Boolean-Arithmetic Equations: Acquisition and Uses

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

Boolean-Arithmetic Equations: Acquisition and Uses

R. Gindullin*[1,2], N. Beldiceanu[1,2], J. Cheukam Ngouonou[1,2,4], R. Douence[1,2,3], C.-G. Quimper[4]

IMT Atlantique[1], LS2N[2], INRIA[3], Université Laval[4]

Abstract. Motivated by identifying equations to automate the discovery of conjectures about sharp bounds on combinatorial objects, we introduce a CP model to acquire Boolean-arithmetic equations (BAE) from a table providing sharp bounds for various combinations of parameters. Boolean-arithmetic expressions consist of simple arithmetic conditions (SAC) connected by a single commutative operator such as '$\wedge$', '$\vee$', '$\oplus$' or '$+$'. Each SAC can use up to three variables, two coefficients, and an arithmetic function such as '$+$', '$-$', '$\times$', 'floor', 'mod' or 'min'. We enhance our CP model in the following way to limit the search space: (i) We break the symmetries linked to multiple instances of similar SACs in the same expression. (ii) We prevent the creation of SAC that could be simplified away. We identify several use cases of our CP model for acquiring BAE and show its applicability for learning sharp bounds for eight types of combinatorial objects as digraphs, forests, and partitions.

Keywords: Boolean-arithmetic equation · equation discovery · bounds.

1 Introduction

In the context of finding conjectures about combinatorial objects, the relevance of Boolean and BAE has been noted but not fully developed. Larson and Cleemput describe in [21] the use of pure Boolean expressions to represent necessary or sufficient conditions for a graph property, while [8] depicts the application of BAE to express sharp bounds of graph characteristics. While the first work uses a systematic generate and test approach, the second does not describe how such BAE were produced. Our work is motivated by the following observations: (i) we want to go beyond a generate and test approach [21], and investigate how CP can be used to identify a wide range of concise BAE in the context of conjecture acquisition; (ii) while the experimental part of [8] indicates the relevance to use BAE to get sharp bounds for digraphs characteristics, it was still unclear whether this applies to other combinatorial objects. The contribution of the paper is threefold.

1. First, it exhibits a variety of Boolean-arithmetic expressions $f(X_1, X_2, \ldots, X_n)$ which occur in practice when looking for sharp bounds.

* R. Gindulling is supported by the EU-funded ASSISTANT project no. 101000165.

2. Second, it provides CP models for acquiring Boolean arithmetic expressions. The main point of these models is to restrict the search space by breaking symmetries between similar simple arithmetic conditions (SACs) and avoiding generating simplifiable SACs.
3. Finally, it shows that Boolean-arithmetic expressions are not only relevant for expressing bounds for digraphs as mentioned in [8], but also for trees, forests, partitions, and some global constraints.

In Sect. 2, we describe four settings for the practical use of Boolean expressions that we observed in the context of sharp bound acquisition. In Sect. 3, we define the Boolean-arithmetic formulae that we consider throughout this paper. In Sect. 4, we provide a core CP model for learning a BAE that explains an output column of a table from a set of input columns. We show in Sect. 5 various extensions of the core model to restrict the search space of Boolean-arithmetic expressions. We evaluate the core model and its extensions in Sect. 6, discuss related work in Sect. 7, and conclude in Sect. 8.

2 The Relevance of Boolean-Arithmetic Equations

To show the expressive power of BAE, let us consider typical situations where they are relevant for acquiring sharp bounds, i.e. an inequality for which the equality holds for at least one example.

1. Using a BAE is a natural option when the codomain of $f(X_1, X_2, \ldots, X_n)$ is equal to $\{0, 1\}$ or more generally consists of only two distinct consecutive values v and $v+1$. For instance, let $v, a, \underline{os}$ and $\underline{s}$ be the number of vertices, of arcs, of smallest strongly connected components and the size of the smallest strongly connected component of a digraph. As found by the CP model of [8] when a is maximal, we have the relation $\underline{os} = \lfloor \frac{v}{\max(-\underline{s}+v, \underline{s})} \rfloor$, which is subsumed by the relation $\underline{os} = 1 + [v = 2 \cdot \underline{s}]$, where the Boolean expression $[v = 2 \cdot \underline{s}]$ is used as an integer, i.e. either 0 for false or 1 for true.
2. Even when the number of distinct values m of the codomain of $f(X_1, X_2, \ldots, X_n)$ is greater than two values, but still very small, we can use Boolean arithmetic expressions to capture concise formulae. This is done by summing up $m-1$ Boolean-arithmetic conditions as illustrated now: e.g., let v, a, c_1 and $\underline{c}$ be the number of vertices, of arcs, of connected components having more than one vertex and the size of the smallest connected component of a digraph. As found by the CP model in [8], when a is maximal, we have the relation $c_1 = \lfloor \frac{(v+\max(-\underline{c}+v, \underline{c}))}{(2 \cdot \max(\max(-\underline{c}+v, \underline{c}), 2) - \max(-\underline{c}+v, \underline{c})+1)} \rfloor$, which is subsumed by the BAE $c_1 = 2 - ([\underline{c} = 1] + [(v - \underline{c}) \leq 1])$.
3. Quite often, using BAE allows one simplifying formulae with min and max as illustrated now. Let v, c, c_{23} and $\overline{s}$ be the number of vertices, connected components, connected components with two or three vertices where the size of each strongly connected component is equal to 1, and the size of the largest strongly connected component of a digraph: e.g for the graph •→•→• •→• •↔• we have $v = 7, c = 3, c_{23} = 2, \overline{s} = 2$. As discovered by the

CP model of [8], when c is minimal, we have $c_{23} = (v.\overline{s} \leq 3\,?\ \min(v-1,1) : 0)$,[1] which can be replaced by the Boolean relation $c_{23} = [\overline{s} = 1 \wedge v \in [2,3]]$.

4. It may occur that a formula can provide an approximate bound with an error of at most 1 on a parameter in $\mathbb{Z}$. Then, a way for getting a sharp bound is to find a Boolean formula which precisely describes the bound discrepancy. For instance, a non-sharp lower bound (with a deviation of at most 1) on the number of connected components c of a digraph $\mathcal{G}$ wrt the number of vertices v of $\mathcal{G}$, the size of the largest connected component $\overline{c}$ of $\mathcal{G}$, and the size of the smallest strongly connected component $\underline{s}$ of $\mathcal{G}$ is given by $\lceil \frac{v}{\overline{c}} \rceil$; but a sharp lower is given by $\lceil \frac{v}{\overline{c}} \rceil + [(\underline{s} < \text{fmod}(v, \overline{c})) \wedge (2 \cdot \underline{s} > \overline{c})]$, where $\text{fmod}(x, y)$ is defined by the conditional expression $(x \bmod y = 0\ ?\ y : x \bmod y)$.

The following points are specific to our equation discovery context [29]:

- As our samples are noise-free, we need to acquire formulae that *correctly represent all the samples* we have.
- As our samples correspond to instances of combinatorial objects reaching a sharp bound, this is why we search for equations rather than for inequalities.
- Simple conditions are not translated into a large set of features, which is the case for most decision tree approaches [3,4,19].
- We keep the original columns of the tables, as using one-hot encoding considerably increases the number of columns and affects the interpretability [28].

3 Describing Boolean-Arithmetic Expressions

The type of Boolean-Arithmetic expressions we consider is dictated by two opposite objectives. (i) On the one hand, we want to focus on concise expressions involving a limited number of variables and constants. This is motivated by the need to generate interpretable formulae and by the necessity to avoid a combinatorial explosion when searching for such formulae [12]. Consequently, we limit the number of variables and constants, as well as the number of subterms of Boolean-Arithmetic expressions. (ii) On the other hand, we aim at covering a variety of Boolean expressions which occurs in practice. This is done by allowing one to use a variety of arithmetic operators and Boolean functions.
To meet the above objectives, we use the following two-level description:

- First, we consider a Boolean-arithmetic condition (BAC) mentioning *no more than two variables and two constants*, the comparison operators $\leq$, $=$, $\geq$, $\in$, $\notin$ and a *variety of arithmetic operators* such as $+$, $-$, $\times$, $\lfloor\text{-}\rfloor$, $\lceil\text{-}\rceil$, mod, min, max. We currently have 53 elementary arithmetic conditions.
- Second, we build a Boolean-arithmetic term by feeding several arithmetic conditions, or their negation, to a *commutative and associative aggregation operator* such as $+$, $\vee$, $\wedge$, $\oplus$, eq, card1, voting, where:
  - $\oplus$ stands for XOR;
  - eq is equal to 1 iff all its conditions are evaluated to the same value;

[1] The expression $(cond\,?\,x : y)$ denotes x if condition $cond$ holds, y otherwise.

- card1 is equal to 1 iff only one of its conditions is evaluated to 1;
- voting is equal to 1 iff the majority of its conditions are evaluated to 1.

The use of a commutative and associative aggregation operator simplifies the interpretability of a formula and reduces the combinatorics, as the order of the BACs within a Boolean-arithmetic term is irrelevant. It allows for a compact representation of some Boolean expressions, that would otherwise be large when expressed in conjunctive or disjunctive normal form without introducing new variables. For instance, the n-ary XOR, i.e. $\oplus_{i=1}^{n} \ell_i$, is represented by a CNF consisting of 2^{n-1} clauses, where each clause mentions all literals $\ell_1, \ell_2, \ldots, \ell_n$. It also permits the use of the '+' operator in a natural way.

4 A Core Model for Acquiring BAE

This section introduces a CP-based core model for acquiring BAE. First, the model relies on soft constraints to represent learned Boolean expressions that mention a restricted number of arithmetic conditions taken from a large set of candidate conditions. Second, the model incorporates symmetry-breaking constraints resulting from the relaxation of arithmetic conditions. Sect. 5 will extend these symmetry-breaking constraints.

4.1 Problem Description

Given a two-dimensional table $\text{tab}[1..r, 1..c]$ of integer values, consisting of r distinct rows and c distinct columns, where column c is functionally determined by columns $1, 2, \ldots, c-1$, the problem is to come up with a constraint model to acquire an equality constraint of the form

$$\forall j \in [1, r] : \ \text{tab}[j, c] = f(\text{tab}[j, 1], \text{tab}[j, 2], \ldots, \text{tab}[j, c-1]) \tag{1}$$

i.e. a constraint that is valid for all rows of the table, where f is a Boolean-arithmetic expression mentioning $c-1$ parameters.

As we want to restrict the complexity of the acquired formulae, the expression f is limited to $n_{\text{AC}} \in \{1, 2, 3\}$ conditions chosen from $m = 53$ potential distinct BACs introduced in Sect. 3 (where a few conditions may be duplicated using different constants), and a single commutative and associative aggregation operator g selected from the set $\{\vee, \wedge, \oplus, +, \text{eq}, \text{card1}, \text{voting}\}$. As the acquisition system successively tries the different aggregation operators, we assume from now on that g is fixed. As we search for Boolean-arithmetic expressions by increasing number of BACs, we also assume that n_{AC} is fixed.

Each potential candidate BAC C_d of f (with $d \in [1, m]$) mentioning ℓ_d columns of the tab table (with $\ell_d \in [1, 3]$) and ℓ'_d coefficients (with $\ell'_d \in [0, 2]$) is represented by the term $C_d \begin{pmatrix} a_{d,1}, \ldots, a_{d,\ell_d}, \\ c_{d,1}, \ldots, c_{d,\ell'_d} \end{pmatrix}$, where:

- the variables $a_{d,1}, \ldots, a_{d,\ell_d}$ denote the indices of the distinct columns of the table $\text{tab}[1..r, 1..c]$ mentioned by condition C_d,

– the variables $c_{d,1}, \ldots, c_{d,\ell'_d}$ represents the coefficients used in the arithmetic expression of condition C_d.

The problem is to come up with a CP-based model which, given (i) a commutative and associative Boolean operator $g \in \{\vee, \wedge, \oplus, +, \text{eq}, \text{card1}, \text{voting}\}$, and (ii) a fixed number of conditions n_{AC}, extracts the subset of relevant conditions for the expression f of Constraint (1), and finds for each used conditions all its parameters, i.e. which columns and which coefficient values it uses.

Example 1. On page 8, the left-hand side of Table 2 provides a table tab[1..9, 1..4] from which we acquire the following BAE $x_4 = [(x_1 - x_2) = 2] \vee [x_3 \leq 4]$. The acquisition process is now explained in Sect. 4.2.

4.2 A CP Core Model

Notation 1 *Given a table* tab[1..r, 1..c]*, the* j*-th row of* tab[1..r, 1..c] *is called a* negative entry *if* tab[j, c] $= 0$*, and a* positive entry *otherwise.*

Selecting the BACs used in f To each potential BAC C_d (with $d \in [1, m]$) of a Boolean-arithmetic expression f, we associate a variable b_d such that:

- $b_d = -1$ means that neither condition C_d, nor condition $\neg C_d$ are used in f,
- $b_d = \ \ 0$ indicates that the condition $\neg C_d$ is used in f, i.e. C_d is negated,
- $b_d = \ \ 1$ signifies that the condition C_d occurs in f.

As f should mention n_{AC} BACs, we set up the following AMONG constraint to specify that $m - n_{\text{AC}}$ conditions must be unused:

$$\text{AMONG}\,(m - n_{\text{AC}}, \langle b_1, b_2, \ldots, b_m \rangle, -1) \tag{2}$$

Selecting the attributes used in each BAC For each potential condition $C_d \begin{pmatrix} a_{d,1}, \ldots, a_{d,\ell_d}, \\ c_{d,1}, \ldots, c_{d,\ell'_d} \end{pmatrix}$ (with $d \in [1, m]$) we set all its variables $a_{d,1}, a_{d,2}, \ldots, a_{d,\ell_d}$ to 0 when the condition C_d is not used, i.e. when $b_d = -1$. We introduce the variables $a'_{d,1}, a'_{d,2}, \ldots, a'_{d,\ell_d}$ corresponding to $a_{d,1} + 1, a_{d,2} + 1, \ldots, a_{d,\ell_d} + 1$: we use the offset $+1$ as these variables will also be used in ELEMENT constraints whose index starts at 1.

For each potential condition C_d, its variables $a'_{d,1}$, $a'_{d,2}$, $\ldots$, a'_{d,ℓ_d} should either be all distinct and greater than or equal to 2, or be all equal to 1. This is expressed by the next GLOBAL_CARDINALITY (GCC) constraint.

$$\text{GCC}\left(\begin{array}{c} \left\langle a'_{d,1}, a'_{d,2}, \ldots, a'_{d,\ell_d} \right\rangle, \\ \langle 1 : \{0, \ell_d\},\ 2 : \{0, 1\},\ \ldots\ ,\ c : \{0, 1\} \rangle \end{array}\right) \tag{3}$$

When the condition C_d is unused, i.e. $b_d = -1$, we set all its variables $a'_{d,1}, a'_{d,2}, \ldots, a'_{d,\ell_d}$ to 1 to break symmetry, i.e. to avoid enumerating over these variables:

$$\forall d \in [1, m], \forall k \in [1, \ell_d] : b_d = -1 \Leftrightarrow a'_{d,k} = 1 \tag{4}$$

To force the use of all attributes from 1 to $c-1$ of the table $\mathrm{tab}[1..r, 1..c]$ across all selected conditions, i.e. those conditions C_d, (with $d \in [1, m]$) such that $b_d \neq -1$, we set up the following GCC constraint:

$$\text{GCC}\left(\left\langle \begin{array}{c} a'_{1,1}, a'_{1,2}, \ldots, a'_{1,\ell_1}, \\ a'_{2,1}, a'_{2,2}, \ldots, a'_{2,\ell_2}, \\ \ldots\ldots\ldots\ldots\ldots\ldots \\ a'_{m,1}, a'_{m,2}, \ldots, a'_{m,\ell_m} \end{array} \right\rangle, \left\langle \begin{array}{c} 2 : o_2, \\ 3 : o_3, \\ \ldots\ldots \\ c : o_c \end{array} \right\rangle\right) \text{ with } o_i \in [1, m], \forall i \in [2, c] \quad (5)$$

Restricting the coefficients of each BAC When the BAC $C_d \begin{pmatrix} a_{d,1}, \ldots, a_{d,\ell_d}, \\ c_{d,1}, \ldots, c_{d,\ell'_d} \end{pmatrix}$ is used, i.e. $b_d \neq -1$, the coefficient variables $c_{d,1}, \ldots, c_{d,\ell'_d}$ denote the coefficients used in the arithmetic expression related to C_d. As we look for simple formulae, the initial domain of such variables is initialised to a small interval, e.g. $[-9, +9]$.

Note that we are not interested in acquiring conditions that can be substituted by `true` or `false` as they could be simplified away. For some types of conditions this would require additional constraints on the condition's coefficients, e.g. for $C_d(a_{d,1}, c_{d,1}, c_{d,2}) \equiv [(a_{d,1} \bmod c_{d,1}) = c_{d,2}]$ we post the additional constraints $b_d \neq -1 \Rightarrow c_{d,1} \geq 2$ and $b_d \neq -1 \Rightarrow c_{d,2} \in [0, c_{d,1} - 1]$:

- If $c_{d,1} = 1$ then the condition $C_d(a_{d,1}, c_{d,1}, c_{d,2})$ is either always `true`, when $c_{d,2} = 0$, or always `false` when $c_{d,2} \neq 0$.
- Otherwise, if $c_{d,1} \geq 2$ and $c_{d,2} \notin [0, c_{d,1} - 1]$ then the condition $C_d(a_{d,1}, c_{d,1}, c_{d,2})$ is always `false` as $(a_{d,1} \bmod c_{d,1}) \in [0, c_{d,1} - 1]$.

When the condition C_d is unused, we have $b_d = -1 \Rightarrow (c_{d,1} = \cdots = c_{d,\ell'_d} = 0)$ to avoid multiple solutions stemming from the coefficients of an unused condition.

How to further restrict the initial domain of the coefficient variables wrt the entries of the table $\mathrm{tab}[1..r, 1..c]$ will be explained in Section 5.3.

Setting row constraints To evaluate each condition C_d wrt the j-th row of the table $\mathrm{tab}[1..r, 1..c]$, we create the variables $v_{d,j,k}$ for the values of its k-th attributes and a variable $b_{d,j}$ for the value of C_d. This is described in the next two items:

- For each condition C_d (with $d \in [1, m]$), for each row j (with $j \in [1, r]$), and for each argument k (with $k \in [1, \ell_d]$) of condition C_d, we create a variable $v_{d,j,k}$ that gives, either the value of the k-th argument of condition C_d wrt the j-th row of the table $\mathrm{tab}[1..r, 1..c]$, or 0 if the condition C_d is unused:
  - $\forall d \in [1, m]$, $\forall j \in [1, r]$, $\forall k \in [1, \ell_d]$:
    $$\text{ELEMENT}\left(a'_{d,k},\ \langle 0, \mathrm{tab}[j, 1], \mathrm{tab}[j, 2], \ldots, \mathrm{tab}[j, c-1]\rangle,\ v_{d,j,k}\right).$$
- We also create a 0–1 variable $b_{d,j}$ which will be set to true iff condition C_d holds for the j-th row of the table $\mathrm{tab}[1..r, 1..c]$:
  - $\forall d \in [1, m]$, $\forall j \in [1, r] : b_{d,j} \Leftrightarrow C_d\left(v_{d,j,1}, v_{d,j,2}, \ldots, v_{d,j,\ell_d}\right)$.

Now, based on the aggregator g, we state a few row constraints for each used condition C_d (with $d \in [1, m]$) and wrt each row of the table tab$[1..r, 1..c]$. These row constraints are related to the type of aggregator g we are using. In this context, we distinguish the following types of aggregators I, II, and III:

I. Aggregators for which (i) positive and negative table entries have distinct row constraints and (ii) a single table entry may determine the Boolean arithmetic expression value. For instance, if g is the '$\wedge$' aggregator then on a positive entry, a condition C_d which is false (with $d \in [1, m]$) falsifies the Boolean arithmetic expression f. Aggregators '$\vee$' and '$\wedge$' belong to this class.
II. Aggregators for which (i) positive and negative table entries have distinct row constraints, and (ii) a single table entry cannot determine the Boolean arithmetic expression value. Aggregator 'eq' belongs to this class.
III. Aggregators for which (i) positive and negative table entries have the same row constraint, and (ii) a single table entry cannot determine the Boolean arithmetic expression value. Aggregators '+', '$\oplus$', 'card1', and 'voting' belong to this class.

Table 1 provides for each class in {I, II, III} of aggregator g the corresponding row constraints that determine the value of the Boolean arithmetic expression f. As mentioned earlier, for the first two classes, these row constraints depend on whether we have a positive or negative table entry; for the third class, the same constraint applies for both a positive and a negative table entry. We now explain the constraints stated in Table 1 for the first aggregator of each class.

[Case aggregator g is '$\vee$']
- For each positive row j (with $j \in [1, r]$), we post the constraint $\vee_{d=1}^{m}[b_d = b_{d,j}]$ to ensure that *at least one condition is true* so that the disjunction of conditions holds.
- For each condition C_d (with $d \in [1, m]$) and each negative row j (with $j \in [1, r]$), we post the constraint $\textsc{table}(\langle(b_d, b_{d,j})\rangle, \langle(-1, 0), (-1, 1), (0, 1), (1, 0)\rangle)$. When the condition C_d is not used, i.e. $b_d = -1$, there is no restriction on $b_{d,j}$, i.e. $b_{d,j} \in \{0, 1\}$; otherwise, *each condition must be falsified*, i.e. $b_{d,j} = 1 - b_d$, so that the corresponding disjunction of conditions is not true.

[Case aggregator g is 'eq']
- For each positive row j (with $j \in [1, r]$), we post the constraint:

$$[\textstyle\sum_{d=1}^{m}[b_d = b_{d,j}] = n_{\text{AC}}] \vee [\sum_{d=1}^{m}[b_d = \neg b_{d,j}] = n_{\text{AC}}]$$

  enforcing *either that all conditions hold or that all conditions are false.*
- For each negative row j (with $j \in [1, r]$) we post the constraint:

$$[\textstyle\sum_{d=1}^{m}[b_d = b_{d,j}] < n_{\text{AC}}] \wedge [\sum_{d=1}^{m}[b_d = \neg b_{d,j}] < n_{\text{AC}}]$$

  imposing that *at least one condition is false and at least one is true.*

[Case aggregator g is '+'] For each row j (with $j \in [1, r]$), we post the constraint tab$[j, c] = \sum_{d=1}^{m}[b_d = b_{d,j}]$ to ensure that *the appropriate number of conditions are satisfied.*

Table 1. Row constraints which are posted on a positive or a negative table entry for computing the value of a Boolean arithmetic expression f, depending on the used aggregator g of classes I, II; for class III the same row constraint is posted for all entries.

Class	g	Positive entries ($\mathrm{tab}[j,c]=1$)	Negative entries ($\mathrm{tab}[j,c]=0$)
I	'$\vee$'	$\vee_{d=1}^{m}[b_d = b_{d,j}]$	$\mathrm{TABLE}\left(\langle(b_d, b_{d,j})\rangle, \left\langle \begin{smallmatrix}(-1,0),(-1,1),\\ (\ 0,1),(\ 1,0)\end{smallmatrix} \right\rangle\right)$
	'$\wedge$'	$\mathrm{TABLE}\left(\langle(b_d, b_{d,j})\rangle, \left\langle \begin{smallmatrix}(-1,0),(-1,1),\\ (\ 0,0),(\ 1,1)\end{smallmatrix} \right\rangle\right)$	$\vee_{d=1}^{m}[b_d = \neg b_{d,j}]$
II	'eq'	$\bigvee\left(\begin{smallmatrix}\sum_{d=1}^{m}[b_d = \ \ b_{d,j}] = n_{\mathrm{AC}},\\ \sum_{d=1}^{m}[b_d = \neg b_{d,j}] = n_{\mathrm{AC}}\end{smallmatrix}\right)$	$\bigwedge\left(\begin{smallmatrix}\sum_{d=1}^{m}[b_d = \ \ b_{d,j}] < n_{\mathrm{AC}},\\ \sum_{d=1}^{m}[b_d = \neg b_{d,j}] < n_{\mathrm{AC}}\end{smallmatrix}\right)$
III	'+'	$\mathrm{tab}[j,c] = \sum_{d=1}^{m}[b_d = b_{d,j}]$	
	'$\oplus$'	$\mathrm{tab}[j,c] = (\sum_{d=1}^{m}[b_d = b_{d,j}]) \bmod 2$	
	'card1'	$\mathrm{tab}[j,c] = [(\sum_{d=1}^{m}[b_d = b_{d,j}]) = 1]$	
	'voting'	$\mathrm{tab}[j,c] = [2 \cdot (\sum_{d=1}^{m}[b_d = b_{d,j}]) > n_{\mathrm{AC}}]$	

Table 2. Illustrating the core model on the table $\mathrm{tab}[1..9, 1..4]$ (with columns x_1, x_2, x_3, x_4) for acquiring a Boolean-arithmetic expression explaining x_4 wrt x_1, x_2, x_3 using the '$\vee$' aggregator with two conditions C_1 and C_2 selected from the following potential candidate conditions $C_1 : x_i - x_j = cst$, $C_2 : x_i \leq cst$ and $C_3 : x_i = x_j$.

	j	table tab x_1 x_2 x_3 x_4	$C_1 = [(x_1 - x_2) = 2]$ b_1=1 $a'_{1,1}$=2 $a'_{1,2}$=3			$C_2 = [x_3 \leq 4]$ b_2=1 $a'_{2,1}$=4		$C_3 = [x_{k_1} = x_{k_2}]$ b_3=−1 $a'_{3,1}$=1 $a'_{3,2}$=1			row constraint satisfaction
			$b_{1,j}$	$v_{1,j,1}$	$v_{1,j,2}$	$b_{2,j}$	$v_{2,j,1}$	$b_{3,j}$	$v_{3,j,1}$	$v_{3,j,2}$	
positive entries	1	4 2 5 **1**	1	4	2	0	5	1	0	0	`true`
	2	3 4 4 **1**	0	3	4	1	4	1	0	0	`true`
	3	1 1 3 **1**	0	1	1	1	3	1	0	0	`true`
	4	3 1 5 **1**	1	3	1	0	5	1	0	0	`true`
	5	4 1 2 **1**	0	4	1	1	2	1	0	0	`true`
negative entries	6	2 4 5 **0**	0	2	4	0	5	1	0	0	`true`
	7	4 1 5 **0**	0	4	1	0	5	1	0	0	`true`
	8	4 3 5 **0**	0	4	3	0	5	1	0	0	`true`
	9	3 5 5 **0**	0	3	5	0	5	1	0	0	`true`

Example 2 (Continuation of Example 1). Table 2 summarises the acquisition of the BAE $x_4 = [(x_1 - x_2) = 2] \vee [x_3 \leq 4]$ from the table $\mathrm{tab}[1..9, 1..4]$: it provides the main variables introduced by the core model. First, note that only conditions C_1 and C_2 are selected, as $b_3 = -1$. For the first positive entry (i.e. $j = 1$) and the first negative entry (i.e. $j = 6$), we now show that the corresponding row constraints described in Table 1 are true:

– As row 1 is a positive entry, we post the constraint $[b_1 = b_{1,1}] \vee [b_2 = b_{2,1}] \vee [b_3 = b_{3,1}]$ which is true as $b_1 = b_{1,1}$ holds.

– As row 6 is a negative entry, we post the constraint $\text{TABLE}(\langle(b_d, b_{d,6})\rangle, \mathcal{T})$, with $\mathcal{T} = \langle(-1,0),(-1,1),(0,1),(1,0)\rangle$, for each condition C_d ($d \in \{1,2,3\}$). All three constraints hold for the sixth row.

5 Enhancing the Core Model

5.1 Linking the Number of Conditions, their Arity, and the Number of Attributes

We introduce the following constraints to explicitly restrict the potential combinations of unary, binary and ternary conditions to consider.

Notation 2 *Within the expression f formed by n_{AC} conditions, let $n_{\text{AC},k}$ denote the number of conditions mentioning k attributes.*

Since we restrict the Boolean-arithmetic expression f to at most three conditions, we state the constraints $n_{\text{AC}} = \sum_{k=1}^{3} n_{\text{AC},k}$ and $\sum_{k=1}^{3} k \cdot n_{\text{AC},k} = \sum_{i=2}^{c} o_i$, where o_i is the number of occurrences of value i in the variables $a'_{d,1}, a'_{d,2}, \ldots, a'_{d,\ell_d}$, as stated by the GCC constraint (3) of the core model. We now state the lower and upper bounds on the number of distinct attributes $c-1$ appearing in the expression f wrt $n_{\text{AC},k}$ (with $k \in [1,3]$):

- $c - 1 \geq \max_{k=1}^{3}(k \cdot \min(1, n_{\text{AC},k}))$,
- $c - 1 \leq \sum_{k=1}^{3}(k \cdot n_{\text{AC},k})$.

5.2 Symmetry Breaking

As a formula may involve commutative arithmetic operators whose arguments can be interchanged, and mention several occurrences of the same condition which can be swapped, we show how to restrict the search space for formulae.

Commutative arithmetic operators For each BAC $C_d\begin{pmatrix} a_{d,1},\ a_{d,2}, \\ c_{d,1}, \ldots, c_{d,\ell'_d} \end{pmatrix}$ (with $d \in [1,m]$) mentioning two attributes $a_{d,1}$ and $a_{d,2}$, as well as a commutative arithmetic operator such as $+$, min, or max, we order its arguments only when the condition is used, by posting a constraint of the form $b_d \neq -1 \Rightarrow a'_{d,1} < a'_{d,2}$ on its variables $a'_{d,1}$ and $a'_{d,2}$.

Conditions mentioning the same comparison and arithmetic operators In case a same condition would occur several times in the expression f, positively or negatively, or with different attributes, we post symmetry-breaking constraints to prevent generating equivalent subexpressions. We order the list of potential BACs $C_1, C_2, \ldots, C_m$ so that conditions that use the same comparison operator $\leq, =, \geq, \in$, as well as the same arithmetic operator $+$, $-$, $\times$, $\lfloor\text{-}\rfloor$, $\lceil\text{-}\rceil$, mod, min, max are located consecutively. For each pair of consecutive conditions $C_d\begin{pmatrix} a_{d,1}, \ldots, a_{d,\ell_d}, \\ c_{d,1}, \ldots, c_{d,\ell'_d} \end{pmatrix}$, $C_{d+1}\begin{pmatrix} a_{d+1,1}, \ldots, a_{d+1,\ell_{d+1}}, \\ c_{d+1,1}, \ldots, c_{d+1,\ell'_{d+1}} \end{pmatrix}$ (with $d \in [1, m-1]$)

using the same comparison and arithmetic operators, we enforce the following symmetry-breaking constraint.

The idea is to impose a strict lexicographic ordering constraint (SLOC) between the variables of such consecutive conditions C_d and C_{d+1}. However, we need to consider the cases where these conditions are unused ($b_d = -1$, $b_{d+1} = -1$), negated ($b_d = 0$, $b_{d+1} = 0$) or positively used ($b_d = 1$, $b_{d+1} = 1$). We use the following idea to adapt the SLOC to our context: a SLOC can be described as a finite automaton whose input alphabet consists of letters that pairwise compare the k-th components of two vectors [6]. We compare the vectors $\overrightarrow{U} = (b_d, a'_{d,1}, a'_{d,2}, \ldots, a'_{d,\ell_d}) = (u_1, u_2, \ldots, u_{\ell_d+1})$ and $\overrightarrow{V} = (b_{d+1}, a'_{d+1,1}, a'_{d+1,2}, \ldots, a'_{d+1,\ell_{d+1}}) = (v_1, v_2, \ldots, v_{\ell_d+1})$. Recall from Sect. 4.2 that (i) depending on whether condition C_d is unused, negated or used positively, b_d will be set to -1, 0 or 1 respectively, and that (ii) the variables $a'_{d,1}, a'_{d,2}, \ldots, a'_{d,\ell_d}$ are all in the range $[2, c]$ as we applied the offset $+1$. By pairwise comparing the k-th components of vectors $\overrightarrow{U}$ and $\overrightarrow{V}$ (with $k \in [1, \ell_d + 1]$) we create the following vector $\overrightarrow{W} = (w_1, w_2, \ldots w_{\ell_d+1})$, where each component is defined by one of the nine letters $0, 1, \ldots, 8$ described in Table 3.

We then force the components of vector $\overrightarrow{W}$ to be accepted by the finite automaton given in Fig. 1. The three accepting states labelled by **n**, **o**, and **t** respectively correspond to the fact that (i) **n**one of the conditions C_d, C_{d+1} is used, (ii) **o**nly the first condition C_d is used, and (iii) the **t**wo conditions C_d, C_{d+1} are both used. The outgoing transitions from state $\boldsymbol{\epsilon}$ to states $\begin{matrix}\mathbf{t}\\ \neq\end{matrix}$

Table 3. Definition of the input letters of the finite automaton depicted in Part (A) of Fig. 1 used for breaking symmetry between two consecutive conditions

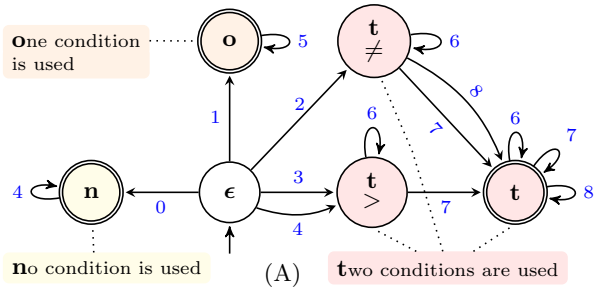
Input letter	Corresponding condition	Comment
$w_k = 0$	$u_k = -1 \wedge v_k = -1$	Both conditions are unused.
$w_k = 1$	$(u_k = 0 \vee u_k = 1) \wedge v_k = -1$	Only one condition is used.
$w_k = 2$	$u_k = 1 \wedge v_k = 0$	The 1st condition is used positively, and the negation of the 2nd condition is used.
$w_k = 3$	$u_k = 0 \wedge v_k = 0$	The negation of the 1st condition is used, and the negation of the 2nd condition is used.
$w_k = 4$	$u_k = 1 \wedge v_k = 1$	$k = 1$: both conditions are used positively, $k > 1$: attributes of both conditions are unused.
$w_k = 5$	$u_k > 1 \wedge v_k = 1$	u_k is an attribute of the 1st condition, and v_k an unused attribute of the 2nd condition, as the 2nd condition is unused.
$w_k = 6$	$u_k > 1 \wedge v_k > 1 \wedge u_k = v_k$	u_k and v_k are attributes of the two used conditions, such that $u_k = v_k$.
$w_k = 7$	$u_k > 1 \wedge v_k > 1 \wedge u_k > v_k$	u_k and v_k are attributes of the two used conditions, such that $u_k > v_k$.
$w_k = 8$	$u_k > 1 \wedge v_k > 1 \wedge u_k < v_k$	u_k and v_k are attributes of the two used conditions, such that $u_k < v_k$.

and $\overset{\mathbf{t}}{>}$ enforce that, when using a condition and its negated form, the negated form is located in the second position. The two outgoing transitions of state $\overset{\mathbf{t}}{>}$ ensure that the arguments of the first used condition are lexicographically strictly greater than the arguments of the second condition, while the two outgoing transitions of state $\overset{\mathbf{t}}{\neq}$ force the two conditions to not use the same arguments.

5.3 Pre-computing the Combinations of Possible Values of the Coefficients of a Condition

Most BACs $C_d \begin{pmatrix} a_{d,1}, \ldots, a_{d,\ell_d}, \\ c_{d,1}, \ldots, c_{d,\ell'_d} \end{pmatrix}$ can be presented as a comparison of the form $C'_d(P) \lozenge c_{d,\ell'_d}$ (with $\lozenge \in \{\leq, =, \geq\}$), where $C'_d(P)$ is an arithmetic expression parameterised by $P = \begin{pmatrix} a'_{d,1}, \ldots, a'_{d,\ell_d}, \\ c_{d,1}, \ldots, c_{d,\ell'_d-1} \end{pmatrix}$. Such BACs in a Boolean formula f must not be equivalent to `true` or `false`, as otherwise they could be simplified away from f. We also want to avoid generating a condition involving an inequality when an equality would suffice. For this purpose we proceed as follows.

- For each possible combination of values p of parameter P wrt the potential values of $a'_{d,1}, \ldots, a'_{d,\ell_d}, c_{d,1}, \ldots, c_{d,\ell'_d-1}$, we compute the feasible values of $C'_d(p)$ wrt all the table entries of tab$[1..r, 1..c]$. We denote by $\mathcal{V}_{d,p}$ such sets.
- Then, depending on the comparison operator $\lozenge$ used in condition C_d, we derive for each combination of values p of parameter P, the set of values of coefficient c_{d,ℓ'_d} which does not make condition C_d always true or always false. We denote such sets as $\mathcal{V}^{\lozenge}_{d,p}$. They are obtained from the sets $\mathcal{V}_{d,p}$ in the following way.
  - [$\lozenge$ is '=']: when the coefficient c_{d,ℓ'_d} is assigned a value outside $\mathcal{V}_{d,p}$ the condition C_d would always be false; if the cardinality of $\mathcal{V}_{d,p}$ is 1 then



(A)

$\vec{U}$	$\vec{V}$	$\vec{W}$	Condition(s)
$(-1,1)$	$(-1,1)$	$(0,4)$	none
$(\ 0,2)$	$(-1,1)$	$(1,5)$	$\neg C_d(1)$
$(\ 0,3)$	$(-1,1)$	$(1,5)$	$\neg C_d(2)$
$(\ 0,3)$	$(\ 0,2)$	$(2,7)$	$\neg C_d(2)\ \neg C_{d+1}(1)$
$(\ 1,2)$	$(-1,1)$	$(1,5)$	$C_d(1)$
$(\ 1,3)$	$(-1,1)$	$(1,5)$	$C_d(2)$
$(\ 1,3)$	$(\ 0,2)$	$(2,7)$	$C_d(2)\ \neg C_{d+1}(1)$
$(\ 1,3)$	$(\ 1,2)$	$(4,7)$	$C_d(2)\ C_{d+1}(1)$

(B)

Fig. 1. (A) Automaton for breaking symmetries between two consecutive conditions C_d, C_{d+1} sharing the same comparison and arithmetic operators, where accepting states are denoted by a double circle; (B) Examples of vectors $\vec{U}$, $\vec{V}$, $\vec{W}$ and corresponding used conditions with their arguments (each condition mentions one single attribute).

Table 4. Example table for pre-computing the possible values of the coefficients for conditions C_1 and C_2

x_1	x_2	x_c	$[x_1 - x_2]$	$[x_2 - x_1]$	$[x_1 \bmod 3]$	$[x_2 \bmod 3]$
1	2	0	-1	1	1	2
2	1	0	1	-1	2	1
1	2	1	-1	1	1	2
1	3	1	-2	2	1	0
1	4	1	-3	3	1	1

$\mathcal{V}^{\Diamond}_{d,p} = \emptyset$ (i.e. if there is only one value the condition would always be true), otherwise $\mathcal{V}^{\Diamond}_{d,p} = \mathcal{V}_{d,p}$.

- [$\Diamond$ is '$\leq$' or '$\geq$']: let α and ω respectively be the smallest and the largest value of the set $\mathcal{V}_{d,p}$; then $\mathcal{V}^{\Diamond}_{d,p} = \mathcal{V}_{d,p} \setminus \{\alpha, \omega\}$. The intuition for $\Diamond =$'$\leq$' is as follows: if we keep α then $\Diamond =$'$\leq$' is equivalent to $\Diamond =$'=' ; if we keep ω the condition will always be true. For '$\geq$' the intuition is symmetrical.

– We may further reduce the set $\mathcal{V}^{\Diamond}_{d,p}$ by considering the aggregator g. First, for each possible combination of values p of parameter P, we compute the feasible values of $C'_d(p)$ wrt all the positive (resp. negative) table entries of $\text{tab}[1..r, 1..c]$. We denote by $\mathcal{V}^{\text{pos}}_{d,p}$ (resp. $\mathcal{V}^{\text{neg}}_{d,p}$) such sets. From these sets, we compute the further restricted set $\mathcal{V}^{\Diamond,g}_{d,p}$ as follows:
  - [g is '$\wedge$']: if $\Diamond$ is '=' then $\mathcal{V}^{\Diamond,g}_{d,p} = \mathcal{V}^{\text{pos}}_{d,p}$ else $\mathcal{V}^{\Diamond,g}_{d,p} = \mathcal{V}^{\text{pos}}_{d,p} \cap \mathcal{V}^{\Diamond}_{d,p}$,
  - [$g \in \{$' $\vee$ ',' + '$\}$]: $\mathcal{V}^{\Diamond,g}_{d,p} = \mathcal{V}^{\Diamond}_{d,p} \setminus \mathcal{V}^{\text{neg}}_{d,p}$,
  - [$g \notin \{$' $\wedge$ ',' $\vee$ ',' + '$\}$]:$\mathcal{V}^{\Diamond,g}_{d,p} = \mathcal{V}^{\Diamond}_{d,p}$.
– Finally, we set up the table constraint $\text{TABLE}\left(\left\langle \begin{matrix} a'_{d,1}, \ldots, a'_{d,\ell_d}, \\ c_{d,1}, \ldots, c_{d,\ell'_d} \end{matrix} \right\rangle, \mathcal{S}\right)$ where $\mathcal{S}$ corresponds to the union of cartesian products $\cup_{p \in P}(p \times \mathcal{V}^{\Diamond,g}_{d,p})$.

Example 3. To illustrate the process, consider Table 4. There are two input columns 1 and 2 and the output column c. Consider the two conditions $C_1 = [a_{1,1} - a_{1,2} = c_{1,1}]$ and $C_2 = [a_{2,1} \bmod c_{2,1} \geq c_{2,2}]$.

• For C_1 we have only two options for $p = \{a_{1,1}, a_{1,2}\}$, namely:

1) $p = \{1,2\}$: $\begin{cases} \mathcal{V}_{1,p} = \{-3,-2,-1,1\},\ \mathcal{V}^{=}_{1,p} = \mathcal{V}_{1,p}, \\ \mathcal{V}^{=,\text{'}\wedge\text{'}}_{1,p} = \{-3,-2,-1\},\ \mathcal{V}^{=,\text{'}\vee\text{'}}_{1,p} = \mathcal{V}^{=}_{1,p} \setminus \{-1,1\} = \{-3,-2\}. \end{cases}$

2) $p = \{2,1\}$: $\begin{cases} \mathcal{V}_{1,p} = \{-1,1,2,3\},\ \mathcal{V}^{=}_{1,p} = \mathcal{V}_{1,p}, \\ \mathcal{V}^{=,\text{'}\wedge\text{'}}_{1,p} = \{1,2,3\}, \quad \mathcal{V}^{=,\text{'}\vee\text{'}}_{1,p} = \mathcal{V}^{=}_{1,p} \setminus \{-1,1\} = \{2,3\}. \end{cases}$

• For C_2 we need to enumerate on $c_{2,1}$. Wlog, we only consider the case $c_{2,1} = 3$. In this context, the options for $p = \{a_{2,1}, c_{2,1}\}$ are:

1) $p = \{1,3\}$: $\mathcal{V}_{2,p} = \{1,2\},\ \alpha = 1,\ \omega = 2,\ \mathcal{V}^{\geq}_{2,p} = \mathcal{V}_{2,p} \setminus \alpha, \omega = \emptyset$, i.e. this set of options for this condition is not considered any further.

2) $p = \{2,3\}$: $\begin{cases} \mathcal{V}_{2,p} = \{0,1,2\},\ \alpha = 0, \omega = 2,\ \mathcal{V}^{\geq}_{1,p} = \mathcal{V}_{2,p} \setminus \{\alpha, \omega\} = \{1\}, \\ \mathcal{V}^{\text{pos}}_{2,p} = \{1\},\ \mathcal{V}^{\text{neg}}_{2,p} = \{1,2\}, \\ \mathcal{V}^{\geq,\text{'}\wedge\text{'}}_{2,p} = \mathcal{V}^{\text{pos}}_{2,p} \cap \mathcal{V}^{\geq}_{2,p} = \{1\}, \mathcal{V}^{\geq,\text{'}\vee\text{'}}_{2,p} = \mathcal{V}^{\geq}_{2,p} \setminus \mathcal{V}^{\text{neg}}_{2,p} = \emptyset. \end{cases}$

6 Evaluation

The CP core model introduced in Sect. 4 and its extension described in Sect. 5 were evaluated in the context of the search of conjectures on sharp bounds on characteristics of several combinatorial objects, which we now describe.

- **digraph (without isolated vertex):** a set of vertices $\mathcal{V}$ and a set of ordered pairs of vertices $\mathcal{A}$ with the restriction that each vertex of $\mathcal{V}$ occurs in at least one pair of $\mathcal{A}$ [5].
- **rooted tree:** a connected acyclic undirected graph where a vertex is designed as the "root" of the tree [18].
- **rooted forest:** a disjoint union of rooted trees [18]; we also consider a variant, **rooted forest2**, where all rooted trees have at least two vertices.
- **partition:** a partition of a set $\mathcal{S}$ is a collection of possibly empty subsets of $\mathcal{S}$ such that every element of $\mathcal{S}$ is in exactly one of the subsets of the collection. The use of a partition was motivated the by fact that a partition can be interpreted as a solution to the conjunction of the NVALUE (i.e. the number of partition subsets, see [26]) and the BALANCE (i.e. the difference between the cardinalities of the largest and smallest subsets of the partition, see [7]) constraints. Motivated by the extension of the BALANCE constraint, i.e. ALL_BALANCE [9], we also consider a version of partition named **partition0** where all subsets of $\mathcal{S}$ are non-empty.
- **stretch:** a solution of a stretch constraint on 0-1 variables, where a subsequence of 1 immediately preceded and followed by a 0 is called a *stretch* [27]; we also consider the variant named **cyclic stretch** where, when the sequence begins and terminates by 1, those two 1 belong to the same stretch.

Table 5 shows for each combinatorial object some characteristics we consider, and some conjectures found using the Boolean model described in this paper. Those conjectures are equalities which express (i) either the value of a characteristic when another characteristic is reaching its sharp bound, (ii) either a sharp bound formulated wrt other characteristics [8]. We evaluate the CP models of Sect. 4 and 5 wrt to the following two aspects:

1. The computing time spent by the core model of Sect. 4 (i.e. C. Model) and by its enhanced version of Sect. 5 (i.e. E. Model) wrt (i) the type of aggregator used in a BAE, and wrt (ii) the kind of combinatorial object. For this aspect, we test 3706 examples of acquired BAE on a MacBookPro with a 2.6 GHz Core i7 and 16Gb of memory using SICStus 4.6.0. Table 6 shows that the E. model acquires a BAE with, on average, 73% less time than the C. Model. Additional tests showed that using just the constraints from Sect. 5.1 increases the speed of the C. Model by $\approx 5\%$, just the constraints from Sect. 5.2 - by $\approx 63\%$ and just the constraints from Sect. 5.3 - by $\approx 48\%$.
2. Using the enhanced model together with the model acquiring polynomial of [8] (i.e. the EP. Model), Table 7 gives (i) the number of Boolean formulae found, i.e. 642 (sum of second columns), replacing a formula with a polynomial, and (ii) the number of new Boolean formulae, i.e. 56 (sum of

Table 5. Examples of characteristics (char.) of combinatorial objects and corresponding conjectures: (i) $c, s, oc, \underline{c}$ and $\overline{s}$: *number of connected components (cc), strongly connected components (scc), connected components with at least two vertices, size of the smallest cc* and *size of the largest scc* of a **digraph**; (ii) c_0: *denote* 0 *if all the cc have same maximal size, and* $\underline{c}$ *otherwise*, for a **digraph**; (iii) v and f: *number of vertices* and *leaves* in a **rooted tree**; (iv) $\overline{d}$: *largest degree of a parent node* in a **rooted tree** or a **rooted forest**; (v) $\underline{p}$ and $\underline{t}$: *minimum depth* and *size of the smallest tree* in a **rooted forest**; (vi) n, $nval$, and $\underline{m}$: *number of elements*, *number of subsets*, and *cardinality of the smallest subset* in a **partition**; (vii) sr, dr, and $\underline{dm}$: *difference between the number of elements of the largest and smallest stretches*, *difference between the maximum and minimum distance of consecutive stretches*, and *minimum distance between consecutive stretches* in **stretch**; $(viii)$ n, ng, and osc: *number of elements, total number of stretches, and number of stretches which have more than one element when the number of element of the largest stretch is maximal* in **cyclic stretch**.

Combinatorial object	Number of char.	Some of the used char.	Examples of discovered conjectures
digraph	20	$c, \underline{c}, s, oc$	$c = 1 + [\neg(s \leq \underline{c} \wedge oc \leq 1)]$
digraph	20	$c_0, c, s, \overline{s}$	$c_0 = [\neg\,\text{voting}(c = s, c = 1, \min(c, \overline{s}) = 1)]$
rooted tree	6	$\overline{d}, v, f$	$\overline{d} = (v = f \vee v = 1\,?\,0 : f)$
rooted forest	11	$\underline{p}, \overline{d}, \underline{t}$	$\overline{d} = 2 - ([\underline{p} = 0] + [(\underline{t} - \underline{p}) = 1])$
partition	14	$n, nval, \underline{m}$	$nval = 1 + [2 \cdot \underline{m} \leq n]$
stretch	26	$sr, dr, \underline{dm}$	$\underline{dm} = [(sr + dr) \geq 1]$
cyclic stretch	26	osc, n, ng	$osc = [\neg\,\text{card1}(n = 2 \cdot ng, n \cdot ng = 3, n \cdot ng \leq 3)]$

Table 6. Computing time for the Core and the Enhanced models wrt aggregators (left side) and combinatorial objects (right side)

g	n_{AC}	Number of conjectures	Average time per conjecture C. Model	Average time per conjecture E. Model	Combinatorial object	Number of conjectures	Average time per conjecture C. Model	Average time per conjecture E. Model
$\wedge$	1	2311	0.36s	0.35s	**digraph**	546	43s	10s
$\wedge$	2	341	25.5s	11s	**rooted tree**	56	1.5s	1s
$\wedge$	3	5	2591s	542s	**rooted forest**	229	3.4s	1.3s
$\vee$	2	190	18.9s	6.1s	**rooted forest2**	586	18.2s	5.9s
$\vee$	3	1	3062s	292s	**partition**	24	2.7s	1.7s
eq	2	143	35.7s	14.7s	**partition0**	24	0.8s	0.5s
eq	3	47	309s	50s	**stretch**	1059	10s	2.7s
$+$	2	662	2s	1.3s	**cyclic stretch**	1182	6.4s	1.8s
card1	3	2	1003s	27.5s				
voting	3	4	345s	66s	Total	3706	14.4h	3.9h

third columns) discovered compared to the model described in [8], which only looked for formulae with polynomials and arithmetic functions involving two polynomials (i.e. the P. Model).

7 Related Work

Learning purely Boolean expressions from data is widely reported in the literature. A significant number of papers explore the acquisition of relevant features, often called the "*relevant features problem*" (RFP). Blum formalises the RFP

in [10], and provides a survey of various algorithms in [11]. The RFP can be applied to features that are Boolean, integer or continuous, each of which requires its own approach [15, chapter 1.2]. Some of the works focusing on purely Boolean RFP are described in [25,23,13]. In [24], Mutlu and Oghaz provide a taxonomy of Boolean and non-Boolean feature extraction techniques applied to graphs. Other works present the acquisition of Boolean expressions as a part of the Boolean rules extraction process for classification problems using SAT [31] or neural networks [22]. Lastly, there are papers [16,14] focusing on the construction and the simplification of Boolean functions. The acquisition of Boolean-arithmetic expressions is often used in the context of classification problems. Random forest [17], decision trees [3,4,19], Bayesian rule lists [30], fuzzy association rules [2] and rough sets [20] approaches are used. Most of the work considers the acquisition of relatively simple Boolean-arithmetic expressions of the type "*attribute has a value of*". The SEEN system [19] extracts more complex Boolean-arithmetic expressions that contain the $+$, $\times$ and $/$ arithmetic operators: it calls such domain "*logical-arithmetic expression mining*".

Beyond the domain of Boolean formulae, synthesising formulae from data [1] mostly relies on a generate and test approach to produce candidate formulae of increasing complexity for a fixed grammar. In our context, applying techniques that minimise an error function produces complicated formulae that are not verified wrt all input data. In [8] we compared our approach to methods used for symbolic regression such as GPlearn and ffx: GPlearn generally found no formulae, while ffx discovered formulae with a large number of terms.

8 Conclusion

The paper presents a CP model for learning BAE that can cope with a variety of expressions involving the most common Boolean aggregators and arithmetic operators. In the context of sharp bound acquisition, this complements the model introduced in [8] for learning equations whose right-hand sides are polynomials. The model is relevant not only in the context of digraphs, but also for other combinatorial objects such as rooted trees or partitions.

Table 7. Contribution of the EP. Model that both acquires Boolean formulae (BF) and polynomials compared to searching only formulae with polynomials with the P. Model

Combinatorial object	Number of (BF) which are: found	replacing polynomials	new
digraph	164	118	46
rooted tree	26	26	0
rooted forest	91	86	5
rooted forest2	149	145	4

Combinatorial object	Number of (BF) which are: found	replacing polynomials	new
partition	20	20	0
partition0	10	10	0
stretch	93	93	0
cyclic stretch	145	144	1

References

1. Alur, R., Singh, R., Fisman, D., Solar-Lezama, A.: Search-based program synthesis. Commun. ACM **61**(12), 84–93 (2018). https://doi.org/10.1145/3208071, https://doi.org/10.1145/3208071
2. Au, W.H., Chan, K.C.: Mining fuzzy association rules in a bank-account database. IEEE Transactions on Fuzzy Systems **11**(2), 238–248 (2003)
3. Aung, M.H., Lisboa, P., Etchells, T.A., Testa, A.C., Calster, B.V., Huffel, S.V., Valentin, L., Timmerman, D.: Comparing analytical decision support models through boolean rule extraction: A case study of ovarian tumour malignancy. In: International Symposium on Neural Networks. pp. 1177–1186. Springer (2007)
4. Barbareschi, M., Barone, S., Mazzocca, N.: Advancing synthesis of decision tree-based multiple classifier systems: an approximate computing case study. Knowledge and Information Systems **63**(6), 1577–1596 (2021)
5. Beldiceanu, N.: Global constraints as graph properties on a structured network of elementary constraints of the same type. In: Dechter, R. (ed.) Principles and Practice of Constraint Programming - CP 2000, 6th International Conference, Singapore, September 18-21, 2000, Proceedings. Lecture Notes in Computer Science, vol. 1894, pp. 52–66. Springer (2000). https://doi.org/10.1007/3-540-45349-0_6, https://doi.org/10.1007/3-540-45349-0_6
6. Beldiceanu, N., Carlsson, M., Petit, T.: Deriving filtering algorithms from constraint checkers. In: Wallace, M. (ed.) Principles and Practice of Constraint Programming - CP 2004, 10th International Conference, CP 2004, Toronto, Canada, September 27 - October 1, 2004, Proceedings. Lecture Notes in Computer Science, vol. 3258, pp. 107–122. Springer (2004). https://doi.org/10.1007/978-3-540-30201-8_11, https://doi.org/10.1007/978-3-540-30201-8_11
7. Beldiceanu, N., Carlsson, M., Rampon, J.X.: Global Constraint Catalog, 2nd Edition (revision a). Tech. Rep. T2012-03, Swedish Institute of Computer Science (2012), available at http://ri.diva-portal.org/smash/get/diva2:1043063/FULLTEXT01.pdf
8. Beldiceanu, N., Cheukam-Ngouonou, J., Douence, R., Gindullin, R., Quimper, C.G.: Acquiring maps of interrelated conjectures on sharp bounds. In: 28th International Conference on Principles and Practice of Constraint Programming (CP 2022). Schloss Dagstuhl-Leibniz-Zentrum für Informatik (2022)
9. Bessière, C., Hebrard, E., Katsirelos, G., Kızıltan, Z., Picard-Cantin, É., Quimper, C.G., Walsh, T.: The balance constraint family. In: O'Sullivan, B. (ed.) Principles and Practice of Constraint Programming. pp. 174–189. Springer International Publishing, Cham (2014)
10. Blum, A.: Relevant examples and relevant features: Thoughts from computational learning theory. In: AAAI Fall Symposium on 'Relevance. vol. 5, p. 1 (1994)
11. Blum, A.L., Langley, P.: Selection of relevant features and examples in machine learning. Artificial intelligence **97**(1-2), 245–271 (1997)
12. Brence, J., Todorovski, L., Džeroski, S.: Probabilistic grammars for equation discovery. Knowledge-Based Systems **224** (2021), https://doi.org/10.1016/j.knosys.2021.107077
13. Forman, G., Kirshenbaum, E.: Extremely fast text feature extraction for classification and indexing. In: Proceedings of the 17th ACM conference on Information and knowledge management. pp. 1221–1230 (2008)
14. Golia, P., Slivovsky, F., Roy, S., Meel, K.S.: Engineering an efficient boolean functional synthesis engine. In: 2021 IEEE/ACM International Conference On Computer Aided Design (ICCAD). pp. 1–9. IEEE (2021)

15. Guyon, I., Elisseeff, A.: An introduction to feature extraction. In: Feature extraction, pp. 1–25. Springer (2006)
16. Jakobovic, D., Picek, S., Martins, M.S., Wagner, M.: Toward more efficient heuristic construction of boolean functions. Applied Soft Computing **107**, 107327 (2021)
17. Jun, S., Lee, S., Chun, H.: Learning dispatching rules using random forest in flexible job shop scheduling problems. International Journal of Production Research **57**(10), 3290–3310 (2019)
18. Knuth, D.: Art of Computer Programming, Volume 4, Generating All Trees, pp. 461-462. Addison-Wesley (2006)
19. Kosman, E., Kolchinsky, I., Schuster, A.: Mining logical arithmetic expressions from proper representations. In: Proceedings of the 2022 SIAM International Conference on Data Mining (SDM). pp. 621–629. SIAM (2022)
20. Lambert-Torres, G.: Application of rough sets in power system control center data mining. In: 2002 IEEE Power Engineering Society Winter Meeting. Conference Proceedings (Cat. No. 02CH37309). vol. 1, pp. 627–631. IEEE (2002)
21. Larson, C.E., Van Cleemput, N.: Automated conjecturing iii. Annals of Mathematics and Artificial Intelligence **81**(3), 315–327 (2017)
22. Mereani, F., Howe, J.M.: Exact and approximate rule extraction from neural networks with boolean features. In: Proceedings of the 11th International Joint Conference on Computational Intelligence. pp. 424–433. SCITEPRESS (2019)
23. Mossel, E., O'Donnell, R., Servedio, R.A.: Learning functions of k relevant variables. Journal of Computer and System Sciences **69**(3), 421–434 (2004)
24. Mutlu, E.C., Oghaz, T.A.: Review on graph feature learning and feature extraction techniques for link prediction. arXiv preprint arXiv:1901.03425 (2019)
25. Nguifo, E.M., Njiwoua, P.: Using lattice-based framework as a tool for feature extraction. In: Feature Extraction, Construction and Selection, pp. 205–218. Springer (1998)
26. Pachet, F., Roy, P.: Automatic generation of music programs. In: Jaffar, J. (ed.) Principles and Practice of Constraint Programming - CP'99, 5th International Conference, Alexandria, Virginia, USA, October 11-14, 1999, Proceedings. Lecture Notes in Computer Science, vol. 1713, pp. 331–345. Springer (1999). https://doi.org/10.1007/978-3-540-48085-3_24, https://doi.org/10.1007/978-3-540-48085-3_24
27. Pesant, G.: A filtering algorithm for the stretch constraint. In: Walsh, T. (ed.) Principles and Practice of Constraint Programming — CP 2001. pp. 183–195. Springer Berlin Heidelberg, Berlin, Heidelberg (2001)
28. Schelldorfer, J., Wuthrich, M.V.: Nesting classical actuarial models into neural networks. Available at SSRN 3320525 (2019)
29. Todorovski, L.: Equation discovery. In: Sammut, C., Webb, G.I. (eds.) Encyclopedia of Machine Learning, pp. 327–330. Springer US, Boston, MA (2010)
30. Yang, H., Rudin, C., Seltzer, M.: Scalable bayesian rule lists. In: International conference on machine learning. pp. 3921–3930. PMLR (2017)
31. Yu, J., Ignatiev, A., Stuckey, P.J., Le Bodic, P.: Computing optimal decision sets with sat. In: International Conference on Principles and Practice of Constraint Programming. pp. 952–970. Springer (2020)