Bringing Order to Data

Heidar Davoudi[1], Parke Godfrey[2], Lukasz Golab[1], Mehdi Kargar[3], Divesh Srivastava[4], and Jaroslaw Szlichta[5]

[1]University of Waterloo, Canada, [2] York University, Canada,
[3] Ted Rogers School of Management, Ryerson University, Canada,
[4] AT&T Labs-Research, USA, [5] University of Ontario Institute of Technology, Canada,
hdavoudi@uwaterloo.ca, godfrey@yorku.ca, lgolab@uwaterloo.ca, kargar@ryerson.ca, divesh@research.att.com,szlichta@uoit.ca

Abstract. Integrity constraints (ICs) are widely used in business intelligence to express and enforce application semantics. However, finding ICs manually is time consuming, requires the involvement of domain experts, and is prone to human error. Thus, techniques have been proposed to automatically find a variety of ICs. We propose an algorithm to automatically discover order dependencies (ODs). Prior work on OD discovery has factorial complexity, is not complete, and is not concise. We propose an algorithm that finds a complete set of ODs with exponential worst-case time complexity in the number of attributes and linear complexity in the number of tuples. We experimentally show that our algorithm is orders of magnitude faster than the prior state-of-the-art.

Keywords: Integrity constraints · Order Dependency · Axiomatization.

1 Introduction

Ordered attributes, such as timestamps and numbers, are common in business data. Order Dependencies (ODs) capture monotonic relationships among such attributes. For instance, Table 1 shows employee tax records in which `tax` is calculated as a `percentage` (`perc`) of `salary` (`sal`). The OD `sal` *orders* `perc` holds: if we sort the table by salary, it is also sorted by percentage. Similarly, `sal` `orders` `grp`, `subg`: if we sort the table by salary, it is also sorted by group with ties broken by subgroup. With interest in data analytics at an all-time high, ODs can improve the consistency dimension of data quality and query optimization [4, 7–9]. ODs can describe business rules (data profiling); and their violations can point out possible data errors. Furthermore, query optimizers can use ODs to eliminate costly operators such as joins and sorts: ordered streams between query operators can exploit available indexes, enable pipelining, and eliminate intermediate partitioning steps. Finally, ODs subsume Function Dependencies (FDs) as any FD can be *mapped* to an equivalent OD by prefixing the left-hand-side attributes onto the right-hand-side [6].

It is time consuming to specify integrity constraints manually, motivating the need for their automatic discovery. However, since ODs are naturally expressed with lists rather than sets of attributes (as in the example above), existing solutions have *factorial* worst-case time complexity in the number of attributes [4]. We describe a more efficient algorithm to discover ODs from data. First, we show that ODs can be expressed with sets of attributes via a *polynomial mapping* into an *equivalent* set-based canonical form. Then, we introduce *sound* and *complete* axioms for set-based canonical ODs,

#	ID	yr	pos	bin	sal	perc	tax	grp	subg
t1	1	16	sec	1	5K	20%	1K	A	III
t2	2	16	dev	2	8K	25%	2K	C	II
t3	3	16	dir	3	10K	30%	3K	D	I
t4	1	15	sec	1	4K	20%	0.8K	A	III
t5	2	15	dev	2	6K	25%	1.5K	C	I
t6	3	15	dir	3	8K	25%	2K	C	II

Table 1: Employee salary data

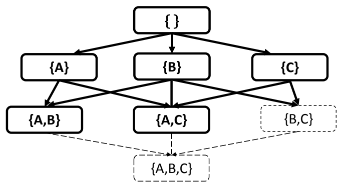

Fig. 1: A set lattice for attributes A, B, C.

which lead to optimizations of the OD discovery algorithm by avoiding redundant computation. This allows us to design a *fast* and *effective* OD discovery algorithm that has *exponential* worst-case complexity, $O(2^{|\mathbf{R}|})$, in the number of attributes $|\mathbf{R}|$, and linear complexity in the number of tuples. We note that this short paper is a summary of our published results in [5, 6, 10].

2 Order Dependency Discovery

2.1 Preliminaries

Order dependencies (ODs) describe relationships among lexicographical orderings of sets of tuples, as in the SQL *order by* statement. Let $\mathbf{X} = [\mathsf{A} \mid \mathbf{T}]$ be a list of attributes, where the attribute A is the *head* of the list, and the list $\mathbf{T}$ is the tail. For two tuples s and t, we write $s \preceq_{\mathbf{X}} t$ *iff* $s_{\mathsf{A}} < t_{\mathsf{A}}$ *or* ($s_{\mathsf{A}} = t_{\mathsf{A}}$ and ($\mathbf{T} = [\,]$ or $s \preceq_{\mathbf{T}} t$)). Given two lists of attributes, $\mathbf{X}$ and $\mathbf{Y}$, $\mathbf{X} \mapsto \mathbf{Y}$ denotes an OD, read as $\mathbf{X}$ *orders* $\mathbf{Y}$. Table $\mathbf{r}$ *satisfies* $\mathbf{X} \mapsto \mathbf{Y}$ *iff*, for all $s, t \in \mathbf{r}$, $s \preceq_{\mathbf{X}} t$ implies $s \preceq_{\mathbf{Y}} t$. Moreover, $\mathbf{X}$ and $\mathbf{Y}$ are *order compatible*, denoted as $\mathbf{X} \sim \mathbf{Y}$ *iff* $\mathbf{XY} \leftrightarrow \mathbf{YX}$. (For example, `month` and `week` of the year in the calendar are order compatible.)

We say that two tuples, s and t, are *equivalent* with respect to an attribute set $\mathcal{X}$ if $s_{\mathcal{X}} = t_{\mathcal{X}}$. Any attribute set $\mathcal{X}$ partitions tuples into *equivalence classes* [3]. We denote the *equivalence class* of a tuple $t \in \mathbf{r}$ with respect to a given set $\mathcal{X}$ by $\mathcal{E}(t_{\mathcal{X}})$; i.e., $\mathcal{E}(t_{\mathcal{X}}) = \{s \in \mathbf{r} \mid s_{\mathcal{X}} = t_{\mathcal{X}}\}$. A *partition* of $\mathbf{r}$ over $\mathcal{X}$ is the set of equivalence classes, $\Pi_{\mathcal{X}} = \{\mathcal{E}(t_{\mathcal{X}}) \mid t \in \mathbf{r}\}$. For instance, in Table 1, $\mathcal{E}(t1_{\{\text{year}\}}) = \mathcal{E}(t2_{\{\text{year}\}}) = \mathcal{E}(t3_{\{\text{year}\}}) = \{t1, t2, t3\}$ and $\Pi_{\text{year}} = \{\{t1, t2, t3\}, \{t4, t5, t6\}\}$.

2.2 Canonical Mapping and Axioms

Expressing ODs in a natural way relies on lists of attributes; e.g., in Table 1, `sal` $\mapsto$ `grp`, `subg` is not the same as `sal` $\mapsto$ `subg`, `grp`. In contrast, the order of attributes in an FD does not matter. However, the list representation leads to high complexity when discovering ODs [4]. Thus, we provide a polynomial *mapping* of list-based ODs into *equivalent* set-based canonical ODs [5, 6, 10]. The mapping allows us to develop an efficient OD discovery algorithm that traverses a much smaller set-containment lattice rather than the list-containment lattice used in [4].

The mapping presented in Theorem 1 (below) converts a list-based OD into *canonical* set-based ODs of two types. First, an attribute A is a *constant* within each equivalence class with respect to a set of attributes $\mathcal{X}$, denoted as $\mathcal{X}$: $[\,] \mapsto \mathsf{A}$, if $\mathbf{X}' \mapsto \mathbf{X}'\mathsf{A}$ for any permutation $\mathbf{X}'$ of $\mathcal{X}$ (note that $\mathcal{X}$ functionally determines $\mathcal{Y}$ *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 $\mathcal{Y}$ [6]). Second, two attributes, A and B, are order-compatible within each equivalence class with respect to the set of attributes $\mathcal{X}$, denoted as $\mathcal{X}$: $\mathsf{A} \sim \mathsf{B}$, if $\mathbf{X}'\mathsf{A} \sim \mathbf{X}'\mathsf{B}$. The set $\mathcal{X}$ is called a *context*. For example, in Table 1, `bin` is a constant in the context of `pos`, written as $\{$`pos`$\}$:

1. Reflexivity
$\mathcal{X}$: $[\,] \mapsto \mathsf{A}, \forall \mathsf{A} \in \mathcal{X}$

2. Identity
$\mathcal{X}$: $\mathsf{A} \sim \mathsf{A}$

3. Commutativity
$$\frac{\mathcal{X}\text{: } \mathsf{A} \sim \mathsf{B}}{\mathcal{X}\text{: } \mathsf{B} \sim \mathsf{A}}$$

4. Strengthen
$$\frac{\begin{array}{l}\mathcal{X}\text{:}[\,] \mapsto \mathsf{A}\\ \mathcal{X}\mathsf{A}\text{:}[\,] \mapsto \mathsf{B}\end{array}}{\mathcal{X}\text{:}[\,] \mapsto \mathsf{B}}$$

5. Propagate
$$\frac{\mathcal{X}\text{:}[\,] \mapsto \mathsf{A}}{\mathcal{X}\text{: } \mathsf{A} \sim \mathsf{B}}$$

6. Augmentation-I
$$\frac{\mathcal{X}\text{:}[\,] \mapsto \mathsf{A}}{\mathcal{Z}\mathcal{X}\text{:}[\,] \mapsto \mathsf{A}}$$

7. Augmentation-II
$$\frac{\mathcal{X}\text{: } \mathsf{A} \sim \mathsf{B}}{\mathcal{Z}\mathcal{X}\text{: } \mathsf{A} \sim \mathsf{B}}$$

8. Chain
$$\frac{\begin{array}{l}\mathcal{X}\text{: } \mathsf{A} \sim \mathsf{B}_1\\ \forall_{i\in[1,n-1]}, \mathcal{X}\text{: } \mathsf{B}_i \sim \mathsf{B}_{i+1}\\ \mathcal{X}\text{: } \mathsf{B}_n \sim \mathsf{C}\\ \forall_{i\in[1,n]}, \mathcal{X}\mathsf{B}_i : \mathsf{A} \sim \mathsf{C}\end{array}}{\mathcal{X}\text{: } \mathsf{A} \sim \mathsf{C}}$$

Fig. 2: Set-based axiomatization for canonical ODs.

$[\,] \mapsto$ `bin` since $\mathcal{E}(t1_{\{\texttt{pos}\}}) \models [\,] \mapsto$ `bin`, $\mathcal{E}(t2_{\{\texttt{pos}\}}) \models [\,] \mapsto$ `bin` and $\mathcal{E}(t3_{\{\texttt{pos}\}}) \models [\,] \mapsto$ `bin`. Also, $\{\texttt{year}\}$: `bin` $\sim$ `salary` because $\mathcal{E}(t1_{\{\texttt{year}\}}) \models$ `bin` $\sim$ `salary` and $\mathcal{E}(t4_{\{\texttt{year}\}}) \models$ `bin` $\sim$ `salary`.

Theorem 1. *$\boldsymbol{X} \mapsto \boldsymbol{Y}$ iff $\forall j$, $\mathcal{X}$:$[\,] \mapsto Y_j$ and $\forall i, j$, $\{X_1, .., X_{i-1}, Y_1, .., Y_{j-1}\}$: $X_i \sim Y_j$.*

Example 1. The OD $[\mathsf{AB}] \mapsto [\mathsf{CD}]$ is mapped into the following set of canonical ODs: $\{\mathsf{A}, \mathsf{B}\}$:$[\,] \mapsto \mathsf{C}$, $\{\mathsf{A}, \mathsf{B}\}$:$[\,] \mapsto \mathsf{D}$, $\{\}$: $\mathsf{A} \sim \mathsf{C}$, $\{\mathsf{A}\}$: $\mathsf{B} \sim \mathsf{C}$, $\{\mathsf{C}\}$: $\mathsf{A} \sim \mathsf{D}$, $\{\mathsf{A}, \mathsf{C}\}$: $\mathsf{B} \sim \mathsf{D}$.

We present a *sound* and *complete set-based axiomatization* for ODs in Fig. 2 [6]. The set-based axioms allow us to design effective pruning rules for our OD discovery algorithm. For example, OD $\mathcal{X}$:$[\,] \mapsto \mathsf{A}$ is *trivial* if $\mathsf{A} \in \mathcal{X}$ by Reflexivity (see also *Example 2*).

Theorem 2. *The axiomatization for canonical ODs in Fig. 2 is sound and complete.*

2.3 Discovery Algorithm

Given the mapping of a list-based OD into equivalent set-based ODs, we present an algorithm, named *FASTOD*, that *efficiently* discovers a complete and minimal set of set-based ODs over a given relation instance. In contrast, the OD discovery algorithm from [4] traverses a lattice of all possible *lists* of attributes, which leads to factorial time complexity. *FASTOD* starts the search from singleton sets of attributes and works its way to larger attribute sets through a set-containment lattice (as in Figure 1), level by level ($l = 0, 1, \ldots$). When the algorithm is processing an attribute set $\mathcal{X}$, it verifies ODs of the form $\mathcal{X} \setminus \mathsf{A}$:$[\,] \mapsto \mathsf{A}$ (let $\mathcal{X} \setminus \mathsf{A}$ be shorthand for $\mathcal{X} \setminus \{\mathsf{A}\}$, where $\mathsf{A} \in \mathcal{X}$) and $\mathcal{X} \setminus \{\mathsf{A}, \mathsf{B}\}$: $\mathsf{A} \sim \mathsf{B}$, where $\mathsf{A}, \mathsf{B} \in \mathcal{X}$ and $\mathsf{A} \neq \mathsf{B}$. Furthermore, an OD $\mathcal{X} \setminus \mathsf{A}$:$[\,] \mapsto \mathsf{A}$ should be minimal, that is, $\nexists\, \mathcal{Y} \subset \mathcal{X}$ such that $\mathcal{Y} \setminus \mathsf{A}$:$[\,] \mapsto \mathsf{A}$ is valid.

The algorithm maintains information about minimal ODs, in the form of $\mathcal{X} \setminus \mathsf{A}$:$[\,] \mapsto \mathsf{A}$, in the *candidate* set $\mathcal{C}_c^+(\mathcal{X})$ [3] (as ODs subsume FDs [7, 9]), where $\mathcal{C}_c^+(\mathcal{X}) = \{\mathsf{A} \in \mathbf{R} \mid \forall_{\mathsf{B}\in\mathcal{X}}\ \mathcal{X} \setminus \{\mathsf{A}, \mathsf{B}\}$:$[\,] \mapsto \mathsf{B}$ does not hold$\}$. Similarly, it stores information about minimal ODs in the form of $\mathcal{X} \setminus \{\mathsf{A}, \mathsf{B}\}$: $\mathsf{A} \sim \mathsf{B}$, in the *candidate* set $\mathcal{C}_s^+(\mathcal{X})$, where $\mathcal{C}_s^+(\mathcal{X}) = \{\{\mathsf{A}, \mathsf{B}\} \in \mathcal{X}^2 \mid \mathsf{A} \neq \mathsf{B}$ and $\forall_{\mathsf{C}\in\mathcal{X}}\ \mathcal{X} \setminus \{\mathsf{A}, \mathsf{B}, \mathsf{C}\}$: $\mathsf{A} \sim \mathsf{B}$ does not hold, and $\forall_{\mathsf{C}\in\mathcal{X}}\ \mathcal{X} \setminus \{\mathsf{A}, \mathsf{B}, \mathsf{C}\}$:$[\,] \mapsto \mathsf{C}$ does not hold$\}$. The following lemmas can be used to prune the search space:

Lemma 1. *An OD $\mathcal{X} \setminus A$:$[\,] \mapsto A$, where $A \in \mathcal{X}$, is minimal iff $\forall_{B\in\mathcal{X}} A \in \mathcal{C}_c^+(\mathcal{X} \setminus B)$.*

Lemma 2. *An OD $\mathcal{X} \setminus \{A, B\}$: $A \sim B$, where A, $B \in \mathcal{X}$ and $A \neq B$, is minimal iff $\forall_{C\in\mathcal{X}\setminus\{A,B\}}$ $\{A, B\} \in \mathcal{C}_s^+(\mathcal{X} \setminus C)$, and $A \in \mathcal{C}_c^+(\mathcal{X} \setminus B)$ and $B \in \mathcal{C}_c^+(\mathcal{X} \setminus A)$.*

According to Lemma 1, we do not need to check $\mathcal{X} \setminus \mathsf{A}$:[] $\mapsto$ A if $\mathsf{A} \notin \mathcal{X} \bigcap_{\mathsf{B} \in \mathcal{X}} \mathcal{C}_c^+(\mathcal{X} \setminus \mathsf{B})$, and based on Lemma 2, we do not need to consider $\mathcal{X} \setminus \{\mathsf{A}, \mathsf{B}\}$: $\mathsf{A} \sim \mathsf{B}$ if $\mathsf{A} \notin \mathcal{C}_c^+(\mathcal{X} \setminus \mathsf{B})$ or $\mathsf{B} \notin \mathcal{C}_c^+(\mathcal{X} \setminus \mathsf{A})$. Moreover, according to Lemma 3, we can delete nodes from the lattice under the following conditions:

Lemma 3. *Deleting node $\mathcal{X}$ from the l^{th} lattice level, where $l \geq 2$, has no effect on the output set of minimal ODs if $\mathcal{C}_c^+(\mathcal{X}) = \{\}$ and $\mathcal{C}_s^+(\mathcal{X}) = \{\}$.*

Example 2. Let A:[] $\mapsto$ B, B:[] $\mapsto$ A and $\{\}$: $\mathsf{A} \sim \mathsf{B}$. Since $\mathcal{C}_c^+(\{\mathsf{A}, \mathsf{B}\}) = \{\}$ and $\mathcal{C}_s^+(\{\mathsf{A}, \mathsf{B}\})$ as well as $l = 2$, by the pruning levels rule (Lemma 3), the node $\{\mathsf{A}, \mathsf{B}\}$ is deleted and the node $\{\mathsf{A}, \mathsf{B}, \mathsf{C}\}$ is not considered *(*see Figure 1*)*. This is justified as $\{\mathsf{AB}\}$:[] $\mapsto$ C is not minimal by the Strengthen axiom, $\{\mathsf{AC}\}$:[] $\mapsto$ B is not minimal by Augmentation–I, $\{\mathsf{BC}\}$:[] $\mapsto$ A is not minimal by Augmentation–I, $\{\mathsf{C}\}$: $\mathsf{A} \sim \mathsf{B}$ is not minimal by Augmentation–II, $\{\mathsf{A}\}$: $\mathsf{B} \sim \mathsf{C}$ is not minimal by Propagate, and $\{\mathsf{B}\}$: $\mathsf{A} \sim \mathsf{C}$ is not minimal by Propagate.

Note that while we provide theoretical guarantees for *FASTOD* to find a complete set of ODs, the *ORDER* algorithm [4] is not complete.

Theorem 3. *The FASTOD algorithm computes a complete and minimal set of ODs.*

In [10], we extend our algorithm to discover *bidirectional* ODs, which allow a mix of ascending and descending (desc) orders. For example, a student with an alphabetically lower letter grade has a higher percentage grade than another student. We develop additional pruning rules and show that efficiency of discovery of bidirectional ODs remains the same as for one-directional ODs.

We experimentally compare *FASTOD* with previous approaches (*ORDER* [4]). Our algorithm can be orders of magnitude faster. For instance, on the `flight` dataset from `http://metanome.de` with 1K tuples and 20 attributes, *FASTOD* finishes the computation in less than 1 second, whereas ORDER did not terminate after 5 hours. Moreover, *FASTOD*'s candidate sets do not increase in size during the execution of the algorithm (unlike *ORDER*) because of the concise candidate representation (e.g., many ODs that are considered minimal by ORDER are found to be redundant by our algorithm).

Finally, in [2], we show that a recent approach to OD discovery called *OCDDISCOVER* in Consonni et al. [1] is incorrect. We show that their claim of completeness of OD discovery is *not* true. Built upon their incorrect claim, *OCDDISCOVER*'s pruning rules are overly aggressive, and prune parts of the search space that contain legitimate ODs. This is the reason their approach appears to be "faster" in practice than our *FASTOD* discovery algorithm despite being significantly worse in asymptotic complexity. Finally, we show that Consonni et al. [1] misinterpret our set-based canonical form for ODs, leading to an incorrect claim that our FASTOD implementation has an error.

3 Conclusions

We presented an efficient algorithm for discovering ODs. The technical innovation that made our algorithm possible is a novel mapping into a set-based canonical form and an axiomatization for set-based canonical ODs. In future work, we plan to study *conditional* ODs that hold over portions of data.

References

1. Consonni, C., Sottovia, P., Montresor, A., Velegrakis, Y.: Discovering Order Dependencies through Order Compatibility. In: EDBT. pp. 409–420 (2019)
2. Godfrey, P., Golab, L., Kargar, M., Srivastava, D., Szlichta, J.: Errata note: Discovering order dependencies through order compatibility. *Technical Report, 5 pages, available at* https://arxiv.org/abs/1905.02010 (2019)
3. Huhtala, Y., Karkkainen, J., Porkka, P., Toivonen, H.: Efficient Discovery of Functional and Approximate Dependencies Using Partitions. In: ICDE. pp. 392–401 (1998)
4. Langer, P., Naumann, F.: Efficient order dependency detection. VLDB J. **25**(2), 223–241 (2016)
5. Mihaylov, A., Godfrey, P., Golab, L., Kargar, M., Srivastava, D., Szlichta, J.: FASTOD: bringing order to data. In: 2018 IEEE 34th International Conference on Data Engineering (ICDE). pp. 1561–1564. IEEE (2018)
6. Szlichta, J., Godfrey, P., Golab, L., Kargar, M., Srivastava, D.: Effective and Complete Discovery of Order Dependencies via Set-based Axiomatization. PVLDB **10**(7), 721–732 (2017)
7. Szlichta, J., Godfrey, P., Gryz, J.: Fundamentals of Order Dependencies. PVLDB, 5(11): 1220-1231 (2012)
8. Szlichta, J., Godfrey, P., Gryz, J., Ma, W., Qiu, W., Zuzarte, C.: Business-Intelligence Queries with Order Dependencies in DB2. In: EDBT, 750-761 (2014)
9. Szlichta, J., Godfrey, P., Gryz, J., Zuzarte, C.: Expressiveness and Complexity of Order Dependencies. PVLDB 6(14): 1858-1869 (2013)
10. Szlichta, J., Godfrey, P., Golab, L., Kargar, M., Srivastava, D.: Effective and complete discovery of bidirectional order dependencies via set-based axioms. The VLDB Journal **27**(4), 573–591 (2018)