Composing Biases by Using CP to Decompose Minimal Functional Dependencies for Acquiring Complex Formulae

Ramiz Gindullin, Nicolas Beldiceanu, Jovial Cheukam-Ngouonou, Rémi Douence, Claude-Guy Quimper

Composing Biases by Using CP to Decompose Minimal Functional Dependencies for Acquiring Complex Formulae

Ramiz Gindullin[1,2*], **Nicolas Beldiceanu**[1,2], **Jovial Cheukam-Ngouonou**[1,2,4†], **Rémi Douence**[1,2,3], **Claude-Guy Quimper**[4]

[1]IMT Atlantique, Nantes, France
[2]LS2N, Nantes, France
[3]INRIA, Nantes, France
[4]Université Laval, Quebec City, Canada
ramiz.gindullin@imt-atlantique.fr, nicolas.beldiceanu@imt-atlantique.fr, jovial.cheukam-ngouonou.1@ulaval.ca, Remi.Douence@imt-atlantique.fr, Claude-Guy.Quimper@ift.ulaval.ca

Abstract

Given a table with a minimal set of input columns that functionally determines an output column, we introduce a method that tries to gradually decompose the corresponding minimal functional dependency (mfd) to acquire a formula expressing the output column in terms of the input columns. A first key element of the method is to create sub-problems that are easier to solve than the original formula acquisition problem, either because it learns formulae with fewer inputs parameters, or as it focuses on formulae of a particular class, such as Boolean formulae; as a result, the acquired formulae can mix different learning biases such as polynomials, conditionals or Boolean expressions. A second key feature of the method is that it can be applied recursively to find formulae that combine polynomial, conditional or Boolean subterms in a nested manner. The method was tested on data for eight families of combinatorial objects; new conjectures were found that were previously unattainable. The method often creates conjectures that combine several formulae into one with a limited number of automatically found Boolean terms.

1 Introduction

While the problem of synthesising formulae from data (Alur et al. 2018) is central to many areas such as programming by example (e.g. finding formulae in spreadsheets (Gulwani 2011; Gulwani, Harris, and Singh 2012; Paramonov et al. 2017)), program verification (e.g. identifying loop invariants (Srivastava, Gulwani, and Foster 2013)), and conjecture generation (e.g. proposing bounds for combinatorial objects (Aouchiche et al. 2005; Larson and Cleemput 2016; Beldiceanu et al. 2022)), acquisition techniques are limited when the learning bias, i.e. "the set of assumptions that the learner uses to predict outputs of given inputs" (Wikimedia Commons 2003), is vast. In this paper, these assumptions correspond to the type of formulae we acquire.

Besides the recent work of S.-M. Udrescu *et al.* (Udrescu et al. 2020) which applies to continuous functions, most approaches for acquiring discrete functions rely on a grammar to define a domain-specific learning bias. They use a generate-and-test method to produce candidate formulae of increasing complexity. Several improvements were made to limit the combinatorial explosion of candidate formulae. They include the use of probabilistic grammar or statistical methods (Brence, Todorovski, and Džeroski 2021) to focus on more likely candidate formulae first, to generate partially instantiated formulae whose coefficients are determined by a CP or a MIP model (Ligeza et al. 2020; Beldiceanu et al. 2022), or to apply metaheuristics (Hansen and Caporossi 2000). However, their main weakness is twofold: first, they usually deal with formulae from a *restricted domain-specific learning bias*; second, they try to *directly acquire a formula* that mentions all the relevant input parameters at once.

Contribution We generally do not know how to effectively combine learning biases for acquiring formulae. A system that knows how to solve problems with learning bias (A) and another system that knows how to solve problems with learning bias (B) can be combined to handle not only problems with learning bias (A) or (B), but also can combine both learning biases in a nested manner. The problem is *how to find decomposition methods to facilitate the discovery and combination of multiple learning biases*.

The proposed method partly answers this question through the following observation. Although many formulae are complex, i.e. they involve various operators in different sub-terms of a same formula, some parameters only appear in a *few sub-terms*, and some sub-terms have a *very specific form*. We show that by analysing an mfd of a table, while relying on the input columns and the output column, it is possible to identify different sub-terms of a formula to be learnt and the operators that connect its sub-terms. We carry out this analysis by using Constraint Programming (CP) to solve certain sub-problems that allows us to decompose the formula we are looking for, into its sub-terms, without knowing yet the formula to be found. In Sect. 2, we give concrete examples of complex formulae obtained using our new decompositions, and describe them in Sect. 3. In Sect. 4, we evaluate the performance of our contribution.

*Funded by the EU ASSISTANT project no. 101000165.
†Funded by the ANR AI@IMT project and by Laval University.

2 Context, Motivation, Running Examples, and Decomposition Types

This section first describes the context behind the decomposition techniques introduced in the paper. Then, it gives typical formulae found using these techniques. Finally, it defines the types of decomposition introduced.

2.1 Context and Motivation

Paper (Beldiceanu et al. 2022) describes a CP system for finding conjectures about sharp bounds on characteristics of combinatorial objects. Their learning process is based on tables, each entry of which is a positive example, specifying the sharp lower (resp. upper) bound of a characteristic of a combinatorial object based on a combination of values for other characteristics. Therefore the acquisition process acquires equality (as they have sharp bounds), corresponding, in the end, to inequalities as the generated conjectures deal with lower (resp. upper) bounds. As all the entries in a table are noise-free, they acquire formulae that match all table entries. Their framework uses three biases:

(*i*) Formulae mentioning *one or two polynomials*, e.g. '$x^3 + y \cdot z$', '$\max(x \cdot y, 2 \cdot z^2)$'; in the case of two polynomials, these can be connected by a usual arithmetic operator.

(*ii*) *Simple conditional formulae* of the form $(cond\ ?\ x\ :\ y)$ denotes x if condition $cond$ holds, y otherwise; the conditional part, the then and else parts have a simple form: each part mentions at most 2 input parameters and one coefficient, e.g. '$(\lceil x/y \rceil = 2\ ?\ 2 \cdot y : x)$'.

(*iii*) *Boolean-arithmetic formulae* consisting of arithmetic conditions using a commutative operator such as '$\wedge$' or '$+$', e.g. '$[x \geq 2] + [(y - x) \geq 2]$', where '[' and ']' stand for Iverson brackets.

Despite using these biases (*i*)–(*iii*), the system (Beldiceanu et al. 2022) missed many conjectures. Rather than extending these biases further or introducing new biases, we combine these biases to catch complex formulae. To avoid a combinatorial explosion, this is not done by assembling a merged grammar associated with different biases, but rather by *devising some recursive decomposition techniques that identify the sub-terms of a formula and how these sub-terms are connected by arithmetic operators*. The base case of such recursion belongs to biases (*i*)–(*iii*). The next section provides a first insight on how this is achieved.

2.2 Running Examples and Intuition of the Decomposition Technique

Conj. (1)–(4) illustrate sharp lower bounds found by the decomposition method that could not be found before, as they were outside the scope of biases (*i*)–(*iii*). They are used as running examples and were proved in (Cheukam-Ngouonou et al. 2023) to stress the fact that the decomposition method can find non-obvious conjectures that turn out to be true. In the first phase, the decomposition method identifies an incomplete formula, i.e. a formula for which some terms are still unknown and will be determined later in the 2nd phase by applying the decomposition method recursively. In Conj. (1)–(4), the right-hand side of each inequality matches terms, highlighted by a brace, connected by one or two arithmetic operators (or a conditional), where:

- Terms with no grey background refer to expressions matching biases (*i*)–(*iii*), that are found in the 1st phase.
- Terms with a grey background are found in the 2nd phase when the decomposition is applied recursively.

Conjecture 1 *Conj. (1) provides a sharp lower bound on the number of connected components c of a digraph where every vertex is adjacent to at least an arc wrt its number of vertices v, the maximum number of vertices $\overline{c}$ inside a connected component, and the smallest number of vertices $\underline{s}$ in a strongly connected component (scc). The right-hand side of Inequality (1) consists of the terms (1.1) and (1.2), resp. referring to biases (i) and (iii), and linked by a sum.*

$$c \geq \underbrace{\lceil v/\overline{c} \rceil}_{\text{(1.1) binary function}} + \underbrace{[\neg((2 \cdot \underline{s} \leq \overline{c}) \vee (\underline{s} \geq (v \bmod \overline{c} = 0\ ?\ \overline{c}\ :\ v \bmod \overline{c}))]}_{\text{(1.2) Boolean term}} \quad (1)$$

In Phase one, the decomposition method finds a formula of the form $g_{1,1}(v, \overline{c}) + g_{1,2}(v, \overline{c}, \underline{s})$, where $g_{1,1}$ has only 2 parameters, and where the codomain of $g_{1,2}$ is the set $\{0, 1\}$; in the 2nd phase, the method finds the functions $g_{1,1}$ and $g_{1,2}$.

Conjecture 2 *Conj. (2) gives a sharp lower bound c of a digraph wrt its number of scc s, and the $\overline{c}$ and $\underline{s}$ characteristics introduced in Ex. 1. The term (2.1) is the isolated input parameter s and the term (2.2), i.e. $\lfloor \overline{c}/\underline{s} \rfloor$, refers to bias (i). These terms are connected by an integer division rounded up.*

$$c \geq \left\lceil \underbrace{s}_{\text{(2.1) unary function}} \Big/ \underbrace{\lfloor \overline{c}/\underline{s} \rfloor}_{\text{(2.2) binary function}} \right\rceil \quad (2)$$

In Phase one, the method finds the formula $\left\lceil \frac{g_{2,1}(s)}{g_{2,2}(\overline{c}, \underline{s})} \right\rceil$, with $g_{2,1}(s) = s$, and where $g_{2,2}$ has only 2 parameters; in the 2nd phase, the method finds the function $g_{2,2}$ itself.

Conjecture 3 *Conj. (3) depicts a sharp lower bound on the maximum number of vertices $\overline{s}$ inside an scc of a digraph wrt the s and $\underline{s}$ characteristics previously introduced. Within Conj. (3), the term (3.1) is a conditional expression, i.e. bias (ii), the term (3.2) refers to a unary function, and the term (3.3), i.e. $[s = 1]$, corresponds to a Boolean expression, i.e. bias (iii). These terms are connected by the division rounded up and the sum operators.*

$$\overline{s} \geq \left\lceil \underbrace{(v = \underline{s}\ ?\ v : v - \underline{s})}_{\substack{\text{(3.1) binary function as} \\ \text{a conditional}}} \Big/ \Big(\underbrace{s - 1}_{\substack{\text{(3.2) unary} \\ \text{function}}} + \underbrace{[s = 1]}_{\substack{\text{(3.3)} \\ \text{Boolean} \\ \text{term}}} \Big) \right\rceil \quad (3)$$

In Phase 1, the decomposition method finds a formula of the form $\left\lceil \frac{g_{3,1}(v, \underline{s})}{g_{3,2}(s) + g_{3,3}(s)} \right\rceil$, with $g_{3,2}(s) = s - 1$, and where $g_{3,1}$ has only 2 parameters, and where the codomain of $g_{3,3}$ is the set $\{0, 1\}$; in the 2nd phase, the method finds $g_{3,1}$ and $g_{3,3}$.

Conjecture 4 *Conj. (4) depicts a sharp lower bound on the maximum number of vertices $\overline{c}$ inside a connected component of a digraph wrt the v, c, $\underline{c}$ and $\overline{s}$ characteristics previously introduced. The right-hand side of Conj. (4) is a complex conditional expression outside the scope of bias (ii), as its 'else' part is too complicated. It consists of three parts:*

- *A simple condition $v = c \cdot \underline{c}$ denoted by (4.1);*
- *A 'then' part, $\underline{c}$, depicted by (4.2);*
- *An 'else' part, $\max\left(\overline{s}, \left\lceil \frac{v-\underline{c}}{c-1} \right\rceil\right)$, referring to a complex term labelled by (4.3).*

$$\overline{c} \geq \Bigg(\underbrace{v = c \cdot \underline{c}?}_{\substack{(4.1)\\ \text{condition}}} \quad \underbrace{\underline{c}}_{\substack{(4.2)\\ \text{'then' part}}} \quad : \quad \underbrace{\max\left(\overline{s}, \left\lceil \frac{v-\underline{c}}{c-1} \right\rceil\right)}_{\text{(4.3) 'else' part}} \Bigg) \tag{4}$$

The method first finds $(v = g_{4,1}(c, \underline{c}) \ ? \ g_{4,2}(\underline{c}) : g_{4,3}(v, c, \underline{c}, \overline{s}))$ with $g_{4,1}(c, \underline{c}) = c \cdot \underline{c}$, $g_{4,2}(\underline{c}) = \underline{c}$; then it finds function $g_{4,3}$ using the method in a recursive way:

- It finds $g_{4,3}(v, c, \underline{c}, \overline{s}) = \max(g_{4,3,1}(\overline{s}), g_{4,3,2}(v, c, \underline{c}))$, with $g_{4,3,1}(\overline{s}) = \overline{s}$, and where $g_{4,3,2}$ has 3 parameters;
- It then finds function $g_{4,3,2}$ using again the decomposition method in a recursive way:
  - It first finds $g_{4,3,2}(v, c, \underline{c}) = \left\lceil \frac{g_{4,3,2,1}(v,\underline{c})}{g_{4,3,2,2}(c)} \right\rceil$ with $g_{4,3,2,2}(c) = c - 1$;
  - It then finds function $g_{4,3,2,1}(v, \underline{c})$ directly as $v - \underline{c}$, a polynomial, i.e. bias (*i*).

Wrapping Up the Examples We saw four conjectures acquired by our decomposition method. They cover all the types of decompositions we found. Each function introduced in Phase 1 when searching for an incomplete formula is simpler than the function initially looked for: (*a*) it has *fewer input parameters*, e.g. in Conj. 1, $g_{1,1}(v, \overline{c}) = \lceil v/\overline{c} \rceil$ does not mention the $\underline{s}$ parameter, or (*b*) its codomain is *restricted to two values*, e.g. in Conj. 1, the codomain of $g_{1,2}$ is the set $\{0, 1\}$, or (*c*) it *only holds if a given condition is met*, e.g. the 'else' part in Conj. (4) only holds if $v \neq c \cdot \underline{c}$.

2.3 Decompositions Definition

By decomposing the formula to acquire into an expression consisting of several simpler terms as mentioned in Sect. 2.2, we aim to solve an easier acquisition problem. To this end, we introduce four ways of decomposing a function. Before formally defining them, we explain each of them intuitively.

- **[Adding a Boolean expression]** The first decomposition breaks down a function as the sum of a function with fewer input parameters, and a function whose codomain is the set $\{0, 1\}$. As shown in Conj. (1), this decomposition was motivated by the possibility of approximating a bound by an error margin of at most 1.
- **[Isolating a parameter]** The 2nd decomposition is based on the idea that one may find a formula in which an input parameter only occurs in the outer term of the formula where no other input parameter is used. In Conj. (2), the parameter s only occurs in the numerator of the top-level of binary function 'integer division rounded up'.
- **[Isolating a parameter and using a** 0-1 **slack]** The 3rd decomposition generalises the 2nd decomposition by introducing a 0-1 slack term that can refer to any subset of the input parameters. In Conj. (3), the input parameter s does not occur in the numerator of the formula; indeed, it is only mentioned by the denominator of the top level function 'integer division rounded up'.
- **[Introducing a conditional]** The 4th decomposition is based on the intuition that it may be easier to find a formula that applies to a subset of the table entries rather than to all. The 4th decomposition divides the table entries into a 'then' and an 'else' set using a simple condition then, for each set, we have to identify a corresponding formula. We use a small set of predefined formulae with a subset of the condition's parameters for the 'then' set, while imposing no restriction on the 'else' set. Conj. (4) illustrates such conditional decomposition.

Problem Given a table $\mathrm{tab}[1..nrow, 1..ncol+1]$ of integer values, consisting of $nrow$ rows and $ncol+1$ columns, where columns $1, 2, \ldots, ncol$ form a mfd determining column $ncol + 1$, we address the following question: How to decompose the problem of discovering a function g satisfying all the following set of equalities

$$\forall j \in [1, nrow] : \ \mathrm{tab}[j, ncol{+}1] = g(\mathrm{tab}[j, 1], \ldots, \mathrm{tab}[j, ncol]) \tag{5}$$

into a set of easier subproblems requiring finding a limited number of functions satisfying one of the decompositions (6)–(9) of Def. 1 that we now introduce.

Definition 1 (Decomposition Types)

$$\forall j \in [1, nrow] : \mathrm{tab}[j, ncol+1] = g_1(\mathrm{tab}[j, a_1], \ldots, \mathrm{tab}[j, a_{\ell_1}]) + g_2(\mathrm{tab}[j, b_1], \ldots, \mathrm{tab}[j, b_{\ell_2}] \tag{6}$$

$$\forall j \in [1, nrow] : \ \mathrm{tab}[j, ncol+1] = g_1(\mathrm{tab}[j, a_1], \ldots, \mathrm{tab}[j, a_{\ell_1}]) \ \oplus g_3(\mathrm{tab}[j, b_1]) \tag{7}$$

$$\forall j \in [1, nrow] : \ \mathrm{tab}[j, ncol+1] = g_1(\mathrm{tab}[j, a_1], \ldots, \mathrm{tab}[j, a_{\ell_1}]) \oplus (g_2(\mathrm{tab}[j, b_1], \ldots, \mathrm{tab}[j, b_{\ell_2}]) + g_3(\mathrm{tab}[j, b_1])) \tag{8}$$

$$\forall j \in [1, nrow] : \ \mathrm{tab}[j, ncol+1] = (\mathrm{cond}(\mathrm{tab}[j, c_1], \ldots, \mathrm{tab}[j, c_{\ell_3}]) \ ? \ g_4(\mathrm{tab}[j, d_1], \ldots, \mathrm{tab}[j, d_{\ell_4}]) : \ g_1(\mathrm{tab}[j, a_1], \ldots, \mathrm{tab}[j, a_{\ell_1}])) \tag{9}$$

1. *$g_1 : \mathbb{Z}^{\ell_1} \to \mathbb{Z}$ refers to one of the biases (*i*)–(*iii*) or to a formula obtained by one of the four decompositions; $a_1, \ldots, a_{\ell_1}$ are distinct indices from $[1, ncol]$ with $\ell_1 \in [1, ncol - 1]$ for (6)–(8) as g_1 does not involve all input parameters, and with $\ell_1 \in [1, ncol]$ for (9). For (7)–(8), b_1 is different from $a_1, \ldots, a_{\ell_1}$.*
2. *$g_2 : \mathbb{Z}^{\ell_2} \to \{0, 1\}$ matches bias (*iii*); $b_1, \ldots, b_{\ell_2}$ are distinct indices from $[1, ncol]$ with $\ell_2 \in [1, ncol]$. Note that in (6), functions g_1 and g_2 may share some parameters.*

3. *$g_3 : \mathbb{Z} \to \mathbb{Z}$ is one of the unary functions $A \cdot x^2 + B \cdot x + C$, $\lfloor \frac{A \cdot x^2 + B \cdot x}{D} \rfloor$, $\lceil \frac{A \cdot x^2 + B \cdot x}{D} \rceil$, $\min(A \cdot x + B, C)$, $\max(A \cdot x + B, C)$, $(A \cdot x + B) \bmod D$, $|A \cdot x + B|$, $[((x + A) \bmod D) = C]$, $[((x + A) \bmod D) \geq C]$, $[((x + A) \bmod D) \leq C]$, with A, B, $C \in \mathbb{Z}$, and $D \in \mathbb{Z}^+$. To limit the search space, we consider unary functions involving up to 3 constants.*
4. *Within (7)–(8), $\oplus$ stands for one of the operators '$+$', '$\cdot$', '$\min$', '$\max$', '$\lfloor \text{-} \rfloor$', or '$\lceil \text{-} \rceil$'.*
5. *Within (9), cond is a condition mentioning at most 3 parameters, i.e. $\ell_3 \in [1, 3]$, of the form '$x = \min(x)$', '$x = y$', '$x \leq y$', '$x \bmod y = 0$', '$x = y \cdot z$', '$A \cdot x \leq y$', while g_4 is one of the functions 'B', 'x', '$[x = \min(x)]$', '$[x > \min(x)]$', '$x \cdot y$', '$[x = y + B]$', '$x = y + z$', with 'A', 'B' $\in \mathbb{Z}$.*

We introduce a fair number of functions and conditions for g_3, g_4, and cond in Def. 1. To avoid overfitting, they mention at most 3 coefficients whose range is restricted to $[-2, 2]$.

Example 1 *Within Sect. 2.2, the right-hand part of inequalities (1), (2), (3), and (4) resp. matches the following decomposition types:*

- *(6) with $g_1(v, \overline{c}) = \lceil \frac{v}{\overline{c}} \rceil$ and $g_2(v, \overline{c}, \underline{s}) = [\neg((2 \cdot \underline{s} \leq \overline{c}) \vee (\underline{s} \geq (v \bmod \overline{c} = 0 \ ? \ \overline{c} \ : \ v \bmod \overline{c}))]$.*
- *(7) with $g_1(\overline{c}, \underline{s}) = \lfloor \overline{c} / \underline{s} \rfloor$, $g_3(s) = s$, and $\oplus =$ '$\lceil \text{-} \rceil$'.*
- *(8) with $g_1(v, \underline{s}) = (v = \underline{s} \ ? \ v : v - \underline{s})$, $g_2(s) = [s = 1]$, $g_3(s) = s - 1$, and $\oplus =$ '$\lceil \text{-} \rceil$'.*
- *(9) with $g_1(v, c, \underline{c}, \overline{s}) = \max\left(\overline{s}, \left\lceil \frac{v - \underline{c}}{c - 1} \right\rceil\right)$ and $g_4(\underline{c}) = \underline{c}$.*

3 Implementing the Decompositions

The implementation combines (*a*) phases of generating a limited number of alternatives on the type of functions used in the decomposition and on which parameters these functions mention, and (*b*) test phases verifying certain simple conditions and solving a constraint model to find the values of the coefficients of the functions mentioned in the terms of the decomposition. We introduce some notation to refer to intermediate structures used to analyse the consequence of eliminating an input parameter from a mfd.

Notation 1 *Consider the table $\mathrm{tab}[1..nrow, 1..ncol + 1]$, in which the first $ncol$ columns, the input parameters, form a mfd determining column $ncol + 1$, i.e. the output parameter.*

- *Let $\mathrm{tab}_k^{\nearrow}[1..nrow, 1..ncol + 1]$ be the table obtained by sorting the rows of the table $\mathrm{tab}[1..nrow, 1..ncol + 1]$ in increasing lexicographic order wrt columns $1, \ldots, k - 1, k + 1, \ldots, ncol + 1$, i.e. column k is skipped; to make the correspondence between the entries of the tables tab_k and $\mathrm{tab}_k^{\nearrow}$, let σ_k denote the permutation that maps the j-th row of the table tab_k to the $\sigma_k(j)$-th row of the table $\mathrm{tab}_k^{\nearrow}$ (with $j \in [1, nrow]$).*
- *Let $\mathcal{I}$ denote the parameters associated with columns $1, 2, \ldots, ncol$ of the table* tab*, and let $\mathcal{I}_j$ (with $j \in [1, nrow]$) represents the corresponding* parameter values *for the j-th row of the table* tab*, i.e. values $\mathrm{tab}[j, 1], \mathrm{tab}[j, 2], \ldots, \mathrm{tab}[j, ncol]$.*
- *Let $\mathcal{I}^k$ (with $k \in [1, ncol]$) denote the parameters associated with columns $1, \ldots, k - 1, k + 1, \ldots, ncol$ of the table* tab*, while let $\mathcal{I}_j^k$ (with $j \in [1, nrow]$) represents the corresponding* parameter values $\mathrm{tab}[j, 1], \ldots, \mathrm{tab}[j, k-1], \mathrm{tab}[j, k+1], \ldots, \mathrm{tab}[j, ncol]$.

3.1 Decomposition of Type (6) [adding a Boolean expression]

Question We want to check whether there is a $k \in [1, ncol]$ such that $\forall j \in [1, nrow] : \mathrm{tab}[j, ncol + 1] = g_1(\mathcal{I}_j^k) + g_2(\mathcal{I}_j)$, with $g_2 : \mathbb{Z}^{\ell_2} \to \{0, 1\}$; i.e. we seek an approximation with a maximum error of 1, by using a function g_1, without parameter k, and a correction term g_2.

Steps for Finding a Decomposition of Type (6)

1. **[Determining the parameters of g_1]** First, we successively select the k-th column (with $k \in [1, ncol]$) to remove from the input parameters of the function g_1, and we apply Steps 2. to 4. for each candidate column k.
2. **[Checking whether the codomain of g_2 is the set $\{0, 1\}$]** Second, provided function g_1 does not use the k-th input parameter selected in Step 1, we analyse how this affects the codomain of function g_2, even if functions g_1 and g_2 are yet unknown.

   For each maximum interval of consecutive rows $[\ell, u]$ in the sorted table $\mathrm{tab}_k^{\nearrow}[1..nrow, 1..ncol + 1]$ for which columns $1, \ldots, k - 1, k + 1, \ldots, ncol$ have the same value, we get the maximum $max_{\ell,u}$ and minimum $min_{\ell,u}$ values in the $(ncol + 1)$-th column, and we check that the difference $max_{\ell,u} - min_{\ell,u}$ does not exceed 1. In other words, we test for the table $\mathrm{tab}_k^{\nearrow}$ that, for each combination of identical input parameters, from which the k-th input parameter is ignored, the corresponding output parameter varies by at most 1. When satisfied, this test ensures that the codomain of g_2 is in $\{0, 1\}$.

   For each entry $j \in [\ell, u]$ of the table $\mathrm{tab}_k^{\nearrow}$ we set $\min_k^{\nearrow}[j] = min_{\ell,u}$, where $\min_k^{\nearrow}$ is a one-dimensional table whose entries vary from 1 to $nrow$.
3. **[Determining the values of $g_1(\mathcal{I}_j^k)$ and $g_2(\mathcal{I}_j)$]** Third, for each combination of input parameters of functions g_1 and g_2 we compute their respective output values: $\forall j \in [1, nrow]$, $g_1(\mathcal{I}_j^k) = \min_k^{\nearrow}[\sigma_k(j)]$ and $g_2(\mathcal{I}_j) = \mathrm{tab}_k^{\nearrow}[\sigma_k(j), ncol + 1] - \min_k^{\nearrow}[\sigma_k(j)]$.
4. **[Using $g_1(\mathcal{I}_j^k)$ and $g_2(\mathcal{I}_j)$ for identifying functions g_1 and g_2]** We search for g_1 by using the CP solvers associated with biases (*i*)–(*iii*) or by applying recursively one of the decompositions of this paper. To identify g_2 we call the Boolean solver associated with the bias (*iii*).

Example 2 (Illustrating the Search for a Decomposition of Type (6) for Conj. (1)) *Part (A) of Table 1 provides 9 entries of the table* tab *with input parameters v, $\overline{c}$, $\underline{s}$ and the lower bound of the output parameter c, previously introduced. Assume we skip the third column of table* tab*, $k = 3$, shown in grey in table $\mathrm{tab}_3^{\nearrow}$, i.e. we ignore column $\underline{s}$.*

- *Parts (B1) and (B2) resp. show the tables introduced for finding a decomposition of Type (6), i.e. tables* tab*, $\mathrm{tab}_3^{\nearrow}$, and $\min_3^{\nearrow}$. The permutation σ_3 (with $\sigma_3(3) = 4$, $\sigma_3(4) = 3$, and $\sigma_3(j) = j$ otherwise) maps the entries of table* tab *to the entries of table $\mathrm{tab}_3^{\nearrow}$. The rows of $\mathrm{tab}_3^{\nearrow}$*

	v	$\overline{c}$	$\underline{s}$	c
1	9	2	1	5
2	9	3	1	3
3	**9**	**3**	**2**	**4**
4	9	3	3	3
5	9	4	1	3
6	9	4	2	3
7	9	5	1	2
8	9	5	2	2
9	9	5	4	2

(A) $\mathrm{tab}[1..9,1..4]$

σ_3

	v	$\overline{c}$	$\underline{s}$	c
1	9	2	1	5
2	9	3	1	3
3	9	3	3	3
4	**9**	**3**	**2**	**4**
5	9	4	1	3
6	9	4	2	3
7	9	5	1	2
8	9	5	2	2
9	9	5	4	2

(B1) $\mathrm{tab}_3^{\nearrow}[1..9,1..4]$

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

(B2) $\min_3^{\nearrow}[1..9]$

	v	$\overline{c}$	$g_1(v,\overline{c})$
1	9	2	5
2	9	3	3
3	**9**	**3**	**3**
4	9	3	3
5	9	4	3
6	9	4	3
7	9	5	2
8	9	5	2
9	9	5	2

(C1)

	v	$\overline{c}$	$\underline{s}$	$g_2(v,\overline{c},\underline{s})$
1	9	2	1	0
2	9	3	1	0
3	**9**	**3**	**2**	**1**
4	9	3	3	0
5	9	4	1	0
6	9	4	2	0
7	9	5	1	0
8	9	5	2	0
9	9	5	4	0

(C2)

Table 1: Tables used to find a decomposition of type 6 for Conj. (1); bold entries refer to Ex. 2.

and $\min_3^{\nearrow}$ *can be partitioned in four maximum intervals, depicted in dark and light grey, resp. corresponding to the pair of values* $(9,2)$*,* $(9,3)$*,* $(9,4)$*, and* $(9,5)$ *for the input parameters* v *and* $\overline{c}$*. As for each of these four intervals the difference between the maximum and the minimum value of c does not exceed one, we can compute the values of* $g_1(v,\overline{c})$ *and* $g_2(v,\overline{c},\underline{s})$*.*

- *Parts (C1) and (C2) resp. give the tables used to acquire* $g_1(v,\overline{c})$ *and* $g_2(v,\overline{c},\underline{s})$*, e.g. for* j=3*,* $g_1(\mathcal{I}_j^3)$=$g_1(\mathcal{I}_3^3)$=$g_1(9,3)$=$\min_3^{\nearrow}[\sigma_3(j)]$=$\min_3^{\nearrow}[\sigma_3(3)]$=$\min_3^{\nearrow}[4]$=3*, and* $g_2(\mathcal{I}_j)$=$g_2(9,3,2)$=$\mathrm{tab}_3^{\nearrow}[\sigma_3(j),4] - \min_3^{\nearrow}[\sigma_3(j)]$=$\mathrm{tab}_3^{\nearrow}[4,4] - \min_3^{\nearrow}[4] = 4-3 = 1$*.*

3.2 Decompositions of Types (7) and (8) (isolating a parameter)

Using a binary operator $\oplus$, Decomposition (7) combines 2 sub-terms: a function g_3 involving a single input parameter, with a function g_1 mentioning only all other remaining input parameters. Decomposition (8) extends (7) a bit by adding an extra term whose codomain is the set $\{0,1\}$. As identifying Decomposition (8) is very similar to identifying Decomposition (7), we focus on the latter for space reasons.

Question We want to check whether there is a $k \in [1, ncol]$ such that $\forall j \in [1, nrow] : \mathrm{tab}[j, ncol+1] = g_1(\mathcal{I}_j^k) \oplus g_3(\mathrm{tab}[j,k])$, where $\oplus$ and function g_3 were defined in Def. 1; i.e. we want to see if we can express the formula we are looking for, by restricting one of its parameters to just one of the formula's sub-terms.

1. **[Selecting k, $\oplus$, and g_3]** To determine the images of function g_1 that will be used to find function g_1 itself in Step. 2 we successively consider the $ncol \times 10 \times 6$ combinations of triples $\langle k, g_3, \oplus\rangle$ (with $k \in [1, ncol]$), see Items 3–4 of Def. 1 for g_3 and $\oplus$. To find whether a combination of triples can be used or not to find the images of the function g_1, we apply the following steps:
   (a) **[Creating the "value variables" for the images of g_1]** For each maximum interval of consecutive rows $[\ell, u]$ of the table $\mathrm{tab}_k^{\nearrow}[1..nrow, 1..ncol+1]$ for which columns $1,\ldots,k-1,k+1,\ldots,ncol$ have the same value, we create a single domain variable y_ℓ *representing the value* of $g_1(\mathcal{I}^k_{\sigma_k^{-1}(\ell)})$, where σ_k^{-1} denotes the inverse permutation of permutation σ_k.
   (b) **[Stating the row constraints for finding the coefficients of g_3, and the value of g_1 for each row]** For each entry j of a maximal interval of consecutive rows $[\ell, u]$ of the table $\mathrm{tab}_k^{\nearrow}[1..nrow, 1..ncol+1]$ for which columns $1\ldots,k-1,k+1,\ldots,ncol$ have the same value, we create the constraint $y_\ell \oplus g_3(\mathrm{tab}[\sigma_k^{-1}(j),k]) = \mathrm{tab}[\sigma_k^{-1}(j), ncol+1]$.
   (c) **[Solving the row constraints]** We solve the conjunction of constraints stated in Step 1b: we find the values of the coefficients of the unary function g_3, and the values of the "value variables" of g_1, while minimising the sum of the absolute value of the coefficients of g_3 using a CP solver.

   Among all triples for which Step 1c found a solution, we keep those triples $\langle k, \oplus, g_3\rangle$ which minimise the sum of absolute values of the coefficients of g_3.
2. **[Identifying function g_1]** As for the decomposition of Type (6), we search for function g_1 by employing the existing CP solvers associated with biases (*i*)–(*iii*) or by applying recursively one of the 4 decompositions proposed in this paper. For this purpose we reuse the value of the "value variables" found for g_1 in Step 1c.

Example 3 (Illustrating the Search for a Decomposition of Type (7) for Conj. (2)) *Part (A) of Table 2 provides* 9 *entries of the data for the input parameters s,* $\overline{c}$*,* $\underline{s}$ *and the lower bound of the output parameter c, namely table* tab *previously introduced. Assume we skip the first column of the table* tab*, i.e.* $k=1$ *shown in grey in the table* $\mathrm{tab}_1^{\nearrow}$*, that is we ignore the column labelled by s. The rows of Parts (B)–(C) of Table 2 can be partitioned in three maximum intervals, depicted in dark and light grey, resp. corresponding to the pair of values* $(1,1)$*,* $(3,2)$*, and* $(9,3)$ *for the input parameters* $\overline{c}$ *and* $\underline{s}$*.*

- *Assuming we look for a function* g_3 *of Type* $A\cdot s^2 + B\cdot s + C$*, and for a binary operator* $\oplus$ *of the form '*$\lceil$-$\rceil$*', the three columns in Part (C) resp. show for each row of* $\mathrm{tab}_1^{\nearrow}$*, (*p1*) the unary function* g_3*, (*p2*) the "value variables" for* g_1*, and (*p3*) the corresponding constraints.*
- *Part (D) gives the derived table used to acquire* $g_1(\overline{c},\underline{s})$*.*

3.3 Decomposition of Type (9) (conditional)

Type (9) decomposition combines a simple condition and a simple function g_4 for the 'then' part of the condition, with a

	s	$\overline{c}$	$\underline{s}$	c
1	1	1	1	1
2	2	1	1	2
3	2	3	2	2
4	2	9	3	1
5	3	1	1	3
6	3	3	2	3
7	3	9	3	1
8	4	3	2	4
9	4	9	3	2

(A) tab

σ_1^{-1}

	s	$\overline{c}$	$\underline{s}$	c
1	1	1	1	1
2	1	1	1	2
3	1	1	1	3
4	2	3	2	2
5	3	3	2	3
6	4	3	2	4
7	2	9	3	1
8	3	9	3	1
9	4	9	3	2

(B) $\text{tab}_1^{\nearrow}$

$g_3(s)=A\cdot s^2+B\cdot s+C$	$g_1(\overline{c},\underline{s})$	$\lceil g_3(s)/g_1(\overline{c},\underline{s})\rceil = c$
$A+B+C$	y_1	$\lceil(A+B+C)/y_1\rceil=1$
$4\cdot A+2\cdot B+C$	y_1	$\lceil(4\cdot A+2\cdot B+C)/y_1\rceil=2$
$9\cdot A+3\cdot B+C$	y_1	$\lceil(9\cdot A+3\cdot B+C)/y_1\rceil=3$
$4\cdot A+2\cdot B+C$	y_4	$\lceil(4\cdot A+2\cdot B+C)/y_4\rceil=2$
$9\cdot A+3\cdot B+C$	y_4	$\lceil(9\cdot A+3\cdot B+C)/y_4\rceil=3$
$16\cdot A+4\cdot B+C$	y_4	$\lceil(16\cdot A+4\cdot B+C)/y_4\rceil=4$
$4\cdot A+2\cdot B+C$	y_7	$\lceil(4\cdot A+2\cdot B+C)/y_7\rceil=1$
$9\cdot A+3\cdot B+C$	y_7	$\lceil(9\cdot A+3\cdot B+C)/y_7\rceil=1$
$16\cdot A+4\cdot B+C$	y_7	$\lceil(16\cdot A+4\cdot B+C)/y_7\rceil=2$

(C)

	$\overline{c}$	$\underline{s}$	$g_1(\overline{c},\underline{s})$
1	1	1	1 (y_1)
2	1	1	1 (y_1)
3	3	2	1 (y_4)
4	9	3	3 (y_7)
5	1	1	1 (y_1)
6	3	2	1 (y_4)
7	9	3	3 (y_7)
8	3	2	1 (y_4)
9	9	3	3 (y_7)

(D)

Table 2: (A),(B),(D) Tables, and (C) Constraints used for finding a decomposition of Type 7 for Conj. (2); variables y_1, y_4, and y_7 in Tables (C) and (D) correspond to the "value variables" for g_1.

function g_1 corresponding to biases (*i*)–(*iii*), or obtained by one of the first three decompositions described in this paper, i.e. the conditional decomposition is not applied recursively, as this has proven to be very time-consuming. We use the following steps to search for a decomposition of Type (9):

1. **[Selecting** cond **and** g_4**]** We successively consider the 6×7 combinations of pairs $\langle \text{cond}, g_4 \rangle$, see Item 5 of Def. 1. To determine whether or not a combination of pairs can be used, we create this constraint model:
   (a) The variables $c_1, c_2, \ldots, c_n$ (resp. $d_1, d_2, \ldots, d_m$) denote the indices of the columns of the table $\text{tab}[1..nrow, 1..ncol+1]$ used in the condition cond (resp. in function g_4 of the 'then' part). These variables are in $[1, ncol]$ as they correspond to input parameters, i.e. we state the constraints $\forall i \in [1,n] : c_i \in [1, ncol]$, $\text{alldifferent}([c_1, c_2, \ldots, c_n])$, $\forall i \in [1,m] : d_i \in [1, ncol]$, and $\text{alldifferent}([d_1, d_2, \ldots, d_m])$.
   (b) For each entry j (with $j \in [1, nrow]$) of the table $\text{tab}[1..nrow, 1..ncol+1]$, we state the constraints:
   i. $\forall k \in [1, n\,] : \text{element}(c_k, \text{tab}[j, 1..ncol], v_{j,k}\,)$, $\text{cond}(v_{j,1}, v_{j,2}, \ldots, v_{j,n}) \Leftrightarrow r_j, r_j \in [0,1]$,
   ii. $\forall k \in [1,m] : \text{element}(d_k, \text{tab}[j, 1..ncol], w_{j,k})$,
   iii. $r_j = 1 \Rightarrow g_4(w_{j,1}, \ldots, w_{j,m}) = \text{tab}[j, ncol+1]$.
   (c) By maximising the number of rows in the table $\text{tab}[1..nrow, 1..ncol+1]$ for which condition 'cond' is met, we try to create a smaller subproblem for acquiring the 'else' part. This is done by stating the constraints $cost = \sum_{j \in [1,nrow]} r_j$, $cost > 0$, $cost < nrow$. The last two constraints require the 'then' (or 'else') part to contain at least one row for which condition 'cond' is true (or false) as we want to obtain a non-simplifiable conditional formula. We maximise $cost$ wrt the posted constraints.
2. **[Identifying function** g_1**]** As for decompositions (6)–(8), using Item 5 of Def. 1, we search for function g_1 using the CP solvers related to biases (*i*)–(*iii*), or by recursively applying decompositions (6)–(8). To do this, we focus only on all the j-th rows of the table $\text{tab}[1..nrow, 1..ncol+1]$ for which condition $\text{cond}(v_{j,1}, v_{j,2}, \ldots, v_{j,n})$ does not hold, i.e. the 'else' part of the conditional.

3.4 Integrating Decompositions and Biases (*i*)-(*iii*)

Trying simple formulae first, the decompositions have been integrated with biases (*i*)–(*iii*) in the following order.

1. Boolean-arithmetic formulae: bias (*iii*) when the output column of the table tab has only two values.
2. Simplest polynomial formulae: bias (*i*) with 1 monome.
3. Simple conditional formulae: bias (*ii*).
4. Simple polynomial formulae: bias (*i*), 2 or 3 monomes.
5. The decompositions (6), (7), (8), and (9), in this order.
6. Complex polynomial formulae: bias (*i*), 4 to 6 monomes.

4 Evaluation

Description of the Used Combinatorial Objects We evaluate the decomposition methods on the combinatorial objects DIGRAPH, ROOTED TREE, ROOTED FOREST, ROOTED FOREST2, PARTITION, PARTITION0, STRETCH, and CYCLIC STRETCH introduced in (Gindullin et al. 2023).

Definition 2 *For* DIGRAPH*,* ROOTED TREE*,* ROOTED FOREST*, and* ROOTED FOREST2*,* size *denotes the number of vertices. For* PARTITION0 *and* PARTITIONS*,* size *is the number of elements of the set we partition, and for* STRETCH *and* CYCLIC STRETCH*,* size *is the sequence length.*

Description of Input Tables The data set (Gindullin et al. 2023) consists of a collection of tables giving for any combinatorial object of size at most *size*, for any combination of at most 3 input parameters, for any feasible combinations of values of these input parameters, the sharp lower or the sharp upper bound of a given output parameter, e.g. Table (A) of Table 1 is an excerpt of such an input table. In addition, an input table may contain auxiliary parameters, called *secondary parameters*, all functionally determined by the input parameters. The tables represent 12 GB of data.

Conjectures We Are Looking For We search for (I) conjectures expressing a secondary parameter wrt input parameters, (II) conjectures expressing sharp bound on an output parameter wrt input parameters, (III) conjectures expressing a secondary parameter wrt both input and secondary parameters, (IV) conjectures expressing sharp

combinatorial object	n_o	1st version					2nd version										
		n_t	n_a	n_b	n_i	n_e	n_t	n_a	n_b	n_i	n_e	n_d	n_6	n_7	n_8	n_9	$n_{>1}$
DIGRAPH	2861	3270	2637	434	1789	2	3412	2702	447	1940	6	328	89	71	52	156	66
ROOTED TREE	185	225	138	67	119	0	240	152	77	133	1	39	10	12	6	18	8
ROOTED FOREST	2088	2343	1577	562	1250	4	2672	1697	613	1428	6	433	122	131	112	168	121
ROOTED FOREST2	2861	2404	1639	569	1372	1	2563	1700	607	1459	9	361	71	108	94	103	65
PARTITION	562	572	436	78	279	0	586	453	78	303	0	89	11	27	7	34	9
PARTITION0	235	238	189	37	134	0	282	209	38	162	0	65	14	12	10	20	13
STRETCH	6416	6481	4978	556	2157	4	6981	5237	582	2473	0	660	146	220	118	357	161
CYCLIC STRETCH	6589	5964	4484	521	2041	15	7011	5105	561	2486	32	882	103	299	131	636	309
total	21797	21497	16078	2824	9141	26	23747	17255	3003	10384	54	2857	566	880	530	1492	752

Table 3: Detailed experimental results for the 1st and the 2nd versions of the acquisition tool, where n_o is the number of secondary and output parameters across all tables, n_t is the number of acquired conjectures by the 1st or the 2nd version, n_a is the number of secondary and output parameters for which the 1st or the 2nd version could acquire at least a conjecture, n_b is the number of output parameters for which the 1st or the 2nd version could acquire at least a conjecture, n_i is the number of secondary and output parameters for which the 1st or the 2nd version could acquire at least a conjecture input parameters only, n_e is the number of conjectures invalidated on the largest available size of a combinatorial object, n_d is the number of output parameters for which the 2nd version could acquire at least a conjecture using decompositions (6)–(9), n_6, n_7, n_8, n_9 are the number of output parameters for which we could resp. acquire at least a conjecture using (6), (7), (8), (9), $n_{>1}$ is the number of output parameters for which the 2nd version found at least a conjecture using more than one decompositions.

bound on an output parameter wrt both input and secondary parameters. We prefer conjectures using input parameters only as it allows one to express sharp bounds directly wrt input parameters, i.e. without using secondary parameters. Focusing only on sharp bounds limits the number of conjectures learned, which now depends only on the number of characteristics considered for the combinatorial objects.

Experimental Setting We compare 2 versions of the acquisition tool using SICStus 4.7.1 on an cluster with Intel processors such as Silver 4216 Cascade Lake @ 2.1GHz, and E7-4809 v4 Broadwell @ 2.1Ghz. The source code for the decomposition consists of 1882 commented lines that are written in SICStus Prolog available from https://github.com/cquimper/MapSeekerAAAI24; The 1st version uses biases (*i*)–(*iii*), while the 2nd version uses biases (*i*)–(*iii*) and the 4 decompositions (6)–(9). If one of these versions took more than 96 hours to complete the acquisition for an input table, that table is excluded from the result evaluation, unless otherwise stated. We acquire conjectures on tables of smaller sizes and test them on the largest tables using the method described in (Beldiceanu et al. 2022) on selecting table size. We exclude invalidated conjectures from our evaluation.

Experimental Results Out of a total of 4469 tables, the 1st version had timeouts on 44 tables, the 2nd version on 49 tables, with almost no overlap. We remove these tables and use the remaining 4378 tables to compare 2 versions. The 4378 tables has 21797 secondary and output parameters. As cluster node performance varies and we cannot control allocation of tables over CPUs, we only compare the aggregated full acquisition time for both versions. The 1st version took 5888 hours in total, while the 2nd version took 25053 hours to complete. A total of 26 (resp. 54) conjectures acquired by the 1st (resp. 2nd) version were not validated.

Table 3 shows the detailed results of the experiment. The 1st and the 2nd versions resp. found conjectures for 16078 and 17255 secondary or output parameters. The 2nd version found conjectures of types I–IV (resp. I–II) for 5% more (resp. 14% more) secondary and output parameters compared to the 1st version. The 14% increase reflects the fact that the 2nd version expresses more conjectures with input parameters only, which is one of our goals. Including all 4469 tables, the 2nd version found conjectures for 7% more secondary and output parameters than the 1st version.

The 2nd version found 6% more conjectures of types II and IV, i.e. sharp bounds. In the 2nd version, 2857 secondary or output parameters (16.5% of the 17255 parameters) have conjectures that use decompositions (6)–(9); 26% of them use several decompositions in one conjecture. In (Cheukam-Ngouonou et al. 2023), we proved Conjectures (1)–(4) to show that decompositions find non-trivial sharp bounds, as well as a non-obvious conjecture for ROOTED TREE.

5 Conclusion

Although simple, the proposed method may seem counter-intuitive. It is based on the idea of identifying sub-terms of the formula being searched for *before actually knowing the formula itself*, by combining data analysis and CP wrt minimal functional dependencies. The method helps to find formulae whose sub-terms correspond to various biases, e.g. polynomial, conditional or Boolean expressions, as shown during the search for conjectures on sharp bounds. For our benchmark, the decomposition methods found 14% more conjectures expressed directly wrt input parameters. It also found non-obvious conjectures that we proved. Future work may use acquired conjectures to synthesise efficient filtering algorithms, as a lack of sharp bounds is a weakness of CP.

References

Alur, R.; Singh, R.; Fisman, D.; and Solar-Lezama, A. 2018. Search-based program synthesis. *Commun. ACM*, 61(12): 84–93.

Aouchiche, M.; Caporossi, G.; Hansen, P.; and Laffay, M. 2005. AutoGraphiX: a survey. *Electron. Notes Discret. Math.*, 22: 515–520.

Beldiceanu, N.; Cheukam-Ngouonou, J.; Douence, R.; Gindullin, R.; and Quimper, C. 2022. Acquiring Maps of Interrelated Conjectures on Sharp Bounds. In Solnon, C., ed., *28th International Conference on Principles and Practice of Constraint Programming, CP 2022, July 31 to August 8, 2022, Haifa, Israel*, volume 235 of *LIPIcs*, 6:1–6:18. Schloss Dagstuhl - Leibniz-Zentrum für Informatik.

Brence, J.; Todorovski, L.; and Džeroski, S. 2021. Probabilistic grammars for equation discovery. *Knowledge-Based Systems*, 224: 107077.

Cheukam-Ngouonou, J.; Gindullin, R.; Beldiceanu, N.; Douence, R.; and Quimper, C.-G. 2023. Proving Conjectures Acquired by Composing Multiple Biases. arXiv:2312.08990.

Gindullin, R.; Beldiceanu, N.; Cheukam-Ngouonou, J.; Douence, R.; and Quimper, C. 2023. Boolean-Arithmetic Equations: Acquisition and Uses. In Andre, C., ed., *Integration of AI and OR Techniques in Constraint Programming - 20th International Conference, CPAIOR 2023, Nice, France, May 29-June 1, 2023, Proceedings*, Lecture Notes in Computer Science. Springer.

Gulwani, S. 2011. Automating string processing in spreadsheets using input-output examples. In Ball, T.; and Sagiv, M., eds., *Proceedings of the 38th ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages, POPL 2011, Austin, TX, USA, January 26-28, 2011*, 317–330. ACM.

Gulwani, S.; Harris, W. R.; and Singh, R. 2012. Spreadsheet data manipulation using examples. *Commun. ACM*, 55(8): 97–105.

Hansen, P.; and Caporossi, G. 2000. AutoGraphiX: An Automated System for Finding Conjectures in Graph Theory. *Electron. Notes Discret. Math.*, 5: 158–161.

Larson, C. E.; and Cleemput, N. V. 2016. Automated conjecturing I: Fajtlowicz's Dalmatian heuristic revisited. *Artif. Intell.*, 231: 17–38.

Ligeza, A.; Jemiolo, P.; Adrian, W. T.; Slazynski, M.; Adrian, M.; Jobczyk, K.; Kluza, K.; Stachura-Terlecka, B.; and Wisniewski, P. 2020. Explainable Artificial Intelligence. Model Discovery with Constraint Programming. In Stettinger, M.; Leitner, G.; Felfernig, A.; and Ras, Z. W., eds., *Intelligent Systems in Industrial Applications, 25th International Symposium, ISMIS 2020, Graz, Austria, September 23-25, 2020, Selected Papers from the Industrial Part*, volume 949 of *Studies in Computational Intelligence*, 171–191. Springer.

Paramonov, S.; Kolb, S.; Guns, T.; and Raedt, L. D. 2017. TaCLe: Learning Constraints in Tabular Data. In Lim, E.; Winslett, M.; Sanderson, M.; Fu, A. W.; Sun, J.; Culpepper, J. S.; Lo, E.; Ho, J. C.; Donato, D.; Agrawal, R.; Zheng, Y.; Castillo, C.; Sun, A.; Tseng, V. S.; and Li, C., eds., *Proceedings of the 2017 ACM on Conference on Information and Knowledge Management, CIKM 2017, Singapore, November 06 - 10, 2017*, 2511–2514. ACM.

Srivastava, S.; Gulwani, S.; and Foster, J. S. 2013. Template-based program verification and program synthesis. *Int. J. Softw. Tools Technol. Transf.*, 15(5-6): 497–518.

Udrescu, S.; Tan, A. K.; Feng, J.; Neto, O.; Wu, T.; and Tegmark, M. 2020. AI Feynman 2.0: Pareto-optimal symbolic regression exploiting graph modularity. In Larochelle, H.; Ranzato, M.; Hadsell, R.; Balcan, M.; and Lin, H., eds., *Advances in Neural Information Processing Systems 33: Annual Conference on Neural Information Processing Systems 2020, NeurIPS 2020, December 6-12, 2020, virtual*.

Wikimedia Commons. 2003. Inductive Bias. https://en.wikipedia.org/wiki/Inductive_bias. Accessed: 2023-08-04.