Mining Approximate Acyclic Schemes from Relations

Batya Kenig[1] Pranay Mundra[2] Guna Prasaad[1] Babak Salimi[1] Dan Suciu[1]
[1] Computer Science and Engineering
University of Washington
batyak, guna, bsalimi, suciu@cs.washington.edu
[2] Department of Mathematics
University of Washington
pranay99@uw.edu

ABSTRACT

Acyclic schemes have numerous applications in databases and in machine learning, such as improved design, more efficient storage, and increased performance for queries and machine learning algorithms. Multivalued dependencies (MVDs) are the building blocks of acyclic schemes. The discovery from data of both MVDs and acyclic schemes is more challenging than other forms of data dependencies, such as Functional Dependencies, because these dependencies do not hold on subsets of data, and because they are very sensitive to noise in the data; for example a single wrong or missing tuple may invalidate the schema. In this paper we present Maimon, a system for discovering *approximate* acyclic schemes and MVDs from data. We give a principled definition of approximation, by using notions from information theory, then describe the two components of Maimon: mining for approximate MVDs, then reconstructing acyclic schemes from approximate MVDs. We conduct an experimental evaluation of Maimon on 20 real-world datasets, and show that it can scale up to 1M rows, and up to 30 columns.

ACM Reference Format:
Batya Kenig[1] Pranay Mundra[2] Guna Prasaad[1] Babak Salimi[1] Dan Suciu[1]. 2020. Mining Approximate Acyclic Schemes from Relations. In *Proceedings of the 2020 ACM SIGMOD International Conference on Management of Data (SIGMOD'20), June 14–19, 2020, Portland, OR, USA.* ACM, New York, NY, USA, 16 pages. https://doi.org/10.1145/3318464.3380573

1 INTRODUCTION

Acyclic schemes have numerous applications in databases and in machine learning. Originally introduced by Beeri [5], they have lead to Yannakakis celebrated linear time query evaluation algorithms [43], and are used widely today in database design [13, 31], to speed up query evaluation with multiple aggregates [25], and to speed up machine learning applications such as ridge linear regression, classification trees, and regression trees [24, 39, 40]. When considering which types of schemes to fit the data, acyclic schemes are the natural choice due to their many desirable properties [6]. In this paper we study the following discovery problem: given a database consisting of a single relation, generate a set of acyclic schemes that fit the data to a large extent. For a simple illustration, consider the database shown on the left of Figure 1. It can be decomposed into an acyclic schema with four relations, shown on the right.

The building blocks of acyclic schemes are Multivalued Dependencies, MVDs. Every acyclic schema is fully specified by the set of MVDs that it implies, which we call its *support*. Therefore, when mining acyclic schemes, the first step is to mine the MVDs satisfied by the data. MVDs were first introduced by Fagin [13], which used them to introduce the 4th normal form, a generalization of the Boyce-Codd normal form (BCNF) [10]. They were studied extensively in the database literature [2, 4, 14, 28], and the literature on graphical models [17, 41], and have recently been used as part of a data repairing solution to enforce fairness of ML systems [36, 37]. The methods used to synthesize an acyclic schema from a set of MVDs are well known [3, 7, 13, 32]. However, despite their importance, there is little research on the *discovery* of MVDs from data [1].

Work most closely related to the discovery of MVDs has been on discovering Functional Dependencies (FDs) and Unique Column Combinations (UCCs) [8, 21, 26, 27, 33, 35, 42]. These are special cases of MVDs, but MVDs are more general. Discovering all FDs and all UCCs is insufficient for discovering acyclic schemes. The only work that addressed the discovery problem for MVDs is by Savnik and Flach [38] and a master thesis by Draeger [12], and none of them address the more challenging task of discovering acyclic schemes.

There are two major challenges that make the discovery of MVDs and acyclic schemes, much harder than that of FDs and UCCs. First, they don't hold on subsets of the data. If a relation satisfies an FD, or a UCC, then every subset also satisfies the FD, or UCC, and this is exploited by many discovery algorithms, e.g FastFD [42] mines FDs in all subsets of size 2,

SIGMOD'20, June 14–19, 2020, Portland, OR, USA

ACM ISBN 978-1-4503-6735-6/20/06…$15.00
https://doi.org/10.1145/3318464.3380573

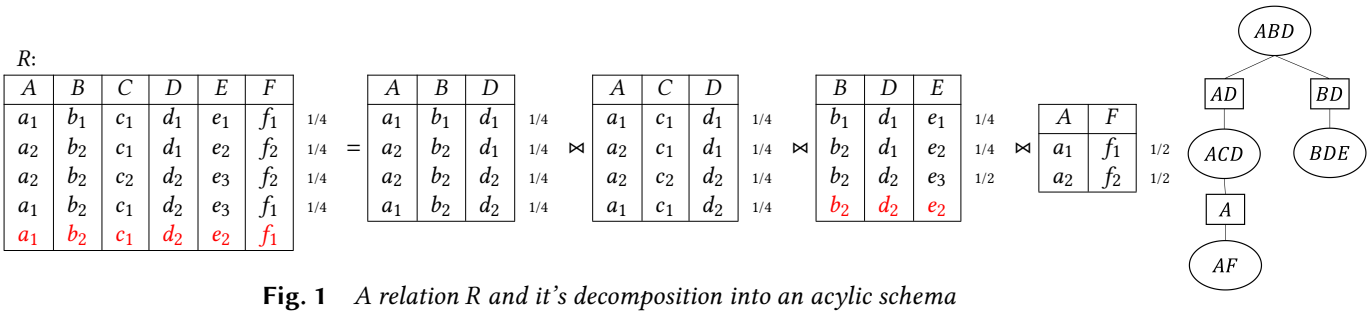

Fig. 1 *A relation R and it's decomposition into an acylic schema*

Fig. 2 *Join Tree*

while HyFD [35] mines FDs in a small subset extracted from the data. This property fails for MVDs, preventing us from considering subsets of the data. Second, MVDs and acyclic schemes are much more sensitive to data errors than FDs and UCCs. Even a single missing tuple may invalidate an MVD or schema. Real-world data often has important dependencies that do not hold exactly, but, if discovered, are very useful for a variety of applications. For that reason, in this paper we study the problem of discovering *approximate* MVDs and consequently, *approximate* acyclic schemes.

We present Maimon[1], the first system for discovering approximate MVDs and acyclic schemes in the data. We introduce a principled notion of approximation, based on information theory, and develop the necessary theory for reasoning about approximate MVDs and schemes. We then describe algorithms for mining MVDs and schemes, and evaluate their scalability on real-world datasets of up to 1M rows, and 30 attributes. By allowing approximations, Maimon finds more interesting schemes without incurring too high a loss (i.e., spurious tuples). We make several contributions.

Our first contribution is to introduce a principled definition of approximation, and study its properties. Kivinen and Mannila [26] give three definitions of approximate functional dependencies, and Kruese and Naumann use one of them in their approximate FDs and UCCs discovery algorithm [27]. We propose a metric of approximation based on information theory. Building on earlier work by Lee [30], each MVD or acyclic schema is associated with an information theoretic expression, which represents the degree of approximation.

Second, we propose novel algorithms for mining approximate MVDs and approximate acyclic schemes. For mining MVDs, our theoretical results prove that we do not need to discover *all* approximate MVDs, but only the so-called *full* MVDs with *minimal* separators. Our algorithm builds on previous results by Gunopulos et al. [20] for discovering the most *specific* sentences in the data that meet a certain criterion (e.g., maximal sets of items whose frequency in the data is above a given threshold). Following the discovery of the MVDs that hold in the data, we turn to the task of enumerating the acyclic schemes that can be synthesized from the set of discovered MVDs. Our algorithm is based on an approach for efficiently enumerating the maximal independent sets of a graph [11, 22], which has also been applied to the problem of enumerating tree decompositions [9].

Third, we evaluate Maimon on 20 real-world datasets that are part of the Metanome project that provides a repository of benchmarks for a variety of data profiling tasks that include the discovery of data dependencies. The datasets chosen for evaluation have been used in a large body of work on mining exact and approximate FDs [8, 12, 27, 33–35]. We show that Maimon scales up to 1M rows, and up to 30 columns. We empirically show that the loss entailed by the generated acyclic schemes (i.e., number of spurious tuples), strongly correlates with the information theoretic measure of approximation we develop herein. We also show that a larger degree of approximation enables the discovery of schemes that exhibit a larger degree of decomposition, that leads to significant savings in storage. These schemes generally have more relations, and the *width* of the schema (i.e., relation with the largest number of attributes), is smaller.

The most expensive operation of Maimon is the computation of the entropy $H(X)$ of a set of attributes X. Each such computation requires a full scan over the data, and this is prohibitively expensive due to the exponential number of subsets of attributes. We describe a novel, efficient approach to computing entropy, which reduces the problem to a set of main-memory SQL queries. Our method is inspired by the *PLI cache* (Position List Indices) data structure used for mining both exact and approximate FDs [21, 27].

To sum up, the contributions of this work are as follows:

(1) We define a principled notion of approximate data dependencies based on information theory, and study its properties; Sec. 4 and 5.
(2) We describe a novel MVD enumeration algorithms and acyclic schema enumeration algorithm; Sec. 6 and 7.
(3) We conduct an extensive experimental evaluation on 20 real datasets; Sec. 8.

[1] Maimon : Multivalued Approximate Inference Mining and NOrmalization.

2 RUNNING EXAMPLE

We will use the following running example in this paper. Consider the relation R over the signature $\Omega = \{A, B, C, D, E, F\}$ in Figure 1. Ignore the probabilities, we will use them in Sec. 3. Also, ignore for now the last row (in red). The table with four rows can be decomposed into four tables, shown in the figure. More precisely, the following join dependency holds: $R = R[ABD] \bowtie R[ACD] \bowtie R[BDE] \bowtie R[AF]$. The schema of these four tables is *acyclic*, because it admits a join tree, shown in Fig. 2 (reviewed in Sec. 3). Our goal is to discover this acyclic schema from the data R. For that, we note that the acyclic schema can be entirely described by three Multivalued Dependencies: $BD \twoheadrightarrow E|ACF$, $AD \twoheadrightarrow CF|BE$, and $A \twoheadrightarrow F|BCDE$. Each corresponds to one edge of the join tree: the left hand size of the MVD (that we call the *key*) is the label of that edge, while the two sets of attributes correspond to the subtrees connected by the edge. For example, the edge $ACD \overset{\text{AD}}{-} ABD$ in the join tree defines the MVD $AD \twoheadrightarrow CF|BE$. The key AD "separates" the attributes CF in one subtree from BE in the other subtree, and we will also call such a set a *separator*. Since MVDs are the building blocks of acyclic schemas, their discovery is a prerequisite for discovering acyclic schemas, and our first task is to discover MVDs from data, then use them to discover acyclic schemas.

Consider the 5'th row in R, shown in red. By adding it, we need to add a 4'th row to $R[BDE]$, also shown in red. However, now the join dependency no longer holds exactly, because $R[ABD] \bowtie R[ACD] \bowtie R[BDE] \bowtie R[AF]$ contains a spurious tuple, namely $(a_2, b_2, c_2, d_2, e_2, f_2)$, which is not in R (it is not shown in the Figure); the first two MVDs no longer hold, only $A \twoheadrightarrow F|BCDE$ still holds, and the acyclic schema is no longer a correct decomposition of R. Yet the schema can still be useful for many applications, even it if leads to a spurious tuple. Insisting on exact acyclic schemas would severely restrict their applications, and also make them very brittle since the addition of one single tuple would invalidate the schema. In this paper we compute *approximate* acyclic schemas, and *approximate* MVDs. By allowing approximations, the schema shown in the figure is still considered valid for the data, despite the spurious tuple.

3 BACKGROUND

Table 1 summarizes the notations in this paper. We denote by $[n] = \{1, \ldots, n\}$. Let Ω be a set of variables, also called attributes. If $X, Y \subseteq \Omega$, then XY denotes $X \cup Y$.

3.1 Data Dependencies

Fix a relation instance R of size $N = |R|$, and schema Ω. For $Y \subseteq \Omega$ we let $R[Y]$ denote the projection of R onto the attributes Y.

Ω	set of variables (attributes)
$n = \|\Omega\|$	number of variables (attributes)
$X, Y, A, B, \ldots$	sets of variables $\subseteq \Omega$
S	a schema $= \{\Omega_1, \ldots, \Omega_m\}$
$X \twoheadrightarrow Y\|Z$	a standard MVD
$X \twoheadrightarrow Y_1\|Y_2\|\cdots\|Y_m$	an MVD [4]
$(\mathcal{T}, \chi)$	a join tree
$H(X)$	entropy of a set of variables X
$H(Y\|X), I(Y;Z\|X)$	entropic measures
$\mathcal{J}(\mathcal{T}, \chi)$	the entropic measure in Eq.(6)
$\mathcal{J}(\mathrm{S})$	$\mathcal{J}$ of any join tree for S
$\mathcal{J}(X \twoheadrightarrow Y_1\|\cdots\|Y_m)$	$\mathcal{J}$ of the schema $\{XY_1, \ldots, XY_m\}$
$\mathcal{J}(X \twoheadrightarrow Y\|Z)$	$= I(Y;Z\|X)$
R	a relation
$N = \|R\|$	number of tuples
$R \models \mathrm{AJD}(\mathrm{S})$	R satisfies an acyclic join dependency
$R \models_\varepsilon \mathrm{AJD}(\mathrm{S})$	R ε-satisfies an acyclic join dependency

Table 1: Notations

Let $X, Y, Z \subseteq \Omega$. A *schema* is a set $\mathrm{S} = \{\Omega_1, \ldots, \Omega_k\}$ such that $\bigcup_{i=1}^k \Omega_i = \Omega$ and $\Omega_i \not\subseteq \Omega_j$ for $i \neq j$. We say that the relation instance R satisfies the *join dependency* JD(S), and write $R \models \mathrm{JD}(\mathrm{S})$, if $R = \bowtie_{i=1}^k R[\Omega_i]$. We say that R satisfies the *multivalued dependency* (MVD) $\phi = X \twoheadrightarrow Y_1|Y_2|\ldots|Y_m$ where $m \geq 2$, the Y_is are pairwise disjoint, and $XY_1 \cdots Y_m = \Omega$, if $R = R[XY_1] \bowtie \cdots \bowtie R[XY_m]$. We call X the *key* of the MVD and $\{Y_1, \ldots, Y_m\}$ it's *dependents*, denoted $\mathrm{key}(\phi) = X$ and $\mathrm{dep}(\phi) = \{Y_1, \ldots, Y_m\}$. Most of the literature considers only MVDs with $m = 2$, which we call here *standard MVDs*. Beeri et al. [4] noted that a generalized MVD can concisely encode multiple MVDs; for example $X \twoheadrightarrow A|B|C$ holds iff $X \twoheadrightarrow AB|C$, $X \twoheadrightarrow A|BC$ and $X \twoheadrightarrow AC|B$ hold. We review a *join tree* from [6]:

Definition 3.1. A *join tree* is a pair $(\mathcal{T}, \chi)$ where $\mathcal{T}$ is an undirected tree, and χ is a function that maps each $u \in \mathsf{nodes}(\mathcal{T})$ to a set of variables $\chi(u)$, called a *bag*, such that the following *running intersection* property holds: for every variable X, the set $\{u \in \mathsf{nodes}(\mathcal{T}) \mid X \in \chi(u)\}$ is a connected component of $\mathcal{T}$. We denote by $\chi(\mathcal{T}) \overset{\text{def}}{=} \bigcup_u \chi(u)$, the set of variables of the join tree.

We often denote the join tree as $\mathcal{T}$, dropping χ when it is clear from the context. The *schema* defined by $\mathcal{T}$ is $\mathrm{S} = \{\Omega_1, \ldots, \Omega_m\}$, where $\Omega_1, \ldots, \Omega_m$ are the bags of $\mathcal{T}$. We call a schema S *acyclic* if there exists a join tree whose schema is S. Since we required $\Omega_i \not\subseteq \Omega_j$ for $i \neq j$, one can prove that any acyclic schema with n attributes and m relations satisfies $m \leq n$. We say that a relation R satisfies the *acyclic join dependency* S, and denote $R \models \mathrm{AJD}(\mathrm{S})$, if S is acyclic and

$R \models \text{JD(S)}$. An MVD $X \twoheadrightarrow Y_1|\cdots|Y_m$ represents a simple acyclic schema, namely $\text{S} = \{XY_1, XY_2, \ldots, XY_m\}$.

Let $\text{S} = \{\Omega_1, \ldots, \Omega_m\}$ be an acyclic schema with join tree $(\mathcal{T}, \chi)$. We associate to every $(u, v) \in \texttt{edges}(\mathcal{T})$ an MVD $\phi_{u,v}$ as follows. Let $\mathcal{T}_u$ and $\mathcal{T}_v$ be the two subtrees obtained by removing the edge (u, v). Then, we denote by $\phi_{u,v} \stackrel{\text{def}}{=} \chi(u) \cap \chi(v) \twoheadrightarrow \chi(\mathcal{T}_u)|\chi(\mathcal{T}_v)$. We call the *support of* $\mathcal{T}$ the set of $m-1$ MVDs associated to its edges, in notation $\text{MVD}(\mathcal{T}) = \{\phi_{u,v} \mid (u, v) \in \texttt{edges}(\mathcal{T})\}$. If $\mathcal{T}$ defines the acyclic schema S, then it satisfies $R \models \text{AJD(S)}$ iff it satisfies all MVDs in its support: $R \models \phi_{u,v}$ for all $\phi_{u,v} \in MVD(\mathcal{T})$ [6, Thm. 8.8].

Example 3.2. We will illustrate with the running example from Sec. 2. The tree in Fig. 2 is a join tree. Its bags are the ovals labeled *AF*, *ACD*, *ABD*, and *BDE*, and it is custom to show the intersection of two bags on the connecting edge. $\text{MVD}(\mathcal{T}) = \{BD \twoheadrightarrow E|ACF, AD \twoheadrightarrow CF|BE, A \twoheadrightarrow F|BCDE\}$.

3.2 Information Theory

Lee [29, 30] gave an equivalent formulation of data dependencies in terms of information measures; we review this briefly here, after a short background on information theory.

Let X be a random variable with a finite domain $\mathcal{D}$ and probability mass p (thus, $\sum_{x \in \mathcal{D}} p(x) = 1$). Its entropy is:

$$H(X) \stackrel{\text{def}}{=} \sum_{x \in \mathcal{D}} p(x) \log \frac{1}{p(x)} \tag{1}$$

If $N = |\mathcal{D}|$ then $H(X) \leq \log N$, and equality holds iff p is uniform. For a set of jointly distributed random variables $\Omega = \{X_1, \ldots, X_n\}$ we define the function $H : 2^\Omega \to \mathbb{R}$ as the entropy of the joint random variables in the set. For example, $H(X_1X_2) = \sum_{x_1 \in \mathcal{D}_1, x_2 \in \mathcal{D}_2} p(x_1, x_2) \log \frac{1}{p(x_1, x_2)}$. Let $A, B, C \subseteq \Omega$. The *mutual information* $I(B; C|A)$ is defined as:

$$I(B; C|A) \stackrel{\text{def}}{=} H(AB) + H(AC) - H(ABC) - H(A) \tag{2}$$

It is known that the conditional independence $p \models B \perp C \mid A$ (i.e., B is independent of C given A) holds iff $I(B; C|A) = 0$.

In this paper we use only the following two properties of the mutual information:

$$I(B; C|A) \geq 0 \tag{3}$$

$$I(B; CD|A) = I(B; C|A) + I(B; D|AC) \tag{4}$$

The first inequality follows from monotonicity and submodularity (it is in fact equivalent to them); the second equality is called the *chain rule*. All consequences of these two (in)equalities are called *Shannon inequalities*; for example, monotonicity $H(AB) \geq H(A)$ is a Shannon inequality because it follows from (3) by setting $B = C$.

Let R be relation with attributes $\Omega = \{X_1, \ldots, X_n\}$ and N tuples. The *empirical distribution* is the uniform distribution over its tuples: $\forall t \in R, p(t) = 1/N$. It's entropy satisfies $H(\Omega) = \log N$. For $\alpha \subseteq [n]$, we denote by X_α the set of variables $X_i, i \in \alpha$, and denote by $R(X_\alpha = x_\alpha)$ the subset of tuples $t \in R$ where $t[X_\alpha] = x_\alpha$, for fixed values x_α. By uniformity, the marginal probability is $p(X_\alpha = x_\alpha) = \frac{|R(X_\alpha = x_\alpha)|}{N}$, and therefore:

$$H(X_\alpha) \stackrel{\text{def}}{=} \log N - \frac{1}{N} \sum_{x_\alpha \in \mathcal{D}_\alpha} |R(X_\alpha = x_\alpha)| \log |R(X_\alpha = x_\alpha)| \tag{5}$$

The sum above can be computed using a simple SQL query: `Select` X_α, `count(*)` $\times \log($`count(*)`$)$ `From` R `Group By` X_α.

Lee [29, 30] formalized the following connection between database constraints, and entropic measures. Let $(\mathcal{T}, \chi)$ be a join tree (def. 3.1). We define the following expression:

$$\mathcal{J}(\mathcal{T}, \chi) \stackrel{\text{def}}{=} \sum_{\substack{v \in \\ \texttt{nodes}(\mathcal{T})}} H(\chi(v)) - \sum_{\substack{(v_1, v_2) \in \\ \texttt{edges}(\mathcal{T})}} H(\chi(v_1) \cap \chi(v_2)) - H(\chi(\mathcal{T})) \tag{6}$$

We abbreviate it with $\mathcal{J}(\mathcal{T})$, or $\mathcal{J}$, when $\mathcal{T}, \chi$ are clear from the context; we will prove later (Th. 5.1) that $\mathcal{J} \geq 0$ is a Shannon inequality. Lee proved that $\mathcal{J}$ depends only on the schema S defined by the join tree, and not on the tree itself. To see this on a simple example, consider the MVD $X \twoheadrightarrow U|V|W$ and its associated acyclic schema $\{XU, XV, XW\}$. If we consider the join tree $XU - XV - XW$, then $\mathcal{J} = H(XU) + H(XV) + H(XW) - 2H(X) - H(XUVW)$. Another join tree is $XU - XW - XV$, and $\mathcal{J}$ is the same. Therefore, if S is acyclic, then we write $\mathcal{J}(\text{S})$ to denote $\mathcal{J}(\mathcal{T})$ for any join tree of S. We denote by $\mathcal{J}(X \twoheadrightarrow Y_1|\cdots|Y_m) \stackrel{\text{def}}{=} H(XY_1) + \cdots + H(XY_m) - (m-1)H(X) - H(XY_1 \cdots Y_m)$ for any sets of variables $X, Y_1, \ldots, Y_m$ where $Y_1, \ldots, Y_m$ are pairwise disjoint, even when $XY_1 \cdots Y_m$ is not necessarily Ω. When $m = 2$, then $J(X \twoheadrightarrow Y|Z) = I(Y; Z|X)$.

Theorem 3.3. ([30]) *Let H be the entropy of the empirical distribution on R, and let* S *be any acyclic schema. Then $R \models \text{AJD(S)}$ iff $\mathcal{J}(\text{S}) = 0$.*

In the particular case of a standard MVD, Lee's result implies that $R \models X \twoheadrightarrow Y|Z$ if and only if $I(Y; Z|X) = 0$.

Example 3.4. Continuing Example 3.2, the empirical distribution of the relation R in Fig 1 (without the red tuple) assigns probability $1/4$ to each tuple. Thus, $H(ABCDEF) = \log 4 = 2$. The marginal probabilities need not be uniform, e.g. the marginals for BDE are $1/4, 1/4, 1/2$, and thus $H(BDE) = 1/4 \log 4 + 1/4 \log 4 + 1/2 \log 2 = 3/2$. The value of $\mathcal{J}$ is: $\mathcal{J}(\mathcal{T}) = H(AF) + H(ACD) + H(ABD) + H(BDE) - H(A) - H(AD) - H(BD) - H(ABCDEF)$. For the empirical distribution in the figure, this quantity is 0.

4 PROBLEM STATEMENT

Our main goal is to discover an acyclic schema for a given relation instance R. Since exact schemas are very sensitive to data errors, Maimon discovers approximate schemas.

Definition 4.1 (Approximate Acyclic Schema). Fix a relation instance R, and $\varepsilon \geq 0$. We say that an acyclic schema $\mathbf{S}$ is an *ε-schema* for R, or simply *approximate schema*, if $\mathcal{J}(\mathbf{S}) \leq \varepsilon$. In notation, $R \models_\varepsilon \text{AJD}(\mathbf{S})$.

Maimon takes as input $\varepsilon \geq 0$ and discovers approximate acyclic schemas for R. By Lee's theorem, if we set $\varepsilon = 0$, then Maimon returns exact schemas. In practice, a relation R may not have any exact schemas, or may have very limited schemas; by allowing $\varepsilon \geq 0$ we may find approximate schemas that are quite useful for many applications.

Problem 4.1 (Schema Enumeration Problem). *Given a relational instance R, enumerate the approximate acyclic schemas of R.*

In practice, we are not interested in enumerating *all* approximate acyclic schemas of R. This would take a prohibitively long time, and some acyclic schemas are superior to others. For example, consider a relation over four attributes that satisfies the acyclic join dependency $\mathbf{S} = \{XA, XB, XC\}$. The following acyclic join dependencies also hold in R: $\{XAB, XC\}$, $\{XAC, XB\}$, and $\{XA, XBC\}$. The latter schemas are less useful than $\mathbf{S} = \{XA, XB, XC\}$ that leads to a larger degree of decomposition. Therefore, in this paper we address the problem of enumerating acyclic schemas that cannot be extended (i.e., with additional relational instances) while continuing to satisfy the accuracy threshold $\mathcal{J}(\mathbf{S}) \leq \varepsilon$.

We derive the approximate schemas from the MVDs in their support. Since an MVD is, in particular, an acyclic schema, Def. 4.1 applies to them as well: a ε-MVD is one for which $\mathcal{J}(X \twoheadrightarrow Y_1|\cdots|Y_m) \leq \varepsilon$. Our second problem is:

Problem 4.2 (MVD Enumeration Problem). *Given a relational instance R, enumerate the approximate MVDs of R.*

Maimon works as follows. The user provides a parameter $\varepsilon \geq 0$. In the first phase, Maimon enumerates ε-MVDs, using the algorithm in Sec. 6. When it finishes, or after a timeout, it starts the second phase, where it enumerates approximate schemas with support from the set returned by the first phase, using the algorithm in Sec. 7. The support of an approximate schema $\mathbf{S}$ with m relations consists of $m-1$ ε-MVDs. In the following section we show that $\mathcal{J}(\mathbf{S}) \leq (m-1)\varepsilon \leq n\varepsilon$. This allows the user to configure ε according to the worst-case information theoretic error entailed by the schema.

5 THREE MAIN TECHNIQUES

We describe here three main techniques that allow us to design efficient schema- and MVD-discovery algorithms. The first reduces the approximate schema discovery to approximate MVD discovery, the next two prune the space of MVDs that need to be discovered.

5.1 From MVDs to Acyclic Schemas

Beeri at al. [6] showed that, for exact constraints, an acyclic schema over m relations is equivalent to the set of $m-1$ MVDs in its support. We give here a non-trivial generalization to approximate schemas and MVDs. We start with two simple inequalities which we need throughout the paper:

Proposition 5.1. *Let $Y_1, Z_1, \ldots, Y_m, Z_m$ be pairwise disjoint sets of variables, and let X be any set of variables. Then the following are Shannon inequalities:*

$$\mathcal{J}(X \twoheadrightarrow Y_1|\cdots|Y_m) \leq \mathcal{J}(X \twoheadrightarrow Y_1Z_1|\cdots|Y_mZ_m) \quad (7)$$

$$\mathcal{J}(XZ_1\cdots Z_m \twoheadrightarrow Y_1|\cdots|Y_m) \leq \mathcal{J}(X \twoheadrightarrow Y_1Z_1|\cdots|Y_mZ_m) \quad (8)$$

Proof. The first inequality follows from this chain of inequalities: $\mathcal{J}(X \twoheadrightarrow Y_1|\cdots|Y_m) \leq \mathcal{J}(X \twoheadrightarrow Y_1Z_1|Y_2|\cdots|Y_m) \leq \mathcal{J}(X \twoheadrightarrow Y_1Z_1|Y_2Z_2|\cdots|Y_m) \leq \cdots$; to prove it, we show only the first step (the others are similar), which follows by observing $\mathcal{J}(X \twoheadrightarrow Y_1|\cdots|Y_m) + I(Z_1; Y_2\cdots Y_m|XY_1) = \mathcal{J}(X \twoheadrightarrow Y_1Z_1|\cdots|Y_m)$ then using inequality (3). The second inequality follows from a similar chain, where the first step follows from $\mathcal{J}(XZ_1 \twoheadrightarrow Y_1|\cdots|Y_m) + \sum_{i=2}^m I(Y_i; Z_1|X) = \mathcal{J}(X \twoheadrightarrow Y_1Z_1|Y_2|\cdots|Y_m)$ and the inequality follows from (3). □

Let $(\mathcal{T}, \chi)$ be a join tree, defining an acyclic schema $\mathbf{S}$ over the variables $\chi(\mathcal{T}) = \Omega$. Choose an arbitrary root, orient the tree accordingly, and let $u_1, \ldots, u_m$ be a depth-first enumeration of $\texttt{nodes}(\mathcal{T})$. Thus, u_1 is the root, and for every $i > 1$, $\texttt{parent}(u_i)$ is some node u_j with $j < i$. For every i, we define $\Omega_i \stackrel{\text{def}}{=} \chi(u_i)$, $\Omega_{i:j} \stackrel{\text{def}}{=} \bigcup_{\ell=i,j} \Omega_\ell$, and $\Delta_i \stackrel{\text{def}}{=} \chi(\texttt{parent}(u_i)) \cap \chi(u_i)$ (by the running intersection property this is equal to $\Omega_{1:(i-1)} \cap \Omega_i$). We prove:

Theorem 5.1. *The following hold:*

$$\mathcal{J}(\mathcal{T}) = \sum_{i=2}^m I(\Omega_{1:(i-1)}; \Omega_i|\Delta_i) \quad (9)$$

$$\max_{i=2,m} I(\Omega_{1:(i-1)}; \Omega_{i:m}|\Delta_i) \leq \mathcal{J}(\mathcal{T}) \leq \sum_{i=2}^m I(\Omega_{1:(i-1)}; \Omega_{i:m}|\Delta_i) \quad (10)$$

The first is an identity, and the second is a Shannon inequality.

The identity (9) captures precisely the intuition that the information measure associated with a join tree $\mathcal{T}$ is equivalent to $m-1$ mutual information. This identity implies that $\mathcal{J}(\mathcal{T}) \geq 0$, because $I(\cdots) \geq 0$. But the expressions $I(\cdots)$ in (9) do not correspond to MVDs, because they do not include all variables Ω. The Shannon inequality (10) rectifies this, by showing that $\mathcal{J}(\mathcal{T})$ lies between the max and the sum of $m-1$ MVDs. Notice that the MVDs $\Delta_i \twoheadrightarrow \Omega_{1:(i-1)}|\Omega_{i:m}$, $i = 2, m$ are precisely the support of $\mathcal{T}$, $\text{MVD}(\mathcal{T})$, thus (10) generalizes Beeri's observation to approximate schemas. An immediate consequence of (10) is the following relationship between an acyclic schema $\mathbf{S}$ and its support.

Corollary 5.2. *Let* S *be an acyclic schema with join tree* $(\mathcal{T}, \chi)$. *Then: (1) if* $R \models_\varepsilon AJD(\mathrm{S})$ *then* $R \models_\varepsilon \mathrm{MVD}(\mathcal{T})$. *(2) If* $R \models_\varepsilon \mathrm{MVD}(\mathcal{T})$ *then* $R \models_{(m-1)\varepsilon} AJD(\mathrm{S})$. *In particular, (1) and (2) are equivalent if* $\varepsilon = 0$. *Here* $R \models_\varepsilon \mathrm{MVD}(\mathcal{T})$ *means* $R \models_\varepsilon \phi$, *forall* $\phi \in \mathrm{MVD}(\mathcal{T})$.

Proof. (of Theorem 5.1) Let $\mathcal{T}_i$ denote the subtree consisting of the nodes $u_1, \ldots, u_i$. We prove (9) by induction on m. Assume the identity holds for $m-1$. Compared to $\mathcal{T}_{m-1}$, the tree $\mathcal{T}_m$ has one extra node u_m and one extra edge $(\mathtt{parent}(u_m), u_m)$, hence by the definition of $\mathcal{J}$ in (6):

$$\begin{aligned}\mathcal{J}(\mathcal{T}_m) =& \mathcal{J}(\mathcal{T}_{m-1}) + H(\chi(u_m)) - H(\chi(u_m) \cap \chi(\mathtt{parent}(u_m)) \\ &+ H(\chi(\mathcal{T}_{m-1})) - H(\chi(\mathcal{T}_m)) \\ =& \mathcal{J}(\mathcal{T}_{m-1}) + H(\Omega_m) - H(\Delta_m) + H(\Omega_{1:(m-1)}) - H(\Omega_{1:m}) \\ =& \mathcal{J}(\mathcal{T}_{m-1}) + I(\Omega_{1:(m-1)}; \Omega_m | \Delta_m)\end{aligned}$$

The claim follows from the induction hypothesis on $\mathcal{J}(\mathcal{T}_{m-1})$.

We prove (10). The right inequality follows from the fact that $I(\Omega_{1:(i-1)}; \Omega_i | \Delta_i) \le I(\Omega_{1:(i-1)}; \Omega_{i:m} | \Delta_i)$ (which holds by Eq. (7)). For the left inequality, we make the following observation. If $\mathcal{T}$ is any join tree and $\mathcal{T}'$ is obtained by mergining two adjacent nodes $(u, v) \in \mathtt{edges}(\mathcal{T})$, then $\mathcal{J}(\mathcal{T}) \ge \mathcal{J}(\mathcal{T}')$. This is because $\mathcal{J}(\mathcal{T}) = \mathcal{J}(\mathcal{T}') + H(\chi(u)) + H(\chi(v)) - H(\chi(u) \cap \chi(v)) - H(\chi(u) \cup \chi(v)) = \mathcal{J}(\mathcal{T}') + I(\chi(u); \chi(v) | \chi(u) \cap \chi(v))$. To prove (10), we fix one edge $(\mathtt{parent}(u_i), u_i)$ and repeatedly merge all other edges, until we end with a tree $\mathcal{T}'$ with two bags, $\Omega_{1:(i-1)}$ and $\Omega_{i:m}$ respectively. Then $\mathcal{J}(\mathcal{T}) \ge \mathcal{J}(T') = I(\Omega_{1:(i-1)}; \Omega_{i:m} | \Delta_i)$. The claim follows from the fact that this holds for any $i = 2, m$. □

Example 5.3. We illustrate the first part of the theorem on the example in Fig. 2 and Example 3.4. Enumerating the nodes depth-first (ABD, ACD, AF, BDE), Eq. (9) and (10) become:

$$\begin{aligned}\mathcal{J}(\mathcal{T}) =& I(C; B|AD) + I(F; BCD|A) + I(ACF; E|BD) \\ \max(\cdots) \le \mathcal{J}(\mathcal{T}) \le& I(CF; BE|AD) + I(F; BCDE|A) + I(ACF; E|BD)\end{aligned}$$

5.2 Full MVDs

The number of candidate MVDs is very large: there are[2] $(3^n + 1)/2 - 2^n = O(3^n)$ standard MVD's $X \twoheadrightarrow Y|Z$, which is too large to consider for practical datasets. Here, and in the next section, we describe two techniques that allow us to restrict the search space. Consider a fixed key X. In the exact case, if any MVD $X \twoheadrightarrow \ldots$ holds in the data, then there exists a "best" one [4]. For example if both $X \twoheadrightarrow AB|C$ and $X \twoheadrightarrow A|BC$ hold exactly, then so does $X \twoheadrightarrow A|B|C$, and it suffices to discover only the latter. Unfortunately, this fails for approximate MVDs, as we explain here.

We say that $\phi = X \twoheadrightarrow A_1|\ldots|A_m$ *refines* $\psi = X \twoheadrightarrow B_1|\ldots|B_k$, denoted by $\phi \ge \psi$ if they both have the same key (i.e., $\mathrm{key}(\phi) = \mathrm{key}(\psi) = X$) and for every $A_i \in \mathrm{dep}(\phi)$ there exists $B_j \in \mathrm{dep}(\psi)$ such that $A_i \subseteq B_j$. For example, $X \twoheadrightarrow A|B|C$ refines $X \twoheadrightarrow AB|C$.

Proposition 5.2. *If* $\phi \ge \psi$ *then* $\mathcal{J}(\phi) \ge \mathcal{J}(\psi)$.

Proof. It suffices to consider the case when two dependents in ϕ are replaced by their union in ψ, e.g. $\phi = X \twoheadrightarrow A|B|\cdots$ and $\psi = X \twoheadrightarrow AB|\cdots$, since any refinement is a sequence of such steps. In that case, by inspecting Eq.(6) we observe $\mathcal{J}(\phi) = \mathcal{J}(\psi) + H(XA) + H(XB) - H(XAB) - H(X) = \mathcal{J}(\psi) + I(A; B|X) \ge \mathcal{J}(\psi)$ proving the claim. □

We say that an MVD ψ is *ε-full*, or simply *full*, if $R \models_\varepsilon \psi$ and, for all strict refinements $\phi > \psi$, $R \not\models_\varepsilon \phi$. We denote by $\mathrm{FullMVD}_\varepsilon(R, X)$ the set of all full ε-MVDs with key X. Thus, we only need to discover the sets $\mathrm{FullMVD}_\varepsilon(R, X)$, for all $X \subseteq \Omega$, because all other MVDs can be derived using Shannon inequalities.

Beeri proved that, in the exact case, $\mathrm{FullMVD}_0(R, X)$ has at most one element. We next present Lemma 5.4 that shows what happens in the approximate case, and allows us to derive Beeri's result as a special case. Given two MVDs $\phi = X \twoheadrightarrow A_1|\ldots|A_m$ and $\psi = X \twoheadrightarrow B_1|\ldots|B_k$, define their *join* as $\phi \vee \psi = X \twoheadrightarrow C_{11}|C_{12}|\cdots|C_{mk}$, where $C_{ij} = A_i \cap B_j$. Clearly, $\phi \vee \psi$ refines both ϕ and ψ, i.e. $\mathcal{J}(\phi \vee \psi) \ge \max(\mathcal{J}(\phi), \mathcal{J}(\psi))$. We prove a weak form of converse:

Lemma 5.4. *The following are Shannon inequalities:* $\mathcal{J}(\phi \vee \psi) \le \mathcal{J}(\phi) + m\mathcal{J}(\psi)$ *and* $\mathcal{J}(\phi \vee \psi) \le k\mathcal{J}(\phi) + \mathcal{J}(\psi)$.

By this result, $\mathcal{J}(\phi) = \mathcal{J}(\psi) = 0$ implies $\mathcal{J}(\phi \vee \psi) = 0$, which proves Beeri's theorem that $\mathrm{FullMVD}_\varepsilon(R, X)$ has at most one element, because if $\phi_1, \phi_2, \cdots$ are all MVD's with key X that hold exactly on R, then $\phi_1 \vee \phi_2 \vee \cdots$ refines all of them and holds too. This property was also used by Draeger [12] in his MVD discovery algorithm. When $\varepsilon > 0$ however, then this fails. For a very simple example, consider a relation with two tuples,

X	A	B	C
0	0	0	0
0	1	1	1

and fix $\varepsilon = 1$. Then $R \models_\varepsilon X \twoheadrightarrow AB|C, X \twoheadrightarrow AC|B, X \twoheadrightarrow BC|A$, but $\not\models_\varepsilon X \twoheadrightarrow A|B|C$; indeed, $H(\emptyset) = H(X) = 0$ and $H(W) = 1$ for all other sets W, and the reader can check $\mathcal{J}(X \twoheadrightarrow AB|C) = \mathcal{J}(X \twoheadrightarrow AC|B) = \mathcal{J}(X \twoheadrightarrow BC|A) = 1$ but $\mathcal{J}(X \twoheadrightarrow A|B|C) = 2$.

In summary, our algorithm discovers $\mathrm{FullMVD}_\varepsilon(R, X)$, for every X. Unlike the exact case, $\mathrm{FullMVD}_\varepsilon(R, X)$ may contain more than one MVD.

5.3 Minimal Separators

We now show that it is not necessary to discover the sets $\mathrm{FullMVD}_\varepsilon(R, X)$ for all subset of attributes $X \subset \Omega$, but only those where X is a *minimal separator*.

Definition 5.5. Fix a relation R and $\varepsilon \ge 0$. We say that a set X *separates* two variables $A, B \notin X$ if there exists an

[2] There are 3^n ways to partition Ω into three sets X, Y, Z. We rule out the 2^n partitions that have $Y = \emptyset$ and the 2^n partitions that have $Z = \emptyset$, and add back the 1 partition that has $Y = Z = \emptyset$, for a total of $3^n - 2^{n+1} + 1$. Finally, we divide by 2 since $X \twoheadrightarrow Y|Z$ and $X \twoheadrightarrow Z|Y$ are the same MVD.

ε-MVD $X \twoheadrightarrow Y_1|\cdots|Y_m$ that separates A, B, i.e. A, B occur in different sets Y_i, Y_j. We say X is a *minimal A, B-separator* if there is no $X_0 \subsetneq X$ that separates A, B.

For a pair $A, B \in \Omega$, we denote by $\textsc{MinSep}_\varepsilon(R, A, B)$ the set of minimal A, B separators in R, and for a minimal AB separator X we denote by $\textsc{FullMVD}_\varepsilon(R, X, A, B)$ the set of full MVDs with key X that separate A, B. Notice that:

$$\textsc{FullMVD}_\varepsilon(R, X) = \bigcup_{A,B \in \Omega \setminus X} \textsc{FullMVD}_\varepsilon(R, X, A, B).$$

Example 5.6. Let R be a relation over $\Omega = \{A, \ldots, E\}$. Suppose $R \models_\varepsilon CD \twoheadrightarrow A|BE$. By (8) we also have $R \models_\varepsilon CDE \twoheadrightarrow A|B$, which means that CDE cannot be a minimal separator for A, B. To check that CD is a minimal A, B-separator, we need to check that neither C nor D separates A, B

The main result in this section is that we only need to compute the full MVDs with minimal separators, denoted:

$$\mathcal{M}_\varepsilon \stackrel{\text{def}}{=} \bigcup_{A,B \in \Omega} \bigcup_{X \in \textsc{MinSep}_\varepsilon(R,A,B)} \textsc{FullMVD}_\varepsilon(R, X, A, B) \quad (11)$$

because, as we show, every ε-MVD can be derived from the set $\mathcal{M}_\varepsilon$ by a Shannon inequality.

Theorem 5.7. *Let $X \twoheadrightarrow Y|Z$ be an ε-MVD for R. Then there exist $\phi_1, \ldots, \phi_m \in \mathcal{M}_\varepsilon$, where $m = |Y| \cdot |Z|$, such that the following is a Shannon inequality: $I(Y; Z|X) \le \sum_i \mathcal{J}(\phi_i)$.*

In summary, our algorithm will iterate over pairs of attributes A, B, will compute $\textsc{MinSep}_\varepsilon(R, A, B)$, then, for each X in this set will compute $\textsc{FullMVD}_\varepsilon(R, X, A, B)$, and return their union, $\mathcal{M}_\varepsilon$; we describe it in the next section. We end this section with the proof of Theorem 5.7.

Proof. Let $Y = A_1 \ldots A_m$, and $Z = B_1 \ldots B_k$. By the chain rule (4) it holds that:

$$I(Y; Z|X) = \sum_{i=1}^{m} \sum_{j=1}^{k} I(A_i; B_j | X A_1 \ldots A_{i-1} B_1 \ldots B_{j-1})$$

It suffices to prove that, for each i, j, there exists an MVD $\phi \in \mathcal{M}_\varepsilon$ such that the following is a Shannon inequality:

$$I(A_i; B_j | X A_1 \cdots A_{i-1} B_1 \cdots B_{j-1}) \le \mathcal{J}(\phi)$$

Since $X \twoheadrightarrow Y|Z$ is a ε-MVD for the relation R, then X is an A_i, B_j separator. Let $S \subseteq X$ be any minimal A_i, B_j separator, thus $S \in \textsc{MinSep}_\varepsilon(R, A_i, B_j)$, and let $\phi = S \twoheadrightarrow U_1|\cdots|U_p$ be a full MVD in $\textsc{FullMVD}_\varepsilon(R, S, A_i, B_j) \subseteq \mathcal{M}_\varepsilon$ that separates A_i, B_j. Assume w.l.o.g. $A_i \in U_1$, $B_j \in U_2$, and let $\psi \stackrel{\text{def}}{=} S \twoheadrightarrow W|V$, where $W = U_1$, $V = U_2 U_3 \cdots U_p$. Thus, $\phi \ge \psi$, and therefore by Prop. 5.2 the following Shannon inequality holds: $\mathcal{J}(\phi) \ge \mathcal{J}(\psi)$. Write ψ as $\psi = S \twoheadrightarrow W_0 W_1 | V_0 V_1$, where $W_0 = W \cap (X A_1 \cdots A_i B_1 \cdots B_j)$, $W_1 = W - W_0$, and similarly $V_0 = V \cap (X A_1 \cdots A_i B_1 \cdots B_j)$, $V_1 = V - V_0$. By Prop. 5.1 (7) we have the following Shannon inequality $\mathcal{J}(\psi) = \mathcal{J}(S \twoheadrightarrow W_0 W_1 | V_0 V_1) \ge \mathcal{J}(S \twoheadrightarrow W_0 | V_0)$. Finally, we notice that the set $S W_0 V_0$ is the same as $X A_1 \cdots A_i B_1 \cdots B_j$ and that $A_i \in W_0$, $B_j \in V_0$, therefore by Prop. 5.1, (8), $\mathcal{J}(S \twoheadrightarrow W_0|V_0) \ge \mathcal{J}(X A_1 \cdots A_{i-1} B_1 \cdots B_{j-1} \twoheadrightarrow A_i | B_j)$, proving the claim. □

Algorithm MVDMiner(R, Ω, ε)

1: $\mathcal{M}_\varepsilon \leftarrow \emptyset$
2: **for all** pairs $A, B \in \Omega$ **do**
3: $\quad \textsc{MinSep}_\varepsilon(R, A, B) \leftarrow$ MineMinSeps(R, Ω, ε, (A, B))
4: $\quad$ **for all** $X \in \textsc{MinSep}_\varepsilon(R, A, B)$ **do**
5: $\quad\quad \mathcal{M}_\varepsilon \leftarrow \mathcal{M}_\varepsilon \cup$ getFullMVDs(X, ε, (A, B), ∞)
6: return $\mathcal{M}_\varepsilon$

Fig. 3 *Discover the set $\mathcal{M}_\varepsilon = \cup_{S \in \textsc{MinSep}_R} \textsc{FullMVD}_\varepsilon(S)$.*

6 DISCOVERING ε-MVDS

In this section we present the first phase of Maimon: the algorithm for the discovery of ε-MVDs in a relation R, called MVDMiner, and shown in Figure 3. As explained, the algorithm returns the set $\mathcal{M}_\varepsilon$, defined in Eq.(11); this set is used in the second phase of Maimon to compute ε-schemes.

MVDMiner iterates over all pairs of attributes $A, B \in \Omega$. It first computes the set $\textsc{MinSep}_\varepsilon(R, A, B)$ of minimal A, B-separators (line 3): we describe this step in Sec. 6.1. Then, for each $X \in \textsc{MinSep}_\varepsilon(R, A, B)$, it computes $\textsc{FullMVD}_\varepsilon(R, X, A, B)$ (line 5): we describe this step in Sec. 6.2. Finally, the algorithm returns their union, $\mathcal{M}_\varepsilon$. Both steps require access to an oracle $\text{getEntropy}_R(X)$ for computing the entropy $H(X)$, according to Eq. (5), where H is the entropy associated with the empirical distribution over R. We describe the implementation and optimization of $\text{getEntropy}_R(X)$ in Section 6.3.

6.1 Discovering the Minimal Separators

We describe here how we compute all minimal A, B-separators, $\textsc{MinSep}_\varepsilon(R, A, B)$ (line 3 of MVDMiner). One possible way to do this could be to iterate over sets X top down, because it enables pruning: if X is *not* an A, B-separator, then neither is any subset of X, by (8) in Prop. 5.1. This suggests a top-down algorithm, which starts from the largest set $X = \Omega \setminus \{A, B\}$, and checks if it is an A, B-separator. If not, then none exists. Otherwise, it exhaustively searches over subsets of X, from largest to smallest, returning the minimal (with regard to inclusion) sets that separate A, B. Such an exhaustive search will explore *all* separators, while we only want to find the *minimal* ones. Our approach takes advantage of the fact that

we need to find only the *minimal* separators, and builds on a result by Gunopulos et al. [20].

Let $\mathbf{C} = \{C_1, \ldots, C_m\}$ be a set of distinct subsets of Ω. A set $D \subset \Omega$ is a *transversal* of $\mathbf{C}$ if $D \cap C_i \neq \emptyset$ for every $C_i \in \mathbf{C}$. For a set $D \subseteq \Omega$, we denote by $\overline{D}$ the complement set $\Omega \backslash D$.

Theorem 6.1. *Let $\boldsymbol{C} = \{C_1, \ldots, C_n\}$ denote a set of minimal A, B separators in R. Then there exists a minimal A, B-separator $X \notin \boldsymbol{C}$ iff there exists a minimal (w.r.t inclusion) transversal D of $\boldsymbol{C}$ such that $\overline{D}$ is an A, B-separator.*

Proof. **only if.** Since D is a transversal of $\mathbf{C}$ then it holds that $\bigwedge_{i=1}^{n}(C_i \cap D \neq \emptyset)$ iff $\bigwedge_{i=1}^{n}(\overline{D} \not\supseteq C_i)$. Since $\overline{D}$ is an A, B separator, there exists some minimal separator $X \subseteq \overline{D}$. Assume, by contradiction, that $X \supseteq C_i$ for some $C_i \in \mathbf{C}$. Then $\overline{D} \supseteq X \supseteq C_i$, which is a contradiction.

if. Since X is a minimal A, B separator that is not in $\mathbf{C}$, then $\bigwedge_{i=1}^{n}(X \not\supseteq C_i)$, meaning that $\overline{X}$ is a transveral of $\mathbf{C}$. Then any minimal transversal $D \subseteq \overline{X}$ satisfies the claim. □

Algorithm MineMinSeps (Fig. 5) for discovering all minimal A, B separators, $\textsc{MinSep}_{\varepsilon}(R, A, B)$ is based on Theorem 6.1, and proceeds as follows:

(1) Initialize $\mathbf{C}$ with a minimal A, B-separator (Line 3-5).
(2) Iterate over all minimal transversals D of $\mathbf{C}$ (Line 8):
(3) If $\overline{D}$ separates A, B (Line 11), then:
   (a) Find any minimal A, B separator $X \subseteq \overline{D}$ (Line 12).
   (b) $\mathbf{C} \leftarrow \mathbf{C} \cup \{X\}$.

The function ReduceMinSep called in lines 4 and 12 takes a separator ($\Omega \backslash \{A, B\}$ or $\overline{D}$ respectively) and finds *any* subset that is a minimal separator; this is done greedily in ReduceMinSep (Fig. 4). The function getFullMVDs called in line 10 of MineMinSeps, and in line 4 of ReduceMinSep, takes as input an attribute set X, a pair of attributes A, B, and a threshold ε, and computes full ε-MVDs with key X that separate A, B; a parameter $K > 0$ is used to limit the number of full MVDs returned, and here we set $K = 1$ because we only check if one exists; in line 5 of the main algorithm (Fig. 3) we set $K = \infty$.

The only sets of attributes returned in MineMinSeps are minimal AB-separators returned by ReduceMinSep in lines 4 and 12. The proof of completeness (i.e., the algorithm returns all minimal AB-separators) follows techniques similar to those by Gunopulos et al. [20], and is given in the full version of the paper:

Theorem 6.2. *Algorithm* MineMinSeps *in Figure 5 enumerates all minimal A, B-separators in R.*

We now analyze the runtime between consecutive discoveries of minimal A, B-separators in MineMinSeps. We let Ω be a finite set of cardinality n, and let $\mathbf{C} \subseteq 2^{\Omega}$ be a finite set of sets. The problem of discovering all minimal

Algorithm ReduceMinSep(ε, X, (A,B))

```
1: Let p = X_1, ..., X_m be a predefined ordering of X.
2: S ← X
3: for all i = 1 to m do
4:    M_i ← getFullMVDs(S\{X_i}, ε, (A, B), 1)
5:    if M_i ≠ ∅ then
6:       S ← S\{X_i}
7: return S
```

Fig. 4 *Given a set $X \subset \Omega$, and a pair $(A, B) \in \Omega \backslash X$, find a subset $S \subseteq X$ s.t. S is a minimal A, B-separator in R.*

Algorithm MineMinSeps(R, Ω, ε, (A, B))

```
1: C ← ∅
2: X ← nil
3: if I(A; B|Ω\{A, B}) ≤ ε {by getEntropy_R} then
4:    X ← ReduceMinSep(ε, Ω\{A, B}, (A, B))
5:    C ← C ∪ {X}
6: else
7:    Return ∅
8: while (D ← nextMinTransversal(C)) ≠ nil do
9:    D̄ ← Ω\D
10:   φ ← getFullMVDs(D̄, ε, (A, B), 1)
11:   if φ ≠ ∅ then
12:      X ← ReduceMinSep(ε, D̄, (A, B))
13:      C ← C ∪ {X}
14: return C
```

Fig. 5 *Given a relation R with schema Ω, two attributes $A, B \in \Omega$, and a threshold ε enumerate all minimal A, B-separators in R.*

transversals of $\mathbf{C}$ is called the *hypergraph transversal problem* [23]. The theoretically best known algorithm for solving the hypergraph transversal problem is due to Fredman and Khachiyan [16] and has a quasi incremental-polynomial delay of $poly(n) + m^{O(\log^2 m)}$ where $m = |\mathbf{C}| + n$. Note the dependence on the size of the discovered minimal separators $|\mathbf{C}|$. We denote by $T_{minTrans}(n, \mathbf{C})$ the delay of the minimal transversal algorithm. However, not every minimal transversal D leads to the discovery of a minimal separator if $\overline{D}$ does not separate A and B (i.e., $\phi = \emptyset$ in line 11 of MineMinSeps). In the full version of this paper we show that the number of minimal transversals processed in lines 9-13 before a new

minimal separator is discovered (e.g., in line 12), or before the loop exists, is bounded by $n \cdot |\mathbf{C}|$. This allows us to formalize the delay between the discovery of minimal A, B-separators. We denote by $T(\text{getFullMVDs})$ the runtime of getFullMVDs, which we analyze in the next section.

Corollary 6.3. *Algorithm* MineAllMinseps *enumerates the minimal A, B-separators in R with a delay of $O(n \cdot |\boldsymbol{C}| \cdot T_{minTrans}(n, \boldsymbol{C}) \cdot T(\text{getFullMVDs}))$, where $n = |\Omega|$.*

6.2 Discovering the Full MVDs

Returning to our main algorithm, MVDMiner, we have shown how to compute $\text{MinSep}_\varepsilon(R, A, B)$, the set of minimal A, B separators in R. Next, for each minimal A, B separator $X \in \text{MinSep}_\varepsilon(R, A, B)$, we compute all full MVDs with key X that separate A and B, i.e. the set $\text{FullMVD}_\varepsilon(R, X, A, B)$; this is line 5 of MVDMiner. Recall that *full* means that the MVD cannot be further refined.

The algorithm getFullMVDs starts by checking the most refined MVD with key X, namely $\varphi = X \twoheadrightarrow Y_1 | \ldots | Y_n$ where $Y_1, \ldots, Y_n$ are all attributes not in X (including A, B). If $\mathcal{J}(\varphi) \leq \varepsilon$ then we are done. Otherwise, the algorithm considers all possible ways to *merge* two dependents, while keeping A and B in different dependents; i.e. it tries $X \twoheadrightarrow Y_1Y_2 | \ldots | Y_n, X \twoheadrightarrow Y_1Y_3 | Y_2 | \ldots | Y_n$, etc. We denote the MVD that results from merging dependents Y_i and Y_j in $\text{dep}(\varphi)$ by $\text{merge}_{ij}(\varphi)$. Since φ refines $\text{merge}_{ij}(\varphi)$ then, by Proposition 5.2, it holds that $\mathcal{J}(\text{merge}_{ij}(\varphi)) \leq \mathcal{J}(\varphi)$. This procedure for searching for a full ε-MVD can be viewed as a graph traversal algorithm where every node ϕ is an ε-MVD candidate with key X, dependents $Z_1, \ldots, Z_k$, and its neighbors $\text{Nbr}(\phi)$ are the ε-MVD candidates:

$$\text{Nbr}(\phi) \stackrel{\text{def}}{=} \{\text{merge}_{ij}(\phi) : Z_i, Z_j \in \text{dep}(\phi), A, B \notin Z_iZ_j\} \quad (12)$$

Clearly, if A, B were separated in ϕ, then they remain separated in every MVD in $\text{Nbr}(\phi)$. We present the algorithm as a depth-first traversal, which is how we implemented it. The pseudocode is presented in Figure 6.

6.2.1 An Optimization to getFullMVDs. In the worst case, Algorithm getFullMVDs will traverse the search space of possible ways to partition n attributes into $k \in \{2, \ldots, n-1\}$ sets, and there can be $O(\frac{k^n}{k!})$ such such partitions [3]. While, in general, this is unavoidable, we implemented an optimization, described in the complete version of this paper, that leads to a significant reduction in the search space.

[3]These are *Stirling numbers of the second kind*: https://en.wikipedia.org/wiki/Stirling_numbers_of_the_second_kind

Algorithm getFullMVDs(S, ε, (A, B), K)

```
1:  P ← ∅ {Output set}
2:  Q ← ∅ {Q is a stack}
3:  φ_0 = S ↠ X_1 | ... | X_n where X_i are singletons.
4:  Q.push(φ_0)
5:  while Q ≠ ∅     |P| < K do
6:    φ ← Q.pop()
7:    Compute J(φ) {using getEntropy_R}
8:    if J(φ) ≤ ε then
9:      P ← P ∪ {φ}
10:   else
11:     for all φ ∈ Nbr(φ) do
12:       Q.push(φ) {See (12)}
13: return P
```

Fig. 6 *Returns a set of at most K full MVDs with key S that approximately hold in R (w.r.t ε) in which A and B are in distinct components.*

6.3 Computing Entropies Efficiently

We describe the procedure $\text{getEntropy}_\text{R}$ for calculating the joint entropy of a set of attributes. The efficiency of this procedure is crucial to the performance of MVDMiner, which needs to repeatedly compute mutual information values $I(Y; Z|X)$, and each such computation requires four entropic values $H(XY)$, $H(XZ)$, $H(XYZ)$, and $H(X)$. Repeatedly computing values of the form $H(X_\alpha)$, for $\alpha \subseteq [n]$ requires multiple scans over the data that resides in external memory.

We build on ideas introduced in the PLI cache data structure [21, 27], and reduce the problem of computing $H(X_\alpha)$ to a main memory join-group-by query. For convenience, we repeat the formula for entropy (5):

$$H(X_\alpha) \stackrel{\text{def}}{=} \log N - \frac{1}{N} \sum_{x_\alpha \in \mathcal{D}_\alpha} |R(X_\alpha{=}x_\alpha)| \log |R(X_\alpha{=}x_\alpha)| \quad (13)$$

The algorithm uses two ideas: (1) if x_α is a singleton (i.,e., its frequency $|R(X_\alpha{=}x_\alpha)|{=}1$) then it can be ignored because its contribution to the total entropy in (13) is 0 (due to the logarithm), and (2) given two relations mapping the distinct values of attribute sets X_α, and X_β, respectively, to the tuple ids in the relation R that contain them, then we can derive this mapping for $X_\alpha \cup X_\beta$ by simply joining the two mappings on the tuple IDs. Ignoring singleton valuations makes these mappings highly compressed, enabling us to store them in main memory and perform the join using a main memory database system. We used the in-memory database *H2* [45]. We describe the details next. We let h denote a hash function. In our implementation we use the hash function provided by the database system. Alg. $\text{getEntropy}_\text{R}$ maintains

two sets of relations indexed by $\alpha \subseteq [n]$: $\text{CNT}_\alpha(\texttt{val}, \texttt{cnt})$ and $\text{TID}_\alpha(\texttt{val}, \texttt{tid})$ defined as:

$$\text{CNT}_\alpha = \{(h(x_\alpha), \texttt{cnt}) \mid \texttt{cnt} = |R(X_\alpha = x_\alpha)|, \texttt{cnt} > 1\}$$

$$\text{TID}_\alpha = \{(h(x_\alpha), t[\texttt{tid}]) \mid t \in R, t[X_\alpha] = x_\alpha, h(x_\alpha) \in \Pi_{\texttt{val}}(\text{CNT}_\alpha)\}$$

We compute $H(X_\alpha)$ by scanning table CNT_α. The algorithm starts by computing two sets of relations: (1) $\{\text{CNT}_{\{i\}}\}$ and (2) $\{\text{TID}_{\{i\}}\}$ for every $i \in [n]$. Assume that we have computed the relations CNT_α, CNT_β and TID_α, TID_β for some subsets $\alpha, \beta \subset [n]$ such that $\alpha \cap \beta = \emptyset$. We compute $\text{CNT}_{\alpha\cup\beta}$ as:

```
Select h(A.val, B.val) as val, count(*) as cnt
From TID_α A, TID_β B
Where A.tid = B.tid
Group By h(A.val, B.val) Having count(*) > 1
```

Next, we compute $\text{TID}_{\alpha\cup\beta}$ as:

```
Select h(A.val, B.val) as val, A.tid as tid
From TID_α A, TID_β B, CNT_{α∪β} Z
Where A.tid = B.tid and h(A.val, B.val) = Z.val
```

Pruning the singleton values makes this technique very effective, because as we move up the lattice from smaller α's to larger α's, many more tuples x_α are unique in the data, and the tables CNT_α and TID_α become smaller.

Example 6.4. Fig. 7 shows the tables generated for a 3-attribute relation R. Both types of relations only contain values corresponding to non-singleton valuations in R.

However, even with our compression, generating and storing all $2^n - 1$ tables CNT_α, and TID_α is intractable. Instead, we perform the following optimization. Fix a parameter L (in our implementation we chose $L = 10$), and partition the set Ω into $\lceil \frac{n}{L} \rceil$ disjoint subsets $\Omega_1, \Omega_2, \ldots$ each of size at most L. For each i, compute the tables TID_α and CNT_α for all subsets $\alpha \subseteq \Omega_i$; thus the total number of tables precomputed is $2\lceil \frac{n}{L} \rceil \cdot 2^L$. In order to compute $H(X_\alpha)$, we express $\alpha = (\alpha \cap \Omega_1) \cup (\alpha \cap \Omega_2) \cup \ldots$, where each union is treated as explained above for $\alpha \cup \beta$.

7 ENUMERATING ACYCLIC SCHEMAS

In this section we present the second phase of Maimon: given the set $\mathcal{M}_\varepsilon$ of full ε-MVDs (Eq. (11)), generate acyclic ε-schemes. The algorithm ASMiner is shown in Fig. 8. It searches for subsets of MVDs $\mathcal{Q} \subseteq \mathcal{M}_\varepsilon$, and reconstructs a schema from that set. The key to the algorithm's efficiency is our new definition of compatibility:

Definition 7.1. Let $\phi_1 = X \twoheadrightarrow A_1|\ldots|A_m$ and $\phi_2 = Y \twoheadrightarrow B_1|\ldots|B_k$ be two ε-MVDs. We say that ϕ_1 and ϕ_2 are *compatible* if there exist an $i \in \{1, \ldots, m\}$, and $j \in \{1, \ldots, k\}$ such that:

(1) $Y \subseteq XA_i$, and $X \subseteq YB_j$. In this case we say that the two MVDs are *split-free* [6, 15, 19, 28].

R

tid	A	B	C
t_1	a_1	b_2	c_3
t_2	a_2	b_1	c_1
t_3	a_2	b_2	c_2
t_4	a_3	b_3	c_3
t_5	a_3	b_3	c_4

CNT_{AB}

val	CNT
$h(a_3, b_3)$	2

TID_{AB}

val	tid
$h(a_3, b_3)$	t_4
$h(a_3, b_3)$	t_5

CNT_A

val	CNT
a_2	2
a_3	2

CNT_B

val	CNT
b_2	2
b_3	2

CNT_C

val	CNT
c_3	2

TID_A

val	tid
a_2	t_2
a_2	t_3
a_3	t_4
a_3	t_5

TID_B

val	tid
b_2	t_1
b_2	t_3
b_3	t_4
b_3	t_5

TID_C

val	tid
c_3	t_4
c_3	t_4

Fig. 7 getEntropy_R *example.*

Algorithm ASMiner($\mathcal{M}_\varepsilon$)

1: schemes = $\emptyset$
2: Construct the graph $G = \{(\phi, \psi) \mid \phi, \psi \in \mathcal{M}_\varepsilon, \phi \sharp \psi\}$
3: **for all** $\mathcal{Q} \in$ MaxIndependentSet(G) **do**
4: schemes $\leftarrow$ schemes $\cup$ {BuildAcyclicSchema($\mathcal{Q}$)}
5: return schemes

Fig. 8 *Generate Acyclic Schemas from* $\mathcal{M}_\varepsilon$.

Algorithm BuildAcyclicSchema($\mathcal{Q}$)

1: $\mathbf{S} \leftarrow \{\Omega\}$
2: Sort $\mathcal{Q}$ by ascending order of key cardinality {e.g., $X \twoheadrightarrow A|B$ before $XY \twoheadrightarrow C|D$}
3: **for all** $\phi \in \mathcal{Q}$ **do**
4: Let $\phi = X \twoheadrightarrow C_1|\ldots|C_m$
5: Let $\Omega_i \in \mathbf{S}$ s.t. $X \subseteq \Omega_i$
6: $\mathbf{D}_\phi \leftarrow \{C_j X \cap \Omega_i \mid j \in [i, m]\} \setminus \{X\}$
7: **if** $|\mathbf{D}_\phi| \geq 2$ **then**
8: Replace $\Omega_i \in \mathbf{S}$ with $\mathbf{D}_\phi$ {ϕ is *non-redundant*}
9: return $\mathbf{S}$

Fig. 9 *Gets a set* $\mathcal{Q}$ *of pairwise compatible MVDs, and returns an acyclic schema.*

(2) There exist two distinct indexes $j_1, j_2 \in \{1, \ldots k\}$ such that $XA_i \cap B_{j_1} \neq \emptyset$, and $XA_i \cap B_{j_2} \neq \emptyset$. Likewise, there exist two distinct indexes $i_1, i_2 \in \{1, \ldots m\}$ such that $YB_j \cap A_{i_1} \neq \emptyset$, and $YB_j \cap A_{i_2} \neq \emptyset$.

We write $\phi_1 \sharp \phi_2$ to denote the fact that ϕ_1, ϕ_2 are *in*compatible.

We say that a set $\mathcal{Q}$ of ε-MVDs is *pairwise compatible* if every pair of ε-MVDs in $\mathcal{Q}$ is compatible. Recall that every join tree $\mathcal{T}$ with m nodes defines a set of $m-1$ MVDs called its *support* and denoted by $\text{MVD}(\mathcal{T})$.

Theorem 7.2. *Let* S *be an acyclic schema with join tree* $(\mathcal{T}, \chi)$*. Then the set* $\text{MVD}(\mathcal{T})$ *is pairwise compatible.*

Thus, it suffices to iterate over sets of pairwise compatible ε-MVDs. Specifically, our algorithm enumerates the *maximal* sets of pairwise compatible ε-MVDs, and for this task we use a graph algorithm from the literature. Define the graph $G(\mathcal{M}_\varepsilon, E)$ as follows:

$$E = \{(\phi_1, \phi_2) : \phi_1, \phi_2 \in \mathcal{M}_\varepsilon \text{ and } \phi_1 \sharp \phi_2\} \tag{14}$$

By this definition every maximal independent set in G corresponds to a maximal set of pairwise compatible ε-MVDs. We apply the following result.

Theorem 7.3. ([11, 22]) *Let* $G(V, E)$ *be a graph. The maximal independent sets of* G *can be enumerated such that the delay between consecutive outputs is in* $O(|V|^3)$.

In summary, algorithm ASMiner in Fig. 8 enumerates all maximal independent sets $\mathcal{Q}$, then for each of them constructs one acyclic schema S, by calling BuildAcyclicSchema shown in Fig. 9, and described next.

Algorithm BuildAcyclicSchema starts with a schema that contains a single relation with all attributes (i.e., $\mathbf{S} = \{\Omega\}$). It then builds the acyclic schema for R by repeatedly using an ε-MVD from $\mathcal{Q}$ to decompose one of the relations in S. The MVDs are processed in ascending order of the cardinality of their keys. Therefore, when an MVD $S \twoheadrightarrow C_1 | \ldots | C_m$ is processed, then we know that S is contained in exactly one of the relations in S (e.g., otherwise, S must be contained in a key of a previously processed ε-MVD). The algorithm then applies this ε-MVD to the single relation that contains it, and continues until all ε-MVDs in $\mathcal{Q}$ have been processed. An MVD is said to be *redundant* [18] if it does not split the single relation that contains it (i.e., condition of line 7 does not hold). Redundant MVDs are simply ignored in BuildAcyclicSchema.

Theorem 7.4. *Algorithm* BuildAcyclicSchema *generates an acyclic schema* S *with join tree* $(\mathcal{T}, \chi)$ *such that* $\text{MVD}(\mathcal{T}) \subseteq \mathcal{Q}$*. If* $\mathcal{Q}$ *is a non-redundant set of* ε*-MVDs then* $\text{MVD}(\mathcal{T}) = \mathcal{Q}$*. The algorithm runs in time* $O(n^3)$.

The novel insight of our algorithm is the characterization of (in)compatibility in Definition 7.1, which depends only on the pairwise relationship between the MVDs, and therefore enables the reduction to enumerating maximal independent sets. Previous characterizations [6, 15, 19, 28] are for entire sets of MVDs, and are not pairwise. Algorithms for constructing a (single) acyclic schema from data dependencies have been developed by Bernstein [7] where the input is a set of functional dependencies, and by Beeri et al. and Lien whose algorithms work by combining *conflict-free* MVDs [6, 32].

Dataset			Full MVDs threshold=0.0	
Dataset	Cols.	Rows	Runtime [sec]	Full MVDs
Ditag Feature	13	3960124	TL	NA
Four Square (Spots)	15	973516	17017	105
Image	12	777676	3747	151
FD_Reduced_30	30	250000	8024	21
FD_Reduced_15	15	250000	1006	21
Census	42	199524	TL	NA
SG_Bioentry	7	184292	101	3
Atom Sites	26	160000	TL	242
Classification	12	70859	1327	27
Adult	15	32561	1083	58
Entity Source	33	26139	14155	153
Reflns	27	24769	TL	543
Letter	17	20000	605	44
School Results	27	14384	7202	2394
Voter State	45	10000	TL	262
Abalone	9	4177	602	36
Breast-Cancer	11	699	5	30
Hepatitis	20	155	479	2953
Echocardiogram	13	132	6	104
Bridges	13	108	3.8	60

Table 2: Datasets. We show runtimes (in seconds) for mining full MVDs with threshold 0.0, and time limit of 5 hours.

8 EVALUATION

In this section we conduct an experimental evaluation of Maimon. We start with an end-to-end evaluation of its usefulness in Section 8.1, then evaluate the accuracy of the approximate schemas in terms of the relationship between the J-measure and number of spurious tuples in Section 8.2. Next, we evaluate the efficiency and scalability of Maimon, measuring the time to find the minimal separators in Section 8.3. Finally, we report the rate of enumeration, and some quality metrics of the generated acyclic schemes in Section 8.4.

We used 20 real-world datsets [44] that are part of the Metanome data profiling project [34], shown in Table 2 (we discuss the runtimes in Sec. 8.3). Maimon was implemented in Java 1.8 and all experiments are single threaded and conducted on a 64bit Linux machine with 120 CPUs and 1 TB of memory, running Ubuntu 5.4.0.

8.1 A Use Case: Nursery

To evaluate the usefulness of Maimon we applied it to the Nursery dataset[4], a training data for classifying and ranking applications for nursery schools. The dataset contains

[4]https://archive.ics.uci.edu/ml/datasets/nursery

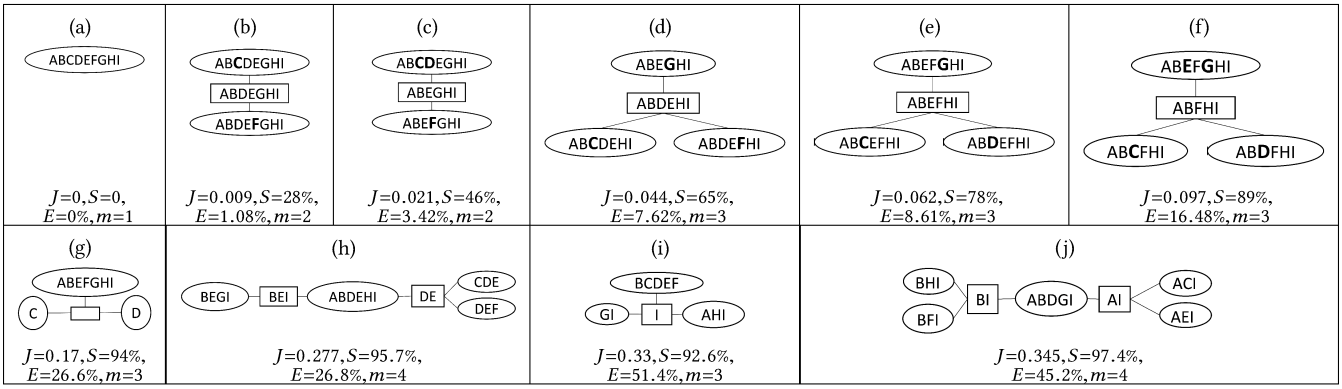

Fig. 10 *The Nursery use case, showing the 10 pareto optimal schemes (out of 415). We encode the 9 attributes as $A, B, \cdots, I$ (top). The data does not admit a exact decomposition (a), but we obtain increasingly better schemes (b)-(j) as we increase the J-measure, with increased space savings S, at the cost of increased rate of spurious tuples E; for example, for $J = 0.277$ the data decomposes into 4 relations, $S = 95.7\%$ (see text for the explanation of why it is so high) and $E = 26.8\%$.*

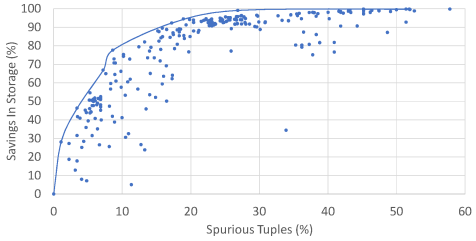

Fig. 11 *All 415 schemes discovered for Nursery. The plot shows the savings S v.s. the spurious tuples E. The line connects the ten pareto-optimal schemes further detailed in Fig. 10. .*

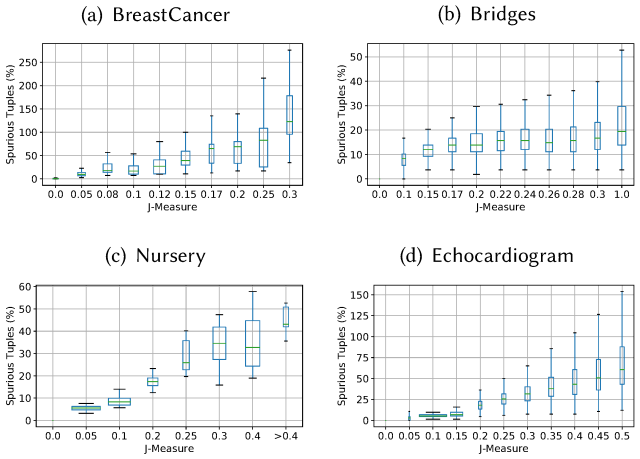

Fig. 12 *Spurious Tuples (%) vs. J-measure (see Sec. 8.2).*

eight attributes describing occupational, financial, social and health conditions of the family, and a classification attribute that indicates the priority of the application; we renamed the attributes $A \ldots I$ for brevity. The data has 12960 tuples and a total of $12960 * 9 = 116640$ cells. By increasing the threshold J from 0 to 0.5, we found 415 acyclic schemes (Fig 11), and show ten of them in detail in Fig. 10. As one can see in Fig. 10(a), when $J = 0$, no exact decomposition is possible. As we increase J, however, we find better and better schemas in Fig. 10 (b)-(j), in the sense that it decomposes into more relations, each with fewer attributes. For example, the schema in (h) ($J = 0.277$) has 4 relations, *BEGI*, *ABDEHI*, *CDE*, *DEF*. For each scheme we report the percentage cell savings, S, and the percentage of spurious tuples, E. There is a good trade-off between space savings and error: several schemes have under 10% spurious tuples yet achieve over 80% space saving. The space savings are very high (e.g. over 90%), because the Nursery data is dense: the attribute domains have sizes 3, 5, 4, 4, 3, 2, 3, 3, 5. For example, the extreme schema where each attribute is a separate relation (not shown in the Figure) has $3 + 5 + 4 + 4 + 3 + 2 + 3 + 3 + 5 = 32$ cells and a savings of $(116640 - 32)/116640$ i.e. $S = 99.9725\%$; however, its fraction of spurious tuples is $(3*5*4*4*3*2*3*3*5-12960)/12960 = 4$, i.e. $E = 400\%$. Fig. 11 shows the values S, E for all 415 schemes. Users are likely to select the pareto optimal schemes, i.e. whose S, E values are not dominated by any other schemes: the ten pareto optimal schemes in this graph are connected by a line, and are precisely those we have selected to show in detail in Fig. 10. In addition to savings S and spurious tuples E, applications are likely to define their own domain specific quality measure for choosing the optimal schema.

8.2 Accuracy

Next, we analyzed the relationship between the J-measure of the acyclic schemes, and the percentage of spurious tuples. There is no tight theoretical connection between these two measures, except that J=0 iff there are no spurious tuples, hence the need for an empirical evaluation. The results are

presented in Figure 12. We generated all acyclic schemes with a threshold $\varepsilon \in [0, 0.5]$, partitioned the schemes into buckets according to their J-measure, and report the quantiles of the number of spurious tuples in each bucket. The experiments confirm a consistent relationship between the J-measure and the percentage of spurious tuples. Assuming we want to have no more than 20% spurious tuples, then we can increase J up to 0.1−0.3, depending on the dataset. The width of the boxes represent the number of acyclic schemes in that bucket. In general, as J increases, the number of acyclic schemes will eventually decrease: this is particularly visible in Fig. 12 (d). The explanation lies in the fact that larger J's reduce the size (and, hence, the number) of minimum separators. If we allowed J to increase further, eventually we find a single schema, where each attribute is a separate relation, and where the sole minimal separator is the empty set.

8.3 Scalability

Next, we evaluated the scalability of Maimon. We started by computing all exact MVDs ($\varepsilon = 0$) on all 20 datasets and report the runtimes in Table 2. On five of the datasets, our system timed out after 5h: for Atom Sites, REFLNS, and Voter State, it did report a large number of full MVDs, while for DITAG Feature and Census it did not find any within this limit, but it terminated on subsets, as we report below.

The discovery of acyclic schemes has three parts: computing all minimal separators (Sec. 6.1), discovering all full MVDs (Sec. 6.2), and enumerating the acyclic schemes (Sec. 7). We found that the first step by far dominates the total runtime, and we report it here; we report the other two runtimes in the technical report. We report here the time to compute all minimal separators as a function of #rows, and of #columns.

8.3.1 Row Scalability. We evaluated the algorithm over three large datsets: Image, foursquare, and Ditag Feature. We included all columns, and a subset of 10% to 100% of the tuples. The results are in Figure 13. In general, we found that the runtime increases mostly linearly with the size of the data even when the number of minimal separators is mostly constant, e.g. for Image and Ditag Feature.

8.3.2 Column Scalability. Next, we varied the number of columns. Here we kept all rows of the datasets, and included between 10% to 100% of the columns. The results are presented in Figure 14. We let the algorithm run for 5 hours and measured the resulting number of minimal separators. For example, in the Voter State dataset with 32 columns Maimon discovered 682, 306 and 242 minimal separators for thresholds 0,0.01, and 0.1 respectively. We found that the runtime is affected both by the number of attributes, and, quite significantly, by the number of minimal separators. This is explained by considering Corollary 6.3 that analyzes the *delay* between the output of minimal separators. First, we note that the delay depends exponentially on the number of attributes (via getFullMVDs, see Sec. 6.2.1) which explains why the delay significantly increases with the number of attributes, leading to an overall reduction in the number of minimal separators returned. Second, it depends on the number of minimal separators generated up to that point, which explains the high runtime when the data contains a large number of minimal separators.

8.4 Quality

We conducted an empirical evaluation of the quality of the schemes generated by Maimon, and report the results in Figure 15. Per threshold, we ran the enumeration algorithm for half an hour and measured the number of schemes generated (i.e., `#schemes`), and the following quality measures, for which we report on their aggregate values.

(1) The number of relations in any scheme S generated, denoted `#relations`(S).
(2) The *width* attained by any generated scheme, where width refers to the largest number of attributes in any relation of S. Formally[5], $\texttt{width}(S) \stackrel{\text{def}}{=} \max_{i\in[1,m]} |\Omega_i|$.
(3) The *intersection width* attained by any scheme generated, where intersection width refers to the largest size of any separator of S: $\texttt{intWidth}(S) \stackrel{\text{def}}{=} \max_{i,j\in[1,m]} |\Omega_i \cap \Omega_j|$.

In Figure 15 we increased the threshold ε, and report for each threshold the maximum `#relations`(S), and the minimum `width`(S), `intWidth`(S) for all schemas at that threshold. In general, we observed that, as we increase the threshold, the system can find more interesting schemes. For example, for Image and Abalone, `width` (blue bar) decreases, which means that the number of attributes in the widest relation decreases. For Adult and BreastCancer the number of relations (`#relations` – gray bar) increases.

9 CONCLUSIONS

We present Maimon, the first system for the discovery of approximate acyclic schemes and approximate MVDs from data. To define "approximate", we used concepts from information theory, where each MVD or acyclic schema is defined by an expression over entropic terms; when the expression is 0, then the MVD or acyclic schema holds exactly. We then presented the two main algorithms in Maimon, mining all full ε-MVDs with minimal separators, and discovering acyclic schemes from a set of ε-MVDs. Both algorithms improve over prior work in the literature. We conducted an experimental evaluation of Maimon on over 20 real-world data sets.

Our approach of using information theory to define approximate data dependencies differs from the previous definitions that rely mostly on counting the number of offending

[5] `width`(S) is precisely the treewidth plus one.

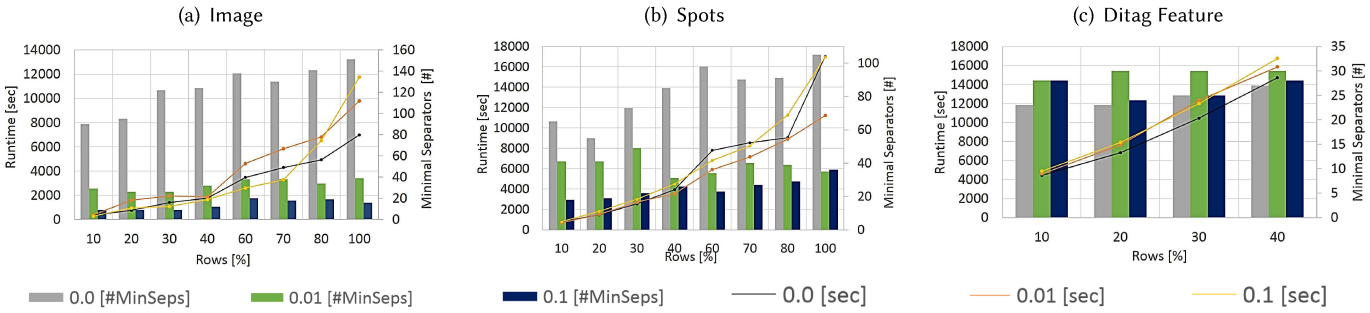

Fig. 13 *Row scalability experiments, for $\varepsilon \in \{0.0, 0.01, 0.1\}$ (Sec 8.3.1).*

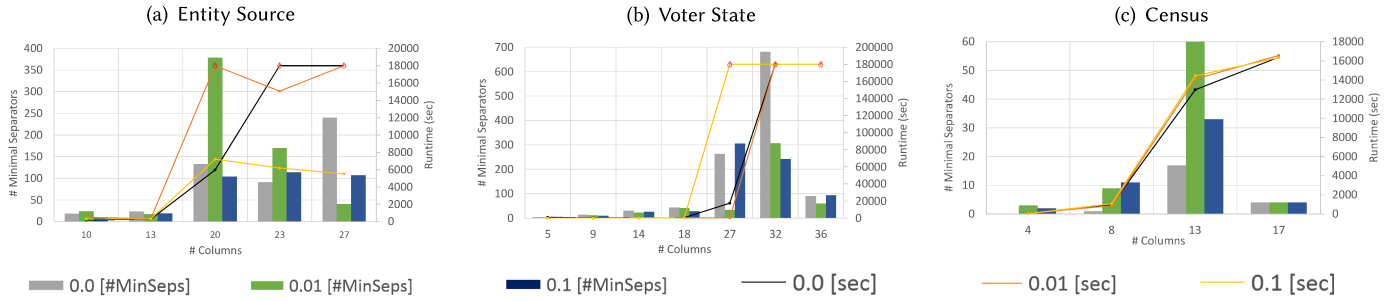

Fig. 14 *Column scalability experimentsfor $\varepsilon \in \{0, 0.01, 0.1\}$ (Sec 8.3.1). We timed out at five hours (red clock).*

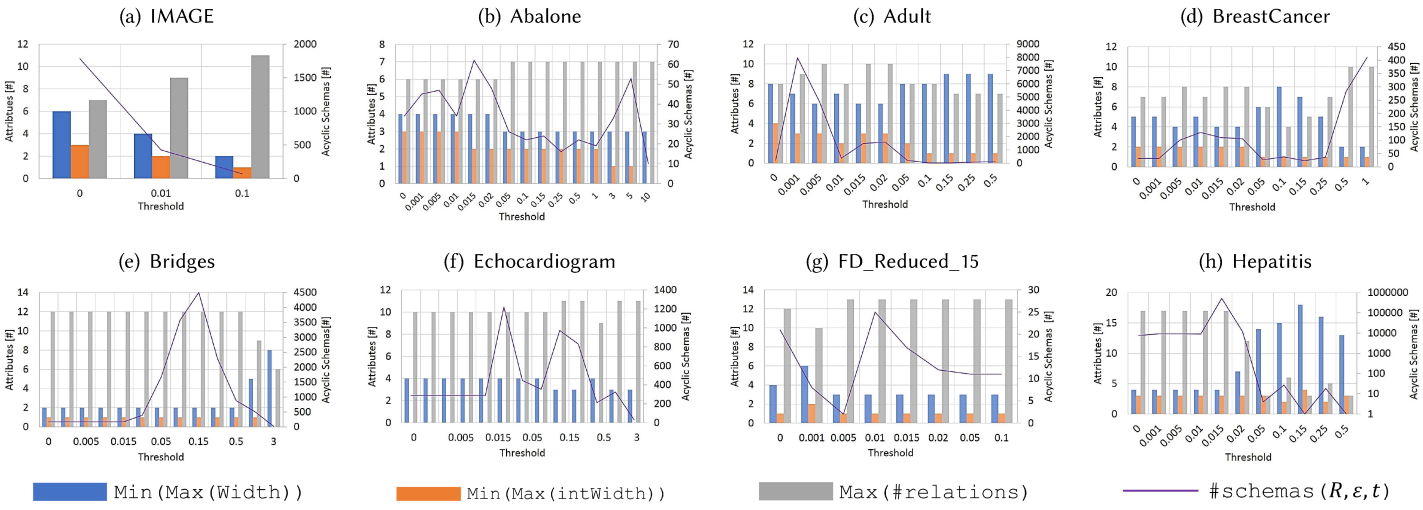

Fig. 15 *Quality of approximate schemas (Sec. 8.4)*

tuples. On one hand, our definitions provide us with more powerful mathematical tools, on the other hand the connection to the actual data quality is less intuitive. We leave it up to future work to explore the connection between information theory and data quality.

Depending on the dataset, Maimon generates hundreds and even thousands of acyclic ε-schemas in as little as 30 minutes. As part of future work we intend to investigate acyclic schema generation in *ranked order*. The categories to rank on may be the extent of decomposition (e.g., `width` of the schema), or other measures indicative of how well the schema meets the requirements of the application.

Acknowledgements. This work was supported by NSF grants III-1614738 and IIS-1907997.

REFERENCES

[1] Ziawasch Abedjan, Lukasz Golab, Felix Naumann, and Thorsten Papenbrock. Data profiling. *Synthesis Lectures on Data Management*, 10(4):1–154, 2018.

[2] Catriel Beeri. On the menbership problem for functional and multivalued dependencies in relational databases. *ACM Trans. Database Syst.*, 5(3):241–259, September 1980.

[3] Catriel Beeri and Philip A. Bernstein. Computational problems related to the design of normal form relational schemas. *ACM Trans. Database Syst.*, 4(1):30–59, March 1979.

[4] Catriel Beeri, Ronald Fagin, and John H. Howard. A complete axiomatization for functional and multivalued dependencies in database relations. In *Proceedings of the 1977 ACM SIGMOD International Conference on Management of Data, Toronto, Canada, August 3-5, 1977.*, pages 47–61, 1977.

[5] Catriel Beeri, Ronald Fagin, David Maier, Alberto O. Mendelzon, Jeffrey D. Ullman, and Mihalis Yannakakis. Properties of acyclic database schemes. In *Proceedings of the 13th Annual ACM Symposium on Theory of Computing, May 11-13, 1981, Milwaukee, Wisconsin, USA*, pages 355–362, 1981.

[6] Catriel Beeri, Ronald Fagin, David Maier, and Mihalis Yannakakis. On the desirability of acyclic database schemes. *J. ACM*, 30(3):479–513, July 1983.

[7] Philip A. Bernstein. Synthesizing third normal form relations from functional dependencies. *ACM Trans. Database Syst.*, 1(4):277–298, 1976.

[8] Tobias Bleifuß, Susanne Bülow, Johannes Frohnhofen, Julian Risch, Georg Wiese, Sebastian Kruse, Thorsten Papenbrock, and Felix Naumann. Approximate discovery of functional dependencies for large datasets. In *Proceedings of the 25th ACM International Conference on Information and Knowledge Management, CIKM 2016, Indianapolis, IN, USA, October 24-28, 2016*, pages 1803–1812, 2016.

[9] Nofar Carmeli, Batya Kenig, and Benny Kimelfeld. Efficiently enumerating minimal triangulations. In *Proceedings of the 36th ACM SIGMOD-SIGACT-SIGAI Symposium on Principles of Database Systems, PODS 2017, Chicago, IL, USA, May 14-19, 2017*, pages 273–287, 2017.

[10] E. F. Codd. Further normalization of the data base relational model. *IBM Research Report, San Jose, California*, RJ909, 1971.

[11] Sara Cohen, Benny Kimelfeld, and Yehoshua Sagiv. Generating all maximal induced subgraphs for hereditary and connected-hereditary graph properties. *J. Comput. Syst. Sci.*, 74(7):1147–1159, 2008.

[12] Tim Draeger. Multivalued dependency discovery, 2016. Master's Thesis, Hasso-Plattner-Institute, Potsdam.

[13] Ronald Fagin. Multivalued dependencies and a new normal form for relational databases. *ACM Trans. Database Syst.*, 2(3):262–278, September 1977.

[14] Ronald Fagin. Horn clauses and database dependencies. *J. ACM*, 29(4):952–985, 1982.

[15] Ronald Fagin, Alberto O. Mendelzon, and Jeffrey D. Ullman. A simplified universal relation assumption and its properties. *ACM Trans. Database Syst.*, 7(3):343–360, 1982.

[16] Michael L. Fredman and Leonid Khachiyan. On the complexity of dualization of monotone disjunctive normal forms. *Journal of Algorithms*, 21(3):618 – 628, 1996.

[17] Dan Geiger and Judea Pearl. Logical and algorithmic properties of conditional independence and graphical models. *The Annals of Statistics*, 21(4):2001–2021, 1993.

[18] Nathan Goodman and Y. C. Tay. A characterization of multivalued dependencies equivalent to a join dependency. *Inf. Process. Lett.*, 18(5):261–266, 1984.

[19] Dirk Van Gucht. Interaction-free multivalued dependency sets. *Theor. Comput. Sci.*, 62(1-2):221–233, 1988.

[20] Dimitrios Gunopulos, Roni Khardon, Heikki Mannila, Sanjeev Saluja, Hannu Toivonen, and Ram Sewak Sharm. Discovering all most specific sentences. *ACM Trans. Database Syst.*, 28(2):140–174, 2003.

[21] Ykä Huhtala, Juha Kärkkäinen, Pasi Porkka, and Hannu Toivonen. TANE: an efficient algorithm for discovering functional and approximate dependencies. *Comput. J.*, 42(2):100–111, 1999.

[22] David S. Johnson, Christos H. Papadimitriou, and Mihalis Yannakakis. On generating all maximal independent sets. *Inf. Process. Lett.*, 27(3):119–123, 1988.

[23] Leonid Khachiyan, Endre Boros, Khaled Elbassioni, and Vladimir Gurvich. An efficient implementation of a quasi-polynomial algorithm for generating hypergraph transversals and its application in joint generation. *Discrete Applied Mathematics*, 154(16):2350 – 2372, 2006. Discrete Algorithms and Optimization, in Honor of Professor Toshihide Ibaraki at His Retirement from Kyoto University.

[24] Mahmoud Abo Khamis, Hung Q. Ngo, XuanLong Nguyen, Dan Olteanu, and Maximilian Schleich. AC/DC: in-database learning thunderstruck. In *Proceedings of the Second Workshop on Data Management for End-To-End Machine Learning, DEEM@SIGMOD 2018, Houston, TX, USA, June 15, 2018*, pages 8:1–8:10, 2018.

[25] Mahmoud Abo Khamis, Hung Q. Ngo, and Atri Rudra. FAQ: questions asked frequently. In *Proceedings of the 35th ACM SIGMOD-SIGACT-SIGAI Symposium on Principles of Database Systems, PODS 2016, San Francisco, CA, USA, June 26 - July 01, 2016*, pages 13–28, 2016.

[26] Jyrki Kivinen and Heikki Mannila. Approximate inference of functional dependencies from relations. *Theor. Comput. Sci.*, 149(1):129–149, 1995.

[27] Sebastian Kruse and Felix Naumann. Efficient discovery of approximate dependencies. *PVLDB*, 11(7):759–772, 2018.

[28] V. S. Lakshmanan. Split-freedom and mvd-intersection: A new characterization of multivalued dependencies having conflict-free covers. *Theor. Comput. Sci.*, 62(1-2):105–122, 1988.

[29] Tony T. Lee. An information-theoretic analysis of relational databases - part I: data dependencies and information metric. *IEEE Trans. Software Eng.*, 13(10):1049–1061, 1987.

[30] Tony T. Lee. An information-theoretic analysis of relational databases - part II: information structures of database schemas. *IEEE Trans. Software Eng.*, 13(10):1061–1072, 1987.

[31] Mark Levene and George Loizou. Why is the snowflake schema a good data warehouse design? *Inf. Syst.*, 28(3):225–240, 2003.

[32] Y. Edmund Lien. Hierarchical schemata for relational databases. *ACM Trans. Database Syst.*, 6(1):48–69, March 1981.

[33] Jixue Liu, Jiuyong Li, Chengfei Liu, and Yongfeng Chen. Discover dependencies from data - A review. *IEEE Trans. Knowl. Data Eng.*, 24(2):251–264, 2012.

[34] Thorsten Papenbrock, Tanja Bergmann, Moritz Finke, Jakob Zwiener, and Felix Naumann. Data profiling with metanome. *Proc. VLDB Endow.*, 8(12):1860–1863, August 2015.

[35] Thorsten Papenbrock and Felix Naumann. A hybrid approach to functional dependency discovery. In *Proceedings of the 2016 International Conference on Management of Data, SIGMOD Conference 2016, San Francisco, CA, USA, June 26 - July 01, 2016*, pages 821–833, 2016.

[36] Babak Salimi, Bill Howe, and Dan Suciu. Data management for causal algorithmic fairness. *IEEE Data Engineering Bulletin*, vol. 42, no. 3, 2019.

[37] Babak Salimi, Luke Rodriguez, Bill Howe, and Dan Suciu. Interventional fairness: Causal database repair for algorithmic fairness. In *Proceedings of the 2019 International Conference on Management of Data, SIGMOD Conference 2019, Amsterdam, The Netherlands, June 30 - July 5, 2019*, pages 793–810, 2019.

[38] Iztok Savnik and Peter A. Flach. Discovery of multivalued dependencies from relations. *Intell. Data Anal.*, 4(3-4):195–211, 2000.

[39] Maximilian Schleich, Dan Olteanu, and Radu Ciucanu. Learning linear regression models over factorized joins. In *Proceedings of the 2016 International Conference on Management of Data, SIGMOD Conference 2016, San Francisco, CA, USA, June 26 - July 01, 2016*, pages 3–18, 2016.

[40] Maximilian Schleich, Dan Olteanu, Mahmoud Abo Khamis, Hung Q. Ngo, and XuanLong Nguyen. A layered aggregate engine for analytics workloads. In *Proceedings of the 2019 International Conference on Management of Data, SIGMOD Conference 2019, Amsterdam, The Netherlands, June 30 - July 5, 2019.*, pages 1642–1659, 2019.

[41] S. K. Michael Wong, Cory J. Butz, and Dan Wu. On the implication problem for probabilistic conditional independency. *IEEE Trans. Systems, Man, and Cybernetics, Part A*, 30(6):785–805, 2000.

[42] Catharine M. Wyss, Chris Giannella, and Edward L. Robertson. Fastfds: A heuristic-driven, depth-first algorithm for mining functional dependencies from relation instances - extended abstract. In *Data Warehousing and Knowledge Discovery, Third International Conference, DaWaK 2001, Munich, Germany, September 5-7, 2001, Proceedings*, pages 101–110, 2001.

[43] Mihalis Yannakakis. Algorithms for acyclic database schemes. In *Proceedings of the Seventh International Conference on Very Large Data Bases - Volume 7*, VLDB '81, pages 82–94. VLDB Endowment, 1981.

[44] Datasets of the metanome data profiling project. https://hpi.de/naumann/projects/repeatability/data-profiling/fds.html#c168191.

[45] *h*2 main memory database. https://www.h2database.com/html/main.html.