# Efficiently Enumerating Hitting Sets of Hypergraphs Arising in Data Profiling*

Thomas Bläsius[a,1], Tobias Friedrich[b,1], Julius Lischeid[1], Kitty Meeks[c,2], Martin Schirneck[b,1,*]

[a] *Karlsruhe Institute of Technology, Karlsruhe, Germany*
[b] *Hasso Plattner Institute, University of Potsdam, Germany*
[c] *University of Glasgow, Glasgow, United Kingdom*

**Abstract**

The transversal hypergraph problem asks to enumerate the minimal hitting sets of a hypergraph. If the solutions have bounded size, Eiter and Gottlob [SICOMP'95] gave an algorithm running in output-polynomial time, but whose space requirement also scales with the output. We improve this to polynomial delay and space. Central to our approach is the extension problem, deciding for a set $X$ of vertices whether it is contained in any minimal hitting set. We show that this is one of the first natural problems to be $\mathsf{W}[3]$-complete. We give an algorithm for the extension problem running in time $\mathrm{O}(m^{|X|+1}n)$ and prove a SETH-lower bound showing that this is close to optimal. We apply our enumeration method to the discovery problem of minimal unique column combinations from data profiling. Our empirical evaluation suggests that the algorithm outperforms its worst-case guarantees on hypergraphs stemming from real-world databases.

*An extended abstract of this work was presented at the 21st Meeting on Algorithm Engineering and Experiments (ALENEX 2019) [8].

*Corresponding author (martin.schirneck@hpi.de).

*Email addresses:* thomas.blaesius@kit.edu (Thomas Bläsius), tobias.friedrich@hpi.de (Tobias Friedrich), jsl71@cantab.ac.uk (Julius Lischeid), kitty.meeks@glasgow.ac.uk (Kitty Meeks), martin.schirneck@hpi.de (Martin Schirneck)

[1]This work originated while all but the fourth author were affiliated with the Hasso Plattner Institute at the University of Potsdam.

[2]Kitty Meeks is supported by a Personal Research Fellowship from the Royal Society of Edinburgh, funded by the Scottish Government.

*Preprint submitted to Journal of Computer and System Sciences.*

*Keywords:* data profiling, enumeration algorithm, minimal hitting set, transversal hypergraph, unique column combination, W[3]-completeness

## 1. Introduction

A recurring computational task in the profiling of relational databases is the discovery of hidden dependencies between attributes. For instance, *unique column combinations* (UCCs) are subsets of attributes such that the value combinations appearing in them are duplicate-free. An inclusion-wise minimal UCC reveals structural properties of the stored information and serves as a small fingerprint of the data. Unique column combinations, however, are equivalent to *hitting sets* in hypergraphs. While finding a single minimal hitting set is trivial, it is usually not enough to decide the existence of a single UCC. Instead, one aims to compile a comprehensive list of all dependencies. The UCCs are then used for subsequent data cleaning and enable certain query optimizations [1, 47]. One thus has to solve the *transversal hypergraph problem*. This is the task of enumerating all minimal hitting sets of a given hypergraph without repetitions.

Besides data profiling, the transversal hypergraph problem also emerges in many other fields, like artificial intelligence [40], machine learning [26], distributed systems [39], integer linear programming [12], and monotone logic [31]. Despite the large interest, the exact complexity of the enumeration problem is still open. A hypergraph can have exponentially many minimal hitting sets, ruling out any polynomial algorithm. Instead, one could hope for an *output-polynomial* method whose running time scales polynomially in the input size and the total number of solutions. Unfortunately, we do not know how to achieve this. The currently fastest algorithm was presented by Fredman and Khachiyan [37] and runs in time $N^{\mathrm{O}(\log N/\log\log N)}$, where $N$ denotes the combined input and output size. It is the major open question in enumeration, whether the transversal hypergraph problem can be solved in output-polynomial time [25, 32, 50, 56].

In the absence of a tractable algorithm for general inputs, special classes of hypergraphs have received a lot of attention. For example, it is known that the transversal hypergraph problem admits an output-polynomial algorithm when restricted to hypergraphs with bounded edge size [11] or dual-conformality [46], as well as acyclic hypergraphs [30, 31].

In this work, we are interested in the case where the maximum *solution size* is small. Given a hypergraph, let $k^*$ be the maximum cardinality of its

minimal hitting sets. This is known as the *transversal rank*. Indeed, it is very common for hypergraphs arising in data profiling to have low transversal rank, see [48, 53]. Eiter and Gottlob [30] gave an output-polynomial algorithm for hypergraphs for which $k^*$ is a constant. We discuss their approach in detail in Section 3.1. Unfortunately, their algorithm is not usable in data profiling applications as its *space* consumption scales with the output size. Also, one would like to have a guarantee on the *delay*, the worst-case time between two consecutive outputs, that is independent of the number of solutions. Lastly, although the transversal rank can be expected to be small usually no a priori bound on $k^*$ is known before the enumeration. In fact, it is $\mathsf{W}[1]$-hard to compute $k^*$ and $\mathsf{NP}$-hard to approximate, see Section 3.1. We are able to improve in all those aspects and obtain an algorithm that is oblivious to $k^*$, has a space requirement independent of the output size, and, in the case that $k^*$ is constant, has polynomial delay.

Central to our approach is a subroutine that decides for a set $X$ of vertices whether it is contained in any minimal hitting set. We examine the parameterised complexity of this *extension problem*, when parameterised by $|X|$. We identify it as one of the first natural problems to be complete for the class $\mathsf{W}[3]$. Prior to the first announcement of this result, there were only two other problems known with this property. The first one was given by Chen and Zhang [19] in the context of supply chain management and Bläsius, Friedrich, and Schirneck [9] added the detection of inclusion dependencies in relational data. Since then, Casel et al. [17] used the techniques developed in Section 4 to show $\mathsf{W}[3]$-hardness already for the special case of extension to minimal dominating sets in bipartite graphs. Very recently, Hannula, Song, and Link [41], building on [9], have proven that independence detection in databases is complete for $\mathsf{W}[3]$ as well.

We also approach the extension problem with tools from fine-grained complexity. Assuming the Strong Exponential Time Hypothesis (SETH), we prove that our subroutine algorithm is almost optimal. Moreover, we argue that closing the remaining gap between the upper and lower bound is likely to be hard, using a nondetermistic extension of SETH recently conjectured by Carmosino et al. [15]. Next, we give an overview of our results in detail.

### *1.1. Our Contribution*

We solve the transversal hypergraph problem with simultaneously polynomial delay and space on hypergraphs with bounded transversal rank. More

generally, we devise an algorithm that does not need to know $k^*$. Notwithstanding, the analysis of the delay depends on the transversal rank.

**Theorem 1.** *There exists an algorithm that on $n$-vertex, $m$-edge hypergraphs enumerates the minimal hitting sets with delay $\mathrm{O}(m^{k^*+1}n^2)$ in $\mathrm{O}(mn)$ space, where $k^*$ is the maximum cardinality of any minimal hitting set.*

At its core, the algorithm is a tree search in the space of all vertex subsets. The tree is pruned by deciding for a given set $X$ whether it can be extended to a minimal hitting set. We analyse the parameterised complexity of this decision with respect to the parameter $|X|$.

**Theorem 2.** *The extension problem for minimal hitting sets is complete for $\mathsf{W}[3]$ when parameterised by the cardinality $|X|$ of the set to be extended.*

It may seem counterintuitive to solve the enumeration problem by reducing it to a hard decision problem. The key property we use is that extension is tractable, provided that $X$ contains only a few vertices.

**Theorem 3.** *There exists an algorithm that decides for an $n$-vertex, $m$-edge hypergraph and a set $X$ of vertices whether $X$ is contained in any minimal hitting set in time $\mathrm{O}(m^{|X|+1}n)$ and space $\mathrm{O}(mn)$.*

It is natural to ask whether the exponential dependency on $|X|$ in the running time can be improved. We give several conditional lower bounds, all of which indicate that Theorem 3 is close to optimal. They present a trade-off between the strength of the conjecture one is willing to assume and the strength of the resulting bound.

**Theorem 4.** *Let $f$ be an arbitrary computable function. No algorithm can decide for an $n$-vertex, $m$-edge hypergraph and a set $X$ of vertices whether $X$ is contained in any minimal hitting set*

*(i) in time $f(|X|) \cdot \mathsf{poly}(m,n)$, unless $\mathsf{W}[3] = \mathsf{FPT}$;*

*(ii) in time $f(|X|) \cdot (m+n)^{o(|X|)}$, unless $\mathsf{W}[2] = \mathsf{FPT}$;*

*(iii) in time $m^{|X|-\varepsilon} \cdot \mathsf{poly}(n)$ for any constant $|X| \geq 2$ and $\varepsilon > 0$, unless the Strong Exponential Time Hypothesis fails.*

The SETH-lower bound matches our algorithmic result up to a factor of $m$. There is a complexity-theoretic obstacle for closing the remaining gap. We argue that if one could show tight SETH-hardness of the extension problem with a time bound of $m^{|X|+1-\varepsilon} \cdot \mathsf{poly}(n)$ via a *deterministic* reduction, this would refute the Nondeterministic Strong Exponential Time Hypothesis (NSETH) [15] and thereby resolve several open problems in circuit complexity and satisfiability.

Finally, we evaluate an implementation of our algorithm by applying it to the discovery problem of minimal UCCs. Our experiments show that our method is much faster on hypergraphs stemming from real-world databases than the running time bounds would suggest. In practice, a few simple checks can avoid the worst-case behaviour on many instances, which boosts the performance. We also confirm the low memory footprint of our approach.

### *1.2. Outline*

Next, we fix notation and recall basic concepts from combinatorics and complexity theory. In Section 3, we first review what is known about the transversal rank and then present our enumeration algorithm. There, the extension problem for minimal hitting sets is only used as a black box. It is discussed in detail in Section 4. Section 5 combines the results of the previous two sections and proves the bounds on the delay and space. In Section 6, we report on the empirical performance of our method in the context of data profiling. The work is concluded in Section 7.

## 2. Preliminaries

For a set $S$, let $\mathcal{P}(S)$ be the power set of $S$. We use $\mathbb{N}^+ = \{1, 2, \dots\}$ for the positive integers, and, for $k \in \mathbb{N}^+$, we set $[k] = \{1, 2, \dots, k\}$. Computational objects are implicitly assumed to be encoded as bit strings from $\{0,1\}^*$.

*2.1. Hypergraphs and Hitting Sets*

A *hypergraph* is a non-empty, finite *vertex set* $V \neq \emptyset$ together with a system of subsets $\mathcal{H} \subseteq \mathcal{P}(V)$, the *(hyper-)edges*. A hypergraph is identified with its edge set $\mathcal{H}$ if this does not create ambiguities. We do not exclude special cases like the empty hypergraph ($\mathcal{H} = \emptyset$), an empty edge ($\emptyset \in \mathcal{H}$), or isolated vertices ($V \supsetneq \bigcup_{E \in \mathcal{H}} E$). The number of edges is $n = |V|$, the number of edges $m = |\mathcal{H}|$. The *rank* of a hypergraph $\mathcal{H}$ is the maximum cardinality of its edges. A *graph* is a hypergraph whose edges all have size exactly 2.

A *transversal* or *hitting set* of a hypergraph $(V, \mathcal{H})$ is a set $H \subseteq V$ such that $H$ has a non-empty intersection with every edge $E \in \mathcal{H}$. A transversal is *(inclusion-wise) minimal* if it does not properly contain any other transversal. We extensively use the following observation.

**Proposition 5** (Folklore)**.** *Let $\mathcal{H}$ be a hypergraph and $H$ a hitting set of $\mathcal{H}$. Then, $H$ is minimal if and only if every $x \in H$ has a* private edge *$E_x \in \mathcal{H}$ such that $E_x \cap H = \{x\}$.*

The minimal hitting sets of $\mathcal{H}$ form the *transversal hypergraph* $\mathrm{Tr}(\mathcal{H})$ on the same vertex set $V$. We occasionally denote the number of its edges by $N_{\min}$. We abbreviate the rank of the transversal hypergraph $\mathrm{rank}(\mathrm{Tr}(\mathcal{H}))$ as *transversal rank*, and, if $\mathcal{H}$ is clear from the context, we use $k^*$ to denote it.

A hypergraph is *Sperner* if none of its edges is properly contained in another. The *minimisation* of $\mathcal{H}$ is the subhypergraph of inclusion-wise minimal edges, $\min(\mathcal{H}) = \{E \in \mathcal{H} \mid \forall E' \in \mathcal{H} \colon E' \subseteq E \Rightarrow E' = E\}$. Note that $\min(\mathcal{H})$ and $\mathrm{Tr}(\mathcal{H})$ are always Sperner hypergraphs. Regarding transversals, it does not make a difference whether the full hypergraph is considered or its minimisation as we have $\mathrm{Tr}(\mathcal{H}) = \mathrm{Tr}(\min(\mathcal{H}))$. Moreover, the minimisation and transversal hypergraph are mutually dual, meaning $\mathrm{Tr}(\mathrm{Tr}(\mathcal{H})) = \min(\mathcal{H})$. For any two Sperner hypergraphs $\mathcal{G}$ and $\mathcal{H}$, we have $\mathcal{G} = \mathrm{Tr}(\mathcal{H})$ if and only if $\mathcal{H} = \mathrm{Tr}(\mathcal{G})$. The minimisation is computable in quadratic time. In this work, we therefore assume all hypergraphs to be Sperner.

Any total ordering $\preccurlyeq$ of the vertex set $V$ induces a *lexicographical order* on $\mathcal{P}(V)$. Following the definition in [45], we say a subset $S \subseteq V$ is *lexicographically smaller (or equal)* than subset $T$, denoted $S \preccurlyeq_{\mathrm{lex}} T$, if either they are equal or the $\preccurlyeq$-smallest element in which $S$ and $T$ differ is in $S$. We call a hypergraph on a totally ordered vertex set an *ordered hypergraph*

*2.2. Parameterised Complexity and the Strong Exponential Time Hypothesis* The *decision problem* of a set $\Pi \subseteq \{0,1\}^*$ is to answer for an *instance* $I \in \{0,1\}^*$ whether $I \in \Pi$. Such a problem is *parameterised* if $I$ comes augmented with a non-negative integer *parameter* $k$. We then have $\Pi \subseteq \{0,1\}^* \times \mathbb{N}^+$. A parameterised problem is *fixed-parameter tractable* (FPT), if there exists a computable function $f\colon \mathbb{N}^+ \to \mathbb{N}^+$ such that the input $(I,k)$ can be decided in time $f(k) \cdot \mathsf{poly}(|I|)$. The class of all fixed-parameter tractable problems is denoted by $\mathsf{FPT}$. Slightly abusing notation, we say any algorithm (not only for decision problems) that takes time $f(k) \cdot \mathsf{poly}(|I|)$ runs in *FPT-time*. We mostly employ more expressive quantizations of the input size, which are still polynomially related to the encoding length $|I|$. For example, if the instance $I = (V, \mathcal{H})$ is a hypergraph, we use $n = |V|$ and $m = |\mathcal{H}|$.

Let $\Pi$ and $\Pi'$ be parameterised problems. A *parameterised reduction* from $\Pi$ to $\Pi'$ is a function computable in FPT-time that maps an instance $(I,k)$ of $\Pi$ to an equivalent instance $(I',k')$ of $\Pi'$ such that there is a computable function $g$ with $k' \leq g(k)$. A parameterised reduction is called *linear* [18] if $g$ is a linear function, that is, if $k' = \mathrm{O}(k)$. All parameterised reductions we give in this work are linear.

Parameterised reductions give rise to a hierarchy of complexity classes, the $\mathsf{W}$*-hierarchy*. There are several equivalent ways to define it [36]; we choose the one in terms of circuits. A *(Boolean) circuit* is a directed acyclic graph whose vertex set consists of input nodes, NOT-, AND-, and OR-gates, with the obvious semantics, and a single output node. AND- and OR-gates have potentially unbounded fan-in. A *(Boolean) formula* is a circuit in which every gate has fan-out 1. The *depth* of a circuit is the maximum length of a path from an input to the output node. The *weft* is the maximum number of *large gates* with fan-in larger than 2 on any path. The Weighted Circuit Satisfiability problem is to decide for a given circuit $C$ and a positive integer $k$ whether $C$ has a satisfying assignment of *(Hamming) weight* $k$, that is, with exactly $k$ input nodes set to true. The parameter is $k$. The class $\mathsf{W}[\mathsf{P}]$ is the collection of all parameterised problems that admit a parameterised reduction to Weighted Circuit Satisfiability. Analogously, for or any positive integer $t$, the Weighted Circuit Satisfiability restricted to circuits of constant depth and weft at most $t$ is the defining complete problem for the class $\mathsf{W}[t]$. The classes $\mathsf{FPT} \subseteq \mathsf{W}[1] \subseteq \mathsf{W}[2] \subseteq \mathsf{W}[3] \subseteq \cdots \subseteq \mathsf{W}[\mathsf{P}]$ form the $\mathsf{W}$*-hierarchy*. All inclusions are conjectured to be strict, see [27, 52]

Another source of conditional lower bounds is the *Strong Exponential Time Hypothesis* (SETH) [44]. It states that, for every $\varepsilon > 0$, there exists

a positive integer $k = k(\varepsilon)$ such that no algorithm can decide the satisfiability of Boolean formulas in conjunctive normal form with $k$ literals per clause ($k$-CNF SAT) on $n$ variables in time $O(2^{(1-\varepsilon)n})$. A weaker assumption is the *Exponential Time Hypothesis* (ETH) [43, 44] that 3-CNF SAT cannot be solved in time $2^{o(n)}$. ETH implies that the $\mathsf{W}[1]$-complete Independent Set problem on $n$-vertex, $m$-edge graphs cannot be solved in time $f(k) \cdot (m+n)^{o(k)}$, whence $\mathsf{W}[t] \neq \mathsf{FPT}$ for all $t \geq 1$ [18].

### *2.3. Enumeration Complexity*

It is enough for our purposes to define *enumeration* informally as the task of computing and outputting all solutions to a computational problem without repetition.[1] We are only concerned with the *transversal hypergraph problem*, that is, given a hypergraph $\mathcal{H}$, enumerating the edges of $\mathrm{Tr}(\mathcal{H})$. An *output-polynomial* enumeration algorithm runs in time polynomial in both the input and output size. That means the enumeration succeeds within $\mathsf{poly}(n, m, N_{\min})$ steps. A seemingly stronger requirement is an *incremental polynomial* algorithm, generating the solutions in such a way that the $i$-th *delay*, the time between the $(i-1)$-st and $i$-th output, is in $\mathsf{poly}(n, m, i)$. This includes the preprocessing time until the first solution arrives ($i = 1$) and the postprocessing time between the last solution and termination ($i = N_{\min}+1$). It is known that the transversal hypergraph problem can be solved in output-polynomial time if and only if it admits an incremental polynomial algorithm [6]. This is not necessarily true for other enumeration problems. The strongest form of output-efficiency is that of *polynomial delay*, where the delay is universally bounded by a polynomial in the input size only. One can also restrict the space consumption. Ideally, the algorithm only uses space polynomial in the input. Even if $N_{\min}$ is guaranteed to be polynomial in $m$ and $n$, the space should be independent of $N_{\min}$.

### *2.4. Relational Databases and Unique Column Combinations*

To describe relational data, we fix a non-empty, finite *(relational) schema* $R$. The elements of $R$ are the *attributes* or *columns* and each attribute comes implicitly associated with a set of admissible values. *Rows* over $R$ are tuples $r$ whose entries are indexed by $R$ such that, for each $a \in R$, the *value* $r[a]$ is admissible for attribute $a$. For a set $X \subseteq R$ of columns, we let $r[X]$ denote

[1] Enumeration should not be confused with merely counting the number of solutions.

the subtuple of $r$ consisting only of the entries indexed by $X$. A *(relational) database* $\mathfrak{r}$ over $R$ is a finite set of rows.

In some database $\mathfrak{r}$ over schema $R$, a set $X \subseteq R$ is a *unique column combination* (UCC) if for any two distinct rows $r, s \in \mathfrak{r}$, $r \neq s$, we have $r[X] \neq s[X]$. A UCC is *(inclusion-wise) minimal* if it does not properly contain any other UCC. There is a one-to-one correspondence between UCCs and transversals. Let $r, s \in \mathfrak{r}$ be distinct rows and $\{a \in R \mid r[a] \neq s[a]\}$ their *difference set*. Then, the (minimal) UCCs are exactly the (minimal) hitting sets of the hypergraph of difference sets for all pairs of rows in $\mathfrak{r}$.

## 3. Enumerating Minimal Hitting Sets

In this section, we outline our enumeration algorithm for minimal hitting sets. Our main motivation comes from data profiling, so we design our method with an eye on instances that have small solutions. Nevertheless, we aim for a general-purpose algorithm and do not restrict the possible input hypergraphs. Therefore, the algorithm does not make any assumptions on the inputs and relies only on the given hypergraph itself. Its analysis, however, incorporates the transversal rank. Before we present our algorithm, we discuss some alternative approaches and asses how we can improve upon them.

### *3.1. On the Transversal Rank*

We review here what is known algorithmically about the transversal rank $k^*$. This also serves to highlight the subtle differences in measuring the complexity of an enumeration algorithm. Eiter and Gottlob [30] showed that the transversal hypergraph problem can be solved in incremental polynomial time on instances for which $k^*$ is bounded. Their result hinges on the following proposition.

**Proposition 6** (Eiter and Gottlob [30])**.** *Let $\mathcal{H}$ and $\mathcal{G}$ be two hypergraphs on the same vertex set $V$ and let $k = \operatorname{rank}(\mathcal{G})$. There exists an algorithm that decides whether $\mathcal{G} = \operatorname{Tr}(\mathcal{H})$ in time $\mathrm{O}(|\mathcal{H}||\mathcal{G}||V| + (|\mathcal{H}|+|\mathcal{G}|)|V|^{k+1} + |\mathcal{H}|^{k+2}|V|)$ and space $\mathrm{O}((|\mathcal{H}|+|\mathcal{G}|)|V|)$. Moreover, if $\mathcal{G} \subsetneq \operatorname{Tr}(\mathcal{H})$, the algorithm finds a minimal transversal $T \in \operatorname{Tr}(\mathcal{H}) \setminus \mathcal{G}$ within the same bounds.*

The enumeration starts with the empty hypergraph $\mathcal{G} = \emptyset$ and repeatedly checks whether $\mathcal{G} = \operatorname{Tr}(\mathcal{H})$, that is, whether $\mathcal{G}$ already contains all solutions. If not, a new solution $T \notin \mathcal{G}$ is computed. Note that $\mathcal{G} \subseteq \operatorname{Tr}(\mathcal{H})$ is an invariant, whence $k = \operatorname{rank}(\mathcal{G})$ is always at most $k^* = \operatorname{rank}(\operatorname{Tr}(\mathcal{H}))$. However,

this approach has two drawbacks. It is already unfortunate that the delay depends on $|\mathcal{G}|$ and thus on $|\mathrm{Tr}(\mathcal{H})|$, but it is indeed prohibitive in practice that the space consumption scales with the number of solutions.

If one is working with a class of hypergraphs for which one suspects $k^*$ to be small, albeit no a priori bound is known, one could be tempted to compute the transversal rank first and then brute-force all sets up to that size. Computing $k^*$ is NP-hard [20, 21]. Bazgan et al. [3] further showed that it is W[1]-hard, parameterised by $k^*$, and that $k^*$ cannot be approximated within a factor of $n^{1-\varepsilon}$ for any constant $\varepsilon > 0$, unless P = NP.

The parameterised hardness stems from the potentially unbounded size of the hyperedges. Fernau [35] showed that the transversal rank of a graph[2] can be computed in FPT-time by presenting an algorithm running in time $\mathrm{O}(2^{k^*}) + \mathsf{poly}(n)$, which was later improved to $1.5397^{k^*} \cdot \mathsf{poly}(n)$ [10]. For an arbitrary constant $c$, Damaschke [24] used Proposition 6 to give an algorithm that computes the transversal rank of a hypergraph whose edges have at most $c$ vertices in time $c^{k^*} \cdot p_c(m, n)$, where $p_c$ is a polynomial whose degree depends on $c$. If $c$ is seen as another parameter (namely, computing $k^*$ parameterized by $c + k^*$), there exists an FPT-algorithm running in time $2^{ck^*} \cdot \mathsf{poly}(m, n)$ [2].

Returning to hypergraphs with unbounded edge size, the parameterized reduction in [3, Theorem 23] that shows the W[1]-hardness of computing the transversal rank has a quadratic blowup in the parameter. The lower bound on Independent Set by Chen et al. [18] (see Section 2.2) thus implies that $k^*$ cannot be computed in time $f(k^*)(m+n)^{\mathrm{o}(\sqrt{k^*})}$ for any computable function $f$, unless the Exponential Time Hypothesis fails. Very recently and independently of each other, Araújo et al. [2] as well as Dublois, Lampis, and Paschos [28] raised the bound to $f(k^*)(m+n)^{\mathrm{o}(k^*)}$. This essentially matches the currently fastest algorithm, which uses the following characterization of the transversal rank by Berge and Duchet [5]. For a hypergraph $(V, \mathcal{H})$, subhypergraph $\mathcal{H}' \subseteq \mathcal{H}$, and vertex $v \in V$, let $\deg_{\mathcal{H}'}(v) = |\{E \in \mathcal{H}' \mid v \in E\}|$ denote the *degree of $v$ in $\mathcal{H}'$*.

**Proposition 7** (Berge and Duchet [4, 5])**.** *Let $(V, \mathcal{H})$ be a Sperner hypergraph and $k \geq 2$ an integer. The transversal rank of $\mathcal{H}$ is at most $k$ if and only if, for all subhypergraphs $\mathcal{H}' \subseteq \mathcal{H}$ with $|\mathcal{H}'| = k + 1$ edges, there exists an edge of $\mathcal{H}$ that is contained in the set $\{v \in V \mid \deg_{\mathcal{H}'}(v) > 1\}$.*

[2] This is more commonly known as Maximum Minimal Vertex Cover [10, 35].

Recall from Section 2.1 that the assumption of the input hypergraph being Sperner does not loose generality. One can thus test the condition of Proposition 7 for increasing $k$. The value that satisfies it for the first time is $k^*$. The last iteration dominates the running time, which gives a bound of $\mathrm{O}\Big(\binom{m}{k^*+1}(k^*+m)\,n\Big) = \mathrm{O}(m^{k^*+2}\,n)$. The subsequent test of the sets with up to $k^*$ vertices adds another $\mathrm{O}(mn^{k^*})$ term. The enumeration time is polynomial for bounded $k^*$, but the algorithm does not admit any non-trivial guarantees on the delay. Also, the space requirement again depends on the total number of solutions since the algorithm has to avoid testing supersets of minimal solutions.

In contrast, we give an algorithm with delay $\mathrm{O}(m^{k^*+1}\,n^2)$, which is better than the bound above for $m > n$. More importantly though, our algorithm uses space that is only linear in the input size regardless of $k^*$.

### *3.2. Backtracking Enumeration with an Extension Oracle*

It is a common pattern in the design of enumeration algorithms to base them on a so-called *extension oracle* as introduced by Lawler [49]. The oracle, tailored to the combinatorial problem at hand, is queried with a set of elements of the underlying universe and decides whether there exists a solution that contains these elements. Applications of this technique usually involve settings in which the extension problem is solvable in polynomial time, like for cycles and spanning trees [55], motif search in graphs [7], or satisfying assignments for restricted fragments of propositional logic [22]. For us, the situation is different in that the extension problem for minimal hitting sets is NP-complete [13]. We show later in Section 4 that the problem is also hard in a parameterised sense. At first, it may seem paradoxical that reducing enumeration to a hard decision problem can speed up the resulting algorithm. We exploit the fact that the time needed to solve the extension problem is small (enough) for sets that contain only a few vertices.

The original oracle technique [49] consists of fixing certain elements of the partial solution and then extending it to the optimum, with respect to a certain ranking function, among all objects that share the fixed elements. During the computation the new candidates are stored in a priority queue. The main bottleneck is the space demand of the queue. For every partial solution, the number of newly introduced candidates can be equal to the size of the universe, meaning exponential growth. Therefore, modifications are necessary to implement the technique efficiently.

In addition to the extension oracle, we use a decision tree to guide the search for minimal solutions in the power lattice of all subsets of the universe. This is known as *backtracking enumeration* [55] or *flashlight technique* [51]. It allows us to reduce the space requirement to only polynomial in the input.

In the following, we show how to combine both ideas. Let $(V, \mathcal{H})$ be a hypergraph. Suppose we are given an oracle that, queried with disjoint sets $X, Y \subseteq V$, answers whether there exists a minimal solution $T \in \mathrm{Tr}(\mathcal{H})$ such that $X \subseteq T \subseteq V \backslash Y$, that is, whether $X$ is extendable avoiding $Y$. We use this to enumerate all such $T$. If $X \cup Y = V$, this can only be $T = X$ itself. Otherwise, we recursively compute the solutions for the pairs $(X \cup \{v\}, Y)$ and $(X, Y \cup \{v\})$, where $v$ is a vertex neither contained in $X$ nor $Y$. In other words, we (implicitly) build a binary tree whose nodes are labelled with the pairs $(X, Y)$. The node $(\emptyset, \emptyset)$ is the root and the children of $(X, Y)$ are $(X \cup \{v\}, Y)$ and $(X, Y \cup \{v\})$. Let $\preccurlyeq$ be a total order on $V$. Always choosing the $v$ as the $\preccurlyeq$-smallest element of $V \backslash (X \cup Y)$ gives a universal branching order. This obviates the need of additional communication between the nodes or any shared memory. It is another ingredient to reduce the space demand. In particular, we do not need to record previously found solutions to guide the search. Distinct branches of the tree are independent making the algorithm trivially parallelisable. This is, however, not the focus of this work.

In the absence of any pruning, the recursion would produce the full binary tree with leaves $(X, V \backslash X)$ for *every* possible set $X \in \mathcal{P}(V)$. However, we only need to enter the subtree if one of its leaves is labelled with a minimal hitting set. For the subtree rooted in $(X, Y)$, this is the case iff $X$ can be extended to a minimal hitting set without the vertices in $Y$.

We formalise this approach in Algorithm 1. Assume for now that subroutine `extendable`$(X, Y)$ solves the extension problem for the given pair of sets and additionally reports if $X$ itself is already a minimal hitting set. Namely, it returns MINIMAL if $X \in \mathrm{Tr}(\mathcal{H})$, TRUE if there exists some $T \in \mathrm{Tr}(\mathcal{H})$ with $X \subsetneq T \subseteq V \backslash Y$, and FALSE otherwise. We defer the implementation details of `extendable` to Section 5. Procedure `enumerate` handles the work inside a node of the decision tree. Besides the depth-first, pre-order traversal of the tree, it also exercises two short-cut evaluations. For this, the ternary variable *isExpendable* holds the result of the first check. If the set $X \cup \{v\}$ is a minimal solution, we output it and return. If it cannot be extended, we immediately recurse on the right-child *without* calling the potentially expensive second check `extendable`$(X, Y \cup \{v\})$. The initial call is `enumerate`$(\emptyset, \emptyset, V)$.

**Algorithm 1:** Recursive algorithm for the Transversal Hypergraph problem. The initial call is `enumerate`$(\emptyset, \emptyset, V)$.

**Data:** Non-empty ordered hypergraph $(V, \preccurlyeq, \mathcal{H})$.
**Input:** Partition $(X, Y, R)$ of the vertex set $V$.
**Output:** The minimal hitting sets $T \in \mathrm{Tr}(\mathcal{H})$ with $X \subseteq H \subseteq V \backslash Y$.

1 **Procedure** `enumerate`$(X, Y, R)$:
2 **if** $R = \emptyset$ **then return** $X$;
3 $v \leftarrow \min_{\preccurlyeq} R$;
4 *isExtendable* $\leftarrow$ `extendable`$(X \cup \{v\}, Y)$;
5 **if** *isExtendable* $==$ MINIMAL **then return** $X \cup \{v\}$;
6 **if** *isExtendable* $==$ TRUE **then** `enumerate`$(X \cup \{v\}, Y, R \backslash \{v\})$;
7 **if** (*isExtendable* $==$ FALSE **or** `extendable`$(X, Y \cup \{v\}) ==$ TRUE)
**then** `enumerate`$(X, Y \cup \{v\}, R \backslash \{v\})$;

**Lemma 8.** *Let $(V, \mathcal{H}, \preccurlyeq)$ be a non-empty ordered hypergraph. Suppose, for disjoint sets $X, Y \subseteq V$, subroutine* `extendable`$(X, Y)$ *decides whether there exists a $T \in \mathrm{Tr}(\mathcal{H})$ with $X \subseteq T \subseteq V \backslash Y$ and, if so, whether $X = T$. Then, Algorithm 1 enumerates the edges of $\mathrm{Tr}(\mathcal{H})$ in $\preccurlyeq$-lexicographical order.*

*Proof.* The correctness is almost immediate from the discussion above. Only the shortcut evaluations have not yet been argued. If the set $X \cup \{v\}$ is not only extendable without $Y$, but even minimal itself, then adding any more vertices from $V \backslash (X \cup Y \cup \{v\})$ will make it unextendable. Adding these vertices to $Y$ instead does not change $X \cup \{v\}$ being a minimal solution. In summary, we already know in advance the outcomes of all extension checks in the whole subtree rooted in $(X \cup \{v\}, Y)$. The set $X \cup \{v\}$ is the only solution that remains in that tree and we can safely output it and backtrack.

Regarding the second shortcut in line 7, the recursion enters the node $(X, Y)$ only if there exists a $T \in \mathrm{Tr}(\mathcal{H})$ with $X \subsetneq T \subseteq V \backslash Y$. If the first check `extendable`$(X \cup \{v\}, Y)$ returns FALSE, no such $T$ contains the vertex $v$. Instead, all solutions occur in the subtree rooted at the right child $(X, Y \cup \{v\})$ and we do not need to perform the second evaluation. Note that `extendable`$(X, Y \cup \{v\})$ cannot return MINIMAL due to $X \neq T$.

In the extreme case of $\mathcal{H}$ having not a single hitting set (that is, $\emptyset \in \mathcal{H}$) both checks in lines 6 and 7 fail already in the root node. The algorithm

then returns immediately without an output. Here, we use the assumption that $\mathcal{H}$ has at least one vertex and thus $R = V \neq \emptyset$ holds in the root.

Finally, we prove that the algorithm outputs the minimal transversals in lexicographical order. First, observe that the labelling of the nodes is injective as it encodes the unique path from the root. To see this, let $v_1 \preccurlyeq v_2 \preccurlyeq \ldots \preccurlyeq v_n$ be the total order. Any node with distance $k$ to the root has $X \cup Y = \{v_1, \ldots, v_k\}$ and $X$ contains exactly those branching nodes at which the recursion entered the left child. Now let $a = (X_a, Y_a)$ and $b = (X_b, Y_b)$ be two distinct leaves such that the pre-order traversal visits $a$ before $b$. We have $X_a \neq X_b$ from the injective labelling, whence the symmetric difference $X_a \triangle X_b = (X_a \backslash X_b) \cup (X_b \backslash X_a)$ is non-empty. Define $v = \min_{\preccurlyeq} X_a \triangle X_b$. This is the branching vertex of the lowest common ancestor of $a$ and $b$. Algorithm 1 first tries to add $v$ to the current partial solution in line 6, from which $v \in X_a$ and $X_a \preccurlyeq_{\text{lex}} X_b$ follow. □

Algorithm 1 bears some similarity to the backtracking method by Elbassioni, Hagen, and Rauf [33, Figure 1]. The main difference is the search for new solutions. In our algorithm, the nodes in the decision tree maintain the partial solution $X$ and additionally the set $Y$ of vertices that have already been excluded. The branching vertex $v$ is chosen, somewhat arbitrarily, by the order $\preccurlyeq$. In contrast, the algorithm in [33] works only on the partial solution $X$ and explicitly computes a new vertex to extend it, which is computationally expensive. Also, their check whether $X$ is already minimal is redundant. This information can be obtained as a by-product of a careful implementation of `extendable` at no extra cost, see Lemma 18.

We employ the order on the vertex set to reduce the need for coordination during the search. The induced lexicographic order on the outputs can also be useful in the application domain. For example in the context of data profiling, it ensures that "interesting" unique column combinations are discovered first. Suppose the attributes of a database are ranked by importance, then the lexicographic enumeration starts with those combinations that contain many important attributes. However, the order also raises some complexity-theoretic issues. Computing the lexicographically smallest minimal hitting set is NP-hard [29, 45]. Therefore, it is unlikely that any implementation of the extension subroutine can lead to Algorithm 1 having polynomial delay on *all* ordered hypergraphs. Notwithstanding, we present an implementation such that our algorithm achieves polynomial delay at least on instances with bounded transversal rank. We also evaluate the impact of the order on the

empirical run time on real-world databases in our experiments in Section 6.

## 4. Minimal Hitting Set Extension

We previously assumed an oracle deciding whether a set of vertices can be extended to a minimal hitting sets. Here, we examine the computational hardness of this decision. The insights gained here will later lead to an algorithm for the subroutine with an almost optimal running time.

It is easy to see that, for a hypergraph $(V, \mathcal{H})$ and disjoint sets $X, Y \subseteq V$, there exists a minimal transversal $T \in \mathrm{Tr}(\mathcal{H})$ such that $X \subseteq T \subseteq V \backslash Y$ if and only if the *truncated* hypergraph $\mathcal{H}' = \{E \backslash Y \mid E \in \mathcal{H}\}$ has a minimal hitting set $T$ with $X \subseteq T$. Indeed, the witnessing transversal $T$ is the same for both $\mathcal{H}$ and $\mathcal{H}'$. We thus define the extension problem as follows.

Minimal Hitting Set Extension (MinHSExt)

*Instance:* A hypergraph $(V, \mathcal{H})$ and a sets $X \subseteq V$.

*Parameter:* The cardinality $|X|$.

*Decision:* Is there a minimal hitting set $T$ of $\mathcal{H}$ with $X \subseteq T$?

Boros, Gurvich, and Hammer showed that the unparameterised variant of MinHSExt is NP-complete in general but tractable if $|X|$ is bounded [13]. This and the fact that minimal hitting sets in many applications are small warrants a parameterised investigation with respect to the cardinality $|X|$. Observe that MinHSExt generalises the extension problem for minimal vertex covers in graphs. Casel et al. [16] proved W[1]-completeness of the latter.

In [13], MinHSExt was reduced to a certain covering problem in hypergraphs. We extend this result by proving that the extension and covering problems are in fact equivalent under parameterised reductions. We then use this equivalence to show that Minimal Hitting Set Extension is one of the first natural problems to be complete for the parameterised complexity class W[3]. We further prove conditional lower bounds on the running time of any algorithm for the extension problem, assuming that certain collapses in the W-hierarchy do not occur or that the Strong Exponential Time Hypothesis is true, respectively.

### *4.1.* W[3]*-Completeness*

We present necessary and sufficient conditions for a set of vertices to be a subset of some minimal hitting set. This naturally extends the characterisa-

tion of minimal transversals in Proposition 5. The result appears implicitly in [13], we give a self-contained proof below.

**Proposition 9** (Boros, Gurvich, and Hammer [13])**.** *Let $(V, \mathcal{H})$ be a hypergraph and $X \subseteq V$ a set of vertices. There is a $T \in \mathrm{Tr}(\mathcal{H})$ with $X \subseteq T$ if and only if there exists a family of edges $\{E_x\}_{x \in X} \subseteq \mathcal{H}$ such that*

*(i) for every vertex $x \in X$, we have $E_x \cap X = \{x\}$;*

*(ii) for every edge $E \in \mathcal{H}$ contained in $\bigcup_{x \in X} E_x$, we have $E \cap X \neq \emptyset$.*

*Proof.* Let $T$ be a minimal hitting set that contains $X$. Proposition 5 guarantees a private edge $E_x \in \mathcal{H}$ with respect to $T$ for every $x \in X$. Let further $E \in \mathcal{H}$ be such that $E \subseteq \bigcup_{x \in X} E_x$. As $T$ is a hitting set, there exists a vertex $y \in E \cap T$. From $(\bigcup_{x \in X} E_x) \cap T = X$, we conclude that $y$ must be in $X$. Hence, the private edges also fulfil Condition *(ii)*.

Conversely, suppose $\{E_x\}_{x \in X}$ is a suitable collection of hyperedges. Condition *(ii)* implies that $H = X \cup (V \setminus \bigcup_{x \in X} E_x)$ is a (not necessarily minimal) hitting set of $\mathcal{H}$. Let $T \subseteq H$ be any *minimal* hitting set, then $T$ contains every $x \in X$ as otherwise $E_x$ would not intersect $T$ by Condition *(i)*. □

We call an edge $E$ a *candidate private edge* for $x \in X$ (with respect to set $X$) if $E \cap X = \{x\}$ holds. The partial solution $X$ has some extension $T \in \mathrm{Tr}(\mathcal{H})$ iff there is a collection of candidate private edges $\{E_x\}_{x \in X}$ that satisfy Condition *(ii)*. Then, the $E_x$ indeed serve as private edges with respect to $T$ in the sense of Proposition 5.

In light of this characterisation, we define an intermediate parameterised problem, which we call Multicoloured Independent Family. It captures the following computational task: given $k$ lists of sets together with an additional collection of "forbidden" sets, one has to select one set from each list such that they do not completely cover any forbidden set.

Multicoloured Independent Family (MultIndFam)

*Instance:* A $(k{+}1)$-tuple $(\mathcal{S}_1, \ldots, \mathcal{S}_k, \mathcal{T})$ of hypergraphs on the common vertex set $U$.

*Parameter:* The non-negative integer $k$.

*Decision:* Are there edges $S_1 \in \mathcal{S}_1, \ldots, S_k \in \mathcal{S}_k$ such that $\bigcup_{i=1}^{k} S_i$ does not contain an edge of $\mathcal{T}$?

The Multicoloured Independent Family problem generalises Multicoloured Independent Set on graphs where the vertex set is partitioned into $k$ "colour classes" and the desired independent set is required to contain one vertex of each colour [23, 34]. In the generalisation, we instead select *sets* of vertices such that their *union* has to be independent. Now the sets have "colours" and the $\mathcal{S}_i$ represent the colour classes. Multicoloured Independent Set is the special case in which the hypergraphs $\mathcal{S}_i$ consist entirely of singletons and $\mathcal{T}$ of the edges of the graph. Evidently, Multicoloured Independent Family is $\mathsf{W}[1]$-hard.

We now prove the equivalence between MultIndFam and MinHSExt. We report the features of the second reduction in full detail as we need them later for the fine-grained lower bounds.

**Lemma 10.** Minimal Hitting Set Extension *and* Multicoloured Independent Family *are equivalent under linear parameterised reductions.*

*The reduction to the* Minimal Hitting Set Extension *problem takes time* $\mathrm{O}((\sum_{i=1}^{k} |\mathcal{S}_i| + |\mathcal{T}|) \cdot |U|)$ *and results in instances with* $n = |U| + k$ *vertices,* $m = \sum_{i=1}^{k} |\mathcal{S}_i| + |\mathcal{T}|$ *edges, and parameter* $|X| = k$.

*Proof.* Let $(\mathcal{H}, X)$ be the input to MinHSExt. The set $X$ is extendable iff there are edges $\{E_x\}_{x \in X} \in \mathcal{H}$ with $E_x \cap X = \{x\}$ and their union $\bigcup_{x \in X} E_x$ does not contain any edge that is disjoint from $X$. This can be phrased as an instance of MultIndFam by defining, for each $x \in X$, the hypergraph $\mathcal{S}_x = \{E \in \mathcal{H} \mid E \cap X = \{x\}\}$. The last hypergraph $\mathcal{T}$ consists of the edges that are disjoint from $X$. Edges that intersect $X$ in more than one vertex can be cast aside. This is indeed a linear parameterised reduction.

For the inverse direction, let $(U, \mathcal{S}_1, \ldots, \mathcal{S}_k, \mathcal{T})$ be the instance of Multicoloured Independent Family. Let $X = \{x_1, \ldots, x_k\}$ be a set of $k$ new vertices not previously in $U$. We define the hypergraph $\mathcal{H}$ on the vertex set $V = U \cup X$ by adding all edges of $\mathcal{T}$ as well as $E \cup \{x_i\}$ for every $i \in [k]$ and $E \in \mathcal{S}_i$. Hypergraph $\mathcal{H}$ can be computed in time

$$\mathrm{O}\left(\sum_{i=1}^{k} \sum_{E \in \mathcal{S}_i} |E| + \sum_{E' \in \mathcal{T}} |E'|\right) = \mathrm{O}\left(\left(\sum_{i=1}^{k} |\mathcal{S}_i| + |\mathcal{T}|\right) \cdot |U|\right).$$

For any set $S \subseteq V$, the containment $S \in \mathcal{S}_i$ is equivalent to $S \cap X = \{x_i\}$. Moreover, the elements of $\mathcal{T}$ are exactly those edges of $\mathcal{H}$ that are disjoint

from $X$. Therefore, there are $S_1 \in \mathcal{S}_1, \ldots, S_k \in \mathcal{S}_k$ such that $E \not\subseteq \bigcup_{i=1}^{k} S_i$ holds for all $E \in \mathcal{T}$ if and only if $\{S_i\}_{i\in[k]}$ satisfies Conditions (*i*) and (*ii*) of Proposition 9, that is, iff $X$ is extendable to a minimal hitting set of $\mathcal{H}$. □

The rich structure of Multicoloured Independent Family is appreciated when designing algorithms. For the discussion of its complexity, however, it is convenient to also have the freedom to choose the sets from a single list. We thus define the following variant without colours.

Independent Family (IndFam)

*Instance:* Two hypergraph $\mathcal{S}$, $\mathcal{T}$ on the common vertex set $U$ and a non-negative integer $k$.

*Parameter:* The non-negative integer $k$.

*Decision:* Are there $k$ distinct edges $S_1, \ldots, S_k \in \mathcal{S}$ such that $\bigcup_{i=1}^{k} S_i$ does not contain an edge of $\mathcal{T}$?

The two variants are indeed equivalent.

**Lemma 11.** Multicoloured Independent Family *and* Independent Family *are equivalent under linear parameterised reductions.*

*Proof.* To reduce MultIndFam to its uncoloured variant, it is enough to enforce that selecting two sets of the same colour is never a correct solution. They must always cover some forbidden set. Let $(\mathcal{S}_1, \ldots, \mathcal{S}_k, \mathcal{T})$ be an instance of MultIndFam. For every index $i \in [k]$, and $S \in \mathcal{S}_i$, we introduce a new element $x_{S,i}$. The sets are augmented with their respective elements, $S \cup \{x_{S,i}\}$, and the results are collected in the single hypergraph $\mathcal{S}$. Adding the pair $\{x_{S,i}, x_{S',i}\}$ to $\mathcal{T}$ for each $i$ and $S \neq S' \in \mathcal{S}_i$ invalidates all unwanted selections. It is easy to check that this destroys no valid solution.

For the other direction, we make $k$ copies of $\mathcal{S}$ and ensure that no two copies of the same set are selected together. In more detail, we take a new element $x_{S,i}$ for each $S \in \mathcal{S}$ and $i \in [k]$, define $\mathcal{S}_i = \{S \cup \{x_{S,i}\} \mid S \in \mathcal{S}\}$, and add the sets $\{x_{S,i}, x_{S,j}\}_{i\neq j}$ to $\mathcal{T}$. □

In the remainder of this section, we prove that Independent Family is complete for the class W[3]. This transfers to Multicoloured Independent Family and eventually to Minimal Hitting Set Extension via the reductions in Lemmas 10 and 11.

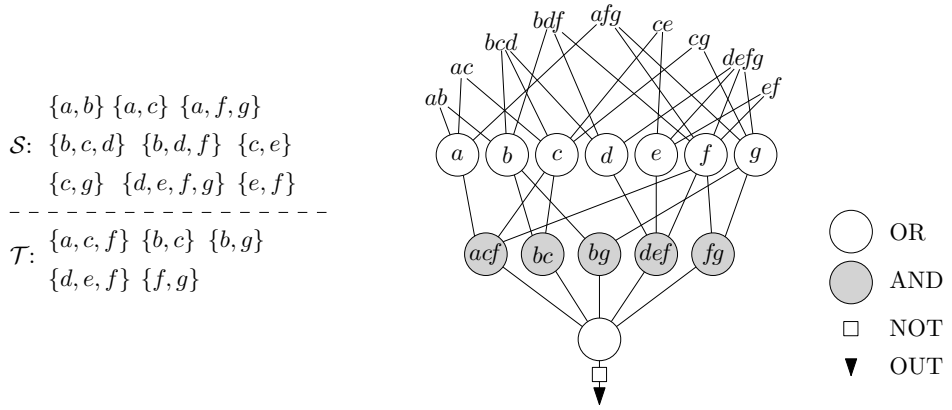



Figure 1: Illustration of Lemma 12. On the left side is an instance of Independent Family, on the right is the resulting circuit of weft 3. All edges are directed downwards. Selecting $\{a,c\}$, $\{c,e\}$, and $\{c,g\}$ from $\mathcal{S}$ solves the instance for parameter $k=3$, any other combination of three sets covers a member of $\mathcal{T}$.

**Lemma 12.** *There is a linear parameterised reduction from* Independent Family *to the* Weighted Circuit Satisfiability *problem on constant-depth circuits of weft* 3. *In particular,* Independent Family *is in* $\mathsf{W}[3]$.

*Proof.* Given an instance $I=(U,\mathcal{S},\mathcal{T},k)$ of IndFam, we build a Boolean circuit $C$ of weft 3 that has a satisfying assignment of Hamming weight $k$ iff $I$ is a yes-instance. Figure 1 shows an example instance and the resulting circuit. The nodes of $C$ are in one-to-one correspondence to objects in $I$, slightly abusing notation we do not distinguish between nodes and their object. The input nodes are the edges of $\mathcal{S}$. Circuit $C$ has a large OR-gate for each vertex in $u \in U$. Node $S \in \mathcal{S}$ is wired to gate $u$ whenever $u \in S$. Next, we introduce a layer of large AND-gates, one for each forbidden set $E \in \mathcal{T}$. Again, $u$ is connected to $E$ iff $u \in E$. The output of all AND-gates lead to a single large OR-gate, its *negated* output is the output of $C$.

Note that the circuit can be constructed from instance $I$ in polynomial time. It has depth 4 and weft 3 as every path from an input node to the output passes through exactly one large gate in each if the 3 layers and the (small) NOT-gate. We claim that $C$ is satisfied by setting the input nodes $S_1, \ldots, S_k$ to TRUE if and only if the union $\bigcup_{i=1}^{k} S_i$ contains no edge of $\mathcal{T}$.

Let $S_1$ to $S_k$ be a selection of $k$ distinct edges of $\mathcal{S}$. Assigning TRUE to the $S_i$ and FALSE to all others satisfies exactly the OR-gates $u \in \bigcup_{i=1}^{k} S_i$. Any AND-gate $E$ of the second layer is satisfied iff all its feeding OR-gates

are satisfied, that is, iff $E \subseteq \bigcup_{i=1}^{k} S_i$. The results for all forbidden edges $E \in \mathcal{T}$ are collected by the large OR-gate in the third layer and subsequently negated. Circuit $C$ being satisfied is thus equivalent to *no* edge $E$ being contained in the union of $S_1, \ldots, S_k$. □

To also show hardness for $\mathsf{W}[3]$, we instead reduce from a problem on Boolean formulas, that is, circuits in which every gate has fan-out 1. A formula is called *antimonotone* and *3-normalised* if it is a conjunction of subformulas in disjunctive normal form (DNF) with only negative literals. An example of an antimonotone, 3-normalised formula is

$$((\overline{x}_1 \wedge \overline{x}_2 \wedge \overline{x}_4) \vee (\overline{x}_3 \wedge \overline{x}_4)) \wedge ((\overline{x}_1 \wedge \overline{x}_3) \vee (\overline{x}_2 \wedge \overline{x}_5) \vee (\overline{x}_1 \wedge \overline{x}_4 \wedge \overline{x}_5)).$$

The example has satisfying assignments of Hamming weight 0, 1, and 2, but none of larger weight. The Weighted Antimonotone 3-normalised Satisfiability problem (WA3NS) is the restriction of Weighted Circuit Satisfiability to antimonotone, 3-normalised formulas. It is complete for the third level of the $\mathsf{W}$-hierarchy [27, 36].

The intuition behind the $\mathsf{W}[3]$-hardness proof is as follows. The circuit $C$ constructed in Lemma 12 has a single NOT-gate as the output node. The OR-gates of the first layer are the only ones with fan-out larger than 1, but they are connected exclusively to gates of the second layer. Moving the negation all the way up to the inputs using De Morgan's laws, and duplicating the first layer at most $|\mathcal{T}|$ times hence results in an antimonotone formula that is indeed 3-normalised. We show that this is not a mere artefact of the reduction, but due to a characteristic property of the problem itself. Namely, every antimonotone, 3-normalised formula can be encoded in an instance of the Independent Family problem.

**Lemma 13.** *There is a linear parameterised reduction from* Weighted Antimonotone 3-normalised Satisfiability *to* Independent Family. *In particular,* Independent Family *is hard for* $\mathsf{W}[3]$.

*Proof.* A Boolean formula $\varphi$ on the variable set $\mathtt{Var}_\varphi$ is antimonotone and 3-normalised if and only if it can be written as

$$\varphi = \bigwedge_{d \in D} \bigvee_{c \in C_d} \bigwedge_{\ell \in L_{d,c}} \neg x_{d,c,\ell},$$

for an index hierarchy $D$, $\{C_d\}_{d \in D}$, $\{L_{d,c}\}_{d \in D, c \in C_d}$, and $x_{d,c,\ell} \in \mathtt{Var}_\varphi$. The

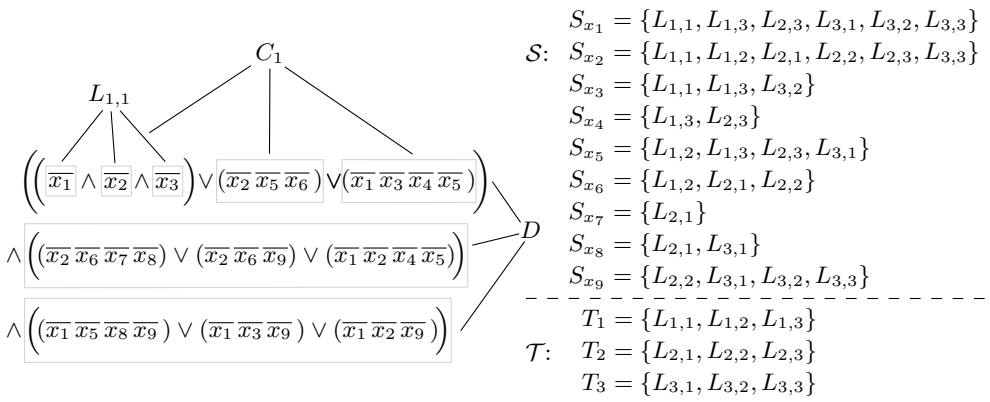



Figure 2: Illustration of Lemma 13. On the left side is an antimonotone, 3-normalised formula. Negative literals $\neg x_i$ are abbreviated as $\overline{x_i}$ and conjunctions inside a clause as juxtaposition. On the right is the resulting instance of Independent Family. Positions marked with grey boxes are indexed by the respective sets: $D$ for the DNF subformulas, $C_1$ for the conjunctive clauses of the first subformula, and $L_{1,1}$ for the first clause of the first subformula. The formula admits a satisfying assignment of weight 4 by setting $x_4$, $x_5$, $x_7$, and $x_8$ to TRUE. Equivalently, the union of the sets $S_{x_4}$, $S_{x_5}$, $S_{x_7}$, and $S_{x_8}$ does not cover any forbidden set in $\mathcal{T}$. No assignment of Hamming weight at least 5 is satisfying.

index $d$ ranges over the constituent DNF subformulas, $c$ over their conjunctive clauses, and $\ell$ over the negative literals in those clauses. Of course, a variable may appear multiple times in the formula, so different triples $(d, c, \ell)$ may point to the same variable.

We construct an instance $(U, \mathcal{S}, \mathcal{T}, k)$ of Multicoloured Independent Family that is a yes-instance if and only if $\varphi$ has a weight-$k$ satisfying assignment. This is illustrated in Figure 2. We take as vertex set the conjunctive clauses $U = \{L_{d,c} \mid d \in D,\, c \in C_d\}$ and add the edge $S_x = \{L_{d,c} \mid \exists \ell\colon x_{d,c,\ell} = x\}$ to $\mathcal{S}$ for each variable $x \in \mathtt{Var}_\varphi$. Namely, $S_x$ contains all clauses in which $x$ occurs. The DNF subformulas are represented in the hypergraph $\mathcal{T}$ via the edges $E_d = \{J_{d,c} \mid c \in C_d\}$ for all $d \in D$.

The key observation of this lemma is the following. Consider a truth assignment represented by the set $A \subseteq \mathtt{Var}_\varphi$ of the variables assigned TRUE. Since $\varphi$ is antimonotone, clause $L_{d,c}$ is satisfied if and only if *none* of its variables $x_{d,c,\ell}$ is in $A$. As a result, subformula $d$ is TRUE if and only if $A$ is *not* a hitting set for the clauses of $d$.

Suppose the assignment $A = \{x_1, \ldots, x_k\}$ is satisfying. Then, the union $\bigcup_{i=1}^{k} S_{x_i}$ contains exactly the conjunctive clauses that are not satisfied. If this union were to cover any forbidden edge in $\mathcal{T}$, the corresponding subformula,

and hence $\varphi$, would be unsatisfied, a contradiction. Therefore, $(U, \mathcal{S}, \mathcal{T}, k)$ is a yes-instance of Independent Family. Conversely, let $S_{x_1}$ through $S_{x_k}$ be a selection of edges from $\mathcal{S}$ such that their union covers no member of $\mathcal{T}$. In other words, each subformula has at least one clause that is disjoint from $\{x_1, \ldots, x_k\}$. Assigning true to (exactly) those variables $k$-satisfies $\varphi$. □

### *4.2. Fine-Grained Lower Bounds*

We now discuss consequences of our reductions beyond parameterised complexity. Namely, they allow us to derive lower bounds on the running time of any algorithm for Minimal Hitting Set Extension from certain hypotheses, which are, however, still unproven.

The common believe that the complexity classes P and NP are different can be seen as a conditional (super-polynomial) lower bound on the time complexity of NP-hard problems. Similar things can be said about the assumption W[1] ≠ FPT. Recently, this perspective has been further developed in the area of fine-grained complexity. It tries to determine the exact exponent of the time needed to solve various problems in the polynomial, exponential, and parameterised domain. The proven conditional lower bounds often match closely with the best known algorithmic results, but they come with the caveat of relying on even more unproven hardness assumptions. Such bounds need to strike a balance between the plausibility of the conjecture and the strength of the result following from it.

We offer three lower bounds on the extension problem. They are presented in order of increasing strength and are respectively derived from ever stronger conjectures about the W-hierarchy and Boolean satisfiability. The first one immediately follows from Minimal Hitting Set Extension being W[3]-complete. If W[3] ≠ FPT, there is no FPT-algorithm for extension running in time $f(|X|) \cdot \mathrm{O}((m+n)^c)$ on hypergraphs with $n$ vertices and $m$ edges for any computable function $f$ and constant $c$. Note that the parameterised reductions above also show that Multicoloured Independent Family and Independent Family cannot be solved respectively in time $f(k) \cdot \mathsf{poly}(|\mathcal{S}_1|, \ldots, |\mathcal{S}_k|, |\mathcal{T}|, |U|)$ and $f(k) \cdot \mathsf{poly}(|\mathcal{S}|, |\mathcal{T}|, |U|)$.

We derive the second lower bound from the stronger assumption that W[2] ≠ FPT. For this, we use the following proposition[3] by Chen et al. [18].

[3]The proposition follows from a more general result [18, Theorem 4.2] on the weighted satisfiability of what the authors call structured $\Pi_t$-circuits. For $t = 3$, the structure coincides with that of antimonotone, 3-normalised formulas.

**Proposition 14** (Chen et al. [18])**.** *Let $f$ be an arbitrary computable function. If there exists an algorithm solving the* Weighted Antimonotone 3-normalised Satisfiability *problem on formulas of size $m$ with $n$ variables in time $f(k)\, n^{o(k)}\, \mathsf{poly}(m)$, then $\mathsf{W}[2] = \mathsf{FPT}$.*

Note that the reductions from WA3NS to IndFam, and further to MultIndFam and MinHSExt in Lemmas 10, 11 and 13 are all polynomial-time computable and linear the sense that they increase the parameter by at most a constant factor. In fact, they even preserve the parameter exactly. Any algorithm solving the Minimal Hitting Set Extension problem in time $f(|X|)\,(m+n)^{o(|X|)}$ on $n$-vertex, $m$-edge hypergraphs would thus give a fast algorithm for Weighted Antimonotone 3-normalised Satisfiability and thus imply the collapse $\mathsf{W}[2] = \mathsf{FPT}$. Similar bounds also hold for the intermediate problems.

The above bound states that the exponent of the worst-case running time for MinHSExt necessarily has a linear dependence on the parameter $|X|$. We show next that the leading coefficient of that dependency is likely to be 1. Consider the so-called Orthogonal Vectors (OV) problem as an illustration of this kind of result. We are given two sets, each with $n$ binary vectors in $d$ dimensions, and we ought to decide whether there is one vector from each set such that their inner product is 0. Straightforwardly testing all pairs yields an $O(n^2 d)$-time algorithm. Maybe surprisingly, Williams [57] showed that this cannot be improved to $n^{2-\varepsilon} \cdot \mathsf{poly}(d)$ for any constant $\varepsilon > 0$, at least not if one believes that CNF SAT on formulas with $n$ variables cannot be solved in time $O(2^{(1-\varepsilon/2)n})$. Such an improved algorithm would be a huge breakthrough in satisfiability, its conjectured non-existence is the core of the Strong Exponential Time Hypothesis.

We derive our hardness result from a generalisation of OV, known as $k$-Orthogonal Vectors. Let $k \geq 2$ be an integer, and let $\mathbf{x}_j$ denote the $j$-th component of a vector $\mathbf{x}$.

$k$-Orthogonal Vectors ($k$-OV)

*Instance:* Sets $A_1, \ldots, A_k \subseteq \{0,1\}^d$ with $|A_1| = \cdots = |A_k| = n$.

*Decision:* Are there vectors $\mathbf{x}^{(1)} \in A_1, \mathbf{x}^{(2)} \in A_2, \ldots, \mathbf{x}^{(k)} \in A_k$ such that $\sum_{j=1}^{d} \prod_{i=1}^{k} \mathbf{x}_j^{(i)} = 0$?

The addition and multiplication are those in $\mathbb{N}$, not the field $\mathbb{F}_2$. We also emphasise that this defines a family of problems, one for each $k \geq 2$, as

opposed to a single parameterised problem.

There are different conjectures on the hardness of OV and $k$-OV in the literature, we follow the nomenclature introduced by Gao et al. [38].

**Conjecture 15** ($k$-Orthogonal Vectors conjecture in moderate dimensions)**.** *For any constants $\varepsilon > 0$ and $k \geq 2$, the* $k$-Orthogonal Vectors *problem cannot be solved in time $n^{k-\varepsilon} \cdot \mathsf{poly}(d)$.*

It is well-known that a slight change of the reduction in [57] proves that SETH implies Conjecture 15. Nevertheless, it is consistent with our current knowledge that the $k$-OV conjecture holds while SETH is false. The assumptions $\mathsf{W}[3] \neq \mathsf{FPT}$ and $\mathsf{W}[2] \neq \mathsf{FPT}$ used above also follow from SETH but are possibly much weaker, see the discussion in [18, 23, 43, 44]. Again, no inverse connection nor any relation between the conjectures on the $\mathsf{W}$-hierarchy and on $k$-Orthogonal Vectors are known.

We aim to disprove the existence of an algorithm for Minimal Hitting Set Extension running in time $m^{|X|-\varepsilon} \cdot \mathsf{poly}(n)$ for any constant $\varepsilon > 0$ and constant parameter $|X|$. By Lemma 10, such an algorithm implies Multicoloured Independent Family being solvable in time $(\sum_{i=1}^{k} |\mathcal{S}_i| + |\mathcal{T}|)^{k-\varepsilon} \cdot \mathsf{poly}(|U|)$. We show that the latter assertion contradicts the $k$-Orthogonal Vectors conjecture.

**Lemma 16.** *If there exists an algorithm solving* Multicoloured Independent Family *in time $(\sum_{i=1}^{k} |\mathcal{S}_i| + |\mathcal{T}|)^{k-\varepsilon} \cdot \mathsf{poly}(|U|)$ for any constants $\varepsilon > 0$ and $k \geq 2$, then the $k$-OV conjecture fails.*

*Proof.* Naturally, we reduce from $k$-OV. The construction can be seen in Figure 3. Let $A_1, \ldots, A_k \subseteq \{0,1\}^d$ be sets with $n$ binary vectors each. The constructed instance of MultIndFam has $U = [k] \times [d]$ as its vertex set. Let $\mathbb{1}(\mathbf{x}) = \{j \in [d] \mid \mathbf{x}_j = 1\}$ be the set with characteristic vector $\mathbf{x}$. For each $i \in [k]$, we let the hypergraph $\mathcal{S}_i$ represent the set $A_i$ by adding the edge $\{i\} \times \mathbb{1}(\mathbf{x}) = \{(i,j) \mid j \in \mathbb{1}(\mathbf{x})\}$ for each $\mathbf{x} \in A_i$. Hypergraph $\mathcal{T}$ contains the edge $F_j = [k] \times \{j\}$ for every $j \in [d]$. Intuitively, $F_j$ being completely covered by the union of hyperedges means that the the corresponding vectors all share a 1 in the $j$-th component.

Let $\mathbf{x}^{(1)} \in A_1, \ldots, \mathbf{x}^{(k)} \in A_k$ be a selection of vectors. For any $i \in [k]$ and $j \in [d]$, we have $\mathbf{x}^{(i)}_j = 0$ if and only if $(i,j)$ is *not* contained in the edge $\{i\} \times \mathbb{1}(\mathbf{x}^{(i)}) \in \mathcal{S}_i$. Moreover, no edge of any other $\mathcal{S}_\ell$, $\ell \neq i$, can contain

$$
\begin{aligned}
A_1 = \{&111110,\\
&001111,\\
&011011,\\
&010101\}
\end{aligned}
\qquad
\begin{aligned}
\mathcal{S}_1 = \{&\{(1,1),(1,2),(1,3),(1,4),(1,5)\},\\
&\{(1,3),(1,4),(1,5),(1,6)\},\\
&\{(1,2),(1,3),(1,5),(1,6)\},\\
&\{(1,2),(1,4),(1,6)\}\}
\end{aligned}
$$

$$
\begin{aligned}
A_2 = \{&010111,\\
&101110,\\
&011101,\\
&111011\}
\end{aligned}
\qquad
\begin{aligned}
\mathcal{S}_2 = \{&\{(2,2),(2,4),(2,5),(2,6)\},\\
&\{(2,1),(2,3),(2,4),(2,5)\},\\
&\{(2,2),(2,3),(2,4),(2,6)\},\\
&\{(2,1),(2,2),(2,3),(2,5),(2,6)\}\}
\end{aligned}
$$

$$
\begin{aligned}
A_3 = \{&011011,\\
&110110,\\
&011100,\\
&101111\}
\end{aligned}
\qquad
\begin{aligned}
\mathcal{S}_3 = \{&\{(3,2),(3,3),(3,5),(3,6)\},\\
&\{(3,1),(3,2),(3,4),(3,5)\},\\
&\{(3,2),(3,3),(3,4)\},\\
&\{(3,1),(3,3),(3,4),(3,5),(3,6)\}\}
\end{aligned}
$$

$$
\begin{aligned}
\mathcal{T} = \{&\{(1,1),(2,1),(3,1)\},\{(1,2),(2,2),(3,2)\},\\
&\{(1,3),(2,3),(3,3)\},\{(1,4),(2,4),(3,4)\},\\
&\{(1,5),(2,5),(3,5)\},\{(1,6),(2,6),(3,6)\}\}
\end{aligned}
$$

Figure 3: Illustration of Lemma 16 for $k = 3$. On the left side is an 3-Orthogonal Vectors instance with $n = 4$ vectors in $d = 6$ dimensions. On the right is the resulting instance of Multicoloured Independent Family. The three vectors $010101 \in A_1$, $101110 \in A_2$, and $011011 \in A_3$ together are orthogonal, the union of the corresponding edges from $\mathcal{S}_1$, $\mathcal{S}_2$, and $\mathcal{S}_3$ does not contain any edge of $\mathcal{T}$.

$(i, j)$. Therefore, we have $F_j \nsubseteq \bigcup_{i=1}^{k}(\{i\} \times \mathbb{1}(\mathbf{x}^{(i)}))$ iff $\prod_{i=1}^{k} \mathbf{x}_j^{(i)} = 0$. Finally, this is the case for all $F_j \in \mathcal{T}$ iff the vectors $\mathbf{x}^{(1)}, \ldots, \mathbf{x}^{(k)}$ are orthogonal.

Recall that $k \geq 2$ is a constant. There are $\sum_{i=1}^{k} |\mathcal{S}_i| + |\mathcal{T}| = kn + d$ edges on $|U| = kd$ vertices and the output instance can be computed in time $\mathrm{O}(knd + kd) = \mathrm{O}(nd)$. Therefore, the assumed algorithm for Multicoloured Independent Family running in time $(\sum_{i=1}^{k} |\mathcal{S}_i| + |\mathcal{T}|)^{k-\varepsilon} \cdot \mathsf{poly}(|U|)$ would solve the $k$-Orthogonal Vectors instance in time

$$\mathrm{O}((n+d)^{k-\varepsilon}) \cdot \mathsf{poly}(d) = \mathrm{O}(n^{k-\varepsilon} + d^{k-\varepsilon}) \cdot \mathsf{poly}(d) = n^{k-\varepsilon} \cdot \mathsf{poly}(d). \qquad \square$$

### *4.3. The Nondeterministic Strong Exponential Time Hypothesis*

Any algorithm solving the Minimal Hitting Set Extension problem in time $m^{|X|-\varepsilon} \cdot \mathsf{poly}(n)$ for arbitrary constants $|X|$ and $\varepsilon > 0$ violates SETH. In Section 5, we present an $\mathrm{O}(m^{|X|+1} n)$-time solution. This raises the question what is the "true" exponent of $m$. Although we believe that our algorithm is optimal with respect to $m$, at least up to subpolynomial factors, we sketch an argument why it might be hard to raise the lower bound of Lemma 16 to,

say, $m^{|X|+1-o(1)} \cdot \mathsf{poly}(n)$ under SETH.

Carmosino et al. [15] identified a fundamental obstacle for proving SETH-hardness. A co-nondeterministic algorithm for some decision problem is one whose computation path may have nondeterministic transitions. On a yes-instance, every path is required to produce the answer TRUE, on a no-instance, there must be at least one path resulting in FALSE. The only known co-nondeterministic algorithm for CNF SAT that improves over brute force is randomized [58]. The *Nondeterministic Strong Exponential Time Hypothesis* (NSETH) conjectures that this behaviour is inherent to the problem in that no non-randomized co-nondeterministic algorithm can break the $2^n$-barrier on formulas with $n$-variables.

**Conjecture 17** (Nondeterministic Strong Exponential Time Hypothesis [15])**.** *For every constant $\varepsilon > 0$, there exists a positive integer $k$ such that no co-nondeterministic algorithm without access to randomness can decide $k$-*CNF SAT *on $n$-variable formulas in time* $\mathrm{O}(2^{(1-\varepsilon)n})$.

NSETH can be seen as a common generalisation of SETH and $\mathsf{NP} \neq \mathsf{coNP}$. The value of the conjecture lies not so much in its plausibility–it is false for randomized algorithms–but the fact that both proving and refuting NSETH has interesting consequences. Finding a fast co-nondeterministic algorithm for satisfiability would immediately yield new circuit lower bounds, see [15]. Proving NSETH would, among other things, resolve the $\mathsf{P}$ vs. $\mathsf{NP}$ problem.

The conjecture also rules out the existence of certain fine-grained reductions. Consider a decision problem $\Pi$ that admits an algorithm $\mathcal{A}$ running in time $T(m, n)$ and also a non-randomized co-nondeterministic algorithm $\mathcal{B}$ running in time $T(m, n)^{1-\varepsilon}$ for some constant $\varepsilon > 0$. If NSETH is true, then no deterministic reduction from CNF SAT to $\Pi$ can prove that algorithm $\mathcal{A}$ is optimal under SETH since the very same reduction would give an improved co-nondeterministic algorithm for CNF SAT using algorithm $\mathcal{B}$.

For the further discussion regarding the hardness of MINIMAL HITTING SET EXTENSION, we use the language of first-order model checking. For an introduction to first-order logic in parameterised complexity, see the textbook Flum and Grohe [36]. The equivalent MULTICOLOURED INDEPENDENT FAMILY problem can be seen as deciding whether the input $(U, \mathcal{S}_1, \ldots, \mathcal{S}_k, \mathcal{T})$

is a model[4] for the formula

$$\varphi \;=\; \exists x_1 \in \mathcal{S}_1 \ldots \exists x_k \in \mathcal{S}_k \; \forall y \in \mathcal{T} \; \exists z \in U \colon z \in y \wedge \bigwedge_{i=1}^{k} z \notin x_i.$$

MultIndFam is a *graph problem* in the sense that the maximum arity of any relation in $(U, \mathcal{S}_1, \ldots, \mathcal{S}_k, \mathcal{T})$ is 2. Formula $\varphi$ has $k$ existential quantifiers, followed by a universal one, and then another existential quantifier. We abbreviate this to $\exists^k\forall\exists$. Since MultIndFam is $\mathsf{W}[3]$-complete, the quantifier structure is a characteristic property of the problem, see [36].

Let $k$ be a positive integer. For a graph problem, let $\ell$ denote the total number of "edges", meaning the tuples in the binary relations. Note that for MultIndFam $\ell$ can be as large as $(\sum_{i=1}^{k} |\mathcal{S}_i| + |\mathcal{T}|) \cdot |U|$. Along the lines sketched above, Carmosino et al. [15, Theorem 4] showed that under NSETH the *only* graph problems with $k+2$ quantifiers that can be proven to be SETH-hard with a time bound $\ell^{k+1-o(1)}$ via deterministic reductions are those with quantifier structure $\exists^{k+1}\forall$ or $\forall^{k+1}\exists$.

Using SETH to disprove the existence of an algorithm for Multicoloured Independent Family running in time $O(\ell^{k+1-\varepsilon})$, that is,

$$O\left(\left(\left(\sum_{i=1}^{k} |\mathcal{S}_i| + |\mathcal{T}|\right) |U|\right)^{k+1-\varepsilon}\right) = \left(\sum_{i=1}^{k} |\mathcal{S}_i| + |\mathcal{T}|\right)^{k+1-\varepsilon} \cdot \mathsf{poly}(|U|),$$

for any $\varepsilon > 0$, would therefore need to introduce randomness in a non-trivial way or provide a breakthrough co-nondeterministic algorithm for CNF SAT.

## 5. An Algorithm for the Extension Problem

To finish the description of our hitting set enumeration algorithm, we need to implement the subroutine for the extension problem. We not only assumed that we can decide for disjoint sets $X$ and $Y$ whether $X$ can be extended to a minimal hitting set avoiding $Y$, we additionally claimed that it is possible to find out whether $X$ is itself a solution at no additional cost.

[4] Strictly speaking, we express the instance $(U, \mathcal{S}_1, \ldots, \mathcal{S}_k, \mathcal{T})$ as a relational structure over the universe $U \cup \mathcal{S}_1 \cup \cdots \cup \mathcal{S}_k \cup \mathcal{T}$ with unary relations $S_1, \ldots, S_k, T$, where $S_i\, x$ is interpreted as $x$ being an edge of $\mathcal{S}_i$, and one binary relation $\in \subseteq U \times (\mathcal{S}_1 \cup \cdots \cup \mathcal{S}_k \cup \mathcal{T})$.

**Algorithm 2:** Algorithm for Minimal Hitting Set Extension.

**Input:** Hypergraph $(V, \mathcal{H})$, $\mathcal{H} \neq \emptyset$, and
disjoint sets $X = \{x_1, \ldots, x_{|X|}\}, Y \subseteq V$.
**Output:** MINIMAL if $X \in \mathrm{Tr}(\mathcal{H})$, TRUE if there is a $T \in \mathrm{Tr}(\mathcal{H})$
with $X \subsetneq T \subseteq V \backslash Y$, and FALSE otherwise.

1 **if** $X = \emptyset$ **then**
2  **if** $V \backslash Y$ *is a hitting set* **then return** TRUE;
3  **else return** FALSE;
4 initialise set system $\mathcal{T} = \emptyset$;
5 **foreach** $x \in X$ **do** initialise set system $\mathcal{S}_x = \emptyset$;
6 **foreach** $E \in \mathcal{H}$ **do**
7  **if** $E \cap X = \{x\}$ **then** add $E \backslash Y$ to $\mathcal{S}_x$;
8  **if** $E \cap X = \emptyset$ **then** add $E \backslash Y$ to $\mathcal{T}$;
9 **if** $\exists x \in X \colon \mathcal{S}_x = \emptyset$ **then return** FALSE;
10 **if** $\mathcal{T} = \emptyset$ **then return** MINIMAL;
11 **foreach** $(E_{x_1}, \ldots, E_{x_{|X|}}) \in \mathcal{S}_{x_1} \times \cdots \times \mathcal{S}_{x_{|X|}}$ **do**
12  $W \leftarrow \bigcup_{i=1}^{|X|} E_{x_i}$;
13  **if** $\forall T \in \mathcal{T} \colon T \not\subseteq W$ **then return** TRUE;
14 **return** FALSE;

Despite the hardness results, the investigation in Section 4 also revealed some structure of the Minimal Hitting Set Extension problem that can be exploited algorithmically. Justified by Lemma 10, we approach it via Multicoloured Independent Family. Let $\mathcal{H}$ be the input hypergraph. If $\mathcal{H} = \emptyset$ does not contain a single edge, $X$ is a minimal transversal if and only if $X = \emptyset$ is empty as well. In the remainder we assume that $\mathcal{H}$ is non-empty and solve the extension problem with Algorithm 2. To handle the set $Y$ of excluded vertices, the algorithm computes the truncated hypergraph $\{E \backslash Y\}_{E \in \mathcal{H}}$ and then reduces it to an instance of MultIndFam. In fact, both steps can be computed in one pass (lines 4–8). Lemma 16 suggests that we cannot improve much over brute force when solving the resulting instance, at least not in the worst case. There are, however, several sanity checks possible that may avoid unnecessary computations in practice. The

first check is the special case of an empty set $X = \emptyset$. It is extendable without using $Y$ if and only if $V \backslash Y$ is a hitting set, that is, iff $Y$ does not contain an edge. The other two checks (in lines 9 & 10) assess whether the instance at hand can be decided immediately. If the checks are inconclusive, the instance is indeed solved by brute force (lines 11–14). Note that the existence of a minimal extension is decided without explicitly computing one. As shown in Section 3, this is enough for the enumeration.

Recall that $n = |V|$ denotes the number of vertices and $m = |\mathcal{H}|$ the number of edges of the hypergraph. We now show that the running time of Algorithm 2 matches the OV-lower bound of Lemma 16 up to an $\mathrm{O}(m)$-factor.

**Lemma 18.** *Let $(V, \mathcal{H})$ be a non-empty hypergraph and $X, Y \subseteq V$ disjoint sets of vertices. Algorithm 2 returns* MINIMAL *if $X \in \mathrm{Tr}(\mathcal{H})$ is a minimal hitting set,* TRUE *if there is a $T \in \mathrm{Tr}(\mathcal{H})$ with $X \subsetneq T \subseteq V \backslash Y$, and* FALSE *otherwise. The algorithms runs in $\mathrm{O}((\frac{m}{|X|})^{|X|} \cdot mn)$ time and $\mathrm{O}(mn)$ space.*

*Proof.* The first part up to line 10 of the algorithm computes the reduction from MinHSExt to MultIndFam (Lemma 10) for the truncated hypergraph $(V \backslash Y, \{E \backslash Y\}_{E \in \mathcal{H}})$. The sanity checks in lines 1, 9, and10 filter out trivial instances. The foreach-loop starting in line 11 is then brute-forcing the result of the reduction, checking all tuples in the Cartesian product $\prod_{x \in X} \mathcal{S}_x$.

Since $\mathcal{H}$ is non-empty, the empty set $X = \emptyset$ cannot be a hitting set of $\mathcal{H}$. For some $X \neq \emptyset$ to be a hitting set, the corresponding hypergraph $\mathcal{T}$ must be empty, as verified in line 10. Observe that this reduces Proposition 9 to Proposition 5. Therefore, such an $X$ is minimal iff every $x \in X$ has a private edge, which is exactly what is tested in line 9. In other words, Algorithm 2 correctly identifies the minimal transversals $X$ and reports this by returning the value Minimal from line 10.

Regarding the time complexity, we assume that all set operations (membership, product, union, intersection, and difference) are implemented such that they take time proportional to the total number of elements contained in the input and output of the operation. Checking whether $V \backslash Y$ is a hitting set and computing the systems $\mathcal{S}_{x_1}, \ldots, \mathcal{S}_{x_{|X|}}$, and $\mathcal{T}$ can thus be done in time $\mathrm{O}(mn)$. The running time is dominated by the brute-force phase. The cardinality of the Cartesian product is maximum if all systems have the same number of sets and no edge is cast aside. There are thus at most $(m/|X|)^{|X|}$ many tuples. For each of them, the algorithm computes the union $W$ in $\mathrm{O}(|X|n)$ time and checks all forbidden sets in $\mathcal{T}$ in $\mathrm{O}(mn)$. The fact

that every element of $X$ has a candidate private edge implies $|X| \leq m$ and $\mathrm{O}(|X|n + mn) = \mathrm{O}(mn)$.

Regarding the space requirement, note that $\mathcal{S}_x$ and $\mathcal{T}$ are all disjoint subhypergraphs of $\{E \backslash Y\}_{E \in \mathcal{H}}$, using at most as much space as $(V, \mathcal{H})$. □

Finally, we use Lemma 18 to prove a guarantee on the maximum delay between consecutive outputs of Algorithm 1. The bound is stated in terms of the transversal rank $k^* = \mathrm{rank}(\mathrm{Tr}(\mathcal{H}))$. Recall that $k^*$ is *not* known to the algorithm, the input consists only of the hypergraph itself. For bounded transversal rank, we achieve polynomial delay. In particular, Algorithm 1 then solves the transversal hypergraph problem in output-polynomial time.

**Lemma 19.** *Consider Algorithm 1 with Algorithm 2 implementing the subroutine* `extendable`. *On input* $(V, \preccurlyeq, \mathcal{H})$, *it enumerates the edges of* $\mathrm{Tr}(\mathcal{H})$ *in* $\preccurlyeq$*-lexicographical order with delay* $\mathrm{O}(m^{k^*+1} n^2)$, *where* $k^* = \mathrm{rank}(\mathrm{Tr}(\mathcal{H}))$. *The algorithm uses* $\mathrm{O}(mn)$ *space.*

*Proof.* The correctness was treated in Lemmas 8 and 18. We have also shown there that the label of the current node contains all relevant information to govern the tree search. In particular, it encodes the path to the node in the (only implicitly constructed) recursion tree for backtracking. The total space usage is thus dominated by the $\mathrm{O}(mn)$ of Algorithm 2.

We are left to bound the delay. The height of the tree is $|V| = n$. After exiting a leaf, the pre-order traversal expands at most $2n - 1$ inner nodes before arriving at the next leaf. In the worst case, method `extendable` is invoked in each of them, even with the shortcut evaluations. The $\mathrm{O}((\frac{m}{|X|})^{|X|} mn) = \mathrm{O}(m^{|X|+1} n)$ subroutine dominates the time spent in each node.

We prove that during the enumeration any set $X$ appearing as the first argument of `extendable` is of cardinality at most $|X| \leq k^*$. To reach a contradiction, assume a node $(X, Y, R = V \backslash (X \cup Y))$ with $|X| > k^*$ is expanded by Algorithm 1. This cannot be the root as $X$ is non-empty. Thus, prior to entering $(X, Y; R)$, either `extendable(X,Y)` has been called or the shortcut evaluation inferred the outcome TRUE from the previous calls. Set $X$ is neither a minimal solution nor can it be extended to one as its cardinality is larger than the transversal rank. The check returned FALSE and $(X, Y, R)$ is never entered, a contradiction. Therefore, the delay is bounded by $(2n - 1) \cdot \mathrm{O}(m^{k^*+1} n) = \mathrm{O}(m^{k^*+1} n^2)$. □

## 6. Enumerating Unique Column Combinations

We apply our enumeration algorithm to hypergraphs arising in data profiling as a proof of concept. Specifically, we want to solve what is known as the the *discovery problem* of minimal unique column combinations. In data profiling this term is more common than enumeration. Recall that a UCC for a database $\mathfrak{r}$ over schema $R$ is a set $X \subseteq R$ of columns such that the value combinations appearing as subtuples $r[X]$, $r \in \mathfrak{r}$, are duplicate-free.

Eiter and Gottlob [30] showed that the minimal UCCs can be discovered in output-polynomial time if and only if the transversal hypergraph problem has an output-polynomial solution. Their proof used a Turing-style reduction that inherently requires exponential space. Additionally, there is a folklore reduction from the discovery of UCCs to the enumeration of the hitting sets of difference sets, which we sketched in Section 2.4. Intuitively, for any two distinct rows $r, s \in \mathfrak{r}$, a UCC must contain at least one attribute in which $r$ and $s$ disagree; otherwise, the rows are indistinguishable. That reduction is *parsimonious*[5] in that it establishes a one-to-one correspondence between the enumeration problems while using only polynomial time and space. It also preserves set inclusions. For more details on parsimonious reductions between enumeration problems, see the work of Capelli and Strozecki [14]. Bläsius, Friedrich, and Schirneck [9] improved upon the Turing-equivalence in [30] by giving a parsimonious, inclusion-preserving reduction also in the opposite direction, that is, from hitting sets to UCCs. Discovering minimal UCCs is thus exactly as hard as the general transversal hypergraph problem.

This implies a two-phased approach for the discovery of UCCs. First, generate the hypergraph of minimal difference sets. Secondly, list its minimal transversals. The first phase takes time polynomial in the size of the database. The second phase, which has exponential complexity in the worst case, is the focus of this paper. In the following, we thus assume that the Sperner hypergraph of minimal difference sets is given as the input.

### *6.1. Data and Experimental Setup*

We evaluate our enumeration algorithm on a total of 12 databases. Ten of them are publicly available. These are the `abalone`, `echocardiogram`, `hepatitis`, and `horse` datasets from the University of California Irvine

[5]The concept of parsimonious reductions between enumeration problems is inspired by but should not be confused with the homonymous reductions for counting problems [14].

(UCI) Machine Learning Repository;[6] `uniprot` from the Universal Protein Resource;[7] `civil_service`,[8] `ncvoter_allc`[9] and `flight_1k`[10] provided by the respective authorities of the City of New York, the state of North Carolina, and the federal government of the United States; `call_a_bike` of the German railway company Deutsche Bahn,[11] as well as `amalgam1` from the Database Lab of the University of Toronto.[12] They are complemented by two randomly generated datasets `fd_reduced_15` and `fd_reduced_30` using the *dbtesma* data generator.[13] Databases with more than 100k rows are cut by choosing 100k rows uniformly at random.

The algorithms are implemented in C++ and run on a Ubuntu 16.04 machine with two Intel® Xeon® E5-2690 v3 2.60 GHz CPUs and 256 GB RAM We made the code and data available.[14] In some experiments, we collect the run times of intermediate steps, for example the calls to the subroutine (Algorithm 2). To avoid interference with the overall run time measurements, we use separate runs for these. Also, we average over multiple runs to reduce the noise of the measurements. See the corresponding sections for details.

Table 1 gives an overview of the data. It lists the number of columns and rows in the database, the number of vertices and edges of the resulting hypergraph, the transversal rank/maximum cardinality of a minimal UCC, as well as the number of solutions. The table is sorted by the number of minimal hitting sets/UCCs. All plots below use this order.

After computing the minimal difference sets, we removed all vertices that do not appear in any edge as they are irrelevant for the enumeration. Therefore, the number $n$ of vertices can be smaller than the number of columns in the database. The particularly stark difference for `flight_1k` stems from a large portions of the columns being empty. The total number of difference sets of a database with $|r|$ rows is $\binom{|r|}{2}$ in the worst case. However, Table 1 shows that the number $m$ of minimal difference sets tends to be much *smaller*

[6] archive.ics.uci.edu/ml
[7] uniprot.org
[8] opendata.cityofnewyork.us
[9] ncsbe.gov
[10] transtats.bts.gov
[11] data.deutschebahn.com
[12] dblab.cs.toronto.edu/~miller/amalgam
[13] sourceforge.net/projects/dbtesma
[14] hpi.de/friedrich/research/enumdat

| Dataset | Columns | Rows | $n$ | $m$ | $k^*$ | UCCs |
|---|---|---|---|---|---|---|
| call_a_bike | 17 | 100 000 | 13 | 6 | 4 | 23 |
| abalone | 9 | 4177 | 9 | 30 | 6 | 29 |
| echocardiogram | 13 | 132 | 12 | 30 | 5 | 72 |
| civil_service | 20 | 100 000 | 14 | 19 | 7 | 81 |
| horse | 29 | 300 | 25 | 39 | 11 | 253 |
| uniprot | 40 | 19 999 | 37 | 28 | 10 | 310 |
| hepatitis | 20 | 155 | 20 | 54 | 9 | 348 |
| fd_reduced_15 | 15 | 100 000 | 15 | 75 | 3 | 416 |
| amalgam1 | 87 | 50 | 87 | 70 | 4 | 2737 |
| fd_reduced_30 | 30 | 100 000 | 30 | 224 | 3 | 3436 |
| flight_1k | 109 | 1000 | 53 | 161 | 8 | 26 652 |
| ncvoter_allc | 94 | 100 000 | 82 | 448 | 15 | 200 907 |

Table 1: The databases used in the evaluation, ordered by the number of minimal UCCs. Columns and Rows denote the respective dimension of the database, $n$ and $m$ refer to the resulting hypergraph of minimal difference sets, $k^*$ is the transversal rank, that is, the size of the largest minimal UCC.

than $r$, let alone quadratic. Put it the other way around, only very few pairs of rows actually contribute to the UCCs and the hypergraph perspective thus provides a very compact representation of the discovery problem. As was observed before by other researchers in data profiling, the maximum cardinality $k^*$ of the minimal UCCs is small in practice. In particular, there does not appear to be any relationship between $k^*$ and the input size.

### *6.2. Run Time, Delay, and Memory*

Our enumeration method (Algorithm 1) branches on the vertices in a certain global order. Although the order does not matter for our asymptotic bounds, it does affect the shape of the explored decision tree, which in turn impacts the practical run time. Even on the theoretical side, it has been shown that there exist orders that render already finding the (lexicographically) first solution an NP-hard search problem [29].

To support the enumeration, we heuristically sort the vertices descendingly by the number of distinct values that appear in the corresponding column of the original database. The intuition is that columns with many

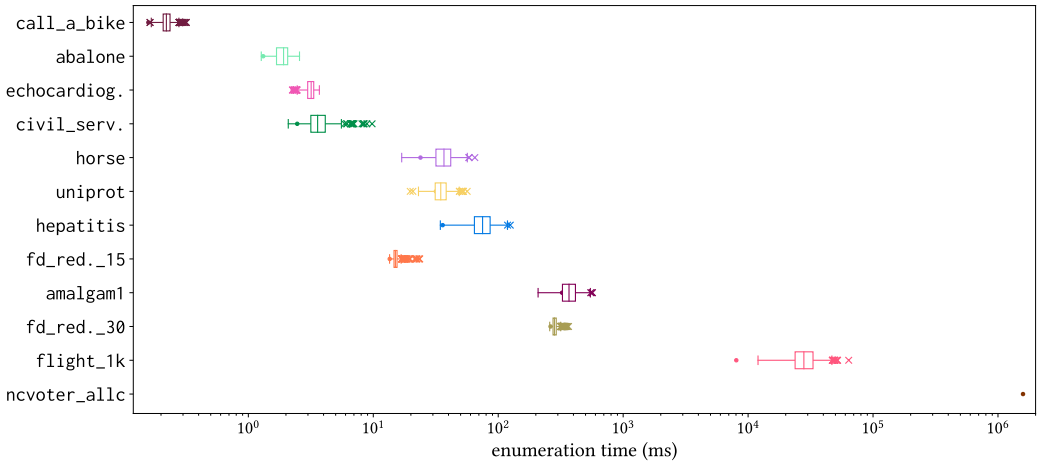



Figure 4: Overall run times of the enumeration algorithm. The dot marks the time using the heuristic branching order. For all datasets except `ncvoter_allc`, the box plot shows the times for 1000 random orders. Their median is indicated by a vertical line, the boxes range from the first to third quartile, and the whiskers chart the 1.5 interquartile range above and below those quartiles. Outliers outside of this range are marked by crosses. Each data point is the average over 10 runs.

values have a higher discriminative power over the pairs of rows and thus are more likely to appear in many minimal UCCs. Including an expressive vertex makes many other vertices obsolete, which should lead to early pruning of the tree. Conversely, excluding such vertices (adding them to the set $Y$ in Algorithm 2) makes it likely that only a few hitting sets survive, which also prunes the tree early. Note that reducing the size of the decision tree, and thus the number of subroutine calls, does not automatically reduce the run time. The remaining calls may have a larger average return time. We discuss this in more detail in Section 6.3. As a side note, preliminary experiments showed that sorting the vertices by their hypergraph degree instead (that is, the number of minimal difference sets in which they appear) resulted in similar but slightly worse run times.

Besides using our heuristic order, we also evaluate the algorithms on 1000 random branching orders per dataset. The `ncvoter_allc` instance is an exception as the larger enumeration times do not permit that many orders. We report on `ncvoter_allc` separately. The run times, averaged over 10 measurements for each data point, are shown in Figure 4. Note that the $x$-axis is scaled logarithmically. The boxes show the first to third quartile of

the samples, with the median indicated as a horizontal line. The whiskers represent the smallest data point within 1.5 interquartile range (IQR) of the lower quartile and the highest one within 1.5 IQR of the upper quartile. We count everything beyond that as outliers.

The median run times generally scale with the number of solutions, which is to be expected. They range from 0.25 ms for the 23 minimal UCCs of `call_a_bike` to roughly 27 min for the more than 200k solutions of `ncvoter_allc`. The only exceptions from this trend, that have shorter enumeration times albeit more solutions, are the artificially generated instances `fd_reduced_15` and `fd_reduced_30`. For most of the instances, the branching order had only little impact and the enumeration times are concentrated around the median. Our heuristic outperformed the median random order on all instances, indicating that it is a solid choice in practice. On the `flight_1k` dataset, the heuristic even resulted in a better run time than any of the random orders. For `ncvoter_allc`, however, the influence of the branching order was significantly larger. Using the heuristic, the enumeration completed in less than half an hour. For comparison, on four out of the eleven random orders we tested, the process only finished after 59.7 h, 105.3 h, 113.7 h, and 167.7 h, respectively. The other seven runs exceeded the time limit of 168 h (one week).

Lemma 19 gives a worst-case guarantee on the delay that depends on the maximum size $k^*$ of a minimal UCC. The box plot in Figure 5 shows the empirical delays when using the heuristic branching order. Again, the time-axis is logarithmic. Recall that the output order of the solutions is entirely determined by the branching order of the vertices. Each data point in Figure 5 corresponds to one output, it was obtained by averaging the delay prior to the same solution over 100 runs. The plot shows that there is a high variance in the delays for the different solutions of an instance. The extreme case is `ncvoter_allc` where the delays range from the order of $1 \times 10^{-1}$ ms to over $1 \times 10^{3}$ ms. Nevertheless, the maximum delay was always less than 2 s, which is reasonably low. The `ncvoter_allc` instance also has the largest solutions with $k^* = 15$. However, the next smaller datasets in that category, `horse` and `uniprot` with transversal ranks of 11 and 10, respectively, have a much lower delay. In general, we cannot confirm a significant correlation between $k^*$ and the empirical delays. In the following section, we investigate the delays more closely by looking at the run time distribution of the calls to the subroutine.

In Lemma 19, we also prove a bound on the space requirement, which is

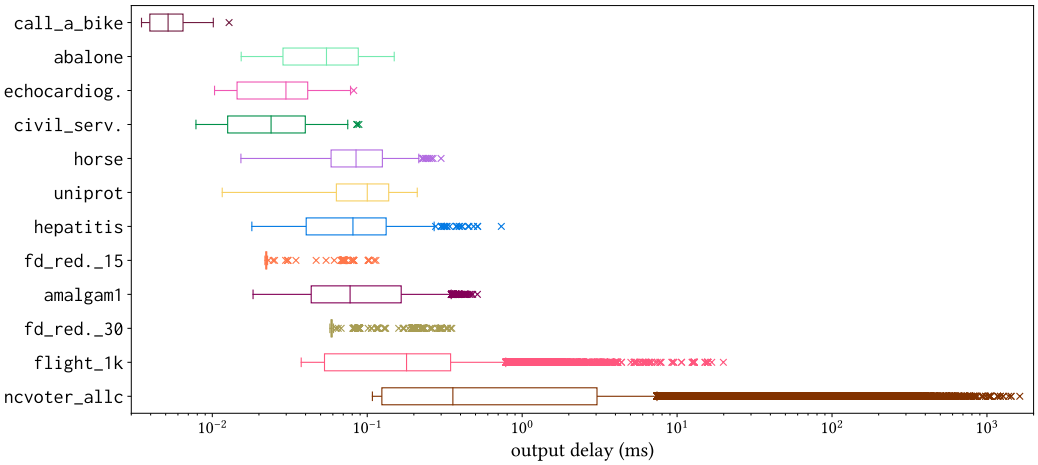


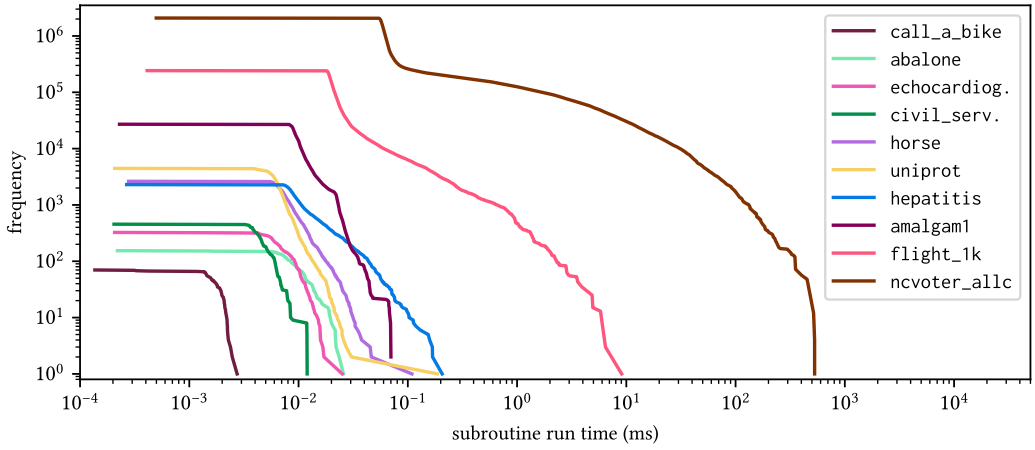

Figure 5: Delays between consecutive outputs of minimal unique column combinations using the heuristic branching order. The box plots show the same quartiles as in Figure 4. Each data point is the average over 100 runs.

independent of the number of solutions. We measured the memory consumption during the enumeration as an average over 5 runs, except for `nc_voter_allc` where we did only a single run. All datasets used between 4.52 MB and 4.68 MB RAM. For comparison, just loading the program without an input takes 4.41 MB. The memory overhead is marginal and independent of the number of solution. In our experiments, it even seemed to be insensitive to the given instance.

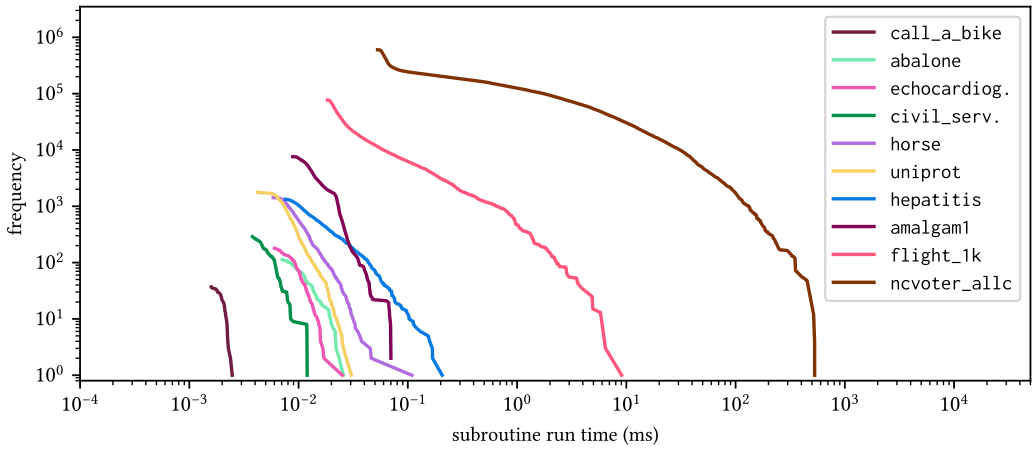
### *6.3. Subroutine Calls*

The only potential reasons for super-polynomial delays are the calls to Algorithm 2. It is interesting to examine how many calls we need during the enumeration and how long they actually take in practice. For our heuristic branching order, we measured the run times of each individual call, averaged over 100 runs to reduce the noise. Figure 6 shows the *complementary cumulative frequencies* (CCF) of the run times in a log-log plot. That means, for each time $t$ on the $x$-axis, the plot shows on the $y$-axis the number of calls with run time at least $t$. We exclude the artificial instances `fd_reduced_15` and `fd_reduced_30` for now, they are reported separately.

First, we examine the impact of the total number of calls on the run time. The legend of Figure 6 is ordered by increasing number of solutions, the same as in the previous plots. For the real-world databases, this is also the same




(a)




(b)

Figure 6: The complementary cumulative frequencies of the run times of the subroutine calls on the real-world databases using the heuristic branching order. Plot (a) shows all calls, (b) only those entering the brute-force loop in line 11 of Algorithm 2. Each data point is the average of the same call over 100 runs.

as ordering them by increasing *enumeration time*. For comparison, the total *number* of subroutine calls is marked by the $y$-value of the left-most endpoint of each curve. The two orders are almost the same. An interesting exception is the `hepatitis` dataset. It has fewer calls than `horse` and `uniprot`, but these calls take more time on average, leading to a higher overall run time. Instance `amalgam1` needs even more calls, which then outweighs the smaller average. Similarly, the calls for `horse` take more time than those for `uniprot`, but the higher number in the latter case causes a longer run time. In preliminary experiments, we observed these effects also when comparing different branching orders for the same dataset. Aiming for a small number of calls is a good strategy, although there are cases where a higher number of easier calls gives a better result.

Next, we discuss the distribution of the calls. The prominent (almost) horizontal lines on the left of Figure 6a stem from the few trivial calls with $X = \emptyset$. Those are one to two orders of magnitude faster than all other calls since they do not need to construct the instance of Independent Family. For the non-trivial cases with $X \neq \emptyset$, the extension algorithm first checks whether the resulting instance can already be decided by the sanity checks in lines 9 and 10 of Algorithm 2. This way, a significant portion of them can be solved in linear time. These calls can be seen in the CCF plots as the steep dip immediately following the horizontal lines. Observe that the $y$-axis is logarithmic, so the proportion of trivial and easy subroutine calls is actually significant. Over all databases, slightly more than half of the calls are solved this way. In fact, for the three instances with the most calls, namely, `amalgam1`, `flight_1k`, and `ncvoter_allc`, no more than 32% of the calls entered the brute-force loop in line 11.

This loop is the only part of the algorithm that may requires super-polynomial running time. Figure 6b shows the CCFs only for the brute-force calls. The differences between Figures 6a and 6b in the lower parts of `call_a_bike` and `uniprot` are artefacts of the separate measurements to create these plots. The run times are heterogeneously distributed with many fast invocations and only a few slow ones. As an example, we investigate the calls of the `flight_1k` instance. The database has 1000 rows over 109 columns of which 39 are empty and 17 more do not participate in any *minimal* difference set. The output of `flight_1k` are 26 652 minimal UCCs. During the enumeration process Algorithm 2 is called 242 449 times, 22 (0.009%) calls are trivial, the vast majority of 165 767 (68.4%) are decided easily by the sanity checks, the remaining 76 660 (31.6%) calls enter the loop.

Of the brute-force calls, 41 353 (53.9%) take only a *single* iteration to find a suitable combination of candidate private edges verifying that the respective input set $X$ is indeed extendable to a minimal solution (line 13). However, there are also two calls that need the maximum of 74 880 iterations, which corresponds to a run time of 16 ms. In those two cases, all possible combinations of potential witnesses had to be tested, only to conclude that the set is not extendable (line 14). It is inherent to the hardness of Minimal Hitting Set Extension that those inputs that are not extendable because all combinations of candidate private edges cover at least one unhit edge incur the highest number of iterations and thus longest subroutine run times, see Lemma 18. Fortunately, those occasions were rare in our experiments. In the case of `flight_1k`, only 622 calls take more than 10 000 iterations, they make up for 0.8% of the brute-force calls and 0.2% of all invocations.

The run time distributions for the other real-world databases are similar to that of `flight_1k`, see Figure 6a. There is always a non-vanishing chance that any given call to the subroutine incurs a high run time, which is hardly avoidable for a worst-case exponential algorithm, but even the slowest calls are reasonably fast in practice. However, the majority of calls is far away from the worst case, leading to a very low run time on average. The heterogeneity of the brute-force calls is also showing in the CCFs (Figure 6b). They roughly resemble a power-law distribution (straight lines in a log-log plot), albeit their tendency towards small run times (concavity of the plots) is stronger than one would expect for a pure power-law.

Another important point of saving related to the subroutine are those calls that are never actually executed due to the shortcut evaluation in line 7 of Algorithm 1. We compared the implementation as presented here with a version in which this optimization is turned off. Still, the latter version outputs a minimal hitting set as soon as it is found in line 5. We used the heuristic branching order again. Over all real-world instances, the ratio of calls of the non-optimized version that are skipped by the shortcuts is between 12.36% for the `abalone` dataset and 66.11% for `ncvoter_allc`, with a median saving of 37.42%. The skipped calls are those for which we can be certain that the given partial solution is indeed extendable, but not yet minimal. Besides the few calls with $X = \emptyset$, all of them would have entered the brute-force phase to find a suitable set of candidate private edges. On the other hand, they do not need to cycle through all possible combinations and thus are not the hardest calls. A given ratio of skipped calls does not directly translate to a certain time saving. Compared to the enumeration time of the

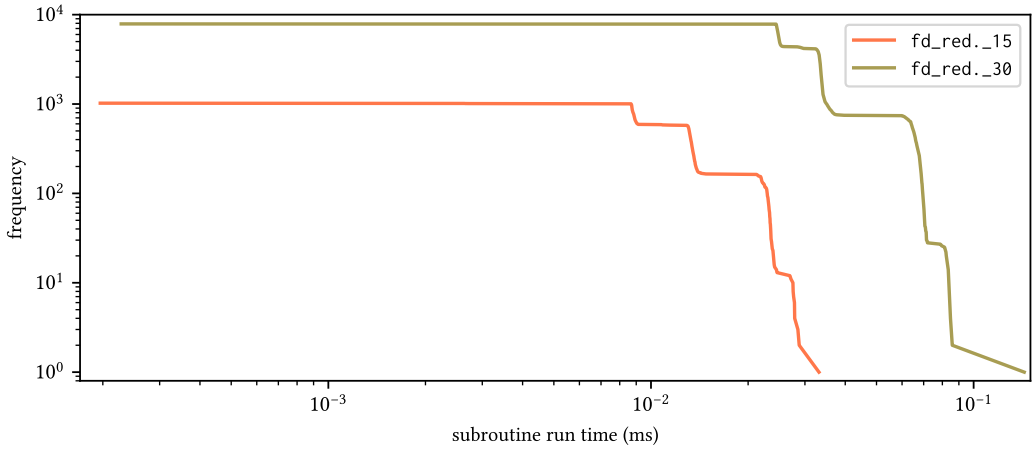



Figure 7: The complementary cumulative frequencies of the run times of the subroutine calls on the artificial using the heuristic branching order.

non-optimized version, the shortcuts gain moderate speedup factors from 1.12 for `abalone` up to 2.26 on the `uniprot` dataset, with a median of 1.43.

Finally, the two artificial instances `fd_reduced_15` and `fd_reduced_30` behave very differently from the real-world databases. Figure 7 shows their CCFs. The staircase shape indicates that there are only five types of calls, with roughly the same run time for all calls of the same type. Also, the shortcut evaluation hardly saves anything on those datasets. Only 1.90% of the calls for `fd_reduced_15` and 4.97% for `fd_reduced_30` are skipped, resulting in a speedup factor of 1.06 on both instances.

## 7. Conclusion

We devised a backtracking algorithm for the transversal hypergraph problem by reducing the enumeration to the NP-complete decision whether a set of vertices can be extended to a minimal solution. Although this may seem counterintuitive, it allowed us to reduce both the space usage of the enumeration and the delay. In particular, we proved that the transversal hypergraph problem can be solved simultaneously with polynomial delay and space on instances whose transversal rank is bounded. We further showed that the extension problem, when parameterised by the size of the set to be extended, is a natural complete problem for the complexity class W[3]. We presented several conditional lower bounds and showed that our extension algorithm

is almost optimal under SETH. With the nondeterministic generalization of SETH, we identified a complexity-theoretic barrier for closing the remaining gap between our algorithmic results and the lower bound.

The features of our enumeration method make it particularly suitable for the profiling of relational databases, an application domain where the solutions are expected to be small. Since the size of the largest solution is the degree of the worst-case time bound, it could have been that the run times are still prohibitively large in practice. To guard against such issues, we evaluated our algorithm by discovering the minimal unique column combinations of several real-world and artificially generated databases. The experiments showed that our method succeeds within a reasonable time frame, even when tasked with computing several hundred thousand solutions.

As the empirical run time depends on the branching order of the vertices, we gave a heuristic that achieves good results in practice by reducing the number of calls to the extension subroutine. We also verified that the main reason for the low overall run times is not only the small number of calls but the fact that the calls are very fast on average. In particular, they regularly avoid the worst case, which was the basis for the large theoretical bound.

The tree search underpinning our algorithm obviates the need of expensive coordination between branches or any post-processing of the solutions. This makes our method easy to implement and memory-efficient. In particular, approaching the discovery of unique column combinations as a hitting set problem resulted in an algorithm that does not need to store previous solutions. This seems to be a major issue even for current state-of-the-art data profiling algorithms such as `DUCC` [42] and `HyUCC` [54]. Papenbrock and Naumann, the authors of `HyUCC`, posed the following challenge [54].

> For future work, we suggest to find novel techniques to deal with the often huge amount of results. Currently, `HyUCC` limits its results if these exceed main memory capacity [...].

We believe that this can be solved by viewing data profiling from a hitting-set perspective. However, there are still some problems that need to be overcome to obtain a ready-to-use algorithm. The enumeration phase is the hard core of the problem, but it does not seem to be the true bottleneck in practice. Instead, the quadratic preprocessing step of preparing the minimal difference sets of the database for our experiments regularly took much longer than actually enumerating all solutions. Here, careful engineering has the

potential of huge speedups on real-world instances. Combining this with the natural advantages of our enumeration algorithm might yield the novel technique we are looking for.

**Acknowledgements.** The authors would like to thank Felix Naumann and Thorsten Papenbrock for the many fruitful discussions about data profiling, and Erik Kohlros for conducting additional experiments.

## References

[1] Ziawasch Abedjan, Lukasz Golab, Felix Naumann, and Thorsten Papenbrock. *Data Profiling*, volume 10 of *Synthesis Lectures on Data Management*. Morgan & Claypool Publishers, San Rafael, CA, USA, 2018. doi:10.2200/S00878ED1V01Y201810DTM052.

[2] Júlio Araújo, Marin Bougeret, Victor A. Campos, and Ignasi Sau. Parameterized Complexity of Computing Maximum Minimal Blocking and Hitting Sets. *CoRR*, abs/2102.03404, 2021. ArXiv preprint. URL: https://arxiv.org/abs/2102.03404, arXiv:2102.03404.

[3] Cristina Bazgan, Ljiljana Brankovic, Katrin Casel, Henning Fernau, Klaus Jansen, Kim-Manuel Klein, Michael Lampis, Mathieu Liedloff, Jérôme Monnot, and Vangelis Th. Paschos. The Many Facets of Upper Domination. *Theoretical Computer Science*, 717:2–25, 2018. doi:10.1016/j.tcs.2017.05.042.

[4] Claude Berge. *Hypergraphs - Combinatorics of Finite Sets*, volume 45 of *North-Holland Mathematical Library*. North-Holland Publishing Company, Amsterdam, Netherlands, 1989.

[5] Claude Berge and Pierre Duchet. A Generalization of Gilmore's Theorem. In *Proceedings of the 2nd Czechoslovak Symposium on Recent Advances in Graph Theory*, pages 49–55, 1975.

[6] Jan C. Bioch and Toshihide Ibaraki. Complexity of Identification and Dualization of Positive Boolean Functions. *Information and Computation*, 123:50–63, 1995. doi:10.1006/inco.1995.1157.

[7] Andreas Björklund, Petteri Kaski, Łukasz Kowalik, and Juho Lauri. Engineering Motif Search for Large Graphs. In *Proceedings of the 17th*

*Meeting on Algorithm Engineering and Experiments (ALENEX)*, pages 104–118, 2015. doi:10.1137/1.9781611973754.10.

[8] Thomas Bläsius, Tobias Friedrich, Julius Lischeid, Kitty Meeks, and Martin Schirneck. Efficiently Enumerating Hitting Sets of Hypergraphs Arising in Data Profiling. In *Proceedings of the 21st Meeting on Algorithm Engineering and Experiments (ALENEX)*, pages 130–143, 2019. doi:10.1137/1.9781611975499.11.

[9] Thomas Bläsius, Tobias Friedrich, and Martin Schirneck. The Parameterized Complexity of Dependency Detection in Relational Databases. In *Proceedings of the 11th International Symposium on Parameterized and Exact Computation (IPEC)*, pages 6:1–6:13, 2016. doi:10.4230/LIPIcs.IPEC.2016.6.

[10] Nicolas Boria, Federico Della Croce, and Vangelis Th. Paschos. On the Max Min Vertex Cover Problem. *Discrete Applied Mathematics*, 196:62–71, 2015. doi:10.1016/j.dam.2014.06.001.

[11] Endre Boros, Khaled M. Elbassioni, Vladimir Gurvich, and Leonid G. Khachiyan. An Efficient Incremental Algorithm for Generating All Maximal Independent Sets in Hypergraphs of Bounded Dimension. *Parallel Processing Letters*, 10:253–266, 2000. doi:10.1142/S0129626400000251.

[12] Endre Boros, Khaled M. Elbassioni, Vladimir Gurvich, Leonid G. Khachiyan, and Kazuhisa Makino. Dual-Bounded Generating Problems: All Minimal Integer Solutions for a Monotone System of Linear Inequalities. *SIAM Journal on Computing*, 31:1624–1643, 2002. doi:10.1137/S0097539701388768.

[13] Endre Boros, Vladimir Gurvich, and Peter L. Hammer. Dual Subimplicants of Positive Boolean Functions. *Optimization Methods and Software*, 10:147–156, 1998. doi:10.1080/10556789808805708.

[14] Florent Capelli and Yann Strozecki. Incremental Delay Enumeration: Space and Time. *Discrete Applied Mathematics*, 268:179–190, 2019. doi:10.1016/j.dam.2018.06.038.

[15] Marco L. Carmosino, Jiawei Gao, Russell Impagliazzo, Ivan Mihajlin, Ramamohan Paturi, and Stefan Schneider. Nondeterministic Extensions

of the Strong Exponential Time Hypothesis and Consequences for Non-reducibility. In *Proceedings of the 7th Conference on Innovations in Theoretical Computer Science (ITCS)*, pages 261–270, 2016. doi:10.1145/2840728.2840746.

[16] Katrin Casel, Henning Fernau, Mehdi Khosravian Ghadikolaei, Jérôme Monnot, and Florian Sikora. Extension of Vertex Cover and Independent Set in Some Classes of Graphs and Generalizations. In *Proceedings of the 11th International Conference on Algorithms and Complexity (CIAC)*, pages 124–136, 2019. doi:10.1007/978-3-030-17402-6_11.

[17] Katrin Casel, Henning Fernau, Mehdi Khosravian Ghadikolaei, Jérôme Monnot, and Florian Sikora. On the Complexity of Solution Extension of Optimization Problems. *Theoretical Computer Science*, 2021. In press. doi:10.1016/j.tcs.2021.10.017.

[18] Jianer Chen, Xiuzhen Huang, Iyad A. Kanj, and Ge Xia. Strong Computational Lower Bounds via Parameterized Complexity. *Journal of Computer and System Sciences*, 72:1346–1367, 2006. doi:10.1016/j.jcss.2006.04.007.

[19] Jianer Chen and Fenghui Zhang. On Product Covering in 3-Tier Supply Chain Models: Natural Complete Problems for $W[3]$ and $W[4]$. *Theoretical Computer Science*, 363:278–288, 2006. doi:10.1016/j.tcs.2006.07.016.

[20] Grant A. Cheston, Gerd H. Fricke, Stephen T. Hedetniemi, and David Pokrass Jacobs. On the Computational Complexity of Upper Fractional Domination. *Discrete Applied Mathematics*, 27:195–207, 1990. doi:10.1016/0166-218X(90)90065-K.

[21] Pierre Colomb and Lhouari Nourine. About Keys of Formal Context and Conformal Hypergraph. In *Proceedings of the 6th International Conference on Formal Concept Analysis (ICFCA)*, pages 140–149, 2008. doi:10.1007/978-3-540-78137-0_10.

[22] Nadia Creignou and Jean-Jacques Hébrard. On Generating All Solutions of Generalized Satisfiability Problems. *RAIRO - Theoretical Informatics and Applications*, 31:499–511, 1997. doi:10.1051/ita/1997310604991.

[23] Marek Cygan, Fedor V. Fomin, Łukasz Kowalik, Daniel Lokshtanov, Dániel Marx, Marcin Pilipczuk, Michał Pilipczuk, and Saket Saurabh. *Parameterized Algorithms*. Springer, Cham, Switzerland, 2015. doi:10.1007/978-3-319-21275-3.

[24] Peter Damaschke. Parameterized Algorithms for Double Hypergraph Dualization with Rank Limitation and Maximum Minimal Vertex Cover. *Discrete Optimization*, 8:18–24, 2011. doi:10.1016/j.disopt.2010.02.006.

[25] János Demetrovics and Vu Duc Thi. Keys, Antikeys and Prime Attributes. *Annales Universitatis Scientiarum Budapestinensis de Rolando Eötvös Nominatae Sectio Computatorica*, 8:35–52, 1987.

[26] Carlos Domingo, Nina Mishra, and Leonard Pitt. Efficient Read-Restricted Monotone CNF/DNF Dualization by Learning with Membership Queries. *Machine Learning*, 37:89–110, 1999. doi:10.1023/A:1007627028578.

[27] Rodney G. Downey and Michael R. Fellows. *Fundamentals of Parameterized Complexity*. Texts in Computer Science. Springer, London, UK, 2013. doi:10.1007/978-1-4471-5559-1.

[28] Louis Dublois, Michael Lampis, and Vangelis Th. Paschos. Upper Dominating Set: Tight Algorithms for Pathwidth and Sub-exponential Approximation. In *Proceedings of the 12th International Conference on Algorithms and Complexity (CIAC)*, pages 202–215, 2021. doi:10.1007/978-3-030-75242-2_14.

[29] Thomas Eiter. Exact Transversal Hypergraphs and Application to Boolean $\mu$-Functions. *Journal of Symbolic Computation*, 17:215–225, 1994. doi:10.1006/jsco.1994.1013.

[30] Thomas Eiter and Georg Gottlob. Identifying the Minimal Transversals of a Hypergraph and Related Problems. *SIAM Journal on Computing*, 24:1278–1304, 1995. doi:10.1137/S0097539793250299.

[31] Thomas Eiter, Georg Gottlob, and Kazuhisa Makino. New Results on Monotone Dualization and Generating Hypergraph Transversals. *SIAM Journal on Computing*, 32:514–537, 2003. doi:10.1137/S009753970240639X.

[32] Thomas Eiter, Kazuhisa Makino, and Georg Gottlob. Computational Aspects of Monotone Dualization: A Brief Survey. *Discrete Applied Mathematics*, 156:2035–2049, 2008. doi:10.1016/j.dam.2007.04.017.

[33] Khaled M. Elbassioni, Matthias Hagen, and Imran Rauf. Some Fixed-Parameter Tractable Classes of Hypergraph Duality and Related Problems. In *Proceedings of the 3rd International Workshop on Parameterized and Exact Computation (IWPEC)*, pages 91–102, 2008. doi:10.1007/978-3-540-79723-4_10.

[34] Michael R. Fellows, Danny Hermelin, Frances A. Rosamond, and Stéphane Vialette. On the Parameterized Complexity of Multiple-Interval Graph Problems. *Theoretical Computer Science*, 410:53–61, 2009. doi:10.1016/j.tcs.2008.09.065.

[35] Henning Fernau. *Parameterized Algorithmics: A Graph-Theoretic Approach*. 2005. University of Tübingen. Habilitationsschrift. URL: https://informatik.uni-trier.de/~fernau/papers/habil.pdf.

[36] Jörg Flum and Martin Grohe. *Parameterized Complexity Theory*. Texts in Theoretical Computer Science. An EATCS Series. Springer, Berlin & Heidelberg, Germany, 2006. doi:10.1007/3-540-29953-X.

[37] Michael L. Fredman and Leonid G. Khachiyan. On the Complexity of Dualization of Monotone Disjunctive Normal Forms. *Journal of Algorithms*, 21:618–628, 1996. doi:10.1006/jagm.1996.0062.

[38] Jiawei Gao, Russell Impagliazzo, Antonina Kolokolova, and R. Ryan Williams. Completeness for First-Order Properties on Sparse Structures With Algorithmic Applications. *ACM Transactions on Algorithms*, 15:23:1–23:35, 2018. doi:10.1145/3196275.

[39] Hector Garcia-Molina and Daniel Barbara. How to Assign Votes in a Distributed System. *Journal of the ACM*, 32:841–860, 1985. doi:10.1145/4221.4223.

[40] Goran Gogic, Christos H. Papadimitriou, and Martha Sideri. Incremental Recompilation of Knowledge. *Journal of Artificial Intelligence Research*, 8:23–37, 1998. doi:10.1613/jair.380.

[41] Miika Hannula, Bor-Kuan Song, and Sebastian Link. An Algorithm for the Discovery of Independence from Data. *CoRR*, abs/2101.02502, 2021. ArXiv preprint. arXiv:2101.02502.

[42] Arvid Heise, Jorge-Arnulfo Quiané-Ruiz, Ziawasch Abedjan, Anja Jentzsch, and Felix Naumann. Scalable Discovery of Unique Column Combinations. *Proceedings of the VLDB Endowment*, 7:301–312, 2013. doi:10.14778/2732240.2732248.

[43] Russell Impagliazzo and Ramamohan Paturi. On the Complexity of $k$-SAT. *Journal of Computer and System Sciences*, 62:367–375, 2001. doi:10.1006/jcss.2000.1727.

[44] Russell Impagliazzo, Ramamohan Paturi, and Francis Zane. Which Problems Have Strongly Exponential Complexity? *Journal of Computer and System Sciences*, 63:512–530, 2001. doi:10.1006/jcss.2001.1774.

[45] David S. Johnson, Christos H. Papadimitriou, and Mihalis Yannakakis. On Generating All Maximal Independent Sets. *Information Processing Letters*, 27:119–123, 1988. doi:10.1016/0020-0190(88)90065-8.

[46] Leonid G. Khachiyan, Endre Boros, Khaled M. Elbassioni, and Vladimir Gurvich. A Global Parallel Algorithm for the Hypergraph Transversal Problem. *Information Processing Letters*, 101:148–155, 2007. doi:10.1016/j.ipl.2006.09.006.

[47] Jan Kossmann, Thorsten Papenbrock, and Felix Naumann. Data Dependencies for Query Optimization: A Survey. *The VLDB Journal*, 30, 2021. doi:10.1007/s00778-021-00676-3.

[48] Sebastian Kruse, Thorsten Papenbrock, Hazar Harmouch, and Felix Naumann. Data Anamnesis: Admitting Raw Data into an Organization. *IEEE Data Engineering Bulletin*, 39:8–20, 2016. URL: http://sites.computer.org/debull/A16june/p8.pdf.

[49] Eugene L. Lawler. A Procedure for Computing the K Best Solutions to Discrete Optimization Problems and Its Application to the Shortest Path Problem. *Management Science*, 18:401–405, 1972. URL: https://www.jstor.org/stable/2629357.

[50] Heikki Mannila and Kari-Jouko Räihä. Dependency Inference. In *Proceedings of the 13th International Conference on Very Large Data Bases (VLDB)*, pages 155–158, 1987. URL: https://www.vldb.org/conf/1987/P155.PDF.

[51] Arnaud Mary and Yann Strozecki. Efficient Enumeration of Solutions Produced by Closure Operations. *Discrete Mathematics & Theoretical Computer Science*, 21, 2019. doi:10.23638/DMTCS-21-3-22.

[52] Rolf Niedermeier. *Invitation to Fixed-Parameter Algorithms*. Oxford Lecture Series in Mathematics and Its Applications. Oxford University Press, Oxford, UK, 2006. doi:10.1093/acprof:oso/9780198566076.001.0001.

[53] Thorsten Papenbrock, Jens Ehrlich, Jannik Marten, Tommy Neubert, Jan-Peer Rudolph, Martin Schönberg, Jakob Zwiener, and Felix Naumann. Functional Dependency Discovery: An Experimental Evaluation of Seven Algorithms. *Proceedings of the VLDB Endowment*, 8:1082–1093, 2015. doi:10.14778/2794367.2794377.

[54] Thorsten Papenbrock and Felix Naumann. A Hybrid Approach for Efficient Unique Column Combination Discovery. In *Proceedings of the 17th Fachtagung Datenbanksysteme in Business, Technologie und Web Technik (BTW)*, pages 195–204, 2017. URL: https://dl.gi.de/20.500.12116/628.

[55] Ronald C. Read and Robert E. Tarjan. Bounds on Backtrack Algorithms for Listing Cycles, Paths, and Spanning Trees. *Networks*, 5:237–252, 1975. doi:10.1002/net.1975.5.3.237.

[56] Raymond Reiter. A Theory of Diagnosis from First Principles. *Artificial Intelligence*, 32:57–95, 1987. doi:10.1016/0004-3702(87)90062-2.

[57] R. Ryan Williams. A New Algorithm for Optimal 2-Constraint Satisfaction and Its Implications. *Theoretical Computer Science*, 348:357–365, 2005. doi:10.1016/j.tcs.2005.09.023.

[58] R. Ryan Williams. Strong ETH Breaks With Merlin and Arthur: Short Non-Interactive Proofs of Batch Evaluation. In *Proceedings of the 31st Conference on Computational Complexity (CCC)*, 2016. doi:10.4230/LIPIcs.CCC.2016.2.