


# Parameterised Treewidth for Constraint Modelling Languages

Justin Pearson

Department of Information Technology

Uppsala University

Uppsala, Sweden

Email: justin.pearson@it.uu.se 

**Abstract**—Constraint programming is a widely used paradigm to solve combinatorial problems. High-level constraint modelling languages, such as MiniZinc, GAMS, OPL, and AMPL, encourage the separation of instance parameters and models, where the number of variables and constraints depend on values of instance parameters. This paper investigates an extension of treewidth suitable for studying the complexity high-level constraint models.

## I. INTRODUCTION

Constraint modelling languages, such as MiniZinc [1], Essence [2], OPL [3], and AMPL [4], have high-level modelling abstractions such as array variables, comprehensions, and quantified formulae that allow complex models to be easily stated. These abstractions are not *directly* supported by constraint solvers (for example Gecode [5]), and there is tool support to translate models into lower-level models suitable for constraint solvers.

In such constraint modelling languages there is a clear separation of input data and model that can be summed up by the following slogan:

Instance = Model + Data.

An *instance* is a low-level translation of a model and input data into a form suitable for a constraint solver, such as Gecode. Translating a model together with input data is often referred to as *flattening*.

A constraint satisfaction problem (CSP) is a fixed collection of variables and constraints over those variables. An instance in the above sense is an example of a CSP. The problem of solving finite domain CSPs is NP-complete. While the study of the polynomial-time solvable classes of CSPs has been studied extensively (see the references in [6]), there has been very little practical application of such results. A fundamental idea is to translate CSPs as formulae in existential conjunctive first-order logic ( $\exists$ - $\wedge$ -FOL) and to study the tree-width of formulae (see Section VI). For example, if a CSP belongs to a class of CSPs of bounded treewidth (see Section VI), then backtrack-free search can be achieved by enforcing local consistency, which can be done in polynomial time. In [7], [8] tree-decompositions of CSP were computed and used to improve such, and even in large instances, there is a payoff in

computing a tree decomposition of a CSP, but there has been no practical applications of such results.

To the best of the author's knowledge, there has been no research on providing theoretical tools to study the computational complexity of models written in a language such as MiniZinc. It is not possible to define the Treewidth of for MiniZinc models, because a MiniZinc model contains instance parameters that are not instantiated until flattening.

The main contribution of this paper include a *new* definition of parameterised treewidth (Section VII) that captures how the treewidth of flattened instances varies with instance data, and a static characterisation (Section VIII) of parameterised treewidth of a high-level model in terms of a new type of structural graph of a high-level model.

This paper is a first attempt to bridge the gap between the theory of CSPs and high-level modelling languages such as MiniZinc. Characterising how the treewidth of instances varies with input data for a given model is a difficult theoretical challenge, and Theorem 1 is a first step.

## II. CONSTRAINT MODELLING LANGUAGES

```

1 int: N;
2 array[1..N, 1..N] of var 1..N: B;
3 constraint forall(i in 1..N) (
4   alldifferent([B[i, j] | j in 1..N]));
5
6 constraint forall(j in 1..N) (
7   alldifferent([B[i, j] | i in 1..N]));
8 solve satisfy;
```

Fig. 1. A MiniZinc Latin Square Model

In this section we give a short overview of the features of high-level constraint modelling languages necessary to understand the rest of the paper.

Languages such as MiniZinc [1], OPL [3] and AMPL [4] have high-level modelling abstractions for arrays and set variables that express constraints over arrays or sets. Essence [2] is even more powerful, and has abstractions for sets of set variables, function variables, and other more complex variable types. In this paper, we only consider high-level modelling abstractions for arrays. In order to illustrate such high-level

This work was partially supported by VR, the Swedish Research Council, under Research Grant number 2018-04813

modelling abstractions we use the language MiniZinc. The same constructs can be found in OPL, AMPL or Essence.

In Figure 1 we see a MiniZinc model for Latin squares<sup>1</sup>. The model has four parts that are shared by almost all MiniZinc models: in line 1 an instance parameter  $N$  is defined; in line 2 the two-dimensional array variable  $B$  is declared; lines 3 to 7 define some constraints on those variables declared in line 2; and line 8 directs the chosen solver to find a solution that satisfies the conjunction of all constraints in the model. In more realistic examples, models will have many instance parameters and variables. Lines 3 and 6 contain a `forall` statement, and here each `forall` statement is a shorthand for the conjunction of  $N$  expressions. Lines 4 and 7 contain the array comprehensions,  $[B[i, j] \mid i \text{ in } 1..N]$  and  $[B[i, j] \mid j \text{ in } 1..N]$ . The expression  $1..N$  denotes the set of integers from 1 to  $N$ . Such expressions are often used to determine array bounds or as parameters for array comprehensions. Finally, `alldifferent` is a constraint with a single parameter, which is a 1-dimensional array. The constraint `alldifferent`( $L$ ) is true when the elements in  $L$  are pairwise different. The model in Figure 1 defines a family of instances depending on the value of the instance parameter  $N$ .

Constraint solvers do not typically support MiniZinc’s modelling abstractions, and the MiniZinc toolchain flattens a MiniZinc model and its instance parameters into the lower-level language FlatZinc. FlatZinc is closer to the modelling style required for a constraint solver. A constraint solver only has to provide an interface to FlatZinc to take advantage of the power offered by the MiniZinc language. FlatZinc has no quantification or comprehensions, and the flattened model has all quantifications and comprehensions rewritten as conjunctions of constraints.

### III. MINIZINC AND MANY-SORTED FIRST-ORDER LOGIC

MiniZinc is essentially a multi-sorted<sup>2</sup> first-order language with syntactic sugar and some restrictions on how quantifiers are used. To save space and to simplify the presentation we only consider 1-dimensional arrays, although it would be easy to extend the theory to multi-dimensional arrays.

We require the following sorts: The sort of values,  $\mathbf{D}$ , an ordered sort used for indexing arrays,  $\mathbf{I}$ , a sort for expressing ranges,  $\mathbf{R}$ , and a sort of 1-dimensional arrays,  $\mathbf{A}_1$ . We need a number of predicates, function symbols and some syntactic sugar. Given two elements  $i$  and  $j$  of  $\mathbf{I}$  with  $i \leq j$ , then  $[i, j]$  is a term of sort  $\mathbf{R}$ . Given an element  $i$  of sort  $\mathbf{I}$  and a variable,  $L$ , of sort  $\mathbf{A}_1$  then  $L[i]$  is a term of sort  $\mathbf{D}$ , and the predicate  $\text{rangedef}_1(L, [l, u])$  asserts that elements of the array  $L$  are from  $L[l]$  to  $L[u]$ . Finally, given a term  $i$  of sort  $\mathbf{I}$  and a term  $r$  of sort  $\mathbf{R}$ , then the predicate  $i \in r$  asserts that  $i$  belongs to the range  $r$ .

<sup>1</sup>A Latin square is a square array of numbers, such that no number appears twice in each row or twice in each column.

<sup>2</sup>Traditionally in first-order logic [9] the types of terms or variables are referred to as sorts.

Global constraints, such as `alldifferent`, are represented as predicates. For example, the predicate `alldifferent` takes a single argument of sort  $\mathbf{A}_1$ . A MiniZinc formula with a universal quantification such as `forall (i in [1..n]) (p(i))` is translated as a universal quantification  $(r = [1, n]) \wedge (\forall(i \in r) \Rightarrow p(i))$ .

For example the expansion of `forall (i in 1..N, j in 1..N where i < j) (X[i] ≠ X[j])` into inequalities is equivalent to the formula  $\phi_{\text{alldifferent}}$  defined as

$$\text{rangedef}_1(X, [1, n]) \wedge \forall(i \in [1, n]) \Rightarrow \forall(j \in [1, n]) \Rightarrow (i < j) \Rightarrow X[i] \neq X[j]. \quad (1)$$

Another example that will be used later is a constraint,  $\phi_{\text{linord}}$ , that asserts that the array  $L$  is ordered, defined as

$$\text{rangedef}_1(L, [1, n]) \wedge \forall(k \in [1, n-1]) \Rightarrow L[k] \leq L[k+1]. \quad (2)$$

An array comprehension is an existential quantification over a new variable together with a formula asserting that the new array variable satisfies the comprehension condition. For example line 4 of Figure 1 is equivalent to the formula

$$\exists T. \exists r. r = [1, n] \wedge \text{rangedef}_1(T, r) \wedge \text{alldifferent}(T) \wedge (\forall(j \in r) \Rightarrow (T[j] = B[i, j])).$$

### IV. CONJUNCTIVE SPECIFICATIONS

In many-sorted first-order logic, quantifiers can be arbitrarily nested. With arbitrarily nested quantifiers, it is possible to express quantified constraint problems that are more complex to solve than CSPs (PSPACE complete). While in MiniZinc, it is only possible to state satisfaction (or optimisation problems) problems. Universal quantifiers can only quantify over sets of variables or sets of values known when flattening.

Below (Definition 1), we define the notation of a conjunctive specification, where only a restricted class of formulae are allowed that mimic how MiniZinc models are used.

Propagation-based constraint solvers, such as Gecode [5], typically work on conjunctions of constraints. Unlike SMT [10], disjunctions are handled with specialised techniques such as constructive disjunction [11], or propagation algorithms for disjunctions of global constraints [12] (as used in the Minion solver [13]). If such techniques are not available, then the modeller has to resort to reification to express disjunction [14]. Currently, all back ends to MiniZinc work through FlatZinc, and in FlatZinc all disjunctions of constraints are made explicit using reification. In this paper, we treat disjunctions as syntactic sugar for reification.

**Definition 1.** A conjunctive specification is a triple  $(\phi, I, O)$ , where  $I$  and  $O$  are disjoint sets of variables where  $I \cup O = \text{FV}(\phi)$ , and there are no variables of sort  $\mathbf{I}$  (no free indexing variables) in  $I$  or  $O$ . The formula  $\phi$  has the following restrictions: all uses of universal quantification are of the form  $\forall(i \in r) \Rightarrow (\psi \Rightarrow \chi)$  for arbitrary formulas  $\psi$  and  $\chi$ ; the arguments,  $s_1, s_2, t$ , to range expressions  $[s_1, s_2]$ , or

array accesses  $L[t]$  can be any term of the correct sort; in predicates,  $R(t_1, \dots, t_m)$  the arguments,  $t_1, \dots, t_m$  are either constants, variables or terms of the form  $L[t]$ ; the only allowed connective is conjunction; and existential quantification is only allowed for variables of sort  $\mathbf{A}_1$  (one dimensional arrays).

The restriction to conjunction forces all disjunctions to be made explicit with reification as is done in FlatZinc. The restriction that only constants or variables or arrays access appearing in relations is to simplify the definition of the structural graph (Definition 4), and the proof of Theorem 1. The restricted use of universal quantification mimics how `forall` expressions are used in MiniZinc.

In a conjunctive specification  $(\phi, I, O)$ ,  $I$  is referred to as the instance variables, and  $O$  is referred to as the output variables. The variables in  $I$  correspond to the parameters of a MiniZinc model, and the variables in  $O$  are the constraint variables of the problem.

## V. FLATTENING CONJUNCTIVE SPECIFICATIONS

FlatZinc does not have arrays, and all MiniZinc arrays are replaced during flattening by sequences of non-array variables, with new names for each element of an array.  $\exists\text{-}\neg\text{-FOL}$  is a single sorted logic without array variables, and array variables in conjunctive specifications are replaced with single sorted variables in  $\exists\text{-}\neg\text{-FOL}$ . Rather than inventing a new naming scheme, array access terms  $L[c]$  where  $c$  is a constant of sort  $\mathbf{I}$  is here considered as terms of sort  $\mathbf{D}$  in  $\exists\text{-}\neg\text{-FOL}$ .

While flattening a MiniZinc model all unknown values are replaced by input data. All `forall` expressions are unrolled as a conjunction of expressions with array index variables replaced by values. It is possible there are array access out of bounds or undetermined values. Similarly, it is possible there are some conjunctive specifications that could not be flattened. Here, for convenience, we assume that conjunctive specifications that are considered can be flattened for all values of the input data.

Given some assignment  $\alpha$ , let  $\phi_\alpha$  be the formula  $\phi$  with  $c$  being substituted for the variable  $v$  for all assignments  $v = c$  in  $\alpha$ . Flattening then works recursively on conjunctions. The only case that requires some care is universal quantification. Given a formula  $\phi$  and a variable  $v$  belonging to the free variables of  $\phi$  and a constant  $c$  then let  $\phi[v \mapsto c]$  denote the formula with all occurrences of  $v$  replaced with the constant  $c$ . Given a formula  $\forall(i \in [\ell, u]) \Rightarrow (\phi \Rightarrow \psi)$  that can be syntactically flattened, its flattened version is the formula

$$\bigwedge \{ \psi[i \mapsto v] \mid v \in [\ell, u] \wedge \alpha \wedge \phi[i \mapsto v] \text{ is satisfiable} \}.$$

The big conjunction is not a new connective, but syntactic sugar for a finite conjunction of  $\exists\text{-}\neg\text{-FOL}$  formulae where the variable  $i$  is replaced by values from the interval  $\ell \leq i \leq u$ . Note that if the universal quantification can be syntactically flattened then  $\ell$  and  $u$  will be replaced by constants. By convention, we assume that an empty conjunction is **true**.

Consider the conjunctive specification  $(\phi_{\text{alldifferent}}, \{n\}, \{X\})$ , where  $\phi_{\text{alldifferent}}$  is defined in

(1), given the assignment  $n = k$  where  $k$  is a positive integer, then the flattened formula,  $\text{flatten}(\phi_{\text{alldifferent}}, X, n = k)$ , is:

$$\bigwedge_{1 \leq i < j \leq k} X[i] \neq X[j]. \quad (3)$$

The conjunctive specification  $(\phi_{\text{linord}}, \{n\}, \{L\})$ , where  $\phi_{\text{linord}}$  is defined in (2), then given the assignment  $n = c$  the flattened formula,  $\text{flatten}(\phi_{\text{linord}}, L, n = c)$ , is equal to:

$$\bigwedge_{1 \leq k < c} L[k] \leq L[k+1]. \quad (4)$$

## VI. BACKGROUND ON TREEWIDTH

In this section, we review treewidth and CSPs. For more details see for example [6], [15] or [16].

A *constraint satisfaction problem* (CSP),  $(V, D, \{R_i(S_i)\}_{i \in I})$ , is a set of variables  $V$ , a domain  $D$ , and a set of constraints  $\{R_i(S_i)\}_{i \in I}$ . A *Constraint* is a relation,  $R_i \subseteq D^{|S_i|}$ , over sequences of variables,  $S_i$ , of length  $|S_i|$ . A constraint restricts the possible values that combinations of variables can take. A *solution* is an assignment,  $f : V \rightarrow D$ , of variables that satisfy all constraints simultaneously. That is, for all  $i$  in  $I$  the tuple  $\langle f(v_{i_1}), \dots, f(v_{i_k}) \rangle$  belongs to  $R_i$ , where  $S_i$  is the sequence  $\langle v_{i_1}, \dots, v_{i_k} \rangle$ . A CSP  $(\{v_1, \dots, v_n\}, D, \{R_i(S_i)\}_{i \in I})$  has a solution, if and only if the formula  $\exists v_1 \dots \exists v_n. \bigwedge_{i \in I} R_i(S_i)$  is satisfiable when values for the variables  $v_i$  taken from the domain  $D$ .

A tree is a directed acyclic graph with a root. The treewidth of a graph measures how close a graph is to a tree. A tree has treewidth 1, and if a graph contains a  $k$ -clique then its treewidth is at least  $k-1$ . The treewidth is determined by a tree decomposition of a graph. Informally, a tree decomposition is a tree with a set of nodes for each element of the tree satisfying certain conditions: in a tree covering every edge from the original graph must appear in some set, and if you restrict the tree to sets only containing some node  $v$  then that forms a connected subset of the tree. The treewidth of a tree decomposition is then the size of the largest set in the tree decomposition minus one. The treewidth of a graph is then the minimum treewidth over all possible tree decompositions. Many graph theoretic algorithms are efficient when restricted to graphs of low treewidth [16].

**Definition 2.** Let  $G = (V, E)$  be a graph and  $T$  be a tree (seen as a graph), and for each node  $t \in T$  let the set  $B_t$  be some subset of  $V$ . The pair  $(T, \{B_t\}_{t \in T})$  is a tree-decomposition [16] of  $G$  if it satisfies the following three conditions:

- T0  $V(G) = \bigcup_{t \in T} B_t$ ;
- T1 for every edge  $e$  in  $E$  there is some node  $t$  of  $T$  such that  $e \subseteq B_t$ ;
- T2 For every  $v \in V$ , the set  $B^{-1}(v) = \{t \in T \mid v \in B_t\}$  is non-empty and connected in the tree  $T$ .

See Figure 2 in Section VIII for an example graph with its tree decomposition. Given a tree decomposition  $(T, \{B_t\}_{t \in T})$ ,

the *width* is defined to be the value  $\max\{|B_t| \mid t \in T\} - 1$ . Given a graph  $G$ , the *treewidth* of  $G$  is the minimum treewidth of all tree decompositions.

Given a formula,  $\phi = \exists v_1 \exists v_2 \dots \exists v_n \bigwedge_{i \in I} R_i(S_i)$ , the *Gaifman graph* [16] of  $\phi$  denoted  $G(\phi)$  is defined as follows:  $G(\phi)$  is the graph  $(V, E)$  where  $V$  is the set,  $\{v_1, \dots, v_n\}$ , of variables appearing in  $\phi$ ; and the set  $E$  consists of pairs,  $\{v_i, v_j\}$ , where  $v_j$  and  $v_j$  are in the scope  $S_i$  of some relation  $R_i$  in  $\phi$ .

For example, the  $\exists\text{-}\wedge\text{-FOL}$  formula  $L_1 \neq L_2 \wedge L_1 \neq L_3 \wedge L_2 \neq L_3$  has a Gaifman graph the clique on the set of nodes  $\{L_1, L_2, L_3\}$  and thus has treewidth 2. While the  $\exists\text{-}\wedge\text{-FOL}$  formula  $L_1 \leq L_2 \wedge L_2 \leq L_3$  has a Gaifman graph that is a single line and hence has treewidth 1.

It is also possible to build a hypergraph from a formula, and define its treewidth. The treewidth in the hypergraph formulation is equivalent to the treewidth of the graph formulation [16]. For more on the relationship between treewidth and complexity, see [17].

## VII. PARAMETERISED TREewidth

Parameterised treewidth (Definition 3) is a function from values of instance parameters to the integers. The value of the parameterised treewidth function is the treewidth of a flattened instance for a particular value of instance parameters.

Abstractly, the flattening process takes a conjunctive specification,  $(\phi, I, O)$ , an assignment,  $\alpha$  of  $I$ , and returns a formula in  $\exists\text{-}\wedge\text{-FOL}$ . Let  $\text{flatten}(\phi, O, \alpha)$  be the flattened  $\exists\text{-}\wedge\text{-FOL}$  formula. For the correctness of flattening process, the set of solutions of  $\phi \wedge \alpha$  must be equal to the set of solutions of  $\text{flatten}(\phi, O, \alpha)$ . Using this notation, we define a *new* notion of treewidth of a conjunctive specification in many-sorted first-order logic parameterised by assignments of instance variables. Let  $\text{tw}_{\exists\text{-}\wedge\text{-FOL}}$  denote the treewidth of a formula in  $\exists\text{-}\wedge\text{-FOL}$  as defined in Section VI.

**Definition 3.** Given a conjunctive specification,  $(\phi, I, O)$ , the parameterised treewidth of  $(\phi, I, O)$  is a function  $\text{tw}_\phi$  from assignments of the inputs variables to the integers defined as

$$\text{tw}_\phi(\alpha) = \text{tw}_{\exists\text{-}\wedge\text{-FOL}}(\text{flatten}(\phi, O, \alpha)) .$$

Consider the conjunctive specification  $(\phi_{\text{alldifferent}}, \{n\}, \{X\})$  where  $\phi_{\text{alldifferent}}$  is defined in (1). The Gaifman graph of the flattened formula (3) for an assignment  $n = k$  is a clique on  $k$  nodes, and hence has treewidth  $k - 1$ . Hence, the parameterised treewidth of the specification is the function that takes the assignment  $n = k$  to  $k - 1$ .

Similarly, the conjunctive specification  $(\phi_{\text{linord}}, \{n\}, \{L\})$  where  $\phi_{\text{linord}}$  is defined in (2), will have parameterised treewidth that takes an assignment  $c$  of  $n$  to 1 because the Gaifman graph of each flattened instance has treewidth 1.

A naive way of computing parameterised treewidth would be to flatten a formula and then compute the treewidth of the flatten instance. Computing the parameterised treewidth directly from a conjunctive specification is harder. In Section VIII we show how to define a graph, the structure graph

of  $(\phi, I, O)$ , that is analogous to the Gaifman graph of a CSP, where the treewidth of the structure graph can be used to compute an upper bound of the treewidth of flattened instances.

## VIII. APPROXIMATING PARAMETERISED TREewidth

In this section, we define the structure graph (Definition 4) of a conjunctive specification, which is an analogue of the Gaifman graph. Using Theorem 1 it is possible to approximate the parameterised treewidth (Definition 3) using the structure graph (Definition 4).

As in [18], we say that a formula  $\phi$  is *straight* if no variable appearing in  $\phi$  is quantified over twice. Any formula is logically equivalent to a straight formula. From now on, we assume that all formulae are straight, including flattened formulae. During flattening, if an existential quantification appears in the scope of a universal quantifier, then the existentially quantified variables are renamed to keep the formula straight. For example  $\text{flatten}(\forall(i \in [1, n]) \Rightarrow \exists T \phi(i), \{n\}, n = 2)$  gets flattened to the formula  $\exists T^1 \phi(1) \wedge \exists T^2 \phi(2)$ .

To construct an analogue of the Gaifman graph for conjunctive specifications, it is essential to know where variables in a specification end up in a flattened formula. Given a specification  $(\phi, I, O)$  and a formula  $\text{flatten}(\phi, O, \alpha)$ , each variable (quantified or free) in  $\phi$  and each term of the form  $L[i]$  will get flattened into several variables for each array element.

We use the function  $\text{expand}(\phi, O, \alpha)$  to keep track of how variables get flattened in the formula  $\text{flatten}(\phi, O, \alpha)$ . For example, given the conjunctive specification  $(\phi_{\text{alldifferent}}, \{n\}, \{X\})$  with the assignment  $n = k$ , then  $\text{expand}(\phi, O, \alpha)(X[j]) = \text{expand}(\phi, O, \alpha)(X[i])$  is the set  $\{X[1], \dots, X[k]\}$ . For an existentially quantified variable,  $T$ , in the scope of a universal quantifier,  $\text{expand}(\phi, O, \alpha)(T)$  is the set of renamed existentially quantified variables. For example, the new variables  $T^1$  and  $T^2$  that appear in the example above.

**Definition 4.** Given a conjunctive specification  $(\phi, I, O)$ , the structure graph  $G_\phi = (V, E)$  is defined as follows:

- 1) The set of nodes  $V$  contains two types of nodes: variables of sort **D** in the set  $O$ ; and terms of the form  $L[t]$  appearing in  $\phi$  where  $L$  appears either in  $O$ , or  $L$  is an existentially quantified variable.
- 2) Given any two nodes  $v$  and  $w$  in  $V$ , if  $v$  and  $w$  appear in the scope of some relation  $R(\dots, v, \dots, w, \dots)$  or in some equality term  $\phi = \psi$  (where  $\phi$  and  $\psi$  are terms), then there is an edge  $\{v, w\}$  in  $E$ .
- 3) Given any two nodes  $L[t]$  and  $L[t']$ , then there is an edge  $\{L[t], L[t']\}$  in  $E$ .

Consider the conjunctive specification  $(\phi_{\text{alldifferent}} \wedge \phi_{\text{linord}} \wedge \phi_s, \{n\}, \{X, L, S\})$  where  $\phi_{\text{alldifferent}}$  is defined in (1),  $\phi_{\text{linord}}$  in (2) and  $\phi_s$  is defined as

$$\forall(\ell \in [1, N])(X[\ell] < S) \wedge (S < L[\ell]) .$$

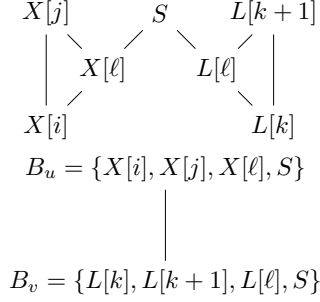


Fig. 2. Structure graph with its tree decomposition

This conjunctive specification has the structure graph given in Figure 2. Its tree decomposition consists of two bags  $B_u = \{X[i], X[j], S\}$  and  $B_v = \{L[i], L[i+1], S\}$ , and the tree with nodes  $u$  and  $v$  with a single edge between  $u$  and  $v$ .

Together with a tree decomposition on a structure graph  $(\phi, I, O)$  and the  $\text{expand}(\phi, O, \alpha)$  function, we define the expansion of the tree decomposition.

**Definition 5.** Given a conjunctive specification  $(\phi, I, O)$  and a tree decomposition  $(T, \{B_t\}_{t \in T})$  of the structure graph of  $(\phi, I, O)$ , the expansion of  $(T, \{B_t\}_{t \in T})$  with respect to an assignment  $\alpha$  is the pair  $(T, \{\text{expand}(\phi, O, \alpha)(B_t)\}_{t \in T})$ . For any set  $B_t$  in the tree decomposition, the expansion of  $B_t$  is defined to be the set

$$\text{expand}(\phi, O, \alpha)(B_t) = \bigcup_{v \in B_t} \text{expand}(\phi, O, \alpha)(v).$$

We now prove that the expansion of a tree decomposition is a tree decomposition of the Gaifman graphs of flattened formulae. Hence expansions provide upper bounds on the treewidth of flattened instances.

**Theorem 1.** Given a specification  $(\phi, I, O)$  and a tree decomposition  $(T, \{B_t\}_{t \in T})$  of the structure graph of  $(\phi, I, O)$ , the expansion  $(T, \{\text{expand}(\phi, O, \alpha)(B_t)\}_{t \in T})$  of  $(T, \{B_t\}_{t \in T})$  with respect to an assignment  $\alpha$  is a tree decomposition of the Gaifman graph of  $\text{flatten}(\phi, O, \alpha)$ .

*Proof.* Let  $G_f$  denote the Gaifman graph of  $\text{flatten}(\phi, O, \alpha)$ . We need to show that Conditions **T1** and **T2** hold (of Definition 2) for  $(T, \{\text{expand}(\phi, O, \alpha)(B_t)\}_{t \in T})$  to be a tree decomposition of  $G_f$ .

First condition **T1**: Given some edge  $\{x, y\}$  in  $G_f$ , then  $x$  and  $y$  appear in the scope of some relation,  $R(\dots, x, \dots, y, \dots)$ , which is the result of the flattening process. Hence in  $\phi$  there must be variables  $u$  and  $v$  such that  $u$  and  $v$  appear in the scope of the same relation,  $R(\dots, u, \dots, v, \dots)$ . This implies that  $u \in \text{expand}(\phi, O, \alpha)(x)$  and  $v \in \text{expand}(\phi, O, \alpha)(y)$ , where  $\{u, v\}$  belongs to  $B_t$  for some  $t$ , therefore  $\{x, y\}$  belongs to  $\text{expand}(\phi, O, \alpha)(B_t)$ .

Let  $\text{expand}(\phi, O, B)^{-1}(x)$  denote the set

$$\{t \in T \mid x \in \text{expand}(\phi, O, \alpha)(B_t)\}.$$

Second, for condition **T2** we need to show for all nodes  $x$  in the Gaifman graph  $G_f$  that the set  $\text{expand}(\phi, O, B)^{-1}(x)$  is non-empty and forms a connected component in the tree  $T$ . Let  $V_s$  denote the nodes of the structure graph. Note that  $\text{expand}(\phi, O, B)^{-1}(x)$  equals

$$\{t \mid \exists v \in V_s. x \in \text{expand}(\phi, O, \alpha)(v) \wedge v \in B_t\}.$$

Hence  $\text{expand}(\phi, O, B)^{-1}(x)$  equals

$$\bigcup \{B^{-1}(v) \mid x \in \text{expand}(\phi, O, \alpha)(v)\}.$$

Note that we are using the notation  $B^{-1}(v)$  from Definition 2.

Thus it is enough to show given any node  $x$  in  $G_f$  where  $x$  belongs to  $\text{expand}(\phi, O, \alpha)(u)$  and  $\text{expand}(\phi, O, \alpha)(v)$  that  $B^{-1}(u)$  and  $B^{-1}(v)$  are connected.

There are two types of nodes in the Gaifman graph  $G_f$ : a node that occurs in the structure graph of  $(\phi, I, O)$ ; a node is of the form  $L[c]$  for a constant  $c$ . Note that for a node  $L[t]$  then  $t$  will be flattened as a constant, because any array access variable of sort **I** must be in the scope of a universal quantifier.

If  $x$  occurs in the structure graph of  $(\phi, I, O)$  then  $\text{expand}(\phi, O, B)^{-1}(x)$  is connected because  $B^{-1}(x) = \text{expand}(\phi, O, B)^{-1}(x)$  must be connected to satisfy Definition 2.

If  $x$  is of the form  $L[c]$  where  $L$  is not in the scope of an existential quantifier, then suppose that  $x$  belongs to more than one set  $\text{expand}(\phi, O, \alpha)(\lambda_t)$  and  $\text{expand}(\phi, O, \alpha)(\lambda_{t'})$ , where  $\lambda_t$  is the term  $L[t]$  and  $\lambda_{t'}$  is the term  $L[t']$ . The sets  $B^{-1}(\lambda_t)$  and  $B^{-1}(\lambda_{t'})$  are connected, because there is an edge between  $L[t]$  and  $L[t']$  in the structure graph by condition 3 of Definition 4. This edge belongs to some set  $B_t$  for  $t \in B^{-1}(L[t_1, \dots, t_n]) \cap B^{-1}(L[t'_1, \dots, t'_n])$ .  $\square$

Using Theorem 1 we can obtain an upper bound on the parameterised treewidth.

**Lemma 1.** Given a specification  $(\phi, I, O)$  and a tree decomposition  $(T, \{B_t\}_{t \in T})$  of its structure graph. Let  $t_s$  be its treewidth, that is  $t_s = \max\{|B_t| \mid t \in T\} - 1$ . Let  $e_{\max}$  be

$$\max_{X \in I} |\text{expand}(\phi, O, \alpha)(L)|.$$

Then any assignment  $\alpha$  of  $I$ , then the treewidth the Gaifman graph of  $\text{flatten}(\phi, O, \alpha)$  is at most  $(t_s + 1)e_{\max} - 1$ .

*Proof.* In the expansion  $(T, \{\text{expand}(\phi, O, \alpha)(B_t)\}_{t \in T})$  (Definition 5) of  $(T, \{B_t\}_{t \in T})$  size of  $\text{expand}(\phi, O, \alpha)(B_t)$  is

$$\sum_{X \in B_t} |\text{expand}(\phi, O, \alpha)(X)|$$

which is at most  $|B_t|t_s$ . Finally, let  $B_s$  be a set of maximum size then  $|B_s|t_s$  equals  $(t_s + 1)e_{\max}$ .  $\square$

As discussed at the end of Section VII, the parameterised treewidth of the conjunctive specification  $(\phi_{\text{alldifferent}}, \{n\}, \{X\})$  is a function from assignments  $n = k$  to  $k - 1$ . Constructing the structural graph and computing the treewidth of each instance using Lemma 1 gives the required

treewidth  $k - 1$ . While for the conjunctive specification  $(\phi_{\text{linord}}, \{n\}, \{L\})$ , the treewidth of each instance is 1, but constructing the structural graph and computing the treewidth of each instance using Lemma 1 gives the value  $n - 1$ . In the conclusion (Section IX), we discuss why this is the case and future work on improving approximations.

## IX. CONCLUSIONS AND FUTURE WORK

Parameterised treewidth is a new way to study the complexity of problems expressed in high-level constraint-modelling languages. Parameterised treewidth captures how the treewidth of an instance varies with the values of instance parameters. With Theorem 1 it is possible to compute an upper bound of parameterised treewidth by studying the structure graph of a specification. It is the first step in identifying tractable classes of constraint models of bounded treewidth.

In the tree decomposition in Figure 2, the two sets  $B_u$  and  $B_v$  share an edge in the tree and have the intersection  $\{S\}$ . The variable  $S$  acts as a separator in the structural graph. In any tree decomposition [16] the intersection of two sets  $B_u$  and  $B_v$  connected by an edge  $\{u, v\}$ , gives a separator. In any expansion of the tree decomposition, the (flattened) variable  $S$  will still act as a separator. A separator divides the CSP instance into two independent sub-problems. Independent sub-problems are good candidates for pre-solving to reduce backtracking. In [19], [20] automatic methods are provided to pre-solve parts of the models, and it would be interesting to investigate the use of separators as a principled way of finding good candidates for presolving.

The expansions of tree decompositions from Theorem 1 often only provide a strict upper bound on treewidth (as discussed at the end of Section VIII). This is because of the static nature of the expansion in Definition 5. The expanded tree decomposition must have the same tree structure as the tree decomposition of the structure graph (Definition 4). Further, the structure graph often contains unnecessary cliques from condition 3: even though there might be an edge between  $L[v]$  and  $L[w]$  when  $v \neq w$  for some flattened instances depending on the instance parameters. This upper bound is probably the best that can be achieved by only considering the structural graph of a model. To improve the approximations of parameterised treewidth a more dynamic method of expanding tree decompositions would have to be found, and further semantic analysis of which edges in condition 3 are unnecessary.

## REFERENCES

- [1] N. Nethercote, P. J. Stuckey, R. Becket, S. Brand, G. J. Duck, and G. Tack, "MiniZinc: Towards a standard CP modelling language," in *CP 2007*, ser. LNCS, C. Bessière, Ed., vol. 4741. Springer, 2007, pp. 529–543.
- [2] A. M. Frisch, M. Grum, C. Jefferson, B. Martinez Hernandez, and I. Miguel, "The design of Essence: A constraint language for specifying combinatorial problems," in *IJCAI 2007*. Morgan Kaufmann, 2007, pp. 80–87.
- [3] P. Van Hentenryck, "Constraint and integer programming in OPL," *INFORMS Journal on Computing*, vol. 14, no. 4, pp. 345–372, 2002.

- [4] R. Fourer, D. M. Gay, and B. W. Kernighan, *AMPL: A Modeling Language for Mathematical Programming*, 2nd ed. Cengage Learning, 2002.
- [5] Gecode Team, "Gecode: A generic constraint development environment," 2024, the Gecode solver and its MiniZinc backend are available at <https://www.gecode.org>.
- [6] C. Carbonnel and M. C. Cooper, "Tractability in constraint satisfaction problems: A survey," *Constraints*, vol. 21, no. 2, pp. 115–144, April 2016.
- [7] P. Jégou and C. Terrioux, "Combining restarts, nogoods and bag-connected decompositions for solving csps," *Constraints An International Journal*, vol. 22, no. 2, pp. 191–229, 2017.
- [8] P. Jégou, H. Kanson, and C. Terrioux, "On the relevance of optimal tree decompositions for constraint networks," in *IEEE 30th International Conference on Tools with Artificial Intelligence, ICTAI 2018, 5-7 November 2018, Volos, Greece*, L. H. Tsoukalas, É. Grégoire, and M. Alamaniotis, Eds. IEEE, 2018, pp. 738–743.
- [9] J. Gallier, *Logic for Computer Science*. Dover, 2015, second Edition.
- [10] D. Kroening and O. Strichman, *Decision Procedures: An Algorithmic Point of View*, 2nd ed., ser. Texts in Theoretical Computer Science, an EATCS Series. Springer, 2016.
- [11] J. Würtz and T. Müller, "Constructive disjunction revisited," in *KI-96: Advances in Artificial Intelligence*, ser. Lecture Notes in Artificial Intelligence, G. Götz and S. Hölldobler, Eds., vol. 1137. Berlin, Heidelberg: Springer Berlin Heidelberg, 1996, pp. 377–386.
- [12] C. Jefferson, N. C. Moore, P. Nightingale, and K. E. Petrie, "Implementing logical connectives in constraint programming," *Artificial Intelligence*, vol. 174, no. 16, pp. 1407–1429, 2020.
- [13] I. P. Gent, C. Jefferson, and I. Miguel, "Minion: A fast scalable constraint solver," in *ECAI 2006*, G. Brewka *et al.*, Eds. IOS Press, 2006, pp. 98–102.
- [14] N. Beldiceanu, M. Carlsson, P. Flener, and J. Pearson, "On the reification of global constraints," *Constraints*, vol. 18, no. 1, pp. 1–6, January 2013.
- [15] G. Gottlob, N. Leone, and F. Scarcello, "Hypertree decompositions and tractable queries," *Journal of Computer and System Sciences*, vol. 64, no. 3, pp. 579–627, 2002.
- [16] J. Flum and M. Grohe, *Parameterized Complexity Theory*, ser. Texts in Theoretical Computer Science. Springer, 2008.
- [17] N. Creignou, P. G. Kolaitis, and H. Vollmer, Eds., *Complexity of Constraints: An Overview of Current Research Themes*, ser. Lecture Notes in Computer Science. Springer-Verlag Berlin Heidelberg, 2008, vol. 5250.
- [18] I. Adler and M. Weyer, "Tree-width for first order formulae," in *Computer Science Logic*, E. Grädel and R. Kahle, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2009, pp. 71–85.
- [19] J. J. Dekker, G. Björndal, M. Carlsson, P. Flener, and J.-N. Monette, "Auto-tabling for subproblem presolving in MiniZinc," *Constraints*, vol. 22, no. 4, pp. 512–529, October 2017.
- [20] Ö. Akgün, I. P. Gent, C. Jefferson, I. Miguel, P. Nightingale, and A. Z. Salamon, "Automatic discovery and exploitation of promising subproblems for tabulation," in *Principles and Practice of Constraint Programming*, ser. Lecture Notes in Computer Science, J. Hooker, Ed., vol. 11008. Springer, 2018, pp. 3–12.