Theoretical Computer Science 942 (2023) 276–296

Contents lists available at ScienceDirect

Theoretical Computer Science

journal homepage: www.elsevier.com/locate/tcs

Tractability beyond β-acyclicity for conjunctive queries with negation and SAT ☆

Matthias Lanzinger *

University of Oxford, Wolfson Building, Parks Road, Oxford OX1 3AQ, United Kingdom

ARTICLE INFO

Article history:
Received 28 April 2022
Received in revised form 21 November 2022
Accepted 4 December 2022
Available online 7 December 2022
Communicated by S. Saurabh.

Keywords:
Conjunctive queries with negation
Beta-acyclic
Hypergraph
Satisfiability
Sat

ABSTRACT

Numerous fundamental database and reasoning problems are known to be NP-hard in general but tractable on instances where the underlying hypergraph structure is β-acyclic. Despite the importance of many of these problems, there has been little success in generalizing these results beyond acyclicity.

In this paper, we take on this challenge and propose *nest-set width*, a novel generalization of hypergraph β-acyclicity. We demonstrate that nest-set width has desirable properties and algorithmic significance. In particular, evaluation of boolean conjunctive queries with negation ($CQ^\neg$) is tractable for classes with bounded nest-set width. Furthermore, propositional satisfiability (SAT) is fixed-parameter tractable when parameterized by nest-set width.

1. Introduction

Hypergraph cyclicity has been identified as a key factor for the computational complexity of multiple fundamental database and reasoning problems. While various natural notions of hypergraph acyclicity exist, the two most general ones — α- and β-acyclicity — have proven to be the most relevant in the study of the complexity of reasoning. Important problems that are NP-hard in general often become tractable when restricted to acyclic instances. An example from databases is the evaluation of conjunctive queries (CQs), which is NP-hard in general but becomes tractable when the underlying hypergraph structure of the query is α-acyclic [30]. The restriction to β-acyclic instances yields tractable classes for a variety of fundamental problems, including SAT [26], #SAT [6], and $CQ^\neg$ evaluation [4,25]. Notably, these problems remain NP-hard when restricted to α-acyclic instances, or #P-hard in the case of #SAT [7].

However, while both types of acyclicity have proven to be interesting, the generalization of α-acyclicity has received significantly more attention. There, a rich hierarchy of width measures has been developed over the last two decades. The *tree-likeness* of α-acyclic hypergraphs has been successfully generalized by hypertree width [16] and even further to fractional hypertree width [19]. What makes these generalizations particularly interesting is that they remain sufficient conditions for tractable CQ evaluation, emphasizing the deep connection between cyclicity and the complexity of CQs. The yet more general submodular width [24] characterizes the fixed-parameter tractability of CQ evaluation on the hypergraph level. Moreover, related research has revealed notable parallels to other fields, e.g., to game theory [15] and information theory [2,22].

☆ This article belongs to Section A: Algorithms, automata, complexity and games, Edited by Paul Spirakis.

* Corresponding author.
E-mail address: matthias.lanzinger@cs.ox.ac.uk.

https://doi.org/10.1016/j.tcs.2022.12.002

Despite the unquestionable success of the generalization of α-acyclicity, the generalization of β-acyclicity has received little attention so far. In the most prominent approach, Gottlob and Pichler [17] introduced β-hypertree width (β-hw) as an analogue to hypertree width. In particular, they define β-hw as the maximum hypertree width over all subhypergraphs, mirroring a characterization of β-acyclicity in terms of every subhypergraph being α-acyclic. However, it is difficult to exploit low β-hw algorithmically. An inherent problem with β-hw is that a witness for low β-hw would need to include a hypertree decomposition for each of the, exponentially many, subhypergraphs. Furthermore, none of the problems listed above as tractable on β-acyclic instances are known to be tractable for bounded β-hw (beyond those that are tractable for the more general bounded hw).

In recent work, Carbonell, et al. [8] introduced point-decompositions and the accompanying point-width (pw), which generalizes both β-acyclicity and the recently proposed maximum induced matching width [28]. They show that, given a point-decomposition of bounded point-width and polynomial size, Max-CSP can be decided in polynomial time. However, just as with β-hw, it is not known if $pw \leq k$ can be decided in polynomial time, even for constant k. In summary, neither of these approaches allows us to fully extend the tractability under β-acyclicity for the problems mentioned above to larger tractable fragments. Considering the importance of the affected problems and the restrictiveness of β-acyclicity the situation is unsatisfactory from a theoretical and practical perspective.

In this paper, we propose a novel generalization of β-acyclicity which we call nest-set width (nsw). In contrast to β-hw and pw, it is not based on decompositions but instead generalizes a characterization of β-acyclicity by the existence of certain kinds of elimination orders. Nest-set width has several attractive properties that suggest it to be a natural extension of β-acyclicity. Importantly, $nsw \leq k$ can be decided in fixed-parameter tractable time when parameterized by k. Furthermore, we show that bounded nsw yields new islands of tractability for SAT and CQ$^\neg$ evaluation. The full contributions of this paper are summarized as follows:

- We introduce a new hypergraph width notion – nest-set width – that generalizes the existence of nest point elimination orders.
- We establish the relationship of nsw to other related widths. In particular, we show that bounded nsw is a special case of bounded β-hw and incomparable to other prevalent width measures such as bounded clique width and treewidth.
- It is shown that deciding $nsw \leq k$ is NP-complete when k is part of the input but fixed-parameter tractable when parameterized by k.
- Building on work by Brault-Baron [4] for the β-acyclic case, we show the tractability of evaluation of boolean CQs with negation for classes with bounded nsw.
- Finally, we demonstrate how to derive the fixed-parameter tractability of SAT parameterized by nsw from our results for CQs with negation, as well as directly via standard Davis-Putnam resolution [11].

Structure The rest of the paper is structured as follows. Section 2 introduces necessary notation and preliminaries. We define nest-set width and establish some basic properties in Section 3. We move on to establish the relationship between nsw and other width measures, most importantly β-hw, in Section 4. The complexity of checking nsw is discussed in Section 5. The tractability of CQ$^\neg$ under bounded nsw is shown in Section 6. The fixed-parameter tractability of SAT parameterized by nsw is proved in Section 7. We end with concluding remarks and a discussion of future work in Section 8.

2. Preliminaries

For positive integers n we will use $[n]$ as a shorthand for the set $\{1, 2, \ldots, n\}$. When X is a set of sets we sometimes write $\bigcup X$ for $\bigcup_{x \in X} x$. The same applies analogously to intersections.

Linear orders will play an important role throughout this paper. Recall, a binary relation R is a *linear order* if it is antisymmetric, transitive and connex (either aRb or bRa for all a and b). We will be particularly interested in whether the subset relation $\subseteq$ is a linear order on some domain. If $\subseteq$ is a linear order for some set of sets X, we say that X is *linearly ordered* by $\subseteq$, or that X is $\subseteq$*-ordered*. We use the same terminology for variations of the $\subseteq$ relation. Note that $\subseteq$ is inherently transitive and antisymmetric and we will only ever consider whether $\subseteq$ is connex for some set of sets X.

We use standard notions from (parameterized) computational complexity theory such as reductions and the classes P and NP. In particular, we recall that the class of *fixed-parameter tractable* problems refers to those problems that can be solved in time $f(k)poly(x)$ where f is a computable function, k the parameter and x is the input size. We refer to [27] and [10] for comprehensive overviews of computational complexity and parameterized complexity, respectively. Furthermore, we assume the reader to be familiar with propositional logic.

2.1. Hypergraphs, acyclicity & width

A *hypergraph* H is a pair $(V(H), E(H))$ where $V(H)$ is a set of *vertices* and $E(H) \subseteq 2^{V(H)}$ is a set of *hyperedges*. For hypergraph H and vertex v, we denote the set of incident edges of v as $I(v, H) := \{e \in E(H) \mid v \in e\}$. The notation is extended to sets of vertices $s = \{v_1, \ldots, v_\ell\}$ as $I(s, H) := \bigcup_{i=1}^{\ell} I(v_i, H)$. We say an edge $e \in I(s, H)$ is *incident* to the set s. If H is clear from the context we drop H in the argument and write only $I(s)$.

A *subhypergraph* H' of H is a hypergraph with $E(H') \subseteq E(H)$ and $V(H') = \bigcup E(H')$. The *vertex induced subhypergraph* $H[U]$ of H is the hypergraph with $V(H[U]) = U$ and $E(H[U]) = \{e \cap U \mid e \in E(H)\} \setminus \{\emptyset\}$. For a set of vertices X we write $H - X$ as shorthand for the vertex induced subhypergraph $H[V(H) \setminus X]$.

All common notions of hypertree acyclicity have numerous equivalent definitions (see e.g., [13,5]). Here we recall only those definitions that are necessary to present the results of this paper.

A *join tree* of H is a pair (T, ϵ) where T is a tree and $\epsilon : T \to E(H)$ is a bijection from the nodes in T to the edges of H such that the following holds: for every $v \in V(H)$ the set $\{u \in T \mid v \in \epsilon(u)\}$ is a subtree of T. If H has a join tree, then we say that H is *α-acyclic*.

A (weak) *β-cycle* is a sequence $(e_1, v_1, e_2, \ldots, v_{n-1}, e_n, v_n, e_{n+1})$ with $n \geq 3$ where $e_1, \ldots, e_n$ are distinct hyperedges, $e_1 = e_{n+1}$, and $v_1, \ldots, v_n$ are distinct vertices. Moreover, for all $i \in [n]$, v_i is in e_i and e_{i+1} and not in any other edge of the sequence. A hypergraph is *β-acyclic* if it has no β-cycle.

An alternative (equivalent) definition of β-acyclicity is that H is β-acyclic if and only if all subhypergraphs of H are α-acyclic. In this paper, a third characterization of β-acyclicity will be important. We call a vertex v of H a *nest-point* if $I(v)$ is linearly ordered by $\subseteq$. We can then characterize β-acyclicity by a kind of elimination order for nest-points (this will be made more precise for a more general case in Definition 2).

Proposition 1 *([12]). A hypergraph H is β-acyclic if and only if the empty hypergraph can be reached by successive removal of nest-points and empty-edges from H.*

Join trees have been successfully generalized to hypertree decompositions. A *hypertree decomposition* [16] of a hypergraph H is a tuple $\langle T, (B_u)_{u \in T}, (\lambda_u)_{u \in T} \rangle$, where T is a rooted tree, for every node u of the tree, $B_u \subseteq V(H)$ is called the *bag* of node u, and $\lambda_u \subseteq E(H)$ is the *cover* of u. Furthermore, $\langle T, (B_u)_{u \in T}, (\lambda_u)_{u \in T} \rangle$ must satisfy the following properties.

1. The subgraph $T_v = \{u \in T \mid v \in B_u\}$ for vertex $v \in V(H)$ is a tree.
2. For every $e \in E(H)$ there exists a $u \in T$ such that $e \subseteq B_u$.
3. For every node u in T it holds that $B_u \subseteq \bigcup \lambda_u$.
4. Let T_u be the subtree of T rooted at node u and let $B(T_u)$ be the union of all bags of nodes in T_u. For every node u in T it holds that $\bigcup \lambda_u \cap B(T_u) \subseteq B_u$.

The first property is commonly referred to as the *connectedness condition* and the fourth property is called the *special condition*. The *hypertree width* (hw) of a hypertree decomposition is $\max_{u \in T}(|\lambda_u|)$ and the hypertree width of H ($hw(H)$) is the minimal width of all hypertree decompositions of H.

If we exclude the special condition in the above list of properties, we obtain the definition of a *generalized hypertree decomposition*. The *generalized hypertree width* of hypergraph H ($ghw(H)$) is defined analogously to before as the minimal width of all generalized hypertree decompositions of H.

It is known that $hw(H) = 1$ if and only if H is α-acyclic [16]. Analogous to the definition of β-acyclicity in terms of every subhypergraph being α-acyclic, Gottlob and Pichler [17] introduced *β-hypertree width* β-hw $(H) = \max\{hw(H') \mid H'$ is a subhypergraph of $H\}$. Note that we therefore also have β-hw $(H) = 1$ if and only if H is β-acyclic.

2.2. Conjunctive queries

A *signature* σ is a finite set of relation symbols with associated arities. We write $ar(R)$ for the arity of relation symbol R. A *database* D (over signature σ) consists of a *finite* domain Dom and a relation R^D for each relation symbol R in the signature.

A *conjunctive query with negation* (over signature σ) is a set of literals. A literal is of the form $L(v_1, \ldots, v_m)$ where $v_1, \ldots, v_m$ are variables and L is either R or $\neg R$ for any m-ary relation symbol R in σ. If L is of the form R we call the literal *positive*, otherwise, if it is of the form $\neg R$ we say that the literal is *negative*. We commonly refer to a CQ$^\neg$ simply as *query*. We write $vars(q)$ for the set of all variables that occur in the literals of query q. We sometimes denote queries like logical formulas, i.e., $R_1(\vec{v_1}) \wedge \cdots \wedge R_n(\vec{v_n})$ with the understanding that the query is simply the set of all conjuncts.

Let q be a query and D a database over the same signature. We call a function $a\colon vars(q) \to Dom$ an *assignment* for q. For a set of variables X we write $a[X]$ for the assignment with domain restricted to X. In a slight abuse of notation we also write $a[\vec{v}]$ for the tuple $(a(v_1), \ldots, a(v_n))$ where $\vec{v} = (v_1, \ldots, v_n)$ is a sequence of variables. An *extension* of an assignment $a\colon Vars \to Dom$ is an assignment $a'\colon Vars' \to Dom$ with $Vars' \supset Vars$ and $a(v) = a'(v)$ for every variable $v \in Vars$.

We say that the assignment *a satisfies a positive literal* $R(\vec{v})$ if $a[\vec{v}] \in R^D$. Similarly, *a satisfies a negative literal* $\neg R(\vec{v})$ if $a[\vec{v}] \notin R^D$. An assignment *satisfies a query* q (over database D) if it satisfies all literals of q. We write $q(D)$ for the set of all satisfying assignments for q over D. We can now define the central decision problem of this paper.

BOOLCQ$^\neg$	
Instance:	A CQ$^\neg$ q and a database D
Question:	$q(D) \neq \emptyset$?

A query q has an associated hypergraph $H(q)$. The vertices of $H(q)$ are the variables of q. Furthermore, $H(q)$ has an edge $\{v_1, \dots, v_n\}$ if and only if there exists a literal $R(v_1, \dots, v_n)$ or $\neg R(v_1, \dots, v_n)$ in q.

To simplify later arguments we will assume that every relation symbol occurs only once in a query. We will therefore sometimes write the relation symbol, without the variables, to identify a literal. Note that every instance of $\texttt{BoolCQ}^\neg$ can be made to satisfy this property: for each literal L with relation symbol R create an copy of relation R^D that is particular to this literal (and rename the relation symbol in L accordingly). This procedure requires at most $O(|q| \ \|D\|)$ time (see below for the definition of $\|D\|$) and increases the size of the database at most by factor $|q|$.

Finally, for our algorithmic considerations we assume a reasonable representation of queries and databases. In particular we assume that a relation R has a representation of size $\|R\| = O(|R| \cdot ar(R) \cdot \log Dom)$. Accordingly, we assume the *size of a database* D as $\|D\| = \|Dom\| + \sum_{R \in \sigma} \|R\|$ and the *size of a query* q as $\|q\| = O\left(\sum_{R \in \sigma} ar(R) \log |vars(q)|\right)$. We refer to the cardinality of the largest relation in D as $|R_{max}(D)| = \max_{R \in \sigma} |R^D|$. When the database is clear from the context we write just $|R_{max}|$.

3. Nest-set width

In this section we introduce nest-set width and establish some of its basic properties. The crucial difference between β-hw and *nsw* is that the generalization is based on a different characterization of β-acyclicity. While β-hw generalizes the condition of every subgraph having a join tree, nest-set width instead builds on the characterization via nest point elimination from Proposition 1. We start by generalizing nest points to *nest-sets*:

Definition 1 *(Nest-set).* Let H be a hypergraph. A non-empty set $s \subseteq V(H)$ of vertices is called a *nest-set* in H if the set

$$I^*(s, H) := \{e \setminus s \mid e \in I(s, H)\}$$

is linearly ordered by $\subseteq$.

As the comparability by $\subseteq$ of sets minus a nest-set will appear frequently, we introduce explicit notation for it. Let H be a hypergraph and $s \subseteq V(H)$. For two sets of vertices $V, U \subseteq V(H)$, we write $V \subseteq_s U$ for $V \setminus s \subseteq U \setminus s$. We could thus alternatively define nest-sets as those sets s for which $I(s, H)$ is linearly ordered by $\subseteq_s$.

In later sections, the maximal elements with respect to $\subseteq_s$ will play an important role. For a nest-set s we will refer to a maximum edge in $I(s)$ w.r.t. $\subseteq_s$ as a *guard* of s. Note that there may be multiple guards. However, in all of the following usage it will make no difference which guard is used and we will implicitly always use the lexicographically first one (and thus refer to *the* guard).

Like for nest points, we want to investigate how a hypergraph can be reduced to the empty hypergraph by successive removal of nest sets. We formalize this notion in the form of *nest-set elimination orderings*.

Definition 2 *(Nest-set elimination ordering).* Let H be a hypergraph and let $\mathcal{O} = (s_1, \dots, s_q)$ be a sequence of sets of vertices. Define $H_0 = H$ and $H_i := H_{i-1} - s_i$. We call $\mathcal{O}$ a *nest-set elimination ordering* (NEO) if, for each $i \in [q]$, s_i is a nest-set of H_{i-1} and H_q is the empty hypergraph.

Note that an elimination ordering is made up of at most $|V(H)|$ nest-sets. We are particularly interested in how large the nest-sets have to be for a NEO to exist. Hence, we introduce notation for restricted-size nest-sets and NEOs:

- If s is a nest-set of H with at most k elements then we call s a *k-nest-set*.
- A nest-set elimination ordering that consists of only k-nest-sets is a *k-nest-set elimination ordering* (k-NEO).
- Finally, the *nest-set width* $nsw(H)$ of a hypergraph H is the lowest k for which there exists a k-NEO.

It is easy to see that a hypergraph has a 1-nest-set $\{v\}$ if and only if v is a nest point. Therefore, a 1-NEO corresponds directly to a sequence of nest point deletions that eventually result in the empty hypergraph. As this is exactly the characterization of β-acyclicity from Proposition 1, we see that *nsw* generalizes β-acyclicity.

Corollary 2. *A hypergraph H has $nsw(H) = 1$ if and only if H is β-acyclic.*

Example 3.1. Let H_0 be the hypergraph with edges $\{a, b, c, d\}$, $\{a, d, e\}$, $\{c, d, f\}$, $\{b, e\}$, and $\{c, f\}$. Fig. 1 illustrates the stepwise elimination of H_0 according to the 2-NEO $(\{c, f\}, \{b, e\}, \{a, d\})$.

For the first nest-set $s_1 = \{c, f\}$ we see that $I(s_1, H_0) = \{\{a, b, c, d\}, \{c, d, f\}, \{c, f\}\}$ and $I^*(s_1, H_0) = \{\{a, b, d\}, \{d\}, \emptyset\}$. To verify that s_1 is a nest-set of H_0 we observe that $\{a, b, d\} \supseteq \{d\} \supseteq \emptyset$. Note that $\{f\}$ is also a nest-set of H_0 whereas $\{c\}$ is not since $\{a, b, d\}$ and $\{d, f\}$ are both in $I^*(\{c\}, H_0)$ and clearly neither $\{a, b, d\} \subseteq \{d, f\}$ nor $\{a, b, d\} \supseteq \{d, f\}$ holds.

In the second step of the elimination process we then consider $H_1 = H_0 - \{c, f\}$ and the nest-set $s_2 = \{e, b\}$. It is again straightforward to verify that $I^*(s_2, H_1) = \{\{a, d\}, \emptyset\}$ is $\subseteq$-ordered. This is in fact the only nest-set of H_1. The third nest-set

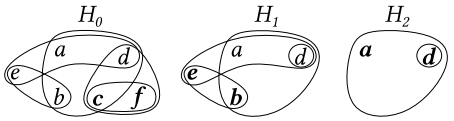

Fig. 1. The nest-set elimination from Example 3.1.

in the NEO, $s_3 = \{a, d\}$ only becomes a nest-set after elimination of s_2: observe that $I^*(s_3, H_1) = \{\{e\}, \{b\}, \emptyset\}$ which is not $\subseteq$-ordered.

In the final step, $H_2 = H_1 - \{e, b\}$ only has two vertices left. The set of all vertices of a hypergraph is trivially a nest-set since $I^*(V(H), H)$ is always $\{\emptyset\}$. Thus, the set $V(H_2) = \{a, d\}$ is a nest-set of H_2. The hypergraph H_0 has no 1-NEO (it has a β-cycle) and therefore $nsw(H_0) = 2$.

An important difference between α- and β-acyclicity is that only the latter is *hereditary*. In particular, if hypergraph H is β-acyclic then so is every subhypergraph of H. In general we say that a width measure w is hereditary if $w(H') \leq w(H)$ for every subhypergraph H of H. Nest-set width, just like β-acyclicity and β-hypertree width, is indeed also a hereditary property. In contrast, hypertree width is not hereditary. In the following two simple but important lemmas, we first establish that NEOs remain valid when vertices are removed from the hypergraph (and the NEO) and then show that this also applies to removing edges.

Note that the construction in the following lemma, and Lemma 4 below, can technically create empty sets in the resulting NEOs. Formally speaking this is not allowed (recall that nest-sets are non-empty). Whenever this occurs the implicit meaning is that all the empty sets are removed from the NEO.

Lemma 3. *Let H be a hypergraph with k-NEO $\mathcal{O} = (s_1, s_2, \ldots, s_\ell)$ and let $r \subseteq V(H)$. Then the sequence $\mathcal{O}' = (s_1 \setminus r, s_2 \setminus r, \ldots, s_\ell \setminus r)$ is a k-NEO of $H - r$.*

Proof. We first show that for any nest-set s we have that $s \setminus r$ is either the empty set or a nest-set of $H - r$.

Suppose $s \setminus r$ is not empty and not a nest-set of $H - r$, then there are $e_1, e_2 \in I(s \setminus r, H - r)$ that are not comparable by $\subseteq_{s \setminus r}$. It is easy to see that there exist $e_1', e_2' \in I(s, H)$ such that $e_1 = e_1' \setminus r$ and $e_2 = e_2' \setminus r$. Since s is a nest-set in H, w.l.o.g., $e_1' \setminus s \subseteq e_2' \setminus s$ and therefore also

$$e_1 \setminus (s \setminus r) = e_1' \setminus (s \cup r) \subseteq e_2' \setminus (s \cup r) = e_2 \setminus (s \setminus r)$$

and we arrive at a contradiction.

It follows that $s_1 \setminus r$ is a k-nest-set of $H - r$. Since $\mathcal{O}$ is a NEO, s_2 must be a nest-set of $H - s_1$. Now, to verify $\mathcal{O}'$ we need to show that $s_2 \setminus r$ is a k-nest-set of $H - r - s_1$. However, this is clearly the same hypergraph as $(H - s_1) - r$ and the above observation applies again. We can repeat this argument for all s_i until s_ℓ and thus $\mathcal{O}'$ is a k-NEO. □

Lemma 4. *Let H be a hypergraph with k-NEO $\mathcal{O} = (s_1, \ldots, s_\ell)$. Let H' be a connected subhypergraph of H and $\Delta = V(H) \setminus V(H')$ the set of vertices no longer present in the subhypergraph. Then the sequence $(s_1 \setminus \Delta, s_2 \setminus \Delta, \ldots s_\ell \setminus \Delta)$ is a k-NEO of H'.*

Proof. From the argument at the beginning of the proof of Lemma 3 we know that $s \setminus \Delta$ is empty or a nest-set of $H - \Delta$. Therefore, $I^*(s \setminus \Delta, H - \Delta)$ is $\subseteq$-ordered. Now, since H' does not contain any vertices from Δ and is a subhypergraph of H we have $E(H') = E(H' - \Delta) \subseteq E(H - \Delta)$ and thus $I^*(s \setminus \Delta, H') \subseteq I^*(s \setminus \Delta, H - \Delta)$. Therefore $I^*(s \setminus \Delta, H')$ is $\subseteq$-ordered and thus $s \setminus \Delta$ is a nest-set. This observation can again be iterated along the NEO in the same fashion as in the proof of Lemma 3 to prove the statement. □

4. Nest-set width vs. β-hypertree width

A wide variety of hypergraph width measures have been studied in the literature. To provide some context for the later algorithmic results, we will first investigate how *nsw* relates to a number of prominent width notions from the literature. In particular, in this section we show that *nsw* is a specialization of β-hypertree width and incomparable to primal and incidence clique width and treewidth. The relationship to β-hypertree width is of particular interest since bounded β-hw also generalizes β-acyclicity. The section is structured around proving the following theorem.

Theorem 5. *Bounded nsw is a strictly less general property than bounded β-hw. In particular, the following two statements hold:*

1. *For every hypergraph H we have $\beta\text{-hw}(H) \leq 3\, nsw(H) + 1$.*
2. *There exists a class of hypergraphs with bounded β-hw and unbounded nsw.*

We begin by establishing a useful technical lemma that will eventually lead us to the second statement of Theorem 5. An important consequence of the following Lemma 6 is that the length (minus 1) of the longest β-cycle of H is a lower bound of $nsw(H)$ since any vertex in a cycle has to be removed at some point in any NEO.

Lemma 6. *Let $C = (e_1, v_1, e_2, v_2 \ldots, e_\ell, v_\ell, e_{\ell+1})$ be a β-cycle in a hypergraph H. For every nest-set s of H we have that $|s \cap \{v_1, \ldots, v_\ell\}|$ is either 0, or at least $\ell - 1$.*

Proof of Lemma 6. Suppose the cardinality of $s \cap \{v_1, \ldots, v_\ell\}$ is not 0. That is, at least one vertex of C is in s. Since we can rotate the indices of a cycle arbitrarily we assume, w.l.o.g., that $v_1 \in s$. Then, e_2 and $e_{\ell+1}$ are both in $I(s)$. Recall that a β-cycle has $\ell \geq 3$ and that v_2 can occur only in e_2 and e_3 and no other edges of the cycle. Similarly, v_ℓ can occur exclusively in e_ℓ and $e_{\ell+1}$ and no other edges in C. We therefore see that $v_2 \notin e_{\ell+1}$ and $v_\ell \notin e_2$. Thus, e_2 and $e_{\ell+1}$ can only be comparable by $\subseteq_s$ if at least one of v_2 or $v_{\ell+1}$ is in s.

Suppose, w.l.o.g., $v_2 \in s$, then we have e_3 and $e_{\ell+1}$ in $I(s)$ and the same argument can be applied again, as long as the two edges are not adjacent in the cycle. The argument can be iterated exhaustively until all edges of the cycle are in $I(s)$ and at least $\ell - 1$ vertices are necessarily in s. □

Lemma 6 further emphasizes the aforementioned distinction between generalizing acyclicity in sense of tree-likeness and our approach. Any cycle graph C_n has hypertree width 2 whereas the lemma shows us that $nsw(C_n) \geq n - 1$ since any nest-set will contain at least one vertex of the cycle, so it must contain at least $n - 1$ of them. Furthermore, cycle graphs have clique width at most 4 [9] and treewidth at most 2. We therefore arrive at the following lemma.

Lemma 7. *There exists a class of hypergraphs that has bounded β-hw, treewidth, and clique width and unbounded nsw.*

The lemma establishes the second statement of Theorem 5. We can derive some further results by combining Lemma 7 with results from [17]. There it was shown that there exist classes of β-acyclic hypergraphs that have unbounded clique width and treewidth.[1] In combination with the previous lemma this demonstrates that bounded clique width and bounded treewidth are incomparable to bounded *nsw*. The results in [17] also apply to incidence clique width and incidence treewidth and since the incidence graph of a cycle graph is also a cycle graph, so does Lemma 7. Thus, bounded *nsw* is also incomparable to bounded incidence clique width and bounded incidence treewidth. The resulting hierarchy is summarized in Fig. 2 at the end of this section.

We move on to show that β-hw $(H) \leq 3\,nsw(H) + 1$. We will give a procedure to construct a generalized hypertree decomposition of width k from a k-NEO. To construct such a decomposition we first observe that a nest-set is connected to the rest of the hypergraph via its *guard* edge. We show that the guard edges of a NEO induce a certain kind of small separators (which we call an *exhaustive hinge*) that can be used to construct the desired decomposition. By a result of Adler, Grohe, and Gottlob in [1] we have that $hw(H) \leq 3\,ghw(H) + 1$. Since k-NEOs are hereditary, every subhypergraph H' of H will also have a generalized hypertree decomposition of width k and therefore $hw(H') \leq 3\,ghw(H) + 1$. From this it is then straightforward to obtain the stated bound of β-hw $(H) \leq 3k + 1$. The necessary details of this observation are captured by the following two definitions and the key Lemma 8 below. The following construction is inspired by the hinge decompositions of Gyssens, Jeavons, and Cohen [20].

Definition 3 *(Exhaustive subhypergraphs).* Let H be a hypergraph and $E' \subseteq E(H)$. Let $E^* := \{e \in E(H) \mid e \subseteq \bigcup E'\}$ be the edges covered by E'. Then we call the subhypergraph H' with $E(H') = E(H) \setminus E^*$ the *exhaustive E'-subhypergraph* of H.

We use the term *connected exhaustive E'-subhypergraphs* of H to refer to the connected components of H' (considering each component as an individual hypergraph).

We use exhaustive subhypergraphs to express that, when we remove a set of edges E' from H, then we also want to remove the edges E^* that are covered by $\bigcup E'$. The following construction of a hypertree decomposition from a NEO will use sets of the form $\bigcup E'$ as its bags. This means that the respective bag also covers all edges in E^*. We are therefore interested in the components resulting from removing all of E^* instead of just E' from H.

In particular, we want to remove sets of edges E' in such a way that the exhaustive E'-subhypergraphs are all connected to E' via a single edge. This will allow us to bring together the decompositions of the subhypergraphs in a way that preserves all properties of hypertree decompositions.

Definition 4 *(Exhaustive hinges).* Let H be a hypergraph, $E' \subseteq E(H)$ and $C_1, \ldots, C_n$ the connected exhaustive E'-subhypergraphs of H. For an $e \in E(H)$ we say that E' is an *exhaustive e-hinge* if for every $i \in [n]$ we have that $V(C_i) \cap \bigcup E' \subseteq e$.

[1] Note that our setting allows for hyperedges of unbounded size. In settings where the hypergraph is derived from a query that adheres to some fixed schema (which bounds the size of hyperedges), this does not hold and a class of β-acyclic hypergraphs always has bounded treewidth.

Lemma 8. *Let s be a k-nest-set of hypergraph H and let e_g be the guard of s. Then there exists an exhaustive e_g-hinge $E' \subseteq E(H)$ with the following properties:*

1. $\bigcup I(s, H) \subseteq \bigcup E'$
2. $|E'| \leq k$

Proof. Let s and e_g be as in the statement. Let λ be a minimal edge cover of $s \setminus e_g$. Observe that $|s \setminus e_g| < k$ as e_g is incident to s and therefore $|\lambda| < k$. We now claim that $E' = \lambda \cup \{e_g\}$ is the required hinge. Clearly we have $|E'| \leq k$. For the first property, recall that for every $e \in I(s, H)$ we have $e \setminus s \subseteq e_g$ and thus also $e \subseteq e_g \cup s$. It is then easy to see from the definition of E' that $e_g \cup s \subseteq \bigcup E'$ and the property follows.

What is left to show is that E' is in fact an exhaustive e_g-hinge. Let C be one of the connected exhaustive E'-subhypergraphs of H and partition the set $V(C) \cap \bigcup E'$ in two parts: $I_1 := V(C) \cap s$ and $I_2 := V(C) \cap \left(\left(\bigcup E'\right) \setminus s\right)$.

First we argue that $I_1 = \emptyset$. It was already established that $\bigcup I(s, H) \subseteq \bigcup E'$, thus every edge incident to s is removed in the exhaustive E'-subhypergraph. It is therefore impossible for a vertex of s to be in $V(C)$.

Second, observe that by construction every edge in E' is incident to s and by definition of the guard of s we thus have $\left(\left(\bigcup E'\right) \setminus s\right) \subseteq e_g$. It follows immediately that $I_1 \cup I_2 \subseteq e_g$ and the statement holds. □

Lemma 8 is the key lemma for our construction procedure. It tells us that we can always find a *small* exhaustive hinge E' in a hypergraph H if H has a k-NEO. By the first property from the lemma, the exhaustive E' subhypergraph no longer contains the vertices s. From the connected exhaustive E'-subhypergraphs we can construct subhypergraphs of H that connect to E' via a single edge and have shorter k-NEOs than H. Since the subhypergraphs are connected to E' via a single edge, it is straightforward to combine individual hypertree decompositions for every subhypergraph into a new decomposition for H. This step can then be applied inductively on the length of the k-NEO to construct a hypertree decomposition of width k for H.

Lemma 9. *For any hypergraph H it holds that $ghw(H) \leq nsw(H)$.*

Proof. We show by induction on $\ell \geq 1$ that if a hypergraph H has a k-NEO of length ℓ then it has a generalized hypertree decomposition of width at most k. For the base case, $\ell = 1$, the NEO consists of a single nest-set $s = V(H)$ with $|s| \leq k$. The base case then follows from the straightforward observation that $ghw(H) \leq |V(H)|$.

Suppose the statement holds for $\ell' < \ell$. We show that it also holds for every k-NEO of length ℓ. Let $\mathcal{O} = (s_1, \ldots, s_\ell)$ be a k-NEO of H. Let e_g be the guard of s_1 and let E' be the exhaustive e_g-hinge from Lemma 8 and let $C'_1, \ldots, C'_n$ be the connected exhaustive E'-subhypergraphs. Finally, for each $i \in [n]$, we add e_g to C'_i to obtain the hypergraph C_i.

By Lemma 4 we see that for each $i \in [n]$, C_i has a k-NEO $\mathcal{O}_i = (s_{1,i}, \ldots, s_{\ell,i})$ since it is a subhypergraph of H. Furthermore, according to Lemma 4, we can assume an $\mathcal{O}_i$ such that $s_{1,i} \subseteq s_i$ and, since e_g in C_i, also $s_{1,i} \neq \emptyset$.

Therefore, $C_i - s_{1,i}$ has a k-NEO of length at most $\ell - 1$ and we can apply the induction hypothesis to obtain a generalized hypertree decomposition $\langle T_i, (B_{u,i})_{u \in T_i}, (\lambda_{u,i})_{u \in T_i} \rangle$ of $C_i - s_{1,i}$ with width at most k. Observe that the hypergraph has an edge $e_g \setminus s_{1,i}$ which has to be covered fully by some node $u_{g,i}$ in T_i.

Let u be a fresh node with $B_u = \bigcup E'$ and $\lambda_u = E'$. For each $i \in [n]$, reroot the tree T_i such that the root becomes $u_{g,i}$ and then attach the tree as a child of u. A cover $\lambda_{w,i}$ of a node w in T_i can contain an edge e' that is not in H because e' is the result of the removal of vertices $s_{1,i}$ from some edge in $E(H)$, i.e., $e' = e \setminus s_{1,i}$ for some $e \in E(H)$. To fix such inconsistencies in the newly constructed decomposition for H we can simply replace any such e' in $\lambda_{w,i}$ by the respective edge e and $B_{w,i}$ remains covered by $\lambda_{w,i}$.

We claim that this newly built decomposition is a generalized hypertree decomposition of H with *ghw* at most k. It is not difficult to verify that this new structure indeed satisfies all proprieties of a generalized hypertree decomposition.

Connectivity Each subtree below the root already satisfies connectivity. The tree structure and the bags in the subtree remains unchanged. Furthermore, by construction of the hypergraphs C_i, the sets $B(T_i)$ of vertices occurring in bags of the tree T_i are pairwise disjoint except for the vertices in e_g. Since e_g is fully in B_u the only issue for connectivity can arise if there is a vertex in $B_u \cap B(T_i)$ but not in $B_{u_{g,i}}$. We argue that this is impossible.

Since E' is an exhaustive e_g-hinge and e_g was added back into each component it is easy to see that

$$B_u \cap B(T_i) = \bigcup E' \cap V(C_i - s_{1,i}) \subseteq e_g \setminus s_{1,i}$$

The rightmost term is exactly the edge that informed our choice of $u_{g,i}$, i.e., we have $B_u \cap B(T_i) = e_g \setminus s_{1,i} \subseteq B_{u_{g,i}}$ by construction.

Every edge of H is covered For every edge $e \in E(H)$ we consider two cases. Either $e \in I(s_1, H)$ or not. In the first case, we have by Lemma 8 that $e \subseteq \bigcup E'$ and therefore e is covered in the root node u. In the second case, $e \notin I(s_1, H)$, e will occur unchanged in one of the hypergraphs $C_i - s_{1,i}$ since the removal of $s_{1,i}$ does not affect it (recall $s_{1,i} \subseteq s_i$). Since the tree decomposition of $C_i - s_{1,i}$ is attached unchanged to u (except for rerooting and fixing the λ labels), e must be covered in the respective subtree corresponding to component $C_i - s_{i,1}$. □

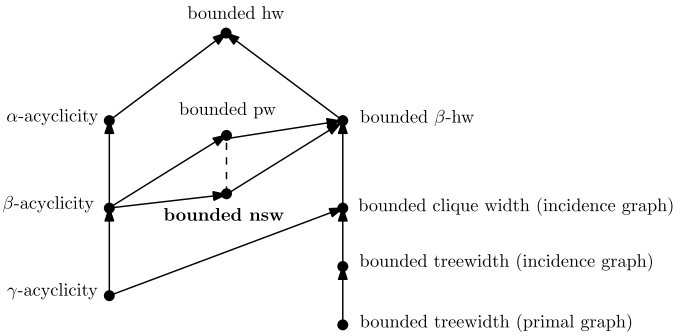

Fig. 2. Expressive power of various hypergraph properties from [17], extended by bounded *nsw* and bounded point-width (pw). Arcs are directed from less general to more general. Properties with no directed connection are incomparable. Dashed edges indicate unknown relationships.

Proof of Theorem 5 (1). By Lemma 9 we have that $ghw(H) \leq nsw(H)$. As mentioned above, we always have $hw(H) \leq 3\,ghw(H)+1$ for every hypergraph and therefore also $hw(H) \leq 3\,nsw(H)+1$. In combination with Lemma 4 we see that for every subhypergraph H' of H we have $hw(H') \leq 3\,nsw(H')+1 \leq 3\,nsw(H)+1$. □

The results of this section are summarized in Fig. 2. The diagram extends the hierarchy given in [17] by bounded *nsw*.

5. The complexity of checking nest-set width

For the existing generalizations of β-acyclicity – β-hw and pw – it is not known whether one can decide in polynomial time if a structure has width $\leq k$, even when k is a constant. This then also means that no efficient algorithm is known to compute the respective witnessing structures. In these situations, tractability results are inherently limited. One must either assume that the witnesses are given as an input or that a tractable algorithm does not use the witness at all. In comparison, deciding treewidth $\leq k$ is fixed-parameter tractable when parameterized by k [3] and checking hypertree width is tractable when k is constant [16].

When k is considered constant, it is straightforward to find a k-NEO in polynomial time, if one exists. We can simply check for all combinations of up to k vertices whether they represent a nest-set. If so, eliminate the nest-set and repeat from the beginning on the new hypergraph until it becomes empty. By Lemma 3, this greedy approach of always using the first found k-nest-set will result in a sound and complete procedure.

However, we can improve on this straightforward case by analyzing the following decision problem where k is part of the input.

Nest-Set-Width	
Instance:	A hypergraph H, integer k
Question:	$nsw(H) \leq k$?

We first observe that Nest-Set-Width is NP-complete in Section 5.1. In more positive news, we are able to show that Nest-Set-Width is fixed-parameter tractable when parameterized by k in Section 5.2. Importantly, the fixed-parameter algorithm explicitly constructs a k-NEO as a witness, if one exists, and can therefore serve as a basis for the algorithmic results in the following sections.

5.1. NP-hardness

We show NP-hardness by reduction from Vertex-Cover, a classical NP-complete problem [21]. Recall, in the Vertex-Cover problem we have as input a graph $G=(V,E)$ and an integer $k \geq 1$. The problem is to decide whether there exists a set $\alpha \subseteq V$ with $|\alpha| \leq k$ such that every edge of G is incident to at least one vertex in α. Such a set α is called a *vertex cover* of G. To simplify the following proof we make two additional assumptions on the instances of Vertex-Cover. We assume that the input graph has at least 2 edges and that k is strictly less than the number of edges in G. If either assumption is violated the problem is trivial.

The goal of this section will be to show the following theorem.

Theorem 10. Nest-Set-Width *is* NP*-complete.*

It is conceptually simpler to discuss the existence of a k-nest-set rather than a whole elimination order and for the sake of a simpler presentation we therefore first prove that it is NP-complete to decide whether a hypergraph has a k-nest-set.

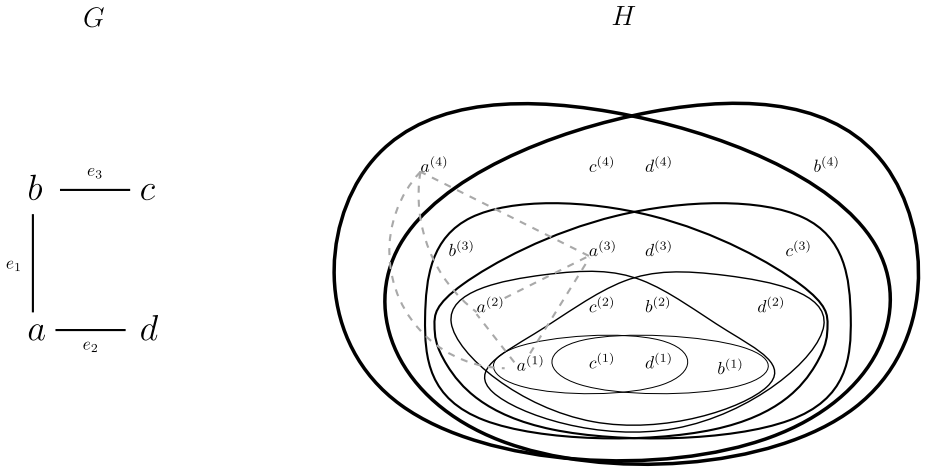

Fig. 3. Example of the from Theorem 11 for an example graph G. The linking clique is only shown for vertex a (in dashed gray edges).

Afterwards we argue that the reduction used for this problem in fact constructs a hypergraph H that has a $(km+k)$-nest-set if and only if it has nest-set width at most $km+k$.

Theorem 11. *Deciding whether a hypergraph H has a k-nest-set is* NP*-complete.*

Proof. Membership is straightforward. Guess up to k vertices s and verify that $I^*(s,H)$ is $\subseteq$-ordered. Hardness is shown by many-one reduction from Vertex-Cover. Hence, let G,k be an instance of Vertex-Cover and let n and m refer to the number of vertices and edges in G, respectively. In the following we construct a hypergraph H such that H has a $(km+k)$-nest-set if and only if G has a vertex cover of size at most k.

Let $\{e_1,\dots,e_m\}$ be the edges of G and let $\{v_1,\dots,v_n\}$ be the vertices of G. The hypergraph H will have as vertices $V(H)=\bigcup\{v_j^{(i)} \mid j\in[n], i\in[m+1]\}$, i.e., $m+1$ copies of every vertex v_j. We refer to the superscript (i) also as the ith *level* of H. We write $V^{\leq i}$ for $\{v_j^\ell \in V(H) \mid \ell \leq i\}$, i.e., all the vertices at level i or lower.

For each edge $e_i=\{a,b\}$ of G, we create two edges $f_{i,1}$ and $f_{i,2}$ in H as follows.

$$f_{i,1}=V^{\leq i}\setminus\{a^{(i)}\} \qquad f_{i,2}=V^{\leq i}\setminus\{b^{(i)}\}$$

Furthermore, we also add two edges $f_{m+1,1}=V^{\leq m+1}\setminus\{a^{(m+1)}\}$ and $f_{m+1,2}=V^{\leq m+1}\setminus\{b^{(m+1)}\}$ at the final level for $e_1=\{a,b\}$. Intuitively, these f edges at level i represent the choice between a and b for edge e_i, as one of $a^{(i)}$, $b^{(i)}$ needs to be deleted for the two edges to be comparable by $\subseteq$. We will therefore refer to them as the *choice edges*. The choice for e_1 is encoded twice, at levels 1 and $m+1$ for technical reasons (it is not of particular importance which specific edge is encoded twice). H also contains the complete graph $K^{(j)}$ over the vertices $\{v_j^{(i)} \mid i\in[m+1]\}$ for every vertex v_j of G, i.e., over every instance of v_j over all the levels. Intuitively, this links the choices at every level to each other and we therefore refer to them as the *linking cliques*. Thus we have

$$E(H)=\{f_{i,1},f_{i,2} \mid i\in[m+1]\}\cup\bigcup_{j\in[n]}E(K^{(j)})$$

Fig. 3 shows an example of our construction.

We first show that if α is a vertex cover of G and $\ell\leq k$, then $s_\alpha=\bigcup_{v_j\in\alpha}\{v_j^{(i)} \mid i\in[m+1]\}$ is a $(km+k)$-nest-set of H. Note that $|s_\alpha|\leq k(m+1)$ follows immediately from the construction.

Claim A. For each $i\in[m+1]$ we have $f_{i,1}\subseteq_{s_\alpha} f_{i,2}$ or vice versa.

Proof of claim: Let $e_i=\{a,b\}$ and, w.l.o.g., assume $a\in\alpha$ and thus also $a^{(i)}\in s_\alpha$. Clearly, $f_{i,2}\setminus\{a^{(i)}\}=V^{\leq i}\setminus\{a^{(i)},b^{(i)}\}\subseteq f_{i,1}$. Any further vertex that is removed is either in both or neither of the edges and can therefore not change the order anymore; thus $f_{i,2}\setminus s_\alpha\subseteq f_{i,1}\setminus s_\alpha$. If $b\in\alpha$ the analogous argument yields the opposite order. The same argument also applies to $f_{m+1,1}$, $f_{m+1,2}$ and e_1. △

By construction, $f_{h,1},f_{h,2}\subseteq f_{i,1},f_{i,2}$ for all $h<i$. Thus, in combination with Claim A we see that all choice edges are linearly ordered by $\subseteq_{s_\alpha}$. The only other edges in $I(s_\alpha)$ are those of the linking cliques $K^{(j)}$ where $v_j\in\alpha$. By definition of s_α we have that if $v_j\in\alpha$, then $V(K^{(j)})\subseteq s_\alpha$, and thus $I^*(s_\alpha)$ is $\subseteq$-ordered.

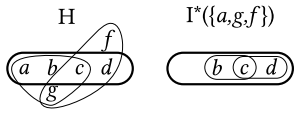

Fig. 4. Illustration of Example 5.1.

For the other direction, suppose s is a $(km+k)$-nest-set of H. We now define α as containing exactly those vertices v_j such that $v_j^{(i)} \in s$ for at least m distinct i. Note that because of the linking cliques and Lemma 6, a vertex $v_j^{(i)}$ occurs in s either for 0 or at least m distinct values of i. Using the assumptions that $2 \leq m$ and $1 \leq k < m$ from above it is straightforward to verify that $\frac{km+k}{m} < k+1$ and therefore also $|\alpha| \leq k$. Note that Lemma 6 guarantees only that m nodes of the $m+1$ vertex linking cliques are in the nest-set and we therefore add the redundant $(m+1)$-th level in the construction to simplify the argument here.

What is left is to show that every edge e_i in G is incident to a vertex in α. For any $e_i = \{a, b\}$ the choice edges $f_{i,1}, f_{i,2}$ are only comparable by $\subseteq_s$ if either $a^{(i)} \in s$ or $b^{(i)} \in s$ (or both). Then, because of the linking cliques, a or b (or both) will be in α.

It follows that if $f_{i,1}, f_{i,2} \in I(s, H)$, then e_i will be covered by α. Then, since s is not empty, there is some v_j such that $v_j^{(i)}$ occurs for m distinct i in s. In particular, then either $v_j^{(1)}$ or $v_j^{(2)}$ are in s and thus for every $2 < i \leq m+1$ we have $f_{i,1}, f_{i,2} \in I(s)$. Thus, for every edge in G there is a pair of choice edges in $I(s)$. By the argument above every edge of G is therefore incident to a vertex in α. □

We now briefly discuss how this reduction also proves Theorem 10. Suppose the same situation as in the above proof, i.e., a vertex cover instance G, k and the hypergraph H from the reduction above. If G has a vertex cover α, then we can eliminate all the vertices s_α in H that encode the graph vertices from α. By the argument above we have that $E(H) = I(s_\alpha)$ is linearly ordered by $\subseteq_{s_\alpha}$. It is not difficult to see that when all edges of a hypergraph are $\subseteq$-ordered then that hypergraph is β-acyclic: if a vertex v is included in every edge then $\{v\}$ is a nest-set. Thus, $H - s_\alpha$ has a 1-NEO $\mathcal{O}'$ and thus prepending s_α to $\mathcal{O}'$ gives us a $(km+k)$-NEO of H. On the other hand, suppose H has a $(km+k)$-NEO $\mathcal{O} = (s_1, \dots)$. As s_1 is a $(km+k)$-nest-set of H, the arguments from the proof apply and we have that G has a vertex cover of size at most k. This now also completes the proof of Theorem 10.

On a final note, one may notice similarities between finding nest-sets and an important work by Yannakakis [31] on vertex-deletion problems in bipartite graphs. Yannakakis shows a complexity characterization for vertex-deletion problems on bipartite graphs that extends to hypergraphs via their incidence graph. Furthermore, the specific problem of finding a vertex-deletion such that the edges of the hypergraph become $\subseteq$-ordered is stated to be polynomial. While this strongly resembles the nest-set problem, the results of Yannakakis are not applicable here since we are not interested in a global property of the hypergraph but only in the orderability of the edges that are incident to the deleted vertices.

5.2. Fixed-parameter tractability

Recall that every nest-set s has a maximal edge with respect to $\subseteq_s$; the guard of s. The main idea behind the algorithm presented in this section is to always fix an edge e_g and check if there exists a nest-set that specifically has e_g as its guard. This will allow us to incrementally build a nest-set s relative to the guard e_g. We first demonstrate this principle in the following example.

Example 5.1. We consider a hypergraph H with three edges $e_1 = \{a, b, c, d\}$, $e_2 = \{a, b, c, g\}$, and $e_3 = \{c, d, g, f\}$. We want to find a nest-set with guard e_1. The hypergraph with e_1 highlighted is shown in Fig. 4. To start, if s is a nest-set with guard e_1, then at least one vertex of e_1 must be in s. For this example let $a \in s$.

Since $a \in s$ we also have that $e_2 \in I(s)$. For s to be a nest-set with guard e_1 it must then hold that $e_2 \setminus s \subseteq e_1 \setminus s$. Since g is in e_2 but not in e_1 we can deduce that also $g \in s$. More generally, any vertex that occurs in an edge from $I(s)$ but not in e_1 must be part of the nest-set s. Now, since $g \in s$ it follows that $e_3 \in I(s)$ and therefore, by the previous observation, also $f \in s$.

At this point we have deduced that if a is in s, then so are g and f. We now have the situation that for every edge $e \in I(s)$ we have $e \setminus \{a, g, f\} \subseteq e_1 \setminus \{a, g, f\}$. However, as illustrated in Fig. 4, $I^*(\{a, g, f\})$ is not $\subseteq$-ordered. A nest-set must therefore contain further vertices. In this case it is easy to see that either removing b or d is enough. In conclusion we have shown that if $a \in s$, then there are two 4-nest-sets $\{a, b, e, f\}$ and $\{a, d, e, f\}$ that have guard e_1.

What makes the problem difficult is that there can be many possible ways of making edges linearly ordered by vertex deletion. In Example 5.1 both choices, removing either b or d, lead to a 4-nest-set. However, suppose there were an additional edge $e_4 = \{b, x\}$. Then, choosing b would also imply $e_4 \in I(s)$ and $x \in s$. Choosing d would lead to a smaller nest-set.

Algorithm 1: Find nest-set with fixed guard.

input: Hypergraph H, edge e_g, and an integer $k \geq 1$.
output: "Accept", if there exists a nest-set s with guard e and $|s| \leq k$
"Reject", otherwise.

Function `NestExpand` *(s: set of vertices)*
  if $|s| > k$ **then**
    return Reject
  $\Delta \leftarrow \bigcup I(s, H) \setminus (s \cup e_g)$;
  if $\Delta \neq \emptyset$ **then**
    return `NestExpand` $(s \cup \Delta)$;
  $H_g \leftarrow$ The hypergraph $I^*(s, H)$;
  if H_g *has no* $\subseteq$*-conflicts* **then**
    return Accept;
  A $\leftarrow$ all minimal vertex covers of the $\subseteq$-conflict graph of H_g with size at most $k - |s|$;
  foreach $\alpha \in$ **A do**
    if `NestExpand` $(s \cup \alpha)$ *accepts* **then**
      return Accept;
  return Reject;
begin `/* Main */`
  foreach $v \in e_g$ **do**
    if `NestExpand` $(\{v\})$ **then**
      return Accept;
  return Reject;

In general, this type of complication can occur repeatedly and it is therefore necessary to continue this expansion procedure for all possible (minimal) ways of ordering the known incident edges of s. We will therefore first establish an upper bound on these possible expansions.

Intuitively, when we have edges $\{a, b\}$ and $\{b, c\}$, the only way they become comparable by $\subseteq$ is if either a or c is removed. The existence of a linear order over all the edges thus requires resolving all such conflicts. By encoding these conflicts in a kind of conflict graph we can see that the problem is equivalent to finding a vertex cover in the conflict graph.

Definition 5 *($\subseteq$-conflict graph).* Let H be a hypergraph, we define the *$\subseteq$-conflict graph* of H as the graph G obtained by the following construction (with $V(G) = \bigcup E(G)$): For every two distinct edges $e_1, e_2 \in E(H)$, if $v \in e_1 \setminus e_2$ and $u \in e_2 \setminus e_1$, then add an edge $\{v, u\}$ to G. We say that u and v have a *$\subseteq$-conflict* in H.

Lemma 12. *Let H be a hypergraph and let $s \subseteq V(H)$. Then $E(H - s)$ is $\subseteq$-ordered if and only if s is a vertex cover of the $\subseteq$-conflict graph of H.*

Proof. Let G be the $\subseteq$-conflict graph of H. We first show the implication from right to left. Let s be a vertex cover for G and suppose that $E(H - s)$ is not $\subseteq$-ordered. Hence, there are two edges $e_1, e_2 \in E(H - s)$ that are incomparable, i.e., there exist vertices $v \in e_1 \setminus e_2$ and $u \in e_2 \setminus e_1$ and neither v nor u is in the vertex cover s. A conflict can not be introduced by removing the vertices of s and therefore it was already present in H. Therefore, there must be an edge $\{v, u\}$ in G that is not covered by s, contradicting that s is a vertex cover.

For the other direction let $s \subseteq V(H)$ such that $E(H - s)$ is a $\subseteq$-ordered. Then for every $\subseteq$-conflict, i.e., every pair of vertices u, v where there are $e_1, e_2 \in E(H)$ with $v \in e_1 \setminus e_2$ and $u \in e_2 \setminus e_1$, at least one of u, v must be in s. All edges of G are exactly between such pairs of vertices, hence s contains at least one vertex of each edge in G. Therefore s is also a vertex cover of G. □

This correspondence allows us to make use of the following classical result by Fernau [14] on the enumeration of all *minimal vertex covers*. Recall, a vertex cover is called minimal if none of its subsets is a vertex cover.

Proposition 13 *([14]). Let G be a graph with n vertices. There exist at most 2^k minimal vertex covers with size $\leq k$ and they can be fully enumerated in $O(2^k k^2 + kn)$ time.*

In combination with Lemma 12 we therefore also have an upper bound on computing all minimal vertex deletions that resolve all $\subseteq$-conflicts. With this we are now ready to state Algorithm 1 which implements the intuition described at the beginning of this section. The algorithm is given a hypergraph and an edge e_g to use as guard and tries to find a k-nest-set with guard e_g by exhaustively following the steps described in Example 5.1. We are able to show that this indeed leads a correct procedure for finding k-nest-sets with a specific guard.

Lemma 14. *Algorithm 1 is sound and complete.*

Proof. The algorithm has one base-case for acceptance, when $\Delta = \emptyset$ and H_g has no $\subseteq$-conflicts. By Lemma 12 this is equivalent to $I^*(s, H)$ being $\subseteq$-ordered and thus s is a nest-set. As $\Delta = \emptyset$, for every (sub)edge $e \in I^*(s, H)$ we have $e \subseteq e_g$, i.e., e_g is a guard of s. From the check at the beginning of NestExpand we have $|s| \leq k$. Hence, if the algorithm accepts then the set s is a k-nest-set with guard e_g.

To establish completeness we show that if a k-nest-set s with guard e_g exists, then the algorithm will accept. In particular we claim that for every call NestExpand(s'): if there exists a k-nest-set s with guard e_g and $s' \subseteq s$, then either s' is a k-nest set or $s' \cup \alpha \subseteq s$ where $s' \cup \alpha$ is the parameter of one of the calls made directly by NestExpand(s').

We distinguish two cases. First suppose there are edges $e \in I(s')$ such that $e \setminus s' \not\subseteq e_g \setminus s'$. Since e_g is the guard of s, and $I(s') \subseteq I(s)$, every element of $e \setminus e_g$ must necessarily also be in s. This corresponds directly to the set Δ in the algorithm. Hence, $s' \cup \Delta \subseteq s$ when $\Delta \neq \emptyset$, which is clearly the only parameter of a child call.

In the other case, there are no such edges. The claim then states that either s' is a k-nest-set and the algorithm accepts, or that $s' \cup \alpha \subseteq s$ for some $\alpha \in \mathbf{A}$. In the first case, H_g will have no $\subseteq$-conflicts and the algorithm accepts in line 9. Otherwise, if some k-nest set s extends s', then all $\subseteq$-conflicts of s' must be resolved by some deleting vertices $\alpha_s \subseteq s$, that is disjoint with s', from H_g since $I(s') \supseteq I(s)$. Since $|s| \leq k$ and $\alpha_s \cap s' = \emptyset$, we have that $|\alpha_s| \leq k - |s'|$ and α_s is a vertex cover of H_g and therefore $\alpha_s \in \mathbf{A}$ by Lemma 12.

With the claim established, completeness then follows from the fact if e_g is a guard of nest-set s, then s must contain at least one vertex v of e_g. Inductive application of the claim then proves that a k-nest-set will be found by the algorithm (and accepted) when starting from NestExpand($\{v\}$). □

For the sake of simplicity, Algorithm 1 is stated as a decision procedure. Even so, it is easy to see that a k-nest-set with the appropriate guard has been constructed at any accepting state. It is then straightforward to use Algorithm 1 to decide in fixed-parameter polynomial time if a hypergraph has any k-nest-set, and if so output one. In the following we use $\|H\| = |V(H)| + |E(H)|$ for the *size* of hypergraph H.

Theorem 15. *There exists a $2^{O(k^2)} poly(\|H\|)$ time algorithm that takes as input hypergraph H and integer $k \geq 1$ and returns a k-nest-set s of H if one exists, or rejects otherwise.*

Proof. We simply call Algorithm 1 once for each edge of H as the guard. Since every nest-set has a guard Lemma 14 implies that this will find an appropriate nest-set if one exists. If all calls reject, then there can be no nest-set with at most k elements as it is not guarded by any edge of H.

What is left to show is that Algorithm 1 terminates in $2^{O(k^2)} poly(k\|H\|)$ time. Calling the procedure $|E(H)|$ times clearly preserves this bound. First, observe that every recursive call of NestExpand increases the cardinality of s by at least one. The call tree of the recursion therefore has maximum depth k. Furthermore, by Proposition 13 every node in the call tree has at most 2^k children if $\Delta = \emptyset$, or exactly one when $\Delta \neq \emptyset$. Hence, at most $2^{(k^2)}|e_g|$ calls to NestExpand are made in one execution of Algorithm 1.

In each call, the computation of Δ and H_g as well as all the checks are feasible in $O(poly(\|H\|))$ time. Since $\|H_g\| \leq \|H\|$, the set $\mathbf{A}$ can be computed in $O(2^k k^2 + poly(k\|H\|))$ time according to Proposition 13. Recall that we assume $k \leq V(H)$ since the problem is trivial otherwise. The overall execution time of Algorithm 1 is therefore in $2^{O(k^2)} poly(\|H\|)$. □

Once we can find individual k-nest-sets, finding k-NEOs becomes simple. Recall from Lemma 3 that vertex removal preserves k-NEOs. Thus straightforward greedy removal of k-nest-sets is a sound and complete algorithm for finding k-NEOs. Since at most $|V(H)|$ nest-set removals are required to reach the empty hypergraph, using the procedure from Theorem 15 to find the k-nest-sets yields a $2^{O(k^2)} poly(\|H\|)$ time algorithm for Nest-Set-Width.

Corollary 16. Nest-Set-Width *parameterized by k is fixed-parameter tractable.*

6. Nest-set width & conjunctive queries with negation

We move on to prove our main algorithmic result. Recall that a query q has an associated hypergraph $H(q)$. We define the nest-set width of the query q as $nsw(q) = nsw(H(q))$. We say that a class $\mathcal{Q}$ of BoolCQ$^\neg$ instances has *bounded nsw* if there exists a constant c, such that every query q in $\mathcal{Q}$ has $nsw(q) \leq c$.

Theorem 17. *Let $\mathcal{Q}$ be a class of BoolCQ$^\neg$ instances with bounded nsw. Then BoolCQ$^\neg$[$\mathcal{Q}$] is decidable in polynomial time.*

As usual, the result can be extended to unions of conjunctive queries with negation (UCQ$^\neg$) when the *nsw* of a UCQ$^\neg$ is defined to be the maximum *nsw* of its CQ$^\neg$ parts.

While the complexity of CQs without negation has been extensively studied and is well understood, few results extend to the case where negation is permitted. When there are only positive literals, then the satisfying assignments for each literal are explicitly present in the database. Finding a solution for the whole query thus becomes a question of finding a consistent combination of these explicitly listed partial assignments. However, with negative literals it is possible to implicitly express a large number of satisfying assignments. Recovering an explicit list of satisfying assignments for a negative literal may require exponential (in the arity) time and space because there can be up to $|Dom|^a$ such assignments, where a is the arity of the literal.

This additional expressiveness of negative literals has important implications for the study of structural parameters. While evaluation of CQs is NP-complete with and without negation, permitting negation allows for expressing problems as queries with a *simpler* hypergraph structure. Such a change in expressiveness relative to structural complexity must also be reflected in structural parameters that capture tractable classes of the problem. This change in expressiveness is well illustrated by the following proposition and the following theorem. We consider the satisfiability problem (SAT) for propositional formulas in conjunctive normal form (CNF). Recall, that for a formula F in CNF, the corresponding hypergraph $H(F)$ has as its vertices the variables of the formula and every edge is the set of variables of some clause in the formula. A clause $C = l_1 \vee \cdots \vee l_q$ has $2^q - 1$ satisfying assignments to the variables of the clause.

Proposition 18. *There exists a many-one reduction r from* SAT *to* `BoolCQ`$^\neg$ *such that for every formula F the hypergraph $H(F)$ is the same as the hypergraph of the query in $r(F)$.*

Proof sketch. Let F be a propositional formula in CNF. The desired reduction is obtained by creating a relation R_i for every clause, $C_i = \ell_{i,1} \vee \cdots \ell_{i,n_i}$ in F, with a position for every distinct variable in C_i, that contains as a single tuple corresponding to the only unsatisfying assignment for variables in C_i. The domain of the `BoolCQ`$^\neg$ instance is $\{0, 1\}$ and the query consists of the conjunction over all $\neg R_i(\vec{vars}(C_i))$. □

In essence, there exists a reduction from SAT to `BoolCQ`$^\neg$ that preserves the natural hypergraph structure. Since this reduction preserves the hypergraph structure of the SAT formula it follows that structural restrictions can only capture a tractable fragment of `BoolCQ`$^\neg$ if they also capture a tractable fragment of SAT. For example, SAT is NP-hard when restricted to α-acyclic formulas [26], and thus so is CQ$^\neg$ evaluation. In contrast, evaluation of CQs without negation is tractable for α-acyclic queries [30]. Notably, this also proves that no such *structure-preserving* reduction from SAT exists to CQ answering without negation.

Theorem 19 *(Implicit in [26]).* `BoolCQ`$^\neg$ *is NP-hard even when restricted to α-acyclic queries.*

Simplifying assumptions To simplify the presentation of this section we make the following assumptions on the instances of `BoolCQ`$^\neg$. First we assume that queries in Nest-Set-Width instances are always *safe*, i.e., no variable occurs only in negative literals. An unsafe query can always be made safe: If a variable v occurs only in negative literals, we simply add a new literal $R(v)$ with $R^D = \{(d) \mid d \in Dom\}$ to the query. The resulting query is clearly equivalent to the unsafe one on the given domain. Importantly, the additional unary literals do not change the nest-set width of the query. At some points in the algorithm we operate on (sub)queries that are not safe. The assumption of safety is made only for the starting point of the procedure.

Our second assumption is that the size of the domain is exactly a power of 2, i.e., $|Dom| = 2^d$ for some integer d. Since we already assume safe queries, increasing the size of the domain has no effect on the solutions since the newly introduced constants cannot be part of any solution. Furthermore, this assumption increases the size of the domain at most by a constant factor less than 2.

6.1. Relation to previous work

The algorithm presented here builds on the work of Brault-Baron [4] for the β-acyclic case. While we can reuse some of the main ideas, the overall approach used there does not generalize to our setting. There the tractability is first shown for boolean domains, i.e., the domain is restricted to only two values. `BoolCQ`$^\neg$ over arbitrary domains is reduced to the problem over the boolean domain by blowing up each variable in such a way as to encode the full domain using boolean variables. This naturally requires every variable in the original query to be replaced by $\log_2 |Dom|$ many new variables. While this operation preserves β-acyclicity, it can increase *nsw* by a factor $\log_2 |Dom|$.

Example 6.1. Consider the following query and a domain with 8 elements.

$$q = \neg R(a, b) \wedge \neg S(b, c) \wedge \neg T(a, c)$$

The reduction to a query over the boolean domain will then replace every variable v by three variables v_1, v_2, v_3, resulting in the equivalent query q_b over the boolean domain

$$q_b = \neg R(a_1, a_2, a_3, b_1, b_2, b_3) \wedge \neg S(b_1, b_2, b_3, c_1, c_2, c_3) \wedge \neg T(a_1, a_2, a_3, c_1, c_2, c_3)$$

It is easy to see that q has $nsw(q) = 2$ because any combination of two variables is a nest-set of q. However, while $\{a, b\}$ is a nest-set of q, this does not translate to the existence of a 2-nest-set in q_b. It is easy to verify that any $\{a_i, b_j\}$ for $i, j \in [3]$ is not a nest-set. Indeed, applying the ideas from Section 5.2 it is easy to see that in general, for such a triangle query, $nsw(q_b) = 2 \log_2 |Dom|$.

A subtle but key observation must be made here. While the previous example shows that the variable blowup from the binary encoding affects the nest-set width in general, this does not happen when $nsw(q) = 1$. Consider a nest-set $\{v\}$ of some hypergraph H. The edges incident to v are linearly ordered by $\subseteq$. If we add a new vertex v' in all the edges that contain v, then clearly the edges incident to v' are the same as those of v and therefore also linearly ordered by $\subseteq$.

Lemma 20. *Let H be a hypergraph with a nest-set $\{v\}$. Let H' be a hypergraph obtained by adding a new variable v' to H that occurs exactly in the same edges as v. Then $\{v\}$ and $\{v'\}$ are both nest-sets of H'.*

This subtle difference between 1-nest-sets and larger nest-sets will be principal to the following section.

6.2. Eliminating variables

The algorithm for BoolCQ$^\neg$ in the following Section 6.3 will be based around successive elimination of variables from the query. This elimination will be guided by a nest-set elimination ordering where we eliminate all variables of a nest-set at once. This elimination of a nest-set s is performed in three steps.

1. Eliminate all occurrences of variables from s in positive literals.
2. Extend the negative literals incident to s in such a way that they form a β-acyclic subquery.
3. Eliminate the variables of s from the β-acyclic subquery.

In this section we introduce the mechanisms used for these steps. For steps 1 and 2 we need to extend literals in such a way that their variables include all variables from some set s. We do this in a straightforward way by simply extending the relation by all possible tuples for the new variables. It is then easy to see that such extensions are equivalent with respect to their set of satisfying solutions.

Definition 6. Consider a literal $L(v_1, \dots, v_n)$ where L is either R or $\neg R$ and the respective relation R^D. Let s be a set of variables and let $s' = s \setminus \{v_1, \dots, v_n\}$ be the variables in s that are not used in the literal. We call the literal $L(v_1 \dots, v_n, \vec{s'})$ with the new relation $R^D \times Dom^{|s'|}$ the *s-extension of R* (where Dom^m represents the m-ary Cartesian power of the set Dom and we use the relational algebra semantics of the product $\times$).

Lemma 21. *Let $L'(\vec{v}, \vec{s'})$ be the s-extension of $L(\vec{v})$. Then an assignment $a\colon vars(L) \to Dom$ satisfies $L(\vec{v})$ if and only if every extension of a to $vars(L')$ satisfies $L'(\vec{v}, \vec{s'})$.*

Proof. Let $L(\vec{v})$ where L is either R or $\neg R$ and let $L'(\vec{v}, \vec{s'})$ be the s-extension.

Let $a\colon vars(L) \to Dom$ be an assignment that satisfies L. If L is positive, we have $a[\vec{v}] \in R^D$ and then every extension of the tuple to $vars(L')$ exists by the semantics of the relational product. If L is negative, then $a[\vec{v}] \notin R^D$. The relational product for creating the relation of the s-extension will therefore also not create any tuples where $a[\vec{v}]$ occurs in the projection to $\vec{v}$.

On the other hand, let $a\colon vars(L) \to Dom$ such that every $a'\colon vars(L') \to Dom$ that extends a satisfies L'. If L is positive, then any such a' also satisfies $\pi_{vars(L)}(L')$ and therefore a satisfies L as the relational product does not change the tuples of L. If L is negative, consider the tuple $t = a[\vec{v}]$. Suppose $t \in R^D$, i.e., a does not satisfy L. But then any extension of a would also be in the relation of the s-extension, contradicting our assumption that every extension satisfies L'. □

The process for positive elimination is simple. Straightforward projection is used to create a positive literal without the variables from s. A new negative literal then restricts the extensions of satisfying assignments for the new positive literal to exactly those that satisfy the old positive literal. A slightly simpler form of this method was already used in [4].

Lemma 22. *Let $R(\vec{x}, \vec{s})$ be a positive literal. Define new literals $P(\vec{x})$ with $P^D = \pi_{\vec{x}}(R^D)$ and $\neg C(\vec{x}, \vec{s})$ with $C^D = P_s^D \setminus R^D$ where P_s is the s-extension of P. Then an assignment a satisfies $R(\vec{x}, \vec{s})$ if and only if a satisfies $P(\vec{x}) \wedge \neg C(\vec{x}, \vec{s})$.*

Proof. Let a be a satisfying assignment for $R(\vec{x}, \vec{s})$. Clearly, a also satisfies $P(\vec{x})$ and by Lemma 21 it also satisfies P_s. Furthermore, by construction $a[\vec{x}, \vec{s}]$ is explicitly not in C^D and hence a also satisfies $\neg C(\vec{x}, \vec{s})$. On the other hand, let a be a satisfying assignment for $P(\vec{x}) \wedge \neg C(\vec{x}, \vec{s})$. By construction it is then clear that a satisfies only those extensions of $\vec{x}$ that

correspond to a tuple in R^D. At the same time, $a[\vec{x}]$ is in $\pi_x(R^D)$. Hence, a always corresponds to a tuple in R^D and we see that a satisfies $R(\vec{x}, \vec{s})$. □

For the elimination of variables that occur only negatively we build upon a key idea from [4]. There, a method for variable elimination is given for the case where the domain is specifically $\{0, 1\}$. We repeat parts of the argument here to highlight some important details. Consider a query $q = \{\neg R_1, \dots, \neg R_n\}$. The main observation is that the satisfiability of the negative literals $\neg R_1, \dots, \neg R_n$ with variables $x_1, \dots, x_m$ is equivalent to satisfiability of the formula

$$\bigwedge_{i=1}^{n} \bigwedge_{(a_1,\dots,a_q)\in R_i} (a_1 \neq x_{i_1} \vee \cdots \vee a_q \neq x_{i_q})$$

Since we are in the domain $\{0, 1\}$ we have only two cases for the inequalities. Either $0 \neq x$ or $1 \neq x$, which are equivalent to $x = 1$ and $x = 0$, respectively. We can therefore equivalently rewrite the clauses in the formula above as the propositional formula

$$\sigma(a_1)x_{i_1} \vee \cdots \vee \sigma(a_q)x_{i_q}$$

where $\sigma(1) = \neg$ and $\sigma(0) = \epsilon$, i.e., the empty string.

Recall, if we have two clauses $x \vee \ell_1 \vee \cdots \vee \ell_\alpha$ and $\overline{x} \vee \ell'_1 \vee \ell'_\beta$ the *x-resolvent* of the two clauses is $\ell_1 \vee \cdots \vee \ell_\alpha \vee \ell'_1 \vee \cdots \vee \ell'_\beta$. Removing all clauses containing variable x and adding all x-resolvents as new clauses to a given formula in CNF yields an equi-satisfiable formula without the variable x. This process is generally referred to as Davis-Putnam resolution [11] and its formal definition is also recalled in Section 7. If we then reverse the initial transformation from query to propositional formula, we obtain a new query q' (and corresponding database) that no longer contains the variable x. The new q' has a solution if and only if q has a solution.

It was already shown in [26] that resolution on a nest point will never increase the number of clauses. After conversion back to the CQ$^\neg$ setting this means that every relation will contain at most as many tuples as it did before the variable elimination. Note that this conversion can be done by simply reversing the encoding as a propositional formula. Further details can be found in [4]. In combination with other standard properties of resolution one then arrives at the following statement.

Proposition 23 *(Implicit in Lemma 16 in [4]). Let q be the query consisting of the literals $\{\neg R_1(\vec{x_1}, y), \neg R_2(\vec{x_2}, y), \dots, \neg R_n(\vec{x_n}, y)\}$ on database D with domain $\{0, 1\}$ and let $\{y\}$ be a nest-set of q. There exists a query q' of the form $\neg R_1(\vec{x_1}), \neg R_2(\vec{x_2}), \dots, \neg R_n(\vec{x_n})$ and a database D' with the following properties:*

1. *If $a \in q(D)$, then $a[vars(q')] \in q'(D')$.*
2. *If $a' \in q'(D')$ then there exists $c \in Dom$ such that $a' \cup \{y \mapsto c\} \in q(D)$.*
3. *q' and D' can be computed in $O(\|D\|)$ time.*
4. *For every $i \in [n]$ we have $|R_i^{D'}| \leq |R_i^D|$*

It was discussed in the previous section that we can not, in general, reduce a query to an equivalent query with 2 element domain without increasing the nest-set width by a $\log|Dom|$ factor. However, as mentioned in the outline above, our plan is to temporarily transform certain subqueries in such a way that they become β-acyclic and that for any variable v that we want to eliminate, $\{v\}$ is a nest-set of the transformed subquery.

By the observation from Lemma 20, the reduction to a 2 element domain by binary encoding preserves β-acyclicity and allows us to eliminate the encoding variables of v one-by-one using Proposition 23. Afterwards, we can revert the binary encoding by mapping everything back into the original domain. This strategy allows us to lift Proposition 23 to a much more general form, allowing for variable elimination in *arbitrarily large* domains. This will ultimately allow us to circumvent the obstacles described in Section 6.1.

Lemma 24. *Let q be the query $\{\neg R_1(\vec{x_1}, y), \neg R_2(\vec{x_2}, y), \dots, \neg R_n(\vec{x_n}, y)\}$ on database D with $|Dom| = 2^k$ and let $\{y\}$ be a nest-set of q. There exists query q' of the form $\neg R_1(\vec{x_1}), \neg R_2(\vec{x_2}), \dots, \neg R_n(\vec{x_n})$ and a database D' with the following properties:*

1. *If $a \in q(D)$, then $a[vars(q')] \in q'(D')$.*
2. *If $a' \in q'(D')$ then there exists $c \in Dom$ such that $a' \cup \{y \mapsto c\} \in q(D)$.*
3. *q' and D' can be computed in $O(\|D\| \log^2 |Dom|)$ time given q and D as input.*
4. *For every $i \in [n]$ we have $|R_i^{D'}| \leq |R_i^D|$.*

Proof. Since we have $|Dom| = 2^k$ there exists a bijection $f : Dom \to \{0, 1\}^k$ that can be efficiently computed. We then consider the binary version q_b of q where every variable x is substituted by variables $x_1, x_2, \dots, x_k$ and the respective

Algorithm 2: Eliminate nest-set s from q, D.

Function `Elim-s-Positive` (q)
  Let $P_1, \dots, P_n$ be the positive literals incident to s in q;
  Let $\neg R_1, \dots, \neg R_m$ be the negative literals incident to s where $vars(R_j) \subseteq \bigcup_{i=1}^{n} vars(P_i)$;
  $q_J \leftarrow \{P_1, \dots, P_n, \neg R_1, \dots, \neg R_m\}$;
  $J \leftarrow$ all solutions of $q_J(D)$;
  $P \leftarrow \pi_{vars(J)\setminus s}(J)$;
  $P_s \leftarrow$ the s-extension of P;
  $C \leftarrow P_s \setminus J$;
  $q_1 \leftarrow (q \setminus q_J) \cup \{P, \neg C\}$;
  return q_1

Function `Elim-s-Negative` (q_1)
  Let $\neg C, \neg N_1, \dots, \neg N_\ell$ be the literals incident to s in q_1;
  foreach $i \in \ell$ **do**
    $N'_i \leftarrow$ the s-extension of N_i;
  $q^\neg \leftarrow \{\neg C, \neg N'_1, \dots, \neg N'_\ell\}$;
  Let $q^* = \{\neg C^*, \neg N^*_1, \dots, \neg N^*_\ell\}$ be the query obtained by successively eliminating every $v \in s$ from $q^\neg$ using Lemma 24;
  $q_{-s} \leftarrow (q_1 \setminus q^\neg) \cup \{\neg N^*_1, \dots, \neg N^*_\ell\}$;
  Update relation P to $P - C^*$ in D^{-s};
  return q_{-s}

begin `/* Main */`
  $q_1 \leftarrow$ `Elim-s-Positive` (q);
  return `Elim-s-Negative` (q_1);

database D_b where every tuple $(a_1, \dots, a_{ar(R)}) \in R^D$ becomes a $(f(a_1), \dots, f(a_{ar(R)}) \in R^{D_b}$. We thus clearly have that $a \in q(D)$ if and only if $f(a) \in q_b(D_b)$.

Observe that for every $i \in [k]$, $\{y_i\}$ is a nest-set of q_b. Using Proposition 23 we can then successively remove the nest-sets $\{y_i\}$ for all $i \in [k]$, with each elimination requiring $O(\|D_b\|)$ time. Recall from Lemma 3 that nest-sets are preserved when vertices are deleted from the hypergraph. Let q'_b and D'_b be the result of eliminating y_i for every $i \in [k]$ in this fashion.

Since exactly the substitution of y was deleted we can then reverse the transformation to binary and create a q' of the form $\neg R_1(\vec{x_1}), \neg R_2(\vec{x_2}), \dots, \neg R_n(\vec{x_n})$ from q_b as well as the corresponding database D' from D'_b. Again, clearly $a \in q'(D')$ if and only if $f(a) \in q'_b(D_b)$.

Now, if $a \in q(D)$, then $f(a) \in q_b(D_b)$. By Proposition 23, it then also holds that $f(a)[vars(q'_b)] \in q'_b(D'_b)$ and in turn also $f^{-1}(f(a)[vars(q'_b)]) = a[vars(q')] \in q'(D')$. For the other direction, we proceed similarly, if $a' \in q'(D')$ then also $f(a') \in q'_b(D'_b)$. Again, by Proposition 23 this can be extended to an assignment $a_b \in q_b(D_b)$ and thus also implicitly to a $f^{-1}(a_b) \in q(D)$. Since $f(a')$ is extended by assignments for y_1 through y_k it follows that $f^{-1}(a_b)$ extends a' by some assignment for y.

Finally, the transformations to binary form and back are simple rewritings and can be performed in linear time. The elimination of the y_i variables requires $O(k\|D\|)$ time and we have $k = \log_2(|Dom|)$. □

6.3. The elimination procedure

We are now ready to define our algorithm for eliminating nest-sets from CQs with negation. Our procedure for eliminating the variables of a nest-set s from a `BoolCQ`$^\neg$ instance q, D is described in Algorithm 2. Updates to the database are implicit in the algorithm. This is to be understood as adding the corresponding relation for every literal that is added to the query and removing the relations for the deleted literals. We refer to the new instance q', D' returned by the algorithm as the *s-elimination* of q, D.

The procedure begins by eliminating all positive occurrences of s via the function `Elim-s-Positive`. To do so, it considers the subquery q_J, which contains all the positive literals incident to s as well as those negative literals that are fully covered (w.r.t. their variable scopes) by these positive literals. It is straightforward to compute all the solutions of q_J by first joining all the positive literals and then incorporating the negative literals via anti-joins. This can be done efficiently since the variables of q_J can be covered by at most k positive literals by a similar argument as in the proof of Lemma 8. The set of solutions of q_J is taken as a new relation J, to which we apply the mechanism from Lemma 22. The resulting literals P and $\neg C$ replace the subquery q_J in q to form q_1.

The resulting query q_1 thus is equivalent to q and has no variable in s occurring in a positive literal. The only literals incident to s that are left are those negative literals that contain variables beyond those in q_J, and the new literal $\neg C$. The second subprocedure, `Elim-s-Negative`, eliminates the variables in s from these negative literals using Lemma 24. To eliminate a variable $v \in s$ with Lemma 24 we need a β-acyclic subquery where v is a nest-point. We therefore do not consider the literals $\neg N_i$ directly but instead operate on their s-extensions. Observe that the literals $\neg N_i$ are all incident to s and therefore their variables are linearly ordered under $\subseteq_s$. Furthermore, for every $i \in [\ell]$ we have $vars(N_i) \setminus s \supseteq vars(C) \setminus s$. Thus, for the s-extensions N'_i of N_i we have

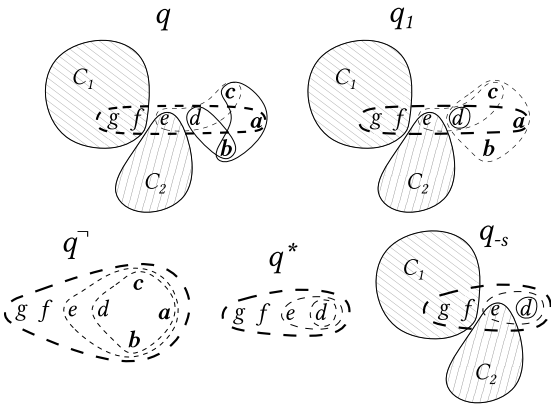

Fig. 5. Example of an s-elimination on hypergraph level. (Dashed edges correspond to negative literals.)

$$vars(C) \subseteq vars(N'_{i_1}) \subseteq vars(N'_{i_2}) \subseteq \cdots \subseteq vars(N'_{i_\ell})$$

for some permutation $N'_{i_1}, \dots N'_{i_\ell}$ of $N'_1, \dots, N'_\ell$. Therefore, $q^\neg$ on line 15 in the algorithm is clearly β-acyclic and all variables in s are present in every literal. Thus also every variable of s is a nest-point of $q^\neg$ and Lemma 24 can be used to eliminate all of them. After the elimination we get the new set of literals q^* which we replace $q^\neg$ with in q_1. The literal $\neg C^*$ always has the same set of variables as the P that was introduced in `Elim-s-Positive`. We thus can simply account for $\neg C^*$ by subtracting the relation of C^* from the relation of P instead of adding $\neg C^*$ as a literal. This way we avoid the possibility of increasing the number of new literals in the resulting final query q_{-s}.

Example 6.2. Consider a query q with nest-set $s = \{a, b, c\}$. The query has literals $P_1(a, b, c)$, $P_2(b, d)$, $\neg N_1(a, d, e, f, g)$, and $\neg N_2(c, d, e)$ incident to s. This setting is illustrated on hypergraph level in Fig. 5 where the components C_1 and C_2 abstractly represent the rest of the query.

Algorithm 2 first computes the intermediate relation J containing all the solutions for $P_1(a, b, c) \wedge P_2(b, d)$. We remove the variables in s from J by projection to obtain the new literal $P(d)$. We furthermore add $\neg C(a, b, c, d)$ as in Lemma 22 to make q_1 equivalent to q. Note that variables from s now occur only in negative literals of q_1.

The procedure then moves on to eliminating s from the negative literals. First, all the negative literals are expanded to cover all variables of s. The expanded negative literals $\neg N'_1(a, d, e, f, g, b, c)$, $\neg N'_2(c, d, e, a, b)$, $\neg C(a, b, c, d)$ make up the subquery $q^\neg$. As discussed above, this expansion modifies the hypergraph structure in such a way that all the variables of s now correspond to 1-nest-sets of $q^\neg$ (see also Fig. 5). They can therefore be eliminated using Lemma 24 to obtain q^*.

Finally, replacing $q^\neg$ in q_1 by q^* (and simplifying $\neg C^*(d) \wedge P(d)$) will produce the final query without variables from s, the s-elimination of q.

What is left is to prove that an s-elimination has a solution if and only if the original query has a solution (Lemma 25) by careful combination of Lemmas 21, 22, and 24 from Section 6.2. Moreover, the s-elimination is always smaller than the original query (Lemma 27) and it can be computed in polynomial time when the size of s is bounded (Lemma 26). From these three properties it will then be straightforward to show our main result in Theorem 17, the tractability of `BoolCQ`$^\neg$ under bounded *nsw*.

Lemma 25. *Let q', D' be the s-elimination of some `BoolCQ`$^\neg$ instance q, D. Then $q'(D') \neq \emptyset$ if and only if $q(D) \neq \emptyset$.*

Proof. From Lemma 22 it follows that $P \wedge \neg C$ is equivalent to q_J. Therefore we have $q_1(D_1) = q(D)$.

Suppose $a \in q_1(D_1)$. Then, for any $i \in [\ell]$ we have that a satisfies $\neg N_i$. From Lemma 21 it follows that a also satisfies $\neg N'_i$. Hence, $a[vars(q^\neg)]$ satisfies $q^\neg$. By Lemma 24 it then follows that $a[vars(q^*)] \in q^*(D^*)$. Since the literals in $q^\neg$ were the only literals incident to s we have that $a[vars(q) \setminus s]$ satisfies all literals in q_{-s}. The update of $P^{D_{-s}}$ in line 18 of the algorithm is trivial since $vars(P) = vars(C^*)$.

For the other side of the equivalence now assume that $a' \in q'(D')$. Then a in particular satisfies all literals of q^*. By Lemma 24 there exists an assignment a_s to s such that $a = (a' \cup a_s)$ satisfies $q^\neg$. Now, observe that for every $j \in [\ell]$, if $a[vars(N'_j)] \notin N'_j$ then also $a[vars(N_j)] \notin N_j$ by Lemma 21. All other literals remain the same between q' and q_1 and thus $a \in q_1(D) = q(D)$. □

Lemma 26. *Let q, D be an instance of `BoolCQ`$^\neg$ and let s be a k-nest-set of q. Then the s-elimination of q, D can be computed in $O(|R_{max}|^k |Dom|^k poly(\|q\| + \|D\|))$ time.*

Proof. We first argue that $vars(q_J)$ can be covered by at most k positive literals. Since s is a nest-set, one of the P_i is maximal (among the positive literals) with regards to $\subseteq_s$. Just like in Lemma 8 we are left with at most $k-1$ variables of s that are not covered by this maximal positive literal. As we assume safety, every such variable requires at most one positive literal to cover it and the claim holds. Thus, J will contain a subset of the tuples formed by a join of k positive literals, hence $|J| \leq O(|R_{max}|^k)$.

A join $A \bowtie B$ is feasible in $O(|A||B| \max\{ar(A), ar(B)\} \log |Dom|)$ time by a straightforward nested loop join. As $ar(R) \leq \|Q\|$ for any relation symbol R we simplify to $O(|R_{max}|^2 \|q\| \log |Dom|)$. As usual, once k positive literals that cover $vars(q_J)$ are joined, any further join is simply a semi-join and requires only linear time. It follows that the joins over all the positive literals of q_J can be computed in $O(|R_{max}|^k \|q\| \log |Dom|)$ time. The negative literals in q_J can then all be removed by anti-joins, which also require linear time, i.e., $A \triangleright B$ can be computed in $O(\|A\| + \|B\|)$ time. At most $|q|$ anti-joins need to be performed and we therefore see that we have an upper bound $O(|R_{max}|^k \|q\|^2 \log |Dom|)$ for the time required to compute J.

Computing P is clearly linear in $\|J\|$ while computing C is feasible in $O(\|J\| \cdot \|Dom^k\|)$ time. We use the fact that $ar(J) \leq \|q\|$ to simplify the final bound.

For every $i \in [\ell]$, the s-extension from N_i to N_i' requires $O(\|R_{max}\| \cdot \|Dom^k\|)$ time. Finally, for each variable $v \in s$, eliminating v requires $O(\|D^\neg\| \log^2 |Dom|)$ time by Lemma 24 and is performed k times. As the s-extension can increase the relation size by a factor of at most $|Dom|^k$ we have $\|D^\neg\| \leq \|D\| \, |Dom|^k$. Note that the linear factor k is simplified away in the final bound by observing $k \leq vars(q)$. □

Lemma 27. *Let q', D' be the s-elimination of some $\texttt{BoolCQ}^\neg$ instance q, D. Then $\|q'\| \leq \|q\|$ and $\|D'\| \leq \|D\|$.*

Proof. For the query, we remove n positive literals. Observe that $n \geq 1$ because we assume q is safe and $s \neq \emptyset$. We only add one new positive literal P. Hence, the number of positive literals can not increase through s-elimination. Every new negative literal in q' corresponds one-to-one to a negative literal that was removed from q. We thus have less or equal literals and strictly less variables in q' than in q.

For the database observe that the variables of the literals $P_1, \ldots, P_n$ can be ordered as follows

$$vars(P_{i_1}) \setminus s \subseteq vars(P_{i_2}) \setminus s \subseteq \cdots \subseteq vars(P_{i_n}) \setminus s$$

By construction we have that $vars(P) = \left(\bigcup_{i=1}^n vars(P_i)\right) \setminus s$. As the union over a chain of subsets is simply the maximal element of the chain we have $vars(P) = vars(P_{i_n}) \setminus s$. It is then easy to see from the construction that $|P^{D'}| \leq |P^D_{i_n}|$. For the new negative literals we have $|N_i^*| \leq |N_i|$ by Lemma 24. Since no arities can increase in the s-elimination we arrive at $\|D'\| \leq \|D\|$. □

Finally, note that the construction of the s-elimination preserves the simplifying assumptions made in the beginning of the section. The domain is never modified by the procedure and if q was safe, then so is its s-elimination q'. Moreover, we also have $H(q') = H(q) - s$ and can therefore repeatedly apply this elimination along a k-NEO to decide whether $q(D) \neq \emptyset$.

Finally, we are ready to show the main result of this section.

Proof of Theorem 17. Let $\mathcal{Q}$ be a class of $\texttt{BoolCQ}^\neg$ instances and say there exists a constant k such that the nest-set width of every query in $\mathcal{Q}$ is at most k.

Let q, D be an instance of $\texttt{BoolCQ}^\neg$ with $nsw(q) \leq k$. First, we compute a k-NEO $\mathcal{O} = (s_1, \ldots, s_\ell)$ which is feasible in polynomial time for constant k by Corollary 16. Then perform the following procedure that starts with $q_0 := q$, $D_0 := D$, $i := 1$:

1. If $i > \ell$, accept the input. Otherwise continue with the next step.
2. Let q_i, D_i be the s_i-elimination of q_{i-1}, D_{i-1}. Rename the new P literal to P_i.
3. If $P_i^{D_i} = \emptyset$, reject the input. Otherwise increment i by 1 and continue from step 1.

In case of acceptance, the procedure has eliminated all variables and only the 0-ary literal P_ℓ is left in q_ℓ. Since the procedure did not reject in the step before, we have $P_\ell^{D_\ell} \neq \emptyset$, i.e., it contains the empty tuple and thus $q_\ell(D_\ell) = \{()\} \neq \emptyset$. On the other hand, if the procedure rejects at step i, then $P_i^{D_i} = \emptyset$. The literal P_i occurs positively in q_i and it follows that $q_i(D_i) = \emptyset$.

By Lemma 25 we have $q_i(D_i) \neq \emptyset$ if and only if $q(D) \neq \emptyset$ for all $i \in [\ell]$. The described procedure is therefore sound and complete. The computation of q_i, D_i from q_{i-1}, D_{i-1} is performed at most $\ell \leq vars(q)$ times. By Lemmas 27 and 26 the procedure requires only polynomial time in $\|q\|$ and $\|D\|$. □

7. Nest-set width & and SAT

In the previous section we have already seen some connections between SAT and nest-set width. Previous work by Brault-Baron [4] already showed that nest-points have desirable properties for resolution procedures of propositional formulas and these observations are critical for Lemma 24 and the overall elimination of negative atoms in the previous section. Ordyniak et al. [26] made this connection more concrete and showed that SAT is tractable for formulas with β-acyclic hypergraphs. Moreover, Proposition 18 in combination with Theorem 17 already shows that SAT is tractable when restricted to classes of formulas with constantly bounded *nsw* when we extend *nsw* naturally to propositional formulas (in CNF) via their hypergraph structure.

In this section we expand further on the connection of nest-set width and SAT. In particular, we show that SAT is not only tractable for classes with bounded *nsw*, but even fixed-parameter tractable when parameterized by *nsw*. We first show that this upper bound can be observed directly via reduction to $\texttt{BoolCQ}^\neg$ and by the results of the previous section. Afterwards, we investigate the behavior of classic Davis-Putnam (DP) resolution [11] under nest-set width. Somewhat surprisingly, standard DP resolution also yields a fixed-parameter polynomial algorithm for SAT as long as the order of variables in the resolution follows a (low-width) NEO.

7.1. SAT as $\texttt{BoolCQ}^\neg$

A closer look at the upper bound from Lemma 26 shows that the nest-set width appears only in the exponent of $|R_{max}|$ and $|Dom|$. A reduction to $\texttt{BoolCQ}^\neg$ where these two cardinalities can be constantly bounded thus shows fixed-parameter tractability of the original problem when parameterized by *nsw*.

Recall the simple reduction from SAT to $\texttt{BoolCQ}^\neg$ given in the proof of Proposition 18. For a formula F in CNF consisting of clauses $C_1, \dots, C_n$. For every clause C_i with variables $v_1, \dots, v_\ell$ we create a literal $\neg R_i(v_1, \dots, v_\ell)$ where the relation R_i^D contains the single tuple corresponding to the only assignment that does not satisfy the clause. To satisfy our assumption of safety we slightly extend the reduction[2] to also create a positive literal $V_j(v_j)$ for every variable v_j with $V_j^D = \{(0), (1)\}$, i.e., the whole domain.

We see that this reduction produces a $\texttt{BoolCQ}^\neg$ instance with $|R_{max}| = |Dom| = 2$ while $\|q\|$ and $\|D\|$ are linear in the size of F. Plugging these values into the bound from Lemma 26 gives us a $2^{O(k)} poly(|F|)$ time bound for an s-elimination in this query, where $k = nsw(H(F))$ is the *nest-set width of the formula* F. Hence, repeated s-elimination along a k-NEO yields a fixed-parameter tractable procedure for SAT.

Theorem 28. SAT *for propositional CNF formulas is fixed-parameter tractable when parameterized by the nest-set width of the formula.*

Building on the discussions from Section 4 it is also interesting to note that SAT is known to be W[1]-hard when parameterized by incidence clique width [26].

7.2. Davis-Putnam resolution over nest-sets

We begin this section by recalling the details of DP resolution. For this section, we consider a *clause* C of a propositional formula to be a set of literals x or $\overline{x}$ where x is a variable. If a literal is of the form x we say it is positive and otherwise it is negative. This is also referred to as the *phase* of the literal. If C is a clause, we write $\overline{C}$ for the clause $\{\overline{\ell} \mid \ell \in C\}$, i.e., every literal in the clause is flipped (note that $\overline{\overline{x}} = x$). A formula F (in CNF) is treated as a set of clauses in the following.

For two clauses $C, D \in F$ with $C \cap \overline{D} = \{x\}$ we call $(C \cup D) \setminus \{x, \overline{x}\}$ the *x-resolvent* of C and D. Note that we do not need to care about cases where $C \cap \overline{D} \supset \{x\}$. Say the intersection equals $\{x, y\}$. Resolving on x would yield a new clause containing both y and $\overline{y}$. Such a new clause is therefore trivially satisfied and of no interest.

Let $DP_x(F)$ be the formula obtained by first adding all x-resolvents to F and then removing all clauses where x occurs. Such a step is commonly called a *(Davis-Putnam) resolution step* [11]. It is well-known that F and $DP_x(F)$ are equisatisfiable for all variables x.

We write $DP_{x_1,\dots,x_q}(F)$ for $DP_{x_q}(DP_{x_{q-1}}(\cdots DP_{x_1}(F)\cdots))$, and simply DP_s for a set $s \subseteq vars(F)$ if the particular order does not matter. The procedure $DP_{vars(F)}(F)$ will produce either an empty clause at some step or end in an empty formula. F is unsatisfiable in the first case and satisfiable in the second case. Let C be a clause and s a set of variables. We write $C - s$ for $C \setminus (s \cup \overline{s})$, the clause with all literals of variables from s removed. We extend this notation to formulas as $F - s := \{C - s \mid C \in F\}$ for formula F.

In general, $DP_x(F)$ can contain more clauses than F and the whole procedure can require exponential time (and space) because of this intermediate blowup. However, as we will see, if we eliminate nest-sets then the increase in clauses is only temporary. We start by showing that resolution on a variable in a nest-set will only produce resolvents that remain, in a sense, local to the nest-set.

[2] Strictly speaking this results in the resulting query having a different hypergraph because of the addition of singleton edges. This is however of no consequence for the *nsw* of the hypergraph.

Lemma 29. *Let F be a propositional formula in CNF and let $s = \{v_1, \dots, v_q\}$ be a nest-set of $H(F)$. For every $s' \subseteq s$, and any clauses $C, D \in DP_{s'}(F)$ that contain a variable from s, we have that $vars(C) \subseteq_s vars(D)$, or vice versa.*

Proof. Proof is by induction on the cardinality of s'. If $s' = \emptyset$ no resolution takes place and the statement is trivial.

Say $s' = \{v_1, \dots, v_k\}$ and consider $C, D \in DP_{v_1,\dots,v_{k-1}}(F)$ such that both clauses contain a variable from s. According to the induction hypothesis, we have either $vars(C) \subseteq_s vars(D)$ or $vars(D) \subseteq_s vars(C)$. We continue with the first case and note that the argument for the other case is symmetric. To show that the statement also holds for s' we show that each v_k-resolvent of $DP_{v_1,\dots,v_{k-1}}(F)$ also satisfies this property.

In particular, for such C, D and $C \cap \overline{D} = \{v_k\}$ we show that $vars(R) \setminus s = vars(D) \setminus s$, where R is the v_k-resolvent of C and D. Since $vars(D) \setminus r$ is comparable to all other clauses that contain a variable of s, so is $vars(R) \setminus s$ and the statement also holds for $DP_{s'}(F)$.

It is not hard to see that the two sets are in fact the same: since $R = (C \cup D) \setminus \{v_k, \overline{v_k}\}$ and $v_k \in s$ we also have $var(R) \setminus s = (var(C) \cup var(D)) \setminus s$. Recall that we have $(var(C) \setminus s) \subseteq (var(D) \setminus s)$ and therefore $(var(C) \cup var(D)) \setminus s = var(D) \setminus s$. Note that the argument remains the same if $D \cap \overline{C} = \{v_k\}$. □

Lemma 30. *Let F be a propositional formula in CNF and let $s = \{v_1, \dots, v_q\}$ be a nest-set of $H(F)$. For every $s' \subseteq s$ and any clause $C \in DP_{s'}(F)$ it holds that $C - s \in F - s$.*

Proof. Proof is by induction on the cardinality of s'. For $s' = \emptyset$ the statement is true by definition.

Suppose the statement holds for $|s'| < k$ and let $s' = \{v_1, \dots, v_k\}$. Consider a $C \in DP_{s'}(F) \setminus DP_{s' \setminus \{v_k\}}(F)$, i.e., a new clause obtained by the resolution on v_k after resolution on all the other variables of s' was already performed. Thus, $C = (C_1 \cup C_2) \setminus \{v_k, \overline{v_k}\}$ for some $C_1, C_2 \in DP_{s' \setminus \{v_k\}}(F)$ where $C_1 \cap \overline{C_2} = \{v_k\}$. By Lemma 29 we see that $vars(C_1) \setminus s \subseteq vars(C_2) \setminus s$ (or the symmetric case).

Now, for every variable $v \in (vars(C_1) \cap vars(C_2)) \setminus s$, we know that v occurs in the same phase in both clauses since $C_1 \cap \overline{C_2} = \{v_k\}$. Hence, $C_1 - s \subseteq C_2 - s$ and therefore also $C - s = C_2 - s$. By the induction hypothesis we have $C_2 - s \in F - s$ and the proof is complete. □

While Lemma 30 has $DP_s(F) \leq |F|$ as a direct consequence, it also gives insight into the size of the intermediate formulas. In particular, for any non-empty $s' \subseteq s$ we have $|DP_{s'}(F)| \leq 3^{k-|s'|}|F|$. This can be observed from the fact that any clause $C \in DP_{s'}(F)$ is an extension of a clause from $F - s$ by any combination of literals for the variables $s \setminus s'$. Specifically, there are three possibilities for every such variable, it either occurs positively, negatively, or not at all in C. Thus, any clause in $F - s$ has at most $3^{|s \setminus s'|} = 3^{k-|s'|}$ extensions.

From Section 5.2 we know that we can compute a k-NEO of $H(F)$ in fixed-parameter polynomial time when parameterized by k. From the size bound above it is then easy to see that each resolution step in a nest-set can be performed in fixed-parameter polynomial time, by performing resolution in order of a k-NEO. Hence, repeating the resolution step along a k-NEO is a fixed-parameter tractable procedure for SAT parameterized by k.

Theorem 31. SAT *for propositional CNF formulas is fixed-parameter tractable when parameterized by the nest-set width of the formula.*

8. Conclusion & outlook

In this paper, we have introduced nest-set width in an effort to generalize tractability results for β-acyclicity. We have established the relationship between nest-set width and related width measures. In particular, *nsw* is a specialization of β-hw. In an improvement over β-hw, checking $nsw(H) \leq k$ is shown to be fixed-parameter tractable when parameterized by *nsw*. Finally, we verify that *nsw* is useful for generalizing tractability from β-acyclicity by proving new tractability results for Boolean CQ$^\neg$ evaluation and SAT.

A natural next question if the search for matching lower bounds. Unfortunately, little can be said on the topic at this point in time. As with other width measures for hypergraphs (with unbounded edge size) it is challenging to understand the exact structural implications of high width. Related lower bounds, e.g., for bounded arity CQ evaluation [18], commonly rely on landmark results on the structure of *graphs* with high treewidth. Similar structural understanding of hypergraphs still eludes us and greatly hinders the search for lower bounds in this space.[3]

An interesting question has been left open in this paper: the relationship between nest-set width and point-width. A direct comparison is difficult at this point: since the (fixed-parameter) tractable computation of point decompositions remains an open question, the applicability of point-width for algorithmic results in our setting is not clearly established. Notably, bounded point-width also implies a tractable algorithm by implying bounded maximum induced matching width (mim-width) [8]. However, the algorithm for SAT requires a branch decomposition of low mim-width and deciding mim-width is known to be W[1]-hard [29]. In consequence, to the best of our knowledge, the complexity of CQ$^\neg$ or SAT for classes

[3] Some minor progress has been made in this area recently in [23], albeit limited to hypergraphs of degree 2.

of bounded point-width is not yet known. On the structural side, we conjecture that bounded point-width is a more general notion than bounded nest-set width. In particular, the main ideas from the proof that point-width generalize β-acyclicity in [8] may be adaptable to nest-set width.

Finally, our results make us hopeful that *nsw* can find broader application beyond the problems tackled in this paper. For example, the tractability of #SAT for β-acyclic formulas from [6] is based on nest point elimination and its generalization is thus a natural candidate for further investigation. Another promising avenue of research is the analysis of the *Minesweeper* [25] algorithm for bounded nest-set width queries. Minesweeper is a algorithm for processing join queries (i.e., CQs with no negation) that depends on a global ordering of attributes in its execution. It has been shown that for every β-acyclic query, there exists such a attribute ordering such that Minesweeper runs in linear time. Importantly, the respective argument and algorithm again critically relies on the characterization of β-acyclicity via elimination orderings that nest-set width naturally generalizes. We are thus presented with the natural question of whether nest-set elimination orderings can similarly be leveraged by Minesweeper. In a less concrete sense, the results of Section 7.2 may also motivate further study of nest-sets in the wider area of resolution procedures.

Declaration of competing interest

The authors declare the following financial interests/personal relationships which may be considered as potential competing interests:

Matthias Lanzinger reports financial support was provided by The Royal Society (Reference No. RP\R1\201074). Matthias Lanzinger reports financial support was provided by Austrian Science Fund project P30930.

Acknowledgements

The author is very grateful to Reinhard Pichler for his valuable comments. This work was supported by the Austrian Science Fund (FWF) project P30930, the Royal Society “RAISON DATA” project (Reference No. RP\R1\201074), and the VADA (Value Added Data Systems, EP/M025268/) extension project by the University of Oxford.

References

[1] I. Adler, G. Gottlob, M. Grohe, Hypertree width and related hypergraph invariants, Eur. J. Comb. 28 (8) (2007) 2167–2181.
[2] A. Atserias, M. Grohe, D. Marx, Size bounds and query plans for relational joins, SIAM J. Comput. 42 (4) (2013) 1737–1767.
[3] H.L. Bodlaender, B. de Fluiter, Reduction algorithms for constructing solutions in graphs with small treewidth, in: Proc. COCOON 1996, Springer, 1996, pp. 199–208.
[4] J. Brault-Baron, A negative conjunctive query is easy if and only if it is beta-acyclic, in: Proc. CSL 2012, 2012, pp. 137–151.
[5] J. Brault-Baron, Hypergraph acyclicity revisited, ACM Comput. Surv. 49 (3) (2016) 54.
[6] J. Brault-Baron, F. Capelli, S. Mengel, Understanding model counting for beta-acyclic CNF-formulas, in: Proc. STACS 2015 , Schloss Dagstuhl, 2015, pp. 143–156.
[7] F. Capelli, A. Durand, S. Mengel, Hypergraph acyclicity and propositional model counting, in: Proc. SAT 2014, Springer, 2014, pp. 399–414.
[8] C. Carbonnel, M. Romero, S. Zivný, Point-width and Max-CSPs, ACM Trans. Algorithms 16 (4) (2020) 54.
[9] B. Courcelle, S. Olariu, Upper bounds to the clique width of graphs, Discrete Appl. Math. 101 (1–3) (2000) 77–114.
[10] M. Cygan, F.V. Fomin, L. Kowalik, D. Lokshtanov, D. Marx, M. Pilipczuk, M. Pilipczuk, S. Saurabh, Parameterized Algorithms, Springer, 2015.
[11] M. Davis, H. Putnam, A computing procedure for quantification theory, J. ACM 7 (3) (1960) 201–215.
[12] D. Duris, Some characterizations of γ and β-acyclicity of hypergraphs, Inf. Process. Lett. 112 (16) (2012) 617–620.
[13] R. Fagin, Degrees of acyclicity for hypergraphs and relational database schemes, J. ACM 30 (3) (1983) 514–550.
[14] H. Fernau, On parameterized enumeration, in: Proc. COCOON 2002, Springer, 2002, pp. 564–573.
[15] G. Gottlob, G. Greco, F. Scarcello, Pure Nash equilibria: hard and easy games, J. Artif. Intell. Res. 24 (2005) 357–406.
[16] G. Gottlob, N. Leone, F. Scarcello, Hypertree decompositions and tractable queries, J. Comput. Syst. Sci. 64 (3) (2002) 579–627.
[17] G. Gottlob, R. Pichler, Hypergraphs in model checking: acyclicity and hypertree-width versus clique-width, in: Proc. ICALP 2001, Springer, 2001, pp. 708–719.
[18] M. Grohe, The complexity of homomorphism and constraint satisfaction problems seen from the other side, J. ACM 54 (1) (2007) 1.
[19] M. Grohe, D. Marx, Constraint solving via fractional edge covers, ACM Trans. Algorithms 11 (1) (2014) 4.
[20] M. Gyssens, P. Jeavons, D.A. Cohen, Decomposing constraint satisfaction problems using database techniques, Artif. Intell. 66 (1) (1994) 57–89.
[21] R.M. Karp, Reducibility among combinatorial problems, in: Proc. Complexity of Computer Computations, 1972, pp. 85–103.
[22] M.A. Khamis, H.Q. Ngo, D. Suciu, What do Shannon-type inequalities, submodular width, and disjunctive datalog have to do with one another?, in: Proc. PODS 2017, ACM, 2017, pp. 429–444.
[23] M. Lanzinger, The complexity of conjunctive queries with degree 2, in: Proc. PODS 2022, ACM, 2022, pp. 91–102.
[24] D. Marx, Tractable hypergraph properties for constraint satisfaction and conjunctive queries, J. ACM 60 (6) (2013) 42.
[25] H.Q. Ngo, D.T. Nguyen, C. Ré, A. Rudra, Beyond worst-case analysis for joins with minesweeper, in: Proc. PODS 2014, ACM, 2014, pp. 234–245.
[26] S. Ordyniak, D. Paulusma, S. Szeider, Satisfiability of acyclic and almost acyclic CNF formulas, Theor. Comput. Sci. 481 (2013) 85–99.
[27] C.H. Papadimitriou, Computational Complexity, Academic Internet Publ., 2007.
[28] S.H. Sæther, J.A. Telle, M. Vatshelle, Solving #SAT and MAXSAT by dynamic programming, J. Artif. Intell. Res. 54 (2015) 59–82.
[29] S.H. Sæther, M. Vatshelle, Hardness of computing width parameters based on branch decompositions over the vertex set, Theor. Comput. Sci. 615 (2016) 120–125.
[30] M. Yannakakis, Algorithms for acyclic database schemes, in: Proc. VLDB 1981, VLDB, 1981, pp. 82–94.
[31] M. Yannakakis, Node-deletion problems on bipartite graphs, SIAM J. Comput. 10 (2) (1981) 310–327.