Lecture 2

Error Detection and Data Quality Rule Discovery

Extracting Information from Data

Data Cleaning Course

VLDB 2017 Summer School, Beijing

Lecture 2

Error Detection and Data Quality Rule Discovery

Extracting Information from Data

Data Cleaning Course

VLDB 2017 Summer School, Beijing

Detecting errors

We have seen many different types of data quality dependencies.

In the constraint-based data quality "paradigm"

"Errors **are** *violations of the constraints"*

When constraints are given

Checking for violations (=errors) is a matter of implementing **"easy" checks** on top of a DBMS.

There has been work on detecting violations by means of `SQL` queries.

Nevertheless, largely unexplored area of research.

- Increased efficiency by using specialised indexes?
- Incremental maintenance (violations are continuously monitored)?
- Distributed violation checking (when data is partitioned)?

Detecting errors

Most work, however, relates to **discovering the constraints**, which can then be used to detect the errors.

We focus on the discovery task ...

Where do data quality constraints/dependencies come from?

- Manual design (expensive and time consuming).
- Business rules (not expressive enough).

Dependency discovery: Idea

Given a sample of the data, find data quality dependencies that hold on the sample.

Sample
Discovery
Data quality dependencies

Inspiration from data mining algorithms:

Data mining techniques have been successfully applied to discover some of the data quality rules that we have seen earlier.

There already many different algorithms for a variety of dependencies!

Existing discovery algorithms (partial list)

FDs TANE: An Efficient Algorithm for Discovering Functional and Approximate Dependencies, Y. Huhtala, J. Kärkkainen, P. Porkka, H. Toivonen, Computer Journal, 1999.

FDs DFD: Efficient Functional Dependency Discovery, Z. Abedjan, P. Schulze, F. Naumann, CIKM 2014.

CFDs Discovering Conditional Functional Dependencies, W. Fan, F. Geerts, L. Jianzhong, M. Xiong, TKDE, 2010.

CFDs Discovering Data Quality Rules, F. Chiang, R. Miller,VLDB, 2008.

CFDs Estimating the confidence of conditional functional dependencies, G. Cormode, L. Golab, F. Korn, A. McGregor, D. Srivastava, X. Zhang, SIGMOD 2009.

DDs Differential dependencies: Reasoning and discovery, S. Song, L. Chen, TODS, 2011

INDs Unary and n-ary inclusion dependency discovery in relational databases. F. De Marchi, S. Lopes, and J.-M. Petit., JIIS 2009.

INDS Divide & conquer-based inclusion dependency discovery. T. Papenbrock, S. Kruse, J.-A. Quianè-Ruiz, and F. Naumann. VLDB, 2015.

CINDs Discovering conditional inclusion dependencies, J. Bauckmann Z. Abedjan, U. Leser, H. Müller, F. Naumann, CIKM 2012.

DCs Discovering denial constraints, X. Chu, I. Ilyas, P. Papotti, VLDB, 2013.

eRs Discovering editing rules for data cleaning. T. Diallo, J.-M. Petit, and S. Servigne. AQB, 2012.

MDs Discovering matching dependencies, S. Song and L. Chen. CIKM, 2009.

Discovery algorithms

Discovery algorithms can be roughly **classified** as:

- **Schema Driven**
  - Usually sensitive to the size of the schema.
  - Good for long thin tables!
- **Instance Driven**
  - Usually sensitive to the size of the data.
  - Good for fat short tables!
- **Hybrid**
  - Try to get the best of both worlds...

We start by looking at **Functional Dependency (FD) discovery**.

Discovering functional dependencies

Problem Statement

Input: Database instance $\mathcal{D}$ over schema R.

Output: Set Σ of **all** FDs $\varphi = R(X \rightarrow Y)$ that hold on $\mathcal{D}$, i.e., such that $\mathcal{D} \models \varphi$.

Uses

Schema design
Key discovery
Query optimization

Data cleaning
Anomaly detection
Index selection

A first observation: Not all FDs are interesting

- **Trivial:** Attributes in RHS [1] are a subset of attributes on LHS.
  - $R([Street, City] \rightarrow [City])$
  - Any trivial FD holds on a dataset.
- **Non-trivial:** At least one attribute in RHS does not appear on LHS.
  - $R([Street, City] \rightarrow [Zip, City])$
- **Completely non-trivial:** Attributes in LHS and RHS are disjoint.
  - $R([Street, City] \rightarrow [Zip])$

When discovering FDs...

Only interested in **completely non-trivial** functional dependencies.

[1] RHS=right hand side; LHS=left hand side

Further observations...

Logical implication

- It suffices to only discover a **minimal set** of FDs from the data, from which **all other FDs** that hold can be **derived**...

⇒ Finding out when FDs can be derived from other FDs is known as an **implication problem**

General implication problem

The implication problem

To determine,

- given a schema R, a set Σ of constraints and a single constraint φ defined on R,
- whether or not Σ **implies** φ, denoted by $\Sigma \models \varphi$.

That is, whether for **any** instance $\mathcal{D}$ of R that satisfies Σ, $\mathcal{D}$ also satisfies φ ($\mathcal{D} \models \varphi$).

Redundancy

To remove redundant data quality rules. Indeed, $\varphi \in \Sigma$ can be removed if $(\Sigma \setminus \{\varphi\}) \models \varphi$.

For FDs, this is easy to check.

Finite axiomatizability of FDs

Armstrong's axioms for FDs [2]:

$$\begin{aligned} \textbf{Reflexivity}: &\quad \text{If } Y \subseteq X, \text{ then } X \to Y \\ \textbf{Augmentation}: &\quad \text{If } X \to Y, \text{ then } XZ \to YZ \\ \textbf{Transitivity}: &\quad \text{If } X \to Y \text{ and } Y \to Z, \text{ then } X \to Z \end{aligned}$$

Sound and **complete**: $\Sigma \models \phi$ iff ϕ can be inferred from Σ using the axioms.

Example

Relation $R = \{A, B, C, G, H, I\}$
FDs $\Sigma = \{A \to B, A \to C, CG \to HCG \to I, B \to H\}$.
Show:

- $\Sigma \models A \to H$. Why?
- $\Sigma \models CG \to HI$. Why?

[2] We use $X \to Y$ to denote FD $R(X \to Y)$

Finite axiomatizability of FDs

Armstrong's axioms for FDs [2]:

Reflexivity : If $Y \subseteq X$, then $X \to Y$

Augmentation : If $X \to Y$, then $XZ \to YZ$

Transitivity : If $X \to Y$ and $Y \to Z$, then $X \to Z$

Sound and **complete**: $\Sigma \models \phi$ iff ϕ can be inferred from Σ using the axioms.

Example

Relation $R = \{A, B, C, G, H, I\}$
FDs $\Sigma = \{A \to B, A \to C, CG \to HCG \to I, B \to H\}$.
Show:

- $\Sigma \models A \to H$. Why? $A \to B$, $B \to H$, transitivity, $A \to H$.
- $\Sigma \models CG \to HI$. Why?

[2] We use $X \to Y$ to denote FD $R(X \to Y)$

Finite axiomatizability of FDs

Armstrong's axioms for FDs [2]:

$$\begin{aligned}\textbf{Reflexivity}: &\quad \text{If } Y \subseteq X, \text{ then } X \to Y\\ \textbf{Augmentation}: &\quad \text{If } X \to Y, \text{ then } XZ \to YZ\\ \textbf{Transitivity}: &\quad \text{If } X \to Y \text{ and } Y \to Z, \text{ then } X \to Z\end{aligned}$$

Sound and **complete**: $\Sigma \models \phi$ iff ϕ can be inferred from Σ using the axioms.

Example

Relation $R = \{A, B, C, G, H, I\}$
FDs $\Sigma = \{A \to B, A \to C, CG \to HCG \to I, B \to H\}$.
Show:

- $\Sigma \models A \to H$. Why? $A \to B$, $B \to H$, transitivity, $A \to H$.
- $\Sigma \models CG \to HI$. Why? Augmentation of $CG \to I$ to infer $CG \to CGI$, augmentation of $CG \to H$ to infer $CGI \to HI$, and then transitivity.

[2] We use $X \to Y$ to denote FD $R(X \to Y)$

Recall, we want to pinpoint precisely which FDs are sufficient to discover.

$\Rightarrow$ They must form a **minimal cover**

Cover of FDs

Minimal Cover

Given a set Σ of FDs, a **minimal cover** of Σ is a set Σ' of FDs

- such that Σ and Σ' are **equivalent**, i.e., $\Sigma \models \varphi'$ for all $\varphi' \in \Sigma'$ and $\Sigma' \models \varphi$ for all $\varphi \in \Sigma$; and
- **no proper subset** of Σ' has the previous property (it is minimal); and
- removing any attribute from a LHS of an FD in Σ' destroys equivalence (**non-redundancy**)

Discovery algorithms should preferably return a cover of all FDs that hold on a given instance!

Discovering covers

Algorithmically, you can either

1. **Post-process** discovered FDs to obtain a cover
   - This can be done using Armstrong's axioms
2. **Interleave redundancy checks** during discovery process
   - Most algorithms follow this approach

Discovering functional dependencies

A lot of different algorithms:

- Schema-driven:
    - TANE [Huhtala et al, Computer Journal 1999]
    - FUN [Novelli et al., 2001]
    - FDMine[Yao et al., 2002]
    - DepMiner[Lopez et al., 2000]
- Instance-driven: FASTFD [Wyss et al, DaWaK, 2001]
- Hybrid:
    - FDEP [Flach et al.,1999]
    - DFD [Abedjan et al. 2015]
    - . . .

Overview FD discovery

- Describe some naive methods
- Describe TANE algorithm in detail
- Mention other methods

Naive FD discovery algorithm

Naive Algorithm

```
Function: find_FDs($\mathcal{D}$)
return All valid FDs $\varphi$ such that $\mathcal{D} \models \varphi$.
─────────────────────────────────────────
for each attribute $A$ in $R$ do
    for each $X \subseteq R \setminus \{A\}$ do
        for each pair $(t_1, t_2) \in \mathcal{D}$ do
            if $t_1[X] = t_2[X]$ &
              $t_1[A] \neq t_2[A]$ then
                break
        return $X \rightarrow A$
```

Complexity: For each of the $|R|$ possibilities for RHS:

- check $2^{|R|-1}$ combinations for LHS
- scan the db $|\mathcal{D}|^2/2$ times for each combination.

Don't use this algorithm!

Very inefficient! No pruning of trivial or inferred FDs.

Slightly less naive FD discovery algorithm

Less Naive Algorithm

```
Function: all_count ($\mathcal{D}$)
return Store count($\mathcal{D}, X$) for all $X \subseteq R$.
```

```
Function: find_FD ($\mathcal{D}$)
return All valid FDs $\varphi$ such that $\mathcal{D} \models \varphi$.

for each attribute $A$ in $R$ do
    for each $X \subseteq R \setminus \{A\}$ do
        if count($\mathcal{D}, X$)=count($\mathcal{D}, X \cup A$) then
            return $X \rightarrow A$
```

Complexity:

- Precompute `SELECT COUNT(DISTINCT X) FROM R` for each $X \subseteq R$.
- For each of the $|R|$ possibilities for RHS: check $2^{|R|-1}$ combinations for LHS.

Also don't use this algorithm!

Database scans are factored out of the loop, but still inefficient!

TANE

TANE algorithm improves on these naive methods.

Idea behind the approach:

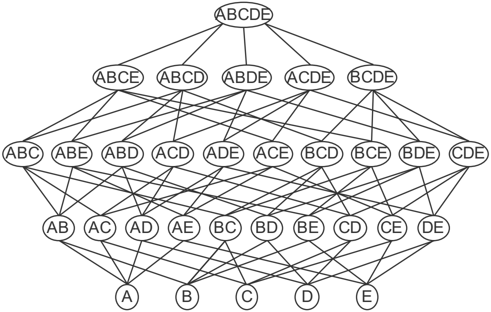
1. **Reduce column combinations** through pruning
   - Modelling of search space as lattice
   - Reasoning over FDs
2. **Reduce tuple sets** through partitioning
   - Partition data according to attribute values
   - Level-wise increase of size of attribute set

Search space modelling

- Model search space as power set lattice.

Power set lattice

- **Elements** in lattice: subsets of attributes in R;
- **Partial order**: $X \subseteq Y$;
- **Join** of two elements X and Y is $X \cup Y$;
- **Meet** of two elements X and Y is $X \cap Y$.



Lattice traversal

The lattice structure brings some **order** in the exploration space.

Bottom up traversal through lattice

- only minimal dependencies
- always tests for $X \setminus A \rightarrow A$ for $A \in X$
- Pruning
- Re-use results from previous level

Main idea:

For each visited element X in the lattice

$\Rightarrow$ maintain a **set of candidate RHS**.

RHS Candidate sets

RHS Candidate set $\mathcal{C}(X)$

- When considering X, it stores **only those attributes that might depend on all other attributes of X.**
  - I.e., those that still need to be checked
  - If $A \in \mathcal{C}(X)$ then A does not depend on any proper subset of X, i.e.,
$$\mathcal{C}(X) = R \setminus \{A \in X \mid D \models X \setminus A \to A\}$$

Let $R = \{ABCD\}$ and suppose that $D \models A \to C$ and $D \models CD \to B$. Then,

- $\mathcal{C}(A) = ABCD \setminus \{\} = C(B) = C(C) = C(D)$
- $\mathcal{C}(AB) = ABCD \setminus \{\}$
- $\mathcal{C}(AC) = ABCD \setminus \{C\} = ABD$
- $\mathcal{C}(CD) = ABCD \setminus \{\}$
- $\mathcal{C}(BCD) = ABCD \setminus \{B\} = ACD$

RHS Candidate pruning

Minimality Check

For minimality it suffices to consider $X \setminus A \to A$ where

- $A \in X$ and $A \in \mathcal{C}(X \setminus \{B\})$ for **all** $B \in X$.
- I.e., A is in **all** candidate sets of the subsets.

Let $X = \{ABC\}$. Assume we know $C \to A$ from previous step.

- Need to test three dependencies: $AB \to C$, $AC \to B$, and $BC \to A$. We should not be testing $BC \to A$, because we know $C \to A$
- Candidate sets of subsets of ABC:
  - $\mathcal{C}(AB) = ABC, \mathcal{C}(AC) = BC, \mathcal{C}(BC) = ABC$
- E.g., $BC \to A$ does not need to be tested for minimality, because A is not in all three candidate sets:

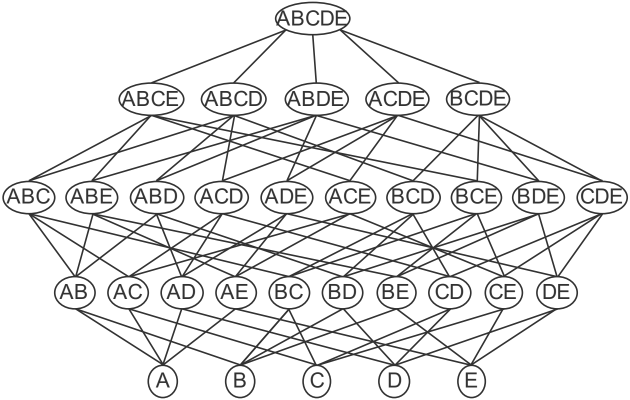
$$A \notin \mathcal{C}(AB) \cap \mathcal{C}(AC) \cap \mathcal{C}(BC) = \{BC\}.$$

- $AB \to C$, $AC \to B$ need to be tested, because B and C appear in all candidate sets.

RHS Candidate pruning

Final pruning step

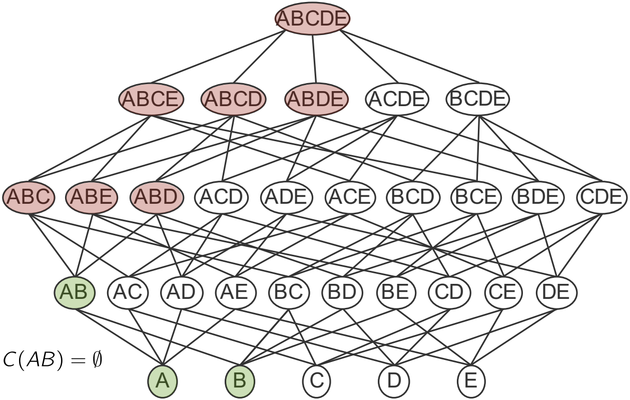
- If $\mathcal{C}(X) = \{\}$ then $\mathcal{C}(Y) = \{\}$ for all $Y \supset X$.
    - I.e, prune all supersets
- No $Y \setminus \{A\} \rightarrow A$ can be minimal and Y can be ignored.



RHS Candidate pruning

Final pruning step

- If $\mathcal{C}(X) = \{\}$ then $\mathcal{C}(Y) = \{\}$ for all $Y \supset X$.
  - I.e, prune all supersets
- No $Y \setminus \{A\} \to A$ can be minimal and Y can be ignored.



Improved RHS candidate pruning

Using implication rules

Let $B \in X$ and let $X \setminus B \to B$ hold. Then,

$$X \to A \text{ implies } X \setminus B \to A.$$

- E.g., $A \to B$ holds. Then $AB \to C$ implies $A \to C$.

Use this to reduce candidate set:

- If $X \setminus B \to B$ for some B, then any dependency with all of X on LHS cannot be minimal.
- Just remove B.

Revised $\mathcal{C}(X)$: $\mathcal{C}^+(X)$

Define

$$\mathcal{C}^+(X) = \{A \in R \mid \text{for all } B \in X, \\ X \setminus \{A, B\} \to B \text{ does not hold}\}$$

Special case: $A = B$, $\mathcal{C}^+(X) = \mathcal{C}(X)$.

Improved RHS candidate pruning

The definition $\mathcal{C}^+(X)$ removes three types of candidates:

- $\mathcal{C}_1 = \{A \in X \mid X \setminus A \rightarrow A$ holds$\}$ (as before)
- $\mathcal{C}_2 = \{R \setminus X \mid$if there exists a $B \in X$ such that $X \setminus B \rightarrow B$ holds.$\}$
- $\mathcal{C}_3 = \{A \in X \mid$ if there exists $B \in X \setminus A$ such that $X \setminus \{A, B\} \rightarrow B$ holds $\}$

Example of $\mathcal{C}_2$:

Recall

$$\mathcal{C}^+(X) = \{A \in R \mid \text{for all } B \in X,\ X \setminus \{A, B\} \to B \text{ does not hold}\}$$

and

$$\mathcal{C}_2 = \{R \backslash X \mid \text{if there exists a } B \in X \text{ such that } X \setminus B \to B \text{ holds.}\}$$

Consider $R = \{ABCD\}$, $X = \{ABC\}$. Assume $\mathcal{C}^+(X) = ABCD$ initially.

- Discovery of $C \to B$
- Remove B from $\mathcal{C}^+(X)$
- Additionally remove $R \setminus X = D$.

Ok, because remaining combination of LHS contains B and C and $ABC \to D$ is not minimal because $C \to B$.
Together: $\mathcal{C}^+(ABC) = \{ABCD\} \setminus \{BD\} = \{AC\}$.

Example of $\mathcal{C}_3$:

Recall

$\mathcal{C}^+(X) = \{A \in R \mid \text{for all } B \in X,\ X \setminus \{A, B\} \to B \text{ does not hold}\}$

and

$$\mathcal{C}_3 = \{A \in X \mid \text{ if there exists } B \in X \setminus A \text{ such that } X \setminus \{A, B\} \to B \text{ holds}$$

Same idea as before, but for subsets. Assume $Y \subset X$ such that $Y \setminus B \to B$ holds for some $B \in Y$. Then we can remove from $\mathcal{C}^+(X)$ all $A \in X \setminus Y$.

Consider $X = ABCD$ and let $C \to B$. We have $BC = Y \subseteq X$ and $X \setminus Y = AD$.

- Thus can remove all AD.

Ok, any remaining combination of LHS contains B and C. Hence $ABC \to D$ and $BCD \to A$. Again, since $C \to B$ any such FD is not minimal.
Together: $\mathcal{C}^+(X) = C$.

Key pruning

Insight

If X is superkey and $X \setminus B \to B$, then $X \setminus B$ is also a superkey.

1. If X is superkey, no need to test any $X \to A$.
2. If X is superkey and not key, any $X \to A$ is not minimal (for any $A \notin X$).
3. If X is superkey and not key, if $A \in X$ and $X \setminus A \to A$ then $X \setminus A$ is superkey, and no need to test.

Can prune all keys and their supersets

TANE Base algorithm

TANE

```
Function: tane($\mathcal{D}$)
return All valid minimal FDs $\varphi$ such that $\mathcal{D} \models \varphi$.
------------------------------------------------------------
$L_0 := \emptyset$
$\mathcal{C}^+(\emptyset) := R$
$L_1 := \{A \mid A \in R\}$
$\ell = 1$
while $L_\ell \neq \emptyset$ do
    compute_dependencies($L_\ell$)
    prune($L_\ell$)
    $L_{\ell+1}$:=generate_next_level($L_\ell$)
    $\ell := \ell + 1$
```

TANE: Generating lattice levels

TANE

```
1 Function: generate_next_level($L_\ell$)
2 return Generate candidate $X \subseteq$, $|X| = \ell + 1$
-----------------------------------------------------------
3 $L_{\ell+1} := \emptyset$
4 for $K \in$ prefix_blocks($L_\ell$) do
5     for $Y, Z \subseteq K$, $Y \neq Z$ do
6         $X := Y \cup Z$
7         if for all $A \in X$, $X \setminus A \in L_\ell$ then
8             $L_{\ell+1} := L_{\ell+1} \cup X$
9 return $L_{\ell+1}$
```

Explanation

- $L_{\ell+1}$ consists of all X of size $\ell + 1$ such that all $Y \subset X$ are in L_ℓ.
- Prefix blocks: disjoint sets from L_ℓ with common prefix of size $\ell - 1$ (all pairs for $\ell = 1$)
- Line 5. All subsets of a new set must appear in a lower level.

TANE: Compute dependencies

TANE

```
1  Function: compute_dependencies($L_\ell$)
2  return Minimal dependencies
   ________________________________
3  for $X \in L_\ell$ do
4      $\mathcal{C}^+(X) := \bigcap_{A \in X} \mathcal{C}^+(X \setminus A)$
5  for $X \in L_\ell$ do
6      for $A \in X \cap \mathcal{C}^+(X)$ do
7          if $X \setminus A \to A$ is valid then
8              return $X \setminus A \to A$
9              Remove $A$ from $\mathcal{C}^+(X)$
10             Remove all $B \in R \setminus X$ from $\mathcal{C}^+(X)$.
```

Explanation

- l4 Create candidate sets; each attribute must appear in all candidate sets of smaller size
- l6 Only test attributes from candidate set
- l7 Actual test on data
- l9 Reduce candidates by newly found dependency
- l10 Reduce candidates by all other attributes:

TANE: Pruning

TANE

```
1  Function: pruning($L_\ell$)
   ----------------------------------------
2  for $X \in L_\ell$ do
3      if $\mathcal{C}^+(X) = \emptyset$ then
4          delete $X$ from $L_\ell$
5      if $X$ is a (super) key then
6          for $A \in \mathcal{C}^+(X) \setminus X$ do
7              $Z := \bigcap_{B \in X} \mathcal{C}^+(X \cup A \setminus B)$
8              if $A \in Z$ then
9                  return $X \to A$
10         delete $X$ from $L_\ell$
```

Explanation

- Line 3: Basic pruning. Deletion from L_ℓ ensures that supersets cannot be created during level generation (loops not executed on empty candidate sets)
- Lines 4-8: Key pruning

TANE Sample run

$R = ABCD$, $C \to B$ and $AB \to D$ are to be discovered (Also: $AC \to D$ by implication)

1. $L_0 = \emptyset$,
   - $\mathcal{C}^+(\emptyset) = ABCD$. Nothing to do
2. $L_1 = \{A, B, C, D\}$.
   - $\mathcal{C}^+(X) = ABCD$ for all $X \in L_1$
   - Still nothing to do: No FDs can be generated from singletons
   - Thus, no pruning
3. $L_2 = \{AB, AC, AD, BC, BD, CD\}$
   - E.g., $\mathcal{C}^+(AB) = \mathcal{C}^+(AB \setminus A) \cap \mathcal{C}^+(AB \setminus B) = ABCD \cap ABCD$
   - $\mathcal{C}^+(X) = ABCD$ for all $X \in L_2$.
   - Dep. checks for $AB : A \to B$ and $B \to A$ Nothing happens

TANE Sample run (cnt'd)

3. $L_2 = \{AB, AC, AD, BC, BD, CD\}$
   - $\mathcal{C}^+(X) = ABCD$ for all $X \in L_2$
   - Dep. checks for BC: $B \to C$ (no!) and $C \to B$ (yes!)
   - Output $C \to B$
   - Delete B from $\mathcal{C}^+(BC) = ACD$
   - Delete $R \setminus \{BC\}$ from $\mathcal{C}^+(BC) = C$
   - (Note $BC \to A$ and $BC \to D$ are not minimal).
4. $L_3 = \{ABC, ABD, ACD, BCD\}$
   - $\mathcal{C}^+(ABC) = \mathcal{C}^+(AB) \cap \mathcal{C}^+(AC) \cap \mathcal{C}^+(BC) = C$
   - $\mathcal{C}^+(BCD) = \mathcal{C}^+(BC) \cap \mathcal{C}^+(BD) \cap \mathcal{C}^+(CD) = C$
   - $\mathcal{C}^+(ABD) = \mathcal{C}^+(ACD) = ABCD$ unchanged
   - Dep. check for ABC: $ABC \cap \mathcal{C}^+(ABC)$ are candidates
   - $AB \to C$ no! Did not check $BC \to A$ and $AC \to B$

TANE Sample run (cnt'd)

4. $L_3 = \{ABC, ABD, ACD, BCD\}$
    - $\mathcal{C}^+(ABC) = \mathcal{C}^+(BCD) = C$
    - $\mathcal{C}^+(ABD) = \mathcal{C}^+(ACD) = ABCD$
    - Dep. check for ABD: $ABD \cap \mathcal{C}^+(ABD)$ are candidates
        - $AD \to B$ and $BD \to A$: no!
        - $AB \to D$: yes! Output $AB \to D$
        - Delete D from $\mathcal{C}^+(ABD) = ABC$
        - Delete $R \setminus ABD$ from $\mathcal{C}^+(ABD) = AB$
    - Dep. check for BCD: $BCD \cap \mathcal{C}^+(BCD)$ are candidates
        - Only need to check $BD \to C$: no!
    - Dep. check for ACD: $ACD \cap \mathcal{C}^+(ACD)$ are candidates
        - $CD \to A$ and $AD \to C$: no!
        - $AC \to D$: yes! Output $AC \to D$
        - Delete D from $\mathcal{C}^+(ABD) = ABC$
        - Delete $R \setminus ACD$ from $\mathcal{C}^+(ABD) = AC$

TANE Sample run (cnt'd)

5. $L_4 = ABCD$
   - $\mathcal{C}^+(ABCD) = \mathcal{C}^+(ABC) \cap \mathcal{C}^+(ABD) \cap \mathcal{C}^+(ACD) \cap \mathcal{C}^+(BCD) = \{\}$
   - Nothing to check
   - Did not need to check
   - $BCD \to A$: Not minimal because $C \to B$
   - $ACD \to B$: Not minimal because $C \to B$
   - $ABD \to C$: Not minimal because $AB \to D$
   - $ABC \to D$: Not minimal because $AC \to D$.
6. Done.
7. Ouput: $C \to B$, $AB \to D$, $AC \to D$.

Dependency checking

X-equivalence

Tuples s and t are X-**equivalent** wrt attribute set X if $t[A] = s[A]$ for all $A \in X$.

X-Partitioning

Attribute set X partitions $\mathcal{D}$ into **equivalence classes:**

$$[t]_X = \{s \in \mathcal{D} \mid \forall A \in X, s[A] = t[A]\}.$$

Clearly,

$$\mathcal{D} = [t_1]_X \,\dot{\cup}\, [t_2]_X \,\dot{\cup} \cdots \dot{\cup}\, [t_k]_X.$$

for some $t_1, \ldots, t_k$. We denote the set of **parts by π_X.**

Partitioning example

tuple id	A	B	C	D
1	a_1	b_1	c_1	d_1
2	a_1	b_2	c_2	d_3
3	a_2	b_2	c_1	d_4
4	a_2	b_2	c_1	d_1
5	a_2	b_3	c_3	d_5
6	a_3	b_3	c_1	d_6
7	a_3	b_4	c_4	d_1
8	a_3	b_4	c_5	d_7

$$[1]_A = [2]_A = \{1, 2\}$$

$$\pi_A = \{\{1, 2\}, \{3, 4, 5\}, \{6, 7, 8\}\}$$

$$\pi_{BC} = \{\{1\}, \{2\}, \{3, 4\}, \{5\}, \{6\}, \{7\}, \{8\}\}$$

$$\pi_D = \{\{1, 4, 7\}, \{2\}, \{3\}, \{5\}, \{6\}, \{8\}\}$$

Partition refinement

Partition π **refines partition** π' if every equivalence class in π is a subset of some equivalence class in π'.

- $X \rightarrow A$ if and only if π_X refines π_A.
- π_X refines π_A if and only if $|\pi_X| = |\pi_{XA}|$
- Why?
    - If π_X refines π_A then $\pi_{AX} = \pi_X$
    - π_{XA} always refines π_A.
    - $\Rightarrow$ If $\pi_{XA} \neq \pi_A$ then $|\pi_X| \neq |\pi_{XA}|$
    - $\Rightarrow$ if $|\pi_X| = |\pi_{XA}|$ then $\pi_{XA} = \pi_X$.

Partition refinement

Partition π **refines partition** π' if every equivalence class in π is a subset of some equivalence class in π'.

- $X \rightarrow A$ if and only if π_X refines π_A.
- π_X refines π_A if and only if $|\pi_X| = |\pi_{XA}|$
- Why?
  - If π_X refines π_A then $\pi_{AX} = \pi_X$
  - π_{XA} always refines π_A.
  - $\Rightarrow$ If $\pi_{XA} \neq \pi_A$ then $|\pi_X| \neq |\pi_{XA}|$
  - $\Rightarrow$ if $|\pi_X| = |\pi_{XA}|$ then $\pi_{XA} = \pi_X$.

Partition refinement

Partition π **refines partition** π' if every equivalence class in π is a subset of some equivalence class in π'.

- $X \to A$ if and only if π_X refines π_A.
- π_X refines π_A if and only if $|\pi_X| = |\pi_{XA}|$
- Why?
  - If π_X refines π_A then $\pi_{AX} = \pi_X$
  - π_{XA} always refines π_A.
  - $\Rightarrow$ If $\pi_{XA} \neq \pi_A$ then $|\pi_X| \neq |\pi_{XA}|$
  - $\Rightarrow$ if $|\pi_X| = |\pi_{XA}|$ then $\pi_{XA} = \pi_X$.

Testing validity of FDs:

We have that $\mathcal{D} \models X \to A$ if and only if $|\pi_X| = |\pi_{XA}|$.

Partition refinement

Testing validity of FDs:

We have that $\mathcal{D} \models X \rightarrow A$ if and only if $|\pi_X| = |\pi_{XA}|$.

Example

tuple id	A	B
1	a_1	b_1
2	a_1	b_1
3	a_2	b_1
4	a_2	b_1
5	a_2	b_1
6	a_3	b_2
7	a_3	b_2
8	a_3	b_2

$$\pi_A = \{\{1,2\},\{3,4,5\},\{6,7,8\},$$
$$\pi_B = \{\{1,2,3,4,5\},\{6,7,8\}\}$$
$$\pi_{AB} = \{\{1,2\},\{3,4,5\},\{6,7,8\}\}$$

Hence, $|\pi_{AB}| = |\pi_A|$ and $A \rightarrow B$. Note, $|\pi_{AB}| > |\pi_B|$ and $B \rightarrow A$ does not hold.

Striped partitions

Idea: Optimization

Remove equivalence classes of size 1 from partitions.

Why? Singleton equivalence class cannot violate any FD.

Issue with striped partitions

tuple id	A	B
1	a_1	b_1
2	a_1	b_2
3	a_1	b_3
4	a_1	b_3
5	a_1	b_4
6	a_2	b_5
7	a_2	b_5
8	a_3	b_6

$$\pi_A = \{\{1,2,3,4,5\},\{6,7\},\{8\}\}$$
$$\pi'_A = \{\{1,2,3,4,5\},\{6,7\}\}$$
$$\pi_{AB} = \{\{1\},\{2\},\{3,4\},\{5\},\{6,7\},\{8\}\}$$
$$\pi'_{AB} = \{\{3,4\},\{6,7\}\}$$

Observe $|\pi'_{AB}| = |\pi'_A|$ yet $A \to B$ does not hold.

Striped partitions

Idea: Optimization

Remove equivalence classes of size 1 from partitions.

Why? Singleton equivalence class cannot violate any FD.

Issue with striped partitions

tuple id	A	B
1	a_1	b_1
2	a_1	b_2
3	a_1	b_3
4	a_1	b_3
5	a_1	b_4
6	a_2	b_5
7	a_2	b_5
8	a_3	b_6

$$\begin{aligned}\pi_A &= \{\{1,2,3,4,5\},\{6,7\},\{8\}\}\\ \pi'_A &= \{\{1,2,3,4,5\},\{6,7\}\}\\ \pi_{AB} &= \{\{1\},\{2\},\{3,4\},\{5\},\{6,7\},\{8\}\}\\ \pi'_{AB} &= \{\{3,4\},\{6,7\}\}\end{aligned}$$

Observe $|\pi'_{AB}| = |\pi'_A|$ yet $A \to B$ does not hold.

Striped partitions

Error functions

For striped partitions define

$$e(X) = \frac{\|\pi'_X\| - |\pi'_X|}{|D|}$$

where $\|\pi'_X\|$ is the sum of sizes of elements in π'_X.
Then, $X \to A$ if and only if $e(X) = e(XA)$.

Example

$$\begin{aligned}
\pi'_A &= \{\{1,2,3,4,5\},\{6,7\}\} \\
\|\pi'_A\| &= 7 \\
\pi'_{AB} &= \{\{3,4\},\{6,7\}\} \\
\|\pi'_{AB}\| &= 4 \\
e(A) &= (7-2)/8 = 5/8 \\
e(AB) &= 4 - 2/8 = 2/8.
\end{aligned}$$

Hence, $e(A) \neq e(AB)$ and $A \to B$ does not hold.

Where are the partitions used?

TANE

Function: compute_dependencies(L_ℓ)

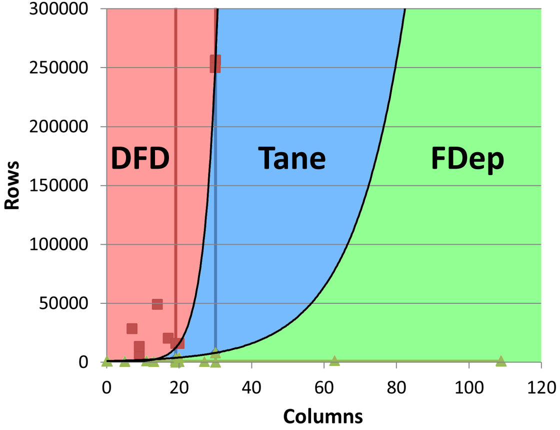
```
for $X \in L_\ell$ do
    $C^+(X) := \bigcap_{A \in X} C^+(X \setminus A)$
for $X \in L_\ell$ do
    for $A \in X \cap C^+(X)$ do
        if $X \setminus A \to A$ is valid then
            return $X \setminus A \to A$
            Remove $A$ from $C^+(X)$
            Remove all $B \in R \setminus X$ from $C^+(X)$.
```

Validity test

I6 $e(X \setminus A) = e(A)$?

Efficient algorithms in place to compute striped partitions.

Experimental comparison



- Source: Functional Dependency Discovery: An Experimental Evaluation of Seven Algorithms, Paperbrock et al, VLDB 2016
- https://hpi.de/naumann/projects/repeatability/data-profiling/fds.html

Approximate FD discovery

We next generalise TANE for discovering **approximate FDs**

Approximate FDs

An approximate FD $X \to A$ holds on $\mathcal{D}$ if

$$\mathtt{err}(X \to A) \leq \varepsilon,$$

where

$$\mathtt{err}(X \to A) = \frac{\min\{|S| \mid S \subseteq \mathcal{D}, \mathcal{D} \setminus S \models X \to A\}}{|\mathcal{D}|},$$

i.e., **minimum number of tuples to be removed from $\mathcal{D}$ such that $X \to A$ holds.**

TANE can be modified for approximate FDs

Approximate FD

Example

tuple id	A	B
1	a_1	b_1
2	a_1	b_2
3	a_1	b_3
4	a_1	b_3
5	a_1	b_4
6	a_2	b_5
7	a_2	b_5
8	a_3	b_6

tuple id	A	B
3	a_1	b_3
4	a_1	b_3
6	a_2	b_5
7	a_2	b_5
8	a_3	b_6

We know $A \rightarrow B$ does not hold.

$$\texttt{err}(X \rightarrow A) = 3/8$$

Error function can be efficiently computed using striped partitions.

Discovering approximate FDs

approx-TANE

```
1  Function: compute_approximate_dependencies($L_\ell$)
   ------------------------------------------------------------
2  for $X \in L_\ell$ do
3      $\mathcal{C}^+(X) := \bigcap_{A \in X} \mathcal{C}^+(X \setminus A)$
4  for $X \in L_\ell$ do
5      for $A \in X \cap \mathcal{C}^+(X)$ do
6          if $X \setminus A \to A$ is valid then
7              return $X \setminus A \to A$
8              Remove $A$ from $\mathcal{C}^+(X)$
9              Remove all $B \in R \setminus X$ from $\mathcal{C}^+(X)$.
   ------------------------------------------------------------
10 Line 6 is replaced by:
11 if_err($X \setminus A \to A$) $\leq \epsilon$ then
12 Line 9 is replace by:
13 if $X \setminus A \to A$ holds exactly then
14     Remove all $B \in R \setminus X$ from $\mathcal{C}^+(X)$.
```

CFD Discovery

TANE can also be generalised for discovering **conditional functional dependencies.**

CFDs

Definition

A CFD is an FD $R(X \rightarrow Y)$ expanded with a pattern tableau

$$T_p = \begin{array}{|c|} XA \\ \hline t_p^1 \\ \vdots \\ t_p^k \\ \hline \end{array}$$

where $t_p^1, \ldots, t_p^k$ are pattern tuples over $X \cup \{A\}$: constants, or wildcard _.

Example CFD

$$\varphi_2 : [CC = 44, ZIP] \rightarrow [STR]$$

- $(cust : [CC, ZIP] \rightarrow [STR], T_p)$
- pattern tableau T_p:

CC	ZIP	STR
44	_	_

CFD Satisfaction

A CFD $R(X \to Y, T_p)$ is satisfied on a database $\mathcal{D}$ iff for any two tuples s and t in $\mathcal{D}$:

- if $s[X] = t[X] \asymp t_p[X]$ for some $t_p \in T_p$;
- then also $s[A] = t[A] \asymp t_p[A]$.

CTANE

Modifications needed to TANE:

1. Lattice: (X, t_p)-pairs where as in TANE, X is set of attributes, but extended with pattern tuple.

   $\Rightarrow$ Search space is much larger!

2. Pruning: Armstrong's axioms need to be revised

   $\Rightarrow$ Needed to define candidate RHS sets $\mathcal{C}^+(X, t_p)$.

3. Traversal of lattice: Ensure that "most general" patterns are considered first

   $\Rightarrow$ If $R([AB] \rightarrow C, (a, \_, \_))$ holds, we don't need $R[([AB] \rightarrow C, (a, b, \_))$.

These modifications suffice to adapt TANE for CFDs!

Most challenging are the modification to Armstrong's axioms.

Axioms for CFDs: Reflexivity, augmentation

- FD1' (reflexivity): If $A \in X$, then $\varphi = (R : X \rightarrow A, t_p)$

Axioms for CFDs: Reflexivity, augmentation

- FD1' (reflexivity): If $A \in X$, then $\varphi = (R : X \to A, t_p)$

X_1	$\cdots$	X_i	$\cdots$	X_k	A
_	. . .	_	$\cdots$	_	_

Axioms for CFDs: Reflexivity, augmentation

- FD1' (reflexivity): If $A \in X$, then $\varphi = (R : X \to A, t_p)$

X_1	$\cdots$	A	$\cdots$	X_k	A
_	$\ldots$	a	$\cdots$	_	a

Axioms for CFDs: Reflexivity, augmentation

- FD1' (reflexivity): If $A \in X$, then $\varphi = (R : X \to A, t_p)$

X_1	$\cdots$	A	$\cdots$	X_k	A
_	$\ldots$	a	$\cdots$	_	a

- FD2' (augmentation):

Axioms for CFDs: Reflexivity, augmentation

- FD1' (reflexivity): If $A \in X$, then $\varphi = (R : X \rightarrow A, t_p)$

X_1	$\cdots$	A	$\cdots$	X_k	A
_	$\ldots$	a	$\cdots$	_	a

- FD2' (augmentation):

$$(R : [X_1, \ldots, X_k] \rightarrow [A], t_p)$$

X_1	$\cdots$	X_k	A
$t_p[X_1]$	$\ldots$	$t_p[X_k]$	$t_p[A]$

Axioms for CFDs: Reflexivity, augmentation

- FD1' (reflexivity): If $A \in X$, then $\varphi = (R : X \to A, t_p)$

X_1	$\cdots$	A	$\cdots$	X_k	A
_	$\ldots$	a	$\cdots$	_	a

- FD2' (augmentation):

$$(R : [X_1, \ldots, X_k, B] \longrightarrow [A], t'_p)$$

X_1	$\cdots$	X_k	B	A
$t_p[X_1]$	$\ldots$	$t_p[X_k]$	_	$t_p[A]$

Axioms for CFDs: Transitivity

- FD3' (transitivity):

Axioms for CFDs: Transitivity

- FD3' (transitivity):
  $(R : [X_1, \ldots, X_k] \to [Y_1, \ldots Y_\ell], t_p)$

X_1	$\cdots$	X_k	Y_1	$\cdots$	Y_ℓ
$t_p[X_1]$	$\ldots$	$t_p[X_k]$	$t_p[Y_1]$	$\cdots$	$t_p[Y_\ell]$

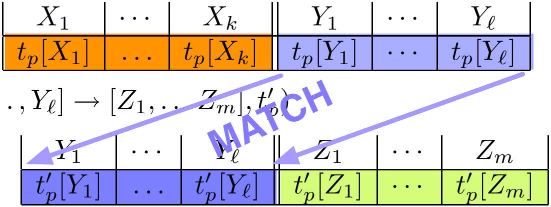
  $(R : [Y_1, \ldots, Y_\ell] \to [Z_1, \ldots Z_m], t'_p)$

Y_1	$\cdots$	Y_ℓ	Z_1	$\cdots$	Z_m
$t'_p[Y_1]$	$\ldots$	$t'_p[Y_\ell]$	$t'_p[Z_1]$	$\cdots$	$t'_p[Z_m]$

Axioms for CFDs: Transitivity

- FD3' (transitivity):

  $(R : [X_1, \ldots, X_k] \rightarrow [Y_1, \ldots Y_\ell], t_p)$



X_1	$\cdots$	X_k	Y_1	$\cdots$	Y_ℓ
$t_p[X_1]$	$\ldots$	$t_p[X_k]$	$t_p[Y_1]$	$\cdots$	$t_p[Y_\ell]$

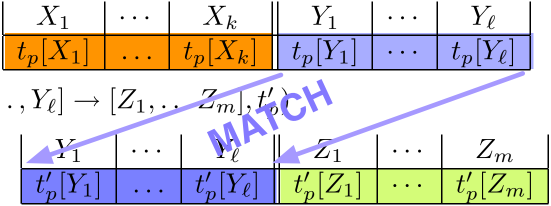
$(R : [Y_1, \ldots, Y_\ell] \rightarrow [Z_1, \ldots Z_m], t'_p)$

Y_1	$\cdots$	Y_ℓ	Z_1	$\cdots$	Z_m
$t'_p[Y_1]$	$\ldots$	$t'_p[Y_\ell]$	$t'_p[Z_1]$	$\cdots$	$t'_p[Z_m]$

Axioms for CFDs: Transitivity

- FD3' (transitivity):

$$(R : [X_1, \ldots, X_k] \rightarrow [Y_1, \ldots Y_\ell], t_p)$$



X_1	$\cdots$	X_k	Y_1	$\cdots$	Y_ℓ
$t_p[X_1]$	$\ldots$	$t_p[X_k]$	$t_p[Y_1]$	$\cdots$	$t_p[Y_\ell]$

$$(R : [Y_1, \ldots, Y_\ell] \rightarrow [Z_1, \ldots Z_m], t'_p)$$

Y_1	$\cdots$	Y_ℓ	Z_1	$\cdots$	Z_m
$t'_p[Y_1]$	$\ldots$	$t'_p[Y_\ell]$	$t'_p[Z_1]$	$\cdots$	$t'_p[Z_m]$

$$(R : [X_1, \ldots, X_k] \rightarrow [Z_1, \ldots Z_m], t''_p)$$

X_1	$\cdots$	X_k	Z_1	$\cdots$	Z_m
$t_p[X_1]$	$\ldots$	$t_p[X_k]$	$t'_p[Z_1]$	$\cdots$	$t'_p[Z_m]$

Axioms for CFDs: Reduction, upgrade

- FD4' (reduction):

$$(R : [X_1, \ldots, X_i, \ldots, X_k] \rightarrow A, t_p)$$

X_1	$\cdots$	X_i	$\ldots$	X_k	A
$t_p[X_1]$	$\ldots$	_	$\ldots$	$t_p[X_k]$	a

[Discovering Conditional Functional Dependencies, W. Fan, F. Geerts, L. Jianzhong, M. Xiong, TKDE, 2010.]

Axioms for CFDs: Reduction, upgrade

- FD4’ (reduction):

$$(R : [X_1, \ldots, \cancel{X_i}, \ldots, X_k] \rightarrow A, t_p)$$

X_1	$\ldots$	$\cancel{X_i}$	$\ldots$	X_k	A
$t_p[X_1]$	$\ldots$	$\cancel{\_}$	$\ldots$	$t_p[X_k]$	a

[Discovering Conditional Functional Dependencies, W. Fan, F. Geerts, L. Jianzhong, M. Xiong, TKDE, 2010.]

Axioms for CFDs: Reduction, upgrade

- FD4' (reduction):

$$(R : [X_1, \ldots, \xcancel{X_i}, \ldots, X_k] \rightarrow A, t_p)$$

X_1	$\ldots$	$\xcancel{X_i}$	$\ldots$	X_k	A
$t_p[X_1]$	$\ldots$	$\xcancel{\_}$	$\ldots$	$t_p[X_k]$	a

- FD5' (finite domain upgrade): suppose that the only consistent values for X_i are $b_1, b_2, \ldots, b_n$ and

[Discovering Conditional Functional Dependencies, W. Fan, F. Geerts, L. Jianzhong, M. Xiong, TKDE, 2010.]

Axioms for CFDs: Reduction, upgrade

- FD4' (reduction):

$$(R : [X_1, \dots, \cancel{X}_i, \dots, X_k] \to A, t_p)$$

X_1	$\dots$	$\cancel{X_i}$	$\dots$	X_k	A
$t_p[X_1]$	$\dots$	$\cancel{\_}$	$\dots$	$t_p[X_k]$	a

- FD5' (finite domain upgrade): suppose that the only consistent values for X_i are $b_1, b_2, \dots, b_n$ and

$$(R : [X_1, \dots, X_i, \dots, X_k] \to A, t_p)$$

X_1	$\dots$	X_i	$\dots$	X_k	A
$t_p[X_1]$	$\dots$	b_1	$\dots$	$t_p[X_k]$	$t_p[A]$
$t_p[X_1]$	$\dots$	b_2	$\dots$	$t_p[X_k]$	$t_p[A]$
$t_p[X_1]$	$\dots$	$\cdots$	$\dots$	$t_p[X_k]$	$t_p[A]$
$t_p[X_1]$	$\dots$	b_n	$\dots$	$t_p[X_k]$	$t_p[A]$

[Discovering Conditional Functional Dependencies, W. Fan, F. Geerts, L. Jianzhong, M. Xiong, TKDE, 2010.]

Axioms for CFDs: Reduction, upgrade

- FD4' (reduction):

$$(R : [X_1, \ldots, \xcancel{X}_i, \ldots, X_k] \rightarrow A, t_p)$$

X_1	$\cdots$	$\xcancel{X_i}$	$\ldots$	X_k	A
$t_p[X_1]$	$\ldots$	$\xcancel{\_}$	$\ldots$	$t_p[X_k]$	a

- FD5' (finite domain upgrade): suppose that the only consistent values for X_i are $b_1, b_2, \ldots, b_n$ and

$$(R : [X_1, \ldots, X_i, \ldots, X_k] \rightarrow A, t_p)$$

X_1	$\cdots$	X_i	$\ldots$	X_k	A
$t_p[X_1]$	$\ldots$	_	$\ldots$	$t_p[X_k]$	$t_p[A]$

[Discovering Conditional Functional Dependencies, W. Fan, F. Geerts, L. Jianzhong, M. Xiong, TKDE, 2010.]

Other CFD discovery tasks

The tableau generation problem

Given global support S and global confidence C, an FD $R(X \rightarrow Y)$ on a relation schema R with instance $\mathcal{D}$:

- Find a pattern tableau T_p of smallest size such that the CFD $R(X \rightarrow Y, T_p)$ is S-frequent and C-confident.

Example: Given [name, type, country] → [price, tax]

tid	name	type	country	price	tax
1	Harry Potter	book	France	10	0
2	Harry Potter	book	France	10	0
3	Harry Potter	book	France	10	0.05
4	The Lord of the Rings	book	France	25	0
5	The Lord of the Rings	book	France	25	0
6	Algorithms	book	USA	30	0.04
7	Algorithms	book	USA	40	0.04
8	Armani suit	clothing	UK	500	0.05
9	Armani suit	clothing	UK	500	0.05
10	Armani slacks	clothing	UK	250	0
11	Armani slacks	clothing	UK	250	0
12	Prada shoes	clothing	USA	200	0.05
13	Prada shoes	clothing	USA	200	0.05
14	Prada shoes	clothing	France	500	0.05
15	Spiderman	DVD	UK	19	0
16	Star Wars	DVD	UK	29	0
17	Star Wars	DVD	UK	25	0
18	Terminator	DVD	France	25	0.08
19	Terminator	DVD	France	25	0
20	Terminator	DVD	France	20	0

Optimal tableau

For FD [name, type, country] → [price, tax]

tableau with best coverage and support:

name	type	country	price	tax
_	clothing	_	_	_
_	book	France	_	0
_	_	UK	_	_

[On Generating Near-Optimal Tableaux for Conditional Functional Dependencies, Golab et al, VLDB 2008.]

Coverage of optimal tableau

tid	name	type	country	price	tax
1	Harry Potter	book	France	10	0
2	Harry Potter	book	France	10	0
3	Harry Potter	book	France	10	0.05
4	The Lord of the Rings	book	France	25	0
5	The Lord of the Rings	book	France	25	0
6	Algorithms	book	USA	30	0.04
7	Algorithms	book	USA	40	0.04
8	Armani suit	clothing	UK	500	0.05
9	Armani suit	clothing	UK	500	0.05
10	Armani slacks	clothing	UK	250	0
11	Armani slacks	clothing	UK	250	0
12	Prada shoes	clothing	USA	200	0.05
13	Prada shoes	clothing	USA	200	0.05
14	Prada shoes	clothing	France	500	0.05
15	Spiderman	DVD	UK	19	0
16	Star Wars	DVD	UK	29	0
17	Star Wars	DVD	UK	25	0
18	Terminator	DVD	France	25	0.08
19	Terminator	DVD	France	25	0
20	Terminator	DVD	France	20	0

A brief word on the discovery of **order dependencies**

Recall order dependencies

A typical salary situation

Records for Employees:

Name	Job	Years	Salary
Mark	Senior Programmer	15	35K
Edith	Junior Programmer	7	22K
Josh	Senior Programmer	11	50K
Ann	Junior Programmer	6	38K

Example order dependency:

> *"The salary of an employee is greater than other employees who have junior job titles, or the same job title but less experience in the company."*

Discovering order dependencies

- List-based lattice approach [Discovering Order Dependencies, Langer, Naumann, VLDB 2015]
  - Apriori-like, but order matters: $XY \rightarrow A$ is different from $YX \rightarrow A$
- Set-based lattice approach [Effective and Complete Discovery of Order Dependencies via Set-based Axiomatization, Szlichta, Godfrey, Golab, Kargar, Srivastava, VLDB 2017]
  - Rewrite ODs using a set-based canonical form

> Both approaches use new pruning rules based on OD semantics/axioms.

We next turn our attention to **denial constraints**

FASTDC Algorithm finds all minimal valid DCs by finding minimal covers [Chu et al, VLDB 2013]

Example DC

Example

"Two people living in the same state should have correct tax rates depending on their income"

$$\forall s, t \in \mathcal{D} \neg (s[\mathrm{AC}] = t[\mathrm{AC}] \wedge s[\mathrm{SAL}] < t[\mathrm{SAL}] \wedge s[\mathrm{TR}] > t[\mathrm{TR}])$$

FASTDC algorithm first computes **predicate space**.

Predicate space

Example

tuple id	A	B	C
1	a_1	a_1	50
2	a_2	a_1	40
3	a_3	a_1	40

Space of predicates $\mathcal{P}$:

$P_1 : t_i.A = t_j.A$ $\quad$ $P_2 = t_i.A \neq t_j.A$

$P_3 : t_i.B = t_j.B$ $\quad$ $P_4 = t_i.B \neq t_j.B$

$P_{11} : t_i.A = t_i.B$ $\quad$ $P_{12} = t_i.A \neq t_i.B$

$P_{21} : t_i.A = t_j.B$ $\quad$ $P_{22} = t_i.A \neq t_j.B$

$P_5 : t_i.C = t_j.C$ $\quad$ $P_6 = t_i.C \neq t_j.C$

$P_6 : t_i.C > t_j.C$ $\quad$ $P_8 = t_i.C \geq t_j.C$

$P_9 : t_i.C < t_i.C$ $\quad$ $P_1 0 = t_i.A \leq t_i.B$

Any combination of these predicates may be a valid DC.

Reduction to coverage

Coverage

$\neg(P_i \wedge P_j \wedge P_k)$ is a valid DC on D

$\Updownarrow$

For every pairs of tuples in I, P_i, P_j and P_k cannot be all true

$\Updownarrow$

For every pairs of tuples in I, at least one of P_i, P_j and P_k if false

$\Updownarrow$

For every pairs of tuples in I, at least one of $\neg P_i$, $\neg P_j$ and $\neg P_k$ is true

$\Updownarrow$

$\neg P_i$, $\neg P_j$ and $\neg P_k$ covers the set of true predicates (evidence) for every tuple pair

Theorem

$\neg(P_1 \wedge \cdots \wedge P_k)$ is a minimal valid DC if and only if $\{P_1, \ldots, P_k\}$ is a **minimal set cover** for all evidence sets. (Coverage means intersection). Minimality is wrt set containment.

FastDC

```
Function: FastDC ($\mathcal{D}$)
return Set $\Sigma$ of all valid denial constraints on $\mathcal{D}$.
$P \leftarrow$ build the predicate space for $\mathcal{D}$
$\mathcal{E} \leftarrow$ build the evidence sets based on $P$ and $\mathcal{D}$
for minimal cover $C$ of $E$ do
    $\Sigma := \Sigma \cup \{\neg \bar{C}\}$
```

Example

tuple id	A	B	C
1	a_1	a_1	50
2	a_2	a_1	40
3	a_3	a_1	40

Evidence sets $\mathcal{E}$:

$$(2,3),(3,2) = \{P_2, P_3, P_5, P_8, P_{10}, P_{12}, P_{14}\}$$

$$(2,1),(3,1) = \{P_2, P_3, P_6, P_8, P_9, P_{12}, P_{14}\}$$

$$(1,2),(1,3) = \{P_2, P_3, P_6, P_7, P_{10}, P_{11}, P_{13}\}$$

$\Rightarrow$ P_2 covers the set of true predicates minimally.

Hence, $\neg(\neg P_2) = \neg P_1$ is a valid minimal DC.

$\Rightarrow$ P_{10}, P_{14} covers the set of true predicates minimally.

Hence, $\neg(\neg P_{10} \wedge \neq P_{14})$ is a valid minimal DC

Some conclusions

Implication problem

- Most algorithm rely in some or other way on the axiomatization of implication of constraints
- Old classical problem, but needs revisiting for data quality constraints

Pruning

- Data mining learns us that in order to explore large spaces to find patterns (rules), pruning is required.
- All discovery algorithm rely on pruning methods based on implied constraints or thresholds for support, confidence (or other measures).

Open problems

- Many of the constraint formalisms do not have discovery algorithms yet
- For those who have, benchmarking is needed, to understand how they can be made more efficient.