Expressiveness and Complexity of Order Dependencies

Jaroslaw Szlichta*
University of Toronto
& IBM CAS Toronto
szlichta@cs.toronto.edu

Parke Godfrey
York University
godfrey@cse.yorku.ca

Jarek Gryz
York University
jarek@cse.yorku.ca

Calisto Zuzarte
IBM Toronto Laboratory
calisto@ca.ibm.com

ABSTRACT

Dependencies play an important role in databases. We study *order dependencies* (ODs)—and *unidirectional* order dependencies (UODs), a proper sub-class of ODs—which describe the relationships among *lexicographical* orderings of sets of tuples. We consider lexicographical ordering, as by the order-by operator in SQL, because this is the notion of order used in SQL and within query optimization. Our main goal is to investigate the *inference problem* for ODs, both in theory and in practice. We show the usefulness of ODs in query optimization. We establish the following theoretical results: (i) a hierarchy of order dependency classes; (ii) a proof of co-NP-completeness of the inference problem for the subclass of UODs (and ODs); (iii) a proof of co-NP-completeness of the inference problem of functional dependencies (FDs) from ODs in general, but demonstrate linear time complexity for the inference of FDs from UODs; (iv) a sound and complete elimination procedure for inference over ODs; and (v) a sound and complete *polynomial* inference algorithm for sets of UODs over *restricted* domains.

1. INTRODUCTION

Understanding the semantics of data is important for query optimization. Ordered streams are prevalent in query plans between operators to provide efficient evaluation. An optimizer must reason extensively over *interesting orders* while building query plans [11, 12]. Order for a tuple stream can be semantically specified via the attributes as by SQL's order-by. The *order specification* order by year desc, name asc requires that the tuple stream be sorted by year in descending order and, within each year group, by name in ascending order. This is a *lexicographical* ordering, a nested sort.

An order dependency (OD) states a semantic relationship between two order specyfications. Say that we knew the OD that id asc *orders* year asc, name asc. Then we would be assured that any tuple stream ordered by id asc would also necessarily be ordered by year asc, name asc. Note the converse is *not* necessarily assured: if the stream were ordered by year asc, name asc, it still might not be ordered by id asc. This is because the tuples within a given partition of year-name might fail to be ordered by id asc.

The concept of an OD is closely related to that of *functional dependency* (FD). Indeed, ODs *subsume* FDs [14]. If id asc *orders* year asc, name asc, then the FD that id *functionally determines* year and name must hold. ODs convey additional semantic information, of course: that of order.

ODs has been studied before with respect to lexicographical orders [10, 14, 18], and with respect to other order definitions (pointwise) [5, 6]. Our focus is on lexicographical orders. While lexicographical OD has been studied before, it has not been well understood. The *inference problem* is to answer whether an OD is logically entailed by a set of ODs. The *complexity* of the inference problem for (lexicographical) OD has heretofore not been known. We address this in this work. Working with (lexicographical) ODs is much more useful for query optimization than working with (pointwise) ODs [5, 6], because the *sequence* of the attributes in order specifications (interesting orders) [11, 12] as in the order-by statement matters. Lexicographical ODs are specified with respect to *lists* of attributes, whereas (pointwise) ODs are specified with respect to *sets* of attributes.

As business-intelligence applications have become more complex and as data volumes have grown, the analytic queries needed to support these applications have become more complex too. The increasing complexity raises performance issues and numerous challenges for query optimization. Traditional optimization methods often fail to apply when logical subtleties in database schemas and in queries circumvent them. Consider the SQL query in Query 1 over a data warehouse schema. The fact table sales has a foreign key S.date_id which references the dimension table date_dim. *Date* is captured in a hierarchical manner by attributes year, quarter or trimester, month, and day. The values of the attribute quarter divide year into four three-month periods, while those of trimester divide it into three four-month periods. Let there be a B+ tree index for date_dim on year, month, day. The optimizer may not employ this index to evaluate either the group-by or the order-by for the query in Query 1, because their specifications do not *match* the index's search key.

Of course, it is clear that month *functionally* determines quarter and trimester. So partitioning by year, trimester, quarter, month, day is the same as just by year, month, day. In fact, optimizers today would eliminate trimester and quar-

QUERY 1. *Eliminating trimester and quarter.*

```
select D.year, D.trimester, D.quarter,
       D.month, D.day, sum(S.sales) as total,
       count(*) as quantity
from date_dim D, sales S
where S.date_id = D.date_id and
      D.year between 2001 and 2004
group by D.year, D.trimester, D.quarter,
         D.month, D.day
order by D.year, D.trimester, D.quarter,
         D.month, D.day;
```

*This work was done when author was a graduate student at York University. He has since joined University of Toronto.

Proceedings of the VLDB Endowment, Vol. 6, No. 14

ter from the group-by via reasoning over the relevant FDs [12], and then employ the index for the group-by operation.[1]

The FD that month → quarter, trimester is not logically sufficient to optimize the order-by operation, however. One would need the additional semantic information of an OD that year, month, day *orders* year, trimester, quarter, month, day. This and similar subtleties cause the optimizer to miss opportunities to use indexes and to pipeline operations. Expensive operations as sort are added to a query plan, even when the data is already sorted properly. By incorporating reasoning over ODs into the optimizer—as has already been done for reasoning over FDs—many new optimizations would be possible.

In Section 2 (*Background*), we introduce a theoretical framework for ODs. The contributions of this paper appear in Section 2.3 and Sections 3 to 5, as follows

1. *Hierarchy of OD classes.* (Section 3)
   (a) Lexicographical ODs as defined in this paper are a proper sub-class of pointwise ODs. (Section 3.2) (The latter is defined in [5].)
   (b) UODs are a proper sub-class of ODs, Section 3.3.
   (c) FDs are a sub-class of UODs [14]. In Section 3.4 we additionally show that FDs are a *proper* sub-class of UODs.
2. *Complexity.* (Section 4)
   (a) The inference problem for UODs is co-NP-complete. (Section 4.1)
   (b) The inference problem of inferring FDs from ODs is also co-NP-complete, but it is only linear for the case of FDs over UODs. (Section 4.2)
3. *Inference Procedures.* (Section 5)
   While the explicitly known ODs may sometimes not be useful in query processing, the dependencies that logically follow from them might. Thus, an OD inference procedure inside the optimizer is needed to take full advantage of the optimization techniques.
   (a) In Section 5.1 we present a sound and complete *elimination procedure* for inference over ODs.[2] We have implemented this elimination procedure in DB2. Our experiments have shown that the cost of running the elimination procedure is marginal for real world business domains. (This improves over the chase procedure for ODs we presented in [13], as discussed in Section 5, Footnote 7.)
   (b) A restricted domain is introduced, the *transitive domain*, which makes reasoning over ODs simpler.[3] (Section 5.2)
   (c) An efficient, polynomial inference procedure for ODs over the transitive domain is presented that is sound and complete. We have implemented it in DB2. (Section 5.2)
4. *Optimizing with ODs.*
   We demonstrate how ODs can be used effectively in optimization in Sections 2.3 and 5.3.
   (a) We incorporate ODs into a canonical form to enable reasoning over ODs in the query optimizer. (Section 2.3) By casting interesting orders into the canonical form, they can be then matched with indexes, sort orders, and such. The utility of this is demonstrated in a case study with experimental results. (Section 5.3)
   (b) Our technique with ODs between columns and SQL functions has been implemented in DB2. A suit of real-world IBM customer queries over TPC-DS schema show a significant performance gain, with an average 30% improvement over a ten-GB database (Section 5.3).

In Section 6, we discuss related work. In Section 7, we conclude and consider future work.

The results enumerated above are entirely new, with the following clarifications. Point 1c, FD subsumption appears in [14] (where we presented a sound and complete axiomatization for UODs), and is included here for understanding; Point 3a revises the chase procedure of [13] as an improved (see Footnote 7) elimination procedure (new in this work).

This work we feel opens exciting venues for future work to develop a powerful new family of query optimization techniques in database systems.

2. BACKGROUND

First, we establish notational conventions and definitions for ODs and UODs. Next, we introduce an axiomatization [14] which is sound and complete for UODs and sound for ODs which we use in proofs in the paper. Lastly, we discuss how ODs arise and how they can be used for optimization.

2.1 Framework

We adopt the following notational conventions.

- **Relations. R** denotes a *relation* and **r** denotes a specific *relation instance* (*table*). A, B and C denote *attributes*. Additionally, s and t denote *tuples* and t_A denotes the value of attribute A in tuple t.
- **Sets.** $\mathcal{X}$, $\mathcal{Y}$, and $\mathcal{Z}$ denote *sets*. Also, $t_{\mathcal{X}}$ denotes the *projection* of tuple t on $\mathcal{X}$. $\mathcal{XY}$ is shorthand for $\mathcal{X} \cup \mathcal{Y}$.
- **Lists.** **X**, **Y** and **Z** denote *lists*. (Note **X** could represent the empty list, [].) List [A, B, C] denotes an explicit list. [A | **T**] denotes a list with *head* A and *tail* **T**. **XY** is shorthand for **X** ∘ **Y** (**X** concatenate **Y**). Set $\mathcal{X}$ denotes the *set* of elements in *list* **X**. Anyplace a set is expected but a list appears, the list is *cast* to a set; e.g., $t_{\mathbf{X}}$ denotes $t_{\mathcal{X}}$.

We model *order specification* as provided by SQL's order-by clause for specifying lexicographical orderings.

Definition 1. (order specification) An *order specification* is a list of *marked attributes*. There are two directionality operators: asc and desc, indicating *ascending* and *descending*, respectively. Each operator is unary, applies over an attribute, and is written postfix; e.g., A asc and B desc. As shorthand, we write $\overrightarrow{\mathsf{A}}$ and $\overleftarrow{\mathsf{A}}$ for A asc and A desc, respectively. In any context an *order specification* is expected but a list of (unmarked) attributes appears, the list is cast to the order specification with each attribute marked as asc; e.g., [A, B, C] is cast to $[\overrightarrow{\mathsf{A}}, \overrightarrow{\mathsf{B}}, \overrightarrow{\mathsf{C}}]$.[4]

[1] IBM DB2 incorporates such rewrites.

[2] The complexity for elimination procedure is exponential. However, the complexity is with respect to the number of unique attributes in the set of prescribed ODs over relation (which is usually a small number), though, not with respect to data complexity. Hence, it can be used in practice.

[3] A domain is *restricted* if an additional order property is guaranteed over the schema.

[4] *Ascending* is the default for SQL in order-by for any attributes for which directionality is not explicitly indicated.

Table 1: Relational instance $\mathbf{r}$.

#	A	B	C	D	E
s	1	4	4	6	3
t	2	3	4	6	4

The order specification $\mathbf{X}$ defines an algebraic relation '$\preceq_{\mathbf{X}}$'. The operator '$\preceq_{\mathbf{X}}$' defines a *weak total order* over any set of tuples.

Definition 2. (algebraic relation '$\preceq_{\mathbf{X}}$') Let $\mathbf{X}$ be a list of marked attributes. For two tuples r and s (over a schema containing the attributes in $\mathbf{X}$), $r \preceq_{\mathbf{X}} s$ *iff*

- $\mathbf{X} = [\,]$; *or*
- $\mathbf{X} = [\overrightarrow{\mathsf{A}} \mid \mathbf{T}]$ and $r_{\mathsf{A}} < s_{\mathsf{A}}$; or
- $\mathbf{X} = [\overleftarrow{\mathsf{A}} \mid \mathbf{T}]$ and $r_{\mathsf{A}} > s_{\mathsf{A}}$; or
- $\mathbf{X} = [\overrightarrow{\mathsf{A}} \mid \mathbf{T}]$ or $\mathbf{X} = [\overleftarrow{\mathsf{A}} \mid \mathbf{T}]$, $r_{\mathsf{A}} = s_{\mathsf{A}}$, and $r \preceq_{\mathbf{T}} s$.

Let $r \prec_{\mathbf{X}} s$ *iff* $r \preceq_{\mathbf{X}} s$ but $s \not\preceq_{\mathbf{X}} r$.

We now define *order dependencies*.

Definition 3. (order dependency) Let $\mathbf{X}$ and $\mathbf{Y}$ be lists of marked attributes. $\mathbf{X} \mapsto \mathbf{Y}$ denotes an *order dependency* (OD), read as $\mathbf{X}$ *orders* $\mathbf{Y}$. We write $\mathbf{X} \leftrightarrow \mathbf{Y}$, read as $\mathbf{X}$ and $\mathbf{Y}$ are *order equivalent*, *iff* $\mathbf{X}$ *orders* $\mathbf{Y}$ and $\mathbf{Y}$ *orders* $\mathbf{X}$. Let $\mathbf{R}$ be a relation (over a schema that contains the attributes that appear in $\mathbf{X}$ and $\mathbf{Y}$), and let $\mathbf{r}$ be a relation instance of $\mathbf{R}$. Table $\mathbf{r}$ *satisfies* $\mathbf{X} \mapsto \mathbf{Y}$ ($\mathbf{r} \models \mathbf{X} \mapsto \mathbf{Y}$) *iff*, for all $s, t \in \mathbf{r}$, $r \preceq_{\mathbf{X}} s$ implies $r \preceq_{\mathbf{Y}} s$. The OD $\mathbf{X} \mapsto \mathbf{Y}$ is said to *hold* for $\mathbf{R}$ ($\mathbf{R} \models \mathbf{X} \mapsto \mathbf{Y}$) *iff*, for each admissible relational instance $\mathbf{r}$ of $\mathbf{R}$, table $\mathbf{r}$ satisfies $\mathbf{X} \mapsto \mathbf{Y}$. A dependency $\mathbf{X} \mapsto \mathbf{Y}$ is *trivial iff*, for all $\mathbf{r}$, $\mathbf{r} \models \mathbf{X} \mapsto \mathbf{Y}$.

Example 1. *Let* $\mathbf{r}$ *be a relation instance over* $\mathbf{R}$ *with attributes* A, B, C, D, *and* E, *as shown in Table 1. Note* $\mathbf{r} \models [\overleftarrow{\mathsf{C}}, \overleftarrow{\mathsf{A}}] \mapsto [\overrightarrow{\mathsf{B}}, \overrightarrow{\mathsf{D}}, \overleftarrow{\mathsf{E}}]$, *but* $\mathbf{r} \not\models [\overleftarrow{\mathsf{C}}, \overleftarrow{\mathsf{A}}] \mapsto [\overrightarrow{\mathsf{E}}, \overrightarrow{\mathsf{B}}, \overrightarrow{\mathsf{D}}]$.

We introduce one additional order relation, *order compatibility*, as the concept proves invaluable for reasoning about ODs. The empty order specification, [ ], is order compatible with *any* order specification.

Definition 4. (order compatible) order specifications $\mathbf{X}$ and $\mathbf{Y}$ are *order compatible*, denoted as $\mathbf{X} \sim \mathbf{Y}$, *iff* $\mathbf{XY} \leftrightarrow \mathbf{YX}$.

2.2 Unidirectional ODs and Axiomatization

One can consider a simplified version of ODs for which we remove bidirectionality (asc and desc). UODs are a sub-class of ODs, by definition.

Definition 5. (UOD) An OD is *unidirectional* when attributes within it are marked all as asc or all as desc.

In [14], we studied UODs and provided a *sound* and *complete* axiomatization for them (Figure 1).

Theorem 1. [14] *(sound and complete) The set of the axioms from Figure 1 is sound and complete over UODs.*

Theorem 2. *(soundness over ODs) The set of the axioms from Figure 1 is sound over ODs.*

Proof

Given Theorem 1, it is straightforward to show that inference rules (Figure 1) remain *sound* over ODs. □

2.3 Optimization with ODs

We describe how order, and ODs, can be used for query optimization. A database administrator who knows well the semantics of the database can declare ODs as integrity constraints, the same as for FDs. (Note that primary and unique keys are usually declared; this provides much FD information to the optimizer. If the schema is normalized, most FDs will have been thus captured.)

1. *Reflexivity.*
$$\mathbf{XY} \mapsto \mathbf{X}$$
2. *Prefix.*
$$\frac{\mathbf{X} \mapsto \mathbf{Y}}{\mathbf{ZX} \mapsto \mathbf{ZY}}$$
3. *Normalization.*
$$\mathbf{WXYXV} \leftrightarrow \mathbf{WXYV}.$$
4. *Transitivity.*
$$\frac{\mathbf{X} \mapsto \mathbf{Y} \quad \mathbf{Y} \mapsto \mathbf{Z}}{\mathbf{X} \mapsto \mathbf{Z}}$$
5. *Suffix.*
$$\frac{\mathbf{X} \mapsto \mathbf{Y}}{\mathbf{X} \leftrightarrow \mathbf{YX}}$$
6. *Chain.*
$$\frac{\mathbf{X} \sim \mathbf{Y}_1 \quad \forall_{i \in [1,n-1]} \mathbf{Y}_i \sim \mathbf{Y}_{i+1} \quad \mathbf{Y}_n \sim \mathbf{Z} \quad \forall_{i \in [1,n]} \mathbf{Y}_i\mathbf{X} \sim \mathbf{Y}_i\mathbf{Z}}{\mathbf{X} \sim \mathbf{Z}}$$

Figure 1: Axioms for UODs.

However, OD optimization techniques are also applicable even when the database has no declared ODs. ODs can be implied by queries' semantics. For example, if there is a predicate $\mathsf{A} = \mathsf{B}$, then an OD $[\mathsf{A}] \leftrightarrow [\mathsf{B}]$ is satisfied within the scope of the query. ODs also arise through SQL functions and algebraic expressions. For instance, UODs [d_date] $\mapsto$ [year(d_date)] and [d_date] $\mapsto$ [d_date + 30 days] hold [16].

Even with the ODs declared for the database and the local ODs deduced within the scope of the query, the optimizer might miss opportunities. There may be an OD that logically follows from the declared and local ODs that would allow for a better plan, while none of the declared or local ODs match directly. For instance, again assume there is a predicate A = B in the where clause and an index on B. If we also knew the declared OD [A] $\mapsto$ [Z], within the query's scope, OD [B] $\mapsto$ [Z] is also satisfied by *transitivity* (Figure 1) of ODs. Therefore, the optimizer has a need to *infer* ODs from others. This is the subject of our work.

We motivate *order dependencies* in analogy to *functional dependencies*. ODs are to order-by as FDs are to group-by. ODs might be used in query optimization [15, 16] just as FDs have been before [12]. In [15], we showed how ODs can provide significant performance improvement by *eliminating joins* from query plans in a data warehouse environment. We built a prototype in IBM DB2 V.9.7 and performed experiments over the TPC-DS benchmark queries. In this paper, we show by running experiments over the TPC-DS schema over IBM customer driven queries how ODs between columns and SQL functions and algebraic expressions over columns can bring benefits for queries that involve a sort operator. (The extended version appears in [16].) This is illustrated by Query 2 and generalized by Reduce Order OD algorithm presented below.

Query 2. *Substring with group-by.*

```
select substr(s_zip, 1, 2) as area,
       count(distinct s_zip) as cnt,
       sum(ss_net_profit) as net
from store_sales, store
where ss_store_sk = s_store_sk
group by substr(s_zip, 1, 2);
```

Let there be an index on s_zip in table store. It is obvious that the column s_zip orders the derived column substr(s_zip, 1, 2), [s_zip] $\mapsto$ [substr(s_zip, 1, 2)]. We call substr(s_zip, 1, 2) a *generated attribute* (Definition 6, below). Given the optimizer detects this OD, it can choose to do an index scan using the index on s_zip to accomplish the group-by on-the-fly (called partial group-by); then no partitioning or sort operator would be needed in the query plan. Note that a clever SQL programmer could not rewrite Query 2 manually with group by s_zip to avoid this issue, since the substring changes the partition of the group-by.

Table 2: An instance of the table date.

date_id	date	year	month	day	quarter	trimester
8300	20100830	2010	08	30	3	2
8301	20100931	2010	09	31	3	3
8302	20110105	2011	01	05	1	1
8304	20110401	2011	04	01	2	1

Definition 6. (generated attribute) A generated attribute is an attribute computed from other columns using algebraic expressions and SQL functions.

EXAMPLE 2. (generated attribute) *Let* A = year(d_date) *100 + month(d_month). *Thus,* A *is a* generated attribute.

The following example shows that DB2 provides the capability to specify generated attributes in a table definition which can also provide implicit ODs.

EXAMPLE 3. *In DB2, the following definition of a table is possible:* create table date_dim ... date d_date, integer d_year generated always as year(d_date), ..., *therefore, the table* date_dim *includes an implicit OD* [d_date] $\mapsto$ [d_year].

Assume that we are aware of an OD $\mathbf{X} \mapsto \mathbf{Y}$. Therefore, a query with `order by` $\mathbf{Y}$ can be rewritten with `order by` $\mathbf{X}$. Note that the original and rewritten queries are *not* semantically equivalent, unless $\mathbf{X} \leftrightarrow \mathbf{Y}$. The rewritten query satisfies the order of the original query, but not necessary vice versa. That is, order *equivalency* is not required for correct query rewrites. Similarly, `group by` $\mathbf{Y}$ can be accomplished on-the-fly with $\mathbf{X}$ (a partial group-by) as described in Query 2. Directional order dependencies ($\mathbf{X} \mapsto \mathbf{Y}$) are sufficient to provide a wide variety of query rewrites.

The critical role of *interesting orders* was recognized quite early on [11, 12]. Because we are interested in ordered streams between operators in the query plan (to allow for pipelining, selecting more efficient procedures, and eliminating intermediate sort and partitioning steps), the optimizer needs to track which stream orders are possible to generate by alternative sub-plans. The ones that the optimizer tracks during query plan construction are called *interesting orders*. This is useful for processing order-by, group-by, distinct, partition-by and join. The authors of [12] designed and implemented in DB2 the algorithm *Reduce Order*, which scans the interesting order list backwards to test if any of the attributes can be eliminated using FD information. (In their Reduce Order algorithm, a given set of FDs can be used to infer other FDs with an inference procedure for FDs [1].) However, this technique relies on FD information; it does not incorporate ODs. We extend further the techniques of [12] by also employing ODs to extend significantly the range of these optimization techniques. We call this extended algorithm *Reduce Order OD*. (To take full advantage of our Reduce Order OD algorithm the optimizer needs an inference procedure for ODs such we provide in Section 5.)

Our changes are for putting ODs into canonical form.[5] We extend Reduce Order algorithm by iterating through the list, additionally checking following.[6]

Algorithm 1 Reduce Order OD

Input: ODs $\mathcal{M}$ and order list $\mathbf{O} = [\mathsf{O}_0, \mathsf{O}_1, ..., \mathsf{O}_{n-1}]$.
Output: The reduced version of $\mathbf{O}$.

1: **for** $i \leftarrow 0$ **to** $n-1$ **do**
2: **if** O_i is generated attribute from G and $\mathsf{O}_i \leftrightarrow \mathsf{G}$ **then**
3: $\mathbf{O} = [\mathsf{O}_0, ..., \mathsf{O}_{i-1}, \mathsf{G}, \mathsf{O}_{i+1}, ..., \mathsf{O}_{n-1}]$
4: Rewrite $\mathbf{O}$ in terms of each column's equivalence class head.
5: **for** $i \leftarrow n-1$ **to** 0 **do**
6: Let $\mathsf{B} = \{\mathsf{O}_0, ..., \mathsf{O}_{i-1}\}$
7: **if** order specification is a single ($\mathbf{O} = [\mathsf{O}_0]$), generated attribute from G and $\mathsf{G} \mapsto \mathsf{O}_0$ **then**
8: $\mathbf{O} = [\mathsf{G}]$
9: **else if** $\mathsf{B} \rightarrow \mathsf{O}_i$ **then**
10: Remove O_i from $\mathbf{O}$
11: **else if** $[\mathsf{O}_0, ..., \mathsf{O}_{i-1}, \mathsf{O}_{i+1}, ..., \mathsf{O}_{n-1}] \mapsto [\mathsf{O}_0, ..., \mathsf{O}_{n-1}]$ **then**
12: Remove O_i from $\mathbf{O}$
13: **return** $\mathbf{O}$

- Whether the currently considered attribute O_i is a generated attribute from G and $\mathsf{O}_i \leftrightarrow \mathsf{G}$. If so, the attribute O_i is replaced by the attribute G in the list, $\mathbf{O} = [\mathsf{O}_0, ..., \mathsf{O}_{i-1}, \mathsf{G}, \mathsf{O}_{i+1}, ..., \mathsf{O}_{n-1}]$.
- If the order specification is a single, generated attribute from an attribute G (Call this attribute O_0.) and $\mathsf{G} \mapsto \mathsf{O}_0$, the attribute O_0 is replaced by G in the list, $\mathbf{O} = [\mathsf{G}]$. (Detecting monotonicity property for generated attributes is described in detail in [16].)
- Whether the *list* without the attribute being currently considered *orders* the full list. If so, the attribute is dropped from the current list.

The algorithm *Reduce Order OD* is correct because removing O_i from the list using a FD $\mathsf{B} \rightarrow \mathsf{O}_i$ is part of *Reduce Order* algorithm described in [12] (Algorithm 1, lines 9–10). Given an OD $\mathbf{X} \mapsto \mathbf{Y}$, the clause order by $\mathbf{Y}$, can be rewritten with order by $\mathbf{X}$, as strengthening the order-by conditions is allowed as described earlier (Algorithm 1, lines 7–8 and Algorithm 1, lines 11–12). It is also sound to replace order equivalent attributes ($\mathsf{O}_i \leftrightarrow \mathsf{G}$, Algorithm 1 lines 2–3).

Time and *date* (Example 4) are supported in the SQL standard in a rich manner. The TPC-DS benchmark consists of 99 queries. Of these, 85 involve date operators and predicates and five involve time operators and predicates.

EXAMPLE 4. (canonical form) *In Table 2 (*date*) an OD* [year, month, day] $\mapsto$ [year, trimester, quarter, month, day] *holds. Based on this OD, Algorithm 1 is able to eliminate* trimester *and* quarter*, simplifying the interesting order list* year, trimester, quarter, month, day *to* year, month, day*. This is useful for Query 1 as described in Section 1.*

Even if the concept of ODs was only applied to date and time, it could still be of great use for query optimization, as shown by Query 1. However, ordered domains are not only limited to date and time. They arise in many other domains from business semantics, such as sequence numbers, surrogate keys, measured values such as sales, salaries, stock prices, and taxes. (See [14] and [16].)

3. A HIERARCHY OF OD CLASSES

[5]The main body of the Reduce Order algorithm from [12] are lines 4 and 9−10 in Algorithm 1.

[6]In [14], we focused on order optimization techniques based on an axiomatization of ODs. In each iteration through the list, we had been additionally checking whether *any* postfix list with respect to the current attribute—that is, a list of the attributes to the right of the current—*orders* the current attribute. Therefore, the main body was a double-nested loop. If so, the attribute is dropped from the list. Our Reduce Order OD algorithm here is a single-nested loop. The algorithm in [14] does not consider generated attributes based on algebraic expressions and SQL functions. Thus, Algorithm 1 is more general and efficient.

One can define *classes* of ODs. We need to say formally what it means for one class of dependencies to *(strictly) generalize* another.

Definition 7. (*Class* **A** *generalizes class* **B**.)

Dependency class **A** *generalizes* dependency class **B** *iff* there is a *semantically preserving mapping* of any arbitrary dependency of class **B** into a set of dependencies of class **A**. Let mapping σ map dependencies from class **B** into sets of dependencies in class **A**. Mapping σ is *semantically preserving iff*, for any table **r**, for any B of class **B**, $\mathbf{t} \models B \iff \mathbf{t} \models \bigwedge \sigma(B)$. (Additionally, mapping σ is polynomial *iff* there is a polynomial-time algorithm that implements it.) Class **A** *strictly generalizes* class **B** *iff* **A** generalizes **B** but **B** does not generalize **A**. If **A** (strictly) generalizes **B**, we also say then that **B** is a (proper) sub-class of **A**.

In Section 3.1, we characterize the inference problem for ODs. We then establish a strict hierarchy of classes of ODs, Sections 3.2–3.4.

3.1 Violations

Table **t** does not *satisfy* an OD if there exist a *pair* of tuples from **t** that *violates* (*falsifies*) the dependency. Since an ordered pair suffices to represent a violation, one can rewrite the two tuples with just the values 0 and 1, while preserving the relative order between columns's values between the two tuples, without loss of generality. Thus, to answer a question of an inference problem for ODs (Definition 8), it would suffice to evaluate every pair of tuples composable over 0 and 1. For every such pair, if the pair does not violate any OD in the prescribed collection, and also does not violate the target dependency, then the target is logically entailed. This procedure directly establishes that the inference problem for ODs is *decidable*.

Definition 8. ($\mathcal{M} \models \mathbf{X} \mapsto \mathbf{Y}$) The inference problem for ODs is, given a set of ODs $\mathcal{M}$ and an OD $\mathbf{X} \mapsto \mathbf{Y}$, to decide whether $\mathcal{M} \models \mathbf{X} \mapsto \mathbf{Y}$.

An OD $\mathbf{X} \mapsto \mathbf{Y}$ can be violated (*falsified*) in two ways, as by Theorem 3: by *splits* and *swaps*.

THEOREM 3. *(order dependency) For every instance* **r** *of relation* ***R****,* $\mathbf{X} \mapsto \mathbf{Y}$ *iff* $\mathbf{X} \mapsto \mathbf{XY}$ *and* $\mathbf{XY} \mapsto \mathbf{YX}$.

Proof

IF: Suppose $\mathbf{X} \mapsto \mathbf{Y}$. By the Suffix rule $\mathbf{X} \leftrightarrow \mathbf{YX}$. By Prefix and Normalization $\mathbf{X} \mapsto \mathbf{XY}$ and $\mathbf{XY} \leftrightarrow \mathbf{YX}$.

ONLY IF: Assume that $\mathbf{X} \mapsto \mathbf{XY}$ and $\mathbf{XY} \leftrightarrow \mathbf{YX}$. By Transitivity, $\mathbf{X} \mapsto \mathbf{YX}$. Therefore, by Reflexivity and Transitivity, $\mathbf{X} \mapsto \mathbf{Y}$. □

Definition 9. (split) A *split* with respect to an OD $\mathbf{X} \mapsto \mathbf{XY}$ is a pair of tuples s and t such that $s_{\mathcal{X}} = t_{\mathcal{X}}$ but $s_{\mathcal{Y}} \neq t_{\mathcal{Y}}$. This says that $\mathcal{X}$ does not functionally determine $\mathcal{Y}$.

Definition 10. (swap) A *swap* with respect to an OD $\mathbf{XY} \leftrightarrow \mathbf{YX}$ is a pair of tuples s and t such that $s \prec_{\mathbf{X}} t$, but $t \prec_{\mathbf{Y}} s$; Thus, the swap falsifies $\mathbf{X} \sim \mathbf{Y}$.

EXAMPLE 5. (split and swap) *There is a split in Table 1 with respect to an OD* $[\overleftarrow{\mathsf{C}}, \overleftarrow{\mathsf{D}}] \mapsto [\overleftarrow{\mathsf{C}}, \overleftarrow{\mathsf{D}}, \overrightarrow{\mathsf{A}}, \overrightarrow{\mathsf{B}}]$ *and a swap in Table 1 with respect to an OD* $[\overleftarrow{\mathsf{C}}, \overrightarrow{\mathsf{A}}] \sim [\overleftarrow{\mathsf{D}}, \overrightarrow{\mathsf{B}}]$.

3.2 Pointwise generalizes Lexicographical

The class of pointwise order dependencies was proposed in the context of database systems in [5]. The type of dependency looks rather different than the lexicographical ODs we have presented. The pointwise OD $\mathcal{X} \rightsquigarrow \mathcal{Y}$ holds if the order over the values of *each* attribute of $\mathcal{X}$ implies order over the values of *each* attribute of $\mathcal{Y}$. Both $\mathcal{X}$ and $\mathcal{Y}$ are *sets*. Let us restrict our interest to domains for which values are comparable. Then, each order condition is a marked attribute A^{op} for which $op \in \{<, >, \leq, \geq, =\}$. For any table **r**, **r** satisfies $\mathcal{X} \rightsquigarrow \mathcal{Y}$ *iff*, for any tuples $s, t \in \mathbf{r}$, if, for each A^{op} in $\mathcal{X}$, s_{A} *op* t_{A}, then, for each B^{op} in $\mathcal{Y}$, s_{B} *op* t_{B}.

While lexicographical ODs have been studied since , it has never been established how they are related with pointwise.

LEMMA 1. *There exists a semantically preserving, polynomial mapping of a lexicographical OD into a set of pointwise ODs.*

Proof

Algorithm 2 provides a polynomial mapping of an arbitrary lexicographical OD into a set of $\mathcal{E}$ of pointwise ODs. Any split or swap that violates the lexicographical OD $\mathbf{X} \mapsto \mathbf{Y}$ violates some pointwise OD in $\mathcal{E}$, and vice versa. □

The mapping requires a quadratic number of pointwise ODs in the size of the lexicographical OD.

LEMMA 2. *There exists a pointwise OD that cannot be mapped into a set of lexicographical ODs.*

Proof

Consider a table **t** (Table 3). Pointwise OD $\mathsf{A}^{>}\mathsf{B}^{>} \rightsquigarrow \mathsf{C}^{>}$ is satisfied by table **t**. However, it is straightforward to show that table **t** as we constructed consists of all possible *splits* (Definition 9, rows a–l) and *swaps* (Definition 10, rows a–f and m–r) defined for ODs over marked attributes $\overleftarrow{\mathsf{A}}$, $\overrightarrow{\mathsf{A}}$, $\overleftarrow{\mathsf{B}}$, $\overrightarrow{\mathsf{B}}$, $\overleftarrow{\mathsf{C}}$, and $\overrightarrow{\mathsf{C}}$ are falsified by table **t**. □

THEOREM 4. *The class of pointwise ODs strictly generalizes the class of lexicographical ODs.*

Proof

There exists a semantically preserving, polynomial mapping

Table 3: Table **t**.

#	A	B	C
a	0	0	0
b	0	1	1
c	2	2	2
d	3	2	3
e	4	4	4
f	5	5	4
g	6	6	6
h	7	6	6
i	8	8	8
j	8	9	8

#	A	B	C
k	10	10	10
l	10	10	11
m	12	12	13
n	12	13	12
o	14	14	15
p	15	14	14
q	16	17	16
r	17	16	16

Algorithm 2 Translation

Input: Lexicographical OD $\mathbf{X} \mapsto \mathbf{Y}$, where $\mathbf{X} = [\mathsf{X}_0, ..., \mathsf{X}_{m-1}]$ and $\mathbf{Y} = [\mathsf{Y}_0, ..., \mathsf{Y}_{n-1}]$

Output: A set of pointwise ODs $\mathcal{E}$ semantically equivalent to lexicographical OD $\mathbf{X} \mapsto \mathbf{Y}$.

1: $\mathcal{E} = \{\mathsf{A}_0^{=}, ..., \mathsf{A}_{m-1}^{=} \rightsquigarrow \mathsf{A}_0^{=}, ..., \mathsf{A}_{n-1}^{=}\}$
2: **for** $i \leftarrow 0$ **to** $m-1$ **do**
3: **for** $j \leftarrow 0$ **to** $n-1$ **do**
4: **if** $\mathsf{X}_i = \overrightarrow{\mathsf{A}}_i$ and $\mathsf{Y}_j = \overrightarrow{\mathsf{B}}_j$ **then**
5: $\mathcal{E} = \mathcal{E} \cup \{\mathsf{A}_0^{=}, ..., \mathsf{A}_i^{>} \rightsquigarrow \mathsf{B}_0^{=}, ..., \mathsf{B}_j^{\geq}\}$
6: **else if** $\mathsf{X}_i = \overrightarrow{\mathsf{A}}_i$ and $\mathsf{Y}_j = \overleftarrow{\mathsf{B}}_j$ **then**
7: $\mathcal{E} = \mathcal{E} \cup \{\mathsf{A}_0^{=}, ..., \mathsf{A}_i^{>} \rightsquigarrow \mathsf{B}_0^{=}, ..., \mathsf{B}_j^{\leq}\}$
8: **else if** $\mathsf{X}_i = \overleftarrow{\mathsf{A}}_i$ and $\mathsf{Y}_j = \overrightarrow{\mathsf{B}}_j$ **then**
9: $\mathcal{E} = \mathcal{E} \cup \{\mathsf{A}_0^{=}, ..., \mathsf{A}_i^{<} \rightsquigarrow \mathsf{B}_0^{=}, ..., \mathsf{B}_j^{\geq}\}$
10: **else if** $\mathsf{X}_i = \overleftarrow{\mathsf{A}}_i$ and $\mathsf{Y}_j = \overleftarrow{\mathsf{B}}_j$ **then**
11: $\mathcal{E} = \mathcal{E} \cup \{\mathsf{A}_0^{=}, ..., \mathsf{A}_i^{<} \rightsquigarrow \mathsf{B}_0^{=}, ..., \mathsf{B}_j^{\leq}\}$

for any set of lexicographical ODs to a set of pointwise ODs, (Lemma 1). Additionally, the class of pointwise ODs are more expressive than lexicographical ODs (Lemma 2). □

In [5], the authors demonstrated that the inference problem for pointwise ODs in general is co-NP-complete. By Theorem 4 and that mapping is polynomial, this sets an upper bound for the inference problem for lexicographical ODs. However, the problem for lexicographical ODs is just as hard, as we prove in Section 4.

3.3 ODs generalize UODs

With bidirectionality, ODs are more expressive than UODs. Given that UODs are a syntactic sub-class of ODs, it then follows that the the class of ODs strictly generalize UODs.

THEOREM 5. *The class of ODs strictly generalizes the class of UODs.*

Proof

An arbitrary UOD is an OD by Definition 5, which establishes direct mapping. By definition ODs are more expressive than UODs. □

3.4 UODs generalize FDs

Any OD implies an FD, modulo lists and sets.

LEMMA 3. *(relationship) For every instance* $\mathbf{r}$ *of relation* $\mathbf{R}$*, if an UOD* $\mathbf{X} \mapsto \mathbf{Y}$ *holds, then the FD* $\mathcal{X} \rightarrow \mathcal{Y}$ *is true.*

Proof

Let rows s and $t \in \mathbf{r}$. Assume that $s_{\mathcal{X}} = t_{\mathcal{X}}$. Hence, $s \preceq_{\mathbf{X}} t$ and $t \preceq_{\mathbf{X}} s$. By definition of an OD, $s \preceq_{\mathbf{Y}} t$ and $t \preceq_{\mathbf{Y}} s$. Therefore, $s_{\mathcal{Y}} = t_{\mathcal{Y}}$ holds. □

Furthermore, there exists a correspondence between FDs and UODs.

THEOREM 6. *(correspondence) For relation* $\mathbf{R}$*, for every instance* $\mathbf{r}$*,* $\mathcal{X} \rightarrow \mathcal{Y}$ iff $\mathbf{X} \mapsto \mathbf{XY}$*, for any list* $\mathbf{X}$ *over the attributes of* $\mathcal{X}$ *and any list* $\mathbf{Y}$ *over the attributes of* $\mathcal{Y}$*.*

Proof

IF: Assume an UOD $\mathbf{X} \mapsto \mathbf{XY}$ does not hold. This means, there exist s and $t \in \mathbf{r}$, such that $s \preceq_{\mathbf{X}} t$ but $s \not\preceq_{\mathbf{XY}} t$ by Definition 5. Therefore, $s_{\mathcal{X}} = t_{\mathcal{X}}$ and $s \prec_{\mathbf{Y}} t$. Also $s \prec_{\mathbf{Y}} t$ implies that $s_{\mathcal{Y}} \neq t_{\mathcal{Y}}$. Therefore, $\mathcal{X} \rightarrow \mathcal{Y}$ is not satisfied.

ONLY IF: By Lemma 3 if $\mathbf{X} \mapsto \mathbf{XY}$, then $\mathcal{X} \rightarrow \mathcal{XY}$. The FD $\mathcal{XY} \rightarrow \mathcal{Y}$ holds by Armstrong's axiom of Reflexivity [1]. Hence by Armstrong's axiom of Transitivity, $\mathcal{X} \rightarrow \mathcal{Y}$. □

LEMMA 4. *The class of UODs is more expressive than the class of FDs.*

Proof

Consider the table $\mathbf{t}'$ with attributes A and B with the values the same as in Table 3 in rows a), b), c) and d), respectively. The UOD $[\mathsf{A}] \sim [\mathsf{B}]$ is satisfied in table $\mathbf{t}'$. However, in $\mathbf{t}'$, all possible non-trivial FDs over A and B are falsified. □

THEOREM 7. *The class of UODs strictly generalizes the class of FDs.*

Proof

Theorem 6 provides a mapping of an arbitrary FD $\mathcal{X} \rightarrow \mathcal{Y}$ into an UOD $\mathbf{X} \mapsto \mathbf{XY}$. Lemma 4 shows that UODs convey additional semantic. □

4. COMPLEXITY

We show that the inference problem for ODs is co-NP-complete. More specifically, we show that inference problem for UODs and the inference problem of FDs from ODs are co-NP-complete. FD inference from UODs, a restricted case, is polynomially decidable, however.

4.1 OD Inference

We introduce first the notation which permits us to translate instances of 3-SAT into instances of the decision problem for inference problem for ODs.

Definition 11. Let $\mathcal{P} = \{p_1, \ldots, p_n\}$ be a set of propositional variables for an arbitrary finite n, and let $\overline{\mathcal{P}} = \{\neg p_1, \ldots, \neg p_n\}$. Let $\mathcal{F}$ be a formula written over the propositional variables in $\mathcal{P}$ and their negations in conjunctive normal form with k clauses, each a disjunction of length three, for an arbitrary finite k. For $i \in \{1, \ldots, k\}$, let $\mathsf{V}_{i,1} \vee \mathsf{V}_{i,2} \vee \mathsf{V}_{i,3}$ represent clause i such that $\mathsf{V}_{i,1} \in (\mathcal{P} \cup \overline{\mathcal{P}})$, $\mathsf{V}_{i,2} \in (\mathcal{P} \cup \overline{\mathcal{P}}) - \{\mathsf{V}_{i,1}\}$, and $\mathsf{V}_{i,3} \in (\mathcal{P} \cup \overline{\mathcal{P}}) - \{\mathsf{V}_{i,1}, \mathsf{V}_{i,2}\}$, without loss of generality. Call any such $\mathcal{F}$ a *3-SAT candidate*. Call any such *3-SAT candidate* $\mathcal{F}$ for which there exists a truth assignment over $\mathcal{F}$'s $\mathcal{P}$ which satisfies $\mathcal{F}$ a *3-SAT instance*. **3-SAT** is the collection of 3-SAT instances.

LEMMA 5. [4] **3-SAT** *is NP-complete.*

LEMMA 6. *Given a set* $\mathcal{M}$ *of UODs and UOD* $[\mathsf{A}] \sim [\mathsf{B}]$*, deciding whether* $\mathcal{M} \models [\mathsf{A}] \sim [\mathsf{B}]$ *is co-NP-complete.*

Proof

Candidate and instance. Given a 3-SAT candidate $\mathcal{F}$ (Definition 11), we construct an UODI candidate $\langle \mathcal{M}_{\mathcal{F}}, [\mathsf{T}] \sim [\mathsf{F}] \rangle$. Let $\langle \mathcal{M}, \mathbf{X} \mapsto \mathbf{Y} \rangle$ be an arbitrary pair of a finite set $\mathcal{M}$ of UODs and a *target* UOD $\mathbf{X} \mapsto \mathbf{Y}$ constructed over the attributes that appear in $\mathcal{M}$. Call any such $\langle \mathcal{M}, \mathbf{X} \mapsto \mathbf{Y} \rangle$ an *UODI candidate*. Call any such $\langle \mathcal{M}, \mathbf{X} \mapsto \mathbf{Y} \rangle$ for which $\mathcal{M} \models \mathbf{X} \mapsto \mathbf{Y}$ an *UODI instance*. **UODI** is the collection of UODI instances. This is the set-theoretic characterization of *the inference decision problem for UODs*.

Reduction from 3-SAT. Construction. $\mathcal{M}_{\mathcal{F}}$ is constructed as follows. For each p_i, $i \in \{1, \ldots, n\}$, from $\mathcal{F}$, we introduce four attributes to appear in $\mathcal{M}_{\mathcal{F}}$: $\mathsf{P}_{i,t}$, $\mathsf{P}_{i,f}$, $\mathsf{Q}_{i,t}$, and $\mathsf{Q}_{i,f}$. (Our intent is that $[\mathsf{P}_{i,t}, \mathsf{P}_{i,f}]$ will mirror the truth value of p_i from $\mathcal{F}$ in a given truth assigment, and $[\mathsf{Q}_{i,t}, \mathsf{Q}_{i,f}]$ will mirror the truth value of $\neg p_i$ in that truth assigment.) For $i \in \{1, \ldots, n\}$, add the following order dependencies for $\mathsf{P}_{i,t}$ and $\mathsf{P}_{i,f}$ to $\mathcal{M}_{\mathcal{F}}$:

1. $[\mathsf{P}_{i,t}] \sim [\mathsf{T}]$ 2. $[\mathsf{P}_{i,f}] \sim [\mathsf{F}]$
3. $[\mathsf{P}_{i,t}] \sim [\mathsf{P}_{i,f}]$ 4. $[\mathsf{P}_{i,t}, \mathsf{P}_{i,f}, \mathsf{T}] \sim [\mathsf{P}_{i,t}, \mathsf{P}_{i,f}, \mathsf{F}]$

Likewise, for $i \in \{1, \ldots, n\}$, symmetrically add the "same" order dependencies for $\mathsf{Q}_{i,t}$ and $\mathsf{Q}_{i,f}$ to $\mathcal{M}_{\mathcal{F}}$:

5. $[\mathsf{Q}_{i,t}] \sim [\mathsf{T}]$ 6. $[\mathsf{Q}_{i,f}] \sim [\mathsf{F}]$
7. $[\mathsf{Q}_{i,t}] \sim [\mathsf{Q}_{i,f}]$ 8. $[\mathsf{Q}_{i,t}, \mathsf{Q}_{i,f}, \mathsf{T}] \sim [\mathsf{Q}_{i,t}, \mathsf{Q}_{i,f}, \mathsf{F}]$

For $i \in \{1, \ldots, n\}$, add to $\mathcal{M}_{\mathcal{F}}$:

9. $[\mathsf{P}_{i,t}, \mathsf{Q}_{i,t}, \mathsf{T}] \sim [\mathsf{P}_{i,t}, \mathsf{Q}_{i,t}, \mathsf{F}]$
10. $[\mathsf{P}_{i,f}, \mathsf{Q}_{i,f}, \mathsf{T}] \sim [\mathsf{P}_{i,f}, \mathsf{Q}_{i,f}, \mathsf{F}]$

Next, we encode the clauses. For each clause, $i \in \{1, \ldots, k\}$, from $\mathcal{F}$, we introduce three attributes: $\mathsf{V}_{i,1}$, $\mathsf{V}_{i,2}$, and $\mathsf{V}_{i,3}$. For $i \in \{1, \ldots, k\}$, $j \in \{1, \ldots, 3\}$, add one OD to $\mathcal{M}_{\mathcal{F}}$ as follows. If $\mathsf{V}_{i,j} = p_l$ (for a given $l \in \{1, \ldots, n\}$) in $\mathcal{F}$, add to $\mathcal{M}_{\mathcal{F}}$:

11. $[\mathsf{V}_{i,j}] \sim [\mathsf{P}_{l,t}, \mathsf{P}_{l,f}]$

Else, $\mathsf{V}_{i,j} = \neg p_l$ (for a given $l \in \{1, \ldots, n\}$) in $\mathcal{F}$; add to $\mathcal{M}_{\mathcal{F}}$:

12. $[\mathsf{V}_{i,j}] \sim [\mathsf{Q}_{l,t}, \mathsf{Q}_{l,f}]$

Finally, for each clause $i \in \{1, \ldots, k\}$ in $\mathcal{F}$, we introduce an attribute C_i, and we add to $\mathcal{M}_{\mathcal{F}}$:

13. $[\mathsf{C}_i] \mapsto [\mathsf{T}]$ 14. $[\mathsf{C}_i] \mapsto [\mathsf{V}_{i,1}, \mathsf{V}_{i,2}, \mathsf{V}_{i,3}, \mathsf{F}]$

Polynomial reduction. The translation procedure above of a 3-SAT candidate into an UODI candidate is clearly polynomial in the size of $\mathcal{F}$.

Witness. We can build a counter-example for a given UODI candidate to demonstrate that it is *not* an UODI in-

stance, in **UODI**. A pair of tuples is necessary and sufficient to falsify $[\mathsf{T}] \sim [\mathsf{F}]$. Therefore, $\mathcal{M}_\mathcal{F} \not\models [\mathsf{T}] \sim [\mathsf{F}]$ *iff* we can construct a two-tuple table $\mathbf{t}$ over the attributes appearing in $\mathcal{M}_\mathcal{F}$ that falsifies $[\mathsf{T}] \sim [\mathsf{F}]$, but that does not falsify any order dependency in $\mathcal{M}_\mathcal{F}$ (thus *satisfies* $\mathcal{M}_\mathcal{F}$). Between the two tuples in $\mathbf{t}$, T will have different values, F will have different values, and the values of T and F will be anti-monotonic. Let the two values for T and for F in $\mathbf{t}$ be 0 and 1, without loss of generality. We write the tuples in $\mathbf{t}$ in a fixed order in our discussion such that $t_{\mathsf{T},\mathsf{F}} = \left[\begin{smallmatrix}0&1\\1&0\end{smallmatrix}\right]$, without loss of generality. Conceptually, a transition from 0 to 1, as in $t_\mathsf{T} = \left[\begin{smallmatrix}0\\1\end{smallmatrix}\right]$, encodes *true*; from 1 to 0, as in $t_\mathsf{F} = \left[\begin{smallmatrix}1\\0\end{smallmatrix}\right]$, represents *false*.

We can always build a two-tuple table $\mathbf{t}$ such that $t_{\mathsf{T},\mathsf{F}} = \left[\begin{smallmatrix}0&1\\1&0\end{smallmatrix}\right]$ (which is necessary and sufficient to falsify $[\mathsf{T}] \sim [\mathsf{F}]$) which satisfies ODs 1–13 of $\mathcal{M}_\mathcal{F}$. Let us construct such a $\mathbf{t}$. Because of OD 1, $t_{\mathsf{P}_{i,t}} = \left[\begin{smallmatrix}0\\0\end{smallmatrix}\right]$ *or* $\left[\begin{smallmatrix}0\\1\end{smallmatrix}\right]$. (If only a single value appears in $\mathbf{t}$ for an attribute, we can assume that value is 0, without loss of generality.) Because of OD 2, $t_{\mathsf{P}_{i,f}} = \left[\begin{smallmatrix}0\\0\end{smallmatrix}\right]$ *or* $\left[\begin{smallmatrix}1\\0\end{smallmatrix}\right]$. Because of OD 3, $t_{\mathsf{P}_{i,t},\mathsf{P}_{i,f}} \neq \left[\begin{smallmatrix}0&1\\1&0\end{smallmatrix}\right]$. Because of OD 4, $t_{\mathsf{P}_{i,t},\mathsf{P}_{i,f}} \neq \left[\begin{smallmatrix}0&0\\0&0\end{smallmatrix}\right]$. (Otherwise, OD 4 would be falsified by $\mathbf{t}$, since $t_{\mathsf{T},\mathsf{F}} = \left[\begin{smallmatrix}0&1\\1&0\end{smallmatrix}\right]$.) Therefore, $t_{\mathsf{P}_{i,t},\mathsf{P}_{i,f}} = \left[\begin{smallmatrix}0&1\\0&0\end{smallmatrix}\right]$ *or* $\left[\begin{smallmatrix}0&0\\1&0\end{smallmatrix}\right]$.

From ODs 5–8, it symmetrically follows that $t_{\mathsf{Q}_{i,t},\mathsf{Q}_{i,f}} = \left[\begin{smallmatrix}0&1\\0&0\end{smallmatrix}\right]$ *or* $\left[\begin{smallmatrix}0&0\\1&0\end{smallmatrix}\right]$.

From ODs 9–10, it further follows that

$t_{\mathsf{P}_{i,t},\mathsf{P}_{i,f},\mathsf{Q}_{i,t},\mathsf{Q}_{i,f}} = \left[\begin{smallmatrix}0&0&0&1\\1&0&0&0\end{smallmatrix}\right]$ *or* $t_{\mathsf{P}_{i,t},\mathsf{P}_{i,f},\mathsf{Q}_{i,t},\mathsf{Q}_{i,f}} = \left[\begin{smallmatrix}0&1&0&0\\0&0&1&0\end{smallmatrix}\right]$.

Thus, $\mathbf{t} \models [\mathsf{P}_{i,t}, \mathsf{P}_{i,f}] \not\sim [\mathsf{Q}_{i,t}, \mathsf{Q}_{i,f}]$.

For any $\mathsf{V}_{i,j}$ such that $V_{i,j} = p_l$ for a given l in $\mathcal{F}$, so OD 11 is in $\mathcal{M}_\mathcal{F}$ for i, we know the following:

- if $t_{\mathsf{P}_{l,t},\mathsf{P}_{l,f}} = \left[\begin{smallmatrix}0&0\\1&0\end{smallmatrix}\right]$, then $t_{\mathsf{V}_{i,j}} = \left[\begin{smallmatrix}0\\0\end{smallmatrix}\right]$ *or* $\left[\begin{smallmatrix}0\\1\end{smallmatrix}\right]$;
- else $t_{\mathsf{P}_{l,t},\mathsf{P}_{l,f}} = \left[\begin{smallmatrix}0&1\\0&0\end{smallmatrix}\right]$ and $t_{\mathsf{V}_{i,j}} = \left[\begin{smallmatrix}0\\0\end{smallmatrix}\right]$ *or* $\left[\begin{smallmatrix}1\\0\end{smallmatrix}\right]$.

For any $\mathsf{V}_{i,j}$ such that $V_{i,j} = \neg p_l$ for a given l in $\mathcal{F}$ *instead*, so OD 12 is in $\mathcal{M}_\mathcal{F}$ for i, we know the following:

- if $t_{\mathsf{P}_{l,t},\mathsf{P}_{l,f}} = \left[\begin{smallmatrix}0&0\\1&0\end{smallmatrix}\right]$, then $t_{\mathsf{Q}_{l,t},\mathsf{Q}_{l,f}} = \left[\begin{smallmatrix}0&1\\0&0\end{smallmatrix}\right]$ and $t_{\mathsf{V}_{i,j}} = \left[\begin{smallmatrix}0\\0\end{smallmatrix}\right]$ *or* $\left[\begin{smallmatrix}1\\0\end{smallmatrix}\right]$;
- else $t_{\mathsf{P}_{l,t},\mathsf{P}_{l,f}} = \left[\begin{smallmatrix}0&1\\0&0\end{smallmatrix}\right]$, $t_{\mathsf{Q}_{l,t},\mathsf{Q}_{l,f}} = \left[\begin{smallmatrix}0&0\\1&0\end{smallmatrix}\right]$ and $t_{\mathsf{V}_{i,j}} = \left[\begin{smallmatrix}0\\0\end{smallmatrix}\right]$ *or* $\left[\begin{smallmatrix}0\\1\end{smallmatrix}\right]$.

To satisfy ODs 13, for $i \in \{1, \ldots, k\}$, it must be that $t_{\mathsf{C}_i} = \left[\begin{smallmatrix}0\\1\end{smallmatrix}\right]$ since $t_\mathsf{T} = \left[\begin{smallmatrix}0\\1\end{smallmatrix}\right]$.

In coNP. It is not always possible further to set values for the $\mathsf{V}_{i,j}$'s in such a way that $\mathbf{t}$ also satisfies the ODs 14, for $i \in \{1, \ldots, n\}$, $j \in \{1, \ldots, 3\}$, and so satisfies $\mathcal{M}_\mathcal{F}$ completely. When we can also set values for the $\mathsf{V}_{i,j}$'s so that $\mathbf{t}$ also satisfies the ODs 14 too, then $\mathbf{t}$ suffices as a *witness* that $\langle \mathcal{M}_\mathcal{F}, [\mathsf{T}] \sim [\mathsf{F}] \rangle \notin$ **UODI**.

Correspondence. $\mathcal{F} \in$ **3-SAT** *iff* $\langle \mathcal{M}_\mathcal{F}, [\mathsf{T}] \sim [\mathsf{F}] \rangle \notin$ **UODI**. Consider two-tuple tables $\mathbf{t}$ that satisfy the ODs 1–10 and 13 from $\mathcal{M}_\mathcal{F}$, but that falsify $[\mathsf{T}] \sim [\mathsf{F}]$. There is a one-to-one mapping between truth assigments over the p_i, for $i \in \{1, \ldots, n\}$, in $\mathcal{F}$ and settings for $\mathsf{P}_{i,t}$ in such $\mathbf{t}$. For $i \in \{1, \ldots, n\}$, if $p_i = \mathit{true}$ in the truth assignment, set

$$t_{\mathsf{P}_{i,t},\mathsf{P}_{i,f},\mathsf{Q}_{i,t},\mathsf{Q}_{i,f}} = \left[\begin{smallmatrix}0&0&0&1\\1&0&0&0\end{smallmatrix}\right];$$

else ($p_i = \mathit{false}$), set

$$t_{\mathsf{P}_{i,t},\mathsf{P}_{i,f},\mathsf{Q}_{i,t},\mathsf{Q}_{i,f}} = \left[\begin{smallmatrix}0&1&0&0\\0&0&1&0\end{smallmatrix}\right].$$

IF: There is some truth assigment over $p_1, \ldots, p_n$ that satisfies $\mathcal{F}$.

We construct a two-tuple table $\mathbf{t}$ based on this truth assignment that satisfies $\mathcal{M}_\mathcal{F}$ for ODs 1–13, and that falsifies $[\mathsf{T}] \sim [\mathsf{F}]$, as above (in the *Witness* part). For $i \in \{1, \ldots, n\}$, assign values for $\mathsf{P}_{i,t}$, $\mathsf{P}_{i,f}$, $\mathsf{Q}_{i,t}$, and $\mathsf{Q}_{i,f}$ according to the truth assignment mapping above.

To satisfy further ODs 14, we must be able to assign values to the $\mathsf{V}_{i,j}$'s that suffice. For $i \in \{1, \ldots, n\}$, $j \in \{1, \ldots, 3\}$, if $V_{i,j} = \mathit{true}$, set $t_{\mathsf{V}_{i,j}} = \left[\begin{smallmatrix}0\\1\end{smallmatrix}\right]$. This satisfies the OD 11 or 12 added to $\mathcal{M}_\mathcal{F}$ for i, given how we assigned $\mathsf{P}_{i,t}$, $\mathsf{P}_{i,f}$, $\mathsf{Q}_{i,t}$, and $\mathsf{Q}_{i,f}$ based on p_i's truth value. Otherwise ($V_{i,j} = \mathit{false}$), set $t_{\mathsf{V}_{i,j}} = \left[\begin{smallmatrix}0\\0\end{smallmatrix}\right]$. This satisfies either the OD 11 or 12 for i, j, vacuously.

Since, for each $i \in \{1, \ldots, k\}$, at least one of $V_{i,1}$, $V_{i,2}$, and $V_{i,3}$ is *true* in the truth assignment, at least one of $t_{\mathsf{V}_{i,1}}$, $t_{\mathsf{V}_{i,2}}$, or $t_{\mathsf{V}_{i,3}}$is $\left[\begin{smallmatrix}0\\1\end{smallmatrix}\right]$. Thus, $\mathbf{t}$ as constructed satisfies ODs 1−14, and so all of $\mathcal{M}_\mathcal{F}$.

ONLY IF: There is no truth assignment that satisfies $\mathcal{F}$.

For any arbitrary truth assignment, we can build a two-tuple table $\mathbf{t}$ that falsifies $[\mathsf{T}] \sim [\mathsf{F}]$ based on the truth assignment mapping that satisfies ODs 1–13, as done in the *if* part. We next try to assign values to the $\mathsf{V}_{i,j}$'s in such a way that $\mathbf{t}$ satisfies ODs 14.

Since the truth assignment does not satisfy $\mathcal{F}$, there is some clause i such that $V_{i,1}$, $V_{i,2}$, and $V_{i,3}$ are each *false*. The OD 14 for i will be falsified. For each $V_{i,j}$, as either the OD 11 or 12 is satisfied accordingly, $t_{\mathsf{V}_{i,j}} = \left[\begin{smallmatrix}0\\0\end{smallmatrix}\right]$ *or* $\left[\begin{smallmatrix}1\\0\end{smallmatrix}\right]$.

If, for any $V_{i,j}$, $t_{\mathsf{V}_{i,j}} = \left[\begin{smallmatrix}1\\0\end{smallmatrix}\right]$, OD 14 is falsified since $t_{\mathsf{C}_i} = \left[\begin{smallmatrix}0\\1\end{smallmatrix}\right]$. If instead, for all $V_{i,j}$, $t_{\mathsf{V}_{i,j}} = \left[\begin{smallmatrix}0\\0\end{smallmatrix}\right]$, OD 14 is still falsified, since $t_\mathsf{F} = \left[\begin{smallmatrix}1\\0\end{smallmatrix}\right]$.

No two-tuple table $\mathbf{t}$ that falsifies $[\mathsf{T}] \sim [\mathsf{F}]$ can be constructed that satisfies $\mathcal{M}_\mathcal{F}$. Any table $\mathbf{t}$ therefore either satisfies $[\mathsf{T}] \sim [\mathsf{F}]$ or falsifies $\mathcal{M}_\mathcal{F}$. □

THEOREM 8. *(single OD)* $\mathbf{X} \sim \mathbf{Y}$ *holds iff* $\mathbf{XY} \mapsto \mathbf{Y}$.

Proof

IF: By Reflexivity axiom, OD $\mathbf{YX} \mapsto \mathbf{Y}$ is true. Therefore, by Transitivity, $\mathbf{XY} \mapsto \mathbf{Y}$.

ONLY IF: By Suffix axiom, $\mathbf{XY} \leftrightarrow \mathbf{YXY}$ is true. Therefore, by Normalization and Transitivity, $\mathbf{XY} \sim \mathbf{YX}$. □

THEOREM 9. *Given a set $\mathcal{M}$ of UODs and UOD* $\mathbf{X} \mapsto \mathbf{Y}$, *deciding whether* $\mathcal{M} \models \mathbf{X} \mapsto \mathbf{Y}$ *is co-NP-complete.*

Proof

By Theorem 8, order compatible $\mathbf{X} \sim \mathbf{Y}$ is equivalent to a single UOD. Therefore, by Lemma 6, deciding whether $\mathcal{M} \models \mathbf{X} \mapsto \mathbf{Y}$ is co-NP-hard.

Witness. Any counter example for a given OD $\mathbf{X} \mapsto \mathbf{Y}$is a pair of tuples (that can be checked in polynomial time). This is necessary and sufficient to falsify $\mathbf{X} \mapsto \mathbf{Y}$, by the definitions of split and swap (Definitions 9 and 10).

Thus, deciding $\mathcal{M} \models \mathbf{X} \mapsto \mathbf{Y}$ is co-NP-complete. □

4.2 FD inference over ODs

COROLLARY 10. *Given a set $\mathcal{M}$ of ODs and OD* $\mathbf{X} \mapsto \mathbf{Y}$, *deciding whether* $\mathcal{M} \models \mathbf{X} \mapsto \mathbf{Y}$ *is co-NP-complete.*

Proof

Follows directly from Theorem 9 as a class of UODs is a proper sub-class of ODs. (Any *witness* that $\mathcal{M} \not\models \mathbf{X} \mapsto \mathbf{Y}$ is a pair of tuples by definitions of split and swap. □

Let the length of the representation of $\mathcal{M}$, the string of concatenated left-hand and right-hand sides of the ODs, be denoted by $|\mathcal{M}|$. FD inference for UODs is polynomial.

THEOREM 11. *(FDs over UODs) Let $\mathcal{M}$ be a set of UODs. Testing whether* $\mathcal{M} \models \mathbf{X} \mapsto \mathbf{XY}$ ($\mathcal{M} \models \mathcal{X} \rightarrow \mathcal{Y}$) *can be accomplished in O(|$\mathcal{M}$|) time. (This includes finding the closure for FDs, $\mathcal{X}^+$.)*

Proof

Assume $\mathcal{M}' = \{\mathbf{X} \mapsto \mathbf{XY}, \mathbf{XY} \leftrightarrow \mathbf{YX} \mid \mathbf{X} \mapsto \mathbf{Y} \in \mathcal{M}\}$. In [14], we have shown that $\mathcal{F} = \{\mathcal{X} \rightarrow \mathcal{Y} \mid \mathbf{X} \mapsto \mathbf{Y} \in \mathcal{M}'\}$ is a set of FDs which enables one to compute the closure for FDs $\mathcal{X}^+$ over the set of UODs $\mathcal{M}$. Inference problem of a FD $\mathcal{X} \rightarrow \mathcal{Y}$ over a set of prescribed FDs has already been shown

Table 4: Table template.

#	X_1	...	X_k	attributes($\mathcal{M}$) − {X_1,...,X_k}		
s	b_1	...	b_k	p_{k+1}	...	p_n
t	b_1	...	b_k	q_{k+1}	...	q_n

(a) Template $\mathbf{r}_0$.

#	X_1	...	X_{j-1}	X_j	attributes($\mathcal{M}$) − {X_1,...,X_j}		
s	b_1	...	b_{j-1}	b_j	p_{j+1}	...	p_n
t	b_1	...	b_{j-1}	t_j	q_{j+1}	...	q_n

(b) Template $\mathbf{r}_j$.

to be linear in [1]. This implies that testing $\mathcal{M} \models \mathcal{X} \rightarrow \mathcal{Y}$ can be also accomplished in $O(|\mathcal{M}|)$. The same applies to $\mathcal{M} \models \mathbf{X} \mapsto \mathbf{XY}$ by Theorem 6. □

This is not the same case, however, for ODs. Both the inference problems for FDs (embedded within the ODs), $\mathbf{X} \mapsto \mathbf{XY}$, and for order compatibility, $\mathbf{X} \sim \mathbf{Y}$, are *hard*.

We call an attribute a *constant* if, for any table that satisfies the set of ODs $\mathcal{M}$, it can have only a single value occurring in the table.

Definition 12. (constant) A marked attribute A is called a *constant* with respect to $\mathcal{M}$ *iff* $\mathcal{M} \models [] \mapsto \mathsf{A}$.

LEMMA 7. [17] *Given a set $\mathcal{M}$ of ODs and a FD $\{\} \rightarrow \mathsf{A}$, deciding whether $\mathcal{M} \models \{\} \rightarrow \mathsf{A}$ is co-NP-complete.*

THEOREM 12. *Given a set $\mathcal{M}$ of ODs and an FD $\mathcal{X} \rightarrow \mathcal{Y}$, deciding whether $\mathcal{M} \models \mathcal{X} \rightarrow \mathcal{Y}$ is co-NP-complete.*

Proof

By Theorem 6 and Lemma 7 deciding whether $\mathcal{M} \models \mathcal{X} \rightarrow \mathcal{Y}$ is co-NP-complete as we can always construct a *witness* (that can be checked in polynomial time) that $\mathcal{M} \not\models \mathcal{X} \rightarrow \mathcal{Y}$, by the definition of split (Definition 9). □

5. INFERENCE PROCEDURES

A goal in any dependency theory is to develop good algorithms for the inference problem. Such an inference procedure can be used in query optimization. First, we present an *elimination* procedure for the inference problem for ODs.[7] We next introduce an inference procedure for the inference problem over a *restricted domain* for UODs. The additional order property to be guaranteed over the schema is intuitive, holds for all real-world business domains that we have encountered, and can easily be verified whether it holds for a given table. We develop an polynomial inference procedure for UODs which is sound and complete when applied over a database that satisfies the property. Lastly, we present a case study for optimization by ODs in Section 5.3.

5.1 Elimination Procedure

We establish a sound and complete elimination procedure for ODs for the *inference problem*, for which the complexity is exponential. This complexity is with respect to number of unique attributes in the set of prescribed ODs over relation, not with respect to data. (Therefore, it can be used in practice as usually this is small.) We have implemented this elimination procedure in IBM DB2.

We define a *table template* over variables with respect to a given OD. We use these table templates to enumerate through all the possible cases where the OD can be falsified by splits and swaps.

Definition 13. (table template) Let $\mathcal{M}$ be a set of ODs with n unique attributes over relation $\mathbf{R}$ and m be an OD $\mathbf{X} \mapsto \mathbf{Y}$, where $\mathbf{X}$ is over attributes $\mathsf{X}_1, ..., \mathsf{X}_k$. A table template for OD m, denoted as $\mathbf{r}_m$, is a table consisting of two tuples s and t, such that it is either $\mathbf{r}_0$ (Table 4a) or $\mathbf{r}_j$ (Table 4b), for j in $[1, ..., k]$. In $\mathbf{r}_0$ and $\mathbf{r}_j$, symbols p_i and q_i represent one of the following three cases, where the *ordering* of variables b_i and t_i is defined as $b_i < t_i$:

1. $p_i = b_i$ and $q_i = b_i$;
2. $p_i = b_i$ and $q_i = t_i$; and
3. $p_i = t_i$ and $q_i = b_i$.

Example 6 presents how to apply a *mapping* (Definition 14) to a table template.

Definition 14. (mapping $\mathbf{r}_m$ to $\varphi(\mathbf{r}_m)$) Let $\mathbf{r}_m$ be a table template from Definition 13. A mapping of $\mathbf{r}_m$ to $\varphi(\mathbf{r}_m)$ is *any* instance with values that satisfy the *ordering* from Definition 13.

EXAMPLE 6. *Consider a table template* $\mathbf{t}_{\mathsf{A,B,C}} = \left[\begin{smallmatrix} b1 & b2 & t3 \\ b1 & t2 & b3 \end{smallmatrix}\right]$ *and a table* $\varphi\ (\mathbf{t}_{\mathsf{A,B,C}}) = \left[\begin{smallmatrix} 0 & 0 & 1 \\ 0 & 1 & 0 \end{smallmatrix}\right]$ *as a possible mapping.*

LEMMA 8. *Let $\mathbf{r}_m$ be a table template (Definition 13) and $\varphi(\mathbf{r}_m)$ be a mapping from $\mathbf{r}_m$ (Definition 14). Then $\mathbf{r}_m \models \mathbf{X} \mapsto \mathbf{Y}$ iff $\varphi(\mathbf{r}_m) \models \mathbf{X} \mapsto \mathbf{Y}$.*

Proof

By Definition 14, ordering of values in $\varphi(\mathbf{r}_m)$ corresponds to the ordering of variables in $\mathbf{r}_m$, respectively. □

Definition 15. (tableaux $\mathbf{T}_m$) Let m be an OD $\mathbf{X} \mapsto \mathbf{Y}$. We define $\mathbf{T}_m$ to be the set of all table templates $\mathbf{r}_m$, as we defined in Definition 13.

Note that $\mathbf{T}_m$ is a *set* of table templates, each consisting of two rows. The *elimination* of $\mathbf{T}_m$ is defined as follows.

Definition 16. (elimination of tableaux $\mathbf{T}_m$) The elimination of $\mathbf{T}_m$ over a set of order dependencies $\mathcal{M}$ denoted as $\text{ELIM}_{\mathbf{T}_m,\mathcal{M}}$ is defined by $\text{ELIM}_{\mathbf{T}_m,\mathcal{M}} = \{\mathbf{r}_m \mid \mathbf{r}_m \in \mathbf{T}_m \wedge \mathbf{r}_m \models \mathcal{M}\}$. Furthermore, $\text{ELIM}_{\mathbf{T}_m,\mathcal{M}}$ satisfies $\mathbf{X} \mapsto \mathbf{Y}$, denoted by $\text{ELIM}_{\mathbf{T}_m,\mathcal{M}} \models \mathbf{X} \mapsto \mathbf{Y}$, *iff*, for all $\mathbf{r}_m \in \text{ELIM}_{\mathbf{T}_m,\mathcal{M}}$, $\mathbf{r}_m \models \mathbf{X} \mapsto \mathbf{Y}$. $\text{ELIM}_{\mathbf{T}_m,\mathcal{M}}$ satisfies the set of ODs $\mathcal{M}'$, which is denoted as $\text{ELIM}_{\mathbf{T}_m,\mathcal{M}} \models \mathcal{M}'$, *iff*, for all $\mathbf{X} \mapsto \mathbf{Y} \in \mathcal{M}'$, $\text{ELIM}_{\mathbf{T}_m,\mathcal{M}} \models \mathbf{X} \mapsto \mathbf{Y}$.

THEOREM 13. *Let $\mathcal{M}$ be a set of ODs over $\mathbf{R}$ and m be an OD $\mathbf{X} \mapsto \mathbf{Y}$. Then $\mathcal{M} \models \mathbf{X} \mapsto \mathbf{Y}$ iff $ELIM_{\mathbf{T}_m,\mathcal{M}} \models \mathbf{X} \mapsto \mathbf{Y}$.*

Proof

IF: Assume $\text{ELIM}_{\mathbf{T}_m,\mathcal{M}} \not\models \mathbf{X} \mapsto \mathbf{Y}$. By Definition 16, there exists $\mathbf{r}_m \in \text{ELIM}_{\mathbf{T}_m,\mathcal{M}}$ such that $\mathbf{r}_m \not\models \mathbf{X} \mapsto \mathbf{Y}$. By Definition 16, $\mathbf{r}_m \models \mathcal{M}$. Hence, there is a mapping φ to generate a relation instance $\varphi(\mathbf{r}_m)$. By Lemma 8, $\varphi(\mathbf{r}_m) \models \mathcal{M}$, but in addition $\varphi(\mathbf{r}_m) \not\models \mathbf{X} \mapsto \mathbf{Y}$. We have found a relation instance which satisfies $\mathcal{M}$ but does not satisfy $\mathbf{X} \mapsto \mathbf{Y}$, which implies that $\mathcal{M} \not\models \mathbf{X} \mapsto \mathbf{Y}$.

ONLY IF: Assume $\text{ELIM}_{\mathbf{T}_m,\mathcal{M}} \models \mathbf{X} \mapsto \mathbf{Y}$. Let s and t be *any* two tuples in *any* relation $\mathbf{r}$ such that $s \preceq_{\mathbf{X}} t$ and that satisfies the set of ODs $\mathcal{M}$. We would like to present that $s \preceq_{\mathbf{Y}} t$. Let $\mathbf{r}_m \in \mathbf{T}_m$. Let $\mathbf{r}_m = \{p, q\}$ be the template relation such that $\varphi(p) = s$ and $\varphi(q) = t$. It is possible always to find such a pair of tuples $\mathbf{r}_m$ since $\mathbf{T}_m$ considers all possibilities of two tuples which satisfy condition $s \preceq_{\mathbf{X}} t$.

[7] In preliminary work [13], we focused on *fixing* the table templates with a *chase* procedure, whereas here, our technique is based on *eliminating* with an *elimination procedure* the table templates which falsify the set of ODs $\mathcal{M}$. The complexity of this elimination procedure is $O(3^n)$, where n is the number of unique attributes in the set of prescribed ODs $\mathcal{M}$ over relation. The complexity of chase procedure is $O(3^l)$, where l denotes the number of attributes in relation $\mathbf{R}$. In most real-world cases, $l \gg n$. Therefore, this revised elimination procedure is simpler and more efficient and can be used effectively in practice.

Therefore, we have $\varphi(\mathbf{r}_m) = \{s, t\}$ and $\varphi(\mathbf{r}_m) \models \mathcal{M}$. By Lemma 8, it follows that $\mathbf{r}_m \models \mathcal{M}$. It follows by Definition 16 that $\mathbf{r}_m \in \text{ELIM}_{\mathbf{T}_m,\mathcal{M}}$. Since we assumed that $\text{ELIM}_{\mathbf{T}_m,\mathcal{M}} \models \mathbf{X} \mapsto \mathbf{Y}$, we have $\mathbf{r}_m \models \mathbf{X} \mapsto \mathbf{Y}$. This implies that $\varphi(\mathbf{r}_m) \models \mathbf{X} \mapsto \mathbf{Y}$ by Lemma 8. Hence, $s \preceq_{\mathbf{Y}} t$. □

THEOREM 14. *(sound and complete) The elimination procedure for ODs is sound and complete.*

Proof

The proof follows directly from Theorem 13. □

EXAMPLE 7. *Let* $\mathcal{M} = \{[\mathsf{A}] \mapsto [\mathsf{B}], [\mathsf{B}] \mapsto [\mathsf{C}]\}$. *Let us test with elimination procedure if* $\mathcal{M} \models [\mathsf{A}] \mapsto [\mathsf{BC}]$. *The mapping instances (Definition 14) of the table templates (Definition 13) after elimination which satisfy* $\mathcal{M}$ *(Definition 16), denoted as* $r_{i\mathsf{A},\mathsf{B},\mathsf{C}} = \varphi(r_{i\mathsf{A},\mathsf{B},\mathsf{C}})$ *for* $i = 1..4$ *are:*

1) $\left[\begin{smallmatrix}0&0&0\\0&0&0\end{smallmatrix}\right]$ *2)* $\left[\begin{smallmatrix}0&0&0\\1&0&0\end{smallmatrix}\right]$ *3)* $\left[\begin{smallmatrix}0&0&0\\1&1&0\end{smallmatrix}\right]$ *4)* $\left[\begin{smallmatrix}0&0&0\\1&1&1\end{smallmatrix}\right]$

All of these mapping instances satisfy an OD $[\mathsf{A}] \mapsto [\mathsf{BC}]$. *Therefore, by Theorem 13,* $\mathcal{M} \models [\mathsf{A}] \mapsto [\mathsf{BC}]$.

THEOREM 15. *(complexity of elimination) The complexity of elimination procedure is* $\mathrm{O}(3^n)$ *(for n being the number of unique attributes in the set of ODs* $\mathcal{M}$ *over relation).*

Proof

By Definition 13, there are 3^{n-k} templates for $\mathbf{r}_0$ and 3^{n-j} templates for each $\mathbf{r}_j$. By geometric progression: $O(3^n)$. □

5.2 Inference Procedure for Transitive Domain

Our axiomatization in Section 2 yields us insight into how to make an inference procedure over a restricted domain of UODs. We find a property by which we can restrict domains to make the polynomial inference procedure, but still cover real-world domains. We observe that a relation satisfying the OD $\mathbf{X} \leftrightarrow \mathbf{Y}$ satisfies the OD $\mathbf{X} \sim \mathbf{Y}$, but not necessarily vice versa, as in the following example.

EXAMPLE 8. *The order dependency* [month] ∼ [quarter] *is satisfied in table* date_dim. *On the other hand, order dependency* [month] ↔ [quarter] *is falsified by table* date_dim.

It is surprising initially that the *order-compatibility* relation '∼' (Definition 4) is *not* transitive as shown in Example 9. (By *Transitivity* axiom the *order* relation ('↦') is.)

EXAMPLE 9. (not transitive) *Let* $\mathcal{M} = \{[\mathsf{A}] \sim [\mathsf{C}], [\mathsf{C}] \sim [\mathsf{B}]\}$. *The Table 1 satisfies the set of UODs* $\mathcal{M}$. *However, it falsifies* $[\mathsf{A}] \sim [\mathsf{B}]$. *This demonstrates the lack transitivity.*

If we restrict our domains to have a property that guarantees a limited form of transitivity over *order-compatibility* (Definition 17), then we can make an efficient inference procedure for UODs. (We have implemented this inference procedure in IBM DB2.) The property we prescribe is *transitivity of order compatibility over single attributes.*

Definition 17. (transitive domain) We call a domain (relation schema) a *transitive* domain *iff* it can be guaranteed that, for each relation $\mathbf{R}$ in the schema, for any three attributes A, B, and C where B is not a constant, if $[\mathsf{A}] \sim [\mathsf{B}]$ and $[\mathsf{B}] \sim [\mathsf{C}]$, then $[\mathsf{A}] \sim [\mathsf{C}]$.

EXAMPLE 10. (transitivity) *ODs* [quarter] ∼ [month] *and* [month] ∼ [trimester] *are satisfied Also, so is* [quarter] ∼ [trimester] *Hence, the transitivity property holds.*

All of the real-world business domains we have explored including the TPC-DS schema, IBM clients schemas, and the examples which are used in this paper are *transitive.* One can argue that breaking the underlying property in data can be only done by contrivance. Domains can be tested if they are transitive in a straightforward way, by enumeration.

We first present the key elements of the algorithm for inference problem for transitive domains of UODs and then we establish it is sound and complete in Theorem 16. The algorithm *OrderDependency* (Algorithm 3) implements an inference procedure for transitive domains of UODs. It invokes algorithms *FunctionalDependency* and *OrderCompatible* (Algorithm 4). Algorithm *FunctionalDependency* performs a test whether $\mathcal{M} \models \mathbf{X} \mapsto \mathbf{XY}$ which by Theorem 6 implies an FD, $\mathcal{M} \models \mathcal{X} \rightarrow \mathcal{Y}$. Algorithm *OrderCompatible* tests whether $\mathcal{M} \models \mathbf{XY} \leftrightarrow \mathbf{YX}$ ($\mathcal{M} \models \mathbf{X} \sim \mathbf{Y}$). These parts combine to complete the proof of soundness and completeness of our inference procedure since by Theorem 3 $\mathbf{X} \mapsto \mathbf{Y}$ holds *iff* $\mathbf{X} \mapsto \mathbf{XY}$ and $\mathbf{XY} \leftrightarrow \mathbf{YX}$.

Algorithm 3 OrderDependecy

Input: a set $\mathcal{M}$ of n unidirectional order dependencies on attributes $\{\mathsf{A}_0, ..., \mathsf{A}_{m-1}\}$ and an UOD $\mathbf{X} \mapsto \mathbf{Y}$
Output: "true" if $\mathcal{M} \models \mathbf{X} \mapsto \mathbf{Y}$; "false" otherwise
Global data structures:

a. Attributes are integers between 0 and m-1.
b. UODs in $\mathcal{M}$ are integers between 0 and n-1.
c. LS[0:$n-1$], RS[0:$n-1$] are arrays of lists, containing the attributes in the left and right side of each UOD.
d. DEPEND[0:$m-1$] is an array of attributes found to be *functionally dependent* on given set of attributes.
e. OC[0:$n-1$; 0:1] is a two dimensional array of *order compatible* dependencies with *single attribute* on the left and right side.
f. LX and LY are lists of attributes represented by integers, corresponding to $\mathbf{X}$ and $\mathbf{Y}$ respectively.

1: DEPEND ← *FunctionalDependency*($\mathcal{M}$, $\mathcal{X}$)
2: **if** exists i in LY such that DEPEND[i] = "false" **then**
3: **return** "false"
4: **else**
5: **return** *OrderCompatible*

Theorem 11 states that testing whether $\mathbf{X} \mapsto \mathbf{XY}$ ($\mathcal{X} \rightarrow \mathcal{Y}$), can be achieved in linear time. Notice that we assume there is a *linear* algorithm *FunctionalDependency* which finds a closure of a given set of attributes $\mathcal{X}$, as in [1]. Testing if $\mathbf{X} \sim \mathbf{Y}$ is more involved and complex. We observe that $\mathcal{M} \not\models \mathbf{X} \sim \mathbf{Y}$ *iff* we are able to construct a table $\mathbf{t}$ that *satisfies* set of UODs $\mathcal{M}$ and consists of two rows which have a *swap* (Definition 10) with respect to $\mathbf{X} \sim \mathbf{Y}$. In the table $\mathbf{t}$ that we construct, we shall use integer values for the *cells*, without lost of generality. We test if $\mathbf{X} \sim \mathbf{Y}$ in the algorithm *OrderCompatible*. For each pair of attributes A in $\mathbf{X}$ and B in $\mathbf{Y}$, we test in an algorithm *SingleOrderCompatible* (Algorithm 5) whether we can construct a table $\mathbf{t}$ described above with a swap with respect to $[\mathsf{A}] \sim [\mathsf{B}]$ with attributes prefixing A and B, in lists $\mathbf{X}$ and $\mathbf{Y}$, respectively, being *constants* (Definition 12) within table $\mathbf{t}$, such that table $\mathbf{t}$ satisfies the set of ODs $\mathcal{M}'$. (Let $\mathbf{P}$ be the concatenated attributes prefixing A and B. We consider $\mathcal{M}' = \mathcal{M} \cup \{[] \mapsto \mathbf{P}\}$.) $[] \mapsto \mathbf{P}$ is a way of forcing each attribute C in list $\mathbf{P}$ to be a *constant.* Once we find a swap, we halt in Algorithm 4.

Algorithm 4 OrderCompatible

Output: result stating if $\mathbf{X} \sim \mathbf{Y}$

1: **for** $i \leftarrow 0$ **to** $|\mathbf{X}| - 1$ **do**
2: **for** $j \leftarrow 0$ **to** $|\mathbf{Y}| - 1$ **do**
3: **if** !*SingleOrderCompatible*(i, j) **then**
4: **return** "false"
5: **return** "true"

Based on Definition 17, order compatibility for single attributes (over the attributes which are non-constant) is transitive for transitive domains. Therefore, we test if there is a

path between A and B in a graph consisting of the first non-constant attributes from the left-hand side and a right-hand side of each UOD from $\mathcal{M}'$. We assume we find this graph in Algorithm *FindOrderCompatibleGraph*. Finding a path by the transitivity property over order-compatibility means that [A] $\sim$ [B] holds. We assume Algorithm *ExistPath* which tests if there exists a path between two nodes. The problem of testing if there *exists* a path is simple. One can track the visited edges during the process of traversing the nodes. We can guarantee that each edge is visited only once. Hence, we can check the existence of the path in linear time. Note there is an edge per OD in $\mathcal{M}'$, so the number of edges (plus number of nodes) is $O(|\mathcal{M}|)$.

Algorithm 5 SingleOrderCompatible

Input: attributes indexes i and j
Output: resulting stating if LX[i] $\sim$ LY[j]

```
1: if LX[i] = LY[j] then
2:    return "true"
3: else
4:    M' ← M ∪ {[] ↦ P}, list P is a concatenation of lists
      LX.subList(0, i − 1) and LY.subList(0, j − 1)
5:    DEPEND ← FunctionalDependency([ ], M')
6:    if DEPEND[LX[i]] ∥ DEPEND[LY[j]] then
7:       return "true"
8:    else
9:       OC ← FindOrderCompatibleGraph
10:      return ExistPath(LX[i], LY[j], OC)
```

THEOREM 16. [17] *Algorithm 3 for inference problem* $\mathcal{M} \models \mathbf{X} \mapsto \mathbf{Y}$ *for transitive domains of UODs is sound and complete with complexity* $O(|\mathbf{X}||\mathbf{Y}||\mathcal{M}|)$.

5.3 Case Study

Most queries in a data warehouse are over fact tables. In TPC-DS, surrogate keys (sequential numbers) are used in the dimension tables, and so for the foreign keys in its fact tables. However, a query plan often uses surrogate date values in its predicates (for example, a) d_date $\geq$ cast('1999-02-22' as date) and d_date $\leq$ (cast('1999-02-22' as date) + 30 days) or b) d_year = 2000). This requires a potentially expensive *join* between the fact and the date dimension table.

In [15], we demonstrated that dramatic gains in query performance can be had in queries by recognizing ordering correspondences between attributes. The surrogate (date) keys in the date_dim dimension table order natural date values. So there is a known order dependency between them. Thus, as [d_date_sk] $\mapsto$ [d_date] (and [d_date_sk] $\mapsto$ [d_year]), two probes can be made into the dimension table to calculate the range of the surrogate keys in the fact table, finding the min_date_sk and max_date_sk surrogate keys. These minimum and maximum surrogate values then replace the predicate in the where clause with the natural date values, so no join with the date dimension table is needed.

However, in [15], we considered queries only with a binary relationship predicate in the query rewrite phase during optimization, where the relational operator in the predicate is one of $\{=, <, \leq, >, \geq\}$. Many more than 13 of the 99 queries in TPC-DS for which we benefitted with an average performance gain of 48% involve date predicates. We know that we can extend our rewrite rules to cover many more of the queries seen in the TPC-DS; for instance, to cover the case of the queries with an in predicate. In Query 3, the values of d_year in the predicate are consecutive.

QUERY 3. *With "*in*" predicate (TPC-DS Query 29).*

```
select ...
from catalog_sales, item  ..., date_dim
where ... cs_sold_date_sk = d_date_sk and
    d_year
       in (1998,1998+1, 1998+2) ...;
```

Hence, as the values are consecutive, we can select two probes and eliminate a join, since [d_date_sk] $\mapsto$ [d_year]. The rewritten query is presented below (Query 4).

QUERY 4. *Rewrite of Query 3 with consecutive values.*

```
select ...,
   (select min(d_date_sk) as min_date_sk
       from date_dim
       where d_year = 1998)
       as A,
   (select max(d_date_sk) as max_date_sk
       from date_dim
       where d_year = (1998+2))
       as Z
from catalog_sales, ..., date_dim
where cs_sold_date_sk between
   A.min_date_sk and Z.max_date_sk ...;
```

Consider a modification of Query 3 with values of the d_year being not consecutive; e.g., d_year in (1998, 1998+2). With an OD [d_date_sk] $\mapsto$ [d_year], the predicate can now be applied directly on the d_date_sk attribute by selecting two fast probes too. Deriving an additional predicate on the d_date_sk is useful even though it does not eliminate join, as it enables an efficient processing technique using *partitioning* to access qualifying tuples. For range partitioning, the predicate derivation results in partition elimination and reduction of processing overhead. (In TPC-DS, tables catalog_sales and d_date are partitioned based on the cs_sold_date_sk and d_date_sk primary keys, respectively.)

QUERY 5. *Rewrite of Query 3 with* ***not*** *consecutive values.*

```
select ...,
from catalog_sales, ..., date_dim
where ... cs_sold_date_sk = d_date_sk and
    cs_sold_date_sk between
       A.min_date_sk and Z.max_date_sk and
    d_year
       in (1998, 1998+2) ...;
```

Above, we had assumed that an OD [d_date_sk] $\mapsto$ [d_year] was declared implicitly as an integrity constraint. Assume instead we have the following ODs prescribed:

$\mathcal{M}$ = {[d_date_sk] $\mapsto$ [d_year, d_month, d_day],
[d_year, d_month, d_day] $\mapsto$ [d_date]}.

From this set of ODs, [d_date_sk] $\mapsto$ [d_date] and [d_date_sk] $\mapsto$ [d_year] can be concluded. Hence, the optimizer needs the means to discover ODs that logically follow from known ODs to benefit most from our techniques. The complexity of our *elimination procedure* is exponential. However, this complexity is with respect to the number of unique attributes in the set of prescribed ODs over relation, not with respect to data (or size of the schema). We implemented the inference procedures presented above in DB2. Our experiments have shown that the cost of running the elimination procedure is not expensive for real world business domains. Our *elimination procedure* is able to infer ODs for relations with the number of unique attributes in prescribed ODs from 5 to 12 attributes in marginal time. For instance, for the time dimension from the TPC-DS schema, the number of unique attributes in the prescribed ODs is 6 out of 10 attributes in the table and, for the date dimension from TPC-DS schema, it is 10 out of 28 attributes in the table. Our experiments have shown that the cost of running inference procedure for *transitive domains* is marginal, even for large domains.

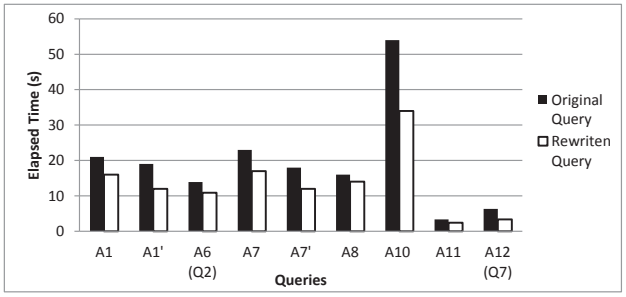

Figure 2: Performance results. (See Footnote 8.)

With Reduce Order OD, we can optimize queries such as Query 1 (or Query 2 with substr SQL function). Our inference procedures infer from the declared ODs the OD [year, month, day] ↦ [year, trimester, quarter, month, day]. Then, the order-by clause can be reduced by removing quarter and trimester. When sorting is required, the reduced version of an interesting order provides a smaller number of sorting columns, which reduces cost. It may also happen that, due to the reduced version, an index can be matched, eliminating the need for a sort operator altogether. Essentially, ODs can further exploit interesting orders which are generated during the order scan. Ordering of the data is useful for processing order-by, group-by, distinct and join. Hence, ODs can be used to eliminate or simplify potentially expensive operators (such as sorts) in the query plan.

The usefulness of ODs can be extended further by the handling of *nearly-sorted* streams. Say that we need an output stream sorted on A, B and that the input is sorted on A. If it is known all partitions of A are suitably small, each partition of A could be sorted on B in the main memory "on-the-fly". No external sort would need to be applied to achieve A, B. Current optimizers such as IBM DB2's recognize near-sortedness and apply this optimization. ODs, of course, can extend the usefulness of near-sortedness. Consider the input is known to be sorted on C and we know [C] ↦ [A]. Then the output is near sorted for A, B if the partitions of A are known to be small. Each partition of A then can be sorted on B once buffered in main memory.

As an example, consider Query 6. The inference algorithm is triggered due to the order-by statement. It detects that [d_date] ↦ [d_year]. Therefore, the optimizer can then take advantage of the index on d_date, simplifying the sort operator in the plan, to accomplish the order-by. Given the optimizer infers this OD, it can choose to do an index scan using the index on d_date to speed up evaluation of the group-by on-the-fly too.

QUERY 6. *TPC-DS Query 3.*

```
select d_year, i_brand_id brand_id, i_brand,
   sum(ss_ext_sales_price) sum_agg
from date_dim, store_sales, item
where d_date_sk = ss_sold_date_sk ...
group by d_year, i_brand, i_brand_id
order by d_year, sum_agg desc, brand_id;
```

Similarly, ODs and near-sortedness can be used when using SQL functions such as year(). In Query 7 the monotonicity detection algorithm is triggered due to the order-by and group-by statements. It detects that [d_date] ↦ [year(d_date)]. Therefore, the optimizer can then take advantage of the index on d_date, speeding up the sort operator in the plan, to accomplish the order-by and group-by.

QUERY 7. *Query with year(d_date).*

```
select year(d_date), sm_type,
     ws_web_name, ...
from web_sales, warehouse, ship_mode,
     web_site, date_dim
where ws_ship_date_sk = d_date_sk and ...
group by year(d_date), sm_type, ws_web_name
order by year(d_date), sm_type, ws_web_name;
```

Furthermore, the monotonicity property can be used to optimize queries with case expressions. There is an OD in the scope of Query 8 between d_date and the output of the case statement. When this relationship is discovered, the index on customer_id can be used, resulting in a more efficient plan. Based on our experience with IBM customers, we observe that these kind of subtleties are common in customer queries created by business-intelligence reporting tools such as Cognos which auto-generates the SQL queries.

QUERY 8. *Query with case expression.*

```
select ..., sum(quantity),
     (case
      when customer_id between 1 and 10
          then 1
               ...
      when customer_id between 91 and 100
          then 10
               ...
      end)
from sales S, ... where ...
group by (case customer_id between...)
order by (case customer_id between ...);
```

Our techniques, as in queries above eliminate or simplify expensive operations such as sort (which is super-linear) which begins to dominate the execution costs as the database size increases. Our experiments over TPC-DS schema have shown that the performance improvement by eliminating or simplifying the sort operators appearing in plans is, on average, 30% over the elapsed time [16]. Our prototype implementation in IBM DB2 V10 covers ODs between columns and functions over columns (SQL functions and algebraic expressions). The optimizer automatically infers the associated OD information and uses it to produce improved query plans. We evaluated these techniques on a ten-GB TPC-DS benchmark database and nine IBM customer inspired queries.[8] The experiments were performed on a performance testing machine with the operating system AIX 6.1 TL6 SP5 with four processors (Intel(R) Xeon(R) CPU) and 1GB of memory. For Query 7 (Query A12 from [16]) the reduction is from 6.35 to 3.34 seconds (47%) and for Query 2 (Query A6 from [16]) from 13.92 seconds to 10.88 seconds (22%).

6. RELATED WORK

Sorting is at the heart of many database operations: sort-merge join, index generation, duplicate elimination, ordering the output through the SQL order-by operator, etc. The importance of sorted sets for query optimization and processing had been recognized very early on. Right from the start, the query optimizer of System R [11] paid particular attention to *interesting orders* by keeping track of all such ordered sets throughout the process of query optimization. In [12],

[8] The performance results are conducted over a suit of nine IBM customers driven queries with generated attributes, presented in [16]. We label them with a letter "A" as prefix to distinguish the numbering of queries in this paper. Two queries from [16] are presented in this paper. (Queries 2 and 7, correspond to Queries A6 and A12, respectively.)

the authors consider how to use FD information to extend matching for interesting orders. A practical application of dependencies for improved index design was presented in [3]. In [7], the authors explored the use of sorted sets for executing nested queries. The importance of sorted sets has prompted researchers to look *beyond* the sets that have been explicitly generated. Thus, [8] showed how to use (but without performing an experimental study) sorted sets created as generated columns in predicates. We showed in [15] how to use relationships between sorted attributes discovered by reasoning over the physical schema.

In [5], a sound and complete set of inference rules for pointwise ODs is presented. The authors prove the inference problem for pointwise order dependencies is co-NP-complete. However, the problem for lexicographical ODs is just as hard (lower bound), as we prove in Section 4. Dependencies defined over lexicographically ordered domains were introduced in [9] under the name *lexicographically ordered functional dependencies*. (We called these UODs.) The same author in [10] developed a theory behind both lexicographical as well as pointwise dependencies. (The latter were simpler than the dependencies defined in [5].) Only a chase procedure was defined for the lexicographical dependencies, for which the order dependencies are defined as here (UODs). Interestingly, the axiomatization and the complexity of the inference problem for ODs had not been studied. In [14], we presented a sound and complete axiomatization for UODs. UODs do not consider bidirectionality (a mix of `asc` and `desc`) as do ODs, which we introduced in [13].

In [9], a restricted domain of UODs is presented. The authors call these *temporal functional dependencies* (TFDs). A TFD $\mathbf{X} \rightarrow \mathcal{Y}$ means that $\forall\ \mathsf{A} \in \mathcal{Y}.\ \mathbf{X} \mapsto [\mathsf{A}]$. The domain is too restricted, unfortunately, to be of use to us. It effectively restricts ODs to the form with just a single attribute on the right-hand side (e.g., $\mathbf{X} \mapsto [\mathsf{A}]$). Furthermore, no inference procedure for TFDs was defined (just an axiomatization). We took the same tactic, however, to find a property (transitive domains) by which we can restrict domains.

An interesting study of establishing whether a given stream is sufficiently nearly-sorted was described in [2]. A novel integrity constraint for ordered data, sequential dependencies (SDs), was introduced in [6]. For example, an SD $\mathsf{sequence\_id} \rightsquigarrow_{[5,6]} \mathsf{time}$ means that time *gaps* between consecutive sequence numbers are between 5 and 6. The authors present a framework for discovering which data obey SDs.

7. CONCLUSIONS

Ordering permeates databases, to such an extent that we take it for granted. We expect it to be exploited wisely in query plans. It is requested by many queries but is relatively expensive to perform. Our empirical studies show the usefulness of ODs for query optimization. To use ODs effectively in optimization requires one to reason over them. We have established the complexity of this inference problem, and presented practical inference procedures.

There remain some other problems to address. Lack of transitivity property over the order-compatibility is at the heart of the complexity (co-NP-completeness). That is why the *Chain* axiom is necessary for a complete axiomatization of UODs. We would like to investigate if there is a polynomial algorithm for reasoning over the first five axioms, excluding Chain (Figure 1). Such an inference procedure would be an alternative approach to the transitive domain.

8. ACKNOWLEDGMENTS

The authors would like to thank the reviewers for the valuable comments and suggestions that helped us to improve our paper. Thanks also go to Wenbin Ma and Weinan Qiu for helping us generating TPC-DS benchmark results.

9. REFERENCES

[1] C. Beeri and P. Bernstein. Computional Problems Related to the Design of Normal Form Relational Schemas. *TODS 4(1):30-59*, 1979.

[2] S. Ben-Moshe, Y. Kanza, E. Fischer, A. Matsliah, M. Fischer, and C. Staelin. Detecting and Exploiting Near-Sortedness for Efficient Relational Query Qvaluation. In *ICDT, 256-267*, 2011.

[3] J. Dong and R. Hull. Applying Approximate Order Dependency to Reduce Indexing Space. In *SIGMOD, 119-127*, 1982.

[4] M. Garey and D. Johnson. Computers and Intractability: A Guide to the Theory of NP-completness. In *W.H. Freeman, 45-60*, 1979.

[5] S. Ginsburg and R. Hull. Order Dependency in the Relational Model. *TCS, 26(1): 149-195*, 1983.

[6] L. Golab, H. Karloff, F. Korn, and D. Srivastava. Sequential Dependencies. *PVLDB, 2(1): 574-585*, 2009.

[7] R. Guravannavar, H. Ramanujam, and S. Sudarshan. Optimizing Nested Queries with Parameter Sort Orders. In *VLDB, 481-492*, 2005.

[8] T. Malkemus, P. S., B. Bhattacharjee, and L. Cranston. Predicate Derivation and Monotonicity Detection in DB2 UDB. In *ICDE, 939-947*, 2005.

[9] W. Ng. Lexicographically Ordered Functional Dependencies and Their Application to Temporal Relations. In *IDEAS, 279-287*, 1999.

[10] W. Ng. An Extension of the Relational Data Model to Incorporate Ordered Domains. *TODS, 26(3) 344-383*, 2001.

[11] P. Selinger and M. Astrahan. Access Path Selection in a Relational Database Management System. In *SIGMOD, 23-34*, 1979.

[12] D. Simmen, E. Shekita, and T. Malkemus. Fundamental Techniques for Order Optimization. In *SIGMOD, 57-67*, 1996.

[13] J. Szlichta, P. Godfrey, and J. Gryz. Chasing Polarized Order Dependencies. In *AMW, 168-179*, 2012.

[14] J. Szlichta, P. Godfrey, and J. Gryz. Fundamentals of Order Dependencies. *PVLDB, 5(11): 1220-1231*, 2012.

[15] J. Szlichta, P. Godfrey, J. Gryz, W. Ma, P. Pawluk, and C. Zuzarte. Queries on Dates: Fast Yet not Blind. In *EDBT, 497-502*, 2011.

[16] J. Szlichta, P. Godfrey, J. Gryz, W. Ma, W. Qiu, and C. Zuzarte. Business-Intelligence Queries in DB2 with Order Dependencies. Technical report, York University, 2012. www.cse.yorku.ca/techreports/2012.

[17] J. Szlichta, P. Godfrey, J. Gryz, and C. Zuzarte. The Complexity of Order Dependency Inference. Technical report, York University, 2012. www.cse.yorku.ca/techreports/2012.

[18] J. Szlichta, P. Godfrey, J. Gryz, and C. Zuzarte. Axiomatic System for Order Dependencies. In *AMW*, 2013.