Efficiently Enumerating Hitting Sets of Hypergraphs Arising in Data Profiling[*]

Thomas Bläsius[†] Tobias Friedrich[†] Julius Lischeid[†] Kitty Meeks[‡*]
Martin Schirneck[†✉]

Abstract

We devise an enumeration method for inclusion-wise minimal hitting sets in hypergraphs. It has delay $\mathrm{O}(m^{k^*+1} \cdot n^2)$ and uses linear space. Hereby, n is the number of vertices, m the number of hyperedges, and k^* the rank of the transversal hypergraph. In particular, on classes of hypergraphs for which the cardinality k^* of the largest minimal hitting set is bounded, the delay is polynomial. The algorithm solves the extension problem for minimal hitting sets as a subroutine. We show that the extension problem is $W[3]$-complete when parameterised by the cardinality of the set which is to be extended. For the subroutine, we give an algorithm that is optimal under the exponential time hypothesis. Despite these lower bounds, we provide empirical evidence showing that the enumeration outperforms the theoretical worst-case guarantee on hypergraphs arising in the profiling of relational databases, namely, in the detection of unique column combinations.

1 Introduction

A reoccurring computational task in the design and profiling of relational databases is the discovery of hidden dependencies between attributes. This metadata helps to organise the dataset and subsequently enables further cleansing and normalisation [1]. For example, *unique column combinations* are subsets of attributes (columns) whose values completely identify any record (row) in the database. A minimal combination can thus serve as a fingerprint for the data as a whole, making it a primary key. Unfortunately, unique column combinations are equivalent to *hitting sets* (transversals) which renders their discovery both NP-hard and $W[2]$-hard [7, 18, 43]. Moreover, it is usually not enough to decide the existence of a single, isolated occurrence; instead, one is interested in compiling a comprehensive list of all dependencies [41]. One thus has to solve the *transversal hypergraph problem*.

This problem comes in two variants, enumeration and recognition. To *enumerate* the transversal hypergraph means computing, from an input hypergraph, the collection of its inclusion-wise minimal hitting sets. If necessary, the remaining non-minimal solutions can be produced by arbitrarily adding vertices. For the recognition variant, one is given a pair of hypergraphs and the task is to decide whether one comprises exactly the minimal transversals of the other. The two variants are intimately connected. For any class of hypergraphs, there is an output-polynomial algorithm (incremental-polynomial even) for the enumeration variant if and only if the transversal hypergraph can be recognised in polynomial time for this class [5]. It is a long-standing open question whether this decision problem can be solved efficiently for arbitrary inputs. The transversal hypergraph problem also emerges in a plethora of fields beyond relational databases, like artificial intelligence [33], machine learning [20], distributed systems [31], integer linear programming [10], and monotone logic [24].

While there is currently no efficient method for general inputs, it is worth exploring the characteristics of the concrete application at hand in order to find tractable cases and develop new techniques. For databases, there are two prominent traits: small solutions and the need for user feedback. Usually the largest minimal unique column combination is significantly smaller than the total number of attributes. As an example, the `call_a_bike` database (see Section 5 for more details) spans 16 columns and a hundred thousand rows; nevertheless, the largest unique column combination is of size 4, see also [41, 50]. Although one can expect the solutions to be small, there is generally no a priori guarantee on their maximum cardinality. One thus aims for an algorithm that is suitable for all hypergraphs and particularly fast on those with small transversal. A typical use case for the enumeration of unique column combinations is to present them to a human user for inspection. This allows to incorporate domain knowledge that might otherwise be inaccessible. The human-computer interaction sparks new algorithmic constraints. The first output should be available quickly and subsequent ones should follow in regular intervals. An algorithm with bounded delay is thus preferred over a mere output-efficient one.

[*]The fourth author is supported by a Personal Research Fellowship of the Royal Society of Edinburgh, funded by the Scottish Government.

[†]Hasso Plattner Institute, University of Potsdam, Germany.
[‡]School of Computing Science, University of Glasgow, UK.
[✉]Corresponding author (`martin.schirneck@hpi.de`).

In this paper, we devise an algorithm for the enumeration of minimal hitting sets. We give a guarantee on its delay between consecutive outputs. On hypergraphs for which the size of the largest minimal transversal is bounded, the guarantee is polynomial. Moreover, our experiments show that the enumeration is fast on hypergraphs arising in the discovery of unique column combinations in relational databases.

Contribution and Outline. After introducing basic concepts and notation in Section 2, we give the general outline of our enumeration method in Section 3. The algorithm consists of a backtracking search that is heavily pruned by an extension oracle for minimal hitting sets. The underlying extension problem is discussed in detail in Section 4. It is known to be NP-complete in general, but efficiently solvable if the set to be extended is small [8]. To interpolate between these two extreme cases and better understand the computational complexity of the extension problem, we employ techniques commonly summarised as *fixed-parameter tractability* [21]. In particular, we prove that the problem is $W[3]$-complete, when parameterised by the cardinality $|X|$ of the set to be extended. To the best of our knowledge, it is only the third known example of a natural problem with this property. The first one was given by Chen and Zhang in the context of supply chain management [12]; Bläsius, Friedrich, and Schirneck added the discovery of inclusion dependencies in relational data [7]. Beyond the $W[3]$-hardness we also derive a lower bound based on the exponential time hypothesis, and give an algorithm for the extension oracle running in time $\mathrm{O}(|\mathcal{H}|^{|X|+1} \cdot |V|)$, meeting the lower bound. Combined with the backtracking algorithm, this gives a method to enumerate all minimal hitting sets lexicographically with delay $\mathrm{O}(|\mathcal{H}|^{k^*+1} \cdot |V|^2)$, where k^* is the rank of the transversal hypergraph. Beyond these theoretical bounds, we introduce different techniques for improving the run time in practice. The empirical evaluation in Section 5 shows that our algorithm is capable of efficiently enumerating all unique column combinations of real-world databases when given the hypergraph of difference sets as input. We in particular observe that the run times of the oracle calls (the only part of our algorithm with super-polynomial run time) are quite fast in practice. Compared to a simple brute force enumeration, we achieve huge speed-ups, in particular for larger instances. The paper is completed by a discussion of the related work in Section 6 and some concluding remarks in Section 7.

2 Preliminaries

Hypergraphs and Hitting Sets. A *hypergraph* is a finite *vertex set* V together with a system of sets $\mathcal{H} \subseteq \mathcal{P}(V)$, its *(hyper-)edges.* We often identify a hypergraph with its edges $\mathcal{H}$. The symbol $n = |V|$ denotes the number of vertices, and $m = |\mathcal{H}|$ the number of edges. The *rank* of a hypergraph $\mathcal{H}$ is the maximum cardinality of its edges, $r(\mathcal{H}) = \max_{E \in \mathcal{H}} |E|$. $\mathcal{H}$ is called *Sperner hypergraph* if no edge is contained in another.

A *transversal* or *hitting set* for $\mathcal{H}$ is a vertex 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 contain any other transversal. The minimal hitting sets of $\mathcal{H}$ form a Sperner hypergraph on V, called the *transversal hypergraph* $Tr(\mathcal{H})$. We denote the number of minimal transversals by $N_{\min} = |Tr(\mathcal{H})|$. It is well-known that a transversal is minimal if and only if for every $x \in H$, there is an edge $E_x \in \mathcal{H}$ such that $E_x \cap H = \{x\}$ [4, 48]. We call this edge E_x a *witness* for x (also known as *critical hyperedge* [47]).

Relational Databases and UCCs. To describe relational data, we fix a finite set R, the *(relational) schema*; its elements are the *attributes* or *columns. Records* or *rows* are tuples whose entries are indexed by R, we use the symbols r_i and r_j for them. For any subset $X \subseteq R$ of columns, we let $r_i[X]$ denote the subtuple of r_i consisting only of the entries indexed by X. A finite set r of rows is a *relation* or *relational database.*

Given a relation r, a set X of columns is a *unique column combination* (UCC), or simply a *unique*, if for any two distinct tuples $r_i \neq r_j$ in r, we have $r_i[X] \neq r_j[X]$. The combination of values in the entries for X thus fully identifies any tuple of the relation r. Otherwise, set X is *non-unique.* Any superset of a unique is unique and any subset of a non-unique is again non-unique. A UCC is *(inclusion-wise) minimal* if it does not contain another UCC. There is a one-to-one correspondence between (minimal) UCCs and the (minimal) transversals; see the beginning of Section 5 for more details.

Parameterised Complexity and ETH. For an instance I and a parameter $k \in \mathbb{N}^+$, (I, k) is an instance of the corresponding *parameterised problem.* A parameterised problem is *fixed-parameter tractable* (FPT), if any instance can be solved in time $\mathrm{O}(f(k) \cdot p(|I|))$ where p is a polynomial and f is a computable function.

Let Π and Π' be two parameterised problems. A *parameterised reduction* from Π to Π' is a function computable in FPT-time that maps an instance (I, k) of Π to an equivalent instance (I', k') of Π' such that the parameter k' depends on k but not on $|I|$. If there is also a parameterised reduction in the opposite direction, the problems are said to be FPT-*equivalent*. We often use a restricted form of parameterised reductions, namely, a *polynomial reduction that preserves the parameter*, $k' = k$. Such an reduction transfers the parameterized *and* the classical (polynomial) complexity.

The notion of parameterised reduction gives rise to a hierarchy of complexity classes, the *W-hierarchy*. It is defined in terms of Boolean circuits. The *depth* of a circuit is the maximum length of any path from an input to the output node. The *weft* is the maximum number of *large gates* (fan-in larger than 2) on such a path. WEIGHTED CIRCUIT SATISFIABILITY asks whether a Boolean circuit has a satisfying assignment of *weight* k (k inputs set to TRUE) parameterised by k. For every integer $t > 0$, $W[t]$ contains all parameterised problems that admit a parameterised reduction to WEIGHTED CIRCUIT SATISFIABILITY restricted to circuits of constant depth and weft at most t.

Another source of conditional lower bounds is the *exponential time hypothesis* (ETH) [36, 37]. It conjectures that there is no algorithm solving 3-SAT on n variables in time $2^{o(n)}$.

Enumeration Complexity. *Enumeration* is the process of outputting all solutions to a computational problem without repetition. For hitting sets, there exists a class of hypergraphs such that the number $N_{\min}$ of minimal transversals growths exponentially in both $n = |V|$ and $m = |\mathcal{H}|$ [5]. This rules out a polynomial algorithm. Instead, one could ask for an *output-polynomial* algorithm running in time polynomial in both the input and output size. A 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 polynomial in the input size and in i. The strongest form of output-efficiency is that of *polynomial delay*, where the delay between any two consecutive solutions is polynomial in the input size only.

3 Enumeration Using Decision Trees

It is a common pattern in the design of enumeration algorithms to base them on an *extension oracle* [6, 26, 38, 39, 42, 45, 46, 51, 52]. The oracle decides, for a given collection of vertices, whether there is a solution using these vertices (possibly avoiding some other set).

EXTENSION MINHS

Input: $(V, \mathcal{H})$ and sets $X, Y \subseteq V$; $X \cap Y = \emptyset$.

Output: TRUE iff there is a minimal hitting set $H \in Tr(\mathcal{H})$ with $X \subseteq H \subseteq V \setminus Y$.

Parameter: $|X|$.

We note that $|X|$ is a natural parameter for this problem. On the one hand, it has been observed that it can be solved faster for small $|X|$ [8]. On the other hand, $|X|$ is usually small in instances arising in data profiling.

In this section, we treat the oracle as a black box and describe how it can be used to solve the transversal hypergraph problem. In Section 4, we then give an algorithm to actually solve EXTENSION MINHS. The idea to use oracles in enumeration is usually attributed to Lawler [42]. His original technique, however, required exponential space. Thus, modifications are necessary for practical use, cf. [46, 52]. For transversals, this space reduction can achieved via a decision tree pruned by the extension oracle. This is known in the literature as the *backtracking method* [51] or the *flashlight technique* [45].

Given a pair (X, Y) of disjoint sets of vertices, we want to enumerate the minimal hitting sets that comprise all of X but no vertex from Y. If $X \cup Y = V$, this can only be X itself. Otherwise, we recursively compute the solutions given by the pairs $(X \cup \{v\}, Y)$ and $(X, Y \cup \{v\})$, where v is some vertex not previously contained in X or Y. This leads to a binary decision tree. Every node in the tree is labelled with the respective pair (X, Y); every level in the tree corresponds to some new vertex v. The algorithm branches, in every node of a level, on the decision whether to add v to X or to Y, in the latter case excluding it from the future search.

Algorithm 1: Recursive enumeration algorithm. Initial call: `enumerate`$(\emptyset, \emptyset, V)$.

```
Data: Ordered hypergraph $(V, \succcurlyeq, \mathcal{H})$.
Input: Partition $(X, Y, R)$ of the vertex set $V$.
1 Procedure enumerate($X$,$Y$,$R$):
2 if $R = \emptyset$ then output $X$; return;
3 $v \leftarrow \min_{\succcurlyeq} R$;
4 if extendable($X \cup \{v\}$, $Y$) then
    enumerate($X \cup \{v\}$, $Y$, $R \setminus \{v\}$);
5 if extendable($X$, $Y \cup \{v\}$) then
    enumerate($X$, $Y \cup \{v\}$, $R \setminus \{v\}$);
```

Algorithm 1 recursively traverses the decision tree when given access to the hypergraph $(V, \mathcal{H})$. Subroutine `extendable`(X,Y) is the extension oracle. The procedure `enumerate` receives the sets X and Y as well as the remaining vertices $R = V \setminus (X \cup Y)$ as input and checks whether a leaf has been reached. If not, it branches on the assignment of some vertex of R to X or Y. The initial call of the recursion is `enumerate`$(\emptyset, \emptyset, V)$. As a technical detail, we assume the vertex set V to be equipped with some total order $\succcurlyeq$.

Lemma 3.1. *Assuming subroutine* `extendable` *solves the* Extension MinHS *problem, then Algorithm 1 on input* $(V, \succcurlyeq, \mathcal{H})$ *enumerates* $Tr(\mathcal{H})$. *That is, it outputs every minimal hitting set of* $\mathcal{H}$ *exactly once.*

Our enumeration procedure is similar to the backtracking method by Elbassioni, Hagen, and Rauf [26, Figure 1]. The main difference is the search for new solutions. Besides the partial solution X, the nodes in the decision tree of Algorithm 1 also maintain the set Y of already excluded vertices. The (somewhat arbitrarily chosen) vertex v is added to one of the two sets depending on whether X or $X \cup \{v\}$ are extendable. In contrast, the algorithm in [26] only works on the partial solution−there denoted by S−and explicitly computes a new vertex extending S which is potentially expensive. Also, their check whether S is already a minimal transversal is redundant since this information can be obtained as a by-product of a careful implementation of the extension oracle itself (see Lemma 4.9 below).

We note that always choosing the $\succcurlyeq$-smallest vertex results in the leaves being lexicographically ordered[1] from left to right. The pre-order traversal of the decision tree transfers this ordering also to the outputs. This can be utilized in the context of databases to output "interesting" UCCs first. Suppose the attributes are ranked by importance, then the enumeration naturally starts with those UCCs that contain many important attributes. On the other hand, this raises some complexity theoretic issues. Computing the lexicographically smallest minimal hitting set is known to be NP-hard [22, 38]. Therefore, Algorithm 1 having polynomial delay on *all* ordered hypergraphs would imply P = NP. From a practical point of view, this raises the interesting question which impact the order has on the run time; also see Section 5.

4 The Extension Problem

We now discuss the extension problem for minimal hitting sets. We first show that it is $W[3]$-complete, one of the first natural examples being so. Thus, there is probably no FPT-algorithm for Extension MinHS. We prove an even stronger lower bounds under the exponential time hypothesis. Then, we give an algorithm for the extension problem whose worst-case run time meets the lower bound under ETH. Finally, in combination with the decision tree, we give some strategies for improving the run time and derive a bound on the delay of Algorithm 1.

4.1 Hardness of the Extension Problem. Boros, Gurvich, and Hammer showed that Extension MinHS is NP-complete [8]. In the same article, they reduced it to a certain covering problem in hypergraphs. We extend this result by proving that the two problems are in fact equivalent under polynomial and parameterised reductions. We use this to establish the $W[3]$-completeness of Extension MinHS, when parameterised by the cardinality $|X|$ of the set that is to be extended.

We call this cover problem Multicoloured Independent Family. It formalises the following computational task: given k lists of sets−each list representing a colour−as well as an additional collection of forbidden sets, one has to select a set of each colour such that their union does *not* completely cover any forbidden set.

Multicoloured Independent Family

Input: A $(k+1)$-tuple $(\mathcal{S}_1, \ldots, \mathcal{S}_k, \mathcal{T})$ of systems of sets over a common finite universe U.

Output: true iff there are sets S_1 to S_k, $S_i \in \mathcal{S}_i$, such that $\forall T \in \mathcal{T} \colon T \nsubseteq \bigcup_{i=1}^{k} S_i$.

Parameter: k.

It will be convenient to also have a single-coloured variant of the problem featuring only a single list.

Independent Family

Input: Two systems $\mathcal{S}$, $\mathcal{T}$ of subsets of a finite universe U and a positive integer k.

Output: true iff there are sets $S_1, \ldots, S_k \in \mathcal{S}$ such that $\forall T \in \mathcal{T} \colon T \nsubseteq \bigcup_{i=1}^{k} S_i$.

Parameter: k.

The problems generalise the Independent Set problem in hypergraphs, which asks for k vertices such that they do not cover any hyperedge, parameterised by k, cf. [17, 27]. For Independent Family, we instead select *sets* of vertices such that their *union* is independent (for the hypergraph $\mathcal{T}$). Thus, Independent Set is the special case of system $\mathcal{S}$ consisting entirely of singletons.

Note that $\mathcal{S}_1, \ldots, \mathcal{S}_k$, $\mathcal{S}$, and $\mathcal{T}$ each fit the definition of a hypergraph (over the common universe). However, to avoid confusion with the other hypergraph problems considered in this paper, we use the term *set system* in this context. The next two lemmas establish the equivalence of both variants of the covering problem with the extension problem for minimal hitting sets.

Lemma 4.1. *The problems* Extension MinHS *and* Multicoloured Independent Family *are equivalent under polynomial reductions that preserve the parameter.*

Proof. The first part has already been proven in [8], we restate the reduction from Extension MinHS to Multicoloured Independent Family for later use. Due to space restrictions, the correctness is only sketched.

[1] A set $S \subseteq V$ is lexicographically smaller than another subset T if the $\succcurlyeq$-first element in which S and T differ is in S, cf. [38].

W.l.o.g., we assume that the set Y of vertices we have to avoid is empty. Let $(V, \mathcal{H}, X)$ be an instance of Extension MinHS. If $X = \emptyset$ is empty, we output a trivial TRUE-instance of Multicoloured Independent Family iff $\emptyset \notin \mathcal{H}$; otherwise, a FALSE-instance. From now on, we assume $X \neq \emptyset$.

Recall that, if H is a minimal hitting set, then every element $x \in H$ has a witness, i.e., there is an edge $E \in \mathcal{H}$ such that $E \cap H = \{x\}$. Therefore, X can only be extendable if, for each $x \in X$, there is an edge $E \in \mathcal{H}$ with $E \cap X = \{x\}$. If some element $x \in X$ has no witness, we return a trivial FALSE-instance of Multicoloured Independent Family. Otherwise, the set systems $\mathcal{S}_x = \{E \in \mathcal{H} \mid E \cap X = \{x\}\}$ of *potential* witnesses are non-empty for all x.

Next, we characterise the conditions under which a selection of potential witnesses actually implies the existence of a minimal extension of X. To this end, we collect in set system $\mathcal{T}$ those edges of $\mathcal{H}$ that share no element with X. Edges that intersect X in more than one vertex can be cast aside. We claim that X can be extended to a minimal transversal of $\mathcal{H}$ if and only if there is a selection $(E_x)_{x \in X}$ of potential witnesses, $E_x \in \mathcal{S}_x$, such that $T \nsubseteq \bigcup_{x \in X} E_x$ holds for all $T \in \mathcal{T}$. This is the case iff the instance $((\mathcal{S}_x)_{x \in X}, \mathcal{T})$ of Multicoloured Independent Family evaluates to TRUE. Note that the reduction preserves the parameter $k = |X|$.

We now treat the converse reduction from Multicoloured Independent Family to Extension MinHS. First, observe that above transformation remains valid if we redefine the set systems via *punctured* edges instead, i.e., $\mathcal{S}_x = \{E \backslash \{x\} \mid E \cap X = \{x\}\}$. It now suffices to demonstrate that this modified reduction is capable of producing any given instance $J = (\mathcal{S}_1, \ldots, \mathcal{S}_k, \mathcal{T})$ of the Multicoloured Independent Family problem. □

In the next step, we show that the multi- and single-coloured variants of the independent family problem are essentially the same. The proof uses standard techniques regularly employed to reduce between parameterised problems and their multicoloured variants [17, 27].

Lemma 4.2. *The problems* Multicoloured Independent Family *and* Independent Family *are equivalent under polynomial reductions that preserve the parameter.*

To show membership in $W[3]$ for Independent Family, we construct a polynomial parameter-preserving reduction to a circuit of bounded depth and weft 3. Selecting some set in system $\mathcal{S}$ corresponds to setting an input node to TRUE, the covering constraint is modelled via large gates representing the elements of the union, the sets in $\mathcal{T}$ and a final large (negated) OR-gate.

Lemma 4.3. Independent Family *is in* $W[3]$.

For the hardness, we reduce form the $W[3]$-complete problem Weighted Circuit Satisfiability for *anti-monotone 3-normalised* formulas [21], which are conjunctions of DNF-subformulas with only negative literals.

Lemma 4.4. Independent Family *is* $W[3]$*-hard.*

Theorem 4.5. Extension MinHS *and* (Multicoloured) Independent Family *are* $W[3]$*-complete.*

Theorem 4.5 excludes algorithms for the extension problem running in time $f(|X|)\,p(m,n)$ for any computable function f and polynomial p, unless $\mathrm{FPT} = W[3]$. Under the assumption of the exponential time hypothesis, we now show a stronger lower bound implying that the *exponent* of the worst-case runtime of any such algorithm is at least linear in $|X|$. That is, we disprove the existence of an algorithm having a run time of $f(|X|)\,(m+n)^{\mathrm{o}(|X|)}$.

There is an established connection between ETH and the complexity of parameterised intractable problems, cf. [17, Chapter 14]. This connection is anchored at the $W[1]$-complete Independent Set problem [15]. The lower bound on the complexity of Independent Set in terms of both the input size and the parameter k is transferred to other parameterised problems via polynomial parameter-preserving reductions [13–15].

Proposition 4.6. ([13–15, 17]) *Unless ETH fails, there is no algorithm for* Independent Set *running in time* $f(k)\,n^{\mathrm{o}(k)}$*, for any computable function* f*. If there is a polynomial parameter-preserving reduction from* Independent Set *to a parameterised problem* Π*, there is no algorithm solving* $(I, k') \in \Pi$ *in time* $f(k')\,|I|^{\mathrm{o}(k')}$.

Proposition 4.6 also rules out any $f(k){\cdot}(m+n)^{\mathrm{o}(k)}$-time algorithm for Independent Set since the number m of edges in a graph is at most quadratic in n. Clearly, the bound also applies to the generalised Independent Family problem. Composing this with the reductions in Lemmas 4.1 and 4.2 yields the following theorem.

Theorem 4.7. *Unless ETH fails, there is no algorithm solving* Extension MinHS *in time* $f(|X|){\cdot}(m+n)^{\mathrm{o}(|X|)}$ *for any computable function* f*. Furthermore,* Independent Family*, resp.* Multicoloured Independent Family *cannot be solved in time* $f(k){\cdot}(|\mathcal{S}|+|\mathcal{T}|+|U|)^{\mathrm{o}(k)}$*, resp. in time* $f(k){\cdot}(\sum_{i=1}^{k}|\mathcal{S}_i|+|\mathcal{T}|+|U|)^{\mathrm{o}(k)}$.

4.2 Solving the Extension Problem.

We still have to solve the extension problem. Justified by the reductions in [8] and Lemma 4.1, we approach it via Multicoloured Independent Family; see Algorithm 2. First, it treats the special case of an empty set X.

Then, in lines 6 to 10, the resulting instance of Multi-coloured Independent Family is constructed. Two sanity checks in lines 11 and 12 assess whether the instance is trivial. We will see later that in practice many calls of the oracle are already decided here. If the checks are inconclusive, we solve the instance by brute force (lines 13–16). The algorithm decides the existence of a minimal extension without explicitly computing one. As shown in Section 3, this is enough for the enumeration.

Algorithm 2: Alg. for Extension MinHS.

```
   Input: Hypergraph $(V, \mathcal{H})$ and disjoint sets
          $X, Y \subseteq V$, where
          $X = \{x_1, x_2, \dots, x_{|X|}\}$.
1  Output: TRUE iff there is a minimal hitting set
2         $H$ of $\mathcal{H}$, with $X \subseteq H \subseteq V \backslash Y$ .
3  if $X = \emptyset$ then
4      if $V \backslash Y$ is a hitting set then return TRUE;
5      else return FALSE;
6  initialise set system $\mathcal{T} = \emptyset$;
7  foreach $x \in X$ do initialise set system $\mathcal{S}_x = \emptyset$;
8  foreach $E \in \mathcal{H}$ do
9      if $E \cap X = \{x\}$ then add $E \backslash Y$ to $\mathcal{S}_x$;
10     if $E \cap X = \emptyset$ then add $E \backslash Y$ to $\mathcal{T}$;
11 if $\exists x \in X \colon \mathcal{S}_x = \emptyset$ then return FALSE;
12 if $\mathcal{T} = \emptyset$ then return TRUE;
13 foreach $(E_{x_1}, \dots, E_{x_{|X|}}) \in \mathcal{S}_{x_1} \times \cdots \times \mathcal{S}_{x_{|X|}}$
     do
14     $W \leftarrow \bigcup_{i=1}^{|X|} E_{x_i}$;
15     if $\forall T \in \mathcal{T} \colon T \nsubseteq W$ then return TRUE;
16 return FALSE;
```

We now give a bound on the worst-case run time, which matches the lower bound in Theorem 4.7. Recall that $|X|$ denotes the cardinality of the set to be extended.

Lemma 4.8. *Algorithm 2 solves* Extension MinHS *in time* $\mathrm{O}((m/|X|)^{|X|} \cdot mn)$ *and linear space.*

4.3 Improved Enumeration. So far, we are solving the extension problem in a vacuum. We can improve upon this by also taking into consideration the interplay with the decision tree of the enumeration procedure. We are looking for the minimal transversals, but the Extension MinHS problem is indifferent to the distinction between sets that are themselves minimal solutions and those that still need extending vertices. As it turns out, the algorithm can tell the two apart.

Lemma 4.9. *Let $\mathcal{H} \neq \emptyset$ be non-empty and $X, Y \subseteq V$ two disjoint sets of vertices. Then, $X \in Tr(\mathcal{H})$ if and only if Algorithm 2 on input $(V, \mathcal{H}, X, Y)$ returns* true *by breaking in line 12.*

We can trivially check whether $\mathcal{H}$ is empty, i.e., whether the empty set is the only minimal transversal, before invoking the enumeration algorithm. Therefore, we always assume $\mathcal{H} \neq \emptyset$ in what follows. The subroutine, on input (X, Y), breaking in line 12 flags the fact that we have found a minimal solution. If so, we know the outcome of every oracle call in the subtree rooted in the current node: Adding further vertices to X can never lead to a set that is extendable; conversely, adding to Y does not change X and yields true deterministically. The subtree rooted in node $(X, Y, V \backslash (X \cup Y))$ has exactly one leaf that holds a minimal solution, which is the node $(X, V \backslash X, \emptyset)$. We can thus safely shortcut the recursion and immediately output solution X.

Another possible improvement stems from the observation that the two extension checks of the enumeration procedure, line 4 and 5 of Algorithm 1, cannot return false at the same time. We only enter a node once we verified that at least one leaf of its subtree holds a minimal solution. That leaf is a descendent of either the left or the right child. Thus, if the first check fails, we do not need to execute the second one. Note that this does not interfere with the above flagging mechanism. We entered the current node because X is extendable but not yet a minimal solution. The call `extendable(`X, $Y \cup \{v\}$`)` has the same first argument, and thus Algorithm 2 computing that call would also not break in line 12.

Finally, we prove a guarantee on the maximum delay between consecutive outputs of Algorithm 1. The key to this is that between two leafs of the decision tree, we visit only $\mathrm{O}(n)$ inner nodes. Moreover, the size $|X|$ of sets we try to extend is bounded by the *rank of the transversal hypergraph* $r(Tr(\mathcal{H}))$, which we denote with k^*. For bounded k^* we achieve polynomial delay.

Theorem 4.10. *Consider Algorithm 1 combined with the (improved) Algorithm 2 as oracle, running on input $(V, \succcurlyeq, \mathcal{H})$. It enumerates $Tr(\mathcal{H})$ with delay $\mathrm{O}(m^{k^*+1} n^2)$, where $k^* = r(Tr(\mathcal{H}))$. The method uses linear space.*

Note that the rank k^* is *not* known to the algorithms, the input consists only of the hypergraph itself. In fact, it has been shown recently by Bazgan et al. that computing k^* is $W[1]$-hard (parameterised by k^*); also, it cannot be approximated with a factor of $n^{1-\varepsilon}$ for any $\varepsilon > 0$, unless $\mathrm{P} = \mathrm{NP}$ [3]. The upper bound on the delay holds regardless of the knowledge of k^*. In particular, on hypergraphs for which the size of the largest minimal hitting set is constant, Algorithm 1 has polynomial delay.

5 Enumerating Unique Column Combinations

We apply our enumeration algorithm to hypergraphs arising in the profiling of relational databases. Specifically, we want to enumerate all minimal UCCs. There is a folklore polynomial reduction from the detection of minimum UCCs to the hitting set problem, cf. e.g. [18, 43]. Let r be a relation over a schema R. Intuitively, for any two rows $r_i, r_j \in r$, any UCC must contain at least one attribute in which r_i and r_j disagree; otherwise, these rows would be indistinguishable. The set of vertices in which r_i and r_j disagree is called their *difference set*. This reduction is parsimonious in that it translates all UCCs into hitting sets while also preserving the inclusion relationships. Since there is also a parsimonious reduction in the opposite direction, listing minimal UCCs is as hard as the general enumeration problem [7].

This reduction implies a two-phased approach. First, generate the hitting set instance. Second, enumerate the minimal transversals. Clearly, the first phase can be done in time polynomial in the size of the database. The second phase, which is the focus of our paper, has exponential complexity in general. In the following, we thus assume, that the Sperner hypergraph of the minimal difference sets is given as input.

5.1 Data and Experimental Setup. We evaluated our enumeration algorithms on a total of twelve databases. These are the `abalone`, `echocardiogram`, `hepatitis`, and `horse` dataset from the UCI Machine Learning Repository[2], `uniprot` from the Universal Protein Resource[3], `civil_service`[4], `ncvoter_allc`[5] and `flight_1k`[6] provided by the respective authorities of the City of New York, North Carolina, and the federal government of the US, `call_a_bike` of the German railway company Deutsche Bahn[7], as well as `amalgam1` from the Database Lab of the University of Toronto[8]. They are complemented by two randomly generated datasets `fd_reduced_15` and `fd_reduced_30` using the *dbtesma* data generator[9]. Databases with more than 100k rows were cut by choosing 100k rows uniformly at random. After computing the Sperner hypergraph, we removed vertices not appearing in any edge, as these are not relevant for the enumeration of minimal hitting sets. Thus, the number of vertices of the hypergraph can be smaller than the number of columns in the database. Table 1 lists the data together with some basic features. The table is sorted by the number $N_{\min}$ of minimal hitting sets, i.e., the solution size. All plots shown in the following use the same order. In our box plots, the boxes range from the first to the third quartile, with the median indicated as horizontal line. The whiskers represent the lowest datum within 1.5 interquartile range (IQR) of the lower quartile and the highest datum within 1.5 IQR of the upper quartile. Everything beyond that counts as outlier and is indicated as small cross.

dataset	cols.	rows	$\|V\|$	$\|\mathcal{H}\|$	k^*	$N_{\min}$
`call_a_bike`	16	100k	13	6	4	23
`abalone`	9	4177	9	30	6	29
`echocardiogr.`	12	132	12	30	5	72
`civil_serv.`	20	100k	14	19	7	81
`horse`	28	300	25	39	11	253
`uniprot`	40	19 999	37	28	10	310
`hepatitis`	20	155	20	54	9	348
`fd_red._15`	15	100k	15	75	3	416
`amalgam1`	87	50	87	70	4	2737
`fd_red._30`	30	100k	30	224	3	3436
`flight_1k`	70	1000	53	161	8	26 652
`ncvoter_allc`	88	100k	82	448	15	200 907

Table 1: The databases used in the evaluation ordered by the number $N_{\min}$ of minimal UCCs. Columns and rows denote the respective dimension of the database, $|V|$ and $|\mathcal{H}|$ refer to the resulting hypergraph of difference sets, k^* is the cardinality of the largest minimal UCC.

All algorithms were implemented[10] in C++ and run on a Ubuntu 16.04 machine using two Intel Xeon E5-2690 v3 2.60Ghz CPUs and 256GB RAM. For some experiments, we collected run times of intermediate steps, e.g., for the oracle calls in Section 5.3. To not interfere with the overall run time measurements, we used separate runs for these experiments. Moreover, we reduced the noise of our run time measurements by averaging over multiple runs. See the corresponding sections for details.

5.2 Vertex Order and Overall Run Time. Recall from Section 3 that our algorithm branches on the vertices of the hypergraph in a certain order. Although the order does not matter for the theoretical bounds, changing it results in a different decision tree, which impacts the practical run time. As a heuristic, we (descendingly) sort the vertices according to the number of different values that appear in the corresponding column of the database. The logic behind this is that columns with many different values are more *expressive* and thus more likely to appear in many minimal UCCs. Including an expressive vertex makes many other vertices

[2]`archive.ics.uci.edu/ml`
[3]`uniprot.org`
[4]`data.cityofnewyork.us`
[5]`ncsbe.gov`
[6]`transstats.bts.gov`
[7]`data.deutschebahn.com`
[8]`dblab.cs.toronto.edu/~miller/amalgam/`
[9]`sourceforge.net/projects/dbtesma`

[10]The code is available at `hpi.de/friedrich/research/enumdat`.

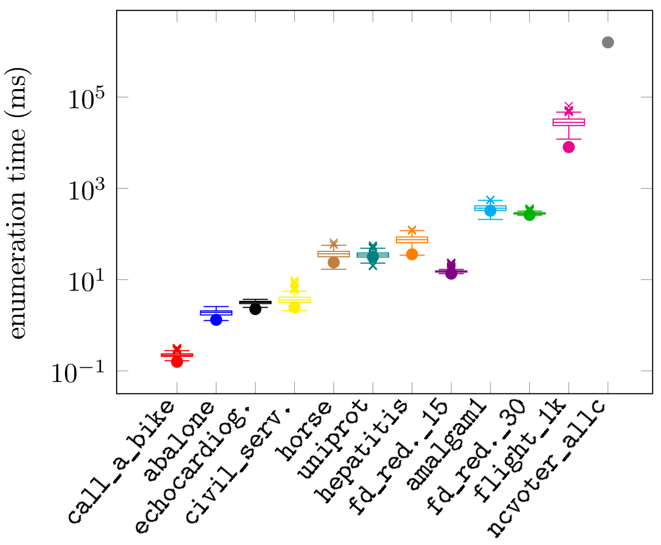

Figure 1: Run times of our enumeration algorithm. The box plots show the run times for 1000 random vertex orders. The filled circles correspond to our heuristic vertex order.

obsolete, which leads to early pruning in the tree. Conversely, excluding expressive vertices makes it likely that no hitting set remains, which also prunes the tree early. Note that reducing the size of the decision tree, and thus the number of oracle calls, does not guarantee a reduced run time, as different oracle calls have different run times. We discuss this aspect in more detail in Section 5.3. Moreover, we note that sorting the vertices by their degree (the number of edges they appear in) resulted in similar but slightly worse run times.

Besides using our heuristic vertex order, we evaluated the algorithms also on 1000 random orders per dataset. The only exception is the `ncvoter_allc` instance, where the higher run time did not permit that many random orders. We report on `ncvoter_allc` separately. The run times, averaged over 10 measurements for each data point, are shown in Figure 1. The instances are sorted by the number of hitting sets. Clearly, the total run time scales with the solution size. The only exceptions seemed to be the two generated instances `fd_reduced_15` and `fd_reduced_30`. For most instances, the order of the vertices had only a minor impact. In any case, our heuristic order, outperforming the median random order for all instances, seemed to be a solid choice. On some instances, the heuristic led to lower run times than any of the random orders.

For `ncvoter_allc`, the influence of the vertex order was much stronger. Using the heuristic, the enumeration completed in 26 min. However, on four out of eleven random orders, the processes 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).

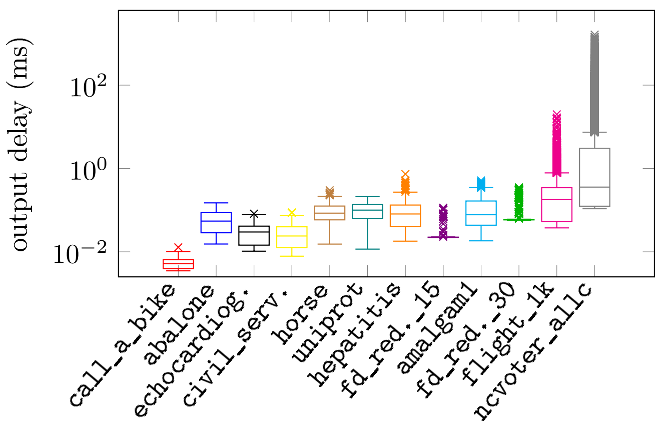

Figure 2: Delays between consecutive outputs of minimal hitting sets for our heuristic vertex order.

Figure 2 shows the delays between consecutive solutions when using the heuristic order. Each data point was obtained by averaging over 100 runs. While there is a high variance in the delays on some instances, e.g., between 10^{-1} ms and 10^3 ms for `ncvoter_allc`, the maximum delay is always less than 2 s, which is reasonably low. In the following section, we investigate the delays more closely by looking at the run time distribution of the oracle calls.

5.3 Oracle Calls. In Section 4.3, we showed that the number of oracle calls between two consecutive outputs is linear in the number of vertices in the hypergraph. Thus, the only part potentially leading to a super-polynomial delay are the oracle calls themselves. It is thus interesting to see how many oracle calls we need and how long these calls actually take in practice.

For our heuristic vertex order, we measured the run times of each individual oracle call, averaged over 100 runs to reduce the noise. Figure 3 shows the *complementary cumulative frequencies* (CCF) of the oracle run times, i.e., for each time t (x-axis), it shows the number of oracle calls with run time at least t (y-axis). We excluded the artificial instances `fd_reduced_15` and `fd_reduced_30`, they are discussed separately. First, note the horizontal lines on the left. They stem from the few oracle calls with $X = \emptyset$. Those are much faster since they do not need to construct the Independent Family instance. Next, we examine the impact of the total number of oracle calls on the run time. The legend of Figure 3 orders the instances by *enumeration time* from left to right. In comparison, the *number* of oracle calls is the y-value of the left endpoint of each curve. The two orders are almost the same. An interesting exception is the `hepatitis` dataset. It has fewer calls

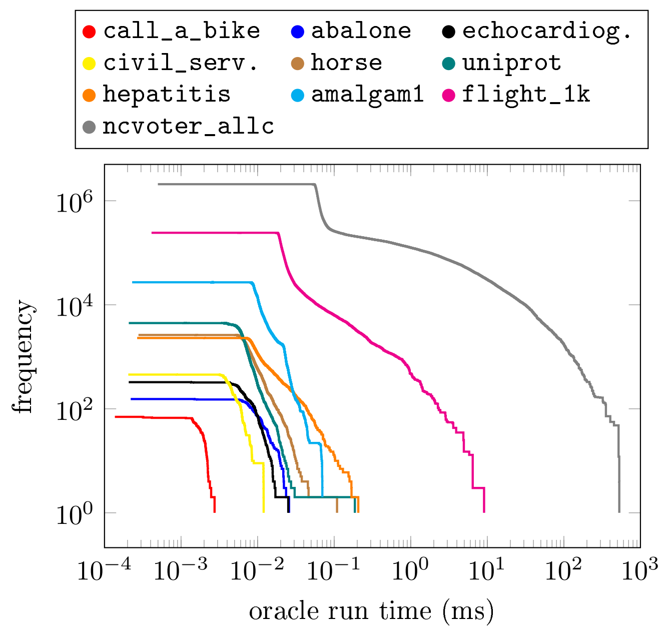

Figure 3: Complementary cumulative frequencies of oracle run times over the enumeration process. Our heuristic was used for the vertex order.

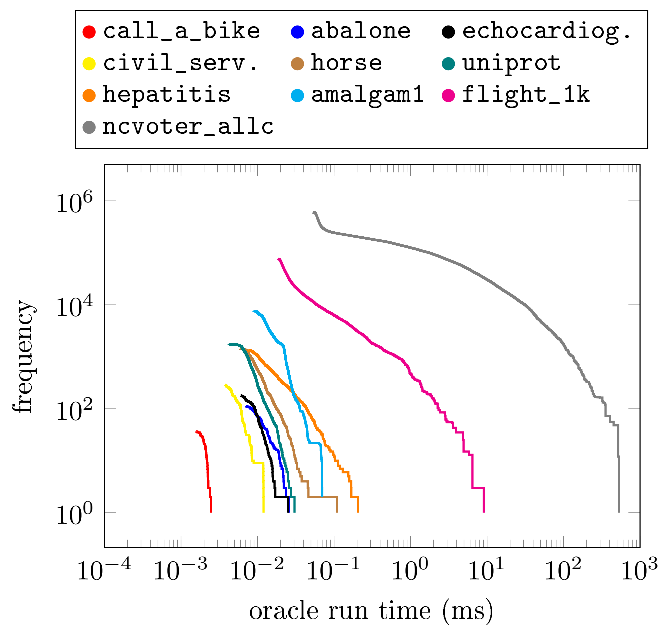

Figure 4: Complementary cumulative frequencies of run times for oracle calls that enter the loop in line 13 of Algorithm 2.

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 oracle calls for **horse** take more time than those for **uniprot**, but the higher number of calls in the latter case causes the longer run time. In preliminary experiments, we observed these effects also when comparing different vertex orders of the same dataset. In general, aiming for the smallest number of oracle calls is a good strategy, although there are cases in which a higher number of easier calls leads to better run times.

Recall that the extension oracle (Algorithm 2) first checks whether the resulting INDEPENDENT FAMILY instance is trivially FALSE or TRUE (lines 11 and 12, respectively). This way, a significant portion calls can be solved in polynomial time. Over all instances, slightly more than half of the oracle calls are solved like that. In fact, for the three instances with the most oracle calls, namely, **amalgam1**, **flight_1k**, and **ncvoter_allc**, no more than 32% of the calls entered the brute-force loop in line 13, which is the only part that actually requires super-polynomial run time. It is thus interesting to see specifically the run times of those calls, their CCF are shown in Figure 4. They are heterogeneously distributed with many fast invocations and only few slow ones. In fact, for most instances, these plots resemble straight lines (in the log-log plot), which indicates that the oracle run times roughly follow a power-law distribution. In conclusion, a few calls with high run times are unavoidable, which is not surprising for an worst-case exponential algorithm. However, even the slowest calls are reasonably fast. Moreover, the majority of oracle calls is far away from the worst case, leading to a very low run time on average.

Figure 5 shows the CCF of the oracle calls of the two artificial instances. The behaviour is very different to the real-world instances. The staircase shape indicates that there are five different types of oracle calls, with roughly the same run times for calls of the same type.

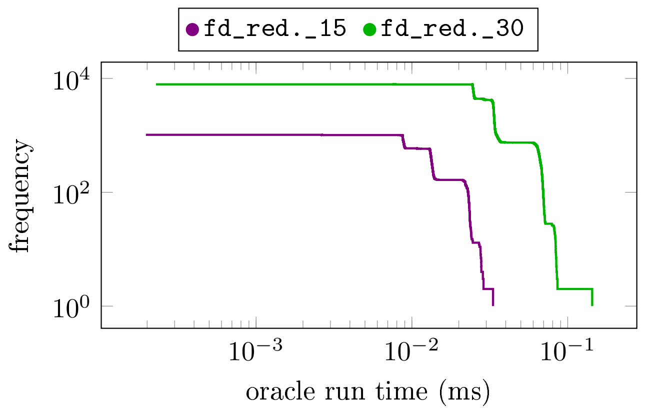

Figure 5: Complementary cumulative frequencies of oracle run times over the enumeration process for the artificial instances. Our heuristic was used for the vertex order.

5.4 Comparison with Brute Force. We used a brute-force search for all minimal UCCs as a baseline for comparison, namely, the `Apriori` algorithm. Originally developed for frequent itemset mining [2], it is well-known that `Apriori` can also be employed to find minimal UCCs and hitting sets in general [1, 32, 43]. For level i, it maintains a collection of candidate sets, all of cardinality i. The candidates of the i+1-st level are generated by combining any two i-candidates that differ in exactly one vertex. The new candidates are then checked whether they are hitting sets of the input hypergraph. If so, they are output and discarded from the candidate collection. To avoid generating unnecessary candidates, i.e., those that are supersets of smaller transversals, they are checked against the collection of found solutions. The bottom-up approach thus ensures that once a hitting set is found, it must be minimal as all of its subsets have been tested before.

After finding all solutions, `Apriori` may take some additional time to terminate as it needs to verify that the search space of future candidates is in fact empty. It is generally agreed upon that this is hard to avoid, finding an efficient stopping criterion here would immediately yield faster algorithms to compute k^*, see [3] and [28] for a more detailed discussion. To make this transparent in our experiments, we measured two run times for the brute-force algorithm: the *enumeration time* is the time it took to find all minimal hitting sets, and the *completion time* is the total run time until termination.

Figure 6 shows the run times of our algorithm in comparison to the enumeration and completion times of the brute-force algorithm. For the smaller instances, both algorithms are fast. For larger instances, the brute-force baseline is clearly outperformed, with the only exceptions being, again, the two randomly generated instances `fd_reduced_15` and `fd_reduced_30`. Beyond the mere run time, the brute-force algorithm also requires a lot of memory if the transversal hypergraph is large, it needs to store all output solution to avoid unnecessary candidates. We thus set a memory limit of 128 GB and a time limit of 12 h. On `flight_1k` and `ncvoter_allc`, `Apriori` found 98.6% and 14.6% of the minimal hitting sets, respectively, before reaching the time limit. For all other instances, brute force enumerated all solutions within the limits. However, for `uniprot` and `amalgam1`, the memory limit was exceeded between the output of the last solution and termination. Still, even if we only take the enumeration time into account, we achieved speed-up factors of 4723 and 14 for these two instances.

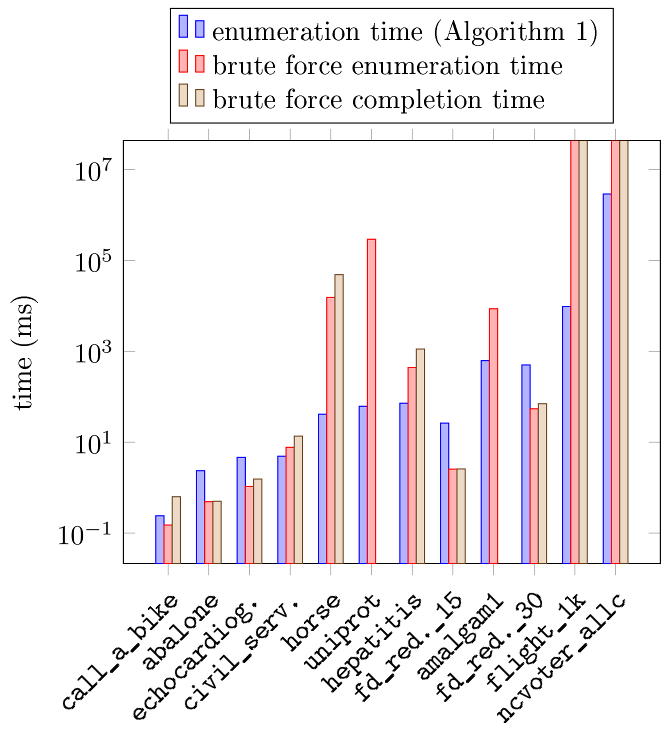

Figure 6: Comparing the run time of our algorithm with the brute-force baseline. The completion time for `uniprot` and `amalgam1` are not reported, as the brute-force algorithm exceeded the memory limit prior to termination. The enumeration and completion time of the brute-force algorithm exceeded the time limit for `flight_1k` and `ncvoter_allc`.

6 Related Work

The topic of enumerating and recognising the transversal hypergraph efficiently is covered in numerous papers over the last three decades, both from a theoretical and an empirical standpoint. For a detailed overview, we direct the reader to the survey by Eiter, Makino, and Gottlob [25] and the textbook by Hagen [34]. It is usually attributed to Demetrovics and Thi [19] as well as Mannila and Räihä [44], independently, to raise the issue of generating all hitting sets. The existence of an output-polynomial algorithm is unknown. However, Fredman and Khachiyan gave a quasi-polynomial algorithm [29]. The exponent of its worst-case run time has only a sub-logarithmic dependence on the combined input and output size. Besides this general-purpose procedure, several tractable special cases have been identified. They exploit structural properties such as bounds on the edge size or degree [9, 20], conformality [40], or different notions of acyclicity [23, 24].

There is an imbalance in the theoretical literature regarding the generation variant, most of the results focus on the properties of the input graph instead of the output. A notable exception is the work of Eiter and Gottlob [23]. They show that the recognition problem is polynomial-time solvable if the rank of the transversal hypergraph is bounded. Via an equivalence result by Bioch and

Ibaraki [5], this implies an incremental-polynomial algorithm for the enumeration of such hypergraphs. However, this algorithm necessarily holds the output hypergraph in memory during the whole computation. We improved upon this result in showing that there exists an enumeration method that has polynomial delay (if the transversal hypergraph has bounded rank), outputs the hitting sets in lexicographical order, and uses space linear in the size of the input. Our method is similar to the backtracking technique proposed by Elbassioni, Hagen, and Rauf [26]. In the theoretical analysis of our algorithm, we used tools from fixed-parameter tractability. In recent years, such tools have been employed increasingly in the analysis of enumeration algorithms. Creignou et al. give a general overview of the field of parameterised enumeration [16]. The parameterised complexity of extension problems in particular has been treated by Meeks [46] and, very recently, by Casel et al. [11].

Complementing the theoretical analysis, there are several surveys reviewing the practical behaviour of enumeration algorithms for the transversal hypergraph. For example, Murakami and Uno compared several methods on artificial and real-world instances, mostly from frequent itemset mining, highlighting different use cases [47]. Gainer-Dewar and Vera-Licona conducted an exhaustive study involving more than 20 algorithms on hypergraphs stemming from various real-world applications [30]. However, it seems that there is no empirical work focussing on the structures arising in data profiling, although the connection to hitting sets is well-known also in practice [1, 18]. There are some articles on the performance of discovery algorithms for unique column combinations, but they do not discuss the topic in the context of hypergraphs. One example is the presentation of the `DUCC` algorithm by Heise et al. [35], where they evaluate their algorithm also on the `ncvoter_allc` and `uniprot`, among others. The running times are not directly comparable since we are not measuring the time it takes to compute the hypergraph of difference sets. Notwithstanding, they were unable to scale their algorithm beyond 60 columns of `ncvoter_allc` due to a timeout after 10 000 s, while our enumeration method can handle all 88 columns. The more recent `HyUCC` algorithm by Papenbrock and Naumann [49] also has some overlap with our work regarding the datasets they look at. Relating their results with our findings suggest that the run times of our implementation, working on arbitrary hypergraphs, is at least competitive with the current domain-specific state of the art. This is particularly interesting since they report the space consumption as a major bottleneck of `HyUCC`, which is not an issue for our algorithm.

7 Conclusion

In this work, we devised a backtracking algorithm to enumerate all minimal hitting sets of a hypergraph. We showed that this algorithm achieves a polynomial delay on inputs whose traversal hypergraphs have bounded rank. This makes it particularly suitable for the task of computing all minimal UCCs of a database, a domain where the solutions are expected to be small. However, the degree of the worst-case bound is dependent on the size of the largest solution. This bears the danger of running times that are, although output-polynomial, still prohibitively large in practise.

To check whether our enumeration technique succeeds within reasonable time frames, we evaluated it on a collection of publicly available datasets. The experiments showed that our method outperforms the brute-force baseline as soon as the solution space is non-trivial. As the total run time depends on the branching order of the vertices, we gave a heuristic based on the original database that achieved good results throughout by reducing the number of oracle calls. The main reason, however, for the good run times is the fact that the oracle calls are very fast on average. In particular, we can regularly avoid the worst-case behaviour in practise that leads to the large theoretical run time bound. The tree-based computation additionally obviates the need of expensive post-processing. Our algorithm terminates almost immediately after outputting the last solution.

Another feature of our algorithm is the low memory consumption. Approaching the enumeration of UCCs from a hitting set perspective and employing an extension oracle naturally forgoes the necessity of holding previous solutions in memory. This, however, seems to be a common problem even for current state-of-the-art discovery algorithms like `DUCC` and `HyUCC`. Papenbrock and Naumann, the authors of `HyUCC`, posed the following challenge in their conclusion [49].

> 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 [. . .].

While our pure enumeration times are already competitive, they seem not to be the true bottleneck of the computation; despite the fact that this is the NP-hard core of the problem. Instead, the preprocessing step of preparing the minimal difference sets of the database as input for the experiments regularly took longer than actually computing all transversals. Here, careful engineering has the potential of huge speed-ups on practical instances. Combining this with the natural advantages of our enumeration algorithm might yield the novel technique we are looking for.

References

[1] Z. Abedjan, L. Golab, and F. Naumann. Profiling Relational Data: A Survey. *The VLDB Journal*, 24:557–581, 2015.

[2] R. Agrawal and R. Srikant. Fast Algorithms for Mining Association Rules in Large Databases. In *Proceedings of the 20th International Conference on Very Large Data Bases (VLDB)*, pp. 487–499, 1994.

[3] C. Bazgan, L. Brankovic, K. Casel, H. Fernau, K. Jansen, K. Klein, M. Lampis, M. Liedloff, J. Monnot, and V. T. Paschos. The Many Facets of Upper Domination. *Theoretical Computer Science*, 717:2–25, 2018.

[4] C. Berge. *Hypergraphs - Combinatorics of Finite Sets*, Vol. 45 of *North-Holland Mathematical Library*. North-Holland, Amsterdam, 1989.

[5] J. C. Bioch and T. Ibaraki. Complexity of Identification and Dualization of Positive Boolean Functions. *Information and Computation*, 123: 50–63, 1995.

[6] A. Björklund, P. Kaski, Ł. Kowalik, and J. Lauri. Engineering Motif Search for Large Graphs. In *Proceedings of the 17th Workshop on Algorithm Engineering and Experiments (ALENEX)*, pp. 104–118, 2015.

[7] T. Bläsius, T. Friedrich, and M. Schirneck. The Parameterized Complexity of Dependency Detection in Relational Databases. In *Proceedings of the 11th International Symposium on Parameterized and Exact Computation (IPEC)*, pp. 6:1–6:13, 2016.

[8] E. Boros, V. Gurvich, and P. L. Hammer. Dual Subimplicants of Positive Boolean Functions. *Optimization Methods and Software*, 10:147–156, 1998.

[9] E. Boros, K. M. Elbassioni, V. Gurvich, and L. Khachiyan. An Efficient Incremental Algorithm for Generating All Maximal Independent Sets in Hypergraphs of Bounded Dimension. *Parallel Processing Letters*, 10:253–266, 2000.

[10] E. Boros, K. Elbassioni, V. Gurvich, L. Khachiyan, and K. Makino. Dual-Bounded Generating Problems: All Minimal Integer Solutions for a Monotone System of Linear Inequalities. *SIAM Journal on Computing*, 31:1624–1643, 2002.

[11] K. Casel, H. Fernau, M. Khosravian Ghadikolaei, J. Monnot, and F. Sikora. On the Complexity of Solution Extension of Optimization Problems. *ArXiv e-prints*, 2018.

[12] J. Chen and F. 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.

[13] J. Chen, B. Chor, M. R. Fellows, X. Huang, D. Juedes, I. A. Kanj, and G. Xia. Tight Lower Bounds for Certain Parameterized NP-hard Problems. *Information and Computation*, 201: 216–231, 2005.

[14] J. Chen, X. Huang, I. A. Kanj, and G. Xia. On the Computational Hardness Based on Linear FPT-Reductions. *Journal of Combinatorial Optimization*, 11:231–247, 2006.

[15] J. Chen, X. Huang, I. A. Kanj, and G. Xia. Strong Computational Lower Bounds via Parameterized Complexity. *Journal of Computer and System Sciences*, 72:1346–1367, 2006.

[16] N. Creignou, A. Meier, J. Müller, J. Schmidt, and H. Vollmer. Paradigms for Parameterized Enumeration. *Theory of Computing Systems*, 60: 737–758, 2017.

[17] M. Cygan, F. V. Fomin, Ł. Kowalik, D. Lokshtanov, D. Marx, M. Pilipczuk, M. Pilipczuk, and S. Saurabh. *Parameterized Algorithms*. Springer, Heidelberg, New York, 2015.

[18] C. Date. *An Introduction to Database Systems*. Addison-Wesley Longman Publishing, Boston, 8th edition, 2003.

[19] J. Demetrovics and V. D. Thi. Keys, Antikeys and Prime Attributes. *Annales Universitatis Scientiarum Budapestinensis de Rolando Eötvös Nominatae Sectio Computatorica*, 8:35–52, 1987.

[20] C. Domingo, N. Mishra, and L. Pitt. Efficient Read-Restricted Monotone CNF/DNF Dualization by Learning with Membership Queries. *Machine Learning*, 37:89–110, 1999.

[21] R. G. Downey and M. R. Fellows. *Fundamentals of Parameterized Complexity*. Texts in Computer Science. Springer, Heidelberg, London, 2013.

[22] T. Eiter. Exact Transversal Hypergraphs and Application to Boolean μ-Functions. *Journal of Symbolic Computation*, 17:215 – 225, 1994.

[23] T. Eiter and G. Gottlob. Identifying the Minimal Transversals of a Hypergraph and Related Problems. *SIAM Journal on Computing*, 24: 1278–1304, 1995.

[24] T. Eiter, G. Gottlob, and K. Makino. New Results on Monotone Dualization and Generating Hypergraph Transversals. *SIAM Journal on Computing*, 32:514–537, 2003.

[25] T. Eiter, K. Makino, and G. Gottlob. Computational Aspects of Monotone Dualization: A Brief Survey. *Discrete Applied Mathematics*, 156: 2035 – 2049, 2008.

[26] K. M. Elbassioni, M. Hagen, and I. Rauf. Some Fixed-Parameter Tractable Classes of Hypergraph Duality and Related Problems. In *Proceedings of the 3rd International Symposium on Parameterized and Exact Computation (IPEC)*, pp. 91–102, 2008.

[27] J. Flum and M. Grohe. *Parameterized Complexity Theory.* Texts in Theoretical Computer Science. An EATCS Series. Springer, Heidelberg, New York, 2006.

[28] F. V. Fomin, F. Grandoni, A. V. Pyatkin, and A. A. Stepanov. Combinatorial Bounds via Measure and Conquer: Bounding Minimal Dominating Sets and Applications. *ACM Transactions on Algorithms*, 5:9:1–9:17, 2008.

[29] M. L. Fredman and L. Khachiyan. On the Complexity of Dualization of Monotone Disjunctive Normal Forms. *Journal of Algorithms*, 21:618–628, 1996.

[30] A. Gainer-Dewar and P. Vera-Licona. The Minimal Hitting Set Generation Problem: Algorithms and Computation. *Journal on Discrete Mathematics*, 31:63–100, 2017.

[31] H. Garcia-Molina and D. Barbara. How to Assign Votes in a Distributed System. *Journal of the ACM*, 32:841–860, 1985.

[32] C. Giannella and C. Wyss. Finding Minimal Keys in a Relation Instance, 1999. `http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.41.7086`.

[33] G. Gogic, C. H. Papadimitriou, and M. Sideri. Incremental Recompilation of Knowledge. *Journal of Artificial Intelligence Research*, 8:23–37, 1998.

[34] M. Hagen. *Algorithmic and Computational Complexity Issues of MONET.* Cuvillier, Göttingen, 2008.

[35] A. Heise, J.-A. Quiané-Ruiz, Z. Abedjan, A. Jentzsch, and F. Naumann. Scalable Discovery of Unique Column Combinations. *Proceedings of the VLDB Endowment*, 7:301–312, 2013.

[36] R. Impagliazzo and R. Paturi. On the Complexity of k-SAT. *Journal of Computer and System Sciences*, 62:367–375, 2001.

[37] R. Impagliazzo, R. Paturi, and F. Zane. Which Problems Have Strongly Exponential Complexity? *Journal of Computer and System Science*, 63: 512–530, 2001.

[38] D. S. Johnson, C. H. Papadimitriou, and M. Yannakakis. On Generating All Maximal Independent Sets. *Information Processing Letters*, 27:119–123, 1988.

[39] M. M. Kanté, V. Limouzy, A. Mary, L. Nourine, and T. Uno. A Polynomial Delay Algorithm for Enumerating Minimal Dominating Sets in Chordal Graphs. In *Proceedings of the 41st International Workshop on Graph-Theoretic Concepts in Computer Science (WG)*, pp. 138–153, 2015.

[40] L. Khachiyan, E. Boros, K. M. Elbassioni, and V. Gurvich. A Global Parallel Algorithm for the Hypergraph Transversal Problem. *Information Processing Letters*, 101:148–155, 2007.

[41] S. Kruse, T. Papenbrock, H. Harmouch, and F. Naumann. Data Anamnesis: Admitting Raw Data into an Organization. *IEEE Data Engineering Bulletin*, 39:8–20, 2016.

[42] E. 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.

[43] D. Maier. *The Theory of Relational Databases.* Computer Science Press, Rockville (MD), 1983.

[44] H. Mannila and K.-J. Räihä. Dependency Inference. In *Proceedings of the 13th International Conference on Very Large Data Bases (VLDB)*, pp. 155–158, 1987.

[45] A. Mary and Y. Strozecki. Efficient Enumeration of Solutions Produced by Closure Operations. In *Proceedings of the 33rd Symposium on Theoretical Aspects of Computer Science (STACS)*, pp. 52:1–52:13, 2016.

[46] K. Meeks. Randomised Enumeration of Small Witnesses Using a Decision Oracle. In *Proceedings*

of the 11th International Symposium on Parameterized and Exact Computation (IPEC), pp. 22:1–22:12, 2016.

[47] K. Murakami and T. Uno. Efficient Algorithms for Dualizing Large-scale Hypergraphs. *Discrete Applied Mathematics*, 170:83 – 94, 2014.

[48] Ø. Ore. *Theory of Graphs*, Vol. 38 of *Colloquium Publications - American Mathematical Society*. American Mathematical Society, Providence, 1962.

[49] T. Papenbrock and F. Naumann. A Hybrid Approach for Efficient Unique Column Combination Discovery. In *Proceedings of the 17th Datenbanksysteme für Business, Technologie und Web (BTW)*, pp. 195–204, 2017.

[50] T. Papenbrock, J. Ehrlich, J. Marten, T. Neubert, J.-P. Rudolph, M. Schönberg, J. Zwiener, and F. Naumann. Functional Dependency Discovery: An Experimental Evaluation of Seven Algorithms. *Proceedings of the VLDB Endowment*, 8:1082–1093, 2015.

[51] R. C. Read and R. E. Tarjan. Bounds on Backtrack Algorithms for Listing Cycles, Paths, and Spanning Trees. *Networks*, 5:237–252, 1975.

[52] S. Tsukiyama, M. Ide, H. Ariyoshi, and I. Shirakawa. A New Algorithm for Generating All the Maximal Independent Sets. *SIAM Journal on Computing*, 6:505–517, 1977.