: Illustration of the MCAE heuristic with a hypothetical $cov(w)$ and its linear expectation $cov_{lin}(w)$. The optimal width w^* is chosen at the point where the gap between the actual cover $cov(w)$ and the linear expectation $cov_{lin}(w)$ is maximized (indicated by the teal double-headed arrow).

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. We intentionally do not use the value r in the new attribute for any variable. This ensures that no other scope of r variables can form a consecutive sequence of attribute values.

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. □

Now, we show that the problem of determining whether $AdmEq(T, N, \Lambda, lb)$ is non-empty is NP-hard. This justifies our decision to employ a CP solver for computing the admissible rules. Let Φ be the set of all attributes in the template T. We represent each trigger function $f_i \in \Lambda$ of arity r as the list of tuples subset of $[0, w(\Phi)]^r$ for which f_i returns 1. Since any equivalent template T' with attributes Φ' must satisfy $w(\Phi') = w(\Phi)$ (by Definition 13), it suffices to consider only tuples within the domain $[0, w(\Phi)]^r$ when encoding trigger functions.

Theorem 3. *Given a template T, a constraint network N, a set of triggers Λ, and a threshold lb, the problem of determining whether $AdmEq(T, N, \Lambda, lb)$ is non-empty is NP-hard.*

Proof. The decision problem can be stated as follows:

- **Input:** A template $T = (X, D, \Phi, P)$, a constraint network $N = (X, D, C)$, a set of triggers Λ, and a real number lb.
- **Question:** Does there exist a tuple (T', R, J, f) such that $T' \equiv T$ and $(R, J, f) \in Adm(T', N, \Lambda, lb)$?

We prove NP-hardness by a reduction from the CLIQUE problem, which is known to be NP-complete. Given a graph $G = (V, E)$ and an integer $k \geq 1$, the CLIQUE problem asks whether there exists a subset of vertices of size k such that every pair of vertices in this subset is connected by an edge in E (a clique). Given a CLIQUE instance (G, k), we construct an instance of our decision problem as follows:

- $T = (X, D, \Phi, P)$ is a template where $X = V \cup \{v_\star\}$ (i.e. the set of variables plus a special variable $v_\star$), $D = \{0\}$, $\Phi = (\phi_1)$ such that $\forall v \in X, \phi_1(v) = 1$ ($w(\phi_1) = 1$), and $P = \emptyset$.
- $N = (X, D, C)$ is a constraint network where $X = V \cup \{v_\star\}$, $D = \{0\}$, and $C = \{(\textsc{Edge}, (u, v)) \mid \{u, v\} \in E\}$ where Edge is the relation that rejects all tuples.
- $\Lambda = \{f_{\textsc{both1}}\}$, where $f_{\textsc{both1}}(a, b) = 1$ if and only if $(a, b) = (1, 1)$.
- $lb = k(k-1) - 1$.

This construction can be done in polynomial time. We now show that G has a clique of size k if and only if $AdmEq(T, N, \Lambda, lb)$ is non-empty.

Assume G has a clique $V' \subseteq V$ with $|V'| = k$. We construct a valid solution (T', R, J, f). Let $\phi'_1 : V \to \{0, 1\}$ be an attribute defined as $\phi'_1(v) = 1$ if $v \in V'$ and $\phi'_1(v) = 0$ otherwise. This implies that $\phi'_1(v_\star) = 0$. Let $T' = (X, D, (\phi'_1), \emptyset)$. T' is equivalent to T because P is empty and $w(\phi'_1) = w(\phi_1)$. Consider the rule $\rho = (\textsc{Edge}, ((1, 1), (1, 2)), f_{\textsc{both1}})$. The set of constraints produced $N(T', \rho)$ is the set $\{(\textsc{Edge}, (u, v)) \mid \phi'_1(u) = 1 \text{ and } \phi'_1(v) = 1\}$. This is precisely the set $\{(\textsc{Edge}, (u, v)) \mid (u, v) \in (V')^2, u \neq v\}$. Since V' is a clique, for any distinct pair (u, v) of V', the edge $\{u, v\}$ is in E. Thus, $N(T', \rho) \subseteq C$. The number of constraints produced $|N(T', \rho)|$ is $k(k-1)$ which is greater than $lb = k(k-1) - 1$. Therefore, $\rho \in Adm(T', N, \Lambda, lb)$, and $AdmEq(T, N, \Lambda, lb)$ is non-empty.

Assume $AdmEq(T, N, \Lambda, lb)$ is not empty, and let (T', R, J, f) be a tuple in the set. We show that this implies that G has a clique of size at least k. $T' = (X, D, (\phi'_1), \emptyset)$ must be equivalent to T, so $w(\phi'_1) = 1$. This attribute $\phi'_1 : V \to \{0, 1\}$ defines a subset of vertices $V' = \{v \in V \mid \phi'_1(v) = 1\}$. The input constraint network N is over the language $\{\textsc{Edge}\}$, thus R must be $\textsc{Edge}$. Λ contains only $f_{\textsc{both1}}$, so f must be $f_{\textsc{both1}}$, and V' cannot be empty because $|N(T', R, J, f) \setminus N(T')| > lb$ implies that the rule (R, J, f) produces at least one constraint with the trigger $f_{\textsc{both1}}$. J must be a pair of pairs from $\{(1, 1), (1, 2)\}$ as there is only one attribute and $\textsc{Edge}$ is a binary relation ($|J|$ is equal to the arity of $\textsc{Edge}$, which is 2). By construction, for any variable $v \in X$, there is no constraint $(\textsc{Edge}, (v, v_\star))$ or $(\textsc{Edge}, (v_\star, v))$ in C. Thus J cannot be $((1, 1), (1, 1))$ or $((1, 2), (1, 2))$ as these would produce for any variable $v' \in V'$ a constraint $(\textsc{Edge}, (v', v_\star))$ which is not in C. The only choices for J is $((1, 1), (1, 2))$ or $((1, 2), (1, 1))$. As $\textsc{Edge}$ is a symmetric relation, we assume $J = ((1, 1), (1, 2))$ without loss of generality. The set of constraints produced by the rule $N(T', (\textsc{Edge}, ((1, 1), (1, 2)), f_{\textsc{both1}}))$ is $\{(\textsc{Edge}, (u, v)) \mid u \in V', v \in V', u \neq v\}$. This means that for every pair of distinct vertices (u, v) in V', the edge $\{u, v\}$ must be in E. Thus, V' is a clique in G. The condition $|N(T', p) \setminus N(T')| > lb$ implies that the number $|V'|(|V'| - 1)$ of pairs of distinct vertices of V' is greater than $k(k-1)$. The existence of such a rule thus implies the existence of a clique of size at least k in G. □

5.4 The 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 5.5 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 $selected_{J1}(R', J_1, f_1) \equiv X_R(R') \wedge X_{f1}(f_1) \wedge X_{J1}(J_1)$ and $selected_{J2}(R', J_2, f_2) \equiv X_R(R') \wedge X_{f2}(f_2) \wedge X_{J2}(J_2)$ 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 $trigger(S, J, f)$ which evaluates to `True` iff:

$$f(\phi_{i_1}(S[k_1]), \ldots, \phi_{i_t}(S[k_t]))$$

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 $(J, f) \in \Lambda' \times \mathcal{J}$:

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

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:

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

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 $trigger(S, J_1, f_1) \wedge trigger(S, J_2, f_2)$ 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 5.3.2, 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 $cov(n-1)$ 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 $cov(n-1)$ calculated in the first phase and maximize

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

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).

Example 5.4.1 (Illustrative Example)**.** *Consider a* 2×2 *grid with variables* $X = \{x_1, x_2, x_3, x_4\}$*, where* x_1, x_2 *are the first row and* x_3, x_4 *are the second row.*

Suppose the initial baseline network N *contains the binary difference constraint* $(\neq)$ *on all pairs in the same row:* $\{(x_1, x_2), (x_2, x_1), (x_3, x_4), (x_4, x_3)\}$*. We start with an empty template* T *and want to find a new attribute* ϕ_{new} *and a rule to cover these constraints. Let* $\Lambda' = \{=, \neq\}$*.*

The CP model variables are defined as follows:

- ***Attribute values and width.*** *Integers* $\{v_{x_1,new}, v_{x_2,new}, v_{x_3,new}, v_{x_4,new}\}$ *with domain* $[0, 3]$*. Integer variable* $d \in [0, 3]$*.*
- ***Rule Selection:*** *Boolean variables to select the relation and function and selector for both parts of the rule:*
  - $X_R(\neq)$ *since* N *only contains constraints with the* $\neq$ *relation.*
  - $X_{f1}(f), X_{f2}(f)$ *with* f *the Boolean functions associated with the binary relations in* $\{=, \neq\}$*.*
  - $X_{J1}(J), X_{J2}(J)$ *where* J *is a selector. Since we only have one attribute* ϕ_{new}*, the possible selectors are:*
    - $J_A = ((new, 1), (new, 2))$
    - $J_B = ((new, 2), (new, 1))$
    - $J_C = ((new, 1), (new, 1))$
    - $J_D = ((new, 2), (new, 2))$

    In the following, we will focus on selector J_A *for illustration which refers to* ϕ_{new} *of the first and then the second variables considered.*
- ***Coverage:*** *Boolean variables* $c_{(\neq,(x_i,x_j))}$ *for each target constraint in* N *not yet produced by* T *(here, all of them because* T *is empty).*

The constraints of the CP model first ensure that d *is the width of the new attribute* $(\forall x \in X, v_{x,new} \leq d)$*. The model also ensures that exactly one function and one selector is chosen for each part of the rule (e.g.,* $\sum X_{f1} = 1$*) and* $X_R(\neq)$ *is set to* `True` *(since we only consider the* $\neq$ *relation here). Then, the key constraints of the CP model are as follows:*

New constraints produced. *For target constraints like* $(\neq, (x_1, x_2))$*, the*

variable $c_{(\neq,(x_1,x_2))}$ must be **`True`** *only if the rule produces it. For example, considering the selector J_A and the Boolean function $f_=$ associated with $=$, we have the implications:*

$$c_{(\neq,(x_1,x_2))} \Rightarrow \neg\Big(X_R(\neq) \wedge (X_{J1}(J_A) \wedge X_{f1}(f_=))\Big) \vee \Big(v_{x_1,new} = v_{x_2,new}\Big)$$
$$c_{(\neq,(x_1,x_2))} \Rightarrow \neg\Big(X_R(\neq) \wedge (X_{J2}(J_A) \wedge X_{f2}(f_=))\Big) \vee \Big(v_{x_1,new} = v_{x_2,new}\Big)$$

Similar implications exist for other combinations of J and f. The interpretation is that if the rule uses this relation/selector/function and the values match, then the constraint can be produced. Because of the maximization objective, the solver will try to make $c_{(\neq,(x_1,x_2))}$ true whenever possible.

Forbidden constraints. *For pairs not in N (e.g. $(\neq,(x_1,x_3))$), the rule must not produce it. We use auxiliary variables t to ensure the conjunction does not produce it. For a specific combination as J_A and $f_=$, we have:*

$$\neg\Big(X_R(\neq) \wedge (X_{J1}(J_A) \wedge X_{f1}(f_=))\Big) \vee \neg\Big(v_{x_1,new} = v_{x_3,new}\Big) \vee t_{(\neq,(x_1,x_3))}$$
$$\neg\Big(X_R(\neq) \wedge (X_{J2}(J_A) \wedge X_{f2}(f_=))\Big) \vee \neg\Big(v_{x_1,new} = v_{x_3,new}\Big) \vee \neg t_{(\neq,(x_1,x_3))}$$

If the first part uses this selector/function and the values match, t becomes **`True`***. If the second part does the same, t becomes* **`False`***. Thus, they cannot both "trigger".*

A possible solution is the following assignment:

- ***Values:*** *$v_{x_1,new} = 0, v_{x_2,new} = 0$, $v_{x_3,new} = 1, v_{x_4,new} = 1$. Width $d = 1$.*
- ***Rule:*** *$X_R(\neq), X_{J1}(J_A), X_{f1}(=)$ and $X_{J2}(J_A), X_{f2}(=)$ all set to* **`True`** *(others to* **`False`***). Since our model requires a conjunction of two parts but we only need one, we simply repeat the equality check in both parts with the same selector.*

The rule will produce all "row" constraints (e.g. $v_{x_1,new} = v_{x_2,new}$ and $v_{x_4,new} = v_{x_3,new}$) and only them (since $v_{x_1,new} \neq v_{x_3,new}$)

Our method first maximizes the number of covered constraints (which is 4 here) without considering the width. Then, it maximizes the MCAE criterion to find the best width.

5.5 Experimental Evaluation

In this section, we evaluate our method (i.e the full framework including the baseline acquisition method), that we denote TAcq, experimentally on several benchmark problems. For each benchmark, we will compare the classification accuracy of the interpretation of the template learned by our method and that of the network obtained 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 our algorithm.

5.5.1 Implementation

As the method used to generate the initial network, we could use any constraint acquisition method, such as ConAcq.1 [10, 12] (with the most specific network it suggests) or BayesAcq [35]. However, we naturally chose to use the LFA method from Chapter 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.

We fix the trigger language Λ to be trigger that can be expressed as conjunctions of two triggers from the set of Boolean functions associated with the binary relations $\{all, =, \neq, <, >, \leq, \geq, +_1, -_1\}$ where *all* is the universal relation, $+_1$ is the integer successor relation, and $-_1$ is the integer predecessor relation. Because we use the conjunction of two triggers, we can express some quaternary relations.

We have implemented the full framework described in Section 2.2, using the LFA baseline from Chapter 4. The underlying CP solver is Google OR-Tools [32]. The complete source code of the implementation and datasets for the following experiments are hosted on GitHub: `github.com/hareski/tacq` under the Academic Free License (AFL-3.0). This implementation is also available as a Python package named `tacq` on the Python Package Index (PyPI) and can be installed using the command `pip install tacq`. The package provides a simple interface for learning templates from training sets stored in CSV files, making it easy to use in practice.

All experiments were conducted on an AMD Epyc 9554 processor (utilizing 8 cores per run) and 16GB of RAM.

5.5.2 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.

We use the same benchmarks as in the experimental evaluation of the baseline method LFA presented in Chapter 4, with identical parameters: Sudoku, Jigsaw Sudoku, Schur's Lemma, Subgraph Isomorphism, N-Queens, Golomb Ruler, Exam Timetabling, and Nurse Rostering. For a presentation of the target networks of these benchmarks, we refer the reader to the previous chapter. We present here only a possible template whose interpretation returns the same constraint network as the target constraint network used to generate the training set.

Sudoku A candidate template contains three attributes ϕ_{row}, ϕ_{col}, ϕ_{square} representing the rows, columns, and squares the variables belong to. Three rules should produce the binary disequality respectively when variables share the same row, column, or square attribute value. A complete template is presented in Example 5.2.1.

Jigsaw Sudoku A candidate template contains three attributes ϕ_{row}, ϕ_{col}, ϕ_{jigsaw} representing the rows, columns, and jigsaw pieces the variables belong to. Three rules should produce the binary disequality respectively when variables share the same row, column, or jigsaw piece attribute value.

Schur's Lemma We ran experiments on this problem with $n = 9$, which is the parameter with the highest number of solutions (546). A candidate template contains an attribute ϕ_{ind} representing the indices of the variable and a rule that produces a ternary NotAllEqual relation applied on x_i, y_j, z_k if $\phi_{ind}(x_i) + \phi_{ind}(y_j) = \phi_{ind}(z_k)$

(for a scope (x_1, x_2, x_3), the rule would be (NotAllEqual, $((\phi_{ind}, x_1)), (\phi_{ind}, x_2)$, $(\phi_{ind}, x_3)), f)$ with $f(a, b, c) = 1$ if $a + b = c$).

Subgraph Isomorphism A candidate template contains only one attribute ϕ_{ind} representing the variable indices. A first rule produces the disequality constraints for all scopes, ensuring that the mapping between the vertices of H and G is a one-to-one function. A second rule produces a constraint with a binary table relation R representing the edges of G applied to all (x_i, x_j) such that (i, j) is an edge in G. This rule is formally $(R, ((\phi_{ind}, x_i), (\phi_{ind}, x_j)), f)$ with $f(a, b) = 1$ iff $\phi_{ind}(x_a) = (\phi_{ind}(x_b)+1$ mod 5). For this benchmark, negative examples are generated as paths and closed walks of G computed using a randomized value selection.

N-Queens (coordinate-based) 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. With the coordinate-based model, there is no clear way to define the necessary attributes and rules compactly. Therefore, we do not provide a candidate template for this benchmark.

Golomb Ruler A candidate template can represent this with a single attribute, ϕ_{ind}, which stores the index of each variable. A single rule produces the required quaternary constraints. This rule applies a relation R (which enforces $|v_1 - v_2| \neq |v_3 - v_4|$) to any scope of four variables (x_i, x_j, x_k, x_l) if $i < j$ and $k < l$ (i.e., the rule is $(R, ((\phi_{ind}, x_a), (\phi_{ind}, x_b), (\phi_{ind}, x_c), (\phi_{ind}, x_d)), f)$ with $f(a, b, c, d) = 1$ iff $a < b$ and $c < d$).

Exam Timetabling A candidate template contains one attribute $\phi_{semester}$ representing the semesters the variables belong to. A first rule, $(\neq, (), f)$ with $f() = 1$, produces the binary disequality for all pairs of variables. This corresponds to an AllDifferent over all variables. A second rule produces a binary table constraint that ensures that two courses from the same semester are not scheduled on the same day. This second rule is, given any scope (x_i, x_j), $(R, ((\phi_{semester}, x_i), (\phi_{semester}, x_j)), f)$ with $f(a, b) = 1$ iff $a = b$ and R the relation that rejects the pairs of courses from

the same semester (i.e. $(a, b) \notin R$ iff $a \mod t \times r = b \mod t \times r$).
We recall the parameters of the three instances used in our experiments:

[#**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.

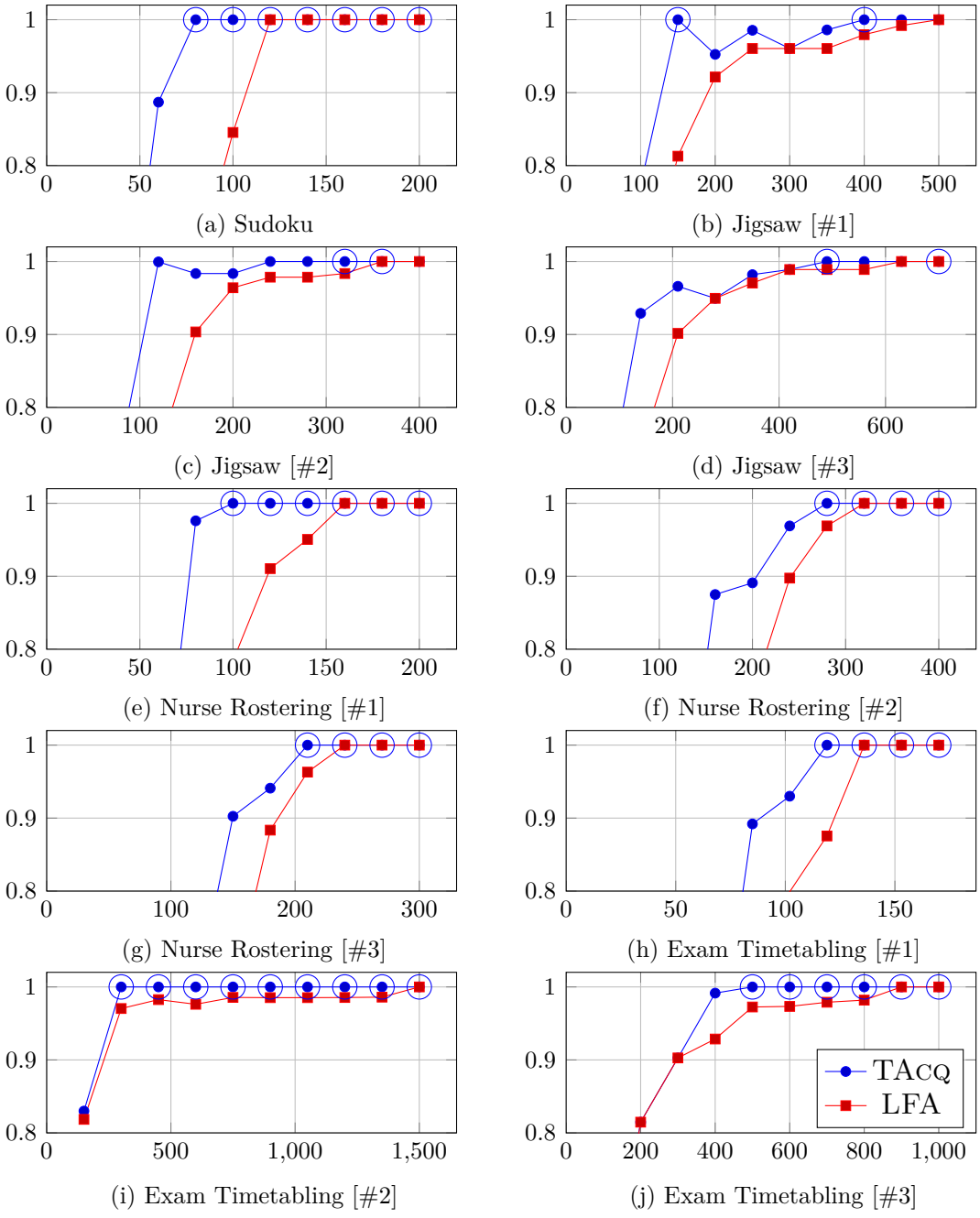
Nurse Rostering A candidate template is given in the Example 5.2.2.
We recall the parameters of the three instances used in our experiments:

[#**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 Chapter 4). 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_{suc}, and f^3_{suc} can be expressed by a conjunction of two binary successor functions. As a result, the conditions of Proposition 2 are satisfied, guaranteeing termination in our experiments.

5.5.3 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.



(a) Sudoku

(b) Jigsaw [#1]

(c) Jigsaw [#2]

(d) Jigsaw [#3]

(e) Nurse Rostering [#1]

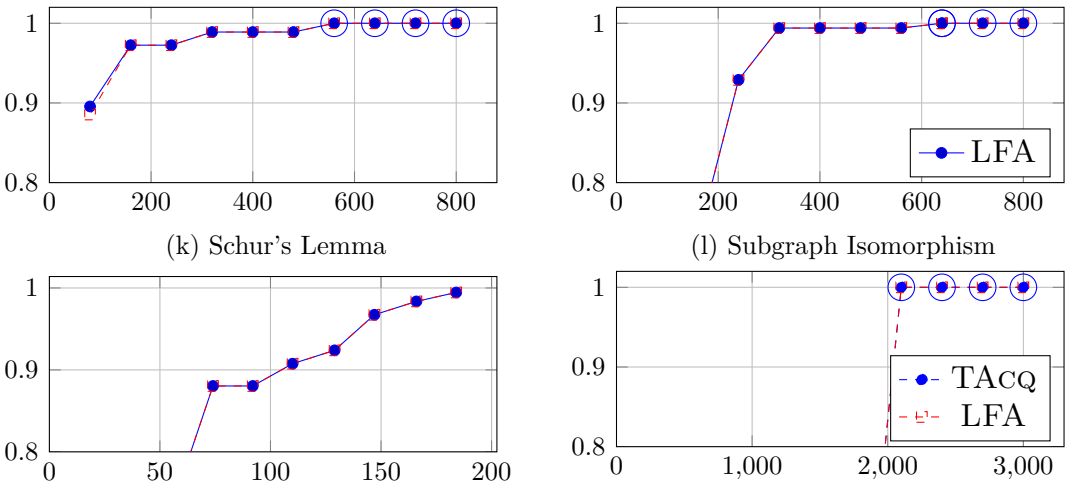
(f) Nurse Rostering [#2]

(g) Nurse Rostering [#3]

(h) Exam Timetabling [#1]

(i) Exam Timetabling [#2]

(j) Exam Timetabling [#3]



(k) Schur's Lemma

(l) Subgraph Isomorphism

(m) 8-Queens (coordinates model)

(n) Golomb Ruler (10 variables)

Figure 5.2: 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.

Accuracy A summary comparing the learning curves of TACQ and LFA for representative benchmarks is shown in Figure 5.2.

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 the assignments of 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.

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 5.1. 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.

5.5.4 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

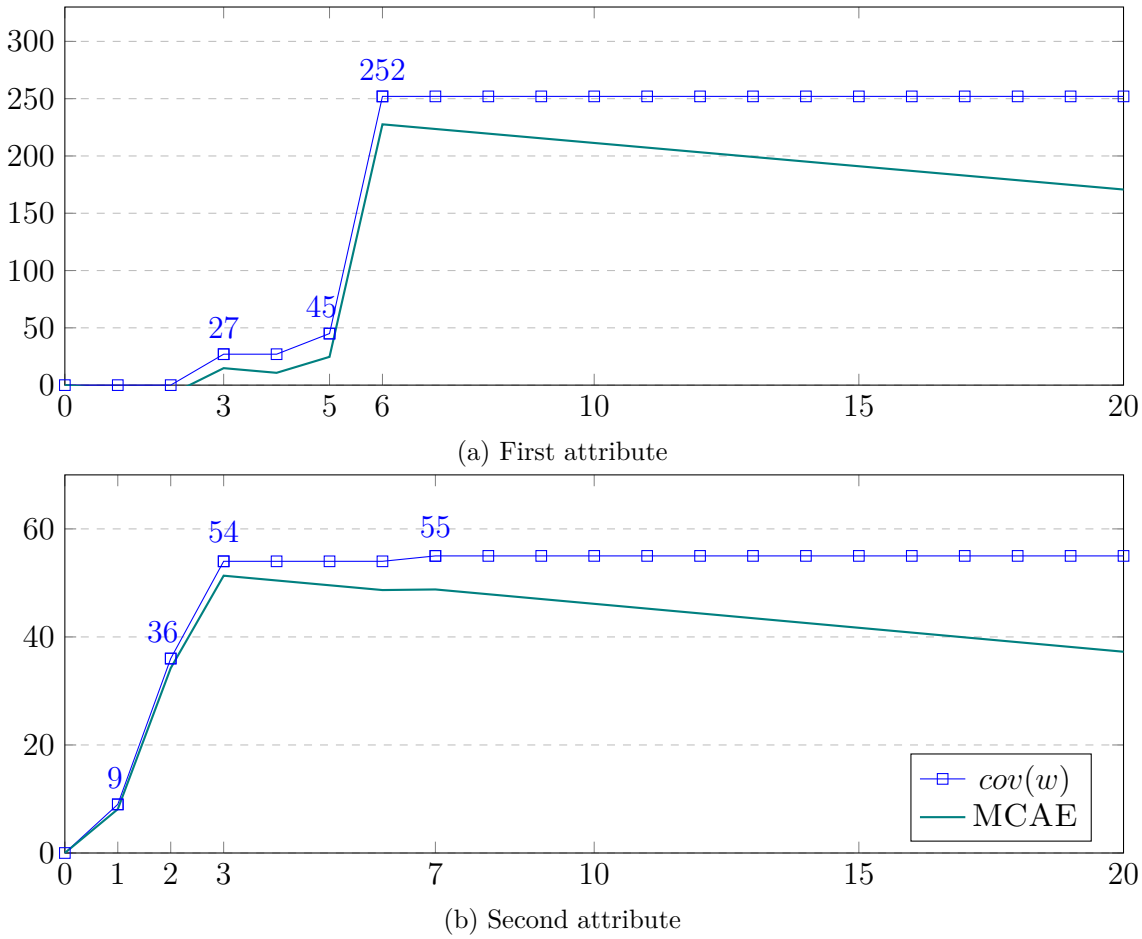
Benchmark	$\|E\|$	LFA	TAcq
Sudoku	80	1m	15h 32m 9s
Jigsaw [#1]	400	39s	31h 36m 27s
Jigsaw [#2]	240	46s	27h 20m 9s
Jigsaw [#3]	490	41s	33h 54m 58s
Schur's Lemma	560	4s	1h 45m 4s
Subgraph Isomorphism	640	12s	19s
8-Queens	184	10s	1m 22s
Golomb Ruler	2100	3m 59s	2h 42m 20s
Exam Timetabling [#1]	119	1s	2s
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 5.1: Comparison of runtimes for LFA and TAcq. Runtimes for TAcq include the time taken by LFA to learn the initial network N.

learning process for an instance of the Nurse Rostering problem and with the Sudoku problem.

Nurse Rostering

First, we analyze the learning process for the instance [#3] of the Nurse Rostering problem (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 5.3 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.



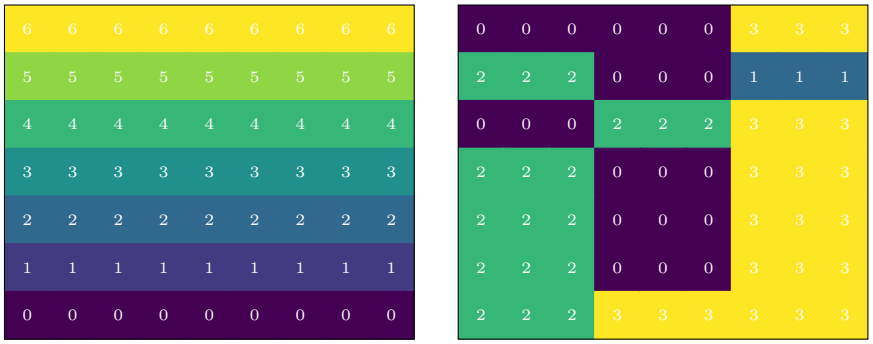
(a) First attribute

(b) Second attribute

Figure 5.3: 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.

The Figure 5.4 illustrates the learned attributes for the instance [#3] of Nurse Rostering. We discuss the interpretation of these attributes and the behavior of the MCAE heuristic in detail below.

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.



(a) Attribute 1 (ϕ_1) (b) Attribute 2 (ϕ_2)

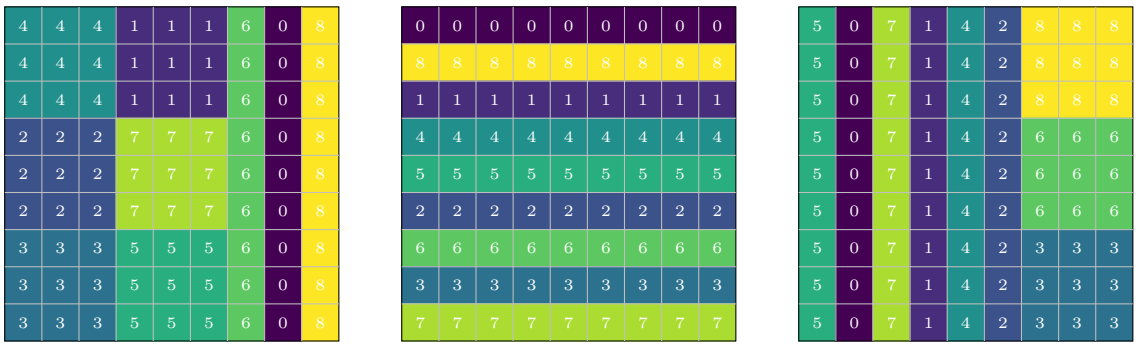
Figure 5.4: Visualization of the learned attributes for the instance [#3] of Nurse Rostering with 210 examples in the training set. Each cell represents the value $\phi_i(x_{j,k})$ with i the attribute number, and (j, k) are the coordinates of the variable on a 7×9 grid. Attribute ϕ_1 (left) captures the days of the week (values $0 - 6$), while attribute ϕ_2 (right) represents shifts within each day, with a structure allowing the identification of consecutive shifts across days.

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 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.

Sudoku

Figure 5.5 illustrates the attributes learned for the Sudoku problem. The first attribute ϕ_1 does not capture the rows of the variable in the Sudoku grid. Instead, it captures for some variables the boxes they belong to, and for others, the columns. The second attribute ϕ_2 captures the rows. Finally, the third attribute ϕ_3 captures the columns and the boxes that are not already captured by the first attribute ϕ_1.



(a) Attribute 1 (ϕ_1) (b) Attribute 2 (ϕ_2) (c) Attribute 3 (ϕ_3)

Figure 5.5: Visualization of the attributes learned for the Sudoku instance with 80 examples in the training set. Each cell represents the value $\phi_i(x_{j,k})$ with i the attribute number, and (j, k) are the coordinates of the variable on a 9×9 grid.

This representation using ϕ_1 and ϕ_3 which both represent the boxes and columns of the Sudoku grid, is used to produce the rules that are needed to model the Sudoku problem. The first rule produced by the template is $(\neq, (x, y))$ when $\phi_1(x) = \phi_1(y)$, the second rule is $(\neq, (x, y))$ when $\phi_2(x) = \phi_2(y)$. Finally, the third rule is $(\neq, (x, y))$ when $\phi_3(x) = \phi_3(y)$. These rules can be interpreted as "no two variables in the same box can have the same value", "no two variables in the same row can have the same value", and "no two variables in the same column can have the same value" with the caveat that the first and third rules cover both boxes and columns, while the second rule only covers rows.

There is no reason for our algorithm to prefer the classic representation using row, column, and box attributes for the Sudoku problem. This highlights the fact that our method is not biased towards a specific representation of the problem. It learns a representation that is compact, but not necessarily the easiest to interpret by a human expert.

5.6 Perspectives

The results of our experiments demonstrate that templates can effectively capture the structure of constraint networks, leading to a significant reduction in the number of training examples required to achieve high accuracy on a wide range of problems. However, there are several avenues for future work that could further enhance the applicability of our approach.

5.6.1 Generalization to Other Instances

A significant frontier for constraint acquisition is moving beyond instance-specific models to those that can generalize across different instances of the same problem. The current TAcq framework learns a template that is tied to the specific structure of the training instance (e.g. a fixed number of days, a fixed grid size). A promising direction for future research is to extend this framework to learn the fundamental rules of a problem, which can then be reapplied with minimal effort to new and unseen instances.

Some of the methods that have been presented in the introduction of this chapter already explore this idea of generalization across different instances of the same problem [8, 21, 41], but they all require either user-provided variable features/types or rely on simple, canonical problem structures.

If we hypothesize that we can find invariant rules from instance-specific attributes using TAcq, an extension of the framework could be designed to learn a set of rules from a single instance of a problem. When presented with a new instance, this extended framework would only need to learn the new attributes that map the instance parameters to the existing rules. This would allow the system to leverage previously learned rules, significantly reducing runtime and the number of examples needed for the new instance. The hypothesis that we can find invariant rules from instance-specific attributes work for many problem classes where the underlying constraint logic (the rules) remains constant, while the structural mapping (the attributes) changes with the instance parameters. For example, the rule "all cells in a row must be different" is fundamental to any Sudoku, but the definition of which cells constitute a row depends on the grid size.

This direction presents several open questions. First, we need to address rule transferability: how can we determine if a set of rules learned from a source instance is applicable to a target instance? This is non-trivial, especially for problems like Exam Timetabling, where the underlying constraint language itself can change based on instance parameters (e.g., number of available days and slots). Second, efficient attribute adaptation remains a challenge. While learning the new attributes from scratch is a viable starting point, more sophisticated techniques could be explored. Could we learn a transformation that maps the attributes from the source instance to the target instance? This would be particularly powerful for problems with regular, predictable structural changes.

Finally, we should consider how to design a system that learns directly a parameterized model that captures the rules and the structure without being tied to a specific instance. This would enable the generation of a constraint network for any instance of a problem class without requiring new examples, effectively creating a "universal template" for that problem class.

5.6.2 Better Interpretability

The analysis of the learned attributes for the Nurse Rostering and Sudoku problems confirmed that our method can discover compact and structured representations. This ability to learn compact models not only improves generalization but also yields models that tend to be more interpretable than a simple enumeration of constraints, as they consist of few attributes and rules that could be easily understood by a human expert. The small number of attributes and rules makes it easy for a human expert to inspect and understand the learned model.

However, as illustrated in the Section 5.5.4, the learned attributes may not always correspond to the most natural representation of the problem. For instance, in the Figure 5.5, the attributes learned for the Sudoku problem do not correspond to the most natural representation of the problem, as they do not capture the rows, columns, and boxes of the Sudoku grid in a straightforward way. Such attributes can be harder for a human to validate or interpret.

A promising direction would be to extract recurring structural patterns, common variable features and indexing schemes from a large corpus of constraint networks. This knowledge could then be incorporated into our template learning framework as

structural priors, guiding the attribute discovery process towards more interpretable representations when possible. However, the systematic collection and analysis of such models is a significant challenge. Currently, there is no comprehensive database that catalogs the common structural patterns across a sufficiently large number of problem classes. While repositories like CSPLib [1] provide problem descriptions, the number of problems listed is relatively small (96 problems in September 2025), and they lack the systematic metadata needed to automatically identify common variable indexing patterns or structural features across problems. The development of such a system would require significant effort in data collection, and to identify and transform these patterns into usable priors for the learning algorithm.

5.7 Conclusion

This chapter introduced a novel framework for passive constraint acquisition that focuses on learning compact and structured representations of constraint networks, which we call templates. Our work was motivated by a key limitation of the method Language-Free Acq presented in the previous chapter, which treated constraint scopes as a flat, unstructured set, making them prone to overfitting and requiring large amounts of examples in the training set. We hypothesized that by explicitly learning a compact representation, we could achieve better generalization.

Our primary contribution is a two-step acquisition pipeline, TAcq. It begins by using a baseline method, LFA, to generate an initial, potentially overly specific, constraint network. The core of our contribution is the second step: an algorithm that refines this network into a compact template. This is achieved by iteratively learning attributes that capture structural properties of the problem variables and rules that generate constraints based on these attributes. This entire learning process is formulated as a sequence of constraint optimization problems, allowing us to search effectively within the vast space of possible templates.

An experimental evaluation across a diverse set of benchmark problems demonstrates the significant benefits of this approach. Compared to the baseline LFA method, TAcq consistently achieves higher accuracy on unseen data with substantially fewer training examples. For structured problems like Sudoku, Exam Timetabling, and Nurse Rostering, our method successfully uncovered the suitable

structure (such as rows, columns, semesters, or days and shifts) without any prior knowledge.

Chapter 6

Perspectives on CNNs as Oracles

This chapter offers a prospective look into using Convolutional Neural Networks (CNNs) as oracles for constraint acquisition, exploring a research direction rather than presenting a complete method. Our goal is to have insights on how we can reconstruct a constraint network from the behavior of a trained CNN.

We argue that the architectural principles of CNNs (locality and weight sharing) are an interesting match for constraint acquisition methods, which typically seek to discover local rules that can be applied across multiple scopes in a problem.

Applying constraint acquisition methods in this context is challenging. However, we propose that the unique properties of using CNNs as oracles can be harnessed to guide the acquisition process. As an illustration of this direction, we suggest employing explainability techniques like Grad-CAM to guide the search for constraint scopes.

Contents

6.1 Introduction

Constraint acquisition has traditionally focused on learning constraint networks from a human expert who acts as an oracle, providing classifications for examples or answering queries about the target problem. While effective, this process is often constrained by the availability, cost, and speed of the human expert. This chapter studies the case where the oracle is a trained Convolutional Neural Network (CNN), which can classify a large number of examples quickly and efficiently. Shifting from human to machine oracles presents both opportunities and challenges for constraint acquisition.

We develop the idea of using constraint acquisition methods with a CNN oracle. Given a trained CNN, our goal is to reconstruct a concept-equivalent or closely-aligned constraint network. If successful, the extracted network serves as a more interpretable, symbolic model. A domain expert can then analyze this model to validate whether the CNN has genuinely learned the intended target concept or has instead converged on spurious correlations or overly simplistic patterns present in its training data.

Closely Related Work. Many approaches have focused on extracting symbolic or rule-based representations from neural networks largely under the name of *rule extraction* [2]. We distinguish two main families of methods. The first family, "pedagogical" methods (or black-box), treats the network as a black-box oracle, while the second, "decompositional" methods, analyzes the network internal structure (architecture, weights, etc.). A classic example of a pedagogical method is the TREPTAN algorithm, which queries a trained network to build a decision tree that mimics its classifications [14]. For CNNs, several methods leverage the network architectural properties. For instance, the ERIC framework approximates a CNN behavior with propositional logic rules by associating each learned convolutional filter with a binary predicate representing a learned concept [40]. The target representation is typically of the form of "IF-THEN" rules, decision trees, or logical formulas (e.g., DNF, Answer Set Programs). This chapter differs from these approaches because the target representation is a constraint network that can, for example, then be used to help generate new examples with a constraint solver.

A distinct and more direct approach to learning constraints with neural networks is presented in [46], which employs an Equation Learner, a specialized neural network architecture where activation functions are replaced by primitive mathematical operations. By coupling this network with a tailored loss function designed to find decision boundaries, their method trains the network to directly output a constraint (linear in the paper). This approach differs fundamentally from our proposal. In their work, the

neural network itself serves as the constraint acquisition algorithm, whereas we use a separate constraint acquisition method to extract constraints from a pre-trained CNN.

This chapter will first introduce some background on convolutional neural networks (CNNs) and their architectural principles in Section 6.2. We will then introduce some insight into how CNNs can be used as oracles in constraint acquisition in Section 6.3. In Section 6.4, we propose to use insights from explainability techniques like Grad-CAM to guide the acquisition process. Finally, we will conclude in Section 6.5.

6.2 Convolutional Neural Networks

A *convolutional neural network* (CNN) is a specialized class of feedforward artificial neural networks designed to process data with a known grid-like topology, such as images. They are particularly effective for tasks like image classification or object/pattern detection. CNNs are characterized by their ability to automatically learn spatial hierarchies of features from the input data, making them well-suited for visual recognition tasks.

To understand how a CNN processes information, we can examine its fundamental building block: the artificial *neuron*, as illustrated in Figure 6.1. Each neuron functions as a simple computational unit within a layer. It receives a set of inputs from the preceding layer and performs a two-step operation. First, it calculates a weighted sum of its inputs, where each input is multiplied by a corresponding *weight* that determines its influence. A *bias* term is then added to this sum, allowing the neuron to adjust its output independently. Second, this result is passed through a non-linear *activation function*, which transforms the value into the final output of the neuron. This entire operation is captured by the formula:

$$y = \sigma(W \cdot X + b) = \sigma(\sum_{i=1}^{n} w_i x_i + b)$$

where:

- $X = (x_1, x_2, \ldots)$ is the vector of inputs (possibly the outputs of the previous layer);

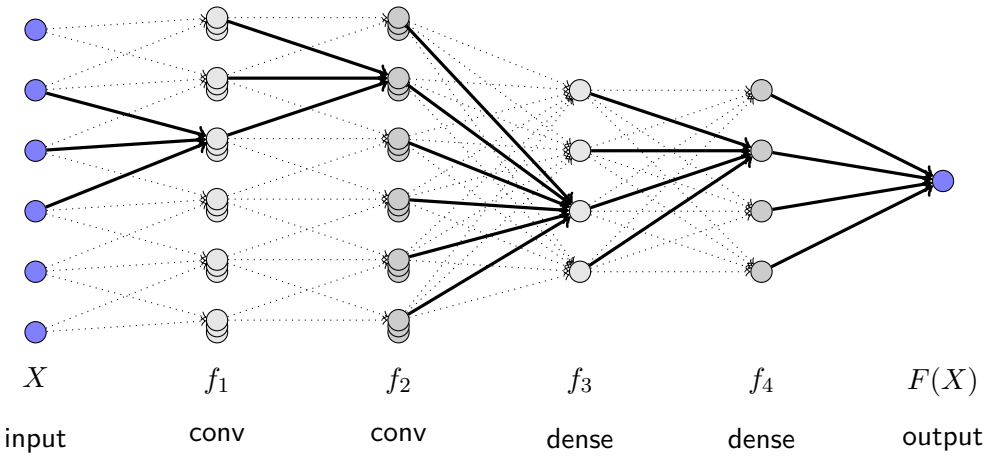


Figure 6.1: Illustration of a five-layer CNN architecture for a 1D input. The circles represent neurons. The input X is processed by two convolutional layers (f_1, f_2) characterized by local receptive fields, followed by two fully connected layers (f_3, f_4) to produce the final output $F(X)$. Dotted arrows show all connections, while bold arrows highlight the input for one example neuron in each subsequent layer. The bold arrows illustrate the locality of the convolutional layers, where each neuron only connects to a small batch of the neurons in the previous layer.

- $W = (w_1, w_2, \ldots)$ is a vector of weights, with each weight w_i modulating the importance or influence of the corresponding input feature x_i;
- b is the bias term, a learnable constant that shifts the activation function;
- σ is a non-linear activation function, such as the Rectified Linear Unit (ReLU) [17], which determines the final output of the neuron.

A CNN is a type of feedforward neural network, meaning that the information "flows" in one direction over the neurons, from the input layer through the hidden layers to the output layer, without any cycles or loops. This is shown in Figure 6.1 by the arrows connecting the neurons. The arrows only go from left to right, indicating that the output of one layer is the input to the next layer, and there are no connections that loop back to previous layers. This is in contrast to recurrent neural networks, which have connections that can loop back on themselves.

Let a CNN be represented by a function $F : \mathcal{X} \to \mathcal{Y}$, which maps an input $X \in \mathcal{X}$ to an output $Y \in \mathcal{Y}$. For simplicity, throughout this chapter, we will assume that the output is a single value, i.e. $Y \in \mathbb{R}$, which is often the case for binary classification tasks which are the focus of this chapter. By denoting f_l as the function implemented

by the l-th layer, we can express the overall function F as a composition of these layer functions:

$$F(X) = (f_L \circ f_{L-1} \circ \cdots \circ f_1)(X)$$

We briefly present the main layer types of a CNN, which are the convolutional layer, pooling layer, and fully connected layer. We omit information not essential for the subsequent discussion. For a more detailed introduction to CNNs, we refer the reader to one of the seminal works [24] and to the comprehensive textbook on deep learning [18].

Convolutional Layer The convolutional layer is the core building block of a CNN. Each neuron multiplies the input values by a vector of weights (called a *filter* or kernel) and sums up the results to produce a single value. There are typically multiple filters in a convolutional layer, each filter produces one output, and these outputs are then stacked together to form the output of the layer. This principle is illustrated in Figure 6.1 by the overlapping circles in the first two layers. This process helps in detecting local patterns, such as edges or textures in an image. There are two key characteristics of convolutional layers that are important for this chapter:

- **Locality:** Unlike fully connected layers, where each neuron is connected to all neurons in the previous layer (i.e. the input X of each layer is a vector containing all the outputs of neurons from the previous layer), neurons in a convolutional layer will only use a small subset of the input, which is called the *receptive field*. This means that each neuron in a convolutional layer is only connected to a small region of the input data. This is illustrated in Figure 6.1 by the arrows, which show the input for one example neuron in each subsequent layer. The receptive field is typically a small patch of the input data. Depending on the dimension of the input, this region can be a 1D (neurons connected to a small segment of the input sequence), 2D (neurons connected to a small patch of the input grid, e.g. a 3×3 patch of pixels in the grid), etc.
- **Weight Sharing:** The same weights (W) are used across all neurons for the same filter in the convolutional layer. This means that the same filter is applied to different parts of the input, allowing the network to learn features that are invariant to their position in the input data.

Throughout this chapter, we call *feature map* the output produced by the last convolutional layer. Because convolution preserves spatial arrangement, each location of the feature map corresponds to a local region in the original input (with potentially a different size depending on the filter).

Pooling Layer Pooling layers are used to reduce the dimensions (in 2D, the width and height) of an input, which helps to reduce the computational load and control overfitting. The most common type of pooling is max pooling where the maximum value from a small local patch of the input is taken.

Fully Connected Layer A fully connected (or dense) layer is a basic type of layer in a feedforward neural network (not specifically CNNs) where every neuron in the layer is connected to every neuron in the previous layer. This is in contrast to convolutional layers, where connections are local. There is no weight sharing in fully connected layers, meaning each neuron has its own vector of weights. Fully connected layers are often used to represent complex relationships in the data, and we assume that they are used after all convolutional and pooling layers to combine the feature maps extracted by the convolutional and pooling layers into a final output.

CNNs are fundamentally designed for training on data rather than explicitly modeling rules. The training process involves optimizing the network parameters (weights and biases) to minimize a loss function that measures the difference between the network output and a training set. This data-driven learning approach means that the knowledge acquired by the network is distributed across millions of parameters in a way that is not directly interpretable by humans. Consequently, CNNs are often referred to as black boxes in the context of machine learning because, while they can make accurate predictions, the reasoning behind their decisions often remains difficult to interpret by humans.

This optimization is achieved through a process called backpropagation, which efficiently computes the gradient of the loss function with respect to every parameter in the network. The gradient essentially measures the influence of each parameter (weights and biases) on the final error, guiding the learning process. By calculating the gradient of the final output with respect to the output of internal neurons, one can determine which neurons were most influential in the network decision for a

specific input. This concept will be crucial in Section 6.4 when we discuss using Grad-CAM.

6.3 CNNs as Oracles

The constraint acquisition framework is flexible concerning the nature of the oracle. While often assumed to be a human expert, the oracle can be any system capable of providing a training set or answering queries about the problem. A trained machine learning model, such as a neural network, can serve as an efficient and powerful oracle. Unlike human experts, who may be limited in the number of examples they can classify or the speed at which they can provide feedback, a trained neural network can classify thousands of examples per second. This rapid classification capability allows for the generation of large training sets, which can be used for constraint acquisition.

CNNs are a domain of interest for constraint acquisition due to their widespread success. Initially renowned for computer vision applications, CNNs are now successfully applied in many other domains where data has an underlying grid structure or exhibits local relations. This includes applications in natural language processing and time-series analysis. CNNs have even been utilized as heuristic approaches for solving combinatorial problems [25]. The broad and growing relevance of CNNs makes them an interesting class of oracles to study.

We focus specifically on CNNs (and not any other type of neural network) because two architectural characteristics make them particularly relevant to constraint network representation, and therefore especially interesting to study as oracles in constraint acquisition.

The first is *locality*, which refers to the fact that filters operate on small local (i.e. spatially or temporally contiguous) parts of the input. This means that the features map during the initial convolutional layers are based on local patterns. When the detection of a conjunction of local features determines the global decision, without the need for complex global relations (using the fully connected layers), the underlying concept can be described using a constraint network with potentially small scopes. This is also the case for disjunctions of local patterns, which we discuss later in this section.

The second important characteristic is *weight sharing*. A single filter is applied across the entire input, meaning that the same local pattern may trigger a filter response regardless of spatial position. This suggests that, if it is possible to learn a constraint network from the CNN, this network will likely be over a constraint language with few relations applied over many scopes.

We propose a research direction that leverages the strengths of CNNs as oracles in constraint acquisition. The goal is to reconstruct a constraint network from the behavior of a trained CNN. This approach is motivated by the desire to understand whether the concept learned by the CNN can be represented as a constraint network, and if so, to extract a symbolic representation that can be analyzed by domain experts. Given a CNN that represents a concept $c : D^X \to \{0, 1\}$, the goal is to reconstruct a constraint network N that represents the concept $f : D^X \to \{0, 1\}$ whose solution space coincides with (or closely approximates) the CNN positive region. To be consistent with classic learning frameworks, we consider a distribution $\mathcal{D}$ over the input space D^X. The general goal should be to minimize the expected loss under this distribution:

$$\mathcal{L}(c, f) = \mathbb{E}_{x \sim \mathcal{D}} \left[|c(x) - f(x)| \right]$$

To achieve this, we treat the CNN as an oracle that can provide examples to the concept c by classifying assignments. Without more information provided by the CNN oracle, this method would be considered as a black-box approach, or a pedagogical approach of rule extraction as specified in the introduction.

In practice, we may be interested in extracting a constraint network whose non-solution space coincides with the CNN positive classifications. This alternative perspective can be valuable when the CNN is trained to detect disjunctions of local patterns. In this case, the CNN positive classifications indicate the detection of the pattern, and therefore can be naturally interpreted as a violation of a constraint. The alternative goal should be to minimize:

$$\mathcal{L}(c, f) = \mathbb{E}_{x \sim \mathcal{D}} \left[||1 - c(x)| - f(x)| \right]$$

Many constraint acquisition methods, in particular LFA presented in Chapter 4 and therefore Tacq with LFA as a baseline in Chapter 5, assume that the oracle

provides perfectly accurate classifications for both positive and negative examples. That is, they assume that the oracle can perfectly distinguish between positive and negative examples of the target concept without any errors. A CNN is trained to approximate a target concept based on its training set, but its learned representation may differ from the initial concept, particularly when evaluating assignments that deviate from the original training distribution. When the CNN encounters such inputs that are significantly different from its training examples, its predictions become unreliable.

There are constraint acquisition methods that are robust to errors, such as those presented in [34, 35]. These methods can handle some level of noise in the training set. However, they rely on additional prior knowledge about the target concept to be represented, such as a constraint language expressive enough to represent the target concept but not too complex to avoid overfitting. In our scenario with a direct interaction with a CNN, we will assume that this prior knowledge is not available.

The model used in LFA (with k and r fixed) ensures consistency with the training set as a set of inviolable hard clauses: any yield constraint network must classify all examples in the training set in the same way as the CNN. The natural modification to handle noise would be to associate an "error cost" with each example with a soft clause, allowing the model to misclassify some examples by paying that cost. However, this leads to a conflict in the general objective function of the LFA method. The method is based on the search for the simplest language (minimizing k and r) to avoid overfitting. This creates a new, two-fold objective: finding a language and a network that minimizes two completely different cost terms, one for the language complexity and another for the errors. There is no clear way to balance these objectives.

6.4 Use of Explainability Techniques

CNNs offer significant advantages over traditional oracles when not treated purely as black boxes. CNNs can provide additional information that can be leveraged to improve the constraint acquisition process. CNNs learn weights in each neuron that encode the concept learned by the network. These weights, along with the architecture of the CNN, can be used to help the constraint acquisition process.

Explainability techniques for CNNs have matured significantly, offering methods to extract interpretable visualizations of the complex decision-making processes of these networks. A common technique focuses on generating visual explanations that highlight the regions of the input that are most influential in the network prediction [26, 36, 37, 39, 48, 49].

Figure 6.2: Example of a heatmap generated by Grad-CAM in the original paper [37] for a CNN trained to detect images with dogs. The input is an image of a dog, and the heatmap highlights the regions of the image that are most influential for the CNN classification decision. The original image is shown on the left, and the heatmap is shown on the right. The heatmap is overlaid on the original image, where warmer colors (e.g., red or yellow) indicate higher influence and transparency indicates lower influence.

These techniques generate visual explanations, often in the form of *heatmaps*. A heatmap is a spatial representation of the input where each variable is assigned a value. The heatmap highlights the variables of an input that are the most influential for the CNN classification decision. A heatmap is often represented using colors overlapping the original input, where warmer colors (e.g., red or yellow) indicate higher influence and cooler colors (e.g., blue or green) or transparency indicate lower influence. This visualization helps to understand which parts of the input the CNN focuses on when making a classification decision.

In this section, we focus on methods specifically designed for CNNs. The Class Activation Mapping (CAM) [49] is one of the pioneering techniques in this area, producing a heatmap that indicates the specific regions in the input that a CNN uses to make a classification decision. However, CAM is only applicable to CNNs that use only convolutional and pooling layers, without any fully-connected layers. (The method can work with a last fully-connected layer, but it modifies the CNN, such that the final fully connected layer is replaced with a global average pooling

layer.) Given an input X and a CNN, CAM computes the feature maps of the final convolutional layer with respect to X and uses the weights of the final pooling layer to generate a heatmap that highlights the regions of the input that contribute most significantly to the classification decision.

While the CAM technique required a specific network architecture without fully-connected layers, the more general Grad-CAM uses output gradients to compute feature importance, making it applicable to any CNN architecture [37]. Given an input X, Grad-CAM computes the feature maps of the final convolutional layer with respect to X as the CAM method and calculates the gradients of the output score with respect to these feature maps. These gradients are then used to weight the importance of each filter and region in the feature map, producing a heatmap that highlights the regions of X most influential for the classification decision.

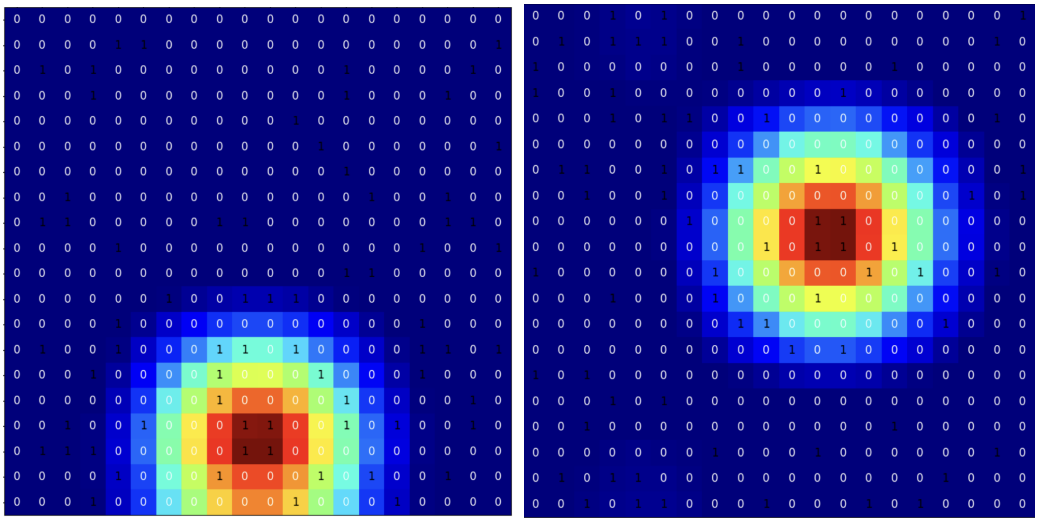


Figure 6.3: Heatmaps generated with the Grad-CAM method for two negative examples of a CNN trained on a simple image classification task consisting of detecting the presence of a 2×2 black square in a 20×20 black and white image. The overlaid white (1) and black (0) numbers indicate the original pixel values of the input image. The color of the heatmap indicates the influence of each variable on the CNN classification decision, with warmer colors (e.g., red or yellow) indicating higher influence and cooler colors (e.g., blue or green) indicating lower influence.

The Figure 6.3 shows two heatmaps for two negative examples generated by Grad-CAM for a dummy CNN trained on a simple image classification task. The input is a 20×20 black and white image, i.e. a grid of pixels that can take the values

0 (white) or 1 (black). The CNN has two convolutional layers followed by a fully connected layer. The classification task is to accept only images that do not contain a 2×2 black square. The CNN is trained with a training set containing 2000 images with 10% of black pixels, with half containing no square and half containing one 2×2 square.[1]

Given the two inputs containing a small black square, the Grad-CAM heatmap highlights the regions of the input that contribute most significantly to the classification decision, which in this case is the black square. The heatmap shows that the CNN has learned to focus on the presence of the black square in the image, which is consistent with the classification task.

We can leverage the heatmap generated by Grad-CAM to extract a constraint network that represents the concept learned by the CNN. The idea is to use the heatmap to identify the regions of the input that are most important for the classification decision for each negative example, and then use these regions to hint at the scopes to be learned during the constraint acquisition. First, let us define a constraint acquisition problem where the scopes of the constraints are hinted for each negative example.

Definition 16 (Constraint Acquisition with Hinted Scopes)**.** *Given a training set $E = E^+ \cup E^-$, where E^+ is the set of positive examples and $E^- = \{x_1^-, x_2^-, \dots x_n^-\}$ is the set of negative examples with, for each negative example $x_i \in E^-$, a set of variables denoted $X_i' \subseteq X$. The goal is to learn a constraint network N that is consistent with the training set E such that there exists for each negative example x_i^- a constraint (R, S) in N that rejects $\alpha(x_i^-)$ and with $S \subseteq X_i'$.*

We can use the heatmap generated by Grad-CAM to identify the variables in the input that are most important for the classification decision. For instance, for each negative example x_i^-, we can define the scopes X_i' as the set of pixels in the heatmap that have a value above a certain threshold, indicating that they are important for the classification decision. This use of internal network information to guide the learning process means our method is not strictly pedagogical (black-box), but rather a decompositional approach that leverages the representation of the CNN to inform the constraint acquisition process.

[1] Details and all the code to reproduce the experiment are available at https://gite.lirmm.fr/coconut/cnn-gradcam-visualization

It is important to note that, if a constraint network that can represent the concept using scopes given by the heatmap information does not exist, it does not necessarily imply that the underlying concept cannot be represented as a constraint network. Rather, it may indicate that the CNN has learned an alternative representation of the same concept. The CNN may rely on its fully connected layers to encode high arity relations between features extracted by the convolutional layers, and such global dependencies cannot be captured by a conjunction of constraints over the scopes defined by the heatmap.

6.5 Conclusion

This chapter has explored the potential of using Convolutional Neural Networks (CNNs) as oracles within the framework of constraint acquisition. We have argued that this is a promising research direction because their fundamental architectural principles (locality and weight sharing) resonate with the structure of many constraint networks.

We introduced the core idea of treating a trained CNN as an oracle to reconstruct a concept-equivalent or closely-approximating constraint network. A key challenge identified is that the predictions of a CNN can be unreliable, especially for inputs that differ from its training distribution.

On the other hand, the CNN can provide additional information to guide the acquisition process. We propose to use a "grey-box" approach that leverages explainability techniques like Grad-CAM. By analyzing heatmaps of influential input variables, we can generate "hints" for the scopes of the underlying constraints. A primary step is the concrete implementation and empirical evaluation of a constraint acquisition algorithm guided by Grad-CAM heatmaps. Such research would need to investigate various strategies for translating heatmap values into candidate scopes and assess the effectiveness of this guidance on different problems.

Chapter 7
Conclusion

This thesis developed and evaluated new methods in passive constraint acquisition designed to make the learning process significantly more automated. This chapter summarizes our key contributions, and we conclude by outlining promising directions for future research.

7.1 Summary of Contributions

First, we introduced LFA (Language-Free Acquisition) in Chapter 4, a novel method that discards the requirement for a predefined constraint language. The method learns both the constraints and a suitable language to express them, thus removing the need to provide a set of candidate relations. This is achieved by framing the problem as an optimization task with a bias towards simplicity by minimizing the arity and number of relations. We proved the underlying decision problem to be NP-complete and proposed a practical algorithm based on a Weighted Partial Max-Sat model. Experimental results demonstrated the viability of this approach across a range of benchmark problems, representing a significant step towards fully automated constraint acquisition.

Second, we identified and addressed a key limitation of LFA: this method learns a simple list of scopes. This can lead to the need for a large number of examples to eliminate few spurious constraints. Chapter 5 tackled this by proposing to learn a novel, more compact and structured, representation of constraint networks called a template. This template consists of attributes of variables and rules that generate constraints based on these attributes. We present the method TAcq, that refines

an initial network generated by LFA into a template, effectively capturing patterns such as "apply this constraint to all variables in the same row". Our experiments showed that TAcq reduces the number of examples needed to achieve high accuracy on structured problems and produces more interpretable models.

Finally, in Chapter 6, we returned to the question of the oracle, shifting the focus from the classic human experts to machine learning models. We proposed the direction of using explainability techniques to exploit new types of data that can be provided by a neural network oracle, such as a CNN.

7.2 Perspectives

The contributions of this thesis open several avenues for future research. We outline some of the most promising directions below.

Simplicity of the Relations

The notion of simplicity of a language is central to the contribution of Chapter 4, guiding the design of methods that learn the constraint language without requiring extensive prior knowledge. While the current work defines simplicity in terms of language size and arity, this notion could be extended to incorporate the inherent complexity of the learned relations themselves. Future work could focus on developing a more nuanced cost function that favors languages constructed from a basis of well-understood, common relations, thereby promoting the discovery of more intuitive and generalizable constraint languages.

Interpretability of the Scopes

While the template representation and learning method presented in Chapter 5 offers a more structured view than a flat list of constraints, the learned attributes may not always align with the most intuitive understanding of the problem by a human expert. Future research could explore methods to bias the learning process towards more canonical attributes, potentially by leveraging statistical analyses of established constraint programming models.

Robustness to Noise

The methods presented in this manuscript assume a completely reliable oracle. However, in some scenarios, the provided classifications may contain noise. Developing more robust constraint acquisition algorithms that can effectively learn from noisy and potentially unreliable data is a critical area for future research. This will be essential for the practical application of these methods in a wider range of domains.

New Class of Oracles

The exploration of learning from CNNs in Chapter 6 is a promising direction, but it presents challenges. The proposed direction of using explainability techniques to guide constraint acquisition is a first step. This will require investigating various strategies for translating the outputs of explainability methods into meaningful guidance for the constraint acquisition process and assessing its effectiveness across a diverse set of problems and model architectures.

Acknowledgments

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). Preliminary experiments were conducted with the support of the ISDM-MESO platform at the University of Montpellier.

I acknowledge the use of multiple AI tools based on large language models mainly Mistral (*mistral-medium-2505*), Anthropic (*Claude Sonnet 4*) and Google (*gemini-2.5-pro*) to identify improvements in the writing style. Example of use: "Please review the following text and suggest improvements for clarity, coherence, and overall writing style: [insert paragraph here]." None of the AI-generated content was directly included in the thesis without thorough verification and validation by the author. The author takes full responsibility for the final content of the thesis.

Current life-cycle analyses of Large Language Model (LLM) inference suggest a carbon footprint of up to 1.14 grams of CO2e (carbon dioxide equivalent) per complex query[1]. For this work, I estimate 500 queries (as a high estimate) with an average of 0.8g per query, reflecting a mixed use of dense and compact models. Based on this usage pattern, the total carbon emissions associated with LLM assistance for this manuscript are estimated at less than 0.4 kg CO2e. This is equivalent to the carbon footprint of driving a standard gas car for approximately 2 kilometers[2]. While this environmental impact is relatively low compared to common daily activities such as transportation and food consumption, it constitutes a new and additional form of consumption that must be assessed.

[1] Based on the study "*Our contribution to a global environmental standard for AI*" by Mistral AI over Mistral Large 2 according to the Frugal AI methodology developed by AFNOR (French Standardization Association). Other studies, as reported in the technical paper "*Measuring the environmental impact of delivering AI at Google Scale*" by Google, estimate that the median Gemini Apps text prompt emits only 0.03 gCO2e. However, the methodology does not detail emissions depending on the model used or the complexity of the queries.

[2] ADEME, French Environment and Energy Management Agency

Bibliography

[1] CSPLib: A problem library for constraints. http://www.csplib.org, 1999.

[2] Robert Andrews, Joachim Diederich, and Alan B. Tickle. Survey and critique of techniques for extracting rules from trained artificial neural networks. *Knowl. Based Syst.*, 8(6):373–389, 1995.

[3] Ekaterina Arafailova, Nicolas Beldiceanu, Rémi Douence, Mats Carlsson, Pierre Flener, María Andreína Francisco Rodríguez, Justin Pearson, and Helmut Simonis. Global constraint catalog, volume ii, time-series constraints. *CoRR*, abs/1609.08925, 2016.

[4] Nicolas Beldiceanu, Mats Carlsson, Sophie Demassey, and Thierry Petit. Global constraint catalogue: Past, present and future. *Constraints An Int. J.*, 12(1):21–62, 2007.

[5] Nicolas Beldiceanu and Helmut Simonis. A model seeker: Extracting global constraint models from positive examples. In Michela 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.

[6] Christian Bessiere, Clément Carbonnel, and Areski 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.

[7] Christian Bessiere, Clément Carbonnel, and Areski Himeur. Apprendre un csp sans connaître son langage. *Journées Francophones de Programmation par Contraintes*, 2024.

[8] Christian Bessiere, Remi Coletta, Abderrazak Daoudi, Nadjib Lazaar, Younes Mechqrane, and El-Houssine Bouyakhf. Boosting constraint acquisition via generalization queries. In Torsten Schaub, Gerhard Friedrich, and Barry O'Sullivan, editors, *ECAI 2014 - 21st European Conference on Artificial Intelligence, 18-22 August 2014, Prague, Czech Republic - Including Prestigious Applications of Intelligent Systems (PAIS 2014)*, volume 263 of *Frontiers in Artificial Intelligence and Applications*, pages 99–104. IOS Press, 2014.

[9] Christian Bessiere, Remi Coletta, Emmanuel Hebrard, George Katsirelos, Nadjib Lazaar, Nina Narodytska, Claude-Guy Quimper, and Toby Walsh. Constraint acquisition via partial queries. In Francesca Rossi, editor, *IJCAI 2013, Proceedings of the 23rd International Joint Conference on Artificial Intelligence, Beijing, China, August 3-9, 2013*, pages 475–481. IJCAI/AAAI, 2013.

[10] Christian Bessiere, Remi Coletta, Frédéric Koriche, and Barry O'Sullivan. A sat-based version space algorithm for acquiring constraint satisfaction problems. In João Gama, Rui Camacho, Pavel Brazdil, Alípio Jorge, and Luís 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.

[11] Christian Bessiere, Emmanuel Hebrard, Brahim Hnich, Zeynep Kiziltan, and Toby Walsh. Range and roots: Two common patterns for specifying and propagating counting and occurrence constraints. *Artif. Intell.*, 173(11):1054–1078, 2009.

[12] Christian Bessiere, Frédéric Koriche, Nadjib Lazaar, and Barry O'Sullivan. Constraint acquisition. *Artif. Intell.*, 244:315–342, 2017.

[13] Stephen A. Cook. The complexity of theorem-proving procedures. In Michael A. Harrison, Ranan B. Banerji, and Jeffrey D. Ullman, editors, *Proceedings of the 3rd Annual ACM Symposium on Theory of Computing, May 3-5, 1971, Shaker Heights, Ohio, USA*, pages 151–158. ACM, 1971.

[14] Mark W. Craven and Jude W. Shavlik. Extracting tree-structured representations of trained networks. In David S. Touretzky, Michael Mozer, and Michael E. Hasselmo, editors, *Advances in Neural Information Processing Systems 8, NIPS, Denver, CO, USA, November 27-30, 1995*, pages 24–30. MIT Press, 1995.

[15] Abderrazak Daoudi, Nadjib Lazaar, Younes Mechqrane, Christian Bessiere, and El-Houssine Bouyakhf. Detecting types of variables for generalization in constraint acquisition. In *27th IEEE International Conference on Tools with Artificial Intelligence, ICTAI 2015, Vietri sul Mare, Italy, November 9-11, 2015*, pages 413–420. IEEE Computer Society, 2015.

[16] Martin Davis, George Logemann, and Donald Loveland. A machine program for theorem-proving. *Communications of the ACM*, 5(7):394–397, 1962.

[17] Xavier Glorot, Antoine Bordes, and Yoshua Bengio. Deep sparse rectifier neural networks. In Geoffrey J. Gordon, David B. Dunson, and Miroslav Dudík, editors, *Proceedings of the Fourteenth International Conference on Artificial Intelligence and Statistics, AISTATS 2011, Fort Lauderdale, USA, April 11-13, 2011*, volume 15 of *JMLR Proceedings*, pages 315–323. JMLR.org, 2011.

[18] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. MIT Press, 2016. http://www.deeplearningbook.org.

[19] Tias Guns. Increasing modeling language convenience with a universal n-dimensional array, cppy as python-embedded example. In *Proceedings of the 18th workshop on Constraint Modeling and Reformulation at CP (Modref 2019)*, volume 19, 2019.

[20] Michael J. Kearns and Umesh V. Vazirani. *An Introduction to Computational Learning Theory*. MIT Press, 1994.

[21] Mohit Kumar, Samuel Kolb, and Tias Guns. Learning constraint programming models from data using generate-and-aggregate. In Christine 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.

[22] Mohit Kumar, Stefano Teso, and Luc De Raedt. Acquiring integer programs from data. In Sarit 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.

[23] Arnaud Lallouet, Matthieu Lopez, Lionel Martin, and Christel 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.

[24] Yann LeCun, Bernhard E. Boser, John S. Denker, Donnie Henderson, Richard E. Howard, Wayne E. Hubbard, and Lawrence D. Jackel. Handwritten digit recognition with a back-propagation network. In David S. Touretzky, editor, *Advances in Neural Information Processing Systems 2, [NIPS Conference, Denver, Colorado, USA, November 27-30, 1989]*, pages 396–404. Morgan Kaufmann, 1989.

[25] Michele Lombardi and Michela Milano. Boosting combinatorial problem modeling with machine learning. In Jérôme Lang, editor, *Proceedings of the Twenty-Seventh International Joint Conference on Artificial Intelligence, IJCAI 2018, July 13-19, 2018, Stockholm, Sweden*, pages 5472–5478. ijcai.org, 2018.

[26] Scott M. Lundberg and Su-In Lee. A unified approach to interpreting model predictions. In Isabelle Guyon, Ulrike von Luxburg, Samy Bengio, Hanna M. Wallach, Rob Fergus, S. V. N. Vishwanathan, and Roman Garnett, editors, *Advances in Neural Information Processing Systems 30: Annual Conference on Neural Information Processing Systems 2017, December 4-9, 2017, Long Beach, CA, USA*, pages 4765–4774, 2017.

[27] Grégoire Menguy, Sébastien Bardin, Nadjib Lazaar, and Arnaud Gotlieb. Automated program analysis: Revisiting precondition inference through constraint acquisition. In Luc De Raedt, editor, *Proceedings of the Thirty-First International Joint Conference on Artificial Intelligence, IJCAI 2022, Vienna, Austria, 23-29 July 2022*, pages 1873–1879. ijcai.org, 2022.

[28] Tom Michael Mitchell. *Version spaces: an approach to concept learning.* Stanford University, 1979.

[29] Tom Michael Mitchell. *Machine Learning.* McGraw Hill, 1997.

[30] Ugo Montanari. Networks of constraints: Fundamental properties and applications to picture processing. *Inf. Sci.*, 7:95–132, 1974.

[31] Mathias Paulin, Christian Bessiere, and Jean Sallantin. Automatic design of robot behaviors through constraint network acquisition. In *20th IEEE International Conference on Tools with Artificial Intelligence (ICTAI 2008),*

November 3-5, 2008, Dayton, Ohio, USA, Volume 1, pages 275–282. IEEE Computer Society, 2008.

[32] Laurent Perron and Vincent Furnon. Or-tools (v9.11), 2025.

[33] Marek Piotrów. Uwrmaxsat: Efficient solver for maxsat and pseudo-boolean problems. In *32nd IEEE International Conference on Tools with Artificial Intelligence, ICTAI 2020, Baltimore, MD, USA, November 9-11, 2020*, pages 132–136. IEEE, 2020.

[34] Steve Prestwich. Robust constraint acquisition by sequential analysis. In Giuseppe De Giacomo, Alejandro Catalá, Bistra Dilkina, Michela Milano, Senén Barro, Alberto Bugarín, and Jérôme Lang, editors, *ECAI 2020 - 24th European Conference on Artificial Intelligence, 29 August-8 September 2020, Santiago de Compostela, Spain, August 29 - September 8, 2020 - Including 10th Conference on Prestigious Applications of Artificial Intelligence (PAIS 2020)*, volume 325 of *Frontiers in Artificial Intelligence and Applications*, pages 355–362. IOS Press, 2020.

[35] Steven D. Prestwich, Eugene C. Freuder, Barry O'Sullivan, and David Browne. Classifier-based constraint acquisition. *Ann. Math. Artif. Intell.*, 89(7):655–674, 2021.

[36] Marco Túlio Ribeiro, Sameer Singh, and Carlos Guestrin. "why should I trust you?": Explaining the predictions of any classifier. In Balaji Krishnapuram, Mohak Shah, Alexander J. Smola, Charu C. Aggarwal, Dou Shen, and Rajeev Rastogi, editors, *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, San Francisco, CA, USA, August 13-17, 2016*, pages 1135–1144. ACM, 2016.

[37] Ramprasaath R. Selvaraju, Michael Cogswell, Abhishek Das, Ramakrishna Vedantam, Devi Parikh, and Dhruv Batra. Grad-cam: Visual explanations from deep networks via gradient-based localization. In *IEEE International Conference on Computer Vision, ICCV 2017, Venice, Italy, October 22-29, 2017*, pages 618–626. IEEE Computer Society, 2017.

[38] JP Marques Silva and Karem A Sakallah. Grasp-a new search algorithm for satisfiability. In *Proceedings of International Conference on Computer Aided Design*, pages 220–227. IEEE, 1996.

[39] Karen Simonyan, Andrea Vedaldi, and Andrew Zisserman. Deep inside convolutional networks: Visualising image classification models and saliency maps. In Yoshua Bengio and Yann LeCun, editors, *2nd International Conference on Learning Representations, ICLR 2014, Banff, AB, Canada, April 14-16, 2014, Workshop Track Proceedings*, 2014.

[40] Joe Townsend, Theodoros Kasioumis, and Hiroya Inakoshi. ERIC: extracting relations inferred from convolutions. In Hiroshi Ishikawa, Cheng-Lin Liu, Tomás Pajdla, and Jianbo Shi, editors, *Computer Vision - ACCV 2020 - 15th Asian Conference on Computer Vision, Kyoto, Japan, November 30 - December 4, 2020, Revised Selected Papers, Part III*, volume 12624 of *Lecture Notes in Computer Science*, pages 206–222. Springer, 2020.

[41] Dimos Tsouros, Senne Berden, Steven Prestwich, and Tias Guns. Generalizing constraint models in constraint acquisition. In Toby Walsh, Julie Shah, and Zico 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.

[42] Dimos Tsouros and Tias Guns. A cpmpy-based python library for constraint acquisition - pycona. In *Proc. AAAI 2025 Bridge on Constraint Programming and Machine Learning (CPML)*, 2025.

[43] Dimosthenis Tsouros, Senne Berden, and Tias Guns. Learning to learn in interactive constraint acquisition. *Proceedings of the AAAI Conference on Artificial Intelligence*, 38(8):8154–8162, Mar. 2024.

[44] Dimosthenis C. Tsouros and Kostas Stergiou. Efficient multiple constraint acquisition. *Constraints An Int. J.*, 25(3-4):180–225, 2020.

[45] Dimosthenis C. Tsouros and Kostas Stergiou. Learning max-csps via active constraint acquisition. In Laurent 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.

[46] Eduardo Vyhmeister, Rocio Paez, and Gabriel González-Castañé. Deep neural network for constraint acquisition through tailored loss function. In Leonardo Franco, Clélia de Mulatier, Maciej Paszynski, Valeria V. Krzhizhanovskaya, Jack J. Dongarra, and Peter M. A. Sloot, editors, *Computational Science - ICCS 2024 - 24th International Conference, Malaga, Spain, July 2-4, 2024, Proceedings, Part V*, volume 14836 of *Lecture Notes in Computer Science*, pages 43–57. Springer, 2024.

[47] Boris Wiegand, Dietrich Klakow, and Jilles Vreeken. What are the rules? discovering constraints from data. In Michael J. Wooldridge, Jennifer G. Dy, and Sriraam Natarajan, editors, *Thirty-Eighth AAAI Conference on Artificial Intelligence, AAAI 2024, Thirty-Sixth Conference on Innovative Applications of Artificial Intelligence, IAAI 2024, Fourteenth Symposium on Educational Advances in Artificial Intelligence, EAAI 2014, February 20-27, 2024, Vancouver, Canada*, pages 8182–8190. AAAI Press, 2024.

[48] Matthew D. Zeiler and Rob Fergus. Visualizing and understanding convolutional networks. In David J. Fleet, Tomás Pajdla, Bernt Schiele, and Tinne Tuytelaars, editors, *Computer Vision - ECCV 2014 - 13th European Conference, Zurich, Switzerland, September 6-12, 2014, Proceedings, Part I*, volume 8689 of *Lecture Notes in Computer Science*, pages 818–833. Springer, 2014.

[49] Bolei Zhou, Aditya Khosla, Àgata Lapedriza, Aude Oliva, and Antonio Torralba. Learning deep features for discriminative localization. In *2016 IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2016, Las Vegas, NV, USA, June 27-30, 2016*, pages 2921–2929. IEEE Computer Society, 2016.