A Boolean Formula Seeker in the Context of Acquiring Maps of Interrelated Conjectures on Sharp Bounds

Ramiz Gindullin
IMT Atlantique, LS2N (TASC), Nantes, France

Abstract

A component of Bound Seeker [1] is Boolean formula seeker, a part devoted to search for arithmetic-Boolean formulae. Here, a arithmetic-Boolean formula involves n arithmetic conditions linked by a single commutative logical operator or by a sum. Part of the originality of the Boolean formula seeker is that it was synthesised by a constraint program. This extended abstract includes (*i*) the type of arithmetic-Boolean formulae we consider, (*ii*) the importance of allowing arithmetic-Boolean formulae in the context of maps of conjectures, (*iii*) the components of the Boolean formula seeker, and (*iv*) a short description of the different steps of the acquisition process of arithmetic-Boolean formulae.

2012 ACM Subject Classification • Computing methodologies Artificial intelligence Search methodologies • Mathematics of computing Discrete mathematics Combinatorics Combinatorial optimization

Keywords and phrases Acquisition of conjectures

Digital Object Identifier 10.4230/LIPIcs.CVIT.2016.23

Funding *Ramiz Gindullin*: funded by the EU-funded ASSISTANT project no. 101000165.

Acknowledgements Supervisor: Nicolas Beldiceanu, Co-authors: Jovial Cheukam-Ngouonou, Rémi Douence, Ramiz Gindullin, Claude-Guy Quimper

1 Type of arithmetic-Boolean formulae

The Boolean formula seeker acquires arithmetic-Boolean formulae from a table with integer data. It explains a selected output column with respect to a subset of selected input columns, where the same explanation must hold for every row of the table. Acquired arithmetic-Boolean formulae involve n arithmetic conditions linked by a single commutative logical operator or by a sum, denoted by g:

$g_{i=1}^{n}\mathcal{C}_i$, with $g \in \{\wedge, \vee, =, \underline{\vee}, +\}$ and $n \geq 1$,

where $\mathcal{C}_i$ is a condition mentioning arithmetic expressions using arithmetic functions such as min, max, mod , +. A condition such as $\underline{s} \geq 2$ is interpreted as an integer, i.e. either 0 for false or 1 for true.

Throughout this extended abstract we use the following notations for some characteristics of a digraph $\mathcal{G}$.

- v stands for number of vertices of $\mathcal{G}$,
- a stands for the number of arcs of $\mathcal{G}$,
- c stands for the number of connected components of $\mathcal{G}$,
- $\overline{c}$ stands for number of vertices of the largest connected component of $\mathcal{G}$,
- $\underline{s}$ stands for number of vertices of the smallest strongly connected component of $\mathcal{G}$,
- $\overline{s}$ stands for number of vertices of the largest strongly connected component of $\mathcal{G}$,
- $c_{>1}$ stands for the number of connected components of $\mathcal{G}$ with more than one vertex,
- $s_{>1}$ stands for the number of strongly connected components with more than one vertex.

▶ **Example 1** (Examples of acquired formulae for maps of interrelated conjectures)**.** We now provide two examples of generated formulae.

1. $c = 1 + (((\overline{c} - 2 \cdot \underline{s}) \leq 0) \wedge ((\overline{c} \mod \underline{s}) \geq 1))$
2. $c_{>1} = 2 - ((\underline{s} = \overline{s}) + (\underline{s} = 1))$

2 Importance of arithmetic-Boolean formulae in maps of interrelated conjectures

Bound Seeker is a CP-based system used for an acquisition of maps of conjectures for characteristics of combinatorial objects. Within Bound Seeker, the work on the Boolean formula seeker was motivated by the following observation. Quite often, Boolean arithmetic formulae are more appropriate than formulae involving polynomials: simpler Boolean arithmetic formulas are obtained, or a Boolean arithmetic formula is found, while a formula involving a polynomial could not be found.

▶ **Example 2.** For instance, without using arithmetic-Boolean formulae (and conditionals), Bound Seeker obtains:

- $a \geq s_{>1} - c_{>1} + v$, with
  - $s_{>1} = \min(\overline{s} - 1, 1)$,
  - $c_{>1} = \min(\min(\underline{c}, 2), \min(\underline{c}, 2) + \overline{c} - \underline{c} - 1)$,
  - $v = \min(\overline{c} + \underline{c}, \underline{c} \cdot \overline{c} - \underline{c}^2 + \overline{c})$.

By enabling arithmetic-Boolean formulae (and conditional formulae), Bound Seeker returns a simpler expression:

- $a \geq s_{>1} - c_{>1} + v$ with
  - $s_{>1} = (\overline{s} \geq 2)$,
  - $c_{>1} = (\underline{c} \geq 2) + ((\overline{c} - \underline{c}) \geq 1)$,
  - $v = (\text{if } (\overline{c} - \underline{c}) = 0 \text{ then } \underline{c}, \text{ else } \underline{c} + \overline{c})$.

3 Components of the Boolean formula seeker

The Boolean formula seeker includes:

1. A *rule generation part*, which generates rules preventing formulae to be rewritten into simpler formulae.
2. A *candidate arithmetic-Boolean formulae generation part*, which produces parameterised candidates arithmetic-Boolean formulae containing a specific Boolean aggregator, a given number of terms, and a candidate set of arithmetic conditions.
3. A *model generation part*, which produces a constraint model corresponding to a given candidate formula generated by Part (2).

4 Steps of the Boolean formula acquisition process

The Boolean formulae acquisition process consists of the following steps:

1. Generating rules that prevent formulae to be rewritten into simpler formulae. This step is performed only once and generates rules that can be used for acquiring arithmetic-Boolean formulae.
2. This second step generates the list of candidate formulae that can potentially explain an output column given a set of input columns for every entry of a table.
3. For each type of arithmetic-Boolean condition, this step calculates sets of possible coefficient values.

4. This step enumerates through the list of candidate formulae in order to:
   a. Generate the constraint model and create a cost criteria reflecting the simplicity of the formula;
   b. Post constraints on coefficient variables based on sets calculated in Step 3;
   c. Post constraints related to the rules generated in Step 1;
   d. Solve the constraint model and minimise its cost.

4.1 Rule generation that prevent rewriting

While these rules were initially handwritten, they are now automatically generated by a rule generation system. This rule generation system is a constraint program that finds rules belonging to one of the following families:

1. [only for $g \in \wedge, \vee$] Such rules avoid generating an arithmetic-Boolean condition that supercedes another arithmetic-Boolean condition. e.g., prevent from generating $[[(\overline{c} - \underline{c}) \geq 2] \wedge [\overline{c} \geq \underline{c}]]$ as it can be simplified to $[(\overline{c} - \underline{c}) \geq 2]$.
2. [for all g] Such rules avoid generating an arithmetic-Boolean condition of the form $cond_1\ g\ cond_2$ which is always true or always false. e.g. assuming that $v > 0, c > 0, d > 1$ then $[[v \leq c] \wedge [d \cdot v \geq c]]$ is always false.
3. [for all g] Such rules avoid generating other simplifiable arithmetic-Boolean conditions $cond_1\ g\ cond_2$ that do not belong to families 1 or 2. It means that there is another arithmetic-Boolean condition that is preferable, e.g.
   - $[[v \geq d] \vee [v = d - 1]]$ is not generated as it can be simplified to $[v \geq d - 1]$;
   - $d > 1 : [[|\overline{c} - \overline{s}| \leq c] \wedge [\overline{c} \leq \overline{s}]]$ is not generated as it can be rewritten as $d > 1 : [[\overline{s} - \overline{c} \leq d] \wedge [\overline{c} \leq \overline{s}]]$.

For the Boolean formula seeker we initially created 75 handwritten rules. The rule generation system found 3768 rules. These rules were combined with 14 handwritten rules that were outside the scope of the rule generator.

4.2 Candidate formulae generation part

All conjectures we generate have the form *characteristic op formula*, where *op* is one of the comparison operators $\leq$, $=$, $\geq$, and *formula* is a formula involving a set of characteristics. Consequently, formulae are described by the following set of simplified grammar rules, where "Small Capitals" indicates a non-terminal symbol, "Roman" denotes a function or a known constant, "**Bold**" denotes an unknown integer constant.

Formula ::= **cst** | Bool | **cst** + Bool | Cond | Pol | PolBinary | PolUnary
Bool ::= BoolOp(BoolConds) | ¬BoolOp(BoolConds)
BoolOp ::= $\wedge \mid \vee \mid = \mid \veebar \mid +$
BoolConds ::= BoolCond, BoolConds | BoolCond
BoolCond ::= Param Cmp **cst**
Cmp ::= $\leq \mid = \mid \geq \mid \neq$
Params ::= Param*
Param ::= Char|BTerm|UTerm
Char ::= $v|c|\underline{c}|\overline{c}|s|\underline{s}|\overline{s}$
BTerm ::= Bt(Char, Char)
UTerm ::= sum_consec(Char) | Ut(Char, **cst**) | Char $\in$ [**cst**, **cst**]
Bt ::= min | max | floor | ceil | mod | cmod | dmod | prod
Ut ::= min | max | floor | ceil | mod | geq | power

For performance reasons, the candidate formula generator first produces candidate formulae that do not mention the MOD function.

4.3 Calculation of sets of possible coefficient values

The goal of this step is to restrict the search space for unknown integer constants **cst**. For this purpose we take advantage of the values present in the source table itself. For example, for the Boolean condition $\overline{c}$ mod $\underline{s}$ =**cst**, we compute the set of possible values of **cst** from the pair of values of $\overline{c}$ and $\underline{s}$ from each entry of the table, and from the mod function.

4.4 Model generation part

Given a candidate arithmetic-Boolean formula for which the coefficients are not yet known, and a table for which the Boolean formula has to hold on every row, the system produces a constraint model using the following steps.

1. For each arithmetic-Boolean condition BOOLCOND in BOOLCONDS, a set of variables is created. This set includes attribute indices and coefficients that the arithmetic expression uses. If the Boolean condition is unused in the final formula, then these variables are set to some default values. Depending on the number of attributes involved in an BOOLCOND, it can either be a unary, binary or ternary term. For some types of Boolean conditions, we post an additional constraints to prevent the generation of either always true or false conditions, or conditions that can be rewritten to a simpler condition. Each Boolean condition has a cost defined by the arithmetic operators used, and by the sum of the absolute values of its coefficients. Such cost is minimised.
2. For each BOOLCOND i we create an integer variable b_i. Variable b_i denotes whether the Boolean condition i is
   - unused in the final formula,
   - used in the final formula,
   - used in the final formula in its negated form.

   The number of used arithmetic-Boolean conditions, whether negated or not, must be equal to the selected number of terms n of the arithmetic-Boolean formula.
3. For each created BOOLCOND i and table row j, a Boolean variable $b_{i,j}$ denotes whether or not the Boolean condition i is true or false for the j-th row of the table.
4. For each table row j a constraint is put on all b_i and $b_{i,j}$ that ensures that the calculated value of the result formula corresponds to the value of the output column for this row. This constraint considers the selected aggregator g and the fact that the condition is negated or not.
5. To ensure the production of non-simplifiable formulae, we add the following constraints:
   - Constraints that restrict the usage of unary, binary or ternary conditions depending on the number of input parameters and the number of terms n. As all input parameters must be used, there are cases where the use of certain types of arithmetic-Boolean conditions is infeasible. e.g. if we have six input parameters and three conditions then unary arithmetic-Boolean conditions must be discarded; or if we have two input parameters and one condition then unary and ternary conditions must be discarded from consideration.
   - Symmetry breaking constraints between two arithmetic-Boolean conditions of the same type.
   - Constraints that limit the search space for unknown integer constants **cst** using the sets calculated during Step 3.

- Constraints that prevent the generation of Boolean conditions pairs that can be simplified. These constraints are based on the rules generated during Step 1.

5 Conclusion

Out of 552 combinations of input characteristics for which Bound Seeker tried to find a sharp bound, using only polynomials, it got at least one sharp bound for 428 combinations of characteristics, as well as 2211 conjectures. Using also arithmetic-Boolean and conditional expressions it found 3 extra bounds and 93 new conjectures. Using arithmetic-Boolean and conditional expressions generates 3.8% new formulae compared to when using polynomials alone; moreover, 31.07% of the formulae that use polynomials are replaced by simpler formulae that use arithmetic-Boolean or conditionals expressions. In future work, the formula seeker may also use the Boolean formula seeker to acquire more complex formulae such as for instance
$v = (\text{if } \underline{c} = 1 \text{ then } 0, \text{ else if } ((\underline{c} = \underline{s}) \vee (2 \cdot \underline{s} \leq \underline{c})) \text{ then } 1, \text{ else } 2)$.

References

1 Nicolas Beldiceanu, Jovial Cheukam-Ngouonou, Rémi Douence, Ramiz Gindullin, and Claude-Guy Quimper. Acquiring maps of interrelated conjectures on sharp bounds. In *The 28th International Conference on Principles and Practice of Constraint Programming, July 31–August 5, 2022, Haifa, Israel*, 2022.