Discovering Domain Orders through Order Dependencies

by

MohammadReza Karegar

A thesis
presented to the University of Waterloo
in fulfillment of the
thesis requirement for the degree of
Master of Mathematics
in
Computer Science

Waterloo, Ontario, Canada, 2021

Author's Declaration

I hereby declare that I am the sole author of this thesis. This is a true copy of the thesis, including any required final revisions, as accepted by my examiners.

I understand that my thesis may be made electronically available to the public.

Abstract

Most real-world data come with explicitly defined domain orders; e.g., lexicographic order for strings, numeric for integers, and chronological for time. Our goal is to discover implicit domain orders that we do not already know; for instance, that the order of months in the Chinese Lunar calendar is *Corner* $\prec$ *Apricot* $\prec$ *Peach*. To do so, we enhance data profiling methods by discovering implicit domain orders in data through order dependencies. We enumerate tractable special cases and proceed towards the most general case, which we prove is NP-complete. We then consider discovering *approximate* implicit orders; i.e., those that exist with some exceptions. We propose definitions of approximate implicit orders and prove that all non-trivial cases are NP-complete. We show that the NP-complete cases nevertheless can be effectively handled by a SAT solver. We also devise an interestingness measure to rank the discovered implicit domain orders. Based on an extensive suite of experiments with real-world data, we establish the efficacy of our algorithms, and the utility of the domain orders discovered by demonstrating significant added value in two applications (data profiling and data mining).

Acknowledgements

I thank my supervisor, Professor Lukasz Golab, for his support and guidance during my studies. I also thank Jarek Szlichta, Parke Godfrey, Mehdi Kargar, and Divesh Srivastava for all their insightful ideas that helped shape this thesis the way it did. Finally, I thank Professors Grant Weddell and Xi He for serving as readers for this thesis.

Dedication

I dedicate this work to my family and friends, who have blessed me with their continuous love and support.

Table of Contents

List of Figures

List of Tables

Chapter 1

Introduction

Most real-world data come with explicitly defined orders; e.g., lexicographic for strings, numeric for integers and floats, and chronological for time. Our goal is to go a step further to discover potential domain orders that are not already known. We call these *implicit orders*. Consider the first 10 tuples in Table 1.1, [1] describing festivals in various countries. The timestamp column has an explicit chronological order. Given this explicit order, we show how to discover the *implicit* order of months in the Gregorian calendar, monthGreg (*January* $\prec$ *February* $\prec$ *March*, etc). Moreover, we will show how to find implicit orders based on other implicit orders. For instance, given the implicit order on monthGreg, we can find the implicit order of months in the traditional Chinese (Lunar) calendar, monthLun (*Corner* $\prec$ *Apricot* $\prec$ *Peach* $\prec$ *Plum* $\prec$ *Pomegranate* $\prec$ *Lotus* $\prec$ *Orchid* $\prec$ *Osmanthus* $\prec$ *Chrysanthemum* $\prec$ *Dew* $\prec$ *Winter* $\prec$ *Ice*). Finally, we discover *approximate* implicit orders that hold with some exceptions in the data. For instance, given the explicit order over count, we can discover the implicit order over size (*Small* $\prec$ *Medium* $\prec$ *Large* $\prec$ *X-Large*) over all 11 tuples in Table 1.1, even though one pair of tuples (t_2 and t_{11}) is an exception to this order.

Domain orders are useful in a number of important applications:

- Implicit orders can enhance data profiling methods to identify new data quality rules, such as order dependencies over implicitly ordered attributes. (See Section 8.3, Exp-7 and Exp-8.)
- Implicit orders can improve the performance of machine learning techniques by turning categorical features into ordinal ones. One case of this is the splitting condition in decision trees. Similarly, implicit orders can produce concise data summaries, with

[1] Unless mentioned otherwise, ignore tuple t_{11} in this table.

Table 1.1: A dataset with festival information in various countries.

	festival	country	timestamp	week	yearGreg	monthGreg	yearLun	monthLun	count	size	ribbon	profit	tax
t_1	*New Year Eve Shanghai*	China	20200125	4	2020	January	4718	Corner	10M	X-Large	Blue	30M	2.7M
t_2	*Tomb Sweeping Day Xi*	China	20200404	14	2020	April	4718	Peach	750K	Medium	Red	1.3M	117K
t_3	*Buddha B day Liuzhou*	China	20200430	18	2020	April	4718	Plum	450K	Medium	White	900K	81K
t_4	*Dragon Boat Hangzhou*	China	20200625	26	2020	June	4718	Pomegranate	2M	X-Large	Red	3M	270K
t_5	*Dongzhi Festival*	China	20201221	52	2020	December	4718	Winter	950K	Large	Red	3.5M	315K
t_6	*New Year Quebec*	Canada	20210101	1	2021	January	4718	Winter	800K	Large	Red	3M	390K
t_7	*TD Toronto Jazz*	Canada	20210618	25	2021	June	4719	Pomegranate	500K	Medium	Blue	1.5M	195K
t_8	*Rogers Tennis Cup*	Canada	20210807	32	2021	August	4719	Lotus	600K	Medium	Red	1.2M	156K
t_9	*Steam Era Ontario*	Canada	20210830	36	2021	August	4719	Osmanthus	125K	Small	Blue	1M	130K
t_{10}	*Octoberfest Waterloo*	Canada	20211009	41	2021	October	4719	Chrys.	50K	Small	White	150K	19.5K
t_{11}	*Christmas Eve Toronto*	Canada	20211224	-	-	-	-	-	700K	Large	Blue	-	-

ranges over ordered attributes instead of individual categories. We demonstrate this by enhancing a recent data summarization method [32] with newly discovered implicit orders. (See Section 8.3, Exp-9.)

1.1 Methodology

Manual specification does not scale as it requires domain experts [13, 10, 17, 24]. This motivates the need to discover implicit orders automatically. *Integrity constraints* (ICs) specify relationships between attributes in databases [1]. To discover implicit orders, we use *order dependencies* (ODs), a class of ICs, which capture relationships between orders [9, 21]. Intuitively, an order dependency asserts that sorting a table according to some attribute(s) implies that the table is also sorted according to some other attribute(s). For instance, in Table 1.1, timestamp orders yearGreg. If the tax per country is a fixed or a progressive percentage of the profit, then sorting the table by country, profit results in the table also being sorted by country, tax. Hence, "country, profit *orders* country, tax." *Functional dependencies* (FDs) are another class of ICs. An FD asserts that the the values of one set of attributes uniquely determine the values of another set of attributes. For example, count *functionally determines* size. The order of attributes on the left- and right-hand sides in an OD matters, as in the SQL *order-by* clause, while the order of attributes in a functional dependency (FD) [3] does not, as in the SQL *group-by* clause.

An OD implies the corresponding FD, modulo lists and sets of attributes but, not vice versa; e.g., country, profit orders country, tax implies that country, profit functionally determines country, tax. *Order compatibility* (OC) [27] is a weaker version of an OD, without the implied FD. Two lists of attributes in a table are said to be *order compatible*

if there exists an arrangement for the tuples in the database in which the tuples are sorted according to both of the lists of attributes. For instance, yearGreg, monthNum *is order compatible* with yearGreg, week, where the attribute monthNum (not included in Table 1.1) denotes the Gregorian month of the year in numeric format and week represents the week of the year. A corresponding FD does *not* hold: yearGreg, monthNum does not functionally determine yearGreg, week (there are multiple weeks in a month) and yearGreg, week does *not* functionally determine yearGreg, monthNum (a week may span two months).

When an OD or OC has a common prefix on its left- and right-side, we can "factor out" the common prefix to increase understandability and refer to it as the *context*. Intuitively, this means that the respective OD or OC holds *within* each partition group of data by the context. For instance, if country, profit orders country, tax, then given a partitioning of the data by country (i.e., the context), profit orders tax within each *group* (that is, for any given *country*). When an OD or OC has no common prefix, we say it has an *empty* context; e.g., the OD of timestamp orders yearGreg has no common prefix, and thus an empty context.

Algorithms for OD and OC discovery from data [17, 24, 25, 31] use *explicit* domain orders. Let us say that they discover *explicit-to-explicit* (E/E) ODs. We discover implicit orders by extending the machinery of OD discovery. We first leverage explicitly known domain orders, where, say, the left-hand-side of a "candidate" OC is an explicit domain order and the right-hand side is a learned, implicit domain order. Call this an E/I OC. For instance, in the context of yearGreg, timestamp is order compatible with monthGreg*, where the star denotes implicit domain order over an attribute. Astonishingly, implicit domain orders can be also discovered from a "candidate" OC for *both* the left- and right-hand sides of the OC! Call this an I/I OC. E/I ODs and I/I ODs simply are E/I OCs and I/I OCs, respectively, for which there is also an FD from the lefthand side to the right. For example, in the context of yearGreg and yearLun, monthGreg* is order compatible with monthLun*.

Due to errors and anomalies, which are prevalent in real-world data, many dependencies do not hold perfectly in the data, even though they are semantically valid. *Approximate* dependencies are introduced to combat this issue, allowing us to discover more general rules and avoid overfitting. Therefore, in this work, we also consider approximate implicit OCs and ODs, which can be valid even if some *exceptions* exist in the data. For instance, over tuples $t_1 - t_{11}$ in Table 1.1, count is *approximately* order compatible with size*, despite the exception between tuples t_2 and t_{11}.

1.2 Overview and Contributions

Our goal is to **discover implicit domain orders**. To do this, we define candidate classes for E/I OCs and I/I OCs, and we extend the discovery methods for these. To the best of our knowledge, this is the first attempt to discover implicit domain orders through ICs. The problem space can be factored by the following dimensions.

- Whether there is a corresponding FD (thus, finding ODs rather than OCs).
- Whether the context is *empty*.
- When the context is non-empty, whether the discovered domain orders across different partition groups with respect to the context are to be considered as independent of each other, and so can be different (*conditional*), or they are to be considered the same across partition groups, and must be consistent (*unconditional*). In Table 1.1, the implicit order monthGreg* is unconditional; however, the implicit order ribbon* is conditional in the relative context of the country with respect to the size of the festival, with *White* $\prec$ *Blue* $\prec$ *Red* in Canada and *White* $\prec$ *Red* $\prec$ *Blue* in China.[2]
- Whether we are considering E/I OCs or I/I OCs.
- Whether the candidate holds *exactly* or *approximately*, i.e., with some exceptions.

Our key contributions are as follows.

1. **Implicit Domain Orders** (*Chapter 2*).
   We formulate a novel data profiling problem of discovering implicit domain orders through a significant broadening of OD / OC discovery, and we parameterize the problem space.

We divide the problem space between *explicit-to-implicit* and *implicit-to-implicit*, which we present in Chapters 3 and 4, respectively. We identify tractable cases, and then proceed towards the general case of I/I OC discovery, which we prove is NP-complete. In Chapter 5, we introduce approximate implicit order compatibilities and prove that all nontrivial cases are NP-complete.

2. **E/I Discovery** (*Chapter 3*).
   For implicit domain order discovery through E/I ODs and E/I OCs, we present efficient algorithms, taking polynomial time in the number of tuples to verify a given OD or OC candidate.
3. **I/I Discovery** (*Chapter 4*).
   For implicit domain order discovery through I/I OCs,

[2]This example is inspired by equine competitions, for which the colors of awarded ribbons differ by country in their meanings of *first*, *second*, *third* place and so forth (https://en.wikipedia.org/wiki/Horse_show#Awards).

(a) we present a polynomial candidate verification algorithm when the context is empty,
(b) we present a polynomial candidate verification algorithm when the context is non-empty but taken as *conditional*,
(c) we show why the candidate set of conditional I/I ODs is always empty, although it is not necessarily empty for unconditional I/I ODs, *and*
(d) we prove that, for non-empty contexts taken as *unconditional*, the problem is NP-complete.

4. **Approximate Implicit Orders** (*Chapter 5*).
For *approximate* implicit domain order discovery,
(a) we present a formal definition of approximate implicit orders for E/I OCs as well as I/I OCs,
(b) we prove, through two different reductions, that except for the trivial case of conditional approximate I/I ODs, even the simplest forms of approximate implicit order discovery, i.e., conditional E/I ODs and I/I OCs, are NP-complete.
5. **Using a SAT Solver and Interestingness** (*Chapters 6 & 7*).
(a) While the general case of implicit order discovery through I/I OCs and the nontrivial approximate cases are NP-hard, we show that these problems can be effectively handled by a SAT solver (Chapter 6). We implement our methods in a lattice-based framework that has been used to mine FDs and ODs from data [25, 17, 24].
(b) We propose an *interestingness measure* to rank the discovered orders and simplify manual validation (Chapter 7).
6. **Experiments** (*Chapter 8*).
We mirror the sub-classes and approaches to discover implicit domain orders defined in Chapters 3, 4, and 5 and the algorithms in Chapters 6 and 7 via experiments over real-world datasets.
(a) **Scalability:** In Section 8.1, we demonstrate scalability in the number of tuples and attributes, and the effectiveness of our method for handling NP-complete instances.
(b) **Effectiveness:** In Section 8.2, we validate the utility of the discovered orders.
(c) **Applications:** In Section 8.3, we demonstrate the usefulness of implicit orders in data profiling (by finding more than double the number of data quality rules by involving orders not found in existing knowledge bases) and data summarization (by increasing the information contained in the summaries by an average of 60%).

We review related work in Chapter 9 and conclude in Chapter 10. Proofs of all theorems can be found in Appendix A.

This work builds on a recent thesis on the discovery of *semantic* order compatibilities [20], where here, the term *implicit* is used instead of *semantic*. While the focus of [20] is on the discovery of order compatibilities over attributes with implicit orders, in this work,

we shift the focus to the discovery of implicit orders themselves. In this work, we extend the ideas presented in [20] by dividing the search space into more fine-grained categories, formalizing the existing ideas and algorithms, and finally, presenting new definitions and solutions for the new cases. The following ideas and algorithms have been first proposed in [20] and are only reformulated here with a focus on implicit orders:

1. **E/I Discovery.**
   the definition of a *valid* E/I OC (Section 3.1) and the algorithms for discovering E/I OCs with an empty or non-empty context (Sections 3.3 and 3.4, respectively); and
2. **I/I Discovery.**
   the definition of valid I/I OCs (Section 4.1), the algorithm for discovering I/I OCs with an empty context (Section 4.2), and the definition of "chain polarity problem" (Section 4.4).

The rest of this work (including the definitions of *strongest derivable orders* in Sections 3.1 and 4.1) presents new ideas not explored in [20].

Chapter 2

Preliminaries

2.1 Domain Orders and Partitions

We first review definitions of order, as introduced in [12]. A glossary of relevant notation is provided in Table 2.1. If we can write down a sequence of a domain's values to represent how they are ordered, this defines a *strong* total order over the values. For two distinct *dates*, for example, one always precedes the other in time.

Definition 1. (strong total order) *Given a domain of values $\mathcal{D}$, a* strong total order *is a relation "$\prec$" which, $\forall x, y, z \in \mathcal{D}$, is*

- transitive. *if $x \prec y$ and $y \prec z$, then $x \prec z$,*
- connex. *$x \prec y$ or $y \prec x$ and*
- irreflexive *$x \not\prec x$.*

One may name a strong total order over $\mathcal{D}$ as $\mathsf{T}^{\prec}_{\mathcal{D}}$. Then $x \prec y \in \mathsf{T}^{\prec}_{\mathcal{D}}$ asks whether x precedes y in the order. We may refer to strong *total orders simply as total orders, to reduce verbosity.*

We might know how *groups* of values are ordered, but not how the values *within* each group are ordered. This is a *weak* total order.

Definition 2. (weak total order) *Given a domain of values $\mathcal{D}$, a* weak total order *is a relation "$\prec$" defined over $\mathcal{D}$, $\mathsf{W}^{\prec}_{\mathcal{D}}$,* iff *there is a partition over $\mathcal{D}$'s values, $\boldsymbol{\mathcal{D}} = \{\mathcal{D}_1, \ldots, \mathcal{D}_k\}$ and a strong total order $\mathsf{T}^{\prec}_{\boldsymbol{\mathcal{D}}}$ over $\boldsymbol{\mathcal{D}}$ such that*

$$\mathsf{W}^{\prec}_{\mathcal{D}} = \{\mathsf{a} \prec \mathsf{b} \mid \mathsf{a} \in \mathcal{D}_\mathsf{i} \wedge \mathsf{b} \in \mathcal{D}_\mathsf{j} \wedge \mathcal{D}_\mathsf{i} \prec \mathcal{D}_\mathsf{j} \in \mathsf{T}^{\prec}_{\boldsymbol{\mathcal{D}}}\}.$$

Table 2.1: Notation

Notation	Description
$\mathcal{D}$, $\boldsymbol{\mathcal{D}}$	Domain of ordered values, a partition
$\mathsf{T}^{\prec}_{\mathcal{D}}$, $\mathsf{W}^{\prec}_{\mathcal{D}}$, $\mathsf{P}^{\prec}_{\mathcal{D}}$	Strong total, weak total, and strong partial order
$\mathbf{R}$, $\mathbf{r}$, A	Relational schema, table instance, and one attribute
$\mathcal{X}$, $\mathcal{XY}$, $\{\}$	Set of attributes, set union, and the empty set
$\mathbf{X}$, $\mathbf{X}'$	List of attributes and arbitrary permutation
$\mathbf{XY}$, $[]$	List concatenation and empty list
t, $\mathsf{t}_{\mathcal{X}}$, $\mathsf{t}_{\mathbf{X}}$	Tuple and projections over attributes (cast to set)
$\mathcal{E}(\mathsf{t}_{\mathcal{X}})$, $\pi_{\mathcal{X}}$, $\tau_{\mathbf{X}}$	Partition group, partition, and sorted partition
$\mathcal{X}\colon \mathsf{A} \sim \mathsf{B}$	Canonical OC
$\mathcal{X}\colon [] \mapsto \mathsf{A}$	Canonical OFD
$\mathsf{T}^{\prec}_{\mathsf{A}^*}$, A^*	Derived and strongest derivable orders over A
$\mathsf{BG}_{\mathsf{A},\mathsf{B}}$, $\mathsf{BG}'_{\mathsf{A},\mathsf{B}}$	Initial and simplified bipartite graphs over A and B

A strong partial order defines order precedence for some pairs of items in the domain, but not all.

Definition 3. (strong partial order) *Given a domain of values $\mathcal{D}$, a* strong partial order *is a relation "$\prec$" which $\forall x, y, z \in \mathcal{D}$ is*

- antisymmetric. *if $x \prec y$, then $y \not\prec x$,*
- transitive. *if $x \prec y$ and $y \prec z$, then $x \prec y$, and*
- irreflexive. *$x \not\prec x$.*

One may name a strong partial order over $\mathcal{D}$ as $\mathsf{P}^{\prec}_{\mathcal{D}}$. Then $x \prec y \in \mathsf{P}^{\prec}_{\mathcal{D}}$ asks whether x precedes y in the order.

For any strong partial order $\mathsf{P}^{\prec}_{\mathcal{D}}$, there exists a strong total order $\mathsf{T}^{\prec}_{\mathcal{D}}$ such that $\mathsf{P}^{\prec}_{\mathcal{D}} \subseteq \mathsf{T}^{\prec}_{\mathcal{D}}$. A *nested* order (*lexicographic* ordering) with respect to a *list* of attributes $\mathbf{X}$ corresponds to the semantics of SQL's *order by*, shown as $\mathsf{s} \preceq_{\mathbf{X}} \mathsf{t}$ or $\mathsf{t} \preceq_{\mathbf{X}} \mathsf{s}$ between tuples s and t. An attribute set can define a *partition* of a table's tuples into *groups*, as by SQL's *group by*.

Definition 4. (partition) *The* partition group *of a tuple $\mathsf{t} \in \mathbf{r}$ over an attribute set $\mathcal{X} \subset \mathbf{R}$ is defined as $\mathcal{E}(\mathsf{t}_{\mathcal{X}}) = \{\mathsf{s} \in \mathbf{r} \,|\, \mathsf{s}_{\mathcal{X}} = \mathsf{t}_{\mathcal{X}}\}$.*

The partition of $\mathbf{r}$ over $\mathcal{X}$ *is the* set *of partition groups $\pi_{\mathcal{X}} = \{\mathcal{E}(\mathsf{t}_{\mathcal{X}}) \,|\, \mathsf{t} \in \mathbf{r}\}$. The* sorted partition $\tau_{\mathbf{X}}$ of $\mathbf{r}$ over $\mathbf{X}$ *is the* list *of partition groups from $\pi_{\mathcal{X}}$* sorted *with respect to*

$\mathbf{X}$ *and the* domain orders *of the attributes in* $\mathbf{X}$*; e.g., partition groups in* $\pi_{\{\mathsf{A},\mathsf{B},\mathsf{C}\}}$ *are sorted in* $\tau_{[\mathsf{A},\mathsf{B},\mathsf{C}]}$ *as per SQL's "order by* A, B, C*"[13, 24].*

Example 1. *In Table 1.1,* $\mathcal{E}(\mathsf{t}_{1\,\mathsf{yearGreg}}) = \{\mathsf{t}_1, \ldots, \mathsf{t}_5\}$, $\pi_{\mathsf{yearGreg}} = \{\{\mathsf{t}_1, \ldots, \mathsf{t}_5\}, \{\mathsf{t}_6, \ldots, \mathsf{t}_{10}\}\}$, *and* $\tau_{\mathsf{yearGreg}} = [\{\mathsf{t}_1, \ldots, \mathsf{t}_5\}, \{\mathsf{t}_6, \ldots, \mathsf{t}_{10}\}]$.

2.2 Order Dependencies and Order Compatibilities

List-based notation. A natural way to describe an order dependency is via two *lists* of attributes. An order dependency $\mathbf{X}$ *orders* $\mathbf{Y}$ means that $\mathbf{Y}$'s values are lexicographically, monotonically non-decreasing with respect to $\mathbf{X}$'s values [17, 22, 24, 25, 27].

Definition 5. (order dependency) *Let* $\mathcal{X}, \mathcal{Y} \subseteq \mathbf{R}$. $\mathbf{X} \mapsto \mathbf{Y}$ *denotes an* order dependency *(*OD*), read as* $\mathbf{X}$ orders $\mathbf{Y}$. $\mathbf{X} \mapsto \mathbf{Y}$ *(*$\mathbf{r} \models \mathbf{X} \mapsto \mathbf{Y}$*) iff, for all* $\mathsf{r}, \mathsf{s} \in \mathbf{r}$, $\mathsf{r} \preceq_{\mathbf{X}} \mathsf{s}$ *implies* $\mathsf{r} \preceq_{\mathbf{Y}} \mathsf{s}$. $\mathbf{X}$ *and* $\mathbf{Y}$ *are* order equivalent*, denoted as* $\mathbf{X} \leftrightarrow \mathbf{Y}$*, iff* $\mathbf{X} \mapsto \mathbf{Y}$ *and* $\mathbf{Y} \mapsto \mathbf{X}$.

Example 2. *The following* ODs *hold in Table 1.1:*
$[\mathsf{timestamp}] \mapsto [\mathsf{year}]$ *and* $[\mathsf{country}, \mathsf{profit}] \mapsto [\mathsf{country}, \mathsf{tax}]$.

Definition 6. (order compatibility) *Two lists* $\mathbf{X}$ *and* $\mathbf{Y}$ *are* order compatible *(*OC*), denoted as* $\mathbf{X} \sim \mathbf{Y}$*, iff* $\mathbf{XY} \leftrightarrow \mathbf{YX}$.

Example 3. *Assume that we add an attribute* monthNum *as a numeric version of Gregorian month in Table 1.1. Then, the* OC $[\mathsf{yearGreg}, \mathsf{monthNum}] \sim [\mathsf{yearGreg}, \mathsf{week}]$ *is valid with respect to the table as sorting by year, month and breaking ties by week is equivalent to sorting by year, week and breaking ties by month.*

There is a strong relationship between ODs and FDs. Any OD implies an FD, modulo lists and sets, but not vice versa [27, 30]. If $\mathbf{R} \models \mathbf{X} \mapsto \mathbf{Y}$, then $\mathbf{R} \models \mathcal{X} \rightarrow \mathcal{Y}$ (FD). There also exists a correspondence between FDs and ODs [27, 30]. $\mathbf{R} \models \mathcal{X} \rightarrow \mathcal{Y}$ (FD*) iff* $\mathbf{R} \models \mathbf{X}' \mapsto \mathbf{X}'\mathbf{Y}'$.

ODs can be violated in two ways [27, 30]. $\mathbf{R} \models \mathbf{X} \mapsto \mathbf{Y}$ iff $\mathbf{R} \models \mathbf{X} \mapsto \mathbf{XY}$ *(*FD*)* and $\mathbf{R} \models \mathbf{X} \sim \mathbf{Y}$ (OC). This offers two sources of OD violations, called *splits* and *swaps*, respectively [24, 27].

Definition 7. (split) *A* split *with respect to the* OD *of* $\mathbf{X} \mapsto \mathbf{XY}$ *(which represents the* FD *of* $\mathcal{X} \rightarrow \mathcal{Y}$*) and table* $\mathbf{r}$ *is a pair of tuples* $\mathsf{s}, \mathsf{t} \in \mathbf{r}$ *such that* $\mathsf{s}_{\mathcal{X}} = \mathsf{t}_{\mathcal{X}}$ *but* $\mathsf{s}_{\mathcal{Y}} \neq \mathsf{t}_{\mathcal{Y}}$.

Definition 8. (swap) *A* swap *with respect to the* OC *of* $\mathbf{X} \sim \mathbf{Y}$ *and table* $\mathbf{r}$ *is a pair of tuples* $\mathsf{s}, \mathsf{t} \in \mathbf{r}$ *such that* $\mathsf{s} \prec_{\mathbf{X}} \mathsf{t}$ *but* $\mathsf{t} \prec_{\mathbf{Y}} \mathsf{s}$.

Example 4. *In Table 1.1, there is a split for the* OD *of* [yearGreg] $\mapsto$ [yearGreg, timestamp] (*an* FD) *with tuples* t_1 *and* t_2*, and a swap for the* OC *of* [count] $\sim$ [profit] *with tuples* t_7 *and* t_8*, invalidating these candidates.*

Set-based notation (and mapping). Expressing ODs in a natural way relies on lists of attributes, as in SQL order-by. However, lists are *not* inherently necessary to express ODs as we can express them in a set-based *canonical* form. The set-based form enables more efficient OD discovery, and there exists a polynomial *mapping* of list-based ODs into *equivalent* set-based canonical ODs [24, 25].

Definition 9. (canonical form) *The* FD *that states that attribute* A *is* constant *within each partition group over the set of attributes* $\mathcal{X}$ *can be written as* $\mathcal{X}\colon [] \mapsto \mathsf{A}$*. This is equivalent to the* OD $\mathbf{X}' \mapsto \mathbf{X}'\mathsf{A}$ *in list notation. Call this an* order *functional dependency (*OFD*). The* canonical OC *that states that* A *and* B *are order compatible within each partition group over the set of attributes* $\mathcal{X}$ *is denoted as* $\mathcal{X}\colon \mathsf{A} \sim \mathsf{B}$*. This is equivalent to the* OC $\mathbf{X}'\mathsf{A} \sim \mathbf{X}'\mathsf{B}$.

The set $\mathcal{X}$ *in this notation is called the* OFD*'s or* OC*'s* context*.* OFD*s and canonical* OC*s constitute the canonical* OD*s, which we express using the notation* $\mathcal{X}\colon \mathsf{A} \mapsto \mathsf{B}$.

We are interested in ODs of the form $\mathbf{X}'\mathsf{A} \mapsto \mathbf{X}'\mathsf{B}$ as written in the canonical form as $\mathcal{X}\colon \mathsf{A} \mapsto \mathsf{B}$. To discover such ODs, we limit the search to find canonical OCs and OFDs. This generalizes: $\mathbf{X} \mapsto \mathbf{Y}$ *iff* $\mathbf{X} \mapsto \mathbf{XY}$ *and* $\mathbf{X} \sim \mathbf{Y}$. These can be encoded into an equivalent set of OCs and OFDs [24, 25]. In the context of $\mathcal{X}$, all attributes in $\mathcal{Y}$ are constants. In the context of all prefixes of $\mathbf{X}$ and of $\mathbf{Y}$, the trailing attributes are order compatible. Thus, we can encode $\mathbf{X} \mapsto \mathbf{Y}$ based on the following polynomial mapping.

$$\mathbf{R} \models \mathbf{X} \mapsto \mathbf{XY} \textit{ iff } \forall \mathsf{A} \in \mathbf{Y}.\ \mathbf{R} \models \mathcal{X}\colon [] \mapsto \mathsf{A} \text{ and}$$

$$\mathbf{R} \models \mathbf{X} \sim \mathbf{Y} \textit{ iff } \forall i, j.\ \mathbf{R} \models [\mathsf{X}_1, \ldots, \mathsf{X}_{i-1}][\mathsf{Y}_1, \ldots, \mathsf{Y}_{j-1}]\colon \mathsf{X}_i \sim \mathsf{Y}_j.$$

This establishes a *mapping* of list-based ODs into *equivalent* set-based canonical ODs; i.e., the OD $\mathbf{X}'\mathsf{A} \mapsto \mathbf{X}'\mathsf{B}$ is logically equivalent to the pair of the OC $\mathcal{X}\colon \mathsf{A} \sim \mathsf{B}$ and OFD $\mathcal{X}\mathsf{A}\colon [] \mapsto \mathsf{B}$. This is because $\mathbf{X}'$, which is a common prefix for both the left and right side of this OD, can be factored out, making $\mathcal{X}\colon \mathsf{A} \sim \mathsf{B}$ the only non-trivial OC that needs to hold. Thus, OD $\equiv$ OC + OFD.

Example 5. *In Table 1.1,* {country, profit}: [] $\mapsto$ tax (OFD) *and* {country}: profit $\sim$ tax (OC). *Hence,* {country}: profit $\mapsto$ tax (OD)*, as tax rates vary in different countries.*

Problem Statement. We want to *find implicit domain orders* over a dataset through exact and approximate E/I OCs and I/I OCs of the form $\mathcal{X}$: A $\sim$ B* and $\mathcal{X}$: A* $\sim$ B*, respectively.

2.3 Discovering ODs and OCs

Since manual specification of dependencies does not scale, being able to discover them automatically from data becomes important. Two main categories of dependency discovery algorithms are known as *schema-driven* and *data-driven* approaches. In schema-driven discovery algorithms, a lattice of all possible sets of attributes is traversed in a level-wise manner (i.e., from smaller to larger sets) and for every set of attributes in the lattice, potential dependency candidates derived from it are validated [13, 24]. Since the time complexity of these algorithms largely depends on the number of attributes (as it affects the lattice size), these algorithms are suitable for large datasets with fewer attributes. Data-driven algorithms, on the other hand, consider pairs of tuples and the dependencies that each pair violates. Dependencies that are not violated by any such pair of tuples must therefore be valid [33]. Since the runtime of these algorithms is mainly affected by the number of tuples, they are generally more suitable for smaller datasets with many attributes. In this work, we extend the schema-driven approach proposed in [24] and implement our algorithm on top of their discovery framework.

The algorithm from [24] starts with single attributes and proceeds to larger attribute sets through the set-containment lattice, level by level, with the ith level containing sets of i attributes. Since single attributes are sorted in the first level of the lattice, larger attribute sets can be easily sorted afterward. When processing an attribute set $\mathcal{X}$, the algorithm verifies (O)FDs of the form $\mathcal{X} \setminus$ A: [] $\mapsto$ A for which A $\in \mathcal{X}$, and OCs of the form $\mathcal{X} \setminus$ {A, B}: A $\sim$ B for which A, B $\in \mathcal{X}$ and A $\neq$ B. This guarantees that only non-trivial ODs are considered. (Trivial ODs are those which must always hold; e.g., A $\mapsto$ A). For instance, Figure 2.1 illustrates the lattice space over the attributes country, profit, and tax from Table 1.1. On the second level of the lattice and when considering the attribute set {country, profit}, we discover the (O)FD profit $\rightarrow$ country. On the third level and with respect to the attribute set {country, profit, tax}, we discover the OC {country}: profit $\sim$ tax. After discovering ODs on each level, inference rules (axioms) are used to prune the search space by removing redundant ODs which follow from others.

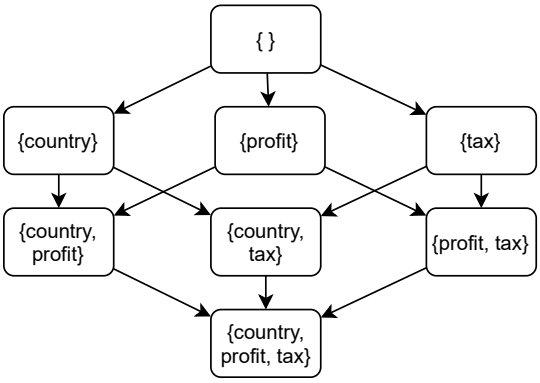


Figure 2.1: Lattice space over three attributes.

That the canonical ODs described in Section 2.2 have a set-based representation rather than list-based means that the search lattice for them is set-based, not list-based, which is significantly smaller. This leads to an OD discovery algorithm that traverses the much smaller set-containment lattice of candidate dependencies [24, 25], rather than a list-containment lattice [17]. This OD-discovery algorithm has exponential worst-time complexity in the number of attributes (to generate the candidate ODs), but linear complexity in the number of tuples (to verify each OD candidate). In practice, this is orders of magnitude faster than the list-based discovery framework in [17] with factorial worst-case time complexity in the number of attributes.

We now describe the framework of our discovery algorithm (iORDER), illustrated in Figure 2.2. First, potential OC candidates are generated for one level of the lattice. These candidates are then pruned using the dependencies found in the previous levels of the lattice and the OC axioms. Next, the existence of an FD is checked for each candidate, and depending on whether an FD holds or not, different types of implicit OCs are examined using the algorithms described in Chapters 3 through 6. Next, valid candidates and the strongest implicit OC types that were validated are stored (e.g., unconditional E/I OCs are preferred over conditional E/I OCs and unconditional I/I OCs). The candidates for the next level of the lattice are then generated, until the search for candidates is finished. In the final step, the discovered implicit orders are ranked based on their interestingness scores (Chapter 7) and can then be manually validated by experts.

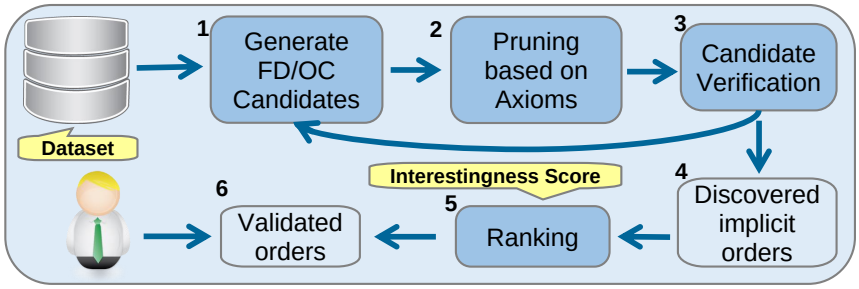


Figure 2.2: System framework.

Chapter 3

E/I Discovery

We begin with explicit-to-implicit (E/I) domain order discovery through ODs and OCs in which an attribute with an explicit order is used to find an implicit domain order over another attribute.

- We first define an *implicit* domain order with respect to the table and an explicit domain order on another attribute (Section 3.1).
- We then subdivide the problem of domain-order discovery via E/I OCs and ODs as follows:
  - with FDs, thus effectively via ODs (Section 3.2);
  - without corresponding FDs, thus effectively via OCs
    - with *empty* contexts (Section 3.3) and
    - with *non-empty* contexts (Section 3.4).

Thus, in Section 3.1, we define when two attributes can be *co-ordered*, given an explicit order on one, and define what the *strongest* derived order is. We then provide *algorithms* to determine when $\mathcal{X}\colon \mathsf{A} \sim \mathsf{B}^*$, and to compute the *strongest* order B^* when it does.

3.1 Implicit Domain Orders

For *explicit-explicit* OC discovery, say, for columns A and B, it suffices to check that the tuples of $\mathbf{r}$ can be ordered in *some* way that is consistent *both* with ordering the tuples of $\mathbf{r}$ with respect to column A's explicit domain order *and* with respect to column B's explicit domain order. That way of ordering the tuples of $\mathbf{r}$ is a *witness* that A and B can be "co-ordered"; it *justifies* that $\mathsf{A} \sim \mathsf{B}$.

To define *explicit-implicit order compatibility*, we want to maintain this same concept: there is a way to order the tuples of $\mathbf{r}$ with respect to column A's explicit domain order *and* for which the projection on B provides a *valid* order over B's values. For $\mathsf{E/I\ OC}$s with a non-empty context, $\mathcal{X}\colon \mathsf{A} \sim \mathsf{B}^*$, there must be a witness total order over $\mathbf{r}$ that is, *within* each partition group of $\mathcal{X}$, compatible with the explicit order of A *and* the order over B dictated by this is valid. This definition is compatible with the one proposed in [20]. Based on the definition from [20], an explicit-implicit OC $\mathcal{X}\colon \mathsf{A} \sim \mathsf{B}^*$ is valid if there exists a total order over the values of B such that within each partition group, *no* swaps exist with respect to the explicit order over the values of A and the new total order over the values of B. This answers one of our two questions: whether the candidate OC of $\mathsf{A} \sim \mathsf{B}^*$ *holds* over $\mathbf{r}$. The second question in this case, which is not answered in [20], is, what is *that* order B^*?

Such a witness order over $\mathbf{r}$ derives a total order (perhaps weak) over B. There may be more than one witness order over $\mathbf{r}$. Consider the OC $\mathsf{monthNum} \sim \mathsf{monthLun}^*$ over the first five tuples in Table 1.1. While the ordering $[\mathsf{t}_1, \mathsf{t}_2, \mathsf{t}_3, \mathsf{t}_4, \mathsf{t}_5]$ is a valid witness that gives the order $Corner \prec Peach \prec Plum \prec Pomegranate \prec Winter$ over $\mathsf{monthLun}$, so is the ordering $[\mathsf{t}_1, \mathsf{t}_3, \mathsf{t}_2, \mathsf{t}_4, \mathsf{t}_5]$, where the order of month values $Peach$ and $Plum$ is swapped. This indicates that we can only derive a *weak* total order $Corner \prec \{Peach, Plum\} \prec Pomegranate \prec Winter$.

If $\mathcal{X}\colon \mathsf{A} \sim \mathsf{B}^*$ holds over $\mathbf{r}$, what then is the "strongest" B^*? We define it as the intersection of all the derived strong total orders over B corresponding to the possible witness total orders of $\mathbf{r}$ that justify $\mathcal{X}\colon \mathsf{A} \sim \mathsf{B}^*$. This is a "model-theoretic" definition. Of course, it would be impractical to find B^* this way. We will present an algorithm, a "proof-theoretic" definition, for discovering B^*.

Definition 10. *(*strongest derived order via an $\mathsf{E/I\ OC}$*) The* $\mathsf{E/I\ OC}$ *of* $\mathcal{X}\colon \mathsf{A} \sim \mathsf{B}^*$ holds *over* $\mathbf{r}$ iff *there exists a* witness *strong total order* $\mathsf{T}_{\mathbf{r}}^{\prec}$ *such that, for* $\mathsf{s}, \mathsf{t} \in \mathbf{r}$*, if* $\mathsf{s} \prec \mathsf{t} \in \mathsf{T}_{\mathbf{r}}^{\prec}$ *and* $\mathsf{s}_{\mathcal{X}} = \mathsf{t}_{\mathcal{X}}$*, then* $\mathsf{s}_{\mathsf{A}} \prec \mathsf{t}_{\mathsf{A}} \in \mathsf{T}_{\mathsf{A}}^{\prec}$*, and the derived "order" relation over* B*,*

$$\mathsf{T}_{\mathsf{B}^*}^{\prec} = \{\mathsf{s}_{\mathsf{B}} \prec \mathsf{t}_{\mathsf{B}} \,|\, \mathsf{s} \prec \mathsf{t} \in \mathsf{T}_{\mathbf{r}}^{\prec} \wedge \mathsf{s}_{\mathsf{B}} \neq \mathsf{t}_{\mathsf{B}} \wedge \mathsf{s}_{\mathcal{X}} = \mathsf{t}_{\mathcal{X}}\},$$

is a strong partial order.

Let $\mathcal{X}\colon \mathsf{A} \sim \mathsf{B}^*$ *hold over* $\mathbf{r}$*, and* $\mathbb{B}$ *be the* collection *of derived order relations over* B *with respect to the witness orders over* $\mathbf{r}$*. The* strongest derivable *(strong partial)* order *over* B *is defined as*

$$\mathsf{B}^* = \bigcap_{\mathsf{T}_{\mathsf{B}}^{\prec} \in \mathbb{B}} \mathsf{T}_{\mathsf{B}}^{\prec}.$$

Example 6. *The* E/I OC *of* yearLun $\sim$ monthLun* *holds over the first five tuples in Table 1.1. There is a single value for* yearLun*. That there exists a total order over the tuples with no cycle over the values of* monthLun *("*B*") means then that there is a valid witness. However, since* all *such total orders do not cause cycles, we have no information regarding the order between month values. Thus,* monthLun$^* = \{\}$*, the* empty *partial order.*

3.2 E/I ODs, Empty Context

We first consider E/I ODs with an empty context; i.e., we are looking to establish whether there is a B* with respect to A over the whole table $\mathbf{r}$ when there is a functional dependency from one side to the other of a candidate. When we have an explicit order over one side, we might discover an implicit order over the other side by finding an E/I OD between them.

Let our pair of attributes be A and B, assume we have an explicit order over A, and we want to discover an implicit order over B (i.e., B*). We have three cases for FDs between the pair: (1) A $\rightarrow$ B and B $\rightarrow$ A; (2) A $\rightarrow$ B but B $\not\rightarrow$ A; or (3) B $\rightarrow$ A but A $\not\rightarrow$ B. We devise efficient algorithms for each case.

The first case above is trivial. There exists exactly *one* implicit order over B, which is a strong total order. To discover this order over B, sort the table over A, and project out B. (If A is not a key of the table and may have duplicates, then B would too; eliminate these duplicates, which must be adjacent.) This is B*. This is unique with respect to A and is a strong total order.

Example 7. *Let the attribute* monthNum *be added to Table 1.1 to denote the Gregorian month of the year in the numeric format. Then the* FD*s* monthNum $\rightarrow$ monthGreg *and* monthGreg $\rightarrow$ monthNum *hold. Thus, the* E/I OC *of* monthNum $\sim$ monthGreg* *is valid with the implied domain order* $\mathsf{P}^{\prec}_{\mathsf{monthGreg}^*}$ *of* Jan $\prec$ Apr $\prec$ Jun $\prec$ Aug $\prec$ Oct $\prec$ Dec.

For the second case, since B $\not\rightarrow$ A, this means some B values are associated with more than one A value. We can partition the tuples of $\mathbf{r}$ by B. This can be done in $\mathcal{O}(|\mathbf{r}|)$ via a hash. Scanning the partition, we find the *minimum* and *maximum* values of A within each B-value group. Then the B-value partition groups are sorted by their associated min-A's. If $|\mathsf{B}| \ll |\mathsf{A}|$ ($\approx |\mathbf{r}|$), this is less expensive than sorting by A. If the intervals of the values of A co-occurring with each value of B do not *overlap*, then this is B*. To formalize this, let us define the notion of an *interval partitioning*.

Definition 11. *Let* $(\pi_{\mathsf{B}})_{\mathsf{A}} = \{\mathcal{E}_1(\mathsf{t}_{\mathsf{B}})_{\mathsf{A}}, \mathcal{E}_2(\mathsf{t}_{\mathsf{B}})_{\mathsf{A}}, \ldots, \mathcal{E}_k(\mathsf{t}_{\mathsf{B}})_{\mathsf{A}}\}$ *be the* partitioning of *an attribute* A by *an attribute* B*. Call the partitioning an* interval partitioning iff *there does not exist* $i, j \in [1, \ldots, k]$ *such that* $i < j$ *and* $min(\mathcal{E}_i(\mathsf{t}_{\mathsf{B}})_{\mathsf{A}}) \prec min(\mathcal{E}_j(\mathsf{t}_{\mathsf{B}})_{\mathsf{A}}) \prec max(\mathcal{E}_i(\mathsf{t}_{\mathsf{B}})_{\mathsf{A}})$.

An interval partitioning allows us to separate the *ranges* of A values w. r. t. the B values such that the ranges do not overlap.

Theorem 1. *Assume the* FD *of* $\mathsf{A} \to \mathsf{B}$ *holds in* $\mathbf{r}$. *Then* $\mathsf{A} \sim \mathsf{B}^*$ *holds* iff $(\pi_{\mathsf{B}})_{\mathsf{A}}$ *is an interval partitioning;* $\mathsf{T}^{\prec}_{\mathsf{B}^*}$ *is the unique order of* B*'s values corresponding to their order in* $\mathbf{r}$ *as sorted by* A.

Algorithm 1 validates an E/I OD candidate $\mathsf{A} \sim \mathsf{B}^*$ when the corresponding FD $\mathsf{A} \to \mathsf{B}$ holds. In Lines 1 and 2, the smallest and largest values of A co-occurring with each $b_i \in \mathsf{B}$ are found; this can be done in a single iteration over the database using a hash table. In Line 3, the values of B are sorted according to the smallest value of A with which they co-occur, and placed in the array B. Lines 4 to 8 check consecutive values in B. If the ranges of co-occurring A values for any two consecutive values in B overlap, this E/I OD candidate is invalid and the algorithm returns "INVALID". Otherwise, the order of values in B is *the* valid implicit total order B^* and is returned.

Algorithm 1 EIOD-FD-LeftToRight

Input: Attributes A and B.
Output: A total order over the values of B or "INVALID".

1: set $min_{\mathsf{A}}(b_i)$ as the smallest value of A co-occurring with b_i
2: set $max_{\mathsf{A}}(b_i)$ as the largest value of A co-occurring with b_i
3: B = order $b_i \in \mathsf{B}$ by $min_{\mathsf{A}}(b_i)$ ASC
4: **for** each i in $\{1, \ldots, size(B)\}$ **do**
5: **if** $min_{\mathsf{A}}(B[i+1]) < max_{\mathsf{A}}(B[i])$ **then**
6: **return** *"INVALID"*
7: **end if**
8: **end for**
9: **return** B

Example 8. *Consider attributes* count *and* size *in Table 1.1. The* FD count $\to$ size *holds;* $(\pi_{\mathsf{size}})_{\mathsf{count}}$ *is an interval partitioning with* $\tau_{\mathsf{count}} = [\mathsf{t}_{10}, \mathsf{t}_9, \mathsf{t}_3, \mathsf{t}_7, \mathsf{t}_8, \mathsf{t}_2, \mathsf{t}_6, \mathsf{t}_5, \mathsf{t}_4, \mathsf{t}_1]$. *Thus, the implied domain order* $\mathsf{T}^{\prec}_{\mathsf{size}^*}$ *is* Small $\prec$ Medium $\prec$ Large $\prec$ X-Large, *as per the* OC count $\sim$ size**; i.e, the* OD *of* count $\mapsto$ size* *holds.*

The third case looks like the second case, *except* the explicit order known over A is now on the right-hand side of our FD, $\mathsf{B} \to \mathsf{A}$. We can take a similar interval-partitioning approach as before. *Sort* the table $\mathbf{r}$ by A. If $|\mathsf{A}| \ll |\mathbf{r}|$, this is more efficient than fully sorting $\mathbf{r}$. This computes the sorted partition τ_{A}. The A values *partition* the B values, since

$B \to A$ and τ_A orders these *groups* of B values. Since there are multiple B values in some of the partition groups of τ_A, given that $A \not\to B$, this does *not* determine an order over B values within the same group. Thus the B^* implied by τ_A is not a *strong* total order, but it is a *weak* total order.

Example 9. *Let the attribute* quarter *be added to Table 1.1 to denote the year quarter; i.e.,* Q1*,* Q2*,* Q3*, and* Q4*. The* FD *of* monthGreg $\to$ quarter *holds as the Gregorian months perfectly align with the quarters. Thus, the* E/I OC quarter $\sim$ monthGreg* *holds;* monthGreg$^{\tau_{\text{quarter}}}$ *is a weak total order with* {January} $\prec$ {April, June} $\prec$ {August} $\prec$ {October, December}*. Between months within each quarter, we cannot infer any order.*

Let the FD be $A \to B$, $m = |B|$ (the number of distinct values of B), and $n = |\mathbf{r}|$ (the number of tuples). In practice, it is common that $m \ll n$.

Lemma 1. *The runtime of discovering* E/I OD*s with empty context is* $\mathcal{O}(m \ln m + n)$*.*

3.3 E/I OCs, Empty Context

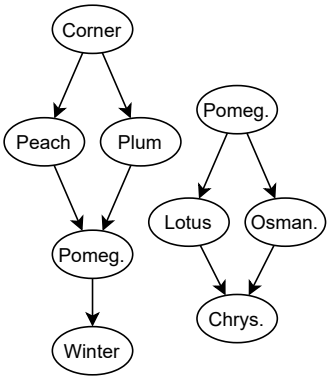
We next consider E/I OCs with an empty context in the form of $A \sim B^*$. Similar to the previous section, the goal is to verify whether there is an order over the values of B with respect to the order over the values of A. Using the sorted partitions of τ_A, we infer the order $b_i \prec b_j$ for every two distinct values of B which co-occur with two *consecutive* partition groups of A. Let B^{τ_A} denote the set of these inferred relations over B. We next check whether B^{τ_A} is a valid weak total order; if it is so, $A \sim B^*$ is a valid E/I OC and $B^* \equiv B^{\tau_A}$. Note that this case has been covered in [20], and the algorithm for it has only been reformulated here.

Theorem 2. $A \sim B^*$ *is valid* iff B^{τ_A} *is a weak total order.*

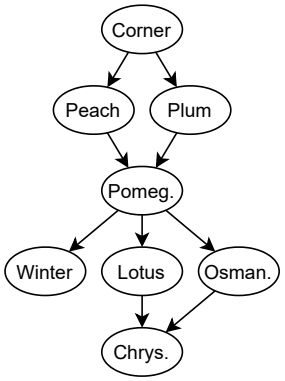
Example 10. *The* E/I OC monthNum $\sim$ monthLun* *does not hold in Table 1.1 since the lunar month* Winter *co-occurs both with December and January, which have numerical ranks of* 6 *and* 1 *in the table, resulting in* monthLun$^{\tau_{\text{monthNum}}}$ *being invalid.*

In the following, n, m, and p denote the number of tuples, the number of distinct values of the candidate attribute(s) with implicit order, and the number of partition groups of the context, respectively.

Lemma 2. *The runtime of discovering* E/I OC*s with empty context is* $\mathcal{O}(n + m^2)$*, given an initial sorting of the values in the first level of the lattice.*



(a) Order in partition groups.



(b) Union order.

Figure 3.1: Partial and union orders.

3.4 E/I ODs and E/I OCs, Non-empty Context

When the context is non-empty, say $\mathcal{X}$, we first consider each partition group in $\pi_{\mathcal{X}}$ independently. This is equivalent, with respect to each partition group, to the empty-context E/I OD and E/I OC discoveries above. For a candidate $\mathcal{X}\colon \mathsf{A} \sim \mathsf{B}^*$, if either of the FDs $\mathcal{X}\mathsf{A} \to \mathsf{B}$ or $\mathcal{X}\mathsf{B} \to \mathsf{A}$ hold, we can use the algorithms in Section 3.2. Otherwise, we use the approach in Section 3.3. If an implicit order is discovered within *each* partition group, then the *conditional* E/I OC (or E/I OD) holds. To verify the unconditional case, we take the *union* of those orders—each of which represents a weak total order—by including the edge (a, b) in the *union graph iff* this edge exists in at least one of the individual orders, and test whether this *union graph* represents a strong partial order (i.e., is cycle free). If so, we have established an unconditional B^* in the context of $\mathcal{X}$. Similar to the previous section, this case is also covered in [20]. However, if an FD holds, i.e., we have an E/I OD candidate, we can first use the more efficient algorithms presented in Section 3.2 *within* each partition group with respect to the context.

Theorem 3. *There exists an implicit domain order* $\mathsf{P}^{\prec}_{\mathsf{B}^*}$ *such that the* E/I OC $\mathcal{X}\colon \mathsf{A} \sim \mathsf{B}^*$ *holds* iff *the union graph is cycle free.*

Example 11. *The* E/I OC $\{\mathsf{yearGreg}, \mathsf{yearLunar}\} : \mathsf{monthNum} \sim \mathsf{monthLun}^{\tau_{\mathsf{monthNum}}}$ *holds* unconditionally *in Table 1.1 since the union graph is cycle free. Figure 3.1a shows the partial orders corresponding to this* E/I OC *for years (2020, 4718) and (2021, 4719) , each derived from one partition group using the algorithm described in Section 3.3. (Note that the partition group for years (2021, 4718) is ignored since it only has one tuple.) Figure 3.1b*

shows the union order. Note that in the resulting union order, an edge is included iff *it belongs to at least one of the order graphs in Figure 3.1a; e.g., the edge* $(Pomeg., Lotus)$ *is included while* $(Lotus, Osman)$ *is not.*

Example 12. *In Table 1.1, the* FD $\mathsf{country}, \mathsf{count} \to \mathsf{ribbon}$ *holds. Given the* E/I OD *candidate* $\{\mathsf{country}\}$*:* $\mathsf{count} \sim \mathsf{ribbon}^*$*,* $(\pi_{\mathsf{ribbon}})_{\mathsf{count}}$ *is an interval partitioning within each partition group with respect to the context. However, the candidate implicit orders over* ribbon—White $\prec$ Red $\prec$ Blue *within* China *and* White $\prec$ Blue $\prec$ Red *within* Canada *—are not consistent, as the* Blue *and* Red *values are flipped, making this candidate hold only* conditionally*.*

Building the graph data-structure for the union of the group orders (DAGs) is simple. This can be done by traversing the order from each partition group and adding each of their edges to the final graph if they have not been added yet. We then walk the resulting graph by *depth-first search* (DFS) to determine whether it is cycle free.

Lemma 3. *The time complexity of discovering* E/I OC*s with non-empty context is* $\mathcal{O}(n \ln n + pm^2)$*.*

Chapter 4

I/I Discovery

A surprising result is that domain orders can also be discovered even when *no* explicit domain orders are known!

- We first must extend what is meant by an implicit domain order as defined in Section 3.1: now it is *two* implicit domain orders that we seek to discover (Section 4.1).
- We then subdivide the problem of domain-order discovery via I/I OCs and ODs as follows:
  - candidates that have an empty context *or* that have a non-empty context that is treated as *conditional* (Section 4.2);
  - candidates that have a corresponding FD (Section 4.3); and
  - candidates that have a non-empty context that is treated as *unconditional* (Section 4.4).

Thus, in Section 4.1, we define when two attributes, A and B, with a context $\mathcal{X}$ can be co-ordered. We also define what *strongest* orders can be derived; i.e., A^* and B^*. The following sections then provide *algorithms* to determine when $\mathcal{X}\colon \mathsf{A}^* \sim \mathsf{B}^*$, and to compute the *strongest* orders A^* and B^* when it does.

4.1 Pairs of Implicit Domain Orders

As in the explicit-implicit case, we have *two* questions to address: when does $\mathcal{X}\colon \mathsf{A}^* \sim \mathsf{B}^*$ hold over $\mathbf{r}$; and, if it does, what are *strongest* partial orders that we can derive for A^* and B^*. Our criterion for whether $\mathcal{X}\colon \mathsf{A}^* \sim \mathsf{B}^*$ holds over $\mathbf{r}$ is the same as before: there exists *some* strong total order $\mathsf{T}^{\prec}_{\mathbf{r}}$ over the tuples in $\mathbf{r}$, a witness, such that A and B's values projected into lists from $\mathbf{r}$ ordered thusly represent strong total orders over A and

B's values, respectively. Similar to E/I OCs in Section 3.1, this definition is compatible with the one proposed in [20], where an I/I OC candidate is valid if there exist total orders over the values of A and B such that there are no swaps with respect to them within each partition group of the data.

Definition 12. *(*I/I OC witness*) The* I/I OC *of* $\mathcal{X}\colon \mathsf{A}^* \sim \mathsf{B}^*$ holds *over* $\mathbf{r}$ iff *there exists a* witness *strong total order* $\mathsf{T}_{\mathbf{r}}^{\prec}$ *such that, for* $\mathsf{s}, \mathsf{t} \in \mathbf{r}$*, if* $\mathsf{s} \prec \mathsf{t} \in \mathsf{T}_{\mathbf{r}}^{\prec}$ *and* $\mathsf{s}_{\mathcal{X}} = \mathsf{t}_{\mathcal{X}}$*, then the derived "order" relation over* A*,*

$$\mathsf{T}_{\mathsf{A}^*}^{\prec} = \{\mathsf{s}_{\mathsf{A}} \prec \mathsf{t}_{\mathsf{A}} \,|\, \mathsf{s} \prec \mathsf{t} \in \mathsf{T}_{\mathbf{r}}^{\prec} \wedge \mathsf{s}_{\mathsf{A}} \neq \mathsf{t}_{\mathsf{A}} \wedge \mathsf{s}_{\mathcal{X}} = \mathsf{t}_{\mathcal{X}}\},$$

is a strong partial order and *the derived "order" relation over* B*,*

$$\mathsf{T}_{\mathsf{B}^*}^{\prec} = \{\mathsf{s}_{\mathsf{B}} \prec \mathsf{t}_{\mathsf{B}} \,|\, \mathsf{s} \prec \mathsf{t} \in \mathsf{T}_{\mathbf{r}}^{\prec} \wedge \mathsf{s}_{\mathsf{B}} \neq \mathsf{t}_{\mathsf{B}} \wedge \mathsf{s}_{\mathcal{X}} = \mathsf{t}_{\mathcal{X}}\},$$

is a strong partial order.

To determine the strongest derivable orders for A^* and B^* is not the same as before, however. We cannot define it as simply, as the intersection of *all* the projected orders. The reason is that there is never a *single* witness; witnesses come in pairs. Since we have no explicit order to anchor the choice, if we have a strong total order on $\mathbf{r}$ that is a witness, then the *reversal* of that order is also a witness. Which direction, "ascending" or "descending", is the right one to choose? The choice is arbitrary. We call this *polarization*.

Example 13. *Consider the* I/I OC *candidate* size $\sim$ ribbon *and the first five rows of Table 1.1. The total order* $\mathsf{T}_{\mathbf{r}}^{\prec}$ *of* $\mathsf{t}_3 \prec \mathsf{t}_2 \prec \mathsf{t}_5 \prec \mathsf{t}_4 \prec \mathsf{t}_1$ *is a valid* witness order*, and results in the* derived order*s* Medium $\prec$ Large $\prec$ X-Large *and* White $\prec$ Red $\prec$ Blue *over the values of* size *and* ribbon*, respectively. However, the reverse of* $\mathsf{T}_{\mathbf{r}}^{\prec}$ *with the opposite* polarization *(i.e.,* $\mathsf{t}_1 \prec \mathsf{t}_4 \prec \mathsf{t}_5 \prec \mathsf{t}_2 \prec \mathsf{t}_3$*) is also a valid witness, resulting in reversed derived orders over* size *and* ribbon *as well. Through* I/I OC*s, it is not possible to assert which one of these polarizations is the correct one.*

To circumvent that a witness order over $\mathbf{r}$ and the reversal of that order which is also a witness "cancel" each other out (that is, their intersection is the empty order), we define a *witness class*, which consists of all the witnesses that are reachable from one another via *transpositions*. That a transposition leads to another order which is also a witness means that the forced order between tuples in the two transposed blocks is immaterial to how A and B can be co-ordered.

Definition 13. *(*transposition*) A strong total order can be represented uniquely by a sequence of the elements. Let* $\mathbf{T}$ *be a sequence of the values of* $\mathcal{D}$ *and* $\mathsf{T}_{\mathcal{D}}^{\prec}$ *be the associated strong total order. A* transposition $\mathsf{S}_{\mathcal{D}}^{\prec}$ *of* $\mathsf{T}_{\mathcal{D}}^{\prec}$ *is the associated total order of* $\mathbf{S}$ *equivalent to* $\mathbf{T}$ *in which a non-empty contiguous sub-sequence has been shifted to a new location; i.e., given* $\mathbf{T} = [\mathsf{t}_1, \ldots, \mathsf{t}_i, \mathsf{s}_1, \ldots, \mathsf{s}_k, \mathsf{t}_{i+1}, \ldots, \mathsf{t}_n]$*, then* $\mathbf{S} = [\mathsf{t}_1, \ldots, \mathsf{t}_j, \mathsf{s}_1, \ldots, \mathsf{s}_k, \mathsf{t}_{j+1}, \ldots, \mathsf{t}_n]$.

Definition 14. *(*witness class*) Let the strong total order* $\mathsf{T}_{\mathbf{r}}^{\prec}$ *be a witness of* $\mathcal{X}\colon \mathsf{A}^* \sim \mathsf{B}^*$*. We define the witness class* $\mathbb{T}_{\mathbf{r}}^{\prec}$ *over* $\mathsf{T}_{\mathbf{r}}^{\prec}$*, a collection of orders, inductively as follows:*

base case. $\mathbb{T}_{\mathbf{r}}^{\prec} = \{\mathsf{T}_{\mathbf{r}}^{\prec}\}$.
induction step. $\mathsf{Q}_{\mathbf{r}}^{\prec}$ *is added to* $\mathbb{T}_{\mathbf{r}}^{\prec}$ if $\mathsf{S}_{\mathbf{r}}^{\prec} \in \mathbb{T}_{\mathbf{r}}^{\prec}$, $\mathsf{Q}_{\mathbf{r}}^{\prec}$ is equivalent to *a transposition of* $\mathsf{S}_{\mathbf{r}}^{\prec}$*, and* $\mathsf{Q}_{\mathbf{r}}^{\prec}$ *is a witness of* $\mathcal{X}\colon \mathsf{A}^* \sim \mathsf{B}^*$.

Definition 15. *(*strongest derivable order pairs via an I/I OC*) Let* $\mathsf{T}_{\mathbf{r}}^{\prec}$ *be a witness that* $\mathcal{X}\colon \mathsf{A}^* \sim \mathsf{B}^*$ *holds over* $\mathbf{r}$*. Let* $\mathbb{A}$ *be the* collection *of derived order relations over* A *with respect to the witness orders in the witness class* $\mathbb{T}_{\mathbf{r}}^{\prec}$ *over* $\mathsf{T}_{\mathbf{r}}^{\prec}$*, and* $\mathbb{B}$ *the same but with respect to* B*. The* strongest derivable orders *for* A^* *and for* B^* via $\mathcal{X}\colon \mathsf{A}^* \sim \mathsf{B}^*$ with respect to *witness* $\mathsf{T}_{\mathbf{r}}^{\prec}$ *are*

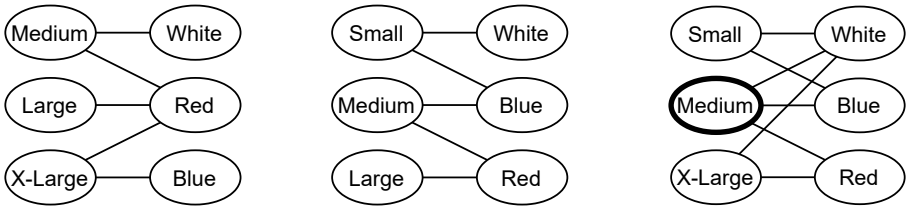
$$\mathsf{A}^* = \bigcap_{\mathsf{T}_{\mathsf{A}}^{\prec} \in \mathbb{A}} \mathsf{T}_{\mathsf{A}}^{\prec} \quad \textit{and} \quad \mathsf{B}^* = \bigcap_{\mathsf{T}_{\mathsf{B}}^{\prec} \in \mathbb{B}} \mathsf{T}_{\mathsf{B}}^{\prec}.$$

4.2 I/I OCs, Empty Context

We first consider the cases of I/I OCs with an empty context and when the context is not empty but for which we treat the partition groups as independent (*conditional*). This case has first been studied in [20], and the conditions and algorithms for it are reformulated here. For the I/I OC candidate $\mathcal{X}\colon \mathsf{A}^* \sim \mathsf{B}^*$, our goal is to discover whether, within each partition group, there exist A^* and B^* such that they can be co-ordered. To do so, we build a bipartite graph, $\mathsf{BG}_{\mathsf{A},\mathsf{B}}$ over $\mathbf{r}$. In this, the nodes on the left represent the partition groups by A's values in $\mathbf{r}$, $\pi_{\mathcal{X}\mathsf{A}}$, and those on the right represent the partition groups by B's values in $\mathbf{r}$, $\pi_{\mathcal{X}\mathsf{B}}$. For each tuple $\mathsf{t} \in \mathbf{r}$, there is an edge between t_{A} (left) and t_{B} (right).

Definition 16. (3-fan-out) *A bipartite graph has a* 3-fan-out iff *it has a node that is connected to at least three other nodes.*

It does not suffice to consider directly $\mathsf{BG}_{\mathsf{A},\mathsf{B}}$ to determine whether $\mathsf{A}^* \sim \mathsf{B}^*$. This is because a node of degree one in the BG over $\mathbf{r}$ can never invalidate the I/I OC. E.g., *White*



(a) China. (b) Canada. (c) All tuples.

Figure 4.1: BGs or BG's for I/I OC.

has just degree one in both of the BGs in Figures 4.1a and 4.1b. These have to be excluded before we check the *3-fan-out* property.

Definition 17. (Singletons and BG′) *Call a node in a* BG *with degree one a* singleton. *Let* BG′ *be the* BG *in which the singletons and their associated edges have been removed.*

With $\mathsf{BG}'_{\mathsf{A,B}}$, we can test whether $\mathsf{A}^* \sim \mathsf{B}^*$.

Theorem 4. $\mathsf{A}^* \sim \mathsf{B}^*$ *is* valid *over* $\mathbf{r}$ iff *both of the following two conditions are* true *for* $\mathsf{BG}'_{\mathsf{A,B}}$ *over* $\mathbf{r}$*:*

1. *it contains no 3-fan-out; and*
2. *it is acyclic.*

The intuition behind the requirement of no 3-fan-outs is that there has to be a way to order the left values in an attribute on the left to order the right values in an attribute on the right such that none of the edges of BG′ *cross*. Also, there is no order if there is a cycle.

Example 14. *The* $\mathsf{BG}'_{\mathsf{size,ribbon}}$ *over Table 1.1 and shown in Figure 4.1c has 3-fan-out:* Medium *connects to* White, Blue, *and* Red. *Thus, the candidate* I/I OC *of* $\{\}\colon \mathsf{size}^* \sim \mathsf{ribbon}^*$ *is not valid.*

Even though the I/I OC candidate $\{\}\colon \mathsf{size}^* \sim \mathsf{ribbon}^*$ over Table 1.1 does not hold, it does not mean that $\mathcal{X}\colon \mathsf{size}^* \sim \mathsf{ribbon}^*$ does not hold with respect to some context $\mathcal{X}$. The latter is a weaker statement.

Example 15. *Consider Table 1.1 and the* I/I OC *of* $\{\mathsf{country}\}\colon \mathsf{size}^* \sim \mathsf{ribbon}^*$. *Figures 4.1a and 4.1b show the two* BG*s for* China *and* Canada *(the values of our context,* country*), respectively. Thus, there exists a co-order between* size *and* ribbon *over* $\mathcal{E}(\mathsf{t}_{1_{\mathsf{country}}})$ *(that is, for* country = 'China'*) and a co-order between* size *and* ribbon *over* $\mathcal{E}(\mathsf{t}_{6_{\mathsf{country}}})$ *(that is, for* country = 'Canada'*).*

We next need to show how to extract a co-order once we know one exists. As with Sections 3.3 and 3.4, we may discover a partial order, this time both for *left* and *right*, within each partition group with respect to the context. The partial order is of a specific type: we find a disjoint collection of *chains*. Each chain is a strong total order over its values. Note that the singleton elements (which were initially ignored in BG') will be inserted into this order, creating the final order. Again, there is no specified direction in which to read each chain; i.e., what its polarity is.

If, for each partition group with respect to $\mathcal{X}$ over $\mathbf{r}$, $\mathsf{BG}'_{\mathsf{A},\mathsf{B}}$ over the partition group satisfies the conditions in Theorem 4, then the conditional I/I OC of $\mathcal{X}\colon \mathsf{A}^* \sim \mathsf{B}^*$ holds over $\mathbf{r}$. $\mathsf{BG}'_{\mathsf{A},\mathsf{B}}$ over each partition group yields a strong partial order—a disjoint collection of chains—for each of A and B. A walk of $\mathsf{BG}'_{\mathsf{A},\mathsf{B}}$ suffices to enumerate the chains, for both A and B.

Example 16. *Consider the* BG *in Figure 4.1b over the* I/I OC *of* $\{\mathsf{country}\}\colon \mathsf{size}^* \sim \mathsf{ribbon}^*$*, over values in Table 1.1. By iteratively zig-zagging from left to right in this bipartite graph, we obtain the chains* [Small, Medium, Large] *and* [White, Blue, Red] *over* size *and* ribbon*, respectively, over partition group* $\mathcal{E}(\mathsf{t}_{6\mathsf{country}})$.

As in Section 3.4, an I/I OC with a non-empty context can be treated either as *conditional* or *unconditional*. Our discovered domain orders between partition groups with respect to the context may differ. For the conditional case, this is considered fine; e.g., in Table 1.1, the order of ribbon colors w. r. t. the festival size differs per country: in *China*, *White* $\prec$ *Red* $\prec$ *Blue*; in *Canada*, *White* $\prec$ *Blue* $\prec$ *Red*.

Example 17. *In Table 1.1, the conditional* I/I OC *of* $\{\mathsf{country}\}\colon \mathsf{size}^* \sim \mathsf{ribbon}^*$ *holds as* $\mathcal{E}(\mathsf{t}_{1\mathsf{country}}) \models \mathsf{size}^* \sim \mathsf{ribbon}^*$ *and* $\mathcal{E}(\mathsf{t}_{6\mathsf{country}}) \models \mathsf{size}^* \sim \mathsf{ribbon}^*$.

Lemma 4. *The runtime of validating a conditional* I/I OC *with empty or non-empty context is* $\mathcal{O}(n)$.

4.3 I/I ODs, Empty Context

Discovery of domain orders via I/I ODs with an *empty context* (or with a non-empty context but considered *conditionally*) is essentially impossible. While we can discover I/I ODs that hold over the data, we can only infer the *empty* order for the domains. The FD essentially masks any information that could be derived about the orders.

Table 4.1: Valid I/I OD.

#	year	month	version#
t_1	2018	Jan	v99
t_2	2018	Feb	v100
t_3	2019	Jan	v99
t_4	2019	March	v100
t_5	2020	Feb	v99
t_6	2020	March	v100

Table 4.2: CPP to I/I OD.

#	C	A	B
$t_{1,1}$	$c_{1,1}$	t_1	b_1
$t'_{1,1}$	$c_{1,1}$	a_1	b_1
$t_{1,2}$	$c_{1,2}$	a_1	b_1
$t'_{1,2}$	$c_{1,2}$	b_1	b'_1
$t_{1,3}$	$c_{1,3}$	b_1	b_1
$t'_{1,3}$	$c_{1,3}$	f_1	b'_1

Table 4.3: I/I OC to SAT.

#	C	A	B
t_1	c_1	a_1	b_1
t_2	c_1	a_2	b_2
t_3	c_1	a_3	b_3
t_4	c_2	a_1	b_4
t_5	c_2	a_2	b_5
t_6	c_2	a_4	b_5
t_7	c_2	a_4	b_6

Theorem 5. *If* $\mathcal{X}\mathsf{A} \to \mathsf{B}$*, then the conditional* I/I OD *candidate* $\mathcal{X}\colon \mathsf{A}^* \sim \mathsf{B}^*$ *must be valid. Furthermore, there is a unique partial order that can be derived for* A^* *and for* B^**: the empty order.*

Example 18. *Consider the* I/I OC $\mathsf{festival}^* \sim \mathsf{monthGreg}^*$ *and Table 1.1. Since the* FD *of* $\mathsf{festival} \to \mathsf{monthGreg}$ *holds, the empty partial order is the implicit order over* monthGreg.

However, in the case of a candidate I/I OC with a non-empty context considered unconditionally paired with an FD that holds also "only within a non-empty context", it *is* possible for us to discover meaningful domain orders. In fact, as is shown in Section 4.4, validating these candidates is NP-complete.

Example 19. *Consider Table 4.1, which shows different versions of a software released in each year and month, and the unconditional* I/I OD *of* $\{\mathsf{year}\}\colon \mathsf{month} \mapsto \mathsf{version\#}^*$*. The only valid strong partial orders over the values of* month *and* version# *are* Jan $\prec$ Feb $\prec$ March *and* v99 $\prec$ v100*, or the reversals of these, respectively.*

4.4 I/I ODs and I/I OCs, Non-empty context

To validate an I/I OD or I/I OC candidate with a non-empty context *unconditionally* and find implicit orders A^* and B^* that hold over the *entire* dataset is significantly harder. The implicit orders for left and for right discovered per partition group must be consistent and polarity choices must be made for them.

For example, the months in the Gregorian and lunar calendars are dependent in the context of the year types with respect to the I/I OC of $\{\mathsf{yearGreg}, \mathsf{yearLun}\}\colon \mathsf{monthGreg}^*$

Table 4.4: A NAE-3SAT instance and the reduced CPP instance.

Clauses	Lists
$(p_1 \vee p_2 \vee \neg p_3)$	$[t_1, a_1, b_1, f_1]$, $[t_2, b_1, c_1, f_2]$, $[f_3, c_1, a_1, t_3]$
$(\neg p_1 \vee p_2 \vee \neg p_3)$	$[f_1, a_2, b_2, t_1]$, $[t_2, b_2, c_2, f_2]$, $[f_3, c_2, a_2, t_3]$
$(\neg p_1 \vee \neg p_2 \vee p_3)$	$[f_1, a_3, b_3, t_1]$, $[f_2, b_3, c_3, t_2]$, $[t_3, c_3, a_3, f_3]$

$\sim \mathsf{monthLun}^*$. In the lunar calendar, there are twelve months (sometimes, thirteen), with the new year starting a bit later than in the Gregorian calendar, with the lunar months overlapping the Gregorian months.

This is proved to be computationally hard in [20]. To do this, the chain polarity problem (CPP) is introduced and proved to be NP-complete. In CPP, the structure of the problem is a set of lists of values. The order of values within each list defines a total order over its values; e.g., list $[a, b, c]$ infers $a \prec b \prec c$. A *polarization* is a new collection of these lists of elements, where each list is included in its original order or after being reversed; e.g., $[a, b, c]$ or $[c, b, a]$. The decision question for CPP is whether there exists a polarization such that the union of all total orders inferred from each list is a strong partial order. CPP is proved to be NP-complete in [20] using a reduction from not-all-equal 3SAT (NAE-3SAT), which is a variation of 3SAT that requires that the three literals in each clause are not all equal to each other.

Example 20. *Table 4.4 illustrates a NAE-3SAT instance with three clauses and its equivalent CPP instance with nine lists. In the CPP instance, $t_i \prec f_i$ in the partial order is interpreted as assigning proposition p_i as* true*, and $f_i \prec t_i$ as assigning it* false*. Also, the variables a_i, b_i and c_i ensure that there exists at least one true and one false assignment for the literals in each clause. This condition is satisfied as among the three lists generated for each clause, exactly one or two of them have to be reversed in order to avoid a cycle among a_i, b_i and c_i. This translates to the corresponding literals having false assignment and the rest true assignments. Hence, any valid polarization for the lists in the CPP instance can be translated to a valid solution for the NAE-3SAT instance.*

Lemma 5. *The Chain Polarization Problem is NP-Complete.*

Using the NP-completeness results for CPP, it is shown that unconditional I/I OCs are NP-complete in [20]. Here, we extend this result and prove that even in the simpler case and in the presence of an FD, i.e., for an I/I OD candidate, validating the unconditional case is still NP-complete. We do so by establishing a mapping from CPP instances to

Unconditional I/I OD instances which takes polynomial time to compute and for which the decision questions are synonymous. Let $v_1, v_2, \ldots, v_k$ be the elements involved in the CPP instance and let $L_i = [V_{i,1}, V_{i,2}, \ldots, V_{i,|L_i|}]$ denote the i-th list of values in the CPP instance, where $V_{i,j}$ is a placeholder representing the corresponding value in a list. Without loss of generality, assume the length of all lists in the CPP instance is greater than one, as lists of length one do not affect the answer to the decision problem. To construct an unconditional I/I OD instance, consider a database $\mathbf{r}$ with attributes A, B, and C and the unconditional I/I OD candidate $\{\mathsf{C}\}$: $\mathsf{A}^* \sim \mathsf{B}^*$. For each pair of consecutive values $V_{i,j}$ and $V_{i,j+1}$ in each L_i, add the following tuples to $\mathbf{r}$: $\mathsf{t}_{i,j} = (c_{i,j}, V_{i,j}, b_i)$ and $\mathsf{t}'_{i,j} = (c_{i,j}, V_{i,j+1}, b'_i)$. Note that this is an I/I OD candidate since the FD CA $\rightarrow$ B holds. The unconditional I/I OD candidate $\{\mathsf{C}\}$: $\mathsf{A}^* \sim \mathsf{B}^*$ is valid iff there is a polarization of the CPP instance that admits a strong partial order.

Example 21. *Consider the CPP instance in Table 4.4 with $L_1 = [t_1, a_1, b_1, f_1]$. Table 4.2 represents the rows corresponding to L_1 in the* I/I OD *instance. Note that* $\{\mathsf{C}\}$*:* $\mathsf{A}^* \sim \mathsf{B}^*$ *is an* I/I OD *instance since the* FD $\{\mathsf{C}, \mathsf{A}\} \rightarrow \mathsf{B}$ *holds. In the* I/I OD *instance, the implicit order over the values b_1 and b'_1 can be established as either $b_1 \prec b'_1$ or $b'_1 \prec b_1$, which correspond to different polarizations of L_1, respectively. Similar correspondences applies to other lists in the CPP instance and other values of b_i.*

Theorem 6. *The problem of validating a given unconditional* I/I OD *with non-empty context is NP-complete.*

We illustrate our approach for validating I/I OD and I/I OC candidates by employing a high quality SAT solver in Chapter 6.

Chapter 5

Approximate Implicit Orders

In practice, many constraints do not hold *perfectly*. Dirty data, which are prevalent in the real world, contain many erroneous values which invalidate semantically valid dependencies. Furthermore, even when facing clean data, there are exceptions to general rules that prevent discovering exact dependencies. *Implicit* OCs are also subject to such inconsistencies in the data, and as a result, the discovery of implicit orders through them is affected by this issue. For instance, consider *all* tuples in Table 1.1, and the E/I OD count $\sim$ size*. This E/I OD and the implicit order over size (i.e., *Small* $\prec$ *Medium* $\prec$ *Large* $\prec$ *X-Large*) are semantically valid since the values of size *generally* increase as count increases. However, due to the exception between tuples t_2 and t_{11}, i.e., one tuple having a *larger* count but *smaller* size, this *exact* E/I OD is invalid.

Approximate dependencies are introduced to address this issue. Approximate dependencies are resistant to a small number of *exceptions* in the data and allow us to avoid overfitting by discovering more meaningful and general dependencies, even in the presence of data errors and exceptions. In this chapter, we leverage the same ideas to introduce approximate classes of implicit OCs, and consequently, approximate implicit orders. In Section 5.1, we provide a formal definition of approximate A E/I OCs and the strongest derivable orders through them. Next, in Section 5.2, we prove that the problem of validating approximate A E/I OC candidates is NP-complete, even in the simplest case of conditional A E/I ODs. In Section 5.3, we extend the definition of approximate implicit OCs to I/I OCs and provide a definition of strongest derivable orders in this case. Finally, in Section 5.4, we show that except for the trivial case of conditional A I/I ODs, all cases of approximate I/I OCs are NP-complete.

5.1 Approximate Implicit Domain Order

Similar to the exact cases for E/I OCs and I/I OCs, we need to answer two questions: first, whether an A E/I OC candidate *holds*, and second, if it does, *what* the approximate implicit order is.

To answer the first question, we again refer to *explicit-explicit* OC discovery. Given relational schema $\mathbf{r}$ and attributes A and B, an exact E/E OC holds if there exists *some* ordering of tuples of $\mathbf{r}$ that is consistent with *both* the ordering of tuples with respect to A's explicit order, and with respect to B's explicit order; i.e., there are no *swaps* in $\mathbf{r}$ with respect to the explicit orders over the values of A and B. However, for an *inexact* E/E OC, i.e., an *approximate* one, there is *no* such consistent ordering of the tuples as there always exists a number of *swaps* with respect to the orders over A and B. Therefore, there are some *exceptions*, i.e., *pairs* of tuples that do not *agree* to the orders over the values of A and B. A natural fix to this issue, then, is to count the number of these *exceptions*, as a measure of how far off the candidate is from an exact OD. This idea has been used for other types of dependencies, e.g., for FDs in [16]. To normalize this measure, we can divide it over the total number of pairs of tuples in $\mathbf{r}$. Given some swap *threshold*, an E/E OC candidate is considered valid if its number of swaps is less than or equal to the provided threshold.

For *implicit* OC candidate $\mathsf{A} \sim \mathsf{B}^*$, however, we cannot immediately use this definition, as we do not yet know the implicit order over the values of B. To resolve this, we first need to *establish* an implicit total order over the values of B. For a total order $\mathsf{T}_\mathsf{B}^\prec$ over the values of B, we use $S(\mathsf{T}_\mathsf{B}^\prec)$ to denote the total number of swaps in $\mathbf{r}$ with respect to the explicit order over the values of A and $\mathsf{T}_\mathsf{B}^\prec$. Given a threshold s for the number of swaps, we consider an A E/I OC candidate *valid* iff there exists some order $\mathsf{T}_\mathsf{B}^\prec$ such that $S(\mathsf{T}_\mathsf{B}^\prec) \leq s$. With this definition, exact E/I OCs become a special case of A E/I OCs, where there exists some order $\mathsf{T}_\mathsf{B}^\prec$ that makes the number of swaps in $\mathbf{r}$ *zero*. For an *unconditional* candidate $\mathcal{X}$: $\mathsf{A} \sim \mathsf{B}^*$ with a non-empty context, the number of swaps with respect to $\mathsf{T}_\mathsf{B}^\prec$ and the total number of swaps are counted within each partition group with respect to the context. If the candidate is to be taken conditionally, it should be valid within each partition group independent of the others.

Definition 18. *Consider an unconditional* A E/I OC *candidate* $\mathcal{X}$*:* $\mathsf{A} \sim \mathsf{B}^*$ *and an approximation threshold* $0 \leq \varepsilon \leq 1$*. This* A E/I OC *candidate is valid* iff *there exists a total order* $\mathsf{T}_\mathsf{B}^\prec$ *over the values of* B *such that* $S(\mathsf{T}_\mathsf{B}^\prec) \leq s$*, where* $s = \left\lfloor \varepsilon \sum_{\mathcal{E}_i \in \pi_\mathcal{X}} \binom{|\mathcal{E}_i|}{2} \right\rfloor$ *denotes the swap threshold.*

The second question still remains unanswered: *what* is the implicit order over the values of B? For a valid A $\mathsf{E/I}$ OC candidate with swap a threshold s, there may be different total orders $\mathsf{T}_{\mathsf{B}}^{\prec}$ that satisfy $S(\mathsf{T}_{\mathsf{B}}^{\prec}) \leq s$ with different values of $S(\mathsf{T}_{\mathsf{B}}^{\prec})$. We define a *best* derived implicit order over the values of B as a total order $\mathsf{T}_{\mathsf{B}^*}^{\prec}$ that *minimizes* the number of swaps; i.e., $S(\mathsf{T}_{\mathsf{B}^*}^{\prec}) = \min\left(\left\{S(\mathsf{T}_{\mathsf{B}}^{\prec}) \mid \mathsf{T}_{\mathsf{B}}^{\prec} \text{ is a total order over } \mathsf{B}\right\}\right)$. This is again an extension of the exact case, in which $\mathsf{T}_{\mathsf{B}^*}^{\prec}$ is derived from some *witness* total order over the tuples and makes the number of swaps zero. In the exact case, we define the *strongest* derivable order as the intersection of derived orders from all witness total orders. Here, however, we cannot directly apply this definition, as there are *no* witness orders over the tuples, because of the exceptions in $\mathbf{r}$ with respect to the order over A and $\mathsf{T}_{\mathsf{B}^*}^{\prec}$. Therefore, we first need to *resolve* all exceptions with respect to the explicit order over the values of A and $\mathsf{T}_{\mathsf{B}^*}^{\prec}$. To do so, we first remove the smallest number of tuples such that at least *one* tuple is removed from each exception with respect to the order over the values of A and B^*, effectively making the number of *swaps* with respect to these orders *zero*, as in the exact case.

With the repaired data, we can now employ the definition of a *strongest* derivable order from Chapter 3. We define the strongest derivable order, denoted by B^*, as the intersection of orders derived from all witness permutations of this repaired dataset, as is for the exact case. This is a *subset* partial order of the derived best total order.

Example 22. *In Table 1.1, consider the* A $\mathsf{E/I}$ OD *candidate* $\mathsf{count} \sim \mathsf{size}^*$ *over all tuples, with an approximation threshold of* 0.1. *Therefore, the swap threshold is computed as* $\left\lfloor 0.1 \cdot \binom{11}{2} \right\rfloor = 5$. *The order* $\mathsf{T}_{\mathsf{size}^*}^{\prec} = \text{Small} \prec \text{Medium} \prec \text{Large} \prec \text{X-Large}$ *is a* best *derived order over the values of* size*, with* $S(\mathsf{T}_{\mathsf{size}^*}^{\prec}) = 1 \leq 5$ *(corresponding to tuples* t_2 *and* t_{11}*), making the candidate valid. To find the strongest derivable order, we first remove one of the tuples involved in the swap and then use the definition of strongest derivable order from Chapter 3. The strongest derivable order in this case is the same as the best derived order; i.e.,* $\mathsf{size}^* = \text{Small} \prec \text{Medium} \prec \text{Large} \prec \text{X-Large}$.

5.2 Hardness of Validating Approximate E/I OCs

We now consider the problem of validating A $\mathsf{E/I}$ OC and A $\mathsf{E/I}$ OD candidates. We show that even in the *simplest* form, i.e., for conditional A $\mathsf{E/I}$ OD candidates, the validation problem is NP-complete. This is unlike the exact case, where validation is possible in polynomial time for *all* cases.

To prove the NP-completeness of the problem of validating conditional A $\mathsf{E/I}$ ODs, we offer a polynomial mapping from instances of the decision variant of minimum feedback

arc set (MFAS) problem, in which the the MFAS instance holds *iff* A E/I OD candidate is valid.

The definition of MFAS is as follows: given a simple directed graph G, find the smallest set of edges (arcs) such that when removed, the graph will become acyclic. Here, we use an equivalent definition: given a simple directed graph G with vertices $v_1, \ldots, v_n \in V$, find a total order $\mathsf{T}_V^{\prec}$ over the vertices such that the number of *back-edge*s in G is minimized. A back edge is defined as an edge (v_i, v_j) where $v_j \prec v_i \in \mathsf{T}_V^{\prec}$. The decision variant of MFAS, here referred to as MFAS for simplicity, is to answer whether there exists a total order $\mathsf{T}_V^{\prec}$ with *at most* t back-edges for a given t, and is known to be NP-complete [8].

Assume G is the graph in the MFAS instance. Let v_1, v_2, ..., $v_n \in V$ and e_1, ..., $e_m \in E$ denote the vertices and edges in G, respectively. Let $e_i[0]$ and $e_i[1]$ denote the beginning and end of e_i, respectively, i.e., $e_i = (e_i[0], e_i[1])$. Finally, let t be the threshold for the number of back-edges; i.e., the MFAS instance holds iff there exists a total order over its vertices with at most t back-edges.

To construct the approximate E/I OD instance, let $\mathbf{r}$ be the relational instance with attributes A and B and let $\mathsf{A} \sim \mathsf{B}^*$ be the approximate E/I OD instance. Let $\mathbb{Z}$ and $\{v_1, v_2, \ldots, v_n\}$ be the domains of A and B, respectively. For each edge e_i, add the following *four* tuples to $\mathbf{r}$: $\mathsf{t}_{-2i} = (-2i, e_i[0])$, $\mathsf{t}_{-2i+1} = (-2i+1, e_i[1])$, $\mathsf{t}_{2i-1} = (2i-1, e_i[0])$, and $\mathsf{t}_{2i} = (2i, e_i[1])$. This is an A E/I OD since the values of A are unique and an FD holds. Call a pair of tuples t_i and t_{i+1} in $\mathbf{r}$ *adjacent* if $i = -2k$ or $i = 2k-1$ for some $1 \leq k \leq m$, and all other pairs of tuples *non-adjacent*. Finally, let $s = C/2 + 2t$ be the threshold for the number of swaps in the A E/I OD instance, where $C = |\{\{\mathsf{s}, \mathsf{t}\} \mid \mathsf{s} \text{ and } \mathsf{t} \text{ are } \textit{non-adjacent} \wedge \mathsf{s_C} \neq \mathsf{t_C} \wedge \mathsf{s_D} \neq \mathsf{t_D}\}|$. Intuitively, C denotes the number of non-adjacent pairs of tuples that *can* be swapped; i.e., have different A- and B-values.

Example 23. *Consider the simple graph in Figure 5.1 as the input to the MFAS problem. Table 5.1 corresponds to the* A E/I OD *instance after the reduction. (Note that this instance holds exactly as well, but is chosen to avoid a large table size.) For edge* (a, b)*, we create pairs of* adjacent *tuples* t_{-2} *and* t_{-1} *as well as* t_1 *and* t_2*, and similarly the remaining pairs of tuples for the edge* (b, c)*. Here, one* best *derived order in the* A E/I OD *instance is* $\mathsf{T}_{\mathsf{B}^*}^{\prec} = a \prec b \prec c$*, which corresponds to the order* $\mathsf{T}_V^{\prec} = a \prec b \prec c$ *over the vertices in the MFAS instance. Note that there are no swaps between* adjacent *pairs of tuples in the* A E/I OD *instance with respect to* $\mathsf{T}_{\mathsf{B}^*}^{\prec}$*, which is consistent with there being no* back-edges *in the MFAS instance with respect to* $\mathsf{T}_V^{\prec}$*.*

Theorem 7. *Validating conditional* A E/I OD*s is NP-complete.*

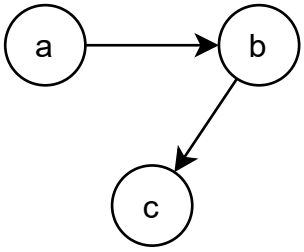


Figure 5.1: MFAS instance.

Table 5.1: A E/I OD instance.

#	A	B
t_{-4}	-4	b
t_{-3}	-3	c
t_{-2}	-2	a
t_{-1}	-1	b
t_1	1	a
t_2	2	b
t_3	3	b
t_4	4	c

5.3 Pairs of Approximate Implicit Domain Orders

Here, again, we have two questions regarding pairs of implicit orders: first, defining when an A I/I OC candidate is valid, and second, defining the implicit orders that can be discovered if the candidate is valid. Similar to A E/I OCs, we cannot directly apply the definition of valid I/I OCs to A I/I OCs since there are *no* witness total orders over the tuples. However, we can still define the validity of a candidate based on the number of exceptions, i.e., swaps, that exist in the data with respect to implicit orders.

Unlike A E/I OCs, where we only need to establish an implicit order over the values of one of the attributes, for an A I/I OC candidate, a *pair* of implicit orders have to be established over the values of the attributes. For an A I/I OC candidate $\mathsf{A}^* \sim \mathsf{B}^*$ in relational schema $\mathbf{r}$ and total orders $\mathsf{T}_\mathsf{A}^\prec$ and $\mathsf{T}_\mathsf{B}^\prec$ over the values of A and B, respectively, let $S(\mathsf{T}_\mathsf{A}^\prec, \mathsf{T}_\mathsf{B}^\prec)$ denote the number of swaps in $\mathbf{r}$ with respect to $\mathsf{T}_\mathsf{A}^\prec$ and $\mathsf{T}_\mathsf{B}^\prec$. Given a swap threshold s, the validation problem for A I/I OCs then becomes determining if there exist total orders $\mathsf{T}_\mathsf{A}^\prec$ and $\mathsf{T}_\mathsf{B}^\prec$ over the values of A and B such that $S(\mathsf{T}_\mathsf{A}^\prec, \mathsf{T}_\mathsf{B}^\prec) \leq s$. Similar to A E/I OCs in Section 5.1, dividing s over the total number of tuples in $\mathbf{r}$ normalizes the swap threshold as the approximation threshold. Again, exact I/I OCs become a special case of A I/I OCs where the swap threshold is set to *zero*. For A I/I OCs with a non-empty context taken *unconditionally*, pairs of tuples are considered only *within* each partition group of the data with respect to the context. For *conditional* candidates, the validity of the candidates is considered within each partition group *independent* of the others.

Definition 19. *Consider an unconditional* A I/I OC *candidate* $\mathcal{X}$*:* $\mathsf{A}^* \sim \mathsf{B}^*$ *and an approximation threshold* $0 \leq \varepsilon \leq 1$*. This* A I/I OC *candidate is valid* iff *there exist total orders* $\mathsf{T}_\mathsf{A}^\prec$ *and* $\mathsf{T}_\mathsf{B}^\prec$ *over the values of* A *and* B*, respectively, such that* $S(\mathsf{T}_\mathsf{A}^\prec, \mathsf{T}_\mathsf{B}^\prec) \leq s$*, where*

$s = \left\lfloor \varepsilon \sum_{\mathcal{E}_i \in \pi_{\mathcal{X}}} \binom{|\mathcal{E}_i|}{2} \right\rfloor$ *denotes the swap thresholds.*

As for a pair of *strongest* derivable orders, we first define a pair of *best* derived orders $\mathsf{T}^{\prec}_{\mathsf{A}^*}$ and $\mathsf{T}^{\prec}_{\mathsf{B}^*}$ as total orders over the values of A and B, respectively, that minimize the number of swaps in $\mathbf{r}$; i.e., $S(\mathsf{T}^{\prec}_{\mathsf{A}^*}, \mathsf{T}^{\prec}_{\mathsf{B}^*}) = \min\big(\{S(\mathsf{T}^{\prec}_{\mathsf{A}}, \mathsf{T}^{\prec}_{\mathsf{B}}) \mid \mathsf{T}^{\prec}_{\mathsf{A}} \text{ and } \mathsf{T}^{\prec}_{\mathsf{B}} \text{ are total orders over } \mathsf{A} \text{ and } \mathsf{B}\}\big)$. After discovering a pair of best derived orders, we again *resolve* all of the exceptions with respect to the *best* implicit orders discovered; i.e., we make the number of *swaps* zero by removing the minimum number of tuples possible. With this repaired data, our definition of strongest derivable orders from Section 4.1 becomes applicable again. Therefore, the strongest derivable orders A^* and B^* are partial order *subsets* of $\mathsf{T}^{\prec}_{\mathsf{A}^*}$ and $\mathsf{T}^{\prec}_{\mathsf{B}^*}$, respectively.

Example 24. *Consider the* A I/I OC *candidate* $\mathsf{size}^* \sim \mathsf{ribbon}^*$ *over tuples* t_6 – t_{11} *in Table 1.1 (i.e., all festivals in Canada) with* 0.1 *as the approximation threshold. the swap threshold is computed as* $\lfloor 0.1 \cdot \binom{6}{2} \rfloor = 1$. *The orders* $\mathsf{T}^{\prec}_{\mathsf{size}^*} =$ Small $\prec$ Medium $\prec$ Large *and* $\mathsf{T}^{\prec}_{\mathsf{ribbon}^*} =$ White $\prec$ Blue $\prec$ Red *constitute a pair of* best *derived orders over the values of* size *and* ribbon*, with* $S(\mathsf{T}^{\prec}_{\mathsf{size}^*}, \mathsf{T}^{\prec}_{\mathsf{ribbon}^*}) = 1$ *(corresponding to tuples* t_8 *and* t_{11}*). Since the number of swaps is within the swap threshold, this candidate holds approximately. After resolving this swap by removing one of the tuple* t_8 *or* t_{11}*, the strongest derivable orders* size^* *and* ribbon^* *can be derived and are the same as* $\mathsf{T}^{\prec}_{\mathsf{size}^*}$ *and* $\mathsf{T}^{\prec}_{\mathsf{ribbon}^*}$*, respectively.*

5.4 Hardness of Validating Approximate I/I OCs

As has been proved in Section 4.4, even *exact* unconditional I/I OCs and I/I ODs with a non-empty context are NP-complete to validate. Conditional exact I/I ODs, however, as has been shown in Section 4.3, are *always* valid and the strongest derivable orders using them are *empty* partial orders. These results translate to the approximate case: validating unconditional A I/I OCs and A I/I ODs is NP-complete, while conditional A I/I ODs are *always* valid (i.e., there always exist total orders over the values of attributes such that there are *zero* swaps in the data) and the strongest derivable orders using them are empty partial orders. However, we cannot directly infer the complexity of *conditional* A I/I OCs from previous results, as exact conditional I/I OCs take polynomial time to validate. Here, we show that validating conditional A I/I OCs is NP-complete. This concludes that, except for the trivial case of conditional A I/I ODs, validating *all* types of approximate implicit OCs is NP-complete. We offer a polynomial reduction from instances of A E/I OC with empty context to instances of A I/I OC with empty context, in which the A E/I OC instance is valid *iff* the A I/I OC instance is valid.

Let relational instance $\mathbf{r}$ with attributes A and B be the input to the A $\mathsf{E/I}$ OC instance $\mathsf{A} \sim \mathsf{B}^*$. Let $|\mathbf{r}| = n$ and $\{a_1, \ldots, a_l\}$ and $\{b_1, \ldots, b_m\}$ be the active domains of A and B, respectively. Without loss of generality, let $a_1 < a_2 \cdots < a_l$ be the *explicit* order over the values of A. For the corresponding A $\mathsf{I/I}$ OC instance, let $\mathbf{t}$ be the relational instance with attributes C and D and let $\mathsf{C}^* \sim \mathsf{D}^*$ be the A $\mathsf{I/I}$ OC candidate. Let $\{c_1, c_2, \ldots, c_{2l+1}\}$ and $\{D_{1,2}, D_{2,3}, \ldots, D_{2l,2l+1}, d_1, d_2, \ldots, d_m\}$ be the domains of C and D, respectively. Let $N = 8n^2$, and add the following *three* types of tuples to $\mathbf{t}$.

1. For all $1 \leq i \leq 2l-1$ add $2N$ *dummy* tuples $\mathsf{d}_{2N(i-1)+1}, \mathsf{d}_{2N(i-1)+2}, \ldots, \mathsf{d}_{2N(i-1)+2N}$ to $\mathbf{t}$, where N of the tuples have values $(c_i, D_{i,i+1})$ and the other N tuples have values $(c_{i+1}, D_{i,i+1})$.
2. For all $1 \leq i \leq m$ add N *anchor* tuples $\mathsf{a}_{N(i-1)+1}$, $\mathsf{a}_{N(i-1)+2}$, $\ldots$, $\mathsf{a}_{N(i-1)+N}$ to $\mathbf{t}$, with values (c_{l+1}, d_i).
3. Finally, for each tuple $\mathsf{t}_k = (a_i, b_j)$ in $\mathbf{r}$, add the *real* tuples $\mathsf{r}'_k = (c_i, d_j)$ and $\mathsf{r}''_k = (c_i, d_j)$ to $\mathbf{t}$.

Let d, a, and r denote the sets of *dummy*, *anchor*, and *real* tuples, respectively. Moreover, the set of real tuples is itself divided into two disjoint sets, r' and r'', containing tuples r'_k and r''_k, respectively. For classes of tuples s and t, let $S_{\mathsf{s},\mathsf{t}}(\mathsf{T}^{\prec}_{\mathsf{C}}, \mathsf{T}^{\prec}_{\mathsf{D}})$ denote the number of swaps between tuples in these classes with respect to orders $\mathsf{T}^{\prec}_{\mathsf{C}}$ and $\mathsf{T}^{\prec}_{\mathsf{D}}$; i.e., $S_{\mathsf{s},\mathsf{t}}(\mathsf{T}^{\prec}_{\mathsf{C}}, \mathsf{T}^{\prec}_{\mathsf{D}}) = \big|\big\{\{\mathsf{s}_i, \mathsf{t}_j\} \mid \mathsf{s}_i \text{ and } \mathsf{t}_j \text{ are swapped w.r.t. } \mathsf{T}^{\prec}_{\mathsf{C}} \text{ and } \mathsf{T}^{\prec}_{\mathsf{D}}\big\}\big|$. When $\mathsf{T}^{\prec}_{\mathsf{C}}$ and $\mathsf{T}^{\prec}_{\mathsf{D}}$ are clear from the context, we use $S_{\mathsf{s},\mathsf{t}}$ to reduce verbosity. Let $C_{\mathsf{s},\mathsf{t}}$ denote the number of tuples of type s and t that have different left- *and* right-side values and *can* be swapped; i.e., $\big|\big\{\{\mathsf{s}_i, \mathsf{t}_j\} \mid \mathsf{s}_{i\mathsf{C}} \neq \mathsf{t}_{j\mathsf{C}} \wedge \mathsf{s}_{i\mathsf{D}} \neq \mathsf{t}_{j\mathsf{D}}\big\}\big|$. Finally, let s be the threshold for the number of swaps in the A $\mathsf{E/I}$ OC and set $C/2 + 2s$ as the threshold for the number of swaps in the A $\mathsf{I/I}$ OC instance, where $C = C_{\mathsf{a},\mathsf{r}} + C_{\mathsf{d},\mathsf{r}} + C_{\mathsf{r}',\mathsf{r}''}$. (Assume $s \leq \binom{n}{2}$, as otherwise the candidate is always valid.)

Example 25. *Consider the* A $\mathsf{E/I}$ OC *candidate* $\mathsf{A} \sim \mathsf{B}^*$ *in Table 5.2 and let* $a_1 < a_2$ *be the explicit order over the values of* A. *Figure 5.2 corresponds to the* A $\mathsf{I/I}$ OC *instance, where the bipartite graph is used as the tabular representation would be too large. (Note that the* A $\mathsf{E/I}$ OC *instance holds exactly as well, but is chosen to avoid making the graph for the* A $\mathsf{I/I}$ OC *instance too complex.) Here, the* A $\mathsf{I/I}$ OC *candidate is* $\mathsf{C}^* \sim \mathsf{D}^*$ *and* $\{c_1, \ldots, c_5\}$ *and* $\{D_{1,2}, \ldots, D_{4,5}, d_1, d_2\}$ *denote the domains of* C *and* D, *respectively. In Figure 5.2, thin edges correspond to* real *tuples, e.g., the edge* (c_1, d_1), *while wide edges correspond to* dummy *or* anchor *tuples which are repeated* $N = 8n^2 = 8 \cdot 2^2 = 32$ *times; e.g., the edge* (c_3, d_1) *corresponds to* N *anchor tuples. A pair of* best *orders* $\mathsf{T}^{\prec}_{\mathsf{C}^*} = c_1 \prec c_2 \prec \cdots \prec c_5$

Table 5.2: A E/I OC instance.

#	A	B
t_1	a_1	b_1
t_2	a_2	b_2

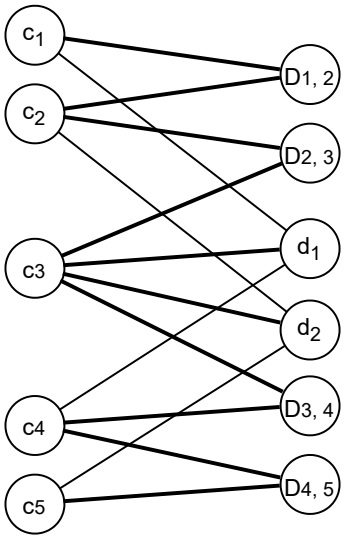


Figure 5.2: Bipartite graph of the A I/I OC instance.

Table 5.3: Summary of complexity results.

	E/I				I/I			
	Cond.		Uncond.		Cond.		Uncond.	
	OD	OC	OD	OC	OD	OC	OD	OC
Exact	Poly	Poly	Poly	Poly	Trivial	Poly	NPC	NPC
Approx.	NPC	NPC	NPC	NPC	Trivial	NPC	NPC	NPC

and $\mathsf{T}^{\prec}_{\mathsf{D}^*} = D_{1,2} \prec D_{2,3} \prec d_1 \prec d_2 \prec D_{3,4} \prec D_{4,5}$ *for the* A I/I OC *instance corresponds to a best order* $\mathsf{T}^{\prec}_{\mathsf{B}^*} = b_1 \prec b_2$ *for the* A E/I OC *instance.*

Theorem 8. *Validating conditional* A I/I OC*s is NP-complete.*

Table 5.3 summarizes the complexity results regarding exact and approximate explicit-implicit and implicit-implicit candidates of different classes.

Chapter 6

Using SAT Solvers for NP-complete Cases

We now describe three reductions to SAT instances for the three NP-complete cases, i.e., *exact* I/I OCs, A E/I OCs, and A I/I OCs. (Note that here, the OC and OD cases are not distinguished as they are solved using the same algorithms.)

In our translation to SAT instances, we construct variables such that a solution to the SAT instance corresponds to a *total* order over the values of the attribute(s) with an implicit order. To ensure that the final relation corresponds to a valid total order and is consistent with the constraints imposed by the data, we rely on *two* types of clauses.

1. **Valid Order.** Recall from Definition 1 that a valid (strong) total order must satisfy three conditions: irreflexivity, connexivity, and transitivity. Irreflexivity is ensured by the way we generate our SAT variables. We create two subtype clauses to guarantee a valid order: *connexivity* and *transitivity* clauses.

2. **Swaps.** The second type of clauses corresponds to the constraints imposed by the data; i.e., not having *any* swaps in the exact case, and *minimizing* the number of swaps in the approximate cases.

6.1 Exact Unconditional I/I OCs

Given that discovering implicit domain orders via I/I OCs is NP-complete, we reduce it to an instance of the SAT problem to validate the candidate and then to establish valid strong

partial orders. The first step is similar to the conditional case in Section 4.2: we derive bipartite graphs, BG_i's, for the tuples from each partition group. Presence of cyclicity or 3-fan-out invalidates the candidate, as by Theorem 4. Thus next, we check each BG_i for cyclicity or 3-fan-out; this validates or invalidates the candidate in linear time.[1]

We now explain the translation to SAT for an I/I OC candidate $\mathcal{X}$: $\mathsf{A}^* \sim \mathsf{B}^*$. The input is a list of bipartite graphs, BG's, which indicate the co-occurrence of values in each partition. For each bipartite graph BG_j, the left-hand side (LHS) and right-hand side (RHS) denote the values of the two attributes of the I/I OC candidate, respectively. Let $a_1, \ldots, a_{m'}$ and $b_1, \ldots, b_{m''}$ denote the distinct values of each.

We define two sets of propositional variables: $\forall 1 \leq i, j \leq m' : a_{i,j}$ and $\forall 1 \leq i, j \leq m'' : b_{i,j}$, for distinct i and j values. Assigning *true* to a variable $a_{i,j}$ indicates $a_i \prec a_j$, while assigning *false* means that $a_j \prec a_i$. As discussed before, we need to add *two* types of clauses, one to ensure the *validity* of the order, and the second to guarantee not having any *swaps*.

1. **Valid Order.** *Irreflexivity* is automatically satisfied since no variables $a_{i,i}$ or $b_{i,i}$ are created. However, we need two subtypes of clauses to guarantee a valid order:
   (a) *Connexivity.* For all distinct i and j such that $1 \leq i, j \leq m'$, we add clauses $\neg(a_{i,j} \wedge a_{j,i}) \equiv (\neg a_{i,j} \vee \neg a_{j,i})$ and $(a_{i,j} \vee a_{j,i})$, as exactly *one* of variables $a_{i,j}$ and $a_{j,i}$ has to be *true*.
   (b) *Transitivity.* For all distinct i, j, and k such that $1 \leq i, j, k \leq m'$, we add clause $(a_{i,j} \wedge a_{j,k}) \implies a_{i,k} \equiv (\neg a_{i,j} \vee \neg a_{j,k} \vee a_{i,k})$.

   We add similar clauses for variables $b_{i,j}$.
2. **No Swaps.** For all $(a_i, b_u), (a_j, b_v) \in \mathsf{BG_k}$ such that $a_i \neq a_j$ and $b_u \neq b_v$, we add the clauses $(a_{i,j} \wedge b_{u,v}) \vee (a_{j,i} \wedge b_{v,u}) \equiv (a_{i,j} \vee b_{v,u}) \wedge (a_{j,i} \vee b_{u,v})$. Note that the initial conditions $((\neg a_{i,j} \vee \neg a_{j,i})$ and $(\neg b_{u,v} \vee \neg b_{v,u}))$ were used to simplify these conditions.

Theorem 9. *The unconditional* I/I OC *candidate is valid* iff *the corresponding SAT instance is satisfiable.*

If the SAT instance is satisfiable, to derive the final partial orders over the values of A and B, we take the satisfying assignment and set $i \prec j$ for A iff $a_{i,j} = \mathit{true}$, and similarly

[1] The constraint that each BG_i has no 3-fan-out restricts the size of the graph to be linear in the number of distinct values of the domain. Without this, the size of the graph could be quadratic in the number of distinct values of the domain.

for the values of B. To achieve a pair of *strongest derivable orders*, we remove the order over pairs of values which should not exist in the final order, while keeping the order graph valid. For every pair of distinct values $a_u, a_v \in \mathsf{PG}_i$ where $a_u \prec a_v$, we keep them in the final order *iff* one of these conditions holds (and similarly for the RHS values): 1) the nodes a_u and a_v are in the same connected component in $\mathsf{BG_i}$ and the path from a_u to a_v contains at least two nodes with degree two or larger; 2) the nodes a_u and a_v are in the same connected component and the first condition does not hold, however, there exists $\mathsf{BG_j}$ with $j \neq i$ such that $a_u, a_v \in \mathsf{BG}_j$; and 3) the nodes a_u and a_v are in different connected components, but there exists $\mathsf{BG_j}$ with $j \neq i$ and distinct values v_r, v_s belonging to the same attribute such that v_r, $v_s \in \mathsf{BG}_j$ and v_r and v_s are in the same connected components in BG_i as a_u and a_v, respectively (note that v_r and v_s could be the same as a_u and a_v). Intuitively, either of these conditions would make it impossible to *remove* an order between two values through valid *transposition*s within the same *witness class*, as defined in Section 4.1.

Algorithms 2 and 3 demonstrate the main steps to remove edges from the partial order output of the SAT solver in order to achieve a pair of strongest derivable orders. Each of these algorithms generates a set of pairs of values (i.e., *nonremovables*), which must be kept in the final *strongest* derivable order graph. Note that these algorithms describe the steps taken only for values on one side of the candidate, which can be repeated for the other side as well.

Algorithm 2 corresponds to the first condition described above. Lines 2 to 10 find pairs of values that satisfy this condition, i.e., pairs which exist within the same connected component of some PG_i such that there are at least two nodes with degree two or larger along their connecting path. This is done by running a DFS from every node a_u in the graph and detecting all nodes a_v which satisfy this condition. Algorithm 3 corresponds to the second and third conditions described above. Line 2 creates a hashmap, which, for each two values a_u and a_v, stores the bipartite graphs in which a_u and a_v are connected. Lines 3 to 7 store the co-occurrences of such pairs of values in the variable *samePG*. Lines 9 to 13 correspond to second condition. They check all singleton values a_u and a_v that are connected to the same value and add this pair to *nonremovables* if they co-occur in at least one other partition group. Lines 14 to 20 check the third condition. To do so, they traverse over all pairs of connected components CC_j and CC_k in a BG and check for values on the same side of these connected components that co-occur in at least another partition group. If this condition holds, all pairs of values from these connected components are added to *nonremovables*.

The remaining step is to compute the union of output sets of Algorithms 2 and 3 and to traverse over all the edges of the initial partial order graph (i.e., the output of the SAT

solver) and remove those which do not exist in the union set, creating the final graph for a strongest derivable order.

Algorithm 2 IIOC-SameConnectedComponent

Input: set of bipartite graphs $\mathsf{BG} = \{\mathsf{BG}_1, \ldots, \mathsf{BG}_k\}$.
Output: set of pairs of values that must be kept in the final order.

```
1: set nonremovables as an empty set
2: for each BG_i ∈ BG do
3:    for each a_u ∈ BG_i do
4:       run DFS from a_u and add nodes with degree ≥ 2 to degTwo
5:       when visiting node a_v:
6:       if size(degTwo) ≥ 2 then
7:          add the pair (a_u, a_v) to nonremovables
8:       end if
9:    end for
10: end for
11: return nonremovables
```

Example 26. *Consider Table 4.3 and the* I/I OC $\{\mathsf{C}\}$*:* $\mathsf{A}^* \sim \mathsf{B}^*$ *candidate. The propositional variables for the reduction are* $a_{1,2}$, $a_{2,1}$, …, $a_{4,3}$ *and* $b_{1,2}$, …, $b_{6,5}$. *First,* connexity *clauses, e.g.,* $(\neg a_{1,2} \vee \neg a_{2,1}) \wedge (\neg a_{2,3} \vee \neg a_{3,2})$ *and* transitivity *clauses, e.g.,* $(a_{1,2} \wedge a_{2,3}) \implies a_{1,3}$, $(a_{1,3} \wedge a_{3,2}) \implies a_{1,2}$ *are created to ensure a valid strong total order. Next, for the* no swaps *condition, the clauses* $(a_{1,2} \vee b_{2,1}) \wedge (a_{2,1} \vee b_{1,2})$ *are generated for tuples* t_1 *and* t_2*, and similarly for the rest of the pairs of tuples in* BG_1 *and* BG_2.

Since this is a valid I/I OC*, some strong partial order can be derived using the SAT variable assignments. To derive the* strongest *orders from this pair, the final orders between values such as* (a_2, a_4) *and* (a_1, a_2) *on the LHS are kept, as these pairs satisfy the first and third condition, respectively (values* a_2 *and* a_4 *exist in a connected component in* BG_2 *with two nodes with degree two along their path, and values* a_1 *and* a_2 *are present on the LHS of both* BG_1 *and* BG_2*). However, the order between values* a_1 *and* a_3 *(and similarly* a_2 *and* a_3*) is removed since these values do not satisfy any of the conditions. This process is repeated for the RHS as well.*

Lemma 6. *The reduction to SAT for* I/I OC*s is* $\mathcal{O}(n + pm^2 + m^3)$.

Since m tends to be small in practice (i.e., $m \ll |\mathbf{r}|$) for meaningful cases and p heavily depends on m as well, this runtime is manageable in real-life applications (Chapter 8).

We also use an optimization based on the overlap between values in different partition groups to decrease the number of variables and clauses. Initially, we compute disjoint sets

Algorithm 3 IIOC-MutualPairsOfValuesInPGs

Input: set of bipartite graphs $\mathsf{BG} = \{\mathsf{BG}_1, \ldots, \mathsf{BG}_k\}$.
Output: set of pairs of values that must be kept in the final order.

set *nonremovables* as an empty set
set *samePG* as a hashmap from pairs of values in L to empty sets
for each $\mathsf{BG}_i \in \mathsf{BG}$ **do**
  for each a_u, $a_v \in \mathsf{BG}_i$ **do**
    add i to $samePG[a_u, a_v]$
  end for
end for
for each $\mathsf{BG}_i \in \mathsf{BG}$ **do**
  for each two singleton values a_u and a_v connected to b_r **do**
    if $size(samePG[a_u, a_v]) \geq 2$ **then**
      add the pair (a_u, a_v) to *nonremovables*
    end if
  end for
  for each two connected components CC_j and CC_k in BG_i **do**
    if there exist v_r, v_s in the same attribute with $v_r \in CC_j$ and $v_s \in CC_k$ such that $size(samePG[v_r, v_s]) \geq 2$ **then**
      for each $a_u \in CC_j$ and $a_v \in CC_k$ **do**
        add the pair (a_u, a_v) to *nonremovables*
      end for
    end if
  end for
end for
return *nonremovables*

of values that have co-appeared in the same partition groups. While considering pairs or triples of values to generate variables or transitivity clauses, respectively, we consider these sets of attributes separately, which reduces the runtime of the algorithm, without affecting the correctness of the reduction. Furthermore, before reducing to a SAT instance, we consider pairwise BGs and check their compatibility, in order to falsify impossible cases as early as possible.

6.2 Approximate E/I OCs

We offer a reduction from approximate E/I OCs to *partial* MAX-SAT, which allows us to use an efficient MAX-SAT solvers to solve these instances. Partial MAX-SAT is a variant of MAX-SAT, in which there are two groups of *hard* and *soft* clauses. The goal is to satisfy *all* of the hard clauses, while maximizing the number of satisfied soft clauses. A *solution* to the MAX-SAT instance is an assignment for the boolean variables such that all hard clauses are satisfied.

Let $\mathbf{r}$ be a relational schema instance with n tuples. Let $\mathcal{X}$: $\mathsf{A} \sim \mathsf{B}^*$ and s be the A E/I OC instance and the input swap threshold, respectively. Let $a_1, a_2, \ldots, a_l$ and $b_1, \ldots, b_m$ denote the values of A and B, respectively. Without loss of generality, assume $a_1 \prec \cdots \prec a_l$ is the explicit order over the values of A. We define a SAT variable $b_{i,j}$ for all distinct i and j such that $1 \leq i, j \leq m$. Intuitively, we assume that in the MAX-SAT solution, having $b_{i,j} = \mathit{true}$ implies $b_i \prec b_j \in \mathsf{T}^{\prec}_{\mathsf{B}}$. Similar to unconditional exact I/I OCs, we utilize two types of clauses: *valid order* and *swaps*. Valid order clauses are defined as *hard* clauses since they all must be satisfied to ensure a valid strong total order. However, as the candidate does not hold *exactly*; i.e., there are always swaps with respect to an order $\mathsf{T}^{\prec}_{\mathsf{B}}$, we define the swaps clauses as *soft* clauses in order to minimize the number of swaps. These clauses are defined as follows.

1. ***Hard* Valid Order.**
   (a) *Connexivity.* For all distinct i and j such that $1 \leq i, j \leq m$, we add clauses $(b_{i,j} \vee b_{j,i})$ and $(\neg b_{i,j} \vee \neg b_{j,i})$.
   (b) *Transitivity.* For all distinct i, j, and k such that $1 \leq i, j, k \leq m$, we add clause $(\neg b_{i,j} \vee \neg b_{j,k} \vee b_{i,k})$.
2. ***Soft* Minimum Swaps**. Here, each *unsatisfied* clause corresponds to a *swapped* pair of tuples in the database. For each pair of tuples $\mathsf{t}_r = (u, b_i)$ and $\mathsf{t}_s = (v, b_j)$ within the *same* partition group of $\mathbf{r}$ such that $u < v$, we add the clause $(b_{i,j})$.

Theorem 10. *The* A E/I OC *instance has a solution with at most* t *swaps if the MAX-SAT instance has a solution with* t unsatisfied *soft clauses, and vice versa.*

Therefore, the A E/I OC instance is valid *iff* there exists a solution to the MAX-SAT instance with at most s *unsatisfied* clauses. If the A E/I OC instance is valid, we can generate a *best* order solution $\mathsf{T}^{\prec}_{\mathsf{B}^*}$ in polynomial time using a solution of the MAX-SAT

instance that maximizes the number of satisfied soft clauses, or equivalently, minimizes the number of *unsatisfied* soft clauses. To do so, we add $b_i \prec b_j$ to $\mathsf{T}^{\prec}_{\mathsf{B}^*}$ iff variable $b_{i,j}$ is set to *true* in the MAX-SAT solution. To derive a *strongest* derivable order B^*, we can use the algorithm proposed in [15] with $\mathcal{O}(n \ln n)$ runtime to resolve all swaps with respect to the explicit order over A and $\mathsf{T}^{\prec}_{\mathsf{B}^*}$ by removing the minimum number of tuples from $\mathbf{r}$. Let $\mathbf{t}$ denote the repaired data. We construct B^* by including a relation $b_i \prec b_j$ from $\mathsf{T}^{\prec}_{\mathsf{B}^*}$ iff there exist tuples $\mathsf{t}_r = (u, b_i)$ and $\mathsf{t}_s = (v, b_j)$ within the *same* partition group of $\mathbf{t}$ such that $u < v$; intuitively, *pruning* $\mathsf{T}^{\prec}_{\mathsf{B}^*}$ to derive B^*.

Although the naive implementation of this reduction takes quadratic time in the number of tuples, it is possible to do it more efficiently. For a candidate $\mathcal{X}$: $\mathsf{A} \sim \mathsf{B}^*$, we create a matrix M storing the number of times a value b_i has occurred before value b_j with respect to the sorted partitions of τ_{A} within each partition group. For each sorted partition of τ_{A} in a partition group, we keep the number of occurrences of values of B with the previous sorted partitions. Using this, we can update the entries of M when traversing over each value of A in its sorted partition. For example, if in some partition group, b_j has occurred r times with a_v and b_i has occurred s times with values a_u such that $a_u < a_v$, we increase the entry $M[b_i, b_j]$ by $r \cdot s$. We use a reduction to *weighted* partial MAX-SAT, where the weight of a soft clause $(b_{i,j})$ is set to $M[b_i, b_j]$; i.e., based on how many times the value b_i has occurred before the value b_j. (A similar approach can be used to derive a strongest derivable order from the repaired data.) Let l denote the number of unique values of the side with an *explicit* domain order.

Lemma 7. *The reduction to MAX-SAT for* A $\mathsf{E/I}$ OC*s is* $\mathcal{O}(n \ln n + m^3 + plm^2)$.

6.3 Approximate I/I OCs

Finally, we offer a reduction from A $\mathsf{I/I}$ OCs to partial MAX-SAT. For this reduction, we use ideas from both the exact $\mathsf{I/I}$ OC reduction as well as the A $\mathsf{E/I}$ OC reduction. Namely, we use the same types of clauses as in the exact $\mathsf{I/I}$ OC reduction, while assigning *softness* or *hardness* to clauses similar to the A $\mathsf{E/I}$ OC reduction. Again, a *solution* is a binary assignment to the variables that satisfies all hard clauses.

Let $\mathbf{r}$ be a relational schema instance with n tuples and $\mathcal{X}$: $\mathsf{A}^* \sim \mathsf{B}^*$ be the A $\mathsf{I/I}$ OC instance with the swap threshold s. Let $a_1, a_2, \ldots, a_{m'}$ and $b_1, b_2, \ldots, b_{m''}$ be the values of A and B, respectively. Define two sets of SAT variables: $a_{i,j}$ for all distinct i and j such that $1 \leq i, j \leq m'$, and similarly $b_{i,j}$ for the values of B. We assume that in the MAX-SAT solution, having $a_{i,j} = \mathit{true}$ implies $a_i \prec a_j \in \mathsf{T}^{\prec}_{\mathsf{A}}$ and $b_{i,j} = \mathit{true}$ implies $b_i \prec b_j \in \mathsf{T}^{\prec}_{\mathsf{B}}$. The

two sets of *valid order* and *minimum swaps* clauses are generated as *soft* and *hard* clauses, respectively, as follows.

1. ***Hard* Valid Order.**
   (a) *Connexivity.* For all distinct i and j such that $1 \leq i, j \leq m'$, we add clauses $(a_{i,j} \vee a_{j,i})$ and $(\neg a_{i,j} \vee \neg a_{j,i})$.
   (b) *Transitivity.* For all distinct i, j, and k such that $1 \leq i, j, k \leq m'$, we add clause $(\neg l_{i,j} \vee \neg l_{j,k} \vee l_{i,k})$.

   Similar clauses are added for the values of B.

2. ***Soft* Minimum Swaps.** Here, each *pair* of clauses corresponds to a potential pair of *swapped* tuples in the database. For each two tuples $\mathtt{t}_r = (a_i, b_u)$ and $\mathtt{t}_s = (a_j, b_v)$ within the *same* partition group such that $a_i \neq a_j$ and $b_u \neq b_v$, we add clauses $(a_{i,j} \vee b_{v,u})$ and $(a_{j,i} \vee b_{u,v})$.

Theorem 11. *The* A I/I OC *instance has a solution with at most t swaps if the MAX-SAT instance has a solution with t* unsatisfied *soft clauses, and vice versa.*

Therefore, the A I/I OC instance is valid *iff* the MAX-SAT instance has a solution with at most s unsatisfied soft clauses. Assuming the A I/I OC instance is valid, a *best* pair of orders $\mathsf{T}^{\prec}_{\mathsf{A}^*}$ and $\mathsf{T}^{\prec}_{\mathsf{B}^*}$ can be derived from a solution to the MAX-SAT instance that satisfies the maximum number of soft clauses, as done for exact I/I OCs and A E/I OCs in previous sections. We can again use the algorithm from [15] to resolve all swaps with respect to $\mathsf{T}^{\prec}_{\mathsf{A}^*}$ and $\mathsf{T}^{\prec}_{\mathsf{B}^*}$ and create the repaired data $\mathbf{t}$. Next, the algorithms described in Section 6.1 are used to remove *unnecessary* relations $a_i \prec a_j$ from $\mathsf{T}^{\prec}_{\mathsf{A}^*}$, and similarly for $\mathsf{T}^{\prec}_{\mathsf{B}^*}$, to derive a pair of strongest derivable orders A^* and B^*.

Similar to the reduction for A E/I OCs in Section 6.2, considering pairs of *tuples* is not necessary for this reduction. When constructing the bipartite graphs BGs as in Section 4.2, we use weighted edges, where the weight of an edge (a_i, b_j) denotes the number of tuples s with $\mathsf{s}_\mathsf{A} = a_i$ and $\mathsf{s}_\mathsf{B} = b_j$ within that partition group. We can then consider pairs of edges in each BG, and for edges (a_i, b_u) and (a_j, b_v) with weights r and s, respectively, add *soft* clauses $(a_{i,j} \vee b_{v,u})$ and $(a_{j,i} \vee b_{u,v})$ to the *weighted* partial MAX-SAT instance, with weight $r \cdot s$ each.

Lemma 8. *The reduction to MAX-SAT for* A I/I OC*s is* $\mathcal{O}(n + pm^4)$.

Chapter 7

Measure of Interestingness

The search space and the number of discovered implicit domain orders may be large in practice. Inspired by previous work [24, 2], to decrease the cognitive burden of human verification, we propose a measure of *interestingness* to rank the discovered domain orders based on how close each is to being a strong total order. We argue that by focusing on similarity to a strong total order, this measure is successful in detecting meaningful and accurate implicit orders.

Given a DAG G representing a strong partial order, the *pairwise* interestingness measure is defined as $\mathit{pairwise}(G) = |\mathit{pairs}(G)|/\binom{m}{2}$, where $\mathit{pairs}(G) = \{(u, v) : u, v \in G \text{ and there is a path between } v \text{ and } u\}$, and m is the number of vertices in G. The number of pairs of vertices that are connected demonstrates the quality of the found strong partial orders, while the binomial coefficient in the denominator is added for normalization purposes over the possible pairs with respect to the number of unique values m. Based on this measure, a strong total order graph has the perfect score of 1, while a completely disconnected graph has a score of 0.

Example 27. *Consider the order graph G presented in Figure 3.1b. There are 23 pairs of connected vertices and $\binom{8}{2} = 28$ possible pairs. Thus, the pairwise score is* $\text{pairwise}(G) = \frac{23}{28} \approx 0.82$.

For conditional implicit orders, we divide the number of *pairs* in each partition group over the total number of pairs possible among *all* the values in the attribute, and then compute their average. This is to prevent candidates with many partition groups with less interesting partial orders from achieving a high score. To achieve a score of 1, the partial order in each partition group needs to be strong total order over *all* the values in the

attribute. Our algorithm for computing this measure may take quadratic time $\mathcal{O}(m^2)$ in the number of vertices in the graph, which corresponds to the number of unique elements in the attribute. This is not significant, in practice (Chapter 8).

Chapter 8

Experiments

We implemented our implicit domain order discovery algorithm, named iORDER, on top of a Java implementation of the set-based E/E OD discovery algorithm [24, 25]. Furthermore, we use Sat4j, which is an efficient SAT solver library [18], and set the approximation factor to 10% when discovering approximate orders. Our experiments were run on a machine with a Core i9 2.9 GHz CPU with 128 GB RAM. We use two integrated datasets from the Bureau of Transportation Statistics (BTS) and the North Carolina State Board of Elections (NCSBE):

- **Flight** contains information about flights in the US with 1M tuples and 35 attributes (https://www.bts.gov).
- **Voter** contains data about voters in the US with 1M tuples and 35 attributes (https://www.ncsbe.gov).

We chose these datasets due to their size for scalability experiments and for having real-life attributes with interesting implicit orders.

8.1 Scalability

Exp-1: Scalability in |r|. We measure the running time of iORDER by varying the number of tuples (Figure 8.1). We use the Flight and Voter datasets with 10 attributes and up to 1M tuples, by showing data samples to users and asking them to mark attributes as potential exact or approximate order candidates. In the absence of user annotation, the algorithm can be run multiple times over different subsets of attributes to capture potential implicit orders. Figure 8.1 shows the runtime of our framework when discovering exact or

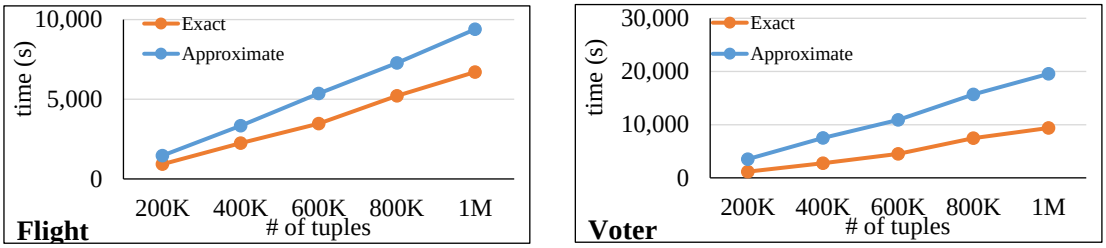


Figure 8.1: Scalability and effectiveness in $|\mathbf{r}|$.

approximate implicit orders. In both cases, the runtime has a linear growth in the number of tuples as computation is dominated by the linear steps of the discovery process and the non-linear factors in our algorithm tend to have less impact. Thus, iORDER scales well for large datasets. The runtime is on average twice as long when discovering approximate implicit orders. This makes intuitive sense as many of the linear steps are repeated for a candidate when discovering approximate implicit orders. We argue that this runtime overhead, which is not significant, is worth it as approximate implicit orders can cover interesting examples that are missed by the exact algorithm (as discussed in Exp-5).

Exp-2: Scalability in $|\mathbf{R}|$. Next, we vary the number of attributes. We use the Flight and Voter datasets with 1K tuples (to allow experiments with a large number of attributes in reasonable time) and up to 35 attributes. For the approximate candidates, we only check implicit orders for candidates that have at most 25 distinct values, as the runtime gets prohibitively expensive due to the large number of candidates in this experiment and the extra sensitivity of MAX-SAT reductions for approximate OCs to the number of distinct values. This can be ameliorated by running the algorithm on random smaller subsets of the attributes. Figure 8.2 illustrates that the running times of both exact and approximate discovery frameworks increase exponentially with the number of attributes (the Y-axis is in log scale). This is because the number of implicit order candidates is exponential in the worst case. The Voter dataset requires more time for the same number of attributes due to a larger number of candidates.

Exp-3: NP-complete Cases in Practice. The most general case of implicit domain order discovery through unconditional I/I OCs as well as the approximate cases are NP-complete (Sections 4.4, 5.2, and 5.4). However, the majority of observed cases took a short time. The cases reduced to SAT or MAX-SAT were on average solved in under 60 ms in Exp-1 and Exp-2, indicating that NP-complete cases are handled well in practice. In Exp-2, with varying the number of attributes and 1K tuples, on average, 33% of the total runtime was spent on reducing to, and solving, the SAT instances. However, in the corresponding Exp-1, with varying the number of tuples up to 1M tuples, this ratio was

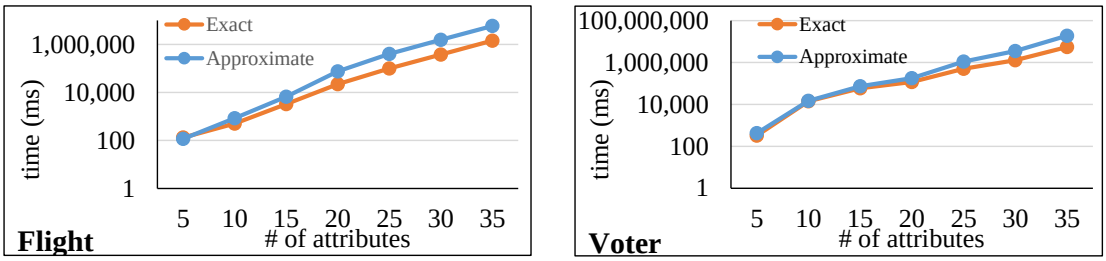


Figure 8.2: Scalability and effectiveness in $|\mathbf{R}|$.

less than 1%. This is attributed to the large number of candidates in Exp-2 and the small number of tuples, so the linear steps of the algorithm do not dominate the runtime.

8.2 Effectiveness

Exp-4: Effectiveness over lattice levels. Here, we measure the running time and the number of discovered implicit domain orders at different levels of the lattice (Figure 8.3). We report the results with 10 attributes over 1M tuples (from Exp-1 using the framework for exact order discovery) in the Flight and Voter datasets. Note that in this figure, the first level corresponds to the lattice level with attribute sets of size two, as no OCs exist in the previous lattice levels. Since the attribute lattice is diamond-shaped and nodes are pruned over time through axioms, the time to process each level first increases, up to level five, and decreases thereafter.

As most of the interesting implicit orders are found at the top levels with respect to a smaller context (as verified in Exp-5), we can prune the lower levels to reduce the total time. In the Flight and Voter datasets, approximately 85% and 73% of the orders are found in the first three levels, taking about 45% and 19% of the total time, respectively. In the Voter dataset, fewer implicit orders are found in the first levels of the lattice, creating fewer pruning opportunities. Therefore, more time is spent on validating candidates with larger contexts, explaining the runtime difference between the two datasets.

An interesting result when discovering approximate implicit orders is that the discovered orders are, on average, 0.3 levels higher on the lattice (with smaller contexts) compared to when discovering only exact implicit orders. This is because when discovering approximate orders, the algorithm is able to ignore some exceptions and discover more general implicit orders with smaller contexts. As shown in Exp-5, orders on upper levels of the lattice tend to be more interesting.

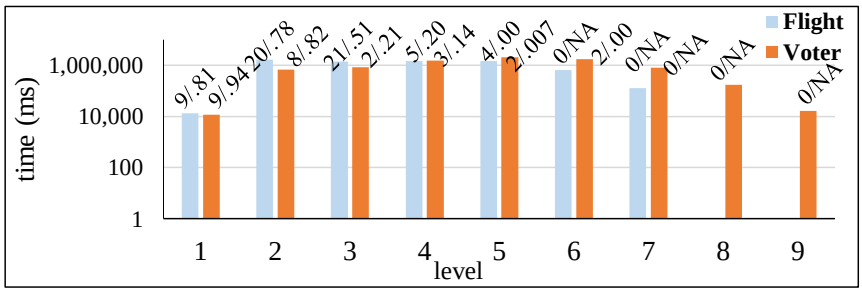


Figure 8.3: Runtime, number of orders, and average interestingness.

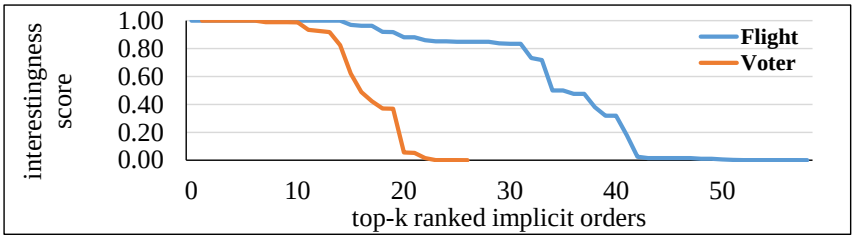


Figure 8.4: Interestingness scores of discovered implicit orders.

Exp-5: Interestingness of implicit orders. We argue that implicit domain orders found at upper levels of the lattice are the most interesting. Implicit orders found with respect to a context with more attributes contain more partition groups. Hence, an implicit order with respect to a less compact context may hold, but may not be as meaningful, due to overfitting. Figure 8.3 illustrates that the interestingness score drops from the fourth level on for the Flight dataset and from the third level on for the Voter dataset. Figure 8.4 illustrates that our interestingness measure can help reduce the number of implicit domain orders, by discarding the orders that achieve a very low score.

In the Flight dataset, we discover orders over monthGreg* and monthLunar* through an unconditional I/I OC with respect to the context of yearLunar (as the attribute yearGreg has a fixed value), as well as the A I/I OC monthGreg* $\sim$ monthLunar*, since the exceptions in boundary months, i.e., December and January, can be ignored in the approximate case. This is valuable in some instances, for example, when the year attributes are not provided. We also found a high-scoring order delayDesc*, which orders flight delay as *Early* $\prec$ *On-time* $\prec$ *Short delay* $\prec$ *Long delay*, and is discovered through the E/I OD delay $\sim$ delayDesc*. This order is interesting since long delays may result in fines on the airline, so detecting these instances is valuable. Implicit orders over distanceRangeMile and distanceRangeKM were discovered through the I/I OC distanceRangeMile* $\sim$ distanceRangeKM*, as the categories

for these attributes are overlapping (e.g., ranges $0-700$ and $700-3000$ miles overlap with the range of $1100-4800$ kilometers). Another ordered attribute in this dataset is flightLength (*Short-haul* $\prec$ *Medium-haul* $\prec$ *Long-haul*), which was discovered through the E/I OD {airline}: flightDuration $\sim$ flightLength*. The non-empty context is due to different airlines using different ranges to define flight duration.

In the Voter dataset, the attributes ageRange ($12-17 \prec 18-24 \prec \ldots \prec +75$) and generation (e.g., *Baby Boomer* $\prec$ *Generation X* $\prec$ *Millennial* $\prec$ *Generation Z*) are discovered through E/I ODs age $\sim$ ageRange* and A E/I OD birthYear $\sim$ generation*, respectively, as well as the A I/I OC ageRange* $\sim$ generation* as the categories are overlapping. The reason for discovering these orders through approximate cases is different age ranges being used to define generations, e.g., 1981–1996 or 1980–1994 for "Millanial". Finally, an order over birthYearAbbr was detected using the *conditional* E/I OD {isCentenarian}: birthYear $\sim$ birthYearAbbr*. This is because for the people born in the early 1917 or before with isCentenarian = True we have $'97 \prec '98 \prec \ldots \prec '16 \prec '17$, while for people born after this date with isCentenarian = False we have $'17 \prec '18 \prec \ldots \prec '98 \prec '99$. Thus an unconditional OD is not possible. However, when discovering *approximate* implicit orders, we discover the *unconditional* A E/I OD birthYear $\sim$ birthYearAbbr*. This is because approximate implicit orders can ignore the exceptional birth years, i.e., those before 1900, and discover a more general implicit order.

Exp-6: Time to compute interestingness. We consider the effect of computing the pairwise measure of interestingness on the algorithm runtime. We observed that in Exp-1 in the number of tuples, the runtime increase is less than 1%. In Exp-2 in the number of attributes, this increase is at most around 10%, which is attributed to the higher ratio of unique values to the number of tuples.

8.3 Applications

Exp-7: Data Profiling. Table 8.1 illustrates that iORDER finds implicit ODs in both Flight and Voter with 10 attributes and 1M tuples. For each dataset, the first row represents the number of implicit orders when using the exact implicit order discovery framework and the second row when using the approximate framework. The second number in the round brackets represents the number of various types of discovered ODs. As can be seen, we can discover more than double the number dependencies when using our implicit order discovery framework. To investigate the importance of different types of implicit OCs, we categorize the number of top-k exact orders found in the Voter dataset using each type of implicit OC in Table 8.2. It can be seen that all types of implicit OCs contribute to the

Table 8.1: The number of **implicit orders** (data dependencies) when using 1M tuples and 10 attributes.

dataset	FD	OC	E/I				I/I OC	
			cond.		**uncond.**		**cond.**	**uncond.**
			OD	OC	OD	OC		
Flight (Exact)	**0** (20)	**0** (12)	**4** (4)	**30** (30)	**3** (3)	**6** (6)	**0** (0)	**16** (8)
Flight (Approx.)	**0** (20)	**0** (7)	**2** (2)	**14** (14)	**6** (6)	**22** (22)	**8** (4)	**2** (1)
Voter (Exact)	**0** (5)	**0** (1)	**1** (1)	**10** (10)	**1** (1)	**6** (6)	**6** (3)	**2** (1)
Voter (Approx.)	**0** (5)	**0** (1)	**1** (1)	**20** (20)	**1** (1)	**9** (9)	**2** (1)	**4** (2)

Table 8.2: Types of top-k orders.

dataset	**k**	**cond.** E/I OC/OD	**ucond.** E/I OC/OD	**cond.** I/I OC	**ucond.** I/I OC
Voter	**5**	0	4/1	0	0
	10	2/1	4/1	1	1
	15	3/1	6/1	2	2

most interesting orders found. A similar pattern was observed in the Flight dataset. When discovering approximate orders, we can discover on average 20% more implicit orders in the data. In many cases, discovering approximate orders allows us to discover implicit orders through more general types of dependencies; e.g., *unconditional* dependencies instead of *conditional* ones, since we are able to ignore some exceptions in the data.

Exp-8: Knowledge Base Enhancement. As another application of implicit domain orders in data profiling, we now compare with an open-source manually curated knowledge-base: YAGO. We quantify the percentage of automatically discovered implicit domain orders by our algorithm among the top-5 (ranked by our pairwise measure of interestingness) that exist in YAGO. The existence of an implicit order in YAGO is evaluated by considering pairs of values within the ordered domain and verifying if there exist knowledge triples specifying the relationship between the two entities. The result is that only 20% of the top discovered orders exist in YAGO. Thus, existing knowledge bases can be enhanced by our techniques, especially in instances where the discovered orders are domain-specific or in knowledge bases that focus on objects rather than concepts, where implicit orders are more common. This may be done by incorporating pairs of ordered values as knowledge triples, e.g., (*Corner*, *Less than*, *Peach*).

Exp-9: Data Mining. We next evaluate the impact of implicit orders on the task of data summarization in data mining. We use the techniques described in [7] and [32],

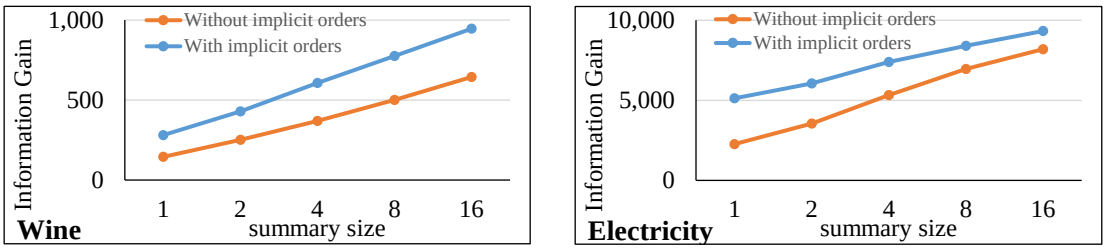


Figure 8.5: Information gain with and without implicit orders.

which aim to summarize a database in k rows, each representing a subset of the data, by maximizing the *information gain* of the summary. We consider two scenarios: one with all the attributes treated as categorical, and one with implicitly ordered attributes treated as ordinal, allowing the summary to refer to value ranges of these attributes rather than individual values. We compare the information content of summaries of two datasets, Wine and Electricity, which are available in the UCI (https://archive.ics.uci.edu/) and openML (https://www.openml.org/) repositories, respectively. These datasets contain implicitly ordered attributes such as dayOfWeek. To create more implicit orders for this experiment, we converted some numeric attributes into implicitly-ordered ordinal ones (e.g., representing price as $cheap \prec moderate \prec expensive$).

There are two parameters in the summarization approach from [32]: a sample size from which to mine summary patterns, and the number of top patterns to explore in each iteration of summary construction. We use default values of these parameters: 4 and 16, respectively, and experiment with values of k (summary size) in $\{1, 2, 4, 8, 16\}$. As reported in Figure 8.5, utilizing implicit orders allows on average 60% more information gain using summaries of the same size (65% and 54% for Wine and Electricity datasets, respectively). The larger improvement on the Wine dataset could be due to having more attributes with implicit orders in this dataset.

Chapter 9

Related Work

Previous work investigated the properties of and relationships between sorted sets [5]. However, to the best of our knowledge, besides [20], which focuses on extending OCs to attributes with implicit orders, no algorithms for discovering implicit domain orders exist.

Existing OD discovery algorithms require some notion of explicit order [25, 24, 17, 14] and can benefit from implicit orders to find "hidden" ODs that have not been feasible before. In our solution, we use the set-based OD discovery algorithm [24, 25] since other approaches cannot discover a complete set of non-trivial ODs. For example, the list-based approach in [17] is intentionally incomplete in order to prune the much larger list-based search space. A similar approach, recently shown in [14] is also incomplete despite the authors' claim of completeness: it omits ODs in which the same attributes are repeated in the left- and the right-hand side, such as [country, profit] $\mapsto$ [country, tax] and reports an OD only when both the corresponding OFD and OC hold. Thus, it leaves out cases when only an OFD or only an OC is true (e.g, OC week $\sim$ month holds, but OFDs {week}: [] $\mapsto$ month and {month}: [] $\mapsto$ week do not hold over the tuples within a *single* year). Additionally, the algorithm recently presented in [4] is incomplete, as shown in [26].

The importance of sorted sets has been recognized for query optimization and data cleaning. In [11], the authors explored sorted sets for executing nested queries. Sorted sets created as generated columns (SQL functions and algebraic expressions) were used in predicates for query optimization [19, 29]. Relationships between sorted attributes have been also used to eliminate joins [28] and to generate interesting orders [23, 30]. A practical application of sorted sets to reduce the indexing space was presented in [6]. So far, the focus of this line of work has been on explicit orders, while these applications can also benefit from implicit orders found using our algorithm.

Chapter 10

Conclusions

We devised the first techniques to discover implicit domain orders. We factored the problem space across multiple dimensions and presented definitions, hardness results, and algorithms for each of these cases. Finally, we demonstrated the scalability of our algorithm, its effectiveness in discovering interesting orders, and the applicability of the discovered orders in different domains. While in this work, we discover implicit domain orders with respect to a single set-based OD, we plan to extend our framework to merge orders found for a given attribute with multiple set-based ODs. We will also address implicit order discovery in dynamic tables, as was recently done for explicit OD discovery [31]. We are also interested in studying the foundations of E/I and I/I ODs / OCs, including *inference*.

References

[1] S. Abiteboul, R. Hull, and V. Vianu. *Foundations of Databases: The Logical Level*. Addison-Wesley Longman Publishing Co., Inc., USA, 1st edition, 1995.

[2] F. Chiang and R. J. Miller. Discovering data quality rules. *PVLDB*, 1(1):1166–1177, Aug. 2008.

[3] E. F. Codd. Relational completeness of data base sublanguages. *Research Report / RJ / IBM / San Jose, California*, RJ987, 1972.

[4] C. Consonni, P. Sottovia, A. Montresor, and Y. Velegrakis. Discovering order dependencies through order compatibility. In *EDBT*, pages 409–420, 2019.

[5] B. Davey and H. Priestley. *Introduction to lattices and order*. Cambridge University Press; 2nd edition, 2002.

[6] J. Dong and R. Hull. Applying approximate order dependency to reduce indexing space. In *SIGMOD*, pages 119–127, 1982.

[7] K. El Gebaly, P. Agrawal, L. Golab, F. Korn, and D. Srivastava. Interpretable and informative explanations of outcomes. *PVLDB*, 8(1):61–72, Sept. 2014.

[8] M. R. Garey and D. S. Johnson. *Computers and intractability; A guide to the theory of NP-completeness*. W. H. Freeman & Co., 1990.

[9] S. Ginsburg and R. Hull. Order dependency in the relational model. *Theoretical Computer Science*, 26(1):149 – 195, 1983.

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

[11] R. Guravannavar, H. S. Ramanujam, and S. Sudarshan. Optimizing nested queries with parameter sort orders. In *Proceedings of the 31st International Conference on Very Large Data Bases*, VLDB '05, page 481–492. VLDB Endowment, 2005.

[12] F. Harary. *Graph Theory*. Addison Wesley series in mathematics. Addison-Wesley, 1971.

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

[14] Y. Jinand, L. Zhu, and Z. Tan. Efficient bidirectional order dependency discovery. In *ICDE*, pages 61–72, 2020.

[15] R. Karegar, P. Godfrey, L. Golab, M. Kargar, D. Srivastava, and J. Szlichta. Efficient discovery of approximate order dependencies. In *EDBT*, 2021. to appear.

[16] J. Kivinen and H. Mannila. Approximate inference of functional dependencies from relations. *Theoretical Computer Science*, 149(1):129–149, 1995. Fourth International Conference on Database Theory (ICDT '92).

[17] P. Langer and F. Naumann. Efficient order dependency detection. *VLDB J.*, 25(2):223–241, 2016.

[18] D. Le Berre and A. Parrain. The sat4j library, release 2.2. *Journal on Satisfiability, Boolean Modeling and Computation*, 7(2-3):59–64, 2010.

[19] T. Malkemus, S. Padmanabhan, B. Bhattacharjee, and L. Cranston. Predicate derivation and monotonicity detection in DB2 UDB. In *ICDE*, pages 939–947, 2005.

[20] M. Mirsafian. Semantic order compatibilities and their discovery. *UWSpace*, 2019.

[21] W. Ng. Lexicographically ordered functional dependencies and their application to temporal relations. In *Proceedings. IDEAS'99. International Database Engineering and Applications Symposium (Cat. No.PR00265)*, pages 279–287, 1999.

[22] W. Ng. An extension of the relational data model to incorporate ordered domains. *ACM TODS*, 26(3):344–383, 2001.

[23] D. Simmen, E. Shekita, and T. Malkemus. Fundamental techniques for order optimization. In *SIGMOD*, pages 57–67, 1996.

[24] J. Szlichta, P. Godfrey, L. Golab, M. Kargar, and D. Srivastava. Effective and complete discovery of order dependencies via set-based axiomatization. *PVLDB*, 10(7):721–732, 2017.

[25] J. Szlichta, P. Godfrey, L. Golab, M. Kargar, and D. Srivastava. Effective and complete discovery of bidirectional order dependencies via set-based axioms. *VLDB J.*, 27(4):573–591, 2018.

[26] J. Szlichta, P. Godfrey, L. Golab, M. Kargar, and D. Srivastava. Erratum for discovering order dependencies through order compatibility. In *EDBT*, pages 659–663, 2020.

[27] J. Szlichta, P. Godfrey, and J. Gryz. Fundamentals of order dependencies. *PVLDB, 5(11): 1220-1231*, 2012.

[28] J. Szlichta, P. Godfrey, J. Gryz, W. Ma, P. Pawluk, and C. Zuzarte. Queries on dates: fast yet not blind. In *EDBT*, pages 497–502, 2011.

[29] J. Szlichta, P. Godfrey, J. Gryz, W. Ma, W. Qiu, and C. Zuzarte. Business-intelligence queries with order dependencies in DB2. In *EDBT*, pages 750–761, 2014.

[30] J. Szlichta, P. Godfrey, J. Gryz, and C. Zuzarte. Expressiveness and complexity of order dependencies. *PVLDB*, 6(14):1858–1869, Sept. 2013.

[31] Z. Tan, A. Ran, S. Ma, and S. Qin. Fast incremental discovery of pointwise order dependencies. *PVLDB*, 13(10):1669–1681, June 2020.

[32] M. Vollmer, L. Golab, K. Böhm, and D. Srivastava. Informative summarization of numeric data. In *Proceedings of the 31st International Conference on Scientific and Statistical Database Management*, SSDBM '19, page 97–108, New York, NY, USA, 2019. Association for Computing Machinery.

[33] C. Wyss, C. Giannella, and E. Robertson. Fastfds: A heuristic-driven, depth-first algorithm for mining functional dependencies from relation instances extended abstract. In Y. Kambayashi, W. Winiwarter, and M. Arikawa, editors, *Data Warehousing and Knowledge Discovery*, pages 101–110, Berlin, Heidelberg, 2001. Springer Berlin Heidelberg.

APPENDICES

Appendix A

Proofs

Proof of Theorem 1
Assume $(\pi_{\mathsf{B}})_{\mathsf{A}}$ is an interval partitioning and, without loss of generality, let $a_1, \ldots, a_{p_q}$ be the explicit order over values in A and the value of tuples t_1 to t_{p_q} projected on columns A and B be the following: (a_1, b_1), (a_2, b_1), $\ldots$, (a_{p_1}, b_1), (a_{p_1+1}, b_2), $\ldots$, (a_{p_2}, b_2), $\ldots$, $(a_{p_{q-1}+1}, b_q)$, $\ldots$, (a_{p_q}, b_q) (tuples with duplicate values are represented as one tuple for simplicity). Therefore, $\mathcal{E}_j(\mathsf{t}_{\mathsf{B}})_{\mathsf{A}} = \{a_{p_{j-1}+1}, \ldots, a_{p_j}\}$ (assuming $p_0 = 0$). The total order $\mathsf{T}_{\mathbf{r}}^{\prec}$, $\mathsf{t}_1 \prec \ldots \prec \mathsf{t}_{p_q}$ is a valid *witness* and its projection on B results in total order $b_1 \prec \ldots \prec b_q$. Since this is the only valid order (as it is enforced by the explicit order over the values of A), $\mathsf{T}_{\mathsf{B}_*}^{\prec}$ is the only valid—and therefore the strongest derivable—implicit order over the values of B.

If $(\pi_{\mathsf{B}})_{\mathsf{A}}$ is not an interval partitioning, then there exist $i, j \in [1, \ldots, k]$ such that $i < j$ and $\min(\mathcal{E}_i(\mathsf{t}_{\mathsf{B}})_{\mathsf{A}}) \prec \min(\mathcal{E}_j(\mathsf{t}_{\mathsf{B}})_{\mathsf{A}}) \prec \max(\mathcal{E}_i(\mathsf{t}_{\mathsf{B}})_{\mathsf{A}})$. Let $\mathsf{t}_{i_1} = (\min(\mathcal{E}_i(\mathsf{t}_{\mathsf{B}})_{\mathsf{A}}), b_i)$, $\mathsf{t}_{i_2} = (\max(\mathcal{E}_i(\mathsf{t}_{\mathsf{B}})_{\mathsf{A}}), b_i)$, and $\mathsf{t}_j = (\min(\mathcal{E}_j(\mathsf{t}_{\mathsf{B}})_{\mathsf{A}}), b_j)$. If this $\mathsf{E/I\ OD}$ candidate is valid, then a total order $\mathsf{T}_{\mathbf{r}}^{\prec}$ must exist that is: 1) compatible with the explicit order over the values of A and 2) its projection produces a valid total order over the values of B. Given the first condition, in any total order $\mathsf{T}_{\mathbf{r}}^{\prec}$, there must be $\mathsf{t}_{i_1} \prec \mathsf{t}_j \prec \mathsf{t}_{i_2}$, while the projection of this order would produce a relation over B with $b_i \prec b_j$ and $b_j \prec b_i$, which cannot be a valid total order. Therefore, by contradiction, this $\mathsf{E/I\ OC}$ candidate is not valid. □

Proof of Lemma 1
Let the attributes involved in the $\mathsf{E/I\ OD}$ candidate with an empty context be A and B, and assume the FD $\mathsf{A} \rightarrow \mathsf{B}$ holds. There are three cases (as enumerated in Section 3.2) that depend on which side has the implicit order and whether the FD $\mathsf{B} \rightarrow \mathsf{A}$ holds. In

all cases, we only need to sort m values (the number of distinct values of B), for a cost of $m \ln m$. The other steps can be done in linear time using a hash table. Thus, the total runtime is $\mathcal{O}(m \ln m + n)$. □

Proof of Theorem 2
Assume $\mathsf{B}^{\tau_\mathsf{A}}$ is a valid weak total order and let $\mathsf{T}^{\prec}_{\mathsf{B}^*}$ denote an arbitrary strong total order compatible with—that is, is a superset of—$\mathsf{B}^{\tau_\mathsf{A}}$. To find a *witness* order $\mathsf{T}^{\prec}_{\mathbf{r}}$ over the *tuple*s of the database, sort the tuples based on the explicit order over A and break ties by $\mathsf{T}^{\prec}_{\mathsf{B}^*}$. Considering the projected order of this witness over B, the $b_i \prec b_j$ derived *within* a partition group of A cannot result in a cycle, as they were initially enforced by $\mathsf{T}^{\prec}_{\mathsf{B}^*}$. For the $b_i \prec b_j$ derived from *different* partition groups of A, it suffices to verify *consecutive* partition groups of A due to the *transitivity* of the order relation. Assume the relation $b_i \prec b_j$ can be derived from two tuples in different partition groups PG_p and PG_q where, without loss of generality, $p < q$. If $q = p + 1$, these partition groups are consecutive and $b_i \prec b_j$ is derived using our algorithm. Otherwise, the partition group $\mathsf{PG}_{p+1} \neq \mathsf{PG}_q$ contains some tuple with the B-value of b_k (note that b_k could be the same as b_i or b_j) since otherwise PG_p and PG_q would have been consecutive. Using our algorithm, the order $b_i \prec b_k$ is derived (if $b_i \neq b_k$). Due to transitivity, it is now only required for $b_k \prec b_j$ to be derived (if $b_j \neq b_k$), which is inductively derived from PG_{p+1} and PG_q. Therefore, the validity of $\mathsf{B}^{\tau_\mathsf{A}}$ implies the validity of the witness order over tuples.

Now we prove the other direction. Without loss of generality, let PG_i denote the i-th sorted partition group of A and $\mathsf{B_i}$ the set of values of B that co-occur with PG_i. Assume $\mathsf{A} \sim \mathsf{B}^*$ is valid and that $\mathsf{T}^{\prec}_{\mathbf{r}}$ is some witness order over the tuples. Based on the definition of a witness order, the partition groups of A and B need to be placed consecutively (and in ascending order for A) in $\mathsf{T}^{\prec}_{\mathbf{r}}$. Since the projected order of $\mathsf{T}^{\prec}_{\mathbf{r}}$ over B is a valid total order, for any two consecutive PG_i and PG_{i+1}, $\mathsf{B_i}$ and $\mathsf{B_{i+1}}$ have at most one value in common. For each three consecutive partition groups PG_{i-1}, PG_i, and PG_{i+1}, let $\mathsf{B_{i-1,i}} = \mathsf{B_{i-1}} \cap \mathsf{B_i}$, $\mathsf{B_{i,i+1}} = \mathsf{B_i} \cap \mathsf{B_{i+1}}$, and $\mathsf{B'_i} = \mathsf{B_i} \setminus \mathsf{B_{i-1}} \setminus \mathsf{B_{i+1}}$. Since $\mathsf{T}^{\prec}_{\mathbf{r}}$ is a valid order, for any two $i \neq j$, $\mathsf{B'_i} \cap \mathsf{B'_j} = \emptyset$. Therefore, by deriving pairs of ordered values from consecutive partition groups of A, the resulting order graph corresponds to a valid weak total order where each partition over the values of B correspond to the value(s) within a set $\mathsf{B'_i}$ or $\mathsf{B_{i,i+1}}$ for some i (in cases where $\mathsf{B'_i} = \emptyset$ and $\mathsf{B_{i-1,i}} = \mathsf{B_{i,i+1}}$, consecutive partitions with the same values can be considered as one). □

Proof of Lemma 2
Consider the E/I OC candidate $\mathsf{A} \sim \mathsf{B}^*$. Since the attributes in the OD-discovery algorithm in Chapter 2 have been sorted in advance for the first level of the lattice, the sorted partition

groups over the values of A can be created in $\mathcal{O}(n)$ time. A similar argument about the *3-fan-out* rule for I/I OCs in Section 4.2 also applies to E/I OCs with an empty context as E/I OCs are more restrictive than I/I OCs. This ensures that when traversing consecutive partition groups of A and inferring relations of the form $b_i \prec b_j$, the number of these relations will be bounded by m^2. Since the graph storing the partial order over the values of B will at most have size $\mathcal{O}(m^2)$ as well, the total runtime for an E/I OC candidate with an empty context is $\mathcal{O}(n + m^2)$. □

Proof of Theorem 3
Let G_i denote the set of strongest derivable orders from within each partition group, and let G be the *union graph* generated from these graphs using the procedure described in Section 3.4. Each G_i corresponds to a valid partial order over the values of the attribute with the implicit order. First, assume G corresponds to a valid partial order; i.e., it is acyclic. The *witness* order over the tuples within each partition group represents a valid witness order over the entire dataset, since the union of all the derived relations over the values corresponds to a valid partial order.

Assume that G is cyclic and, without loss of generality, let $\{(b_1, b_2), (b_2, b_3), \ldots, (b_{p-1}, b_p), (b_p, b_1)\}$ denote the set of edges involved in a cycle in this graph. Clearly, each edge (b_j, b_k) in this cycle must exist within some graph G_i. Since the order graphs G_i correspond to the *strongest* derivable order over each partition group, an edge $(b_j, b_k) \in G_i$ must be present in *all* witness total orders of the corresponding partition group. Therefore, for any witness order over the partition groups, and consequently over the entire dataset, $b_j \prec b_k$ for any b_j and b_k will be derived, meaning that the partial order derived over the entire dataset cannot be valid. Thus, the unconditional E/I OC candidate with a non-empty context is invalid. □

Proof of Lemma 3
Given an E/I OC candidate with a non-empty context $\mathcal{X}$, the algorithms in Sections 3.2 or 3.3 run for each partition group over $\mathcal{X}$. The runtime of discovering an implicit order over each partition group is $\mathcal{O}(k \ln k + m^2)$, where k denotes the number of tuples in the partition group. Furthermore, the size of the order graph constructed from each partition group is $\mathcal{O}(m^2)$. Since creating the union graph using the individual orders and checking for cycles can be done in linear time in size of the graphs, the total runtime is $\mathcal{O}(pk \ln k + pm^2)$ which is bounded by $\mathcal{O}(n \ln n + pm^2)$. □

Proof of Theorem 4
Assume that $\mathsf{BG}'_{\mathsf{A,B}}$ contains a 3-fan-out. Without loss of generality, let the tuples participating in this 3-fan-out be as follows (the minimal required tuples for a node a_4 containing

the 3-fan-out): $\mathsf{t}_1 = (a_1, b_1)$, $\mathsf{t}_2 = (a_2, b_2)$, $\mathsf{t}_3 = (a_3, b_3)$, $\mathsf{t}_4 = (a_4, b_1)$, $\mathsf{t}_5 = (a_4, b_2)$, and $\mathsf{t}_6 = (a_4, b_3)$. Assume that a *witness* total order $\mathsf{T}^{\prec}_{\mathbf{r}}$ exists over the tuples, and, without loss of generality, $\mathsf{t}_4 \prec \mathsf{t}_5 \prec \mathsf{t}_6$. Since the projections of $\mathsf{T}^{\prec}_{\mathbf{r}}$ over both A and B need to result in a valid total order, it must be that $\mathsf{t}_1 \prec \mathsf{t}_4$ and $\mathsf{t}_6 \prec \mathsf{t}_3$. However, it is impossible to find a valid placement for t_2, as any placement results in a cycle in the projected order over A or B. Therefore, by contradiction, the candidate is invalid.

Assume $\mathsf{BG}'_{\mathsf{A},\mathsf{B}}$ contains a cycle. Let a cycle with length k be over tuples $\mathsf{t}_1 = (a_1, b_1)$, $\mathsf{t}_2 = (a_2, b_1)$, $\ldots$, $\mathsf{t}_{k-1} = (a_{k/2}, b_{k/2})$, and $\mathsf{t}_k = (a_1, b_{k/2})$, without loss of generality. Assume that a *witness* total order $\mathsf{T}^{\prec}_{\mathbf{r}}$ exists over the tuples, and without loss of generality, $\mathsf{t}_1 \prec \mathsf{t}_2$. Then $\mathsf{t}_2 \prec \mathsf{t}_3$, and, inductively, $\mathsf{t}_i \prec \mathsf{t}_{i+1}$ must hold. Thus, $\mathsf{t}_1 \prec \mathsf{t}_2 \prec \ldots \prec \mathsf{t}_k$. However, this results in $a_1 \prec a_{k/2}$ and $a_{k/2} \prec a_1$, which makes a valid order over A impossible. Therefore, by contradiction, the candidate is invalid.

The other direction follows the correctness argument for the algorithm used to derive the orders using an I/I OC candidate; i.e., by choosing an arbitrary node with degree one in BG', and *zig-zagging* to an unvisited node in each step and adding it to the inferred order based on the side it belongs to. Let $\mathsf{T}^{\prec}_{\mathsf{A}'*}$ and $\mathsf{T}^{\prec}_{\mathsf{B}'*}$ denote the orders derived using this algorithm (which are unique modulo polarity). Without loss of generality, let $a_1 \prec a_2 \prec \cdots \prec a_k$ and $b_1 \prec b_2 \prec \cdots \prec b_k$ denote these total orders (the case where the length of one of the chains is longer by *one* value is resolved similarly). Without loss of generality, let a_1 denote the first node in $\mathsf{BG}'_{\mathsf{A},\mathsf{B}}$ with degree 1. This implies that the tuples need to be of either of two patterns: 1) (a_1, b_1); or 2) (a_i, b_{i-1}) and (a_i, b_i) for $i > 1$. The total order $\mathsf{T}^{\prec}_{\mathbf{r}'}$ derived by sorting the tuples by $\mathsf{T}^{\prec}_{\mathsf{A}'*}$ and breaking ties by $\mathsf{T}^{\prec}_{\mathsf{B}'*}$ results in a valid *witness* order. This is because any cycles in the derived orders over A' or B' would indicate the existence of a cycle (e.g., some a_i occurring with b_{i-2}) or a 3-fan-out (e.g., some a_i occurring with three values b_{i-1}, b_i, and b_{i+1}). For the *singleton* values removed in A' and B', they can easily be added to $\mathsf{T}^{\prec}_{\mathbf{r}'}$, where a tuple (a_i, b_k) can be added between the tuples (a_i, b_{i-1}) and (a_i, b_i), resulting in the valid final witness order $\mathsf{T}^{\prec}_{\mathbf{r}}$, implying the validity of this I/I OC candidate. □

Proof of Lemma 4

To validate an I/I OC candidate with an empty context involves generating the BG, iterating over the tuples once, and then using DFS traversal to check for cycles and 3-fan-outs. This can be done in linear time in the number of tuples. Note that if the 3-fan-out condition holds, the size of BG is linear in the number of distinct values (m). Thus, deriving an order can be done in $\mathcal{O}(m)$ time, as it only requires traversing the bipartite graph a constant number of times. For non-empty contexts, validation of I/I OCs requires the above steps for each partition group. Thus, the overall runtime remains $\mathcal{O}(n)$. □

Proof of Theorem 5
Since the I/I OD candidate is considered conditionally, the validity of the candidates and the *strongest derivable* orders over A^* and B^* are derived independently from within each partition group. Consider an arbitrary partition group and, since $\mathcal{X}\mathsf{A} \rightarrow \mathsf{B}$, without loss of generality, let the value of tuples t_1 to t_{p_q} in the columns A and B be the following: (a_1, b_1), (a_2, b_1), ..., (a_{p_1}, b_1), (a_{p_1+1}, b_2), ..., (a_{p_2}, b_2), ..., $(a_{p_{q-1}+1}, b_q)$, ..., (a_{p_q}, b_q) (tuples with duplicate values are represented as one tuple for simplicity). The total order $\mathsf{T}_\mathbf{r}^{\prec}$ $\mathsf{t}_1 \prec \ldots \prec \mathsf{t}_{p_q}$ over the tuples is a valid *witness* and its projection would result in total orders $a_1 \prec \ldots \prec a_{p_1}$ and $b_1 \prec \ldots \prec b_q$ over the values of A and B, respectively. The same argument can be applied to the rest of the partition groups, meaning that a *conditional* E/I OD candidate is always valid.

As for the *strongest derivable* orders over A^* and B^*, without loss of generality, let the initial witness order be $\mathsf{T}_\mathbf{r}^{\prec}$ and consider *transposition*s of either of these types applied on it: 1) relocating the tuple t_i, $p_{j-1}+1 \leq i \leq p_j$, within the range $[p_{j-1}+1, p_j]$; 2) relocating the consecutive set of tuples $\mathsf{t}_{p_{j-1}+1}$, ..., t_{p_j} to any location either between t_{p_k} and t_{p_k+1}, or the beginning or end of $\mathsf{T}_\mathbf{r}^{\prec}$. Both transpositions are *valid* since the resulting projected orders over the values of A and B are valid. The intersection of all such transposed witness orders for each side is the empty partial order. Therefore, the strongest derivable orders over A and B for this partition group (and, subsequently, for the other partition groups) are empty partial orders. □

Proof of Theorem 6
Validating I/I ODs is in class NP as a positive answer can be verified in polynomial time using a valid *witness* order $\mathsf{T}_\mathbf{r}^{\prec}$, as defined in Chapter 4. We now show that the CPP instance admits a valid polarization *iff* the I/I OD instance is valid. Since the reduction takes polynomial time in the size of the CPP instance, this proves that validating I/I ODs is NP-complete.

Assume the unconditional I/I OC candidate is valid and let $\mathsf{T}_\mathbf{r}^{\prec}$ denote a witness order. For any list L_i in the CPP instance, keep L_i's initial polarity if $\mathsf{t}_{i,1} \prec \mathsf{t}'_{i,1} \in \mathsf{T}_\mathbf{r}^{\prec}$ and reverse its polarity otherwise (i.e., if $\mathsf{t}'_{i,1} \prec \mathsf{t}_{i,1} \in \mathsf{T}_\mathbf{r}^{\prec}$). Note that the B-values in all partition groups of $c_{i,j}$ for a fixed i are b_i and b'_i, and therefore, whether $b_i \prec b'_i \in \mathsf{T}_\mathsf{B}^{\prec}$ or $b'_i \prec b_i \in \mathsf{T}_\mathsf{B}^{\prec}$ uniquely determines the polarity of the list L_i. Since $\mathsf{T}_\mathbf{r}^{\prec}$ is a valid witness order and its projected order over the values of A is a strong partial order as well, the corresponding polarization of the CPP instance also admits a strong partial order. Therefore, the CPP instance is valid.

In the other direction, let some polarization of the CPP instance exist which admits a strong partial order. To construct a valid witness order $\mathsf{T}_\mathbf{r}^{\prec}$ for the I/I OD candidate, the

tuples in each partition group $c_{i,j}$ are added to the witness order as $\mathsf{t}_{i,j} \prec \mathsf{t}'_{i,j}$ if list L_i is not reversed in the CPP instance, and as $\mathsf{t}'_{i,j} \prec \mathsf{t}_{i,j}$ otherwise. Since the polarization of the CPP instance admits a strong partial order, the projected order of $\mathsf{T}_{\mathbf{r}}^{\prec}$ over A will also correspond to a strong partial order. Furthermore, the projected order over B will too be a strong partial order since the order between each two values of $b_{i,j}$ and $b'_{i,j}$ is consistent across all partition groups (as it directly corresponds to the polarity of list L_i). Therefore, $\mathsf{T}_{\mathbf{r}}^{\prec}$ is a valid witness and the corresponding unconditional $\mathsf{I/I\ OD}$ candidate is valid. □

Proof of Theorem 7

Lemma 9. *For* any *total order* $\mathsf{T}_{\mathsf{B}}^{\prec}$ *over the values of* B*, the number of swaps between* non-adjacent *tuples is equal to* $C/2$*.*

Proof
Remember that C denotes the number of pairs of *non-adjacent* tuple that have different A- and B-values. We show a one-to-one correspondence between pairs of non-adjacent tuples, such that exactly *one* of the pairs of corresponding tuples are swapped, regardless of the order over the values of B.

Consider two non-adjacent tuples t_i and t_j such that $i < j$ and let $\mathsf{T}_{\mathsf{B}}^{\prec}$ be the total order over the values of B. Let $i' = -i - 1$ if i is positive and odd, or negative and even. Otherwise, let $i' = -i + 1$. Similarly define j' as $-j - 1$ or $-j + 1$. Consider the *corresponding* pairs of tuples $\mathsf{t}_{i'}$ and $\mathsf{t}_{j'}$. Since $i < j$ and t_i and t_j are non-adjacent, $j' < i'$ and $\mathsf{t}_{i'}$ and $\mathsf{t}_{j'}$ are non-adjacent as well. Note that in the other direction, tuples $\mathsf{t}_{i'}$ and $\mathsf{t}_{j'}$ correspond to tuples t_i and t_j, respectively, implying a one-to-one correspondence. $\mathsf{t}_{i\mathsf{B}} = \mathsf{t}_{i'\mathsf{B}}$ and $\mathsf{t}_{j\mathsf{B}} = \mathsf{t}_{j'\mathsf{B}}$, based on how the $\mathsf{A\ E/I\ OD}$ instance is constructed. Since $i < j$, $\mathsf{t}_{i\mathsf{A}} < \mathsf{t}_{i'\mathsf{A}}$ and tuples t_i and t_j are swapped *iff* $\mathsf{t}_{j\mathsf{B}} \prec \mathsf{t}_{i\mathsf{B}} \in \mathsf{T}_{\mathsf{B}}^{\prec}$. On the other hand, since $j' < i'$, $\mathsf{t}_{i\mathsf{A}} < \mathsf{t}_{i'\mathsf{A}}$ and tuples $\mathsf{t}_{i'}$ and $\mathsf{t}_{j'}$ are swapped *iff* $\mathsf{t}_{i'\mathsf{B}} \prec \mathsf{t}_{j'\mathsf{B}} \in \mathsf{T}_{\mathsf{B}}^{\prec}$; i.e., $\mathsf{t}_{i\mathsf{B}} \prec \mathsf{t}_{j\mathsf{B}} \in \mathsf{T}_{\mathsf{B}}^{\prec}$. Therefore, exactly *one* of pairs of tuples t_i and t_j or $\mathsf{t}_{i'}$ and $\mathsf{t}_{j'}$ is swapped for any given $\mathsf{T}_{\mathsf{B}}^{\prec}$. Since the number of pairs of non-adjacent tuples that can be swapped is C, the number of swapped pairs of non-adjacent tuples for *any* $\mathsf{T}_{\mathsf{B}}^{\prec}$ is $C/2$. □

We can now prove Theorem 7.

Proof
Validating $\mathsf{A\ E/I\ OD}$s is in class NP as a positive answer can be verified in polynomial time using a total order over the values of B as verifier. We now show that the MFAS instance holds *iff* the $\mathsf{A\ E/I\ OD}$ instance is valid. Since the reduction takes polynomial time in the size of the MFAS instance, this proves that validating $\mathsf{A\ E/I\ OD}$s is NP-complete.

First, assume the MFAS instance is holds; i.e., there exists a total order $\mathsf{T}_V^{\prec}$ over its vertices with at most t back-edges. Let $\mathsf{T}_\mathsf{B}^{\prec} = \mathsf{T}_V^{\prec}$ be a total order over the values of B. For any $1 \leq i \leq m$, there are *two* pairs of adjacent tuples in $\mathbf{r}$, i.e., $\mathtt{t}_{-2i}$ and $\mathtt{t}_{-2i+1}$ as well as $\mathtt{t}_{2i-1}$ and $\mathtt{t}_{2i}$, which correspond to the edge e_i in G. Since $-2i < -2i+1$, tuples $\mathtt{t}_{-2i}$ and $\mathtt{t}_{-2i+1}$ (and equivalently, tuples $\mathtt{t}_{2i-1}$ and $\mathtt{t}_{2i}$) are swapped, i.e., $\mathtt{t}_{-2i+1\mathsf{B}} \prec \mathtt{t}_{-2i\mathsf{B}} \in \mathsf{T}_\mathsf{B}^{\prec}$ *iff* e_i is a back-edge with respect to π, i.e., $e_i[1] \prec e_i[0] \in \pi$. A similar argument applies to tuples $\mathtt{t}_{2i-1}$ and $\mathtt{t}_{2i}$ as they are equivalent to tuples $\mathtt{t}_{-2i}$ and $\mathtt{t}_{-2i+1}$, respectively. Therefore, there exists at least one back-edge in the MFAS solution for any two swaps between *adjacent* tuples in the A E/I OD instance. Thus, the number of swaps between adjacent tuples is bounded by $2t$. Based on Lemma 9, there exists exactly $C/2$ swaps between non-adjacent tuples in $\mathbf{r}$ with respect to $\mathsf{T}_\mathsf{B}^{\prec}$. Therefore, the total number of swaps in $\mathbf{r}$ with respect to $\mathsf{T}_\mathsf{B}^{\prec}$ is at most $C/2 + 2t$ and the A E/I OD instance is valid.

For the other direction, assume the A E/I OD instance is valid and $\mathsf{T}_\mathsf{B}^{\prec}$ is a total order over the values of B with at most $C/2 + 2t$ swaps. Let $\mathsf{T}_V^{\prec} = \mathsf{T}_\mathsf{B}^{\prec}$ be an answer to the MFAS instance. Based on Lemma 9, exactly $C/2$ swaps exist between *non-adjacent* tuples in $\mathbf{r}$; therefore, the number of swaps between *adjacent* tuples is at most $2t$. Similar to the arguments for the other direction, edge e_i in the MFAS instance is a back-edge *iff* there are at least *two* pairs of swapped adjacent tuples in $\mathbf{r}$, i.e., tuples $\mathtt{t}_{-2i}$ and $\mathtt{t}_{-2i+1}$ as well as $\mathtt{t}_{2i-1}$ and $\mathtt{t}_{2i}$. Since the number of swaps between adjacent tuples is bounded by $2t$, the number of back-edges in the MFAS instance with respect to $\mathsf{T}_V^{\prec}$ is bounded by t. Therefore, the MFAS instance holds as well. □

Proof of Theorem 8

Lemma 10. *If the* A I/I OC *instance is valid, then in any solution with at most* $C/2 + 2t$ *swaps the orders* $\mathsf{T}_\mathsf{C}^{\prec}$ *and* $\mathsf{T}_\mathsf{D}^{\prec}$ *are* always $c_1 \prec c_2 \prec \cdots \prec c_{2k+1}$ *and* $D_{1,2} \prec \cdots \prec D_{k,k+1} \prec d_{\pi(1)} \prec d_{\pi(2)} \prec \cdots \prec d_{\pi(m)} \prec D_{k+1,k+2} \prec \cdots \prec D_{2k,2k+1}$*, respectively, or their reversals, where* π *is some permutation of numbers in range* $[1, m]$ *and* $\pi(i)$ *denotes the* i*-th number in the permutation. We refer to this solution structure as the* desired *structure.*

Proof

First, note that $C/2 + 2s < 8n^2N$, as $C = (C_{\mathsf{a},\mathsf{r}} + C_{\mathsf{d},\mathsf{r}} + C_{\mathsf{r}',\mathsf{r}''})/2$ and the following inequalities hold:

- $C_{\mathsf{a},\mathsf{r}}/2 \leq 2n \cdot nN/2 < 2n^2N$,
- $C_{\mathsf{d},\mathsf{r}}/2 \leq 2 \cdot n \cdot 2nN/2 < 4n^2N$,
- $C_{\mathsf{r}',\mathsf{r}''}/2 \leq n^2/2 < n^2N$, and
- $2s \leq 2\binom{n}{2} \leq n(n-1) < n^2N$.

Assume the A I/I OC candidate is valid and $\mathsf{T}_\mathsf{C}^\prec$ and $\mathsf{T}_\mathsf{D}^\prec$ are solutions with at most $C/2+2t$ swaps that do not follow this structure. Therefore, there exist swaps with respect to at least two tuples of type *dummy*, or one tuple of type *dummy* and one *anchor* tuple. Sine there are N copies of each dummy and anchor tuples in $\mathbf{t}$, each N copies of one tuple are swapped with the N copies of the other tuples, and therefore, $S(\mathsf{T}_\mathsf{C}^\prec, \mathsf{T}_\mathsf{D}^\prec) \geq N^2 = 64n^4$. On the other hand, we have proved that $C/2+2t < 8n^2N = 64n^4$. However, this contradicts the assumption that there are at most $C/2+2t$ swaps in $\mathbf{t}$ with respect to $\mathsf{T}_\mathsf{C}^\prec$ and $\mathsf{T}_\mathsf{D}^\prec$; i.e., $S(\mathsf{T}_\mathsf{C}^\prec, \mathsf{T}_\mathsf{D}^\prec) > C/2 + 2t$. □

Lemma 11. *In a solution to the* A I/I OC *instance with the* desired *structure,* $S_{\mathsf{a},\mathsf{r}} + S_{\mathsf{d},\mathsf{r}} + S_{\mathsf{r}',\mathsf{r}''} = C/2$ independent *of the order over* d_i*-values.*

Proof
Let $\mathsf{T}_\mathsf{C}^\prec$ and $\mathsf{T}_\mathsf{D}^\prec$ constitute a solution to the A I/I OC instance with the desired structure from Lemma 10 and, without loss of generality, let $\mathsf{T}_\mathsf{A}^\prec$ be $c_1 \prec c_2 \prec \cdots \prec c_{2k+1}$. We show a one-to-one correspondence from each pair of tuples $(\mathsf{a}_p, \mathsf{r}_q)$ to a pair of tuples $(\mathsf{a}_p, \mathsf{r}_{q'})$ such that exactly one of these pairs of tuples is swapped with respect to the solution with the desired structure. Consider tuples $\mathsf{a}_p = (c_{k+1}, d_i)$ and $\mathsf{r}_q = (c_l, d_j)$, where, without loss of generality, $l < k+1$. Consider the corresponding tuple $\mathsf{r}_{q'} = (c_{l+k+1}, d_j)$. Tuples a_p and r_q are swapped with respect to this solution iff $d_i \prec d_j \in \mathsf{T}_\mathsf{D}^\prec$. On the other hand, tuples a_p and $\mathsf{r}_{q'}$ are swapped iff $d_j \prec d_i \in \mathsf{T}_\mathsf{D}^\prec$. Therefore, in any solution with the desired structure, exactly half of the pairs of tuples of type *anchor* and *real* are swapped; i.e., $S_{\mathsf{a},\mathsf{r}} = C_{\mathsf{a},\mathsf{r}}/2$. A similar argument applies to $S_{\mathsf{d},\mathsf{r}}$ and $S_{\mathsf{r}',\mathsf{r}''}$. Therefore, $S_{\mathsf{a},\mathsf{r}} + S_{\mathsf{d},\mathsf{r}} + C_{\mathsf{r}',\mathsf{r}''} = C_{\mathsf{a},\mathsf{r}}/2 + S_{\mathsf{d},\mathsf{r}}/2 + C_{\mathsf{r}',\mathsf{r}''}/2 = C/2$ □

We can now prove Theorem 8.

Proof
Validating A I/I OCs is in class NP as a positive answer can be verified in polynomial time using two total orders over the values of A and B as verifier. We now show that the A E/I OC instance is valid *iff* the A I/I OC instance is valid. Since the reduction takes polynomial time in the size of the A E/I OC instance, this proves that validating A I/I OCs is NP-complete.

First, note that for any $\mathsf{T}_\mathsf{C}^\prec$ and $\mathsf{T}_\mathsf{D}^\prec$, $S(\mathsf{T}_\mathsf{C}^\prec, \mathsf{T}_\mathsf{D}^\prec) = S_{\mathsf{d},\mathsf{d}} + S_{\mathsf{a},\mathsf{d}} + S_{\mathsf{a},\mathsf{r}} + S_{\mathsf{d},\mathsf{r}} + S_{\mathsf{r},\mathsf{r}}$, using the union rule and the fact that $S_{\mathsf{a},\mathsf{a}} = 0$, as all tuples of type a have the same C-value. Also, $S_{\mathsf{r},\mathsf{r}} = S_{\mathsf{r}',\mathsf{r}''} + S_{\mathsf{r}',\mathsf{r}'} + S_{\mathsf{r}'',\mathsf{r}''}$.

Assume that the A E/I OC instance is valid and that $\mathsf{T}_\mathsf{B}^\prec$ is a solution with at most s swaps in $\mathbf{r}$. Construct a solution to the A I/I OC instance with $\mathsf{T}_\mathsf{C}^\prec$ and $\mathsf{T}_\mathsf{D}^\prec$ with a *desired*

(non-reversed) structure as described in Lemma 10, where $d_i \prec d_j \in \mathsf{T}_\mathsf{D}^\prec$ iff $b_i \prec b_j \in \mathsf{T}_\mathsf{B}^\prec$. First, $S_{\mathsf{a},\mathsf{r}} + S_{\mathsf{d},\mathsf{r}} + S_{\mathsf{r}',\mathsf{r}''} = C/2$ as per Lemma 11. Next, consider a swap between tuples $\mathsf{r}'_u = (c_i, d_r)$ and $\mathsf{r}'_v = (c_j, d_s)$ in $\mathbf{t}$ (the A I/I OC instance). There exist tuples $\mathsf{t}_u = (a_i, b_r)$ and $\mathsf{t}_v = (a_j, b_s)$ in $\mathbf{r}$ (the A E/I OC instance), since the *real* tuples correspond to tuples in $\mathbf{r}$. Without loss of generality, assume $c_i \prec c_j \in \mathsf{T}_\mathsf{C}^\prec$ and $d_s \prec d_r \in \mathsf{T}_\mathsf{D}^\prec$. On the other hand, $a_i < a_j$ and $b_s \prec b_r \in \mathsf{T}_\mathsf{B}^\prec$, due to the correspondence between the instances. Thus, tuples t_u and t_v in $\mathbf{r}$ are swapped with respect to $\mathsf{T}_\mathsf{B}^\prec$. Since tuples r''_u and r''_v are equivalent to tuples r'_u and r'_v, respectively, they are also swapped with respect to $\mathsf{T}_\mathsf{C}^\prec$ and $\mathsf{T}_\mathsf{D}^\prec$. Therefore, each *two* swaps in the A I/I OC instance within tuples of type r' and r'', respectively, correspond to at least one swap in the A E/I OC instance and $S_{\mathsf{r}',\mathsf{r}'} + S_{\mathsf{r}'',\mathsf{r}''} \leq 2s$. Therefore, $S(\mathsf{T}_\mathsf{C}^\prec, \mathsf{T}_\mathsf{D}^\prec) \leq C/2 + 2s$ and the A I/I OC instance is valid.

Next, assume that the A I/I OC instance is valid and let $\mathsf{T}_\mathsf{C}^\prec$ and $\mathsf{T}_\mathsf{D}^\prec$ constitute a solution with at most $C/2 + 2s$ swaps. Construct a solution to the A E/I OC instance $\mathsf{T}_\mathsf{B}^\prec$, where $b_i \prec b_j \in \mathsf{T}_\mathsf{B}^\prec$ iff $d_i \prec d_j \in \mathsf{T}_\mathsf{D}^\prec$. First, $\mathsf{T}_\mathsf{C}^\prec$ and $\mathsf{T}_\mathsf{D}^\prec$ have a *desired* structure due to Lemma 10 (if they have a *reversed* desired order, we first reverse both orders without affecting the number of swaps). Second, as per Lemma 11, $S_{\mathsf{a},\mathsf{r}} + S_{\mathsf{d},\mathsf{r}} + S_{\mathsf{r}',\mathsf{r}''} = C/2$ and thus, $S_{\mathsf{r}',\mathsf{r}'} + S_{\mathsf{r}'',\mathsf{r}''} \leq 2s$. Now, consider a swap between tuples $\mathsf{t}_u = (a_i, b_r)$ and $\mathsf{t}_v = (a_j, b_s)$ in $\mathbf{r}$ (the A E/I OC instance), where, without loss of generality, $a_i < a_j$ and $b_s \prec b_r \in \mathsf{T}_\mathsf{B}^\prec$. This swap corresponds to *two* pairs of swaps between tuples $\mathsf{r}'_u = (c_i, d_r)$ and $\mathsf{r}'_v = (c_j, d_s)$ as well as $\mathsf{r}''_u = (c_i, d_r)$ and $\mathsf{r}''_v = (c_j, d_s)$ in $\mathbf{t}$ (the A I/I OC instance). Therefore, each swap in the A E/I OC instance corresponds to at least two swaps of types $S_{\mathsf{r}',\mathsf{r}'}$ or $S_{\mathsf{r}'',\mathsf{r}''}$ in the A I/I OC instance. Since $S_{\mathsf{r}',\mathsf{r}'} + S_{\mathsf{r}'',\mathsf{r}''} \leq 2s$, as shown before, $S(\mathsf{T}_\mathsf{B}^\prec) \leq s$ and the A E/I OC instance is also valid. □

Proof of Theorem 9

Assume the SAT instance is satisfiable. A truth assignment to the SAT instance variables that satisfies all the clauses can be translated to two valid implicit orders for the I/I OC instance, by considering the variables for each pair of values, where a *true* assignment to $a_{i,j}$ or $a_{j,i}$ (or $b_{i,j}$ and $b_{j,i}$, accordingly) would result in the implied order $a_i \prec a_j$ or $a_j \prec a_i$, respectively. Exactly one of these two variables is set to *true*, given the *connexivity* clauses generated for each pair of variables. Furthermore, *transitivity* clauses ensure that this relation is transitive. Therefore, the derived implicit orders over A and B are valid strong total orders. To derive a witness order $\mathsf{T}_\mathbf{r}^\prec$, it is enough to sort the tuples in each partition group by the implicit order derived over the values of A, and break ties by the implicit order derived over the values of B (or vice versa). The projection of $\mathsf{T}_\mathbf{r}^\prec$ over A and B results in a valid total order, as all pairs of tuples within each partition group were considered when generating the *no swap* clauses, meaning that the projected order over the attributes

is perfectly captured by these clauses. Therefore, $\mathsf{T}_{\mathbf{r}}^{\prec}$ is a valid witness and the I/I OC instance is valid.

In the other direction, assume that the I/I OC candidate is valid and let $\mathsf{T}_{\mathbf{r}}^{\prec}$ denote a valid witness order over the tuples. To derive a solution to the SAT instance, the reverse of the previous algorithm can be performed; i.e., the truth assignment for variables $a_{i,j}$ and $a_{j,i}$ (or $b_{i,j}$ and $b_{j,i}$) can be set based on the orders between values in the projected order. If no order between some values has been established, ties are broken using an arbitrary *explicit* order, e.g., alphabetical order. Since $\mathsf{T}_{\mathbf{r}}^{\prec}$ is a valid witness and the explicit order used to break ties a valid total order, the projected order over each attribute will be a valid order. Therefore, the truth assignments for the SAT variables can be uniquely determined, and would satisfy the *connexivity* and *transitivity* conditions. Furthermore, this variable assignment would also satisfy the *no swaps* clauses, as the projected orders over variables involved in these clauses are directly derived from *pairs* of tuples. Therefore, the SAT instance is satisfiable. □

Proof of Lemma 6
The SAT representation has $\mathcal{O}(m^2)$ propositional variables. Creating the connexivity and transitivity clauses take $\mathcal{O}(m^2)$ and $\mathcal{O}(m^3)$ time, respectively. Generating the no-swap clauses for each BG takes $\mathcal{O}(m^2)$ time, as the number of edges in the bipartite graph derived from each partition group is $\mathcal{O}(m)$, since the initial BG is acyclic and does not contain any *3-fan-outs*. This makes the runtime of this step (and the number of generated clauses) $\mathcal{O}(pm^2)$, where p denotes the number of partition groups. Since the initial traversal of the database takes linear time, the total cost of the reduction to the SAT problem $\mathcal{O}(n + pm^2 + m^3)$. □

Proof of Theorem 10
Assume the MAX-SAT instance has a solution with satisfied hard clauses and t unsatisfied soft clauses. Construct a solution $\mathsf{T}_{\mathsf{B}}^{\prec}$ to the E/I OC instance as follows. $b_i \prec b_j \in \mathsf{T}_{\mathsf{B}}^{\prec}$ iff $b_{i,j} = \mathit{true}$. First, note that $\mathsf{T}_{\mathsf{B}}^{\prec}$ is a total order and thus, a valid solution to the E/I OC instance since *antisymmetry* and *transitivity* are satisfied by the *hard* clauses. (*Irreflexivity* is trivially satisfied since no clause $b_{i,i} = \mathit{true}$ exists.) Consider a swap in the E/I OC instance, i.e., tuples $\mathsf{t}_r = (u, b_i)$ and $\mathsf{t}_s = (v, b_j)$ such that $u < v$, but $b_j \prec b_i \in \mathsf{T}_{\mathsf{B}}^{\prec}$. This pair of tuples corresponds to the soft clause $(b_{i,j})$, which is set to *false* since $b_i \prec b_j \notin \mathsf{T}_{\mathsf{B}}^{\prec}$. Therefore, this tuples is *not* satisfied in the MAX-SAT instance. Therefore, there exists at least one unsatisfied soft clause for every pair of swapped tuples in $\mathbf{r}$ with respect to $\mathsf{T}_{\mathsf{B}}^{\prec}$. Thus, the number of swapped tuples in the A E/I OC instance is at most t.

For the other direction, assume the A E/I OC instance has a solution $\mathsf{T}_\mathsf{B}^{\prec}$ with $S(\mathsf{T}_\mathsf{B}^{\prec}) = t$. Create a solution to the MAX-SAT instance as follows. Set $b_{i,j} = \mathit{true}$ iff $b_i \prec b_j \in \mathsf{T}_\mathsf{B}^{\prec}$. First, note that all of the *hard* clauses in the MAX-SAT instance are satisfied in this solution, as $\mathsf{T}_\mathsf{B}^{\prec}$ is a valid total order. Next, consider an *unsatisfied* soft clause $(b_{i,j})$ in the MAX-SAT instance. This clause is unsatisfied only if $b_{i,j} = \mathit{false}$. Therefore, $b_i \prec b_j \notin \mathsf{T}_\mathsf{B}^{\prec}$, based on how the MAX-SAT solution is constructed. On the other hand, this clause corresponds to some pair of tuples $\mathsf{t}_r = (u, b_i)$ and $\mathsf{t}_s = (v, b_j)$ in $\mathbf{r}$ such that $u < v$. Since $b_i \prec b_j \notin \mathsf{T}_\mathsf{B}^{\prec}$, $b_j \prec b_i \in \mathsf{T}_\mathsf{B}^{\prec}$, which means that tuples t_r and t_s are swapped with respect to $\mathsf{T}_\mathsf{B}^{\prec}$. Therefore, there exists at least one swap in the A E/I OC instance corresponding to every unsatisfied clause in the MAX-SAT instance. Thus, the number of unsatisfied soft clauses is at most t. □

Proof of Lemma 7

The MAX-SAT representation has $\mathcal{O}(m^2)$ and $\mathcal{O}(m^3)$ propositional variables and *hard* clauses, which can be generated in $\mathcal{O}(m^3)$ time. To generate the *soft* clauses, after getting the sorted partitions of τ_A within each partition group in $\mathcal{O}(n \ln n)$, the tuples within each sorted partition can be traversed in linear time, as the cumulative occurrences of the values of B can be updated in $\mathcal{O}(1)$ for each tuple. However, for each sorted partition in τ_A, the values of the matrix M should be updated, which takes $\mathcal{O}(m^2)$ in the worst case, making this cost $\mathcal{O}(lm^2)$ within each partition group. As there are p partition groups with respect to the context, the total cost of the reduction is $\mathcal{O}(n \ln n + m^3 + plm^2)$. □

Proof of Theorem 11

Assume there exist a truth assignment to the MAX-SAT instance that satisfies all hard clauses and has t unsatisfied soft clauses. Construct a solution to the A I/I OC instance with $\mathsf{T}_\mathsf{A}^{\prec}$ and $\mathsf{T}_\mathsf{B}^{\prec}$ where $a_i \prec a_j \in \mathsf{T}_\mathsf{A}^{\prec}$ iff $a_{i,j} = \mathit{true}$ and $b_i \prec b_j \in \mathsf{T}_\mathsf{B}^{\prec}$ iff $b_{i,j} = \mathit{true}$. $\mathsf{T}_\mathsf{A}^{\prec}$ and $\mathsf{T}_\mathsf{B}^{\prec}$ are valid total orders since all hard clauses in the MAX-SAT instance are satisfied. Consider a swap in the A I/I OC instance between tuples $\mathsf{t}_r = (a_i, b_u)$ and $\mathsf{t}_s = (a_j, b_v)$, and without loss of generality, assume $a_i \prec a_j \in \mathsf{T}_\mathsf{A}^{\prec}$ and $b_v \prec b_u \in \mathsf{T}_\mathsf{B}^{\prec}$. Out of the two MAX-SAT clauses corresponding to this pair of tuples, the clause $(a_{j,i} \vee b_{u,v})$ is unsatisfied, as $a_{j,i} = b_{p,q} = \mathit{false}$. Therefore, there exists at least one unsatisfied clause in the MAX-SAT instance for any swapped pair of tuples in the A I/I OC instance. Thus, $S(\mathsf{T}_\mathsf{A}^{\prec}, \mathsf{T}_\mathsf{B}^{\prec}) \leq t$.

For the other direction, assume that $\mathsf{T}_\mathsf{A}^{\prec}$ and $\mathsf{T}_\mathsf{B}^{\prec}$ constitute a solution to the A I/I OC instance with t swaps. Construct a solution to the MAX-SAT instance in which $a_{i,j} = \mathit{true}$ *iff* $a_i \prec a_j \in \mathsf{T}_\mathsf{A}^{\prec}$, and $b_{i,j} = \mathit{true}$ *iff* $b_i \prec b_j \in \mathsf{T}_\mathsf{B}^{\prec}$. First, note that all *hard* clauses are satisfied, due to $\mathsf{T}_\mathsf{A}^{\prec}$ and $\mathsf{T}_\mathsf{B}^{\prec}$ being total orders. Next, consider tuples $\mathsf{t}_r = (a_i, b_u)$ and $\mathsf{t}_s = (a_j, b_v)$ in $\mathbf{r}$, and the corresponding *soft* tuples $(a_{i,j} \vee b_{v,u})$ and $(a_{j,i} \vee b_{u,v})$. Since the

hard clauses are satisfied, *at least* one of these soft clauses is *always* satisfied. Assume one of these clauses is not satisfied, and without loss of generality, let it be $(a_{i,j} \vee b_{v,u})$. Since this clause is not satisfied, $a_{i,j} = b_{v,u} = \mathit{false}$, and therefore, $a_j \prec a_i \in \mathsf{T}_{\mathsf{A}}^{\prec}$ and $b_u \prec b_v \in \mathsf{T}_{\mathsf{B}}^{\prec}$. Thus, tuples t_r and t_s are swapped with respect to $\mathsf{T}_{\mathsf{A}}^{\prec}$ and $\mathsf{T}_{\mathsf{B}}^{\prec}$ and there exists at least one swap in $\mathbf{r}$ for every unsatisfied clause in the MAX-SAT instance. Therefore, the number of unsatisfied clauses is at most t. □

Proof of Lemma 8

Generating the initial propositional variables and the *hard* clauses takes $\mathcal{O}(n + m^3)$ time. To generate the *soft* clauses, since there may be *3-fan-outs* in the bipartite graphs (unlike for the exact case), the size of BGs can be *quadratic* in the number of distinct values. Therefore, considering pairs of edges of bipartite graphs within each partition group can take up to $\mathcal{O}(m^4)$, making the total worst-case cost of this reduction $\mathcal{O}(n + pm^4)$. □