Provenance-aware Discovery of Functional Dependencies on Integrated Views

Ugo Comignani
Tyrex team, Grenoble INP, INRIA, France
ugo.comignani@inria.fr

Laure Berti-Equille
IRD, ESPACE-DEV
Montpellier, France
laure.berti@ird.fr

Noël Novelli
Aix-Marseille Univ.
LIS CNRS, France
noel.novelli@lis-lab.fr

Angela Bonifati
Lyon 1 University
Lyon, France
angela.bonifati@univ-lyon1.fr

2022 IEEE 38th International Conference on Data Engineering (ICDE) | 978-1-6654-0883-7/22/$31.00 ©2022 IEEE | DOI: 10.1109/ICDE53745.2022.00051

***Abstract*—The automatic discovery of functional dependencies (FDs) has been widely studied as one of the hardest problems in data profiling. Existing approaches have focused on making the FD computation efficient while inspecting single relations at a time. In this paper, for the first time we address the problem of inferring FDs for multiple relations as they occur in integrated views by solely using the functional dependencies of the base relations of the view itself. To this purpose, we leverage logical inference and selective mining and show that we can discover most of the exact FDs from the base relations and avoid the full computation of the FDs for the integrated view itself, while at the same time preserving the lineage of FDs of base relations. We propose algorithms to speedup the inferred FD discovery process and mine FDs on-the-fly only from necessary data partitions. We present InFine (*INferred FunctIoNal dEpendency*), an end-to-end solution to discover inferred FDs on integrated views by leveraging provenance information of base relations. Our experiments on a range of real-world and synthetic datasets demonstrate the benefits of our method over existing FD discovery methods that need to rerun the discovery process on the view from scratch and cannot exploit lineage information on the FDs. We show that InFine outperforms traditional methods necessitating the full integrated view computation by one to two order of magnitude in terms of runtime. It is also the most memory efficient method while preserving FD provenance information using mainly inference from base table with negligible execution time.**

I. Introduction

The automatic discovery of all functional dependencies (FDs) holding in a single relation is amongst the hardest problems in data profiling [1], [25]. Typically, an FD $X \rightarrow Y$ with attribute sets X and Y in a given table allows to enforce that the combination of values in the set X uniquely determines the values of every attribute in the set Y. Functional dependencies are key ingredients in database design, table decomposition, database normalization, and for several other data management tasks, such as data cleaning [29] and query optimization [14], [27]. Due to the high complexity of FD discovery [20] (exponential in the number of attributes and quadratic in the number of records of a relation), a wealth of algorithms have been proposed by relying on aggressive pruning and sophisticated validation. Many of these algorithms [12], [13], [30], [22], [26] have the same theoretical complexity and are able to compute all minimal FDs in less runtime.

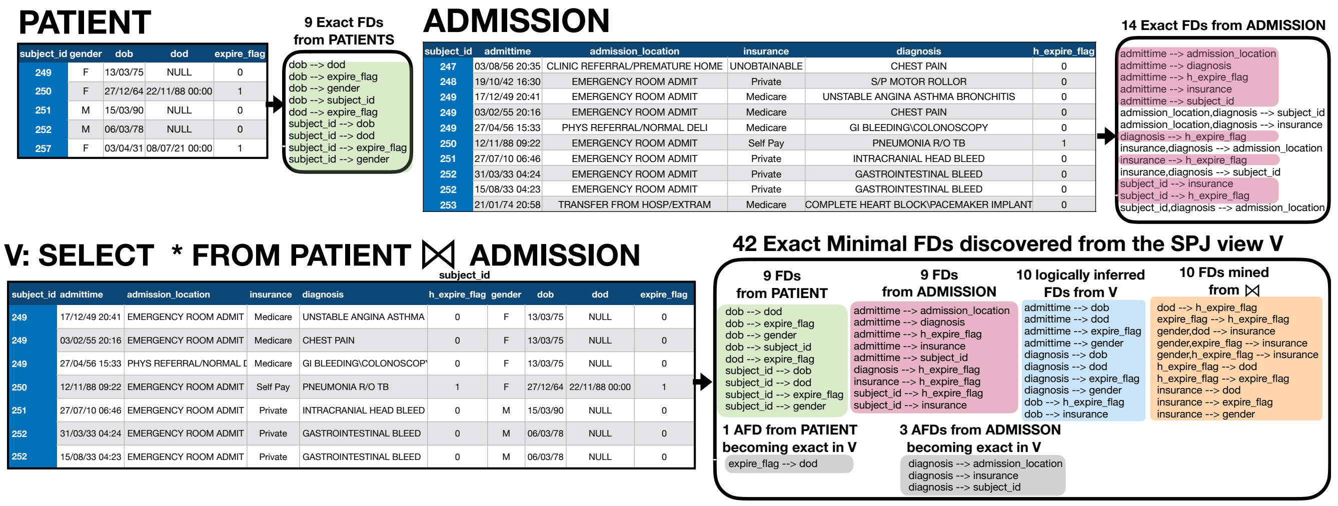
Despite the vast literature on the topic, all existing algorithms focus on the FD discovery problem for one relation at a time and do not explore reuse of computation when dealing with multiple relations. However, relations are oftentimes involved in view computation operations, among which the notable class of SPJ (Select-Project-Join) view operations. In this case, the existing algorithms addressing FD discovery would have to entirely recompute the set of FDs for the obtained integrated view without being able to reuse any of the previous computations on the individual base tables of the view. In this paper, for the first time we tackle the problem of inferring FDs on integrated views (also called Inferred FD discovery problem) by solely relying on the FDs available from the base tables of the views. To do so, we employ the well-known concept of why-provenance [4], which allows us to identify the lineage of the FDs on the integrated view and throughout the view computation operations.

In our work, we address the Inferred FD discovery problem, which consists of considering a SPJ view on top of a set of base tables and computing the FDs holding on the view by reusing as much as possible the discovered FDs on the base tables, and, in turn, reduce the SPJ view computation with only the needed attributes.

Explicitly relating the FDs of the base table to the FDs of the view has interest on its own as it allows for instance to understand whether FDs on the base tables are persistent on the view. It can also be beneficial whenever the user needs help while debugging the FDs on the base tables by choosing the most relevant ones if they also apply to the view. Since SPJ views might be obtained as results of data integration and ETL scenarios [11], our method allows to understand how constraints (namely the FDs) are actually affected by the integration process. New FDs that hold on the view but not on the base tables may help the user better understand and explain the result of the data integration process.

Our method allows us to obtain time savings in terms of FD discovery from integrated views. By leveraging logical inference and provenance triples, we can avoid recomputing the FDs holding on the view from scratch and only focus on the new FDs that are not holding on the base tables.

To address the Inferred FD discovery problem, we use a multi-step pipeline encompassing the discovery of different types of FDs and leveraging logical inference. To this end, we propose InFine, an efficient solution for automatically discovering multi-relation FDs starting from the FDs of the


PATIENT
9 Exact FDs from PATIENTS
ADMISSION
14 Exact FDs from ADMISSION
V: SELECT * FROM PATIENT ⋈ ADMISSION
subject_id
42 Exact Minimal FDs discovered from the SPJ view V
9 FDs from PATIENT
9 FDs from ADMISSION
10 logically inferred FDs from V
10 FDs mined from ⋈
1 AFD from PATIENT becoming exact in V
3 AFDs from ADMISSON becoming exact in V

PATIENT

subject_id	gender	dob	dod	expire_flag
249	F	13/03/75	NULL	0
250	F	27/12/64	22/11/88 00:00	1
251	M	15/03/90	NULL	0
252	M	06/03/78	NULL	0
257	F	03/04/31	08/07/21 00:00	1

9 Exact FDs from PATIENTS

```
dob --> dod
dob --> expire_flag
dob --> gender
dob --> subject_id
dod --> expire_flag
subject_id --> dob
subject_id --> dod
subject_id --> expire_flag
subject_id --> gender
```

ADMISSION

subject_id	admittime	admission_location	insurance	diagnosis	h_expire_flag
247	03/08/56 20:35	CLINIC REFERRAL/PREMATURE HOME	UNOBTAINABLE	CHEST PAIN	0
248	19/10/42 16:30	EMERGENCY ROOM ADMIT	Private	S/P MOTOR ROLLOR	0
249	17/12/49 20:41	EMERGENCY ROOM ADMIT	Medicare	UNSTABLE ANGINA ASTHMA BRONCHITIS	0
249	03/02/55 20:16	EMERGENCY ROOM ADMIT	Medicare	CHEST PAIN	0
249	27/04/56 15:33	PHYS REFERRAL/NORMAL DELI	Medicare	GI BLEEDING\COLONOSCOPY	0
250	12/11/88 09:22	EMERGENCY ROOM ADMIT	Self Pay	PNEUMONIA R/O TB	1
251	27/07/10 06:46	EMERGENCY ROOM ADMIT	Private	INTRACRANIAL HEAD BLEED	0
252	31/03/33 04:24	EMERGENCY ROOM ADMIT	Private	GASTROINTESTINAL BLEED	0
252	15/08/33 04:23	EMERGENCY ROOM ADMIT	Private	GASTROINTESTINAL BLEED	0
253	21/01/74 20:58	TRANSFER FROM HOSP/EXTRAM	Medicare	COMPLETE HEART BLOCK\PACEMAKER IMPLANT	0

14 Exact FDs from ADMISSION

```
admittime --> admission_location
admittime --> diagnosis
admittime --> h_expire_flag
admittime --> insurance
admittime --> subject_id
admission_location,diagnosis --> subject_id
admission_location,diagnosis --> insurance
diagnosis --> h_expire_flag
insurance,diagnosis --> admission_location
insurance --> h_expire_flag
insurance,diagnosis --> subject_id
subject_id --> insurance
subject_id --> h_expire_flag
subject_id,diagnosis --> admission_location
```

V: SELECT * FROM PATIENT ⋈ ADMISSION

subject_id	admittime	admission_location	insurance	diagnosis	h_expire_flag	gender	dob	dod	expire_flag
249	17/12/49 20:41	EMERGENCY ROOM ADMIT	Medicare	UNSTABLE ANGINA ASTHMA	0	F	13/03/75	NULL	0
249	03/02/55 20:16	EMERGENCY ROOM ADMIT	Medicare	CHEST PAIN	0	F	13/03/75	NULL	0
249	27/04/56 15:33	PHYS REFERRAL/NORMAL D	Medicare	GI BLEEDING\COLONOSCOP	0	F	13/03/75	NULL	0
250	12/11/88 09:22	EMERGENCY ROOM ADMIT	Self Pay	PNEUMONIA R/O TB	1	F	27/12/64	22/11/88 00:00	1
251	27/07/10 06:46	EMERGENCY ROOM ADMIT	Private	INTRACRANIAL HEAD BLEED	0	M	15/03/90	NULL	0
252	31/03/33 04:24	EMERGENCY ROOM ADMIT	Private	GASTROINTESTINAL BLEED	0	M	06/03/78	NULL	0
252	15/08/33 04:23	EMERGENCY ROOM ADMIT	Private	GASTROINTESTINAL BLEED	0	M	06/03/78	NULL	0

42 Exact Minimal FDs discovered from the SPJ view V

9 FDs from PATIENT	9 FDs from ADMISSION	10 logically inferred FDs from V	10 FDs mined from ⋈
dob --> dod	admittime --> admission_location	admittime --> dob	dod --> h_expire_flag
dob --> expire_flag	admittime --> diagnosis	admittime --> dod	expire_flag --> h_expire_flag
dob --> gender	admittime --> h_expire_flag	admittime --> expire_flag	gender,dod --> insurance
dob --> subject_id	admittime --> insurance	admittime --> gender	gender,expire_flag --> insurance
dod --> expire_flag	admittime --> subject_id	diagnosis --> dob	gender,h_expire_flag --> insurance
subject_id --> dob	diagnosis --> h_expire_flag	diagnosis --> dod	h_expire_flag --> dod
subject_id --> dod	insurance --> h_expire_flag	diagnosis --> expire_flag	h_expire_flag --> expire_flag
subject_id --> expire_flag	subject_id --> h_expire_flag	diagnosis --> gender	insurance --> dod
subject_id --> gender	subject_id --> insurance	dob --> h_expire_flag	insurance --> expire_flag
		dob --> insurance	insurance --> gender

1 AFD from PATIENT becoming exact in V

```
expire_flag --> dod
```

3 AFDs from ADMISSON becoming exact in V

```
diagnosis --> admission_location
diagnosis --> insurance
diagnosis --> subject_id
```

Fig. 1: Excerpt of the MIMIC-III clinical database with FDs on base tables and integrated view (best viewed in color).

base tables. Our main contributions are as follows:

- We propose a provenance-based mechanism capable of generating and exploiting provenance triples in a view specification. In particular, for each FD, we capture the type of the FD and a subquery of the view in which the FD holds.
- We design five algorithms to compute the provenance of FDs (from base FDs, inference, or join operations) from the results of the FD discovery on the base tables; these algorithms seamlessly address the case of SPJ views, a significant and representative query fragment.
- We propose InFine, a full-fledged system implementing our algorithms. InFine is available online[1] with code, scripts, and datasets for the reproducibility of our experiments;
- We gauge the effectiveness of our system through an extensive experimental evaluation. We compare InFine against state-of-the-art FD discovery methods over a rich number of SPJ views on real-world and synthetic datasets. We find that InFine outperforms the competing methods by one order up to two orders of magnitude in terms of execution time for discovering exact FDs while preserving the smallest memory consumption on average.

Outline. Section II presents an illustrative example. Section III presents the necessary background and notations. In Section IV, we formalize the Inferred FD discovery problem and provide an overview of InFine. We also present our main contributions and the algorithms at the core of InFine. We describe our performance experiments evaluating the efficiency and accuracy of InFine in Section V. Finally, we discuss related work in Section VI and conclude in Section VII.

II. Motivating Example

Let us consider a real-world clinical database MIMIC-III[2] [15], out of which we extract the PATIENT table containing information about patients: their identifier (subject_id), gender, date of birth (dob), date of death (dod), and a boolean expire_flag indicating whether the patient passed away, and the table ADMISSION containing administrative and clinical information about patients such as the hospital admission time (admittime), the admission location (admission_location), the insurance, the diagnosis, and a Boolean h_expire_flag indicating whether the patient died at the hospital. Let us consider V, the SPJ view (illustrated in Fig. 1) that computes a join between the two above tables: SELECT * FROM PATIENT, ADMISSION WHERE PATIENT.subject_id=ADMISSION.subject_id. We are interested in discovering the set of FDs that hold over the SPJ view by reusing as much as possible the FDs of the base tables PATIENT and ADMISSION. For ease of exposition, we focus on FDs with RHS limited to one single attribute (as it is the case in a canonical cover of FDs). The LHS might consist of multiple attributes as those mined from the join in Fig. 1. A total of 42 minimal and canonical FDs is considered in this running example thus making the example far from being trivial. These FDs can be obtained by FD discovery methods running on the view. However, their lineage is also important as they carry the information of whether they are valid in the base tables or they solely hold on the view. Precisely, we use color coding to encode the FD provenance in the figure. We can observe that the 9 exact FDs from the base table PATIENT (highlighted in green), as well as 9 (out of the 14) exact FDs from the base table ADMISSION (highlighted in pink) are preserved in the result of the integrated view. Imagine that a data steward would like to investigate why a few valid FDs in the base tables are no longer valid in the integrated view result or, conversely, why some FDs that are not valid in the base tables become valid in the view. Better informed, s/he may change and adapt the

[1]https://github.com/ucomignani/InFine
[2]https://physionet.org/content/mimiciii/1.4/

constraint (FD) enforcement strategies when curating the data using the FDs to ensure data quality [3]. Moreover, 10 FDs (highlighted in blue) from the view result can be obtained by logical inference over the sets of exact FDs discovered from each base table: for instance, `diagnosis` $\rightarrow$ `dod` is obtained from `insurance,diagnosis` $\rightarrow$ `subject_id` and `subject_id` $\rightarrow$ `insurance` in `ADMISSION` and `subject_id` $\rightarrow$ `dod` in `PATIENT`. As a side note, we are not addressing here the problem of meaningfulness of the FDs, which is orthogonal and of independent interest. Indeed, a valid FD may not be semantically meaningful and judging whether a valid FD is relevant typically needs human intervention.

When the two tables are joined, patient #257 is removed due to the absence of the corresponding `subject_id` value in the other table and approximate FDs (AFD) (such as `expire_flag` $\rightarrow^1$ `dod` in `PATIENT`) become exact in the SPJ view. These FDs are highlighted in grey in the figure.

Finally, only 10 exact FDs (highlighted in orange) that hold over the view result have to be discovered from scratch from the view. However, if we partially join the two tables, only with the following combinations of tuples: [(#249,#252) or (#249,#251)] and [(#250,#251) or (#250, #252)], we can obtain the remaining 10 join FDs without having to compute the entire view beforehand.

This example show that with existing FD discovery approaches, the FDs would have to be computed on both the base tables and the integrated view result to preserve the FD provenance information. Furthermore, to identify the provenance of the FDs, a comparison among the two FD sets would have to be performed. We will show the overhead of this process in our experimental study in Section V.

Hence, a better understanding of the mechanisms underlying the provenance of FDs from integrated views is highly needed. Based on these observations, in our work, we address the following key questions: How can we preserve the provenance of FDs discovered from integrated views? Instead of executing FD discovery over each base table and a SPJ view result independently, can we infer most of the FDs on the view as well as reuse the FDs from the base tables and achieve a non negligible speedup? We answer these key questions in the rest of the paper and propose an efficient solution for discovering FDs from integrated views.

III. Preliminaries

Next, we recall the necessary definitions of FDs and join operators with their application to our problem.

Definition 1 (Functional dependency satisfaction): Let I be an instance over a relation schema R, and $\boldsymbol{X}$, $\boldsymbol{Y}$ be two sets of attributes from R. I satisfies a functional dependency $d : \boldsymbol{X} \rightarrow \boldsymbol{Y}$, denoted by $I \models d$ if and only if:

$$\forall t_1, t_2 \in I, t_1[\boldsymbol{X}] = t_2[\boldsymbol{X}] \Rightarrow t_1[\boldsymbol{Y}] = t_2[\boldsymbol{Y}]. \qquad (1)$$

Note that the above definition holds regardless of the selected null semantics. This implies that our approach is not depending on the underlying semantics of null values, in line with other approaches in the literature [2], [9], [19].

Following the convention, we use uppercase letters for attribute sets and lowercase letters for single attributes. Moreover, in the rest of the paper we use canonical FDs, i.e., minimal FDs with only one attribute in their right-hand part.We recall that for any set of FDs, it can be computed a logically equivalent set of canonical FDs, thus this is done without loss of generality. We define the SPJ view specification and the set of projected attributes over this view as follows:

Definition 2 (SPJ view specification): Let $\mathbf{R} = \{R_1; \ldots; R_n\}$ be a set of relational instances. We define a view specification $V_{\mathbf{R}}$ as a relational algebra formula over relations in $\mathbf{R}$ and limited to the following set of operators: $\{\pi; \sigma; \bowtie; ⟗; ⟖; ⟕; \ltimes; \rtimes\}$.

Definition 3 (Projected attributes set): Let V, V_1, and V_2 be view specifications. Let R be a relational instance. Let X be a set of attributes. Let ρ be a set of constraints. Let $atts(S)$ denotes the set of attributes over a relational instance S. Let $\diamond$ be a join operator in $\{\bowtie; ⟗; ⟖; ⟕\}$. Then the set of projected attributes, denoted by $proj()$ is defined as follows:

$$proj(R) = atts(R) \qquad proj(V_1 \ltimes V_2) = proj(V_1)$$
$$proj(\pi_X(V)) = X \qquad proj(V_1 \rtimes V_2) = proj(V_2)$$
$$proj(\sigma_\rho(V)) = proj(V) \quad proj(V_1 \diamond V_2) = proj(V_1) \cup proj(V_2)$$

Finally, we define the types of FDs and the provenance triples derived in our framework and used to maintain the provenance information of every discovered FD. The base FDs are defined as follows:

Definition 4 (Base FD): Let $\mathbf{R} = \{R_1; \ldots; R_n\}$ be a set of relational instances. Let $V_{\mathbf{R}}$ be a view specification over relations in $\mathbf{R}$. Let $fds(R_i)$ denotes the set of minimal FDs over the relation $R_i \in \mathbf{R}$. Let $fds(V_{\mathbf{R}})$ denotes the set of minimal FDs over the view specified by $V_{\mathbf{R}}$. An FD d is a base FD if $d \in fds(V_{\mathbf{R}})$ and $\exists R_i \in \mathbf{R}$ s.t. $d \in fds(R_i)$.
We now define the notion of upstaged FD, which can occurs either in the case of a selection or of a join operation:

Definition 5 (Upstaged FD): Let R_1, R_2 be two relational instances. Let $fds(R_i)$ denotes the set of minimal FDs over a relational instance R_i. An FD d is an upstaged FD for R_1 if d is defined over attributes from R_1 and:

- in the case of a selection $\sigma_\rho(R_1)$: $d \in fds(\sigma_\rho(R_1))$ and $d \notin fds(R_1)$
- in the case of a join $R_1 \diamond R_2$: $d \in fds(R_1 \diamond R_2)$ and $d \notin fds(R_1)$

In the following, we distinguish the upstaged FDs by considering if they come from a selection operation or a join operation.

We also define two types of FDs that arise specifically after a join operation is performed:

Definition 6 (Inferred FD): Let R_1, R_2 be two relational instances. Let $fds(R_i)$ denotes the set of minimal FDs over a relational instance R_i. Let $atts(R_i)$ denotes the set of attributes of R_i. Let $d : X \rightarrow y$ be a FD such that $d \in fds(R \diamond R')$. Then d is an inferred FD if

$$(X \cup \{y\}) \cap atts(R_1) \neq \emptyset \wedge (X \cup \{y\}) \cap atts(R_2) \neq \emptyset$$

and either:

- d can be inferred through the use of Armstrong's axioms;
- $\exists d' : X' \to y$ such that $X \subset X'$ and d' can be inferred through Armstrong's axioms.

Definition 7 (Join FD): Let $d : X \to y$ be a FD such that $d \in fds(R \diamond R')$. Then d is a join FD if

$$(X \cup \{y\}) \cap atts(R_1) \neq \emptyset \wedge (X \cup \{y\}) \cap atts(R_2) \neq \emptyset$$

and d did not belong to the set of inferred FDs.
From these types of FDs, we define the provenance triples as follows:

Definition 8 (FD Provenance Triple): Let $\mathbf{R} = \{R_1; \ldots; R_n\}$ be a set of relational instances. Let $V_{\mathbf{R}}$ be a view specification over relations in $\mathbf{R}$. A provenance triple (d, t, s) for an FD d over the view specified by $V_{\mathbf{R}}$ is a triple composed of:

- *the FD d whose provenance is described;*
- *the type t of d, taking one of the following values: "base", "upstaged selection", "upstaged left", "upstaged right", "inferred" or "joinFD";*
- *the first sub-query s over the view specification $V_{\mathbf{R}}$ in which d holds during the view computation.*

We illustrate the provenance triples in the following example:
Example 1: Following the running example of Figure 1, the provenance triple for FDs `subject_id` $\to$ `dob`; `expire_flag` $\to$ `dod` and `gender, h_expire_flag` $\to$ `insurance` will be the following:

$$(\texttt{subject\_id} \to \texttt{dob}, \text{"}base\text{"}, \texttt{ADMISSION})$$
$$(\texttt{expire\_flag} \to \texttt{dod}, \text{"}upstaged\ left\text{"}, \texttt{PATIENT} \bowtie_{subject\_id=subject\_id} \texttt{ADMISSION})$$
$$(\texttt{gender, h\_expire\_flag} \to \texttt{insurance}, \text{"}join\ FD\text{"}, \texttt{PATIENT} \bowtie_{subject\_id=subject\_id} \texttt{ADMISSION}).$$

IV. Mining FDs provenance triples over Views

In this section, we describe the workflow of InFine to compute FDs and their provenance information from a view specification and the base relations used in the view specification. In this workflow, after discovering FDs from the base tables, we mine the FDs appearing during the view computation by relying as much as possible on inference methods as well as on efficient methods to discover the remaining FDs that cannot be inferred.

Problem statement. *Let $\mathbf{R}$ be a set of base tables; $V_{\mathbf{R}}$ be a view specification over $\mathbf{R}$. The Inferred FD discovery problem consists of efficiently inferring functional dependencies over the view specified by $V_{\mathbf{R}}$ by reusing functional dependencies from $\mathbf{R}$ and annotating each FD with its provenance information under the form of a provenance triple.*

In order to compute the functional dependencies for a given view, one needs to tackle the complexity of the FD mining problem and thus to tame the size of explored FDs lattices.

In the following theorem, we state that at each computation step of a view without projection, the FDs previously mined

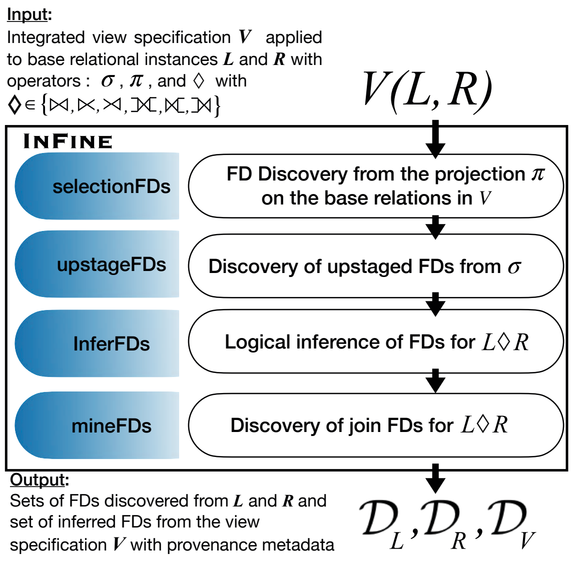

Input:
Integrated view specification V applied to base relational instances L and R with operators : σ , π , and $\Diamond$ with $\Diamond \in \{\bowtie, \ltimes, \rtimes, ⟗, ⟕, ⟖\}$
$V(L,R)$
InFine
selectionFDs
FD Discovery from the projection π on the base relations in V
upstageFDs
Discovery of upstaged FDs from σ
InferFDs
Logical inference of FDs for $L \Diamond R$
mineFDs
Discovery of join FDs for $L \Diamond R$
Output:
Sets of FDs discovered from L and R and set of inferred FDs from the view specification V with provenance metadata
$\mathcal{D}_L, \mathcal{D}_R, \mathcal{D}_V$

Fig. 2: Workflow of InFine for discovering inferred FDs from a view specification $V(L, R)$

continue to be valid. Conversely, we show that the application of a projection can only lead to the suppression of previously valid FDs, and never to the discovery of new FDs.

*Theorem 1: Let V, V_1, and V_2 be view specifications. Let $\mathcal{D}$, $\mathcal{D}_1$ and $\mathcal{D}_2$ be the sets of FDs over views specified by V, V_1 and V_2, respectively. Let $fds(V)$ denotes the set of FDs over a view specified by V.
Then:*

$$fds(\pi_X(V)) \subseteq \mathcal{D}, \qquad fds(\sigma_\rho(V)) \supseteq \mathcal{D}, \text{ and}$$

$$fds(V_1 \diamond V_2) \supseteq \mathcal{D}_1 \cup \mathcal{D}_2, \text{ with } \diamond \in \{\bowtie; ⟗; ⟖; ⟕; \ltimes; \rtimes\}.$$

Our Solution. In order to compute the set of FDs over integrated views with their provenance information, we propose the workflow illustrated in Figure 2. It consists of three main steps as follows: (1) Mining of FDs over the base relations, limited to FDs with attributes projected in the resulting view (InFine step 1); (2) Discovery of approximate single-table FDs that are upstaged and become exact FDs via the selection operations (InFine step 2); (3) Discovery of all FDs that can appear via the join operations (InFine step 3). In other words, Step 2 allows to retrieve FDs that are approximate in the individual base tables but become exact due to the selection operations of the SPJ views, whereas Step 3 retrieves FDs due to join operations between multiple base tables including upstaged, inferred and join FDs.

These steps and their corresponding algorithms are detailed in the next sections, respectively. Their application is done by InFine main algorithm (Algorithm 1) either during initialisation (lines#1–5 for Step 1) or during the recursive traversal of the view specification tree (Subroutine `provFDs` for Steps 2-3).

Algorithm 1: `InFine`

```
Input: R: a set of relational instances;
       $V_{\mathbf{R}}$: a view specification over instances in R.
Result: the set of provenance triples of the FDs over $V_{\mathbf{R}}$.
1  $\mathcal{E}_{\mathcal{D}} \leftarrow \emptyset$;
2  $\mathcal{A}_V \leftarrow$ compute the set of attributes $proj(V_{\mathbf{R}})$ ;
3  for each relation $R_i \in \mathbf{R}$ do
4      $\mathcal{D}_{R_i} \leftarrow$ compute FDs in $R_i$ limited to attributes in $\mathcal{A}_V$;
5      add $\mathcal{D}_{R_i}$ to $\mathcal{E}_{\mathcal{D}}$;
6  return provFDs($\mathbf{R}$, $V_{\mathbf{R}}$, $\mathcal{E}_{\mathcal{D}}$, $\mathcal{A}_V$);
7  Subroutine provFDs($\mathbf{R}$, $V_{\mathbf{R}}$, $\mathcal{E}_{\mathcal{D}}$, $\mathcal{A}_V$)
8      switch $V_{\mathbf{R}}$ do
9          case $R \in \mathbf{R}$ do
10             $\mathcal{P} \leftarrow \emptyset$;
11             $\mathcal{D} \leftarrow$ get $\mathcal{D}_R$ in $\mathcal{E}_{\mathcal{D}}$;
12             for each FD $d \in \mathcal{D}$ do
13                 add triple $(d, \text{"base"}, R)$ to $\mathcal{P}$
14             return $\mathcal{P}$;
15         case $\pi_X(V'_{\mathbf{R}})$ do
16             return provFDs$(\mathbf{R}, V'_{\mathbf{R}}, \mathcal{E}_{\mathcal{D}}, \mathcal{A}_V)$;
17         case $\sigma_\rho(V'_{\mathbf{R}})$ do
18             $\mathcal{P} \leftarrow$ provFDs$(\mathbf{R}, V'_{\mathbf{R}}, \mathcal{E}_{\mathcal{D}}, \mathcal{A}_V)$;
19             return $\mathcal{P} \cup$ selectionFDs$(\mathbf{R}, V'_{\mathbf{R}}, \mathcal{P}, \rho)$;
20         case $V'_{\mathbf{R}} \diamond V''_{\mathbf{R}}$ do
21             $\mathcal{P}_{rec} \leftarrow$ provFDs$(\mathbf{R}, V'_{\mathbf{R}}, \mathcal{E}_{\mathcal{D}}, \mathcal{A}_V)$;
22             $\mathcal{P}_{rec} \leftarrow$ provFDs$(\mathbf{R}, V''_{\mathbf{R}}, \mathcal{E}_{\mathcal{D}}, \mathcal{A}_V)$;
23             $\mathcal{P}_{up} \leftarrow$ joinUpFDs ($\mathcal{A}_V$);
24             $\mathcal{P}_{inf} \leftarrow$inferFDs ();
25             $\mathcal{P}_{join} \leftarrow$joinFDs ();
26             return $\mathcal{P}_{rec} \cup \mathcal{P}_{up} \cup \mathcal{P}_{inf} \cup \mathcal{P}_{join}$ ;
```

A. Reducing Space of Explored Candidates using Projections

As our goal is to discover the FDs occurring on top of an integrated view, we focus on mining FDs containing only attributes that are retrieved in the output SPJ view.

The number of considered attributes greatly influences the size of the explored lattices during FD mining. Thus, we make use of the knowledge of the projected attribute set at the very first step of our framework, in order to efficiently reduce the cost of mining FDs from the base relations. This can be seen in the main algorithm of our framework, Algorithm 1, during the mining of the FDs on the base relations (lines # 3–5). After this step, each following step of our framework relies on the FDs mined at the previous steps. Thus, additionally to reducing the FD discovery complexity over the base relations, removing unwanted FDs at this step also prevents useless computation during the following steps of our framework.

B. Mining of FDs Through View Selections

New FDs may appear due to selections in the view definition. This will occur when the selection operation leads to filter the tuples violating an FD, making this FD valid in the resulting instance. In such a case, our framework aims at producing provenance triples for these new FDs with Algorithm 2. This algorithm checks if the selection operation leads to some tuple filtering (from line #4) to avoid unnecessary mining step if the filter is not applicable. If some tuples are filtered by the selection, then upstaged FDs need to be mined and, to do so, we rely on a level-wise approach for FD mining. The intuition behind such an approach is to explore the possible sets of attributes involved in the candidate FDs, by organizing them into a lattice beginning from an infimum with FDs with singleton as `lhs` (line #5). Then, FD validity is tested and next level candidates are generated (line #11) while taking into account the FDs discovered at previous levels to prune the candidates FDs (lines #8–9). The algorithm stops when no candidates are left, and then the resulting set of minimal FDs are labelled with their provenance triples, the latter being returned by the algorithm.

Algorithm 2: `selectionFDs`

```
Input: R: a set of relational instances;
       $V_{\mathbf{R}}$: a view specification over instances in R;
       $\rho$: the selection condition;
       $\mathcal{P}_V$: the sets of provenance triples of $V_{\mathbf{R}}$;
       $\mathcal{A}_V$: the set of projected attributes to explore.
Result: upstaged FDs provenance triples over $V_{\mathbf{R}}$
1  $\mathcal{P} \leftarrow \emptyset$;
2  $\mathcal{D}_V \leftarrow$ get the set of FDs in the triples in $\mathcal{P}_V$;
3  $V_{sel} \leftarrow \sigma_\rho(V_{\mathbf{R}})$;
4  if size($V_{sel}$) < size($V_{\mathbf{R}}$) then
5      $\mathcal{D}_{cand} \leftarrow$ generate candidate FDs for first level of $V_{\mathbf{R}}$;
6      prune FDs in $\mathcal{D}_{sel}$ with attributes not in $\mathcal{A}_V$;
7      repeat
8          prune non-minimal FDs in $\mathcal{D}_{cand}$ knowing $\mathcal{D}_{out}$ ;
9          prune non-minimal FDs in $\mathcal{D}_{cand}$ knowing $\mathcal{D}_V$ ;
10         add to $\mathcal{D}_{out}$ the FDs from $\mathcal{D}_{cand}$ holding in $V$;
11         $\mathcal{D}_{cand} \leftarrow$ generate candidate FDs for next level;
12     until $\mathcal{D}_{cand} = \emptyset$;
13     for each FD $d \in \mathcal{D}_{out}$ do
14         add triple $(d, \text{"upstaged selection"}, V_{\mathbf{R}})$ to $\mathcal{P}$
15     return $\mathcal{P}$
16 else
17     return $\emptyset$;
```

C. Mining of FDs from Joined Tables

In the next sections, we will show how we can mine the upstaged, inferred and join FDs generated during a join operation. In the next theorem, we state that the join order can only affect the type of upstaged FDs which can switch between left or right upstaged FDs:

Lemma 1 (FD types preservation though join operations): Let $\mathbf{R} = \{R_1; \ldots; R_n\}$ be a set of relational instances. Let $fds(R_i)$ denotes the set of minimal FDs over the relational instance R_i. Let $\Sigma_{left\_up}, \Sigma_{right\_up}, \Sigma_{inf}$ and Σ_{join} denote the sets of left upstaged, right upstaged, inferred and join FDs over $R_1 \diamond \cdots \diamond R_n$, respectively. Then, the sets $\Sigma_{left\_up} \cup \Sigma_{right\_up}, \Sigma_{inf}$ and Σ_{join} are equal regardless of the join order. At the opposite, equality between sets Σ_{left_up} and Σ_{right_up} cannot be guaranteed.

Next, we show how upstaged FDs can be mined over joins.

1) Mining of Upstaged FDs from Joined Tables: Upstaged FDs may appear due to the join operations in the SPJ view when tuples from one base relation cannot be joined with their counterpart in the other base relation, i.e., when some join attribute values are missing in one of the tables. This mechanism is expressed more formally in the following lemma:

Lemma 2 (Upstaged join FDs): Let L and R be two instances over relations $\mathbf{S}$ *and* $\mathbf{T}$*, respectively, and $\mathcal{D}_L$ and $\mathcal{D}_R$ be the two sets of all FDs such that $L \models \mathcal{D}_L$ and $R \models \mathcal{D}_R$, respectively.*
Then the sets of upstaged join FDs denoted $\mathcal{D}_L^{new}$ and $\mathcal{D}_R^{new}$ are the sets:

$$\mathcal{D}_L^{new} = \{d \mid d \notin \mathcal{D}_L \wedge (L \Diamond_{X=Y}(\pi_Y(R)) \models d)\} \quad (2)$$
$$\mathcal{D}_R^{new} = \{d \mid d \notin \mathcal{D}_R \wedge (\pi_X(L) \Diamond_{X=Y} R) \models d)\} \quad (3)$$

Example 2: To illustrate the case of upstaged join FDs we use the example in Figure 1. The FD `expire_flag` $\not\to^1$ `dod` in Table `PATIENT` is violated by the presence of two tuples for patient (#257). However, in the join result of `PATIENT` $\bowtie_{\texttt{subject\_id}}$ `ADMISSION`, the violating tuple #257 has no counterpart in the `ADMISSION` table and it disappears from the join result. Consequently, the FD `expire_flag` $\to$ `dod` becomes valid in the join result.

Algorithm 3: `joinUpFDs`

```
Input: $V_L, V_R$: two view specifications;
       $X, Y$: the sets of join attributes for $V_L$ and $V_R$;
       $\Diamond$: a join operator in $\{\bowtie; \ltimes; \rtimes; ⟗; ⟖; ⟕\}$;
       $\mathcal{P}_{V_L}, \mathcal{P}_{V_R}$: the provenance triples sets of $V_L$ and $V_R$;
       $\mathcal{A}_V$: the set of projected attributes to explore.
Result: upstaged FDs provenance triples over $V_L$ and $V_R$
1  $\mathcal{P} \leftarrow \emptyset$;
2  $\mathcal{D}_{V_L} \leftarrow$ get the set of FDs in the triples in $\mathcal{P}_{V_L}$;
3  $\mathcal{D}^{up}_{left} \leftarrow$ upstagedFDs($V_L$, $V_R$, $X$, $Y$, $\mathcal{D}_{V_L}$,$\mathcal{A}_V$);
4  for each FD $d \in \mathcal{D}^{up}_{left}$ do
5      add triple $(d, ”\text{upstaged left}”, V_L \Diamond V_R)$ to $\mathcal{P}$;
6  $\mathcal{D}_{V_R} \leftarrow$ get the set of FDs in the triples in $\mathcal{D}V_R$;
7  $\mathcal{D}^{up}_{right} \leftarrow$ upstagedFDs($V_R$, $V_L$, $Y$, $X$, $\mathcal{D}_{V_R}$,$\mathcal{A}_V$);
8  for each FD $d \in \mathcal{D}^{up}_{right}$ do
9      add triple $(d, ”\text{upstaged right}”, V_L \Diamond V_R)$ to $\mathcal{P}$;
10 return $\mathcal{P}$;
11 Subroutine upstagedFDs($I$,$J$,$X$,$Y$,$\mathcal{D}$,$\mathcal{A}_V$)
12     $\mathcal{D}_{out} \leftarrow \emptyset$;
13     $I_{join} \leftarrow I \Diamond_{X=Y}(\pi_Y(J))$;
14     if size($I_{join}$) < size($I$) then
15         $\mathcal{D}_{cand} \leftarrow$ generate candidate FDs for first level of $I_{join}$;
16         prune FDs in $\mathcal{D}_{cand}$ with attributes not in $\mathcal{A}_V$;
17         repeat
18             prune non-minimal FDs in $\mathcal{D}_{cand}$ knowing $\mathcal{D}_{out}$;
19             prune non-minimal FDs in $\mathcal{D}_{cand}$ knowing $\mathcal{D}_V$;
20             add to $\mathcal{D}_{out}$ the FDs from $\mathcal{D}_{cand}$ holding in $I$;
21             $\mathcal{D}_{cand} \leftarrow$ generate candidate FDs for next level;
22         until $\mathcal{D}_{cand} = \emptyset$;
23     return $\mathcal{D}_{out}$
```

To compute upstaged FDs, we propose Algorithm 3. Lines #2–9 handle the inputs of the SPJ view for each table participating in the join operation. For each side of the join, the subroutine `upstagedFDs` is executed (lines #3 and 7) and computes partially the join only with the join attributes from the left side table (line #13) to check the assumption of the join value set preservation [10]. If the assumption is violated (i.e., if some tuples have been deleted through the join operation, as checked at line #14), some upstaged join FDs are produced. The subroutine works analogously to Algorithm 2 to discover the FDs in the input instance by using a level-wise approach where the previously discovered FDs are used to improves the pruning of candidates (lines #18–19).

In this algorithm, the computation is performed over upstaged FDs only. Next, we discover the inferred FDs by relying on the characteristics of the join and on the FDs discovered previously.

2) Mining of Inferred FDs from Joined Tables: We will now show how the inferred FDs can be deduced from the sets of FDs discovered from the tables involved in the join operation. In the next lemma, we show that if the right-hand side (`rhs`) of an FD is not functionally defined by the set of join attributes, then this FD cannot be an inferred FD in a join result. This property of the join operation is formally defined as follow:

Lemma 3: Let L and R be two instances over relations $\mathbf{S}$ *and* $\mathbf{T}$*, respectively. Let $L \Diamond_{X=Y} R$ be a join result with $X \subseteq atts(L)$, $Y \subseteq atts(R)$. For all $A \subseteq atts(L) \setminus X$ and $B \subseteq atts(R) \setminus Y$:*

$$\textit{if } L \Diamond_{X=Y} R \not\models X \to B \textit{ then } L \Diamond_{X=Y} R \not\models A \to B$$

Example 3: To illustrate the property proved in Lemma 3, we observe that the diagnosis is not determined by the patient identifier in Figure 1, for example patient #249 has been admitted three times for a different pathology each time, i.e., `PATIENT` $\bowtie_{\texttt{subject\_id}}$ `ADMISSION` $\not\models$ `subject_id` $\to$ `diagnosis`. From Lemma 3, we know that `diagnosis` in the join result `PATIENT` $\bowtie_{\texttt{subject\_id}}$ `ADMISSION` cannot be determined by any set of attributes coming from `PATIENT` table. Such similar inferences may be trivial for the user, but they usually require the knowledge of the attribute semantics. If not encoded, they are difficult to capture by a system. However, the property shown in Lemma 3 can be used to drastically reduce the set of possible FDs that can appear after a join operation.
Intuitively, from Lemma 3 follow that inferring FDs in the results of a join operation using Armstrong's transitivity axiom can only be done if the transitivity is done through the join attributes. This is formalized in the following theorem:

Theorem 2: Let L and R be two instances over relations $\mathbf{S}$ *and* $\mathbf{T}$*, respectively. Let $L \Diamond_{X=Y} R$ be a join result with $X \subseteq atts(L)$, $Y \subseteq atts(R)$. For all $A \subseteq atts(L) \setminus X$ and $B \subseteq atts(R) \setminus Y$,*
If $L \Diamond_{X=Y} R \models A \to X \wedge L \Diamond_{X=Y} R \models X \to B$,
Then $L \Diamond_{X=Y} R \models A \to B$.

Example 4: In `PATIENT` $\bowtie$ $_{\texttt{subject\_id}}$ `ADMISSION` result illustrated in Figure 1, we observe that the diagnosis determines the date of birth, i.e., `diagnosis` $\to$ `dob`. The reason is that we have: `admission_location`, `diagnosis` $\to$ `subject_id` in `ADMISSION` and `subject_id` $\to$ `dob` in `PATIENT`. Since these tables do not contain any null values for the `lhs` and `rhs` attributes, joining them with attribute `subject_id` leaves these FDs unchanged with no violation. By transitivity, we obtain: `diagnosis` $\rightarrow$ `dob`.

Algorithm 4: `inferFDs`

```
Input: $V_L, V_R$: two view specifications;
       $X, Y$: the sets of join attributes for $V_L$ and $V_R$;
       $\Diamond$: a join operator in $\{\bowtie; \ltimes; \rtimes; ⟗; ⟖; ⟕\}$;
       $\mathcal{P}_{V_L}, \mathcal{P}_{V_R}$: the provenance triples sets of $V_L$ and $V_R$;
       $\mathcal{A}_V$: the set of projected attributes to explore.
Result: inferred FDs provenance triples over $L\Diamond_{X=Y}R$
1  $\mathcal{D}_{V_L} \leftarrow$ get the set of FDs in the triples in $\mathcal{P}_{V_L}$;
2  $\mathcal{D}_{V_R} \leftarrow$ get the set of FDs in the triples in $\mathcal{P}_{V_R}$;
3  $\mathcal{D}_{infL} \leftarrow$ infer($X$,$Y$,$\mathcal{D}_{V_L}$,$\mathcal{D}_{V_R}$);
4  $\mathcal{D}_{infR} \leftarrow$ infer($Y$,$X$,$\mathcal{D}_{V_L}$,$\mathcal{D}_{V_R}$);
5  $\mathcal{D}_{inf} \leftarrow \mathcal{D}_{infL} \cup \mathcal{D}_{infR}$;
6  $\mathcal{D}_{ref} \leftarrow$ refine($L$,$R$,$X$,$Y$,$\mathcal{D}_{inf}$, $\Diamond$);
7  for each FD $d \in \mathcal{D}_{ref}$ do
8      add triple $(d, \text{"inferred"}, V_L\Diamond V_R)$ to $\mathcal{P}$
9  return $\mathcal{P}$;
10 Subroutine infer($X$,$Y$,$\mathcal{D}$, $\mathcal{D}'$)
11     $\mathcal{D}_{out} \leftarrow \emptyset$;
12     forall $A \rightarrow X$ in $\mathcal{D}$ do
13         forall $Y \rightarrow b$ in $\mathcal{D}'$ do
14             add $A \rightarrow b$ to $\mathcal{D}_{out}$;
15     return $\mathcal{D}_{out}$;
16 Subroutine refine($L$,$R$,$X$,$Y$,$\mathcal{D}_{inf}$, $\Diamond$)
17     Let $\mathcal{D}_{out} \leftarrow \mathcal{D}_{inf}$;
18     forall $A \rightarrow b$ in $\mathcal{D}_{inf}$ do
19         Let $I \leftarrow \pi_{X\cup A}(L)\Diamond\pi_{Y\cup\{b\}}(R)$;
20         forall $A' \subset A$ do
21             if $A' \rightarrow b$ holds in $I$ then
22                 add $A' \rightarrow b$ to $\mathcal{D}_{out}$;
23                 prune non-minimal FDs in $\mathcal{D}_{out}$ knowing
                   that $A' \rightarrow b$ is valid;
24     return $\mathcal{D}_{out}$;
```

To compute the set of FDs with `lhs` attributes coming from a single instance, as described in Theorem 2, we propose Algorithm 4. First, the subroutine `infer` extracts the FDs that can be retrieved by transitivity (lines #12 and 13 in subroutine `infer`). Note that in the case of equijoins, equality of values might be enforced between sets of attributes with different names (i.e., X and Y might be different), thus for the general case, the FD (line #13) cannot be simplified into an FD $X \rightarrow b$. Conversely, if we restrict the join operations to natural joins only, such a simplification can be made.

Then, for each FD returned by `infer`, the subroutine `refine` checks whether the FD is minimal or if a subset of its `lhs` leads to a minimal FD. To do so, subroutine `refine` uses an horizontal partition of the joined instances in which only the necessary attributes to perform the verification are considered (line #19). These necessary attributes are the join attributes (to perform the join operation), and the `lhs` and `rhs` attributes of the refined FD $A \rightarrow b$ (line #18) as `refine` only considers candidates with subsets of A as `lhs` and b as `rhs` (lines #20 and 21).

3) Mining of Join FDs from Joined Tables: Now, we characterize the set of join FDs which hold on a join result. Contrarily to the inferred FDs that can be deduced directly using a simple logical reasoning, these join FDs need to be discovered and validated from the data. For example, `gender, expire_flag`$\rightarrow$ `insurance` of our example has attributes from `PATIENT` in `lhs` and attributes from `ADMISSION` in `rhs` and it cannot be inferred logically. Other FDs with the same properties are illustrated in orange in Figure 1. In the following theorem, we show that if `lhs` attributes of an FD come from the instances participating in the join, then we cannot predict their validity without checking them directly with some representative (if not all) tuples of the join result:

Theorem 3: Let L and R be two instances over relations $\mathbf{S}$ *and* $\mathbf{T}$*, respectively. Let $L\Diamond_{X=Y}R$ be a join result with $X \subseteq atts(L)$, $Y \subseteq atts(R)$. We cannot guarantee that all FDs over $L\Diamond_{X=Y}R$ can be inferred from Armstrong's axioms over the FDs over L and R taken separately.*

Such FDs are illustrated in the following example:

Example 5: In our example of Figure 1, FDs that cannot be inferred are highlighted in orange. For example, `gender, h_expire_flag`$\rightarrow$ `insurance` is specific to the join of `PATIENT` and `ADMISSION`. It holds in `PATIENT` $\bowtie$ `ADMISSION`. Attributes `gender` and `expire_flag` come from `PATIENT` and attribute `insurance` comes from `ADMISSION`.

Theorem 3 motivates the need for designing a new method for computing FDs from partial join results, as we cannot always infer all the FDs only using logical reasoning. However, we can rely on the following theorem to greatly reduce number of remaining FDs to check from the data:

Theorem 4: Let L and R be two instances over relations $\mathbf{S}$ *and* $\mathbf{T}$*, respectively. Let $L\Diamond_{X=Y}R$ be a join result with $X \subseteq atts(L)$, $Y \subseteq atts(R)$. For all $A \subseteq atts(L)$, $A' \subseteq atts(R)$ and $b \in atts(R)$: If $L\Diamond_{X=Y}R \models AA' \rightarrow b$, Then $L\Diamond_{X=Y}R \models YA' \rightarrow b$.*

In-line with Theorem 4, we propose Algorithm 5 for selective mining and use the FDs previously discovered with Algorithms 3 and 4 to compute the remaining join FDs. Intuitively, Theorem 4 shows that a given attribute b can be a `rhs` of a remaining join FDs only if we have previously found an FD of the form $YA \rightarrow b$ with Y being the join attributes of the instance containing b. Thus, it allows us to focus only on the plausible `rhs` (lines #11 and 15 in subroutine `discover`) and explore their candidate `lhs` (lines #12-13 and 16-17). In practice, there is no need to generate every candidate FDs initially. Instead, candidate FDs can be explored by generating a first level containing only the smallest candidates and by generating upper levels only when currently evaluated candidates are not valid. Moreover, we can avoid the computation of the full join by deleting a given `lhs` attribute a if a is not a possible `rhs` and, for every FD candidate $d : A \rightarrow b$ such that $a \in A$, d is logically implied by previously discovered FDs.

D. Completeness, Correctness, and Complexity

In this section we show that the set of FDs retrieved by InFine is both complete and correct, and present our solution

Algorithm 5: `mineFDs`

```
Input: $V_L, V_R$: two view specifications;
       $X, Y$: the sets of join attributes for $V_L$ and $V_R$;
       $\Diamond$: a join operator in $\{\bowtie; \ltimes; \rtimes; ⟗; ⟖; ⟕\}$;
       $\mathcal{P}_{V_L}, \mathcal{P}_{V_R}$: the provenance triples sets of $V_L$ and $V_R$;
       $\mathcal{P}^{inf}_{V_L \Diamond V_R}$: the inferred provenance triples set.
Result: join FDs provenance triples over $L \Diamond_{X=Y} R$
$\mathcal{D}_{inf} \leftarrow$ get the set of FDs in the triples in $\mathcal{P}^{inf}_{V_L \Diamond V_R}$;
$\mathcal{D}_{V_L} \leftarrow$ get the set of FDs in the triples in $\mathcal{P}_{V_L}$;
$\mathcal{D}_{left} \leftarrow$ mine($L,R,X,Y,\mathcal{D}_{V_R},\mathcal{D}_{inf}$);
$\mathcal{D}_{V_R} \leftarrow$ get the set of FDs in the triples in $\mathcal{P}_{V_R}$;
$\mathcal{D}_{right} \leftarrow$ mine($R,L,Y,X,\mathcal{D}_{V_L},\mathcal{D}_{inf}$);
for each FD $d \in \mathcal{D}_{left} \cup \mathcal{D}_{right}$ do
    add triple $(d, \text{"joinFD"}, V_L \Diamond V_R)$ to $\mathcal{P}$
return $\mathcal{P}$;
Subroutine mine($I,J,X,Y,\mathcal{D}_J,\mathcal{D}_{inf}$)
    $\mathcal{D}_{out} \leftarrow \emptyset$;
    forall $Y \rightarrow b$ in $\mathcal{D}_J$ do
        forall $A \subseteq atts(I) \setminus X$ do
            if $\nexists A' \subset A, A' \rightarrow b \in \mathcal{D}_{inf}$ and $A \rightarrow b$ holds in $I \Diamond J$ then
                add $A \rightarrow b$ to $\mathcal{D}_{out}$;
    forall $YA' \rightarrow b \in \mathcal{D}_J$ such that $A' \not\rightarrow b$ do
        forall $A \cup A' \rightarrow b$ such that $A \subseteq atts(I) \setminus X$ do
            if $A \cup A' \rightarrow b$ holds in $I \Diamond J$ then
                add $A \cup A' \rightarrow b$ to $\mathcal{D}_{out}$;
    return $\mathcal{D}_{out}$;
```

complexity. First, we show that InFine always retrieve every functional dependency in the specified SPJ view.

Theorem 5 (InFine completeness): Let $\mathbf{R} = \{R_1; \ldots; R_n\}$ be a set of relational instances. Let $V_\mathbf{R}$ be a view specification over relations in $\mathbf{R}$. Let $fds(V_\mathbf{R})$ denotes the set of minimal FDs over the view specified by $V_\mathbf{R}$. Let $fds(\textit{InFine})$ denotes the set of FDs computed by InFine over the view specified by $V_\mathbf{R}$. Then:

$$\forall d \in fds(V_\mathbf{R}), \exists d' \in fds(\textit{InFine}) \text{ s.t. } d \equiv d'$$

This establish the completeness of our approach. Now, in the next theorem, we show that every FD retrieved by InFine is valid in the specified SPJ view:

Theorem 6 (InFine correctness): Let $\mathbf{R} = \{R_1; \ldots; R_n\}$ be a set of relational instances. Let $V_\mathbf{R}$ be a view specification over relations in $\mathbf{R}$. Let $I_{V_\mathbf{R}}$ be the view specified by $V_\mathbf{R}$. Let $fds(\textit{InFine})$ denotes the set of FDs computed by InFine over the view specified by $V_\mathbf{R}$. Then:

$$\forall d \in fds(\textit{InFine}), I_{V_\mathbf{R}} \models d$$

We now detail the complexity of our algorithms. Algorithms 2 and 3 are based on level-wise algorithms through the attributes lattices. Their complexity is exponential in the number of attributes of the considered table. They prune candidates at each level when it is possible. In terms of memory, only two levels are required. The memory size is bounded by $\mathcal{O}\binom{k}{k/2}$ where k is the number of attributes. Algorithm 4 infers and refines FDs coming from the previous step with complexity $\mathcal{O}(n \cdot f)$, where f is the number of

DB	Table	(Att# ; Tuple#)	FD#
MIMIC3	**Patients**	(7 ; 46.52k)	11
	Admissions	(18 ; 58.976k)	631
	Diagnoses_icd	(4 ; 651.047k)	2
	D_icd_Diagnoses	(3 ; 14.710k)	2
PTE	**active**	(2; 300)	1
	bond	(4 ; 9.317k)	3
	atm	(5 ; 9.189k)	5
	drug	(1 ; 340)	0
PTC	**atom**	(3; 12.333k)	2
	connected	(3 ; 24.758k)	3
	bond	(3 ; 12.379k)	2
	molecule	(2 ; 343)	1
TPC-H	**Supplier**	(7 ; 10k)	34
	Customer	(8 ; 150k)	51
	Orders	(9 ; 1.5M)	53
	LineItem	(16 ; 6M)	3946
	Nation	(4 ; 23)	9
	Region	(3 ; 5)	6
	Part	(7 ; 200k)	99
	Partsupp	(5 ; 800k)	11

TABLE I: Data characteristics.

validated FDs and n the maximal number of tuples in the left or right instance. The complexity of Algorithm 5 is $\mathcal{O}(f \cdot f_j)$ where f is the maximal number of validated FDs in the left or right instance and f_j the number of validated FDs in the join instance.

V. Experiments

Evaluation Goals. The two main points we seek to validate in our experimental study are as follows: (1) Does our approach enable us to discover all FDs with their provenance information in an efficient manner and faster than the straightforward approach? (2) What is the impact of different data and SPJ view characteristics on the InFine performance?

Setup. We perform all experiments on a laptop HP ZBook 15 machine with an Intel Core i7-4900MQ, 2.8 GHz, 32 GB RAM, powered by Windows 10 pro 64-bit. Our implementation in C++ uses only one thread. Sharable datasets, SPJ queries, scripts, and code are available at https://github.com/ucomignani/InFine.

Data. We use three real-world datasets and one synthetic dataset in our experiments: (1) MIMIC-3, a clinical database [3] [15]; (2) PTE[4], a database for predictive toxicology evaluation, used to predict whether a compound is carcinogenic, and (3) PTC[5], the dataset from the Predictive Toxicology Challenge that consists of more than three hundreds of organic molecules marked according to their carcinogenicity on male and female mice and rats; and (4) the TPC-H Benchmark[6] with scale-factor 1. The datasets characteristics are given in Table I and the characteristics of the queries corresponding to the views on the above datasets are provided in Table II. To the best of our knowledge, all the datasets used in the literature for

[3]https://physionet.org/content/mimiciii/1.4/
[4]https://relational.fit.cvut.cz/dataset/PTE
[5]https://relational.fit.cvut.cz/dataset/PTC
[6]http://www.tpc.org/tpch/

DB	SPJ View	Tuple#	FD#
MIMIC3	Q(patients $\bowtie$ admissions)	58,976	16
	diagnosesicd $\bowtie$ patients	58,798	12
	dicddiagnoses $\bowtie$ diagnosesicd	658,498	22
	[diagnosesicd$\bowtie$patients]$\bowtie$ dicddiagnoses	658,498	44
PTC	atom $\bowtie$ molecule	9,111	4
	connected $\bowtie$ bond	24,758	8
	[connected $\bowtie$ bond] $\bowtie$ molecule	18,312	12
	connected $\bowtie_{id1}$ [atom $\bowtie$ molecule]	18,312	12
PTE	atm $\bowtie$ drug	9,189	5
	active $\bowtie$ drug	299	1
	[bond $\bowtie$ drug] $\bowtie$ active	7,994	6
	[atm $\bowtie$ bond $\bowtie$ atm] $\bowtie$ drug	9,317	24
TPC-H	Q2*(P $\bowtie$ PS $\bowtie$ S $\bowtie$ N $\bowtie$ R)	21,696	69
	Q3*(C $\bowtie$ O $\bowtie$ L)	60,150	14
	Q9*(P $\bowtie$ PS $\bowtie$ S $\bowtie$ L $\bowtie$ O $\bowtie$ N)	3,735,632	8
	Q11*(P $\bowtie$ S $\bowtie$ N)	284,160	151

TABLE II: SPJ queries considered in our experiments.

benchmarking FD discovery methods consist of single tables, while we use multi-table scenarios. Note that the numbers of discovered FDs in the literature datasets are not comparable as we do not employ the same datasets due to the SPJ views studied in our approach. To handle multiple tables with SPJ queries, we propose our own benchmark datasets along with views on top of them. We adapted TPC-H queries by removing group-by and order-by statements (that are not addressed in our approach) and used the specified constants[7]. SPJ queries for the other datasets were manually crafted to make use of multiple tables and obtain joins of various sizes (from 2 to 6 tables), number of tuples (from 299 to 3 millions), and coverage values (from 0.12 to 25,812.67) to show the performances of our method comparatively in a large and representative range of SPJ views.

We observed that the cardinalities and overlap of the join attribute values in the datasets are rarely preserved through a join operation and this has a great impact on FD discovery from SPJ queries depending on the join operator used. To quantify this phenomenon, we define a measure called coverage as follows:

$$Coverage(R\Diamond L) = \frac{1}{2}\Big(Cov(R\Diamond L, L, X) + Cov(R\Diamond L, R, Y)\Big)$$

$$\text{with } Cov(Join, I, a) = \frac{1}{|\pi_a(I)|} \sum_{\forall v \in \pi_a(I)} \frac{|\sigma_{a=v}(Join))|}{|\sigma_{a=v}(I)|}.$$

where X and Y denote the join attributes of L and R base tables respectively. I is a considered instance and a the considered join attribute. If $Coverage(R\Diamond L) = 0$, no tuple from L can be joined with tuples in R. For $Coverage(R\Diamond L) < 1$, some tuples in L (or R) may be missing from the join result, as it is the case for patients #257 in `PATIENT` and #247, #248, and #253 in `ADMISSION` that do not have their counterparts in the other table in our example. For $Coverage(R\Diamond L) = 1$, there are as many tuples in both tables L and R as in the join result.

[7]see query validation sections of the TPC documentation at http://tpc.org/tpc_documents_current_versions/pdf/tpc-h_v3.0.0.pdf

For $Coverage(R\Diamond L) > 1$, there are more tuples in the join result than in tables L or R as some tuples may be repeated through the join. The coverage values of the corresponding views are given in Table III.

Methods. We compare the InFine algorithms applied to base tables against four state-of-the-art FD discovery methods, i.e.: (1) TANE [12], [13], (2) Fast_FDs [30], and (3) FUN [22], and (4) HyFD [26] with Java implementation of Metanome [24] using command line. The datasets employed in our study are stored in a PostgresSQL DBMS (version 12.7). Join attributes are indexed with both B-Tree and hash indexes. The reported results correspond to the average over 10 runs for each query.

Metrics. We compute accuracy, average runtime, and maximal memory consumption for each view. In our setting, accuracy corresponds to precision and is defined as the fraction of the number of FDs correctly discovered by InFine applied only to the base tables over the total number of FDs found by baseline methods classically applied to the SPJ view results. Accuracy values are given in Table III and represented as percentages in pie charts of Fig. 5. All methods (including ours) reach accuracy of 1 at the end of their execution.

Comparison Setup. Our goal is to compare our method which discovers FDs from a SPJ view specification and base tables and provides FD provenance information against the straightforward approach which consists of discovering all the FDs from each base table, computing the view result, and discovering the FDs on top of the latter. Classical methods do not provide provenance information. Then, a fair comparison requires the preservation of FD provenance information from the base tables to the view. To know the origin of each FD, both the baselines and our approach have to discover FDs from the base tables first. Since these execution times are the same in both cases, we don't include them. On the one hand, the competing methods are applied to each SPJ view result and we report their average execution time to which we added the execution time of the full SPJ view computation. On the other hand, InFine is applied to the base tables only and it computes a partial SPJ view depending on the view specification (query sub-tree) and generate FD provenance triples; we report InFine average execution time with time breakdown per algorithm. The time of the partial SPJ view computation is included in `mineFDs`. The generation of provenance triples is included in the execution of each InFine algorithm in charge of annotating each FD accordingly. Total I/O times of InFine are also reported in Table III.

A. Efficiency Evaluation

In a first set of experiments, we evaluate the runtime and memory consumption of the InFine algorithms compared to the state-of-the-art FD discovery methods that follow the straightforward approach over the 16 SPJ queries on the real-world and synthetic datasets with a wide range of coverage values. The selected queries are representative in terms of coverage and size (number of tables, tuples, and attributes). Results for other queries follow the same trend and are available as supplementary material on Github.

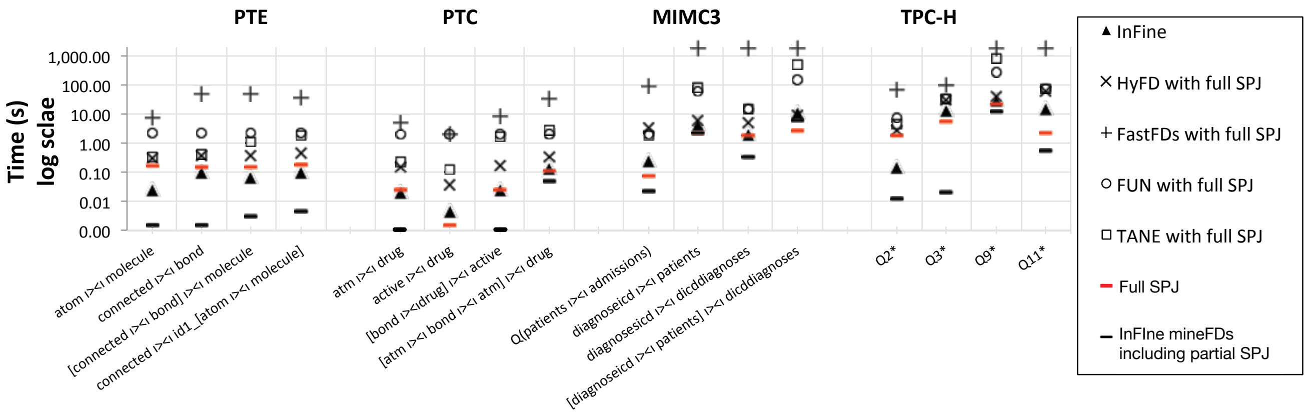


Fig. 3: Average runtime (in seconds): InFine against HyFD, FastFDs, FUN, and TANE with full and partial SPJ computation

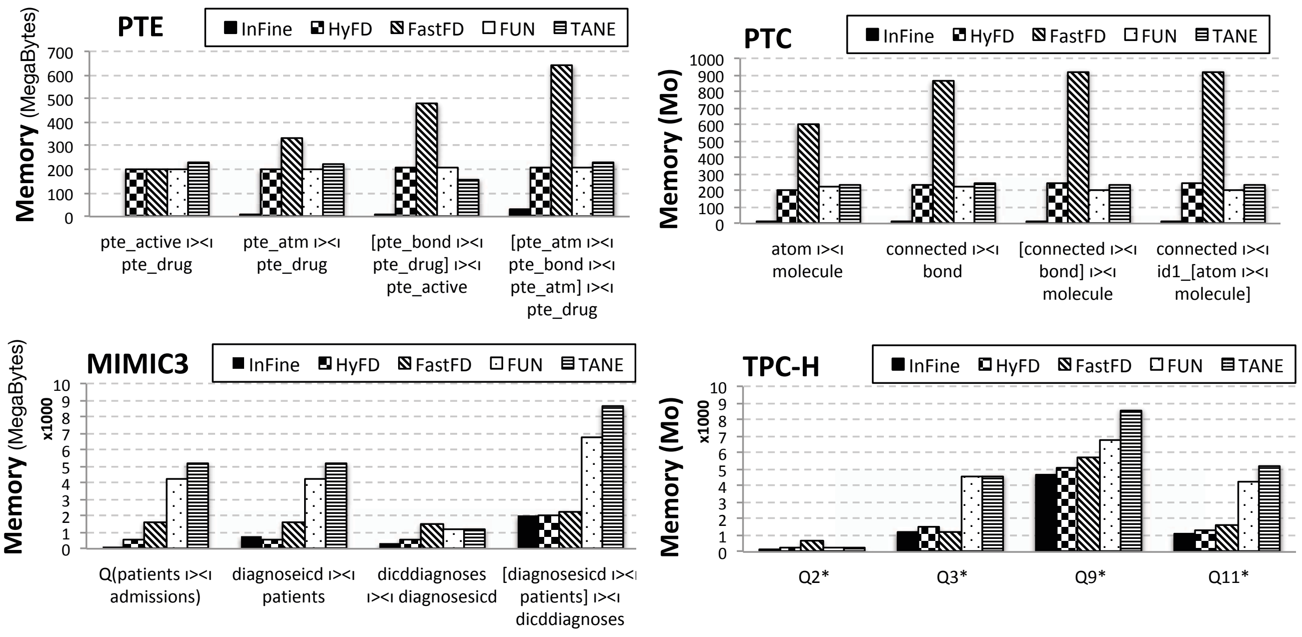


Fig. 4: Maximal memory consumption (in MegaBytes) of InFine against HyFD, FastFDs, FUN, and TANE algorithms

1) Runtime: Fig. 3 presents, for each method, the average total runtime in seconds for FD discovery (including data loading) in log scale. For the competing methods, we included the average execution time of the SPJ view over the indexed data. Similarly, we included the partial SPJ view computation time included in `mineFDs`. For all methods including ours, we did not include the time for discovering FDs from base tables since these costs are the same. Since our method does not require the full SPJ view computation to discover FDs and does not operate over the query result but only on the single base tables, it is much faster than the traditional methods with one order of magnitude on average up to two orders of magnitude compared to Fast_FDs ($> 2,000$ seconds). A provenance triple is generated each time an FD is validated by one of the InFine algorithms and is appended to the corresponding FD data structures with negligible time (included in each algorithm execution time). For MIMIC3 and TPC-H SPJ queries with the highest values for coverage and number of tuples (e.g., Q9* and Q11*), InFine is still outperforming the other methods. The time difference between full and partial SPJ computations shown in Fig. 3 also explains the advantage of our approach.

2) Memory Consumption: As shown in Fig. 4, the average maximal memory consumption of InFine (for accuracy equals 1) is the lowest for all queries across the datasets. HyFD has the second position in terms of memory consumption efficiency and TANE is the worst one. For all methods, Q9* has the highest memory consumption due to its large size of

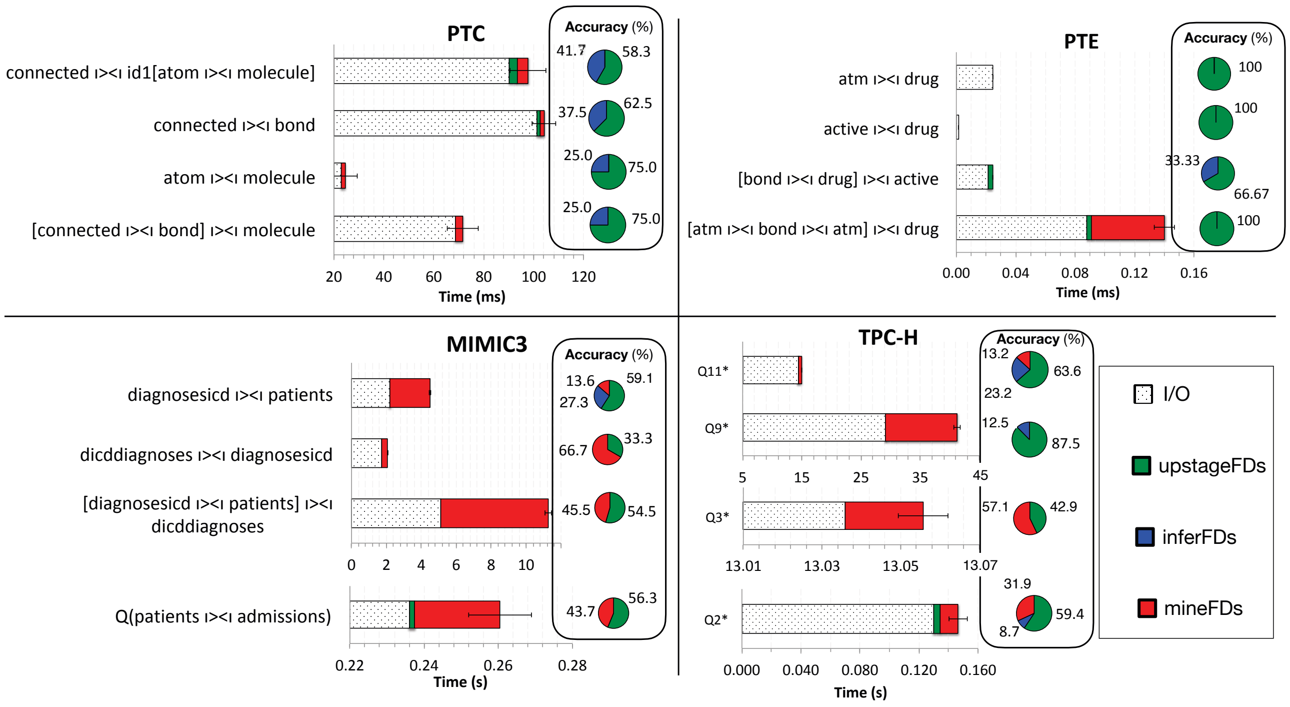


Fig. 5: Average runtime and accuracy of InFine with breakdown per algorithm

3.73 millions of tuples, very high coverage (25.8k), and the number of joins between 6 tables. This case is an example of a worst case scenario where many tuples are repeated through the join of the base tables. FD discovery from Q9* requires the mining and checking of all the valid FDs and storing all the relevant information. In this particular case, InFine time performance becomes comparable to HyFD over the full SJP view computation (due to `mineFDs`), although all FDs were discovered before by `upstageFDs` and `inferFDs` (see Fig. 5).

B. Quality Evaluation

In Fig. 5, we report the average runtime breakdown of each algorithm `upstageFDs`, `inferFDs`, and `mineFDs` of InFine (as horizontal histograms) and their respective percentages of discovered FDs (in the corresponding pie charts with the same color coding). Error bars represent the standard deviation of the average total runtime of InFine algorithms. `selectionFDs`'s time is included in `upstageFDs`'s time. Various base table data distributions, SPJ views, and coverage values with very different characteristics across the datasets illustrate the behavior of our algorithms.

Noticeably, `upstageFDs` can retrieve from 33.3% to 59% of the FDs in MIMIC3 in 0.75ms on average, from 58.3% to 75% in PTC in 1.12ms, from 66.7% to 100% in PTE in 1.5ms, and from 42.8.7% to 87.5% in TPC-H in 7.65ms, which are negligible times compared to `mineFDs` time and I/O time, the most time-consuming steps. Table III gives the number of FDs retrieved by each algorithm and time breakdowns.

The logical inference times are negligible (below 0.0001 ms) and are not reported in the table. `inferFDs` can retrieve 100% of the FDs for 3 queries over PTE. In some cases, `mineFDs` is executed but does not return any new FD (e.g., Q*9 in TPC-H or `[atm ⋈ bond ⋈ atm]⋈drug`), whereas in the other queries, mining subsets of the SPJ view using `mineFDs` is necessary to recover the remaining FDs. Without computing and mining the join results, `upstageFDs` and `inferFDs` can retrieve 83.01% of FDs (68.38 ± 20.00 and 14.63 ± 15.47, respectively) on average across all the datasets. We also observed that join ordering does not affect the total number of discovered FDs, but it changes FD provenance. The main reason is due to the difference in the query execution sub-trees triggering different FD validation by each InFine algorithm.

It should be noted that, due to the NP-completeness of the FD mining problem (cf. [8]), this exploration is necessary to ensure that no valid FD is missed.

These results clearly show the main advantages of our approach, reducing drastically the execution time of FD discovery from SPJ queries with minimal memory consumption, outperforming all the state-of-the art methods tested in our experiments over a large representative range of queries.

VI. Related Work

In the last three decades, numerous approaches from the database and the data mining communities have been proposed to extract automatically valid exact and approximate FDs from single relational tables [17], [6]. Liu et al. [20] have shown

DB	SPJ View	(Att# ; Tuple#)	Cov.	UpstageFDs Accuracy	InferFDs Accuracy	MineFDs Accuracy	Total Accuracy (FD#)	I/O (s)	upstageFDs (s)	mineFDs (s)
PTE	atm $\bowtie$ drug	(5 ; 9,189)	14.01	1	0	0	1 (5 FDs)	0.0246	0.0000	0.0000
	active $\bowtie$ drug	(2 : 299)	0.94	1	0	0	1 (1 FD)	0.0015	0.0000	0.0000
	[bond $\bowtie$ drug] $\bowtie$ active	(6 ; 7,994)	13.83	0.67	0.33	0	1 (6 FDs)	0.0215	0.0030	0.0000
	[atm $\bowtie$ bond $\bowtie$ atm] $\bowtie$ drug	(14 ; 9,317)	14.20	1	0	0	1 (24 FDs)	0.0879	0.0030	0.0492
PTC	atom $\bowtie$ molecule	(4 ; 9,111)	13.67	0.75	0.25	0	1 (4 FDs)	0.0231	0.0000	0.0015
	connected $\bowtie$ bond	(5 ; 24,758)	1.50	0.625	0.375	0	1 (8 FDs)	0.1012	0.0015	0.0015
	[connected $\bowtie$ bond] $\bowtie$ molecule	(6 ; 18,312)	27.08	0.75	0.25	0	12 (12FDs)	0.0686	0.0000	0.0030
	connected $\bowtie_{id1}$ [atom $\bowtie$ molecule]	(6 ; 18,312)	27.08	0.583	0.417	0	1 (12 FDs)	0.0903	0.0030	0.0045
MIMIC3	diagnosesicd $\bowtie$ patients	(12 ; 651,047)	7.50	0.591	0.273	0.136	1 (22 FDs)	2.1876	0.0015	2.3120
	dicddiagnoses $\bowtie$ diagnosesicd	(7 ; 658,498)	22.84	0.333	0	0.667	1 (12 FDs)	1.7202	0.0000	0.3497
	[diagnosesicd $\bowtie$ patients] $\bowtie$ dicddiagnoses	(14 ; 658,498)	22.84	0.545	0	0.455	1 (44 FDs)	5.1232	0.0000	6.1325
	Q(patients $\bowtie$ admissions)	(10 ; 6,736)	0.79	0.563	0	0437	1 (16 FDs)	0.2360	0.0015	0.0230
TPC-H	Q2*(P $\bowtie$ PS $\bowtie$ S $\bowtie$ N $\bowtie$ R)	(10 ; 21,696)	1.50	0.594	0.087	0.319	1 (69 FDs)	0.1299	0.0045	0.0120
	Q3*(C $\bowtie$ O $\bowtie$ L)	(6 ; 60,150)	0.12	0.429	0	0.571	1 (14 FDs)	13.036	0.0000	0.0198
	Q9*(P $\bowtie$ PS $\bowtie$ S $\bowtie$ L $\bowtie$ O $\bowtie$ N)	(9 ; 3,735,632)	25,813	0.875	0.125	0	1 (8 FDs)	16.967	0.0015	12.1261
	Q11*(P $\bowtie$ S $\bowtie$ N)	(15 ; 284,160)	80.09	0.636	0.232	0.132	1 (151 FDs)	13.771	0.0246	0.5777

TABLE III: Accuracy and time breakdowns of InFine algorithms

that the complexity of FD discovery is in $O(n^2(\frac{k}{2})^2 2^k)$ where k is the number of attributes and n the number of records considered. To find FDs efficiently, existing approaches can be classified into three categories: (1) Tuple-oriented methods (e.g., FastFDs [30], DepMiner [21]) that exploit the notion of tuples agreeing on the same values to determine the combinations of attributes of an FD; (2) Attribute-oriented methods (e.g., Tane [12], [13], Fun [22], [23], FDMine [31]) that use pruning techniques and reduce the search space to the necessary set of attributes of the relation to discover exact and approximate FDs. HyFD [26] exploits simultaneously the tuple- and attribute-oriented approaches to outperform the previous approaches; and more recently (3) Structure learning methods relying on sparse regression [32], or on entropy-based measures [16] to score candidate constraints (not limited to FDs alone). More particularly, FDX [32] performs structure learning over a sample constructed by taking the value differences over sampled pairs of tuples from the raw data. In addition, incremental approaches (e.g., [28], [5]) have been developed to tackle data volume and velocity with updating all valid FDs when new tuples are inserted outperforming classical approaches that recalculate all FDs after each data update. Extensive evaluations of FD discovery algorithms can be found in [9], [25]. To the best of our knowledge, previous work on FD discovery did not attempt to address the problem of FD discovery from integrated views in an efficient manner while preserving data provenance. Our approach combining logical inference and selective mining from the base tables adapting FD lattice approach avoids the full computation of FDs from the integrated views and it is the first solution in this direction.

The problem of deciding whether a semantic constraint (being a FD or a Join Dependency) is valid on a tableau view, knowing that it is valid on the base relations has been addressed by Klug et al. [18]. However, their view constraint problem and FD implication is inherently different from our FD inference problem, and allowing to reuse discovered FDs of base tables while annotating them with provenance information and efficiently recomputing FDs that are valid on the view and are not valid on the base tables (as stated in Th. 5). Moreover, the underlying technique employed in [18] and used for checking the validity of FDs leverages the tableau chase as opposed to using the FD lattice, the latter being a well established efficient method for FD discovery.

The problem of propagating XML keys to relations is an orthogonal problem with respect to ours [7]. The simple mapping language from XML to relations and the restriction to XML keys, that cannot capture relational functional dependencies, is specific to this work as also stated in their paper [7].

VII. Conclusions

We addressed the problem of FD discovery from integrated views starting from the FDs of multiple base tables by avoiding the full computation of the view beforehand. The salient features of our work are the following: (1) We leverage single-table approximate FDs that become exact FDs due to the join operation; (2) We leverage logical inference to discover FDs from the base tables without computing the full view result; and (3) We find new multi-table join FDs from partial join using selective mining on the necessary attributes. We empirically show that InFine outperforms, both in terms of runtime and memory consumption, the state-of-the-art FD discovery methods applied to the SPJ views that have to be computed beforehand. We hope that our work will open a new line of research for reusing the FDs discovered from multiple base tables. Various orderings of the base tables lead to different sets of potential upstaged FDs, which, in turn, may trigger different logical inferences. Future work will be to find the optimal ordering of the base tables to reduce the overall execution time and memory consumption.

VIII. Acknowledgment

This work is funded by the French National Research Agency, project QualiHealth, ANR-18-CE23-0002.

References

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

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

[3] L. Berti-Equille and F. Moussouni. Quality-aware integration and warehousing of genomic data. In *ICIQ'05-10th International Conference on Information Quality*, pages 1–15, 2005.

[4] P. Buneman, S. Khanna, and W. C. Tan. Why and where: A characterization of data provenance. In *Database Theory - ICDT 2001, 8th International Conference, London, UK, January 4-6, 2001, Proceedings*, pages 316–330, 2001.

[5] L. Caruccio, S. Cirillo, V. Deufemia, and G. Polese. Incremental discovery of functional dependencies with a bit-vector algorithm. In *Proc. of CEUR Workshop, vol. 2400*, 2019.

[6] L. Caruccio, V. Deufemia, and G. Polese. Relaxed functional dependencies—a survey of approaches. *IEEE TKDE*, 28(1):147–165, 2015.

[7] S. B. Davidson, W. Fan, and C. S. Hara. Propagating XML constraints to relations. *J. Comput. Syst. Sci.*, 73(3):316–361, 2007.

[8] S. Davies and S. Russell. Np-completeness of searches for smallest possible feature sets. In *AAAI Symposium on Intelligent Relevance*, pages 37–39. AAAI Press, 1994.

[9] F. Dürsch, A. Stebner, F. Windheuser, M. Fischer, T. Friedrich, N. Strelow, T. Bleifuß, H. Harmouch, L. Jiang, T. Papenbrock, and F. Naumann. Inclusion dependency discovery: An experimental evaluation of thirteen algorithms. In *CIKM 2019*, pages 219–228, 2019.

[10] H. Garcia-Molina, J. D. Ullman, and J. Widom. *Database systems - the complete book (International Ed.).* Pearson Education, 2002.

[11] B. Golshan, A. Y. Halevy, G. A. Mihaila, and W. Tan. Data integration: After the teenage years. In *Proceedings of the 36th ACM SIGMOD-SIGACT-SIGAI Symposium on Principles of Database Systems, PODS 2017, Chicago, IL, USA, May 14-19, 2017*, pages 101–106, 2017.

[12] Y. Huhtala, J. Kärkkäinen, P. Porkka, and H. Toivonen. Efficient discovery of functional and approximate dependencies using partitions. In *Proc. of ICDE*, pages 392–401, 1998.

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

[14] I. F. Ilyas, V. Markl, P. J. Haas, P. Brown, and A. Aboulnaga. CORDS: automatic discovery of correlations and soft functional dependencies. In *Proc. of ACM SIGMOD*, pages 647–658, 2004.

[15] A. E. Johnson, T. J. Pollard, L. Shen, H. L. Li-wei, M. Feng, M. Ghassemi, B. Moody, P. Szolovits, L. A. Celi, and R. G. Mark. MIMIC-III, a freely accessible critical care database. *Scientific data*, 3, 2016.

[16] B. Kenig, P. Mundra, G. Prasad, B. Salimi, and D. Suciu. Mining approximate acyclic schemes from relations, 2019.

[17] J. Kivinen and H. Mannila. Approximate inference of functional dependencies from relations. *Theoretical Computer Science*, 149(1):129–149, 1995.

[18] A. C. Klug and R. Price. Determining view dependencies using tableaux. *ACM Trans. Database Syst.*, 7(3):361–380, 1982.

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

[20] J. Liu, J. Li, C. Liu, and Y. Chen. Discover dependencies from data—a review. *IEEE Trans. on Knowl. and Data Eng.*, 24(2):251–264, Feb. 2012.

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

[22] N. Novelli and R. Cicchetti. FUN: an efficient algorithm for mining functional and embedded dependencies. In *Proc. of ICDT*, volume 1973 of *LNCS*, pages 189–203, 2001.

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

[24] T. Papenbrock, T. Bergmann, M. Finke, J. Zwiener, and F. Naumann. Data profiling with metanome. *PVLDB*, 8(12):1860–1863, 2015.

[25] 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. *Proc. VLDB Endow.*, 8(10):1082–1093, 2015.

[26] T. Papenbrock and F. Naumann. A hybrid approach to functional dependency discovery. In *SIGMOD'16*, page 821–833, 2016.

[27] G. N. Paulley. *Exploiting Functional Dependence in Query Optimization*. Citeseer, 2000.

[28] P. Schirmer, T. Papenbrock, S. Kruse, F. Naumann, D. Hempfing, T. Mayer, and D. Neuschäfer-Rube. DynFD: Functional dependency discovery in dynamic datasets. In *Proc. of EDBT*, pages 253–264, 2019.

[29] S. Thirumuruganathan, L. Berti-Équille, M. Ouzzani, J. Quiané-Ruiz, and N. Tang. Uguide: User-guided discovery of fd-detectable errors. In *Proc. of ACM SIGMOD*, pages 1385–1397, 2017.

[30] C. M. Wyss, C. Giannella, and E. L. Robertson. FastFDs: A heuristic-driven, depth-first algorithm for mining functional dependencies from relation instances - extended abstract. In *Proc. of DaWaK*, volume 2114 of *LNCS*, pages 101–110, 2001.

[31] H. Yao and H. J. Hamilton. Mining functional dependencies from data. *Data Min. Knowl. Discov.*, 16(2):197–219, 2008.

[32] Y. Zhang, Z. Guo, and T. Rekatsinas. A statistical perspective on discovering functional dependencies in noisy data. In *Proc. of ACM SIGMOD*, 2020.