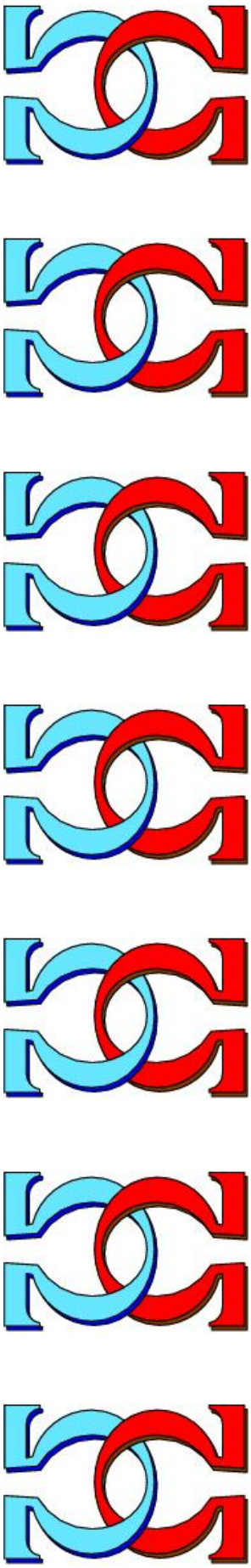

CDMTCS
Research
Report
Series

Algorithms for the Discovery of Embedded Functional Dependencies

Ziheng Wei
The University of Auckland

Sven Hartmann
Clausthal University of Technology

Sebastian Link
The University of Auckland

CDMTCS-542
March 2020

Centre for Discrete Mathematics and
Theoretical Computer Science

Algorithms for the Discovery of Embedded Functional Dependencies

Ziheng Wei
The University of Auckland, New Zealand
z.wei@auckland.ac.nz

Sven Hartmann
Clausthal University of Technology, Germany
sven.hartmann@tu-clausthal.de

Sebastian Link
The University of Auckland, New Zealand
s.link@auckland.ac.nz

March 24, 2020

Abstract

Embedded functional dependencies (eFDs) were recently introduced to tailor relational schema design to data completeness requirements of applications. They also facilitate data cleaning and data integration. A problem that is essential to unlocking these applications is the discovery of all eFDs that hold on a given data set. We show that the discovery problem of eFDs is NP-complete, W[2]-complete in the output, and has a minimum solution space that is larger than the maximum solution space for functional dependencies. Despite these computational challenges, we use novel data structures and search strategies to develop row-efficient, column-efficient, and hybrid algorithms that can efficiently solve the discovery problem for eFDs on large real-world benchmark data sets. Our experiments also demonstrate that the algorithms scale well in terms of their design targets, and that ranking the eFDs by the number of redundant data values they cause can provide useful guidance in identifying meaningful eFDs for applications. Finally, we demonstrate the benefits of introducing completeness requirements and ranking by the number of redundant data values for approximate and genuine functional dependencies.
Keywords: Discovery, Missing data, Embedded Functional Dependency

1 Introduction

Data profiling computes interesting meta data for a given data set [1]. It is of great importance to many applications such as data integration, data cleaning, and database schema design. One of the most important tasks in data profiling is the discovery of data dependencies that hold on the given data set. Somewhat surprisingly, profiling data with missing values has not received much attention yet despite being the standard case in the real-world. In this paper, we investigate the discovery problem of embedded functional dependencies (eFDs) on data with missing values. The class of eFDs was recently introduced to facilitate data-quality driven schema design for data with missing values [30]. While the discovery of eFDs from legacy or sample data is essential to determine which eFDs are good candidates to drive schema design and data cleaning, this problem has not been investigated before.

An eFD is a statement $E : X \rightarrow Y$ where $X, Y \subseteq E$ and X, Y are attribute sets from a relation schema R. It states that the FD $X \rightarrow Y$ holds on the subset r^E of those tuples from the given relation r that have no missing values on any of the attributes in E. The two major advantages of eFDs over FDs is their independence of the interpretation of null markers, and their ability to accommodate data completeness requirements [30]. Hence, eFDs hold on a data set independently of which information a null marker represents. This means the validity of an eFD is beyond guesswork, and strongly supports data integration scenarios in which one must assume that different null marker occurrences may represent different types of missing values, such as inapplicable information or unknown values. Secondly, the ability to accommodate data completeness requirements enables them to tailor relational schema design to application requirements [30].

As an illustration consider the sample r from [30] in Table 1. Here, the *benefit* that a *parent* pays for the *child*ren is uniquely determined by the parent. Hence, the FD $p \rightarrow b$ should hold, but the third tuple causes violations with the first two tuples. In fact, the FD $p \rightarrow b$ is only meaningful for tuples with no missing values on child (the benefit is calculated based on known children), and on parent and benefit as well. This leads to the eFD $cpb : p \rightarrow b$. Embedded FDs can identify redundant data values that cannot be identified by prior work. In the example, both occurrences of **610** are redundant since changing one of these values to any other value will violate the eFD $c : p \rightarrow b$. As the FD $X \rightarrow Y$ holds on the E-complete subrelation r^E, r^E can be decomposed into $r^E[XY]$ and $r^E[X(R-Y)]$ without loss of information. This eliminates redundant data values on the application-relevant part r^E. The reason why $c : p \rightarrow b$ causes redundant data values is because the embedded uniqueness constraint (eUC) $cpb : p$ does not hold: The first two tuples are complete on c, b, p but have matching values on p. Since eFDs cause data redundancy on r^E, we want to restructure r^E such that the corresponding eUC becomes valid and eliminates the data redundancy. The relations on the right column of Table 1 illustrates the decomposition on our example.

Indeed, the decomposition has turned the eFD $cpb : p \rightarrow b$ on r into the eUC $cpb : p$ on $r[pb]$ (the value 610 only occurs once). Similar to how keys prohibit redundant data values while FDs cause them, eUCs prohibit redundant data values on the E-complete part while eFDs cause them there. For this, however, we need completeness requirements. Such requirements have not been considered in previous work on FDs. For instance, every

Table 1: Sample r, lossless decomposition of r^{pbc} into $r^{pbc}[pc]$ and $r^{pbc}[pb]$ (on the right), & tuple in $r - r^{pbc}$

sample r

p(arent)	*b(enefit)*	*c(hild)*
Homer	**610**	Bart
Homer	**610**	Lisa
Homer	915	$\perp$

$r^{pbc}[pc]$

p(arent)	*c(hild)*
Homer	Bart
Homer	Lisa

application-irrelevant $r - r^{pbc}$

p(arent)	*b(enefit)*	*c(hild)*
Homer	915	$\perp$

$r^{pbc}[pb]$

p(arent)	*b(enefit)*
Homer	610

FD holds on any data set with some (approximation) error ratio, i.e. the ratio of pairs of different tuples that violate an FD over all pairs of different tuples. In our example, the aFD $p \to b$ has the ratio 2/3. Typically, aFDs are aimed at permitting small numbers of errors to enable their discovery under dirty data. However, even if we did identify $p \to b$ during aFD discovery, it cannot give us any insight which part of the data can be normalised. Even though completeness requirements are not part of aFD discovery, they enable us to distinguish between errors in tuples that meet the completeness requirements, and those that are caused by missing values. For instance, the error ratio of $p \to b$ on r^{pbc} is 0. Hence, errors come only from tuples that are not as complete as required.

Due to their strong applications for data with missing values [30], the problem of discovering eFDs from given data arises. While the discovery of eUCs has been addressed recently [28], no previous work has investigated eFD discovery. The discovery problem is already challenging for traditional FDs: It's decision variant is both NP- and W[2]-complete in the output, and the number of independent FDs that may hold in a relation with n attributes is exponential in the number of columns [8]. Despite these challenges, there are algorithms that can efficiently solve the problem on large real-world data sets [24, 23, 29]. It is tempting to apply a state-of-the-art FD discovery algorithm to discover eFDs: one can apply such an algorithm to all subsets of tuples that meet any possible completeness requirements, and then aggregate the results. Table 2 shows the discovery times on some benchmark data sets, based on the FD discovery algorithm DHyFD [29]. Already on *hepatitis* this approach requires more than 32.5 hours to discover just over 11k eFDs, compared to discovering just over 8k FDs in under 0.2 seconds with DHyFD. Hence, this is not a viable solution. As completeness requirements are a novel feature of eFDs, and since there are exponentially many possibilities for them, it is unclear how to efficiently cope with a search space that is much larger than that of FDs. For instance, once the FD $X \to A$ is known to hold we do not need to check any FD $XB \to A$ as it is implied by $X \to B$. However, if the eFD $E : X \to A$ is known to hold, we still need to check whether $E' : XB \to A$ holds unless E' contains E. Moreover, while eUC discovery [28] must traverse all possibilities for E such that $E : U$ holds for any given U, eFD discovery must traverse all possibilities for E such that $E : X \to Y$ holds for any fixed combination of X and Y. A second challenge is raised by pure combinatorial

Table 2: Brute force eFD discovery with DHyFD

Data set	#Cols	#Rows	#eFDs	Time (s)
breast-cancer	11	691	48	14.867
bridges	13	108	169	18.95
echocardiogram	13	132	438	13.218
ncvoter	19	1000	1443	8775.43
hepatitis	20	155	11087	117165

arguments. The best known upper bound for the maximum number of independent FDs on a schema with n attributes is $2^n - 1$ [8]. However, we establish in this article the lower bound of $\frac{1}{\sqrt{n}} \cdot (\frac{3\sqrt{3}}{2})^n$ for the maximum number of independent eFDs on a schema with n attributes. Hence, the solution space for eFDs is much larger than that for FDs can ever be. For $n = 2, \ldots, 11$ the best known lower (l_{FD} and l_{eFD}) and upper (u_{FD}) bounds for FDs and eFDs are:

n	2	3	4	5	6	7	8	9	10	11
l_{FD}	2	4	7	11	21	36	71	127	253	465
u_{FD}	3	7	15	31	63	127	255	511	1023	2047
l_{eFD}	2	7	16	51	126	393	1016	3139	8440	25653

We point out an important difference between keys and FDs, and eUCs and eFDs. While a relation over schema R will satisfy the key X if and only if it will satisfy the FD $X \to R$, a relation may satisfy the eFD $E : X \to E$ but violate the eUC $E : X$. The equivalence between $E : X \to E$ and $E : X$ only holds when $E = R$. For instance, the following table

p(arent)	*b(enefit)*	*c(hild)*
Homer	610	Bart
Homer	610	$\perp$

satisfies the eFD $pb : p \to b$ but violates the eUC $pb : p$. Consequently, eUCs are not special cases of eFDs.

In summary, we conclude that eFD discovery is an important problem and requires dedicated algorithms with new search strategies and data structures that can efficiently handle the large solution space. Our main contributions are: (1) We show that the decision variant of eFD discovery is NP- and W[2]-complete. (2) We establish a lower bound for the maximum number of independent eFDs by providing a general construction of a family with that bound. (3) We introduce a novel data structure called *eFD-trees* to store and search eFDs. (4) We introduce the first row-efficient, column-efficient and hybrid algorithms for eFD discovery. (5) We demonstrate the practicality of our algorithms based on their performance on real-world data sets. (6) We show that completeness requirements and ranking by the number of redundant data values are useful for the analysis of approximate and genuine FDs. This is done qualitatively and quantitatively. Data sets are available at `https://bit.ly/2N0pTOS?`.

2 Related Work

Since the 1980s many FD discovery algorithms have been developed for data sets with a large number of *either* rows *or* columns. Row-efficient algorithms, going back to TANE [13], model the search space of FDs as an *attribute lattice*. The lattice is traversed level by level from smaller to larger sets of attributes. An attribute set is pruned if no attributes are functionally dependent on the set. Subsequent algorithms such as FUN [22], FD_MINE [32] and DFD [2] introduced different pruning and lattice traversal strategies. The authors of FUN [22] define *embedded FDs* as FDs that hold on a projection of a given relation onto a subset of columns. These are different from our notion of embedded FDs which are FDs embedded in complete fragments of a relation. Column-efficient algorithms use *agree sets*, which is the collection of columns on which pairs of distinct rows have matching values. By using either maximal agree sets [21] or minimal complements of agree sets [31], column-efficient algorithms use hypergraph transversals. Flach and Savnik used FD-trees to manage the FD set [9]. Examining agree sets iteratively, an FD-tree is updated until it represents the output set after all pairs of rows have been processed. These data structures cannot be efficient in handling the search space for eFDs, since we would require exponentially many attribute lattices and also agree sets in order to deal with all the possible subsets on which records are complete.

A row-efficient hybrid algorithm for the discovery of minimal keys was introduced in [11]. It traverses the attribute lattice simultaneously from the top and bottom, essentially 'halving' the search space by faster pruning. A recent hybrid FD discovery algorithm [24] switches between the column-efficient algorithm from [13] and the row-efficient algorithm from [9]. The row-efficient part validates the FDs of an FD-tree [9] and switches to the column-efficient algorithm when too many FDs are invalidated. The latter generates FDs that do not hold on the input, and switches to the row-efficient part whenever too few invalid FDs are found. The recent hybrid FD discovery algorithm from [29] uses a novel hybridization strategy and the dynamic computation of stripped partitions. Here, stripped partitions will only be updated whenever it is likely that many new FDs can be validated. This balances runtime efficiency with memory consumption.

None of these previous techniques can help with the search of valid eFDs, since such a search needs to accommodate not only (non-)matching values of records but also pay attention to null marker occurrences. For example, the same FD (eg. $A \rightarrow B$) could be valid in many different complete data fragments (eg. ABC, ABD, ABE, ...). Hence, new data structures and strategies need to be developed to accommodate the larger search and solution space. Our new Theorem 3 shows that the potential solution space for eFDs is always larger than that for FDs can ever be.

Wei et al. [28] recently introduced row-efficient, column-efficient, and hybrid algorithms for the discovery of eUCs. Table 3 shows a brief comparison between eUCs and eFDs. This is similar to the difference between keys and FDs from relational databases, but - as pointed out in the introduction - eUCs are not special cases of eFDs. In particular, the inference rules in the eUC (eFD) column show an axiomatization for the individual class of eUCs (eFDs), and adding the two mixed rules from the last row gives us an axiomatization for the combined class of eUCs and eFDs [30]. All discovery algorithms for eUCs [28] work on eUC trees, that have distinct labels for the attributes of

Table 3: Comparison between eUCs and eFDs

Property	eUCs	eFDs
Syntax	$E : U$	$E : X \to Y$
Semantics	no two different E-complete tuples have matching values on U	no two different E-complete tuples with matching values on X have matching values on Y
Example	$cpb : p$	$cpb : p \to b$
Schema design	prohibit E-redundancy	cause E-redundancy
Data cleaning	finds errors as E-complete tuples with matching values on U	finds errors as E-complete tuples with matching values on X and different values on Y
Axiomatization	$\frac{}{R : R}$ $\frac{E : U}{EE' : UU'}$	$\frac{E : X \to Y}{E : XY \to X}$ $\frac{E : X \to Y}{E : X \to XY}$ $\frac{E : X \to Y \quad E' : Y \to Z}{EE' : X \to Z}$
	$\frac{E : X}{E : X \to E}$	$\frac{E : XY \quad E : X \to Y}{E : X}$

E, followed by attributes in U. Simply extending this data structure to eFDs would be inefficient since too many candidate embeddings E would need to be generated for too many different combinations of the left-hand and right-hand side attribute sets X and Y, respectively. The data structure of eFDs we introduce here uses a novel representation that stores as many valid eFDs in a single path as possible. In contrast to eUC trees we use here the concise form of eFDs which only represents attributes in E that are not part of X or Y. This helps with the search, update, and induction of new candidates. For the row-efficient algorithms, both eUC and eFD discovery must traverse embeddings in contrast to previous work. However, the conditions characterizing when embeddings exist, how the candidates for embeddings can be traversed efficiently, and how they can be validated is much simpler for a given unique constraint than for a given FD, based on the deeper semantics and interaction of eFDs. For column-efficient algorithms, both eUC and eFD discovery are based on the generation of new candidates from data samples that invalidate previous candidates. New eUC candidates are generated from embedded non-uniques, but to generate new eFD candidates we need to introduce non-eFDs. Since the interaction of non-eFDs is very different from that of embedded non-uniques, novel techniques are required to generate new candidate eFDs effectively. For hybrid algorithms, both eUC and eFD discovery use sampled violations from the column-efficient part to reduce the search space in the row-efficient part, and both reduce the number of new candidates in the column-efficient part by directly validating candidates in the row-efficient part. However, for eFD discovery the sampled violations can only reduce the local search space and a new technique is necessary to remove all implied non-eFDs from the eFD tree.

Approximate FDs [19, 7] permit few violations of FDs to increase the recall of meaningful FDs. In [19] the authors define an *aFD* as an FD $X \to A$ with its (approximation) error ratio ϵ_r on a relation r, which is the ratio of those pairs of distinct tuples in r that agree on X and differ on A over all pairs of distinct tuples in r. Note that aFDs are only defined for singleton attributes on the right-hand side, and not for occurrences of null markers. In general, every FD holds with some ϵ_r on any given relation r. In principle, aFDs were not defined to include completeness requirements, and are thus different from eFDs. For example, the FD $p \to b$ may be discovered as an aFD with $\epsilon_r = 2/3$ in Table 1, but does not give us any hint whether the eFD $cpb : p \to b$ is satisfied. Vice versa, if the eFD $E : X \to Y$ holds on r, then we have the strict upper bound $\epsilon_r < |r^E|/|r|$ for

the FD $X \to Y$. Hence, eFDs tell us something about aFDs, while aFDs cannot tell us anything about eFDs. Nevertheless, it is useful to think about the potential benefit that eFDs may have on aFDs. We will show that completeness requirements and ranking the number of redundant data values are beneficial for aFDs.

Genuine FDs (gFDs) [4] measure the impact of missing values on the validity of (exact and approximate) FDs. The level of genuineness γ of an FD is informally the degree by which the FD holds on the true completion of the data set. The authors compute gFDs by comparing a clean and complete data set to a version of the data set where some percentage of domain values were replaced by the null marker. Then they distinguish between *same FDs* which are those FDs that hold on both data sets, *fake FDs* which are those that hold on the dirty but not the clean data set, and *ghost FDs* which are those that hold on the clean but not the dirty data set. Genuine FDs are formed by the union of the sets of same and ghost FDs. Indeed, gFDs have been defined to estimate the impact of nulls on FDs and aFDs, while eFDs drive schema design for incomplete data. Similar to the case of aFDs, however, it is beneficial to think about the benefits of incorporating completeness requirements in the analysis of gFDs. Indeed, the discovery of $E : X \to Y$ on a relation r guarantees that r^E is the largest subrelation of r on which $X \to Y$ holds without doubt. In addition, if an eFD $E : X \to Y$ holds on the clean (and possibly incomplete) data set r, then an introduction of any null markers in columns from E will not violate the $E : X \to Y$. In this sense, there will not be any ghost FDs and the set of genuine FDs simply becomes the set of the same FDs. Of course, by introducing null markers in columns from E we may still generate fake FDs. Hence, it makes a lot of sense to pivot the computation of gFD along completeness requirements, starting from the set of tuples that is complete on E, for instance. More fundamentally and similar to our observations with aFDs, we can define the levels of genuineness with respect to r^E. Again, this would allow us to distinguish the levels of genuineness based on tuples that meet the completeness requirements of an application, and based on any tuples. Similar to aFDs, we will show that completeness requirements and ranking the number of redundant data values are beneficial for gFDs.

Hence, we are the first to investigate the discovery and ranking of eFDs, raising new challenges over previous work. Efficient solutions help with data-quality driven schema design and other applications. Including completeness requirements and ranking by data redundancy benefits other variants of functional dependencies.

3 Embedded FDs

We fix notions and notation in this section.

A relation schema is a finite, non-empty set R of attributes (also called *column (names)*). With each attribute A we associate a domain $dom(A)$ of values that can occur in A. A tuple t over R (*row* or *record*) is a function that maps each $A \in R$ to a value in $dom(A)$. Two records are equal if they have matching values on all the attributes of R, and distinct otherwise. A relation r over R is a finite set of tuples over R. For finite attribute sets $X = \{A_1, A_2, \ldots A_m\}$ and Y, we write $A_1A_2 \ldots A_m$ for X, and XY instead of the set union $X \cup Y$. Attribute sets may be called *column combinations*. For $X \subseteq R$

and a tuple t over R, we write $t(X)$ for the projection of t onto X. We use the symbol $\perp$ to denote the *null marker*. While $\perp$ is a marker and not a value, we abuse notation for convenience and assume that $\perp$ is a distinct element of each domain. We say a tuple t over R is X-total if $t(A) \neq \perp$ for all $A \in X$. We use r^X to denote the set of all X-total tuples in a relation r, and call r^X the *scope* of r with respect to X. A relation is *complete* when it has no null marker occurrence, that is, when the scope r^R and r coincide.

An *embedded FD* (eFD) over relation schema R is an expression of the form $E : X \to Y$ where $X, Y \subseteq E \subseteq R$. We call E the *embedding*, X the *left-hand-side* (LHS), and Y the *right-hand-side* (RHS). A relation r over R *satisfies* eFD $E : X \to Y$, denoted by $r \vDash E : X \to Y$, if and only if for all $t, t' \in r^E$, $t(X) = t'(X)$ implies $t(Y) = t'(Y)$. If r does not satisfy $E : X \to Y$, we also say r *violates* $E : X \to Y$. The *concise form* of $E : X \to Y$ is the expression $E - XY : X \to Y$.

Stripped partitions are often used to validate FDs [13]. We now define an extension of stripped partitions for the purpose of validating eFDs. Let r be a relation over R and $E \subseteq R$. The *E-equivalence class* of tuple $t \in r$ is the set $[t]_E = \{s \in r^E \mid s[E] = t[E]\}$. The *stripped partition* of a relation r over E is $\pi_E(r) = \{[t]_E \mid t \in r^E, |[t]_E| \geq 2\}$.

The *discovery problem* of eFDs is to compute a representation of all eFDs satisfied by a given relation. In FD discovery, a LHS-reduced cover is widely utilized [23]. For eFDs, our proposed algorithms compute a *canonical* cover for the given relation. That is, eFDs contain one attribute on their RHS, and removing any attribute from their LHS or embedding will cause a violation of the resulting eFD on the relation.

4 Computational Challenges

We settle the computational complexity of the following decision variant eFD for the eFD discovery problem.

Problem:	eFD
Input:	relation r over schema R
	positive integer k
Output:	yes, if there is some $E : X \to A$ satisfied by r
	where $X \subseteq E \subseteq R$, $A \in E - X$ and $\|E\| \leq k$
	no, otherwise

We use the cardinality $|E|$ as the size of eFD $E : X \to A$ because $XA \subseteq E$. Discovering eFDs is at least as hard as discovering FDs in complete relations. The decision variant FD of FD discovery, as defined in the box below, is NP-complete. By reducing FD to eFD, we establish NP-hardness for eFD.

Problem:	FD
Input:	relation r over schema R
	positive integer k
Output:	yes, if there is some $X \to A$ satisfied by r
	where $X \subseteq R$, $A \in R - X$ and $\|XA\| \leq k$
	no, otherwise

Theorem 1 *The problem* eFD *is* NP*-complete.*

Proof eFD is in NP because we can guess $E : X \to A$ with $|E| \leq k$ and verify in polynomial time using Algorithm 4 (see Section 6). For the NP-hardness, we reduce FD to eFD. Take an instance (r, k) of FD where r is a complete relation over relation schema R, and k is positive integer. Let (r', k') be the instance of eFD where $r = r'$ and $k = k'$. Now it follows that a non-trivial $FD\ X \to A$ where $|XA| \leq k$ is satisfied by r if and only if the eFD $X : X \to A$ is satisfied by r'.

FD is W[2]-complete in the output [5]. We show that FD and eFD are FPT-equivalent, so eFD is W[2]-complete in the output. As eFDs cannot express eUCs, we cannot simply reduce the decision variant for eUC discovery [28] to eFD.

Theorem 2 (Fixed-parameter intractability)
The problem eFD *is* W[2]*-complete in the size of the output.*

Proof We show that eFD and FD are equivalent under FPT-reductions. The result then follows from the W[2]-completeness of FD in [5]. For FD$\leq_{FPT}$eFD the PTIME reduction is the same as the construction used for Theorem 1 since parameter k' only depends on k. It remains to show that eFD$\leq_{FPT}$FD holds. Take an instance (r, k) of eFD. We transform (r, k) into an instance (r', k') by defining r' as the result of replacing null marker occurrences in r with unique column values in r', and defining k' to be k. Clearly, this transformation is FPT. Now we claim that there is some eFD $E : X \to A$ satisfied by r if and only if there is some $FD\ E - A \to A$ satisfied by r' $|E| \leq k$. If there is an eFD $E : X \to A$ where $|E| \leq k$ that is satisfied by r, then r^E satisfy $X \to A$ furthermore $r^E \vDash E \to A$; for any other tuples they satisfy $E \to A$ trivially in r' because for any tuple $t, t' \notin r^E$ there is a null maker in $t(E)$ and $t'(E)$ hence $t(E) \neq t'(E)$ with respect to r'. If there is an $FD\ E - A \to A$ where $A \in E$ and $|E| \leq k$ satisfied by r', then there is $E : X \to A$ where $X \subseteq E - A$ satisfied by r since $E : E - A \to A$ must be satisfied by r. This concludes the proof.

Remarkably, recent algorithms can quickly solve large instances for the FD and candidate key discovery problems [24, 26]. However, the efficiency bar is raised even higher for eFDs. This results from the solution space of the eFD discovery problem. For most classes of constraints exact numbers for the maximum solution space are unknown, including for FDs. For example, only lower and upper bounds are known for the maximum cardinality of a *non-redundant* family of FDs over a schema with n attributes [8], and the best known upper bound of $2^n - 1$ is very rough.

For each pair E, X with $X \subset E \subseteq R$ we fix an attribute $B_{E,X} \in E - X$, and let $m = \lceil n/2 \rceil$. Now we define

$$\Sigma := \{E : X \to Y \mid X \subseteq E \subseteq R, Y = XB_{E,X}, |E| = m + j, |X| = m - j - 1, j = 0, 1, \ldots, m - 1\},$$

which is the non-redundant family Σ of eFDs for which we can establish the lower bound in the next theorem. Indeed, for each eFD $E : X \to Y$ in Σ we have $|E| + |X| = 2m - 1$. Using Vandemonde convolution, we can show the following.

Theorem 3 *For every positive integer $n \geq 2$, there is a non-redundant family of eFDs over a schema with n attributes, which has at least $c \cdot \frac{1}{\sqrt{n}} \cdot (\frac{3\sqrt{3}}{2})^n$ elements for some constant c.*

Proof We proceed in three stages.
Stage 1:

Fact. An eFD $\sigma' = F : V \to W$ can be derived from a set of eFDs Σ' if and only if it can be derived from $\Sigma'' = \{F : X \to Y \mid \exists E : X \to Y \in \Sigma' \text{ s.t. } E \subseteq F\}$

Fact. Let $V \subseteq R$ be such that every eFD $\sigma = E : X \to Y$ in Σ satisfies either $X \not\subseteq V$ or $Y \subseteq V$. Then the same property holds for every eFD in Σ^+.

We show this by inspecting each of the inference rules. By the reflexivity axiom, an eFD $\sigma = E : X \to Y$ holds when $Y \subseteq X$. Hence, $X \subseteq V$ immediately yields $Y \subseteq V$.

By the extension rule, an eFD $\sigma = E : X \to Y$ implies an eFD $\sigma' = E : X \to XY$. Assume σ has the property under inspection. Suppose $X \subseteq V$, then $Y \subseteq V$ holds by assumption. This yields $XY \subseteq V$. Hence, σ' has the property, too.

By the transitivity rule, two eFDs $\sigma = E : X \to Y$ and $\sigma' = E' : Y \to Z$ imply an eFD $\sigma'' = EE' : X \to Z$. Assume σ and σ' have the property under inspection. Suppose $X \subseteq V$, then $Y \subseteq V$ holds by assumption, and therefore $Z \subseteq V$ holds by assumption, too. Hence, σ'' has the property, too.
Stage 2:

Fact. Σ is non-redundant.

Consider an eFD $\sigma' = F : V \to W$ in Σ. We will demonstrate that σ' cannot be be derived from $\Sigma' = \Sigma - \sigma$. Recall that σ' can be derived from Σ' if and only if it can be derived from $\Sigma'' = \{F : X \to Y : \exists E : X \to Y \in \Sigma' \text{ s.t. } E \subseteq F\}$. Let $\sigma = E : X \to Y$ be an eFD in $\Sigma' = \Sigma - \sigma$ with $E \subseteq F$. We will show that σ satisfies either $X \not\subseteq V$ or $Y \subseteq V$. Suppose $X \subseteq V$. Due to $E \subseteq F$ we have $|X| \geq |V|$ which yields $X = V$. By definition of Σ we obtain $Y = XB_{E,X} \not\subseteq V = X$. That is, σ has the property under inspection. On the other hand, σ' does not have this property since both, $V \subseteq V$ and $Y = XB_{E,X} \not\subseteq V$ hold. Hence, σ' cannot be derived from $\Sigma' = \Sigma - \sigma$ as claimed.
Stage 3: Next we study the cardinality of Σ. By the definition of Σ we have the following:

$$|\Sigma| = \sum_{j=0}^{m-1} \binom{n}{m+j} \binom{m+j}{m-j-1} \tag{1}$$

$$= \sum_{j=0}^{m-1} \frac{n!}{(n-m-j)! \cdot (m-j-1)! \cdot (2j+1)!} \tag{2}$$

The cardinality of Σ gives us a lower bound for the maximum number of independent eFDs.

We will now provide a straightforward lower estimate to show that the magnitude of $|\Sigma|$ is strictly larger than the magnitude of the maximum number of independent FDs.

For even n (that is, $n = 2m$) the sum in Eq. (1) can also be written as follows:

$$
\begin{aligned}
|\Sigma| &= \sum_{j=0}^{m-1} \frac{n!}{(m-j)! \cdot (m-j-1)! \cdot (2j+1)!} && (3)\\
&= \sum_{j=0}^{m-1} \frac{n!}{j! \cdot (j+1)! \cdot (n-1-2j)!} && (4)\\
&= \sum_{j=0}^{m-1} \frac{(2m)!}{j! \cdot (j+1)! \cdot (2m-1-2j)!} && (5)
\end{aligned}
$$

Similarly, for odd n (that is, $n = 2m-1$) the sum in Eq. (1) can be written as follows:

$$
\begin{aligned}
|\Sigma| &= \sum_{j=0}^{m-1} \frac{n!}{(m-j-1)! \cdot (m-j-1)! \cdot (2j+1)!} && (6)\\
&= \sum_{j=0}^{m-1} \frac{n!}{j! \cdot j! \cdot (n-2j)!} && (7)\\
&= \sum_{j=0}^{m-1} \frac{(2m-1)!}{j! \cdot j! \cdot (2m-1-2j)!} && (8)
\end{aligned}
$$

We can estimate the sum in Eq. (1) using the Vandemonde convolution (for $x \leq m-1+y$), see [10]:

$$
\binom{m-1+y}{x} = \sum_{j=0}^{m-1} \binom{m-1}{j} \binom{y}{x-j} \tag{9}
$$

In case of even n we use $y := n$ and $x := n-1$. This gives

$$
\begin{aligned}
\binom{3m-1}{2m-1} &= \binom{m-1+n}{n-1} = \sum_{j=0}^{m-1} \binom{m-1}{j} \binom{n}{n-1-j} && (10)\\
&= \sum_{j=0}^{m-1} \frac{(2m)! \cdot (m-1)!}{j! \cdot (j+1)! \cdot (m-1-j)! \cdot (2m-1-j)!} && (11)
\end{aligned}
$$

In the case of odd n we use $y := n$ and $x := n$. This gives

$$
\begin{aligned}
\binom{3m-2}{2m-1} &= \binom{m-1+n}{n} = \sum_{j=0}^{m-1} \binom{m-1}{j} \binom{n}{n-j} && (12)\\
&= \sum_{j=0}^{m-1} \frac{(2m-1)! \cdot (m-1)!}{j! \cdot j! \cdot (m-1-j)! \cdot (2m-1-j)!} && (13)
\end{aligned}
$$

Now it is easy to verify that

$$\frac{(m-1)!}{(m-1-j)! \cdot (2m-1-j)!} \leq \frac{1}{(2m-1-2j)!} \tag{14}$$

for every $j = 0, 1, \ldots, m-1$.

This implies $\binom{3m-1}{2m-1} \leq |\Sigma|$ for even $n = 2m$ and $\binom{3m-2}{2m-1} \leq |\Sigma|$ for odd $n = 2m-1$.

For even $n = 2m$ we can further observe $|\Sigma| \geq \binom{3m-1}{2m-1} = \frac{2}{3} \cdot \binom{3m}{2m}$. Similarly for odd $n = 2m-1$ we can further observe $|\Sigma| \geq \binom{3m-2}{2m-1} = \frac{3m-2}{2m-1} \cdot \binom{3m-3}{2m-2} \geq 2 \cdot \binom{3(m-1)}{2(m-1)}$. For good lower and upper bounds of binomial coefficients of this particular form we refer to [27] where, in particular, the following has been shown:

$$\binom{3m}{2m} \geq c \cdot \frac{1}{\sqrt{m}} \cdot (\frac{27}{4})^m \tag{15}$$

Using Eq. (15) we conclude the following estimate for the cardinality of Σ:

$$|\Sigma| \geq c \cdot \frac{1}{\sqrt{n}} \cdot (\frac{3\sqrt{3}}{2})^n \tag{16}$$

This shows that the solution space for eFDs is guaranteed much larger than the solution space for FDs can ever be.

5 eFD-Trees As Data Structures

FD discovery has benefited from special data structures to store and update intermediate results, and provide efficient access, search, and pruning capabilities. Particularly, *FD-trees* [9] have improved the runtime of FD discovery. For the larger search space that eFDs require, we propose a novel data structure for eFD discovery, called *eFD-tree*.

Definition 1 (eFD-tree) *Let R be a relation schema with a total order of attributes. An* eFD-tree *is a tree with a unique* l(eft-hand-side)-root*, multiple* e(mbedding)-roots*,* e(mbedding)-nodes*,* l(eft-hand-side)-nodes *and the following properties: (1) Every node, except root nodes, is an attribute of R; (2) All children of an e-root are e-nodes; (3) E-nodes only have e-node children; (4) eFD-nodes are the e-nodes labeled with RHS attributes of an eFD; (5) L-nodes and the l-root can have l-node children and at most one e-root child; (6) All child nodes, except e-roots, have larger attributes than their non-root parents; (7) Each traversal of the tree from the l-root to an eFD-node represents an eFD in its concise form.*

Example 1 *Figure 1 shows 3 choices to store $\{AB : A \to B, AC : A \to C\}$. Tree* (a) *uses attributes in the embedding as the prefix of a path. Tree* (b) *reverses the construction of* (a) *so that LHS attributes become the prefix of a path. Tree* (c) *is the eFD-tree in Definition 1 which follows* (b) *but stores eFDs in their concise form: $\emptyset : A \to B$, $\emptyset : A \to C$.*

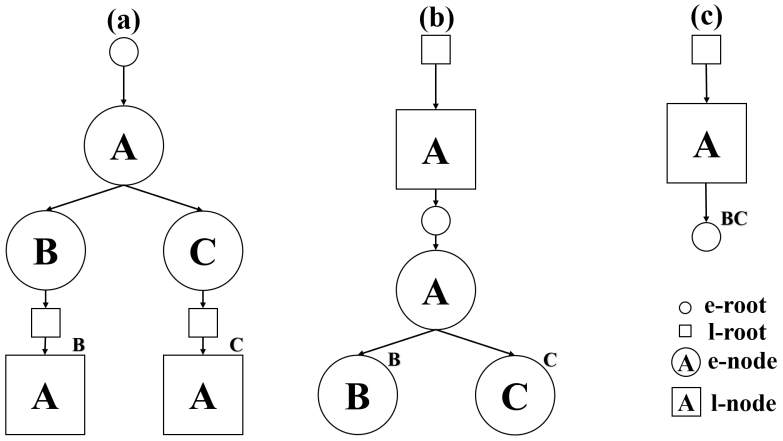

Figure 1: Representing eFDs $AB : A \rightarrow B$ and $AC : A \rightarrow C$

By using the concise form of eFDs, eFD-trees need only one path to represent as many eFDs as possible, which improves search and update time. The eFD-tree *(c)* in Example 1 captures different eFDs by a single path.

During eFD discovery newly discovered eFDs may be implied by previously discovered eFDs. For example, if eFD $AB : A \rightarrow B$ is satisfied, then $ABC : A \rightarrow B$ is also satisfied. In fact, the first and the third rule of the axiomatization in the eFD column of Table 3 show that the FD $AB : A \rightarrow B$ and the trivial FD $ABC : B \rightarrow B$ imply $ABC : A \rightarrow B$. Such implications should be efficiently managed by an eFD-tree.

Algorithm 1 reduces the size of an eFD tree by eliminating eFDs that are implied by eFDs with smaller LHS or embedding. To check if eFD $E : X \rightarrow A$ is implied, Algorithm 1 recursively traverses an eFD tree in a depth-first search (DFS). Starting with the l-root node, eFD $\emptyset : \emptyset \rightarrow A$ is checked for existence (step 5 - 7). After that, subsets of X will be examined recursively by traversing to any child l-node in X (step 13 - 20). Before the next l-node (step 17 - 20), we check if the current node also has an e-root node attached. If it does, subsets of E will be checked for all l-nodes in the current path by traversing any child e-node from the e-root in E (step 8 - 12). During DFS, if any e-node or e-root node contains attribute A as its RHS label, an implied eFD has been found (step 5 - 7). Otherwise, no eFD in the eFD-tree implies the given eFD.

6 Row-efficient Discovery

In this section, we introduce a *row-efficient* (column-based) discovery algorithm for eFDs, whose runtime mainly depends on the number of the given columns.

In FD discovery, a row-efficient algorithm models the LHSs of FDs as an *attribute lattice* (see the left of Figure 2) and uses the FD validation algorithm to find the satisfiable RHS for a given LHS. Classical approaches such as [13] cannot directly be adapted to discover eFDs. To discover eFDs in a row-efficient manner, there are three main challenges to overcome. Firstly, the search space of eFDs including both embeddings and LHSs is much larger than a single attribute lattice. Secondly, discovering eFDs needs to

Algorithm 1

```
1: Input:  E : X → A, l_root of eFD-tree for eFD set Σ
2: Output:  true if there is some E' : X' → A ∈ Σ(E' ⊆ E ∧ X' ⊆ X)
3: return  implied(l_root, E, X, A)
4: function implied(node, E, X, A)
5:     if node is eFD-node then
6:         if A ∈ RHS(node) then
7:             return  true
8:     if node is e-node then
9:         for all child e-node child of node do
10:            Let B be the attribute of child
11:            if B ∈ E and implied(child, E, X, A) then
12:                return  true
13:    if node is l-node then
14:        if node has an e-root node e_root then
15:            if implied(e_root, E, X, A) then
16:                return  true
17:        for all child l-node child of node do
18:            Let B be the attribute of child
19:            if B ∈ X and implied(child, E, X, A) then
20:                return  true
21:    return  false
```

search through all possible combinations of embeddings and LHSs. An efficient strategy must be devised to traverse and prune the search space of eFDs. Thirdly, eliminating eFDs with redundant embeddings must examine $eFDs$ discovered at all lower levels, unlike the row-efficient discovery algorithms for FDs which only need to inspect valid FDs from one level lower.

As shown on the top of Figure 2, each attribute subset of the lattice for LHSs generates a filter in the attribute lattice for the possible embeddings. We therefore propose a *nested traversal strategy* for the row-efficient discovery of eFDs. The nested traversal starts with an *l-traversal* which will examine the attribute lattice of LHSs from lower to higher levels. For each LHS, an *e-traversal* will be used to traverse all possible embeddings for a given pair of LHS and RHS. During an e-traversal, eFDs are examined for validity and implication. If an eFD is valid and non-implied, it will be stored in our eFD-tree. Since e-traversals are nested in an l-traversal, classical pruning strategies are no longer valid. For example, the validity of $E : X \rightarrow A$ does not imply the validity of $E' : XB \rightarrow A$ since E is not necessarily a subset of E'. Hence, finding suitable embeddings for a given LHS may not reduce the attribute lattice of LHSs. For improving the runtime performance of row-efficient eFD discovery we avoid unnecessary e-traversals and reduce attribute lattices of embeddings using the following propositions.

Proposition 1 *Let Σ be the canonical cover of eFDs that hold on the relation r over R. $R(\perp)$ is the set $\{A \in R \mid \exists t \in r : t(A) = \perp\}$. For any eFD $E : X \rightarrow A \in \Sigma$, $E - XA \subseteq R(\perp)$.*

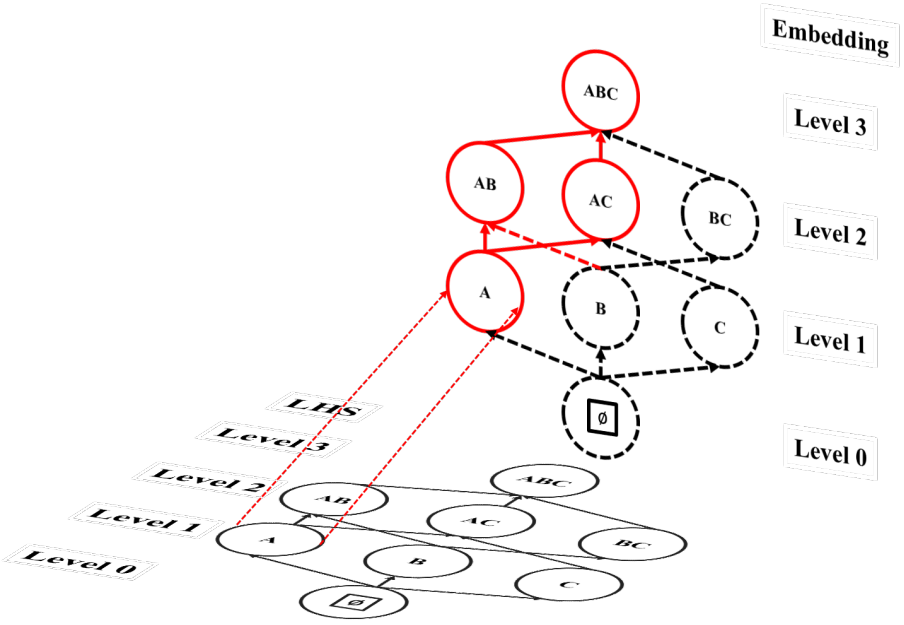

Figure 2: Bottom: the attribute lattice of LHSs over $R = \{A, B, C\}$. Top: the attribute lattice (within the parallelogram) of LHS $\{A\}$'s embeddings

Proof Let r be a relation over relation schema R and $R(\perp) = \{A \in R \mid \exists t \in r : t(A) = \perp\}$. Take any $E : X \to A \in \Sigma$ where $XA \subset E$ and $B \in E - XA$. Assume $B \notin R(\perp)$. By the definition of eFDs, $t_1(X) = t_2(X)$ implies $t_1(A) = t_2(A)$ for any $t_1, t_2 \in r^E$. If $B \notin R(\perp)$, then $t(B) \neq \perp$ for all $t \in r$. In other words, $r^B = r$. Furthermore, we can derive $r^{E-B} = r^E \cup (r^E \cap r^B)$. Namely, $r^{E-B} = r^E$. However, $r \vDash E - B : X \to A$ and $E - B : X \to A$ must be in the canonical cover Σ instead of $E : X \to A$, which draws a contradiction. Therefore, B must be in $R(\perp)$.

Proposition 1 restricts our search for suitable embeddings to attributes on which nulls occur. Due to the concise form of eFDs, an e-traversal for $X \to Y$ over R only needs to traverse the lattice over $R(\perp) - X$.

Proposition 2 *Let r be a relation over relation schema R. For any $X \subseteq R$ and $A \in R - X$, there is some $E \subseteq R$ where $XA \subseteq E$ such that $r \vDash E : X \to A$ if and only if $t_1(A) = t_2(A)$ for all $t_1, t_2 \in S \cap r^R$ and for all $S \in \pi_X$.*

Proof Let r be a relation over relation schema R an. Take any $X \subseteq R$ and $A \in R - X$. π_X is the stripped partition of X over r.

Case $\Rightarrow$*:* Suppose there exists $E \subseteq R$ where $XA \subseteq E$ such that $r \vDash E : X \to A$. Take any $S \in \pi_X$. Assume there exists $t_1, t_2 \in S \cap r^R$ where $t_1(A) \neq t_2(A)$. However, $t_1(X) = t_2(X)$ and $t_1, t_2 \in r^R \subseteq r^E$, which contradicts $r \vDash E : X \to A$.

Case $\Leftarrow$*:* Suppose $t_1(A) = t_2(A)$ for all $t_1, t_2 \in S \cap r^R$ and for all $S \in \pi_X$. Therefore, $t_1(X) = t_2(X)$ implies $t_1(A) = t_2(A)$ if $t_1, t_2 \in r^R$. In other words, $r \vDash R : X \to A$.

Proposition 2 characterizes when an entire e-traversal for $X \to Y$ can be skipped since any possible embeddings will lead to invalid eFDs.

Algorithm 2 L-traversal

Input: relation r over R, the l-root l_root of an eFD-tree
Output: eFDs satisfied by r
$l = 0$
$candidates = \{\emptyset\}$
$rhs[\emptyset] = R$
while $|candidates| > 0$ **do**
    for all $X \in candidates$ **do**
        $Y' = \{A \in rhs[X] \mid r \vDash XA : X \to A\}$
        $Y'' = \{A \in rhs[X] \mid r \vDash R : X \to A\} - Y'$
        if $|Y''| > 0$ **then**
            $e\_traversal(r, R, l\_root, X, Y'', \pi_X)$
        $rhs[X] = rhs[X] - Y'$
    $l = l + 1$
    $candidates = \{X \subseteq R \mid |X| = l\}$
    for all $X \in candidates$ **do**
        $Y' = \bigcap\{rhs[X'] \mid X' \subseteq X \wedge rhs[X'] \neq \emptyset\} - X$
        $Y' = \{A \in Y' \mid XA : X \to A \text{ is not implied}\}$
        if $|Y'| > 0$ **then** $rhs[X] = Y'$
        else Remove X from $candidates$

Algorithm 3 E-traversal

Input: relation r over R, the l_root of an eFD-tree, LHS attributes X, RHS attributes Y, stripped partition π_X
Output: updates on the eFD-tree of l_root with all $E \subseteq R$ such that $r \vDash E : X \to Y$ not implied
$l = 0$
$candidates = \{\emptyset\}$
$rhs[\emptyset] = Y$
while $|candidates| > 0$ **do**
    for all $E \in candidates$ **do**
        $Y' = \{A \in rhs[X] \mid r \vDash EXA : X \to A\}$
        if $|Y'| > 0$ **then**
            Insert $E - XY' : X \to Y'$ into l_root
            $rhs[X] = rhs[X] - Y'$
    $l = l + 1$
    $candidates = \{E \subseteq R(\bot) - X \mid |E| = l\}$
    for all $E \in candidates$ **do**
        $Y' = \bigcap\{rhs[E'] \mid E' \subseteq E \wedge rhs[E'] \neq \emptyset\} - E$
        $Y' = \{A \in Y' \mid E : X \to A \text{ not implied in } l\_root\}$
        if $|Y'| > 0$ **then** $rhs[E] = Y'$
        else Remove E from $candidates$

Algorithms 2 and 3 form the foundation of the row-efficient algorithm. We first use the l-traversal algorithm to search through all possible LHSs. Each LHS candidate has a set of possible RHS attributes. As the level goes up in the l-traversal, sizes of LHS candidates become larger but the size of the RHS assigned to a LHS becomes smaller. For each pair of LHS and RHS candidates, an e-traversal considers all possible embeddings. E-traversals are similar to l-traversals but only examine and validate embeddings with a fixed LHS. Following e-traversals (step 11), the l-traversal will generate new pairs of LHS and RHS candidates for the next level. A new LHS candidate must be a superset of some LHS candidate at the current level (step 14). The new RHS candidate of a new LHS must be the intersection of all the RHSs whose corresponding LHSs are a subset of the new LHS (step 15). For example, if a RHS attribute is not in the intersection, the attribute belongs to the RHS of some eFD that has been validated at a lower level.

Prior to e-traversals, we use two heuristics to reduce the eFD search space. Firstly, we verify which RHS attributes can lead to valid eFDs (step 8). If there is a non-empty subset of the given RHS that leads to a valid eFD, any search for larger LHSs for this RHS are unnecessary (step 12). Indeed, supersets of the LHS will lead to valid eFDs with the same RHS. Secondly, we verify which attributes from the given RHS meet the conditions in Proposition 2 (step 9). Hence, e-traversal will find at least one valid eFD.

Note that eFDs are discovered in their concise form. During l-traversal, RHS candidates do not intersect with LHS candidates. For example, if $X = ABC$ and $Y' = CD$, then $E : ABC \rightarrow C$ is trivial for any E. Similarly, during e-traversal, RHS candidates do not intersect with candidate embeddings. For example, $E = ABC$ and $Y' = CD$ are equivalent to $E = AB$ and $Y' = CD$, which were examined at lower levels of the lattice for embeddings.

Finally, we comment on how to generate candidates for a level of an attribute lattice (LHSs or embeddings), and how to validate eFDs. Candidate attribute sets for given levels are generated efficiently by using *prefix blocks* [13]. Step 14 in Algorithm 2 and step 13 in Algorithm 3 describe which candidates occur at each level, but these are implemented by prefix blocks. Stripped partitions can be efficiently utilized to validate eFDs, as shown in Algorithm 4. As LHS candidates are generated level by level, the stripped partition of a LHS can be computed incrementally by Algorithm 5.

Algorithm 4 Validation

```
1: Input: relation r over R, eFD E : X → A, stripped partition π_X(r)
2: Output: true, if r ⊨ E : X → A
3: for all S ∈ π_X(r) do
4:     S^E = ∅
5:     for all t ∈ S do
6:         if ∀A ∈ E : t(A) ≠⊥ then S^E = S^E ∪ {t}
7:     if |S^E| > 0 then
8:         Let S^E = {t_1, ..., t_n}
9:         for all t_i ∈ S^E where i > 1 do
10:            if t_i(A) ≠ t_1(A) then return false
11: return true
```

Algorithm 5 Stripped partition

Input: relation r over R, stripped partition $\pi_X(r)$, attribute $A \in R - X$
Output: stripped partition $\pi_X(r)$
$\pi_{XA}(r) = \emptyset$
for all $S \in \pi_X(r)$ **do**
    $V = \emptyset$
    Let M be a mapping from $dom(A)$ to sets of tuples
    for all $t \in S$ **do**
        if $t(A)$ does not exist in V **then**
            $M[t(A)] = \{t\}$, $V = V \cup \{t(A)\}$
            continue
        $M[t(A)] = M[t(A)] \cup \{t\}$
    for all $v \in V$ **do**
        if $|M[v]| > 1$ **then** $\pi_{XA}(r) = \pi_{XA}(r) \cup \{M[v]\}$
return $\pi_{XA}(r)$

In summary, Algorithm 2 enumerates all possible eFDs by traversing lattices of LHSs and embeddings, and validates resulting eFDs on the input relation. An l-traversal traverses all possible LHSs from small to large size. Hence, eFD with larger LHS are pruned whenever they are impled by an eFD with smaller LHS. Given a LHS, an e-traversal traverses all possible embeddings for a given LHS. Eventually, every possible eFD will be either validated, violated, or implied. A validated eFD is stored and helps detect implied eFDs later; a violated eFD increments its LHS or embedding; and implied eFDs are discarded. Algorithm 2 terminates once the remaining search space becomes redundant, or the l-traversal has examined the maximum level of the LHSs' lattice.

Theorem 4 *Algorithm 2 computes the canonical cover of all eFDs that are satisfied by the given relation.*

7 Column-efficient Discovery

Column-efficient algorithms extract counter-examples from a given relation and use them to derive valid constraints. We introduce a *column-efficient (row-based)* discovery algorithm for eFDs. Firstly, we define *embedded non-FDs* to represent violations of eFDs in a given relation.

Definition 2 *An* embedded non-FD *(*non-eFD) over relation schema R is an expression $E : X \not\rightarrow Y$ where $X, Y \subseteq E$ and $X \cap Y = \emptyset$. We say a non-eFD $E : X \not\rightarrow Y$ is *valid* in relation r over R if and only if there are tuples $t_1, t_2 \in r^E$ such that $t_1(X) = t_2(X)$ and $t_1(B) \neq t_2(B)$ for all $B \in Y$.

A valid non-eFD $E : X \not\rightarrow Y$ provides a counter-example for the validity of several eFDs. Firstly, for all $E' \subset E$ and $XY \subseteq E'$, the eFD $E' : X \rightarrow Y$ cannot be valid. Secondly, for all $X' \subset X$, the eFD $E : X' \rightarrow Y$ cannot be valid. Thirdly, for all $Y' \subseteq Y$,

the eFD $E : X \to Y'$ cannot be valid. Hence, we say a non-eFD $E : X \not\to Y$ *contradicts* the eFD $E' : X' \to Y'$ if and only if $E' \subseteq E$, $X' \subseteq X$ and $Y' \subseteq Y$. We require $Y' \subseteq Y$ (as opposed to $Y' \cap Y \neq \emptyset$) since we only a get counter-example for the validity of $E' : X' \to Y' \cap Y$ but not for the validity of $E' : X' \to Y' - Y$. Given a set of eFDs, if a non-eFD contradicts some eFD, the eFD is augmented to generate new eFD candidates. Proposition 3 shows that valid eFDs can be found when all valid non-eFDs no longer contradict any of the new eFD candidates.

Proposition 3 *Let r be a relation over R and Σ^{-1} the set of all non-eFDs valid in r. An eFD $E : X \to Y$ is satisfied by r if and only if there is no non-eFD $E' : X' \not\to Y' \in \Sigma^{-1}$ such that $E \subseteq E'$, $X \subseteq X'$, and $Y \subseteq Y'$.*

Proof Let r be a relation over R and Σ^{-1} the set of all non-eFDs in r.

Case $\Rightarrow$: Suppose r satisfies eFD $E : X \to Y$. Assume there is a non-eFD $E' : X' \not\to Y' \in \Sigma^{-1}$ where $E \subseteq E'$, $X \subseteq X'$ and $Y \subseteq Y'$. Then, there must be tuples $t_1, t_2 \in r^{E'}$ such that $t_1(X') = t_2(X')$ and $t_1(A) \neq t_2(A)$ for all $A \in Y'$. However, $t_1(X) = t_2(X)$ and $t_1(A) \neq t_2(A)$ for all $A \in Y$ since $t_1, t_2 \in r^{E'} \subseteq r^E$, $X \subseteq X'$ and $Y \subseteq Y'$, which contradicts to $r \vDash E : X \to Y$.

Case $\Leftarrow$: Suppose r does not satisfy eFD $E : X \to Y$. There must exist tuples $t_1, t_2 \in r^E$ such that $t_1(X) = t_2(X)$ and $t_1(A) \neq t_2(A)$ for all $A \in Y$.

Example 2 illustrates the generation of new candidate eFDs from eFDs that are contradicted by a valid non-eFD.

Example 2 *Let $ABC : A \to BC$ be a candidate eFD over relation schema $R = ABCDE$. Suppose $ABCD : A \not\to B$ is a valid non-eFD. This non-eFD shows that the eFD $ABC : A \to B$ cannot hold. We can augment this eFD to obtain new candidates for valid eFDs. This can be done by adding attributes from $R - ABC$ to the eFD embedding ABC while keeping the original LHS, resulting in $ABCE : A \to B$; or by adding attributes from $R - A$ to the eFD LHS A while keeping the original embedding, resulting in $ABC : AC \to B$.*

The next example illustrates that some non-eFDs are better than others when generating new candidates for valid eFDs.

Example 3 *Let $ABC : A \not\to B$ and $ABC : AC \not\to B$ be non-eFDs over $R = ABCD$. Both contradict $AB : A \to B$. If the non-eFD $ABC : A \not\to B$ augments the eFD first, the new candidate eFD $ABC : AC \to B$ emerges. However, the second non-eFD still contradicts the new candidate. Since the first non-eFD is implied by the second, the second non-eFD can generate better eFD candidates.*

Just like eFDs imply one another, non-eFDs imply one another, too. Given non-eFD $E : X \not\to Y$, the non-eFDs $E' : X' \not\to Y$ where $E' \subseteq E$ and $X' \subseteq X$ need not be considered. Ignoring those implications will result in redundant candidate eFDs, as Example 3 demonstrates. Fortunately, implied non-eFDs can be easily eliminated using eFD-trees. Algorithm 6 demonstrates how non-implied non-eFDs are computed. The

algorithm firstly sorts all the unique non-eFDs extracted from a relation by the size of their embeddings and LHSs (step 13). By inserting smaller non-eFDs first, it facilitates insertion of larger non-eFDs later (step 16) to remove smaller but implied non-eFDs by utilizing the fast search capabilities of the eFD-tree (step 15). The removal process can be easily adapted from Algorithm 1.

Algorithm 6 non-implied non-eFD

1: **Input: relation** r **over** R
2: **Output:** the set of all non-implied non-eFDs
3: Let $r = \{t_1, t_2, \ldots t_n\}$
4: $\Sigma^{-1} = \emptyset$
5: Let T be an empty eFD-tree
6: **for all** $t_i \in r$ **do**
7: Let $E = \emptyset$, $X = \emptyset$, $Y = \emptyset$
8: **for all** $t_j \in r$ where $i + 1 \leq j \leq n$ **do**
9: $E = \{A \in R \mid t_i(A) \neq \perp \neq t_j(A)\}$
10: $X = \{A \in E \mid t_i(A) = t_j(A)\}$
11: $Y = E - X$
12: $\Sigma^{-1} = \Sigma^{-1} \cup \{E : X \not\to Y\}$
13: Sort Σ^{-1} in ascending size of LHSs, then by that of embeddings
14: **for all** $E : X \not\to Y \in \Sigma^{-1}$ **do**
15: Remove all $E' : X' \not\to A \in T$ for all $A \in Y$ with $E' \subseteq E \wedge X' \subseteq X$
16: Insert $E : X \not\to Y$ into T
17: **return** $\{E : X \not\to Y \in T\}$

Algorithm 7 summarizes our column-efficient strategy. All eFDs and non-eFDs appear in their concise form. First, we compute the set of non-eFDs from the given relation by Algorithm 6. To initialize the eFD-tree, only the most general eFD $\emptyset : \emptyset \to R$ is inserted. Next, each non-eFD is applied iteratively to the eFD-tree to generate new eFD candidates. Any eFD candidate that contradicts a valid non-eFD is removed from the tree. The new candidate eFDs are inserted in the eFD-tree only if they are not implied by eFDs already in the tree. Proposition 4 shows that the augmentation of eFDs will always produce a canonical cover of the valid eFDs based on the valid non-eFDs found so far.

Proposition 4 *Let* $\Gamma^{-1} = \Sigma^{-1} \cup \{E : X \not\to A\}$ *contain all non-eFDs of relation* r *over* R. Γ *and* Σ *are the canonical covers of eFDs for* Γ^{-1} *and* Σ^{-1}*, respectively. For all* $E' : X' \to A' \in \Sigma$*, either (1)* $E' : X' \to A' \in \Gamma$*, or all of (2)-(6) hold: (2)* $E'B : X' \to A'$ *is implied by* Γ *for all* $B \in R - E$*, (3)* $E'B : X'B \to A'$ *is implied by* Γ *for all* $B \in E - X$*, (4)* $E'B : X' \to A' \notin \Gamma$ *for all* $B \in E$*, (5)* $E'B : X'B \to A' \notin \Gamma$ *for all* $B \in X$*, and (6)* $E'B : X'B \to A'$ *is implied by* Γ *for all* $B \in R - E$.

Proof Let $\Gamma^{-1} = \Sigma^{-1} \cup \{E : X \not\to A\}$ be the set of all non-eFDs of relation r over R. Γ and Σ are the canonical cover of eFDs with respect to Γ^{-1} and Σ^{-1}. Take any $E' : X' \to A' \in \Sigma$

If $A \neq A'$ or $E' \not\subseteq E$ or $X' \not\subseteq X$, then $E' : X' \to A' \in \Gamma$ because $E' : X' \to A'$ is already a canonical eFD in Σ and it does not imply non-eFD $E : X \to A$.

Suppose $A = A'$ and $E' \subseteq E$ and $X' \subseteq X$. Take any $B \in R - E$. Since $E' : X' \to A' \in \Sigma$, there exists no $E'' : X'' \not\to A' \in \Sigma^{-1}$ by Proposition 3 where $E' \subseteq E''$ and $X' \subseteq X''$. Knowing $E' \subset E'B$, hence, $E'B : X' \to A'$ is a satisfiable eFD with respect to Σ^{-1}. Furthermore, $E'B : X' \to A'$ is a satisfiable eFD with respect to Γ^{-1} because $B \notin E$ and $E'B \not\subseteq E$. If there exists $E'' : X'' \to A' \in \Sigma$ where $E'' \subseteq E'B$ and $X'' \subseteq X'$, then $E'' \not\subseteq E$ because B must be in E''. If $B \notin E''$ and $E'' \subseteq E'$, then $E'' : X'' \to A'$ and $E' : X' \to A'$ are both in Σ and redundant to each other, which draws a contradiction. So, $E'' \not\subseteq E$ and $E'' : X'' \to A' \in \Gamma$. Namely, $E'B : X' \to A'$ is redundant to Γ. Otherwise, $E'B : X' \to A' \in \Gamma$ if it is not implied by other eFD in Σ. Hence, claim 2 is proven.

Take any $B \in E - XA'$. Since $E' : X' \to A' \in \Sigma$, there exists no $E'' : X'' \not\to A' \in \Sigma^{-1}$ where $E' \subseteq E''$ and $X' \subseteq X''$ by Proposition 3. Knowing $E' \subseteq E'B$ and $X' \subset X'B$, hence, $E'B : X'B \to A'$ is a satisfiable eFD with respect to Σ^{-1}. Furthermore, $E'B : X'B \to A'$ is a satisfiable eFD with respect to Γ^{-1} because $B \notin X$ and $X'B \not\subseteq X$. Suppose there exists $E'' : X'' \to A' \in \Sigma$ where $E'' \subseteq E'B$ and $X'' \subseteq X'B$.

Case $B \in X''$*:* $X'' \not\subseteq X$ implies $E'' : X'' \to A' \in \Gamma$. Therefore, $E'B : X'B \to A'$ becomes redundant to Γ.

Case $B \notin X''$*:* we can conclude that $E'' \subseteq E'B \subseteq E$ and $X'' \subseteq X' \subseteq X, X'B$. Since $E'' \subseteq E$ and $X'' \subseteq X$, $E'' : X'' \to A'$ is not in Γ but $E''B : X''B \to A'$ is satisfiable with respect to Γ^{-1}. Since Σ is finite, we can always choose E'' and X'' where there exits no $E''' \subseteq E''B$ or $X''' \subseteq X''B$ such that $E''' : X''' \to A' \in \Sigma$. To choose such $E'' : X'' \to A$, we can keep replacing $E''' : X''' \to A'$ with $E'' : X'' \to A'$ if $E''' : X''' \to A'$ where $E''' \subseteq E''B$ and $X''' \subseteq X''B$ are found in Σ because $E''' \subseteq E''B \subseteq E'B$ and $X''' \subseteq X''B \subseteq X'B$. By choosing such E'' and X'' with respect to B, it is guaranteed that $E''B, X''B \to A \in \Gamma$, which further concludes that $E'B : X'B \to A'$ is redundant to Γ. Otherwise, $E'B : X'B \to A' \in \Gamma$ if it is not implied by other eFDs in Σ. Hence claim 3 is proven.

Take any $B \in E$. Since $E'B \subseteq E$, $E'B : X' \to A'$ is not a satisfiable eFD with respect to Γ^{-1} so it is not in Γ. Hence, claim 4 is proven.

Take any $B \in X$. Since $E'B \subseteq E$ and $X'B \subseteq X$, $E'B : X'B \to A'$ is not a satisfiable eFD with respect to Γ^{-1} so it is not in Γ. Hence, claim 4 is proven.

Take any $B \in R - E$. If $E'B : X'B \to A'$ is not implied by any other eFDs in Σ, it is still implied by $E'B : X' \to A'$ as $B \notin E'$. In other words, $E'B : X'B \to A' \notin \Gamma$ for all $B \in R - E$. Hence, claim 5 is proven.

Algorithm 7 builds up all the valid eFDs starting from the most general eFD by iteratively augmenting embeddings and LHSs of invalidated eFDs. Correctness is guaranteed by Proposition 4: The output is a canonical cover of eFDs with respect to all non-eFDs that are valid on the input relation.

Theorem 5 *Algorithm 7 computes the canonical cover of all eFDs that are satisfied by the given relation.*

Algorithm 7 Column-efficient algorithm

Input: relation r over R
Output: a cover of all eFDs satisfied by r
Let Σ^{-1} be the set of all non-redundant non-eFDs
Let T be an empty eFD-tree
Insert eFD $\emptyset : \emptyset \to R$ into T
Sort Σ^{-1} by descending size of LHSs, then by that of embeddings
for all $E : X \not\to Y \in \Sigma^{-1}$ **do**
    for all $E' : X' \to Y' \in T$ where $E'X' \subseteq EX$, $X \subseteq X'$, $Y' \subseteq Y$ **do**
        Remove $E' : X' \to Y'$ from T
        for all $A \in R - EXY$ **do**
            if $E'A : X' \to Y'$ is not implied by T **then**
                Insert $E'A : X' \to Y'$ into T
        for all $A \in EY$ **do**
            $E'' = E'$, $Y'' = Y'$
            if $A \notin Y'$ **then**
                $E'' = E''A$
            else
                $Y'' = Y'' - \{A\}$
            if $Y'' = \emptyset$ **then** *continue*
            if $E'' : X'A \to Y''$ is not implied by T **then**
                Insert $E'' : X'A \to Y''$ into T
return $\{E : X \to Y \in T\}$

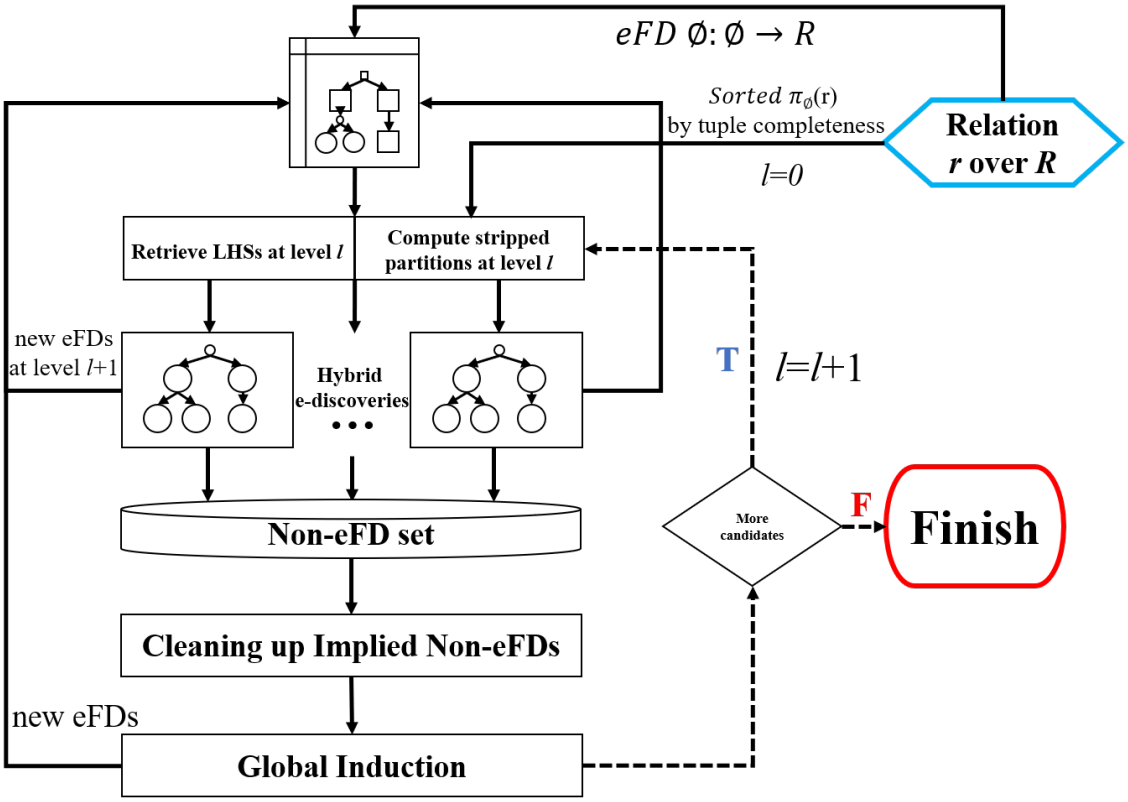

Figure 3: Hybrid discovery algorithm

8 Hybrid Discovery

The row- and column-efficient algorithms are designed for data sets with a large number of either rows or columns. However, real-world data sets typically exhibit larger numbers of rows and columns. We combine the row- and column-efficient algorithms to introduce a *hybrid eFD discovery algorithm* that can handle such real-world data sets.

Overview. Both row- and column-efficient algorithms suffer from drawbacks when handling large data sets. For instance, the row-efficient algorithm does not only need to traverse a large attribute lattice, but also utilize large memory resources to store stripped partitions during the traversal. The column-efficient algorithm needs to process a number of implied non-eFDs that can be quadratic in the number of rows. The trick is to combine the merits of each algorithm to compensate for their drawbacks. In fact, the row-efficient algorithm can use non-eFDs to smartly generate candidate eFDs for the next level, instead of blindly enumerating all candidates. Furthermore, the column-efficient algorithm can directly validate an eFD using the corresponding stripped partition, without checking all non-eFDs.

Figure 3 illustrates how our proposed hybrid algorithm effectively orchestrates the row- and column-efficient algorithms. The hybrid algorithm takes a relation r over schema R as input. Then, it initializes an eFD-tree and a stripped partition based on R and r. At the beginning of an iteration, all of the LHS and RHS candidates at level l of the eFD-tree are retrieved. Meanwhile, a set of stripped partitions for the LHS candidates

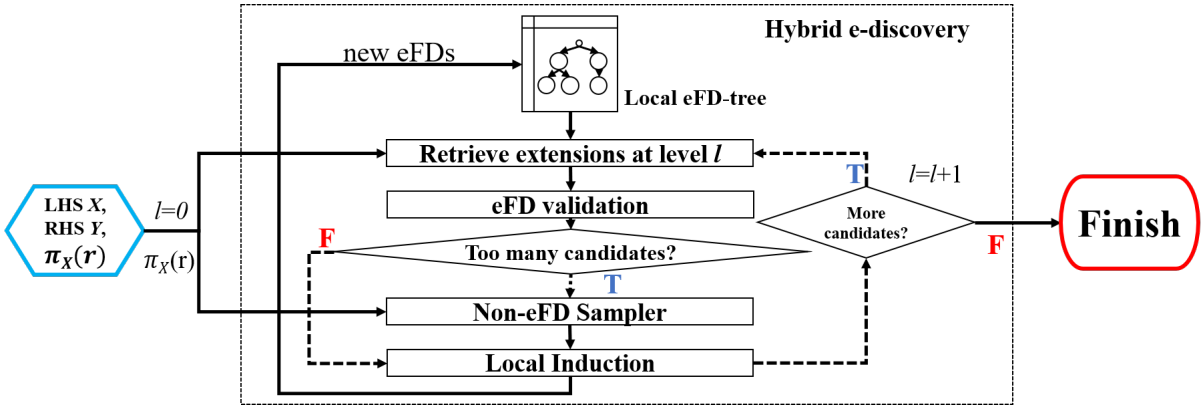

Figure 4: Hybrid e-discovery algorithm

is computed using the stripped partitions from the previous iteration. Next, a *hybrid e-discovery run* is initiated for each pair of LHS and RHS candidates. Once a run has completed, all the embeddings for a given LHS are discovered, the LHSs at level $l+1$ are updated, and all the used non-eFDs are saved. After the non-eFDs are collected from all the runs, they will be refined further by cleaning up implied non-eFDs (similar to Algorithm 6). At the end of an iteration, efficient updates will be made to the eFD-tree using non-eFDs after they have been cleaned up. Eventually, the hybrid algorithm terminates when it has iterated through all levels of the eFD-tree and no more updates can be made.

Hybrid e-discovery. As shown in Figure 3, embeddings of a given pair of LHS and RHS candidates are computed by a separate process called *hybrid e-discovery*. The process discovers the embeddings of a given LHS in a hybrid manner as well. The algorithm iteratively validates candidates for embeddings while updating its local search space (an attribute lattice of embeddings), as shown in Figure 4. The search space of a hybrid e-discovery is a subtree that starts from an e-root. The e-root is easily retrieved since the LHS candidate in the global eFD-tree is given. At the beginning of an iteration, candidate embeddings are retrieved and validated using the given stripped partition. If any violations are detected, the validation process will pass on the valid non-eFDs that cause contradictions to the subsequent processes. After validation, the hybrid e-discovery algorithm decides whether more non-eFDs should be sampled from the stripped partition in order to prune the current search space. This decision depends on the percentage of invalidated eFDs over all the examined candidates. If too many eFDs are invalid, e.g. over 1%, hybrid e-discovery will run a *completeness-driven* non-eFD sampler. At the end of each iteration, all of the non-eFDs from validation and sampling will be used by the local induction process to update the local eFD-tree. The induction process during a hybrid e-discovery is *local*. That is, it only uses non-eFDs to update embeddings of the given LHS and possibly to generate new eFDs whose LHSs have one more attribute than the given LHS candidate.

Hybrid e-discovery implements two heuristics that enable the hybrid algorithm to efficiently handle large data sets (see Section 9). Firstly, non-eFDs are extracted with a completeness-driven sampling method. The core idea is that larger embeddings of a

Algorithm 8 Completeness-driven sampling

Input: a stripped partition $\pi_X(r)$, offset i, relation r over R
Output: non-eFDs
$\Sigma^{-1} = \emptyset$
for all $S \in \pi_X(r)$ **do**
    Let $S = \{t_1, \ldots, t_n\}$
    if $i > 1$ and $t_1(A) \neq \perp$ for $A \in R$ **then** *continue*
    for each $j > i$ and $j \leq n$ **do**
        $E = \{A \in R \mid t_i(A) \neq \perp \neq t_j(A)\}$
        $X' = \{A \in E \mid t_i(A) = t_j(A)\}$
        $Y = \{A \in E \mid t_i(A) \neq t_j(A)\}$
        $\Sigma^{-1} = \Sigma^{-1} \cup \{(E', X \not\to Y)\}$
return Σ^{-1}

non-eFD generate fewer eFD candidates when eFDs with a fixed LHS are updated, as shown in Example 2. In fact, non-eFDs with larger embeddings and LHSs generate better candidates for valid eFDs. Since the LHS candidate is fixed and the stripped partition of the LHS is provided, one can easily find efficient non-eFDs to update the given embeddings by examining the pairs of tuples with fewer missing values. Algorithm 8 demonstrates a single run of completeness-driven sampling. As mentioned in Figure 3, all tuples in $\pi_\emptyset(r)$ are sorted by the number of null marker occurrences, that is tuples with fewer missing values will be sampled first. Although stripped partitions are generated incrementally, tuples in the stripped partitions will remain in the same order. Therefore, Algorithm 8 always finds non-eFDs with larger embeddings first. If the non-eFD sampler is invoked during hybrid e-discovery, it keeps running until the number of newly sampled non-eFDs is below a certain threshold, where the initial threshold is 1.0. Such threshold represents a point when there are sufficiently many non-eFDs to update the current search space effectively. If the non-eFD sampler is invoked again during the same run of hybrid e-discovery, the previous sampling is not sufficient and the current threshold will be halved before the non-eFD sampler is executed. In this way, more new non-eFDs can be used for pruning when there are too many invalidated candidates that will blow up the search space. Secondly, there is no absolutely good way to find non-eFDs that can efficiently update LHSs of eFD candidates because LHSs of eFDs are normally domain dependent. In order to improve the efficiency of updating LHSs, hybrid e-discovery will not update eFDs globally using valid non-eFDs. Instead, all hybrid runs of e-discovery in the same iteration will be coordinated by a cleaning method (as shown in Figure 3) which removes all the implied non-eFDs. Overall, the first heuristics dramatically reduces the number of steps to discover valid eFDs, and the second heuristics saves memory costs by preventing excessive updates on the global eFD-tree.

The hybrid algorithm, as illustrated in Figure 3, performs a breath-first search to traverse all eFDs. During the traversals, eFDs are validated along the way using the row-efficient algorithm. When invalid eFDs are identified, the eFD-tree will be updated according to a set of validated non-eFDs. The correctness of the hybrid algorithm follows from that of the row- and column-efficient algorithms from Section 6 and 7.

9 Experiments

In this section, we present our experimental results about the performance of the proposed algorithms on real-world benchmark data. Particularly, we will show an in-depth analysis of the runtime, memory consumption, and scalability of our algorithms. We implemented all algorithms in Visual C++, and ran all the experiments on an Intel Xeon 3.6 GHz, 256 GB RAM, Windows 10 Dell workstation. The benchmark data is from the UCI machine learning data repository[1] and previous work on discovery algorithms [23]. The data sets are available at `https://bit.ly/2N0pTOS?`.

Runtime. Table 4 shows the runtime of our eFD discovery algorithms on the benchmark data. For each data set, we show its numbers of rows (#R), columns (#C), rows (#IR) and columns (#IC) with null marker occurrence, null marker occurrences (#$\perp$), eFDs in the canonical cover (#eFDs), and the runtimes. While eFDs are different from FDs and from eUCs (none of them are competitors), we still show the number of FDs (#FDs) and the runtime of state-of-the-art algorithms for FD discovery [24, 29], and similar for eUCs [28]. For data with few rows, the column-efficient algorithm typically performs better. Particularly, this algorithm yields much better performance for large numbers of columns, such as *plista, flight*. Although the row-efficient algorithm does not perform better than others in any case, it can still process a large data set with over 260,000 rows (*china*) much faster than the column-efficient algorithm. The hybrid algorithm does not always outperform the other two since it is primarily designed to handle large data sets, such as *diabetic, china, uniprot*. As expected, the performance of the hybrid algorithm are satisfying. For example, it takes less than 15 minutes to process *uniprot* with half a million rows and 30 columns. The comparison to FDs shows how many more eFDs there can be, and that the additional time that our algorithms require to discover eFDs is well justified by the higher number of eFDs that hold on the data sets. These observations are very true for eUCs, too.

Memory use. We also show the memory resources (in MB) consumed by the algorithms in Table 4. The row-efficient algorithm is not memory-friendly since it uses many stripped partitions for eFD validation. On the other hand, the hybrid algorithm takes advantage of the heuristics implemented for hybrid e-discovery runs, so that it is quite memory-friendly even though stripped partitions are used. For example, the hybrid algorithm's memory use is either less or only slightly more than that of the column-efficient algorithm. The higher memory use over eUC discovery algorithms is fully justified by the output size alone.

Runtime scalability. To further demonstrate the practicality of the three algorithms, the left of Figure 5 shows how each of the algorithms scale in terms of the numbers of rows and columns. The data sets *china* and *ncvoter* both start with 1,000 rows. The hybrid algorithm and the row-efficient algorithms have comparable performance in both experiments up to 60,000 rows. The column-efficient algorithm is dramatically outperformed by the other two after 10,000 rows. To demonstrate column-scalability, we use the data set *diabetic* with only 10,000 rows, but where the number of columns ranges from 5 to 45. The row-efficient algorithm performs only well up to 20 columns.

[1] https://archive.ics.uci.edu/ml/

Table 4: Runtime of eFDs, FD, and eUC discovery from real-world benchmark data

Data Sets and their Properties					
Name	#R	#C	#IR	#IC	#⊥
bridges	108	13	38	9	77
echo	132	13	71	12	132
hepatitis	155	20	75	15	167
horse	300	28	294	21	1605
breast	691	11	16	1	16
plista	996	63	996	34	23317
ncvoter	1000	19	1000	5	2863
flight	1000	109	1000	69	51938
diabetic	101766	30	100732	7	192849
china	262920	18	157895	12	418580
uniprot	512000	30	192849	19	3759296
pdbx	17305799	13	683410	6	2035242

Data Sets	eFD				FD			eUC	
Name	#eFDs	Alg. 2	Alg. 7	Hybrid	#*FD*s	HyFD	DHyFD	#*eUC*s	Best
bridges	169	0.021	**0.014**	0.035	142	0.1	0.003	3	0.002
echo	438	0.034	**0.013**	0.028	527	0.1	0.002	45	0.006
hepatitis	11087	322.107	**0.287**	0.431	8250	0.6	0.189	446	0.082
horse	1182385	>4H	**56.371**	72.704	128727	7.1	2.595	5040	1.046
breast	48	0.029	0.167	**0.044**	46	0.2	0.009	2	0.009
plista	298806	>4H	**27.569**	55.818	178152	21.8	15.403	2337	3.369
ncvoter	1443	0.132	0.514	**0.082**	758	0.4	0.029	147	0.067
flight	2597244	>4H	**9.698**	40.88	982631	53.4	9.934	26652	49.367
diabetic	407767	>4H	>4H	**7596.09**	40195	N/a	847.582	20130	1239.25
china	8320	751.277	>4H	**131.247**	918	N/a	49.839	615	77.365
uniprot	194742	>4H	>4H	**727.108**	3703	N/a	75.442	9480	529.478
pdbx	114	ML	>4H	**635.754**	68	240	100.906	15	512.492

Table 5: Memory use (MB) of eFD, FD, eUC discovery

Data Set	Alg. 2	Alg 7	Hybrid	FD	eUC
bridges	0.87	0.92	**0.79**	0.73	1
echo	1.1	0.76	**0.70**	0.76	1
hepatitis	111	**15.6**	16.9	14	3
horse	N/a	**1126.4**	1228.8	268	15
breast	5	1.3	**0.86**	1	1
plista	N/a	**507**	783.5	2048	23
ncvoter	8	4	**3**	3	5
flight	N/a	**1024**	1433.6	2048	179
diabetic	N/a	N/a	**20480**	4300	1024
china	15974.4	N/a	**649**	1024	819
uniprot	N/a	N/a	**6246.4**	4608	4403
pdbx	N/a	N/a	**4096**	6451	35532

The hybrid algorithm starts gaining more advantage than the column-efficient algorithm after the data sets with more than 35 columns. Interestingly, the runtime of the hybrid algorithm exhibits a similar but less steep pattern as the number of eFDs that hold on the data sets.

Memory scalability. In addition, we show the memory scalability of the algorithms in the right of Figure 5 for the same data sets. The hybrid algorithm is more memory-friendly. On one hand, it reduces much more search space and uses fewer stripped partitions than the row-efficient algorithm. On the other hand, the hybrid algorithm efficiently samples non-eFDs from the data while the column-efficient algorithm computes all the non-eFDs from the data. Table 4 shows that the number of non-eFDs grows quadratically in the number of rows of the given data set, which increases the memory consumed by the column-efficient algorithm.

Summary. Our experiments with real-world benchmark data illustrate that our different eFD discovery algorithms perform well for the purpose they have been designed for. Specifically, the hybrid algorithm is able to combine the strengths of the row- and column-efficient algorithms to efficiently process larger data sets, and scale well in terms of growing numbers of rows, columns, and eFDs as far as runtime and memory consumption are concerned.

10 Helping aFD and gFD Analysis

We provide quantitative evidence that i) pivoting along the data completeness dimension, and ii) ranking by the number of redundant data values, helps with aFD and gFD analysis on data with missing values.

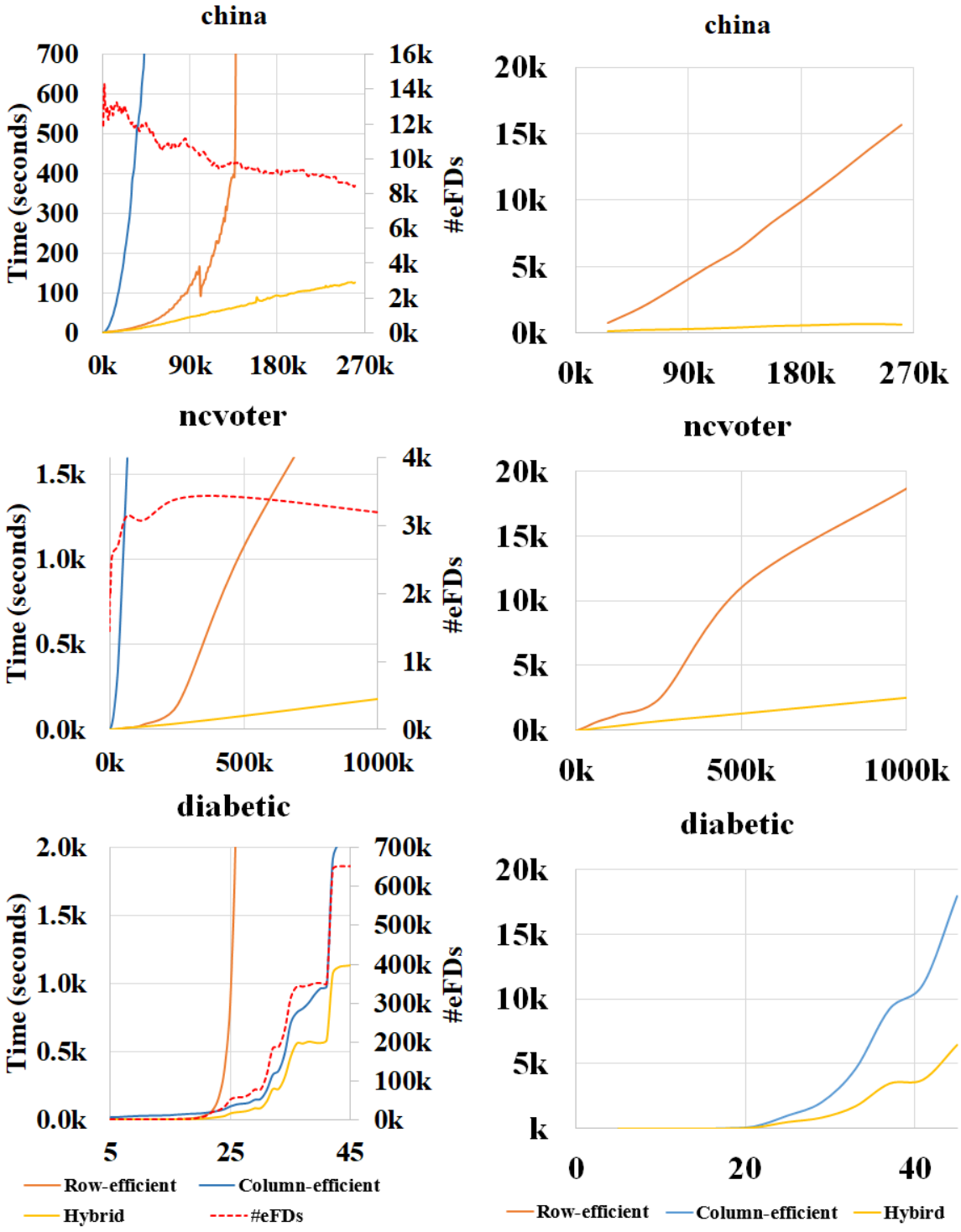

Figure 5: Time/Memory scalability for discovery

10.1 Pivoting along data completeness

It is very natural to pivot aFD and gFD analysis along the data completeness requirements by an application. For data with missing values it is helpful to distinguish approximation error ratios ϵ and levels of genuineness γ by applying them to sets of tuples that meet the requirements, and to those that do not. Hence, in addition to the measures under $\perp=\perp$ (EQ) and $\perp\neq\perp$ (NEQ) null semantics applied to the data set r (denoted by ϵ_r^{EQ}, ϵ_r^{NEQ}, γ_r^{EQ}, γ_r^{NEQ}, respectively), we are also interested in those measures applied to the application-relevant part r^E and the application-irrelevant part $r - r^E$. In fact, if all the attributes of the FDs under consideration are part of E, then $\epsilon_{r^E}^{EQ} = \epsilon_{r^E}^{NEQ}$ $(= \epsilon_r^E)$ and $\gamma_{r^E}^{EQ} = \gamma_{r^E}^{NEQ}$ $(= \gamma_r^E)$. In this case, the measures quantify the errors from complete values on r^E, from nulls under different semantics on $r - r^E$, and their interaction on r.

As illustration consider the FD *city, full_phone_num* $\rightarrow$ *zip_code* from the perspective of aFDs and gFDs. As an aFD it has $\epsilon_r^{EQ} = 0.00598$ and $\epsilon_r^{NEQ} = 0.000198$, which would rank it at positions 1184 and 252, respectively. As a gFD it has $\gamma_r^{EQ} = 0.896$ and $\gamma_r^{NEQ} = 0.991$, which rank it at positions 562 and 106, respectively. However, if the completeness requirements E consist of attributes *name_suffix*, *city*, *full_phone_num*, and *zip_code*, then $\epsilon_{r^E} = 0$ and $\gamma_{r^E} = 1$, and the aFD/gFD ranks at position 1, respectively. This illustrates how completeness requirements can inform the analysis of aFDs and gFDs.

We have also conducted quantitative experiments to illustrate the impact of the different measures for both aFDs and gFDs. Using the output of our discovery algorithms in the form of eFDs $E : X \rightarrow Y$, we removed attributes from the LHS X of the FD $X \rightarrow Y$ until the error ratio of the resulting FD $X' \rightarrow Y$ met a given threshold θ on r^E. In the experiments θ started from $0.001, 0.02, 0.04, \ldots, 0.1$. Figure 6 shows the average number of violations in the y-axis for the FDs that meet threshold θ on the x-axis. Here we distinguish between violations on r^E, and on r and $r - r^E$ based on EQ and NEQ semantics. The results show that null markers themselves, but also the way they are interpreted, have a large impact on the measures. The results suggest that violations are less caused within the application-relevant part r^E or within the application-irrelevant part $r - r^E$, but more so across those parts between nulls and domain values.

Similarly, Figure 7 shows the level of genuineness in the y-axis for the FDs that meet the threshold θ on the x-axis. Whenever the level is lower on r than on r^E, the number of null marker occurrences must exceed that of some domain value. Indeed, the genuineness for NEQ is typically smaller than that for EQ, on both r and $r - r^E$ alone. Under EQ semantics, occurrences of $\perp$ result likely in underestimations of the actual levels. If genuineness is higher on r than on r^E it means that by adding $r - r^E$ to r^E, less errors occur relative to all the tuples under consideration. In particular, $\perp$ occurrences do not result in errors when compared to one another (NEQ). Hence, pivoting aFD and gFD analysis along data completeness requirements is helpful for applications, such as data cleaning.

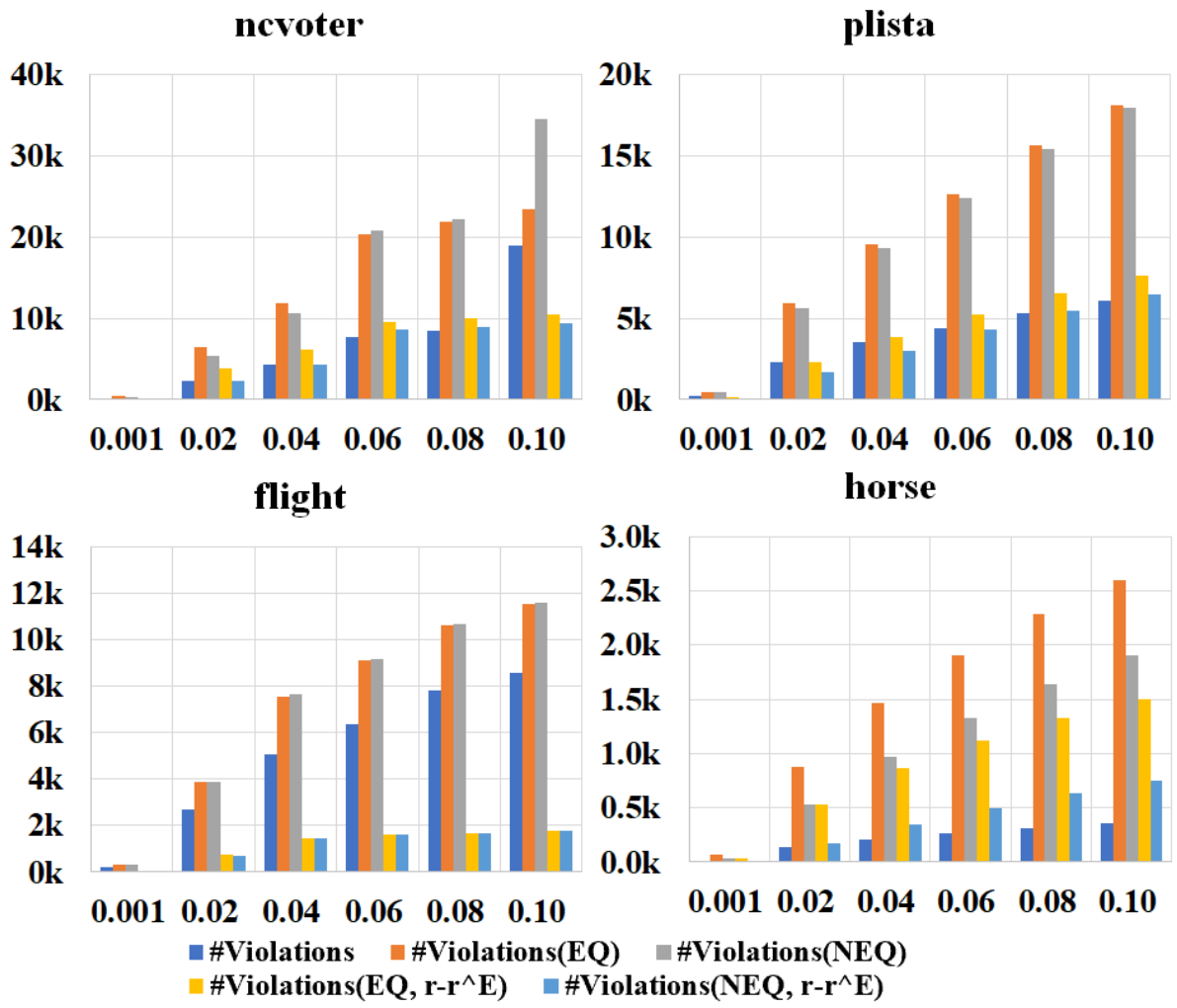

Figure 6: Difference in numbers of violations on benchmarks for various measures of approximation

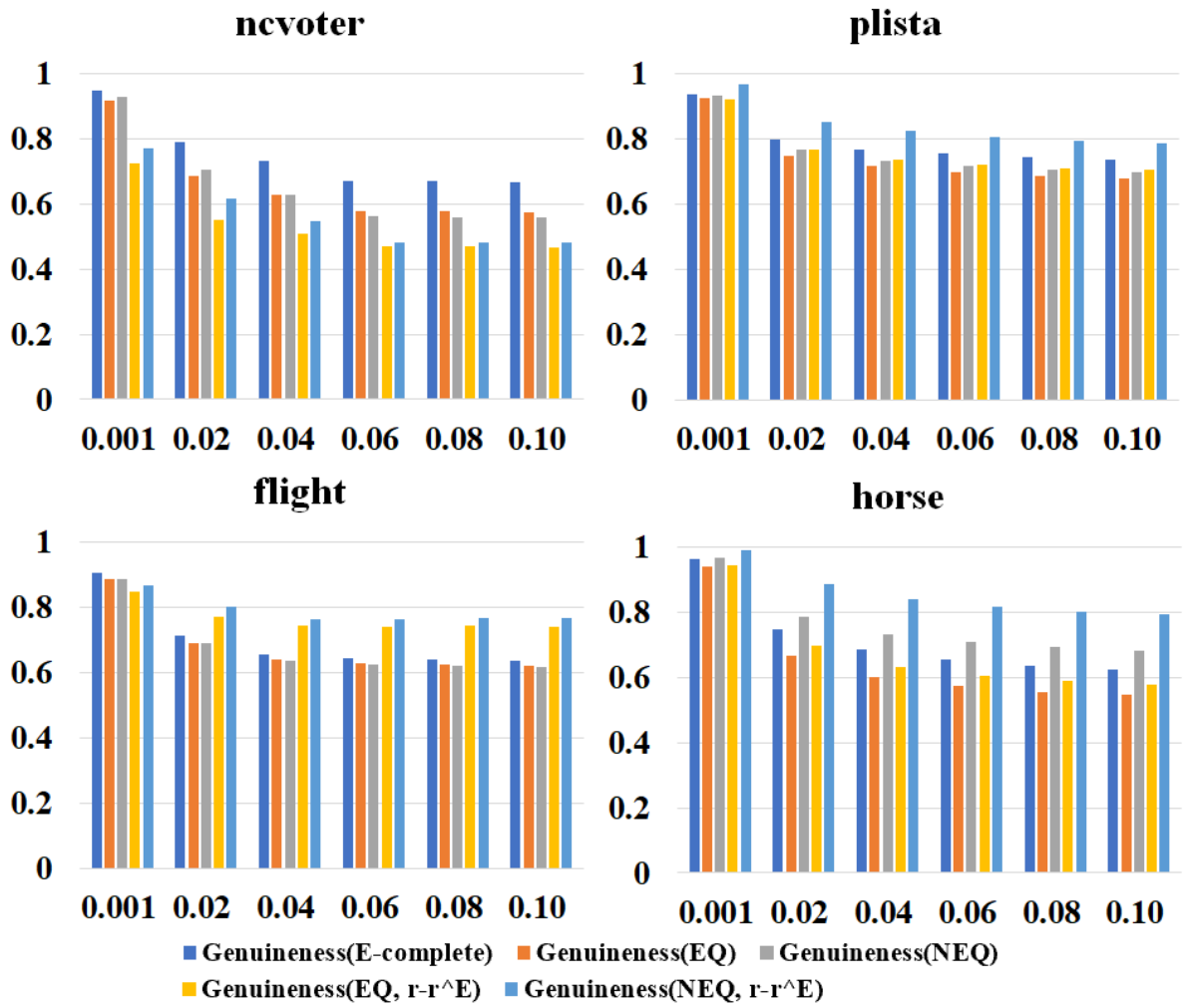

Figure 7: Difference in levels of genuineness on benchmarks for various measures of genuineness

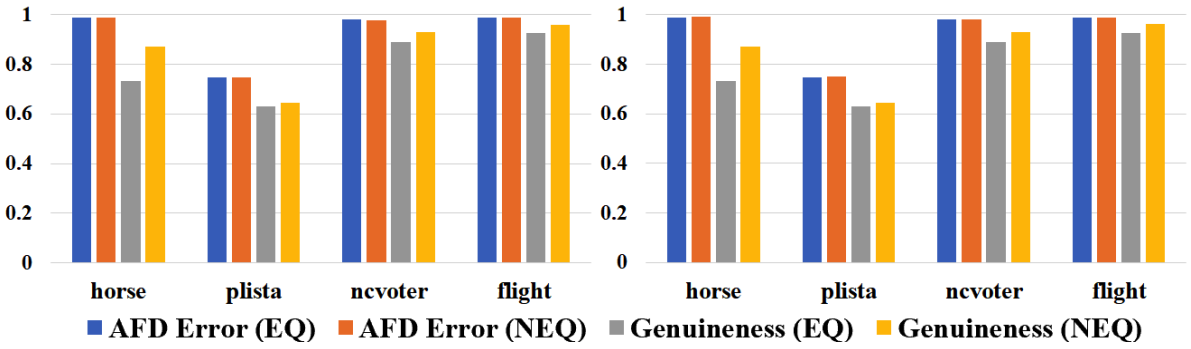

Figure 8: Neither top-a- nor top-g-ranked FDs can identify bottom-e-ranked FDs

10.2 Ranking Data Redundancy

We will provide some quantitative evidence that ranking the number of redundant data values adds another dimension to aFD and gFD analysis. A qualitative example for this is given in the next section on applications.

In general it is clear from the definition of approximation error ratios and levels of genuineness that small error ratios or high levels of genuineness do not imply high numbers of redundant data values, and vice versa. In our experiments we will quantify experimentally that even top-ranked aFDs (gFDs) cannot distinguish between FDs according to the number of redundant data values they cause. For the eFDs $E : X \to Y$ we discovered on the benchmarks, we rank the FDs $X \to Y$ based on the number of redundant data values they cause in r^E (e-ranking), and based on their various error ratios (a-ranking) and levels of genuineness (g-ranking) in r and $r - r^E$. We further refer to the top-20% and the bottom-20% of FDs in the e-ranking by t_e and b_e, and to the top-20% of the FDs in a- and g-rankings by t_a and t_g, respectively.

In the first experiment we are interested in how many of the bottom-e-ranked FDs and top-a-ranked FDs overlap, that is, how many of the FDs in $b_e \cup t_a$ are in $b_e \cap t_a$. We compare this to how many of all the e-ranked FDs are contained in both the bottom-e-ranked and the top-a-ranked FDs, that is, how many of the FDs in the e-ranking are in $b_e \cap t_a$. As it turns out, the first ratio $|b_e \cap t_a|/|b_e \cup t_a|$ is very similar to the second ratio $|b_e \cap t_a|/|\textit{e-ranking}|$, which suggests that even the top-a-ranked FDs cannot distinguish between the bottom-e-ranked FDs and any e-ranked FDs. In the second experiment, we compute similar ratios for the top-e-ranked FDs. The ratios $|t_e \cap t_a|/|t_e \cup t_a|$ and $|t_e \cap t_a|/|\textit{e-ranking}|$ are very similar again, suggesting that top-a-ranked FDs cannot distinguish between bottom-e-ranked and any e-ranked FDs.

Figure 8 shows the various ratios for the first experiment on four benchmarks. Figure 9 shows the same for the second experiment. A common observation is that top-a-ranked and top-g-ranked FDs can identify neither bottom-e-ranked nor top-e-ranked FDs. A difference is that the former occur frequently among top-a-ranked and among top-g-ranked FDs, while the latter occur sparingly among those.

The experiments provide quantitative evidence that ranking by the number of redundant data values adds a new dimension to aFD and gFD analysis.

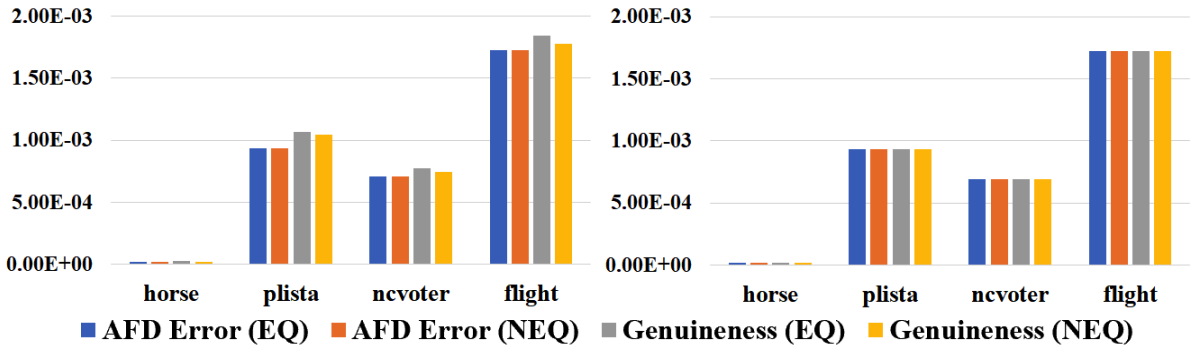

Figure 9: Neither top-a- nor top-g-ranked FDs can identify top-e-ranked FDs

11 Applications

We describe in general how discovered eFDs can be applied to schema design for data with missing values and data cleaning, and provide illustrative examples from the *ncvoter* data set.

11.1 Schema Design for Incomplete Data

We show how some of the eFDs discovered from the real-world census data set *North Carolina Voters (ncvoters)* can help with schema design for data with missing values [15, 16, 20, 30]. The data set suffers from missing values such as phone numbers and addresses, and eFDs are targeted at such data. In general, an eFD $E : X \to Y$ on R can horizontally decompose a given relation into its application-relevant part r^E, consisting of all the tuples that meet the completeness requirements in E, and into the application-irrelevant part $r - r^E$. Since the FD $X \to Y$ holds on r^E, this part can be vertically decomposed into $r^E[XY]$ and $r^E[X(R - Y)]$ without loss of information. For illustration purposes let us only consider the attributes *i(d)*, *a(ddress)*, *c(ity)*, *z(ip)*, and *p(hone)*. The eFD $cpz : z \to c$ constitutes an interesting output of our discovery algorithms because it ranks 23rd with respect to the 259,453 redundant data values it causes on *ncvoter* with 19 columns and over 1 million rows. The completeness requirements on $E = cpz$ shift our focus on the set r^{cpz} of those records from any database instance r that have no missing values on c, p, and z. Since the FD $z \to c$ holds on r^{cpz} we can vertically decompose it into $r^{cpz}[zc]$ and$r^{cpz}[iazp]$, thereby eliminating redundant data value occurrences on c caused by this FD. So far, this is an example for normalizing a schema. For illustrative purposes, Figure 10 shows how the normalization is applied to a small snippet of *ncvoter*. As aFD $z \to c$ has rank 2848 with error ratio 0.00005477234 on r^E, and as gFD it has rank 2149 with genuineness 0.992475 on r^E.

11.2 Data Cleaning

Another interesting application for eFDs is data cleaning. The eFDs $E : X \to Y$ that we discover hold exactly on the given data set r. This means there are no violations of the FD $X \to Y$ on r^E. Our discovery algorithms return eFDs where the LHS X

id	address	city	zip	phone
731	858 stoney creek church rd	burlington	27217	
732	1809 n ncv hwy 49	green level	27217	
287	4190 rascoe dameron rd	burlington	27217	336 327 3712
525	1334 graham st	burlington	27217	226 4719
992	5951 n nc hwy 62	burlington	27217	226 3592
137	326 meadowbrook dr	burlington	27217	336 228 7758

Figure 10: Snippet over unnormalized schema

city	zip
burlington	27217

id	address	zip	phone
287	4190 rascoe dameron rd	27217	336 327 3712
525	1334 graham st	27217	226 4719
992	5951 n nc hwy 62	27217	226 3592
137	326 meadowbrook dr	27217	336 228 7758

Figure 11: Decomposition with $cpz : z \to c$ into application-relevant part

is minimal for the embedding E. That is, for any proper subsets X' of X the resulting FD $X' \to Y$ will have some violations on r^E. FDs $X' \to Y$ that cause a high number of redundant data values are more likely to be meaningful FDs, which means violations of it constitute dirty data. As we have seen, a small number of errors (that is, a small error ratio or a high level of genuineness) is no guarantee that the FD causes a high number of redundant data values. In fact, for any redundant data value to occur we require different tuples with matching values on the LHS X' of an FD, but this is not guaranteed by high ranked aFDs or gFDs. Hence, ranking by redundant data values is also very interesting for aFDs and gFDs. Consider the eFD *birth_place* : *gender*, *street_address*, *zip_code*, *full_phone_num* $\to$ *city* for illustration. It holds on *Ncvoter1m* with $1,024,000$ rows and 19 columns. The eFD only causes 4,642 redundant data values. However, the eFD *birth_place* : *street_address*, *zip_code* $\to$ *city* exhibits $378,950$ redundant data values and is only violated by 64 records. Upon examination, incorrect data values are found in colum *city*. For example, the table

voter_ id	gender	street_ address	zip_ code	city	birth_ place	full_phone _num
741992	f	8374 NC 304	28515	Bayboro	nc	2527457099
403835	f	8374 NC 304	28515	Mesic	nc	2527453967
226136	m	8374 NC 304	28515	Bayboro	nc	$\perp$

shows inconsistent information. In fact, the street address is located in *Mesic* and not

id	address	city	zip
731	858 stoney creek church rd	burlington	27217
732	1809 n ncv hwy 49	green level	27217

Figure 12: Application-irrelevant part of snippet

in *Bayboro*. The eFD *birth_place* : *street_address*, *zip_code* → *city* ranks number 18 with 378,944 redundant data value occurrences on r^E. The following table lists the various levels of genuineness and error ratios for the FD *street_address*, *zip_code* → *city*. It rank much lower for these measures, and is likely not considered for cleaning.

measure	*value of measure*	*rank by measure*
γ_{r^E}	0.9999613648251976	879
γ_r^{EQ}	0.99996679686259	857
γ_r^{NEQ}	0.99996679686259	1602
ϵ_{r^E}	1.1804353519410762e-10	920
ϵ_r^{EQ}	7.24793188274598e-11	850
ϵ_r^{NEQ}	7.24793188274598e-11	1594

We conclude that dirty data can be identified by computing eFDs $E : X' \to Y$ that cause high numbers of redundant data values, and where X' is a proper subset of X for an eFD $E : X \to Y$ discovered by our algorithms. Alternatively, we can also apply our ranking by redundant data values to identify aFDs and gFDs that help with data cleaning.

12 Conclusion and Future Work

Embedded FDs address schema design for data with missing values, and incorporate completeness requirements of applications. They provide a robust semantics for FDs in real-world data sets, since it does not depend on how missing values are interpreted. The discovery of eFDs from a given data set is a core task of data profiling and fundamental to the success of applications. Due to the large search space of eFDs, new search strategies and data structures are necessary to derive efficient solutions to the eFD discovery problem. We have introduced the first three algorithms that solve eFD discovery. The algorithms work as a toolkit in practice: The row- and column-efficient algorithms work well when there are few rows or columns, and the hybrid algorithm scales well in terms of time and space on data sets with many rows and columns. The eFDs in the output of our algorithms are ranked to identify those that are more meaningful for applications, such as schema design and data cleaning. We have demonstrated that adding completeness requirements and ranking by redundant data values adds new dimensions to the analysis of other notions of relaxed FDs, such as approximate and genuine FDs. In the future, we will explore automatic database and data warehouse designs based on eFDs. Similarly, we envision a data quality management framework based on eFDs and other dependencies, including other notions of relaxed uniqueness constraints and functional dependencies [3, 4, 6, 12, 14, 17, 18, 19, 25, 28].

References

[1] Z. Abedjan, L. Golab, F. Naumann, and T. Papenbrock. *Data Profiling*. Synthesis Lectures on Data Management. Morgan & Claypool Publishers, 2018.

[2] Z. Abedjan, P. Schulze, and F. Naumann. DFD: efficient functional dependency discovery. In *CIKM*, pages 949–958, 2014.

[3] N. Balamuralikrishna, Y. Jiang, H. Koehler, U. Leck, S. Link, and H. Prade. Possibilistic keys. *Fuzzy Sets Syst.*, 376:1–36, 2019.

[4] L. Berti-Équille, H. Harmouch, F. Naumann, N. Novelli, and S. Thirumuruganathan. Discovery of genuine functional dependencies from relational data with missing values. *PVLDB*, 11(8):880–892, 2018.

[5] T. Bläsius, T. Friedrich, and M. Schirneck. The parameterized complexity of dependency detection in relational databases. In *LIPIcs-Leibniz International Proceedings in Informatics*, volume 63. Schloss Dagstuhl-Leibniz-Zentrum fuer Informatik, 2017.

[6] P. Brown and S. Link. Probabilistic keys. *IEEE Trans. Knowl. Data Eng.*, 29(3):670–682, 2017.

[7] L. Caruccio, V. Deufemia, and G. Polese. Relaxed functional dependencies - A survey of approaches. *IEEE Trans. Knowl. Data Eng.*, 28(1):147–165, 2016.

[8] J. Demetrovics, G. O. H. Katona, D. Miklós, and B. Thalheim. On the number of independent functional dependencies. In *FoIKS*, pages 83–91, 2006.

[9] P. A. Flach and I. Savnik. Database dependency discovery: A machine learning approach. *AI Commun.*, 12(3):139–160, 1999.

[10] J. Gallier. *Discrete mathematics*. Universitext. Springer, New York, 2011.

[11] C. Giannella and C. Wyss. Finding minimal keys in a relation instance, 1999.

[12] S. Hartmann, U. Leck, and S. Link. On codd families of keys over incomplete relations. *Comput. J.*, 54(7):1166–1180, 2011.

[13] Y. Huhtala, J. Kärkkäinen, P. Porkka, and H. Toivonen. TANE: an efficient algorithm for discovering functional and approximate dependencies. *Comput. J.*, 42(2):100–111, 1999.

[14] H. Köhler, U. Leck, S. Link, and X. Zhou. Possible and certain keys for SQL. *VLDB J.*, 25(4):571–596, 2016.

[15] H. Köhler and S. Link. SQL schema design: Foundations, normal forms, and normalization. In *Proceedings of the 2016 International Conference on Management of Data, SIGMOD Conference 2016, San Francisco, CA, USA, June 26 - July 01, 2016*, pages 267–279, 2016.

[16] H. Köhler and S. Link. SQL schema design: foundations, normal forms, and normalization. *Inf. Syst.*, 76:88–113, 2018.

[17] H. Köhler, S. Link, and X. Zhou. Possible and certain SQL keys. *PVLDB*, 8(11):1118–1129, 2015.

[18] H. Köhler, S. Link, and X. Zhou. Discovering meaningful certain keys from incomplete and inconsistent relations. *IEEE Data Eng. Bull.*, 39(2):21–37, 2016.

[19] S. Kruse and F. Naumann. Efficient discovery of approximate dependencies. *PVLDB*, 11(7):759–772, 2018.

[20] S. Link and H. Prade. Relational database schema design for uncertain data. *Inf. Syst.*, 84:88–110, 2019.

[21] S. Lopes, J. Petit, and L. Lakhal. Efficient discovery of functional dependencies and Armstrong relations. In *EDBT*, pages 350–364, 2000.

[22] N. Novelli and R. Cicchetti. Functional and embedded dependency inference: a data mining point of view. *Inf. Syst.*, 26(7):477–506, 2001.

[23] T. Papenbrock, J. Ehrlich, J. Marten, T. Neubert, J. Rudolph, M. Schönberg, J. Zwiener, and F. Naumann. Functional dependency discovery: An experimental evaluation of seven algorithms. *PVLDB*, 8(10):1082–1093, 2015.

[24] T. Papenbrock and F. Naumann. A hybrid approach to functional dependency discovery. In *SIGMOD*, pages 821–833, 2016.

[25] T. Roblot, M. Hannula, and S. Link. Probabilistic cardinality constraints - validation, reasoning, and semantic summaries. *VLDB J.*, 27(6):771–795, 2018.

[26] Y. Sismanis, P. Brown, P. J. Haas, and B. Reinwald. GORDIAN: efficient and scalable discovery of composite keys. In *VLDB*, pages 691–702, 2006.

[27] P. Stănică. Good lower and upper bounds on binomial coefficients. *JIPAM. J. Inequal. Pure Appl. Math.*, 2(3):Article 30, 5, 2001.

[28] Z. Wei, U. Leck, and S. Link. Discovery and ranking of embedded uniqueness constraints. *PVLDB*, 12(13):2339–2352, 2019.

[29] Z. Wei and S. Link. Discovery and ranking of functional dependencies. In *ICDE*, pages 1526–1537, 2019.

[30] Z. Wei and S. Link. Embedded functional dependencies and data-completeness tailored database design. *PVLDB*, 12(11):1458–1470, 2019.

[31] C. M. Wyss, C. Giannella, and E. L. Robertson. FastFDs: A heuristic-driven, depth-first algorithm for mining functional dependencies from relation instances. In *DaWaK*.

[32] H. Yao, H. J. Hamilton, and C. J. Butz. Fd_mine: Discovering functional dependencies in a database using equivalences. In *ICDM*, pages 729–732, 2002.