Санкт-Петербургский государственный университет

Кафедра информационно-аналитических систем

Группа 21.Б08-мм

Обзор алгоритма обнаружения зависимостей включения MIND

Белоконный Сергей Александрович

Отчёт по учебной практике
в форме «Теоретическое исследование»

Научный руководитель:
Ассистент кафедры информационно-аналитических систем Чернышев Г. А.

Санкт-Петербург
2024

Оглавление

Введение

Современные приложения используют базы данных с таблицами, которые связаны первичными и внешними ключами. Обеспечение целостности данных в таких системах является важной задачей, которая решается с помощью механизмов валидации зависимостей данных. Одними из самых популярных типов зависимостей в базах данных являются функциональные и зависимости включения. В этой работе будут рассматриваться зависимости включения.

Поиск зависимостей включения является важным этапом в процессе восстановления или поддержки работы базы данных, поскольку они позволяют определять внешние ключи и обеспечить ссылочную целостность. Однако ошибки или грязные данные могут затруднить точное определение внешних ключей.

В таком случае для восстановления целостности данных необходимо использовать алгоритмы поиска зависимостей приближённого включения. В данной работе будет рассмотрен алгоритм поиска зависимостей включения MIND, который можно преобразовать для поиска зависимостей приближённого включения.

Одним из инструментов для валидации и поиска зависимостей является профайлер данных Desbordante. Desbordante валидирует и ищет зависимости с помощью специализированных алгоритмов. На данный момент разрабатывается алгоритм для поиска зависимостей включения, однако этот алгоритм не может работать с зависимостями приближённого включения.

Исследование возможности модификации алгоритма поиска зависимостей включения MIND для обнаружения зависимостей приближённого включения будет полезно для дополнения функционала профайлера данных Desbordante.

1. Постановка задачи

Целью работы является обзор работы алгоритма обнаружения включающих зависимостей MIND. Для её выполнения были поставлены следующие задачи:

1. составить обзор предметной области включающих зависимостей;
2. изучить принципы работы алгоритма MIND для поиска зависимостей включения и изложить принцип модификации этого алгоритма для поиска зависимостей приближённого включения;

2. Обзор предметной области

Одними из самых популярных типов связей таблиц в базах данных являются первичные и внешние ключи. Они определяются через функциональные зависимости и зависимости включения соответственно. Функциональные зависимости хорошо исследованы и для них создано достаточно большое количество алгоритмов для их валидации. Включающие зависимости исследованы не так хорошо. Поиск зависимостей включения необходим для поддержания целостности базы данных или для выделения ненадёжных источников данных.

2.1. Зависимости включения

Зависимости включения — это зависимости между столбцами, при которых множество значений одного столбца входит в множество значений другого. В базах данных такие зависимости описываются через внешние ключи. Формальное определение зависимостей включения выглядит так:

Пусть R — конечный набор атрибутов в базе данных. Для каждого атрибута $A \in R$ набор всех возможных значений этого атрибута называется областью определения и обозначается с помощью $Dom(A)$. Кортеж u над R — это отображение из R в $\bigcup_{A \in R} Dom(A)$, где $u(A) \in Dom(A)$, $\forall A \in R$. Набор кортежей r над R называется отношением над R, R называют схемой отношений r. Мощность множества X обозначается с помощью $|X|$. Пусть $X \subseteq R$ — набор атрибутов, u — кортеж, тогда $u[X]$ — ограничение u по X. Проекция отображения отношения r на X обозначается с помощью $\pi_X(r)$ и определяется как $\pi_X(r) = \{u[X] | u \in r\}$.

Схема базы данных $\mathbf{R}$ — конечный набор схем отношений R_i. База данных $\mathbf{d}$ над $\mathbf{R}$ — это набор отношений r_i над каждым R_i в $\mathbf{R}$.

Последовательность атрибутов (например $X =< A\,B\,C >$) — это упорядоченное множество различных атрибутов. $X[i]$ обозначает i'й элемент в последовательности.

Мы говорим, что атрибуты A и B являются совместимыми, если $Dom(A) = Dom(B)$. Мы говорим, что две различные последовательно-

сти атрибутов X и Y являются совместимыми, если $|X| = |Y| = m$ и для каждого $j = [1, m], Dom(X[j]) = Dom(Y[j])$.

Зависимость включения (inclusion dependency, IND) над схемой базы данных $\mathbf{R}$ это утверждение вида $R_i[X] \subseteq R_j[Y]$, где R_i, $R_j \in \mathbf{R}, X \subseteq R_i, Y \subseteq R_j$, X и Y — совместимые последовательности атрибутов. Зависимость включения называется тривиальной, если $R[X] \subseteq R[Y]$. Зависимость включения $R[X] \subseteq R[Y]$ имеет размер i, если $|X| = i$. Мы говорим, что зависимость включения унарная, если её размер равен единице.

Пусть база данных $\mathbf{d}$ над схемой базы данных $\mathbf{R}$, $r_i, r_j \in \mathbf{d}$ — отношения над $R_i, R_j \in \mathbf{R}$ соответственно. Зависимость включения $R_i[X] \subseteq R_j[Y]$ выполняется в базе данных $\mathbf{d}$ над $\mathbf{R}$ если, и только если $\forall u \in r_i, \exists v \in r_j, u[X] = v[Y]$. Обозначение: $\mathbf{d} \vDash R_i[X] \subseteq R_j[Y]$.

Пусть $\mathcal{I}_1$ и $\mathcal{I}_2$ два набора зависимостей включений, $\mathcal{I}_1$ называется покрытием $\mathcal{I}_2$, если $\mathcal{I}_1 \vDash \mathcal{I}_2$ (то есть каждая зависимость в $\mathcal{I}_2$ выполняется в любой базе данных, где выполняются все зависимости в $\mathcal{I}_1$) и $\mathcal{I}_2 \vDash \mathcal{I}_1$.

Свойства зависимостей включения [2]:

1. (Рефлексивность) $R[A_1, ..., A_n] \subseteq R[A_1, ..., A_n]$;

2. (Проекция и перестановка) если $R[A_1, ..., A_n] \subseteq S[B_1, ..., B_n]$, то $R[A_{\sigma 1}, ..., A_{\sigma m}] \subseteq S[B_{\sigma 1}, ..., B_{\sigma m}]$, где $\sigma 1, ..., \sigma m$ — различные числа от 1 до n;

3. (Транзитивность) если $R[A_1, ..., A_n] \subseteq S[B_1, ..., B_n]$ и $S[B_1, ..., B_n] \subseteq T[C_1, ..., C_n]$, то $R[A_1, ..., A_n] \subseteq T[C_1, ..., C_n]$.

Рассмотрим пример на Рисунке 1. В этом примере показаны зависимости включения $S[D] \subseteq R[A]$, $S[E] \subseteq R[B]$ и $S[DE] \subseteq R[AB]$. Зависимость включения $S[F] \subseteq R[C]$ не выполняется, так как в столбце F есть значение 14.0, которого нет в столбце C.

A	B	C
1	5	11.0
2	6	12.0
3	7	13.0
4	8	12.0

(a) r

D	E	F
1	5	11.0
1	5	11.0
2	6	12.0
1	5	14.0

(b) s

Рис. 1: Пример таблиц с зависимостями включения

2.2. Зависимости приближённого включения

Из-за ошибок или грязных данных некоторые зависимости включения, которые должны быть верными в схеме базы данных, могут не выполняться. Для этого нам необходимо определить меру погрешности, которая позволит учесть небольшие несоответствия. Такие зависимости включения, которые выполняются с погрешностью, называются зависимостями приближённого включения.

Для функциональных зависимостей уже существуют метрики для определения погрешности [7]. Для определения погрешности зависимости приближённого включения используется метрика g_3', которая показывает, какую часть значений атрибута X надо убрать, чтобы получить такую базу данных $\mathbf{d}'$, что $\mathbf{d}' \vDash R[X] \subseteq S[Y]$. Формальное определение этой метрики выглядит так [10]:

$$g_3'(R[X] \subseteq S[Y], \mathbf{d}) = 1 - \frac{\max(|\pi_X(r')| \text{ т.ч. } r' \subseteq r \text{ и } (\mathbf{d} - \{r\}) \cup \{r'\} \vDash R[X] \subseteq S[Y])}{|\pi_X(r)|}$$

Если у нас есть заданная погрешность $\epsilon \in [0, 1]$, то зависимость приближённого включения I выполняется в базе данных $\mathbf{d}$ с погрешностью ϵ если, и только если $g_3'(I, \mathbf{d}) \leq \epsilon$. Обозначение: $\mathbf{d} \vDash_\epsilon I$.

На Рисунке 1 $g_3'(S[F] \subseteq R[C], \mathbf{d}) = 1/3$ так как необходимо убрать одно значение 14.0 из набора 11.0, 12.0 и 14.0. Если взять погрешность $\epsilon = 0.34$, то $\mathbf{d} \vDash_{0.34} S[F] \subseteq R[C]$

Из определения метрики g_3' вытекает алгебраическое свойство, так

как достаточно посчитать отношение числа значений в левой части, которых нет в правой, что упрощает вычисление этой метрики.

$$g_3'(R[X] \subseteq S[Y], \mathbf{d}) = \frac{|\pi_X(r) - \pi_Y(s)|}{|\pi_X(r)|}$$

2.3. Алгоритмы поиска зависимостей включения

Для поиска зависимостей включения существует множество алгоритмов. Здесь представлены одни из самых популярных:

1. Spider [1] — алгоритм поиска унарных зависимостей включения, который может быть модифицирован для поиска n-арных зависимостей влючения и зависимостей приближённого включения. Этот алгоритм использует блоки атрибутов с одинаковым значением для поиска зависимостей включения;

2. S-indd [13] — алгоритм поиска унарных зависимостей включения, использующий принципы Spider, однако оптимизирующий работу с файлами;

3. Binder [4] — алгоритм поиска унарных и n-арных зависимостей включения, использующий метод «разделяй и влавствуй»;

4. SINDY [9] — алгоритм поиска унарных зависимостей включения, который позволяет распределить поиск на несколько устройств;

5. FAIDA [5] — приблизительный алгоритм поиска унарных и n-арных зависимостей включения;

6. MANY [15] — алгоритм поиска унарных зависимостей включения, который специализируется на таблицах с большим количеством атрибутов, в которых малое количество значений;

7. MIND [11] — алгоритм поиска унарных и n-арных зависимостей включения, который использует метод уровней для генерации и отбрасывания кандидатов, которые не будут выполняться. Может

быть модифицирован для поиска зависимостей приближённого включения;

8. ZigZag [3] — алгоритм поиска унарных и n-арных зависимостей включения, использующий 3 набора зависимостей включения для поиска зависимостей включения: позитивную границу, которая содержит в себе все максимальные выполняющиеся зависимости включения, негативную границу, которая содержит в себе все минимальные выполняющиеся зависимости включения, и оптимистичную позитивную границу, которая содержит в себе максимальных кандадатов в выполняющиеся зависимости включения;

9. Find2 [8] — алгоритм поиска n-арных зависимостей включения. Этот алгоритм генерирует кандидатов для n-арных зависимостей включения, проверяет их на базе данных и, если зависимости выполняются, отбрасывает все зависимостей включения, входящие в них, иначе разбивает кандидата на меньшие зависимостей включения и проверяет уже их;

10. MIND2 [14] — алгоритм поиска унарных и n-арных зависимостей включения, который использует так называемые координаты унарных зависимостей включения, которые генерируются из базы данных, а после может сгенерировать n-арные зависимости включения без необходимости доступа к базе данных.

Подробное сравнение производительности алгоритмов можно найти тут [6].

3. Обзор MIND

Основная идея алгоритма MIND формулируется так: пусть есть база данных $\mathbf{d}$ над схемой $\mathbf{R}$ и набор унарных зависимостей включения, выполняющихся в $\mathbf{d}$, найдём покрытие $R[X] \subseteq S[Y]$, $R, S \in \mathbf{R}$ всех нетривиальных зависимостей включения такое, что $\mathbf{d} \vDash R[X] \subseteq S[Y]$. Пользуясь свойствами, мы можем искать зависимости включения с помощью уровней, генерируя кандидатов в набор зависимостей включения размера $i+1$ из выполняющихся зависимостей включения размера i.

3.1. Подготовка данных

Пусть есть база данных $\mathbf{d}$ над схемой $\mathbf{R}$ и тип данных t в $\mathbf{d}$, контекстом извлечения называют тройку значений $\mathbb{D}_t(\mathbf{d}) = (\mathbb{U}, \mathbb{V}, \mathbb{B})$, где:

- $\mathbb{U} = \{R[A]$, где A имеет тип t, $A \in R$, $R \in \mathbf{R}\}$, то есть $\mathbb{U}$ — набор атрибутов с типом t;
- $\mathbb{V} = \{v \in \pi_A(r) \mid R[A] \in \mathbb{U}, r \in \mathbf{d}, r$ определён над $R\}$, то есть $\mathbb{V}$ — набор значений, которые принимают атрибуты в отношениях;
- $\mathbb{B} \subseteq \mathbb{V} \times \mathbb{U}$ — бинарное отношение, определённое как $(v, R[A]) \in \mathbb{B} \iff v \in \pi_A(r)$, где $r \in \mathbf{d}$ и r определён над R.

Пример контекста извлечения таблиц, представленных на Рисунке 1, можно увидеть на Рисунке 2.

3.2. Поиск унарных зависимостей включения

Пусть $\mathbf{d}$ — база данных, t — тип данных и $\mathbb{D}_t(\mathbf{d}) = (\mathbb{U}, \mathbb{V}, \mathbb{B})$ — контекст извлечения, тогда

$$\mathbf{d} \vDash A \subseteq B \iff B \in \bigcap_{v \in \mathbb{V} \mid (v,A) \in \mathbb{B}} \{C \in \mathbb{U} \mid (v, C) \in \mathbb{B}\}$$

где $A, B \in \mathbb{U}$.

$\mathbb{V}$	$\mathbb{U}$
1	A D
2	A D
3	A
4	A
5	B E
6	B E
7	B
8	B

(a) Целые числа

$\mathbb{V}$	$\mathbb{U}$
11.0	C F
12.0	C F
13.0	C
14.0	F

(b) Вещественные числа

Рис. 2: Пример контекста извлечения

// Вводные данные: контекст извлечения $(\mathbb{U}, \mathbb{V}, \mathbb{B})$ из базы данных $\mathbf{d}$ и типа данных t.
// Выходные данные: $\mathcal{I}_1$ --- набор унарных зависимостей включения, которые выполняются в базе данных $\mathbf{d}$ между атрибутами типа t.
1 Для каждого $A \in \mathbb{U}$: $rhs(A) = \mathbb{U}$;
2 Для каждого $v \in \mathbb{V}$:
3 Для каждого A такого, что $(v, A) \in \mathbb{B}$:
4 $rhs(A) = rhs(A) \cap \{B \mid (v, B) \in \mathbb{B}\}$;
5 Для каждого $A \in \mathbb{U}$:
6 Для каждого $B \in rhs(A) \setminus \{A\}$:
7 $\mathcal{I}_1 = \mathcal{I}_1 \cup \{A \subseteq B\}$;
8 Вернуть $\mathcal{I}_1$.

Рис. 3: Алгоритм поиска унарных зависимостей включения

Алгоритм, представленный на Рисунке 3, позволяет найти все унарные зависимости включения за один проход.

Сложность алгоритма на Рисунке 3 линейно зависит от размера бинарного отношения в контексте извлечения, так как пересечение двух отсортированных множеств равняется двойному размеру наименьшего множества, а так как размер $rhs(A)$ значительно меньше размера $\mathbb{U}$, то сложность пересечения можно принять за константу.

Рассмотрим работу алгоритма на примере в Рисунке 1 над целыми числами.

В первой строчке инициализируем $rhs(A) = rhs(B) = rhs(D) = rhs(E) = \{A, B, D, E\}$.

Дальше, после прохождения одного значения в цикле на второй строчке получаем:

- $rhs(A) = \{A, D\}$
- $rhs(B) = \{A, B, D, E\}$ (не изменилось)
- $rhs(D) = \{A, D\}$
- $rhs(E) = \{A, B, D, E\}$ (не изменилось)

После прохождения цикла на второй строчке получаем:

- $rhs(A) = \{A\}$
- $rhs(B) = \{B\}$
- $rhs(D) = \{A, D\}$
- $rhs(E) = \{B, E\}$

Дальше считаем унарные зависимости включения на строчках 5-7: $S[D] \subseteq R[A]$, $S[E] \subseteq R[B]$.

3.3. Генерация кандидатов размера i+1

Сначала определим область поиска. Кандидаты м собираются из атрибутов правой и левой частей зависимости включения. Заметим, что третьему свойству зависимостей включения, порядок атрибутов не важен, поэтому для простоты будем использовать лексикографический порядок по названиям атрибутов.

Определим отношение $\preceq$ между двумя кандидатами:

Пусть $I_1 : R_i[X] \subseteq R_j[Y]$ и $I_2 : R'_i[X'] \subseteq R'_j[Y']$ — два кандидата в набор зависимостей включения. Тогда $I_2 \preceq I_1$ при:

1. $R_i = R'_i$ и $R_j = R'_j$;
2. $X' =< A_1, \ldots, A_k >$, $Y' =< B_1, \ldots, B_k >$ и есть набор индексов $i_1 < \cdots < i_h \in \{1, \ldots, k\}$, где $h \leq K$, такой, что $X =< A_{i_1}, \ldots, A_{i_h} >$, $Y =< B_{i_1}, \ldots, B_{i_h} >$.

Определим $I_1 \prec I_2$, если $I_1 \preceq I_2$ и $I_2 \npreceq I_1$.

Пусть I_1, I_2 — 2 кандидата в набор зависимостей включения таких, что $I_1 \preceq I_2$. Если $\mathbf{d} \nvdash I_1$, тогда $\mathbf{d} \nvdash I_2$. Это утверждение позволит сокращать область поиска, так как для генерации следующих кандидатов мы будем использовать только те зависимости включения, которые выполняются на текущем.

Так же введём обозначения, которые будем использовать дальше:

- $\mathcal{C}_i$ — кандидаты в набор зависимостей включения размера i;
- $\mathcal{I}_i$ — набор выполняющихся зависимостей включения размера i;
- $I.lhs$ — левая последовательность атрибутов зависимости включения I;
- $I.rhs$ — правая последовательность атрибутов зависимости включения I;
- $X.rel$ — схема отношений атрибутов последовательности X.

```
// Вводные данные: I_i --- набор зависимостей включения размера i.
// Выходные данные: C_{i+1} --- последовательность кандидатов в набор зависимостей включения размера i+1.
01 Для каждого p, q из I_i:
02     Если p.lhs.rel = q.lhs.rel и p.rhs.rel = q.rhs.rel и
03          p.lhs[1] = q.lhs[1] и p.rhs[1] = q.rhs[1] и
04          ...
05          p.lhs[i-1] = q.lhs[i-1] и p.rhs[i-1] = q.rhs[i-1] и
06          p.lhs[i] < q.lhs[i] и
07          p.rhs[i] ≠ q.rhs[i] тогда:
08         Добавить в C_{i+1}:
09              p.lhs.rel[p.lhs[1], p.lhs[2], ..., p.lhs[i], q.lhs[i]] ⊆
   p.rhs.rel[p.rhs[1], p.rhs[2], ..., p.rhs[i], q.rhs[i]];
10 Для каждого I ∈ C_{i+1}:
11     Для каджого J ≺ I и J размера i:
12         Если J ∉ I_i:
13             C_{i+1} = C_{i+1} \ {I};
14 Вернуть C_{i+1}.
```

Рис. 4: *GenNext*: Алгоритм генерации кандидатов в набор зависимостей включения размера i+1

На Рисунке 4 представлен алгоритм $GenNext$, который позволяет сгенерировать кандидаты в набор зависимостей включения размера i+1. Этот алгоритм состоит из двух частей:

- Первая часть (строчки 1-9) — шаг генерации.
- Вторая часть (строчки 10-13) — шаг отсечения.

Шаг генерации собирает кандидатов из зависимостей зависимостей, которые выполняются в базе данных. Рассмотрим пример: пусть $I_1 = R_i[XA] \subseteq R_j[YC]$ и $I_2 = R_i[XB] \subseteq R_j[YD]$ — две выполняющиеся зависимости включения размера i, где $|X| = |Y| = i - 1$, $A < B$ и $C \neq D$. Тогда сгенерируется $I_3 = R_i[XAB] \subseteq R_j[YCD]$.

Шаг отсечения отбрасывает кандидаты, в которых есть зависимости включения, которые не выполняются в текущей базе данных. В качестве примера предположим, что у нас есть только две зависимости включения размера 2, которые выполняются: $R[AB] \subseteq S[EF]$ и $R[AC] \subseteq S[EG]$. На шаге генерации кандидатов мы получим $R[ABC] \subseteq S[EFG]$, однако этот кандидат содержит в себе зависимость включения $R[BC] \subseteq S[FG]$ размера 2, которого нет в наборе выполняющихся зависимостей включения, поэтому мы отбрасываем этот кандидат.

Так же могут быть полезны повторения атрибутов в правой и левой частях [12]. Для того, чтобы разрешить повторения в левой части, необходимо заменить знак $<$ на $\leq$ в строчке 6. Для того, чтобы разрешить повторения в правой части, надо убрать строчку 7.

3.4. Алгоритм MIND

На Рисунке 5 представлен алгоритм для поиска покрытий, выполняющихся в базе данных **d**, MIND. Этот алгоритм принимает в качестве входных данных набор унарных зависимостей включения, генерирует кандидатов размера 2 и проверяет их на базе данных. Из тех, что выполняются, генерируются кандидаты размера 3 и новые кандидаты проверяются на базе данных. Этот процесс продолжается до тех пор, пока не получится сгенерировать новых кандидатов.

// Вводные данные: база данных $\mathbf{d}$, $\mathcal{I}_i$ --- набор унарных зависимостей включения, выполняющихся в $\mathbf{d}$.
// Выходные данные: Зависимости включения, выполняющиеся в $\mathbf{d}$.
1 $\mathcal{C}_2 \coloneqq GenNext(\mathcal{I}_1)$;
2 i $\coloneqq$ 2;
3 Пока $\mathcal{C}_i \neq \varnothing$:
4 Для каждого $I \in \mathcal{C}_i$:
5 Если $\mathbf{d} \models I$, тогда:
6 $\mathcal{I}_i \coloneqq \mathcal{I}_i \cup \{I\}$;
7 $\mathcal{C}_{i+1} \coloneqq GenNext(\mathcal{I}_i)$;
8 $i \coloneqq i + 1$;
9 Вернуть $\bigcup_{j<i} \mathcal{I}_j$.

Рис. 5: Алгоритм поиска покрытия MIND

Теоретическая сложность алгоритма равняется затратам на одну проверку на базе данных умноженную на количество выполняющихся зависимостей включения плюс количество не выполняющихся зависимостей включения, для которых выполняется отношение $\prec$.

3.5. Модификация алгоритмов для поиска зависимостей приближённого включения

Унарные зависимости приближённого включения могут быть найдены с помощью тех же данных, которые используются для поиска точных унарных зависимостей включения. Для этого необходимо посчитать контекст извлечения, дальше можно воспользоваться характеристикой унарных зависимостей приближённого включения:

Если у нас есть заданная погрешность ϵ, база данных $\mathbf{d}$, тип данных t и соответствующий контекст извлечения $\mathbb{D}_t(\mathbf{d}) = (\mathbb{U}, \mathbb{V}, \mathbb{B})$, тогда:

// Вводные данные: контекст извлечения $(\mathcal{V},\mathcal{U},\mathcal{B})$ над базой данных $\mathbf{d}$ и типом t, погрешность ϵ.
// Выходные данные: $\mathcal{AI}_1$ --- набор зависимостей приближённого включения, которые выполняются в $\mathbf{d}$ между атрибутами типа t.
01 Для каждого $A \in \mathbb{U}$: $rhs(A) = \{< B, 0 > \mid B \in \mathbb{U}\}$;
02 Для каждого $v \in \mathbb{V}$:
03 Для каждого A такого, что $(v, A) \in \mathbb{B}$:
04 Для каждого $< B, n_{AB} > \in rhs(A)$ таких, что $(v, B) \in \mathbb{B}$:
05 $n_{AB} = n_{AB} + 1$;
06 Для каждого $A \in \mathbb{U}$:
07 Для каждого $< B, n_{AB} > \in rhs(A) \setminus \{< A, n_{AA} >\}$:
08 Если $1 - (n_{AB}/n_{AA}) \leq \epsilon$:
09 $\mathcal{AI}_1 = \mathcal{AI}_1 \cup \{A \subseteq B\}$;
10 Вернуть $\mathcal{AI}_1$.

Рис. 6: Алгоритм поиска унарных зависимостей приближённого включения

$$\mathbf{d} \vDash_\epsilon A \subseteq B$$
$$\Longleftrightarrow$$
$$1 - \sum_{v \in \mathbb{V} | (v,A) \in \mathbb{B}} \frac{|\{v \in \mathbb{V} | (v, A) \in \mathbb{B} \text{ и } (v, B) \in \mathbb{B}\}|}{|\{v \in \mathbb{V} | (v, A) \in \mathbb{B}\}|} \leq \epsilon$$

где $A, B \in \mathbb{U}$.

Благодаря этой характеристике, выполняющиеся зависимости приближённого включения могут быть найдены за один проход. Алгоритм на Рисунке 6 позволяет найти унарные зависимости приближённого включения для заданного типа t и погрешности ϵ. Этот алгоритм похож на алгоритм поиска точных зависимостей включения. Для каждого атрибута A мы вычисляем множество $rhs(A) = \{< B, n_{AB} >\}$, где B — атрибут того же типа, что и A, и n_{AB} — число строк в контексте извлечения, в которых A и B входят вместе.

Сложность алгоритма, представленного на Рисунке 6, близка к сложности алгоритма, представленного на Рисунке 3. Единственные накладные расходы, это необходимость подсчитывать количество атрибутов,

которых вместе входят в контекст извлечения.

Рассмотрим работу алгоритма, представленного на Рисунке 6, на примере в Рисунке 1 над вещественными числами.

Инициализируем $rhs(C) = rhs(F) = \{< C, 0 >, < F, 0 >\}$.

Дальше рассмотрим первую строчку контекста извлечения $l_1 = \{C, F\}$ в цикле на второй строчке:

- $rhs(C) = \{< C, 1 >, < F, 1 >\}$
- $rhs(F) = \{< C, 1 >, < F, 1 >\}$

Для второй строчки контекста извлечения $l_2 = \{C, F\}$ получаем:

- $rhs(C) = \{< C, 2 >, < F, 2 >\}$
- $rhs(F) = \{< C, 2 >, < F, 2 >\}$

Для третьей строчки контекста извлечения $l_3 = \{C\}$ получаем:

- $rhs(C) = \{< C, 3 >, < F, 2 >\}$
- $rhs(F) = \{< C, 2 >, < F, 2 >\}$

Для последней строчки контекста извлечения $l_4 = \{F\}$ получаем:

- $rhs(C) = \{< C, 3 >, < F, 2 >\}$
- $rhs(F) = \{< C, 2 >, < F, 3 >\}$

Из этих наборов мы можем посчитать метрику g_3' для каждой унарной зависимости приближённого включения. Например: $g_3'(F \subseteq C, \mathbf{d}) = 1 - \frac{2}{3}$. На выходе получается набор зависимостей приближённого включения, посчитанная погрешность которых меньше или равна заданной погрешности.

N-арные зависимости приближённого включения размера 2 и больше также могут быть найдены с помощью уровней.

Пусть I_1, I_2 — 2 кандидата в набор зависимостей включения таких, что $I_1 \preceq I_2$. Если $\mathbf{d} \not\models_\epsilon I_1$, тогда $\mathbf{d} \not\models_\epsilon I_2$.

Это утверждение следует из соображения, что если I_1 и I_2 — два кандидата в набор зависимостей включения в базе данных $\mathbf{d}$, то из $I_1 \preceq I_2$ следует $g'_3(I_1, \mathbf{d}) \leq g'_3(I_2, \mathbf{d})$, что очевидно из определения g'_3 и отношения $\preceq$.

Таким образом мы можем применить алгоритм, представленный на Рисунке 5, с небольшими изменениями, а именно добавив погрешность во входные данные и заменив $\mathbf{d} \models I$ на $\mathbf{d} \models_\epsilon I$ на пятой строчке.

Заключение

В результате работы были выполнены следующие задачи:

1. составлен обзор предметной область включающих зависимостей;
2. изучен принцип работы алгоритма MIND для поиска зависимостей включения и изложен принцип модификации этого алгоритма для поиска зависимостей приближённого включения;

Список литературы

[1] Bauckmann J., Leser U., Naumann F. Efficient and Exact Computation of Inclusion Dependencies for Data Integration. Technische Berichte des Hasso-Plattner-Instituts für Softwaresystemtechnik an der Universität Potsdam. — Univ.-Verlag, 2010. — ISBN: 9783869560489. — URL: https://books.google.ru/books?id=A2UqdXlpVOEC.

[2] Casanova Marco A., Fagin Ronald, Papadimitriou Christos H. Inclusion dependencies and their interaction with functional dependencies // Journal of Computer and System Sciences. — 1984. — Vol. 28, no. 1. — P. 29–59. — URL: https://www.sciencedirect.com/science/article/pii/0022000084900758.

[3] De Marchi F., Petit J.-M. Zigzag: a new algorithm for mining large inclusion dependencies in databases // Third IEEE International Conference on Data Mining. — 2003. — P. 27–34.

[4] Divide & Conquer-Based Inclusion Dependency Discovery / Thorsten Papenbrock, Sebastian Kruse, Jorge-Arnulfo Quiané-Ruiz, Felix Naumann // Proc. VLDB Endow. — 2015. — feb. — Vol. 8, no. 7. — P. 774–785. — URL: https://doi.org/10.14778/2752939.2752946.

[5] Fast Approximate Discovery of Inclusion Dependencies / Sebastian Kruse, Thorsten Papenbrock, Christian Dullweber et al. // Datenbanksysteme fur Business, Technologie und Web (BTW 2017), 17. Fachtagung des GI-Fachbereichs „Datenbanken und Informationssysteme (DBIS), 6.-10. Marz 2017, Stuttgart, Germany, Proceedings / Ed. by Bernhard Mitschang, Daniela Nicklas, Frank Leymann et al. — Vol. P-265 of LNI. — GI, 2017. — P. 207–226. — URL: https://dl.gi.de/20.500.12116/629.

[6] Inclusion Dependency Discovery: An Experimental Evaluation of Thirteen Algorithms / Falco Dürsch, Axel Stebner, Fabian Windheuser et al. // Proceedings of the 28th ACM International Conference on

Information and Knowledge Management. — CIKM '19. — New York, NY, USA : Association for Computing Machinery, 2019. — P. 219–228. — URL: https://doi.org/10.1145/3357384.3357916.

[7] Kivinen Jyrki, Mannila Heikki. Approximate inference of functional dependencies from relations // Theoretical Computer Science. — 1995. — Vol. 149, no. 1. — P. 129–149. — Fourth International Conference on Database Theory (ICDT '92). URL: https://www.sciencedirect.com/science/article/pii/030439759500028U.

[8] Koeller A., Rundensteiner E.A. Discovery of high-dimensional inclusion dependencies // Proceedings 19th International Conference on Data Engineering (Cat. No.03CH37405). — 2003. — P. 683–685.

[9] Kruse Sebastian, Papenbrock Thorsten, Naumann Felix. Scaling out the discovery of inclusion dependencies // Datenbanksysteme für Business, Technologie und Web (BTW 2015). — 2015.

[10] Lopes Stéphane, Petit Jean-Marc, Toumani Farouk. Discovering Interesting Inclusion Dependencies: Application to Logical Database Tuning // Inf. Syst. — 2002. — mar. — Vol. 27, no. 1. — P. 1–19. — URL: https://doi.org/10.1016/S0306-4379(01)00027-8.

[11] Marchi Fabien De, Lopes Stéphane, Petit Jean-Marc. Unary and n-ary inclusion dependency discovery in relational databases // Journal of Intelligent Information Systems. — 2009. — Feb. — Vol. 32, no. 1. — P. 53–73. — URL: https://doi.org/10.1007/s10844-007-0048-x.

[12] Mitchell John C. The implication problem for functional and inclusion dependencies // Information and Control. — 1983. — Vol. 56, no. 3. — P. 154–173. — URL: https://www.sciencedirect.com/science/article/pii/S0019995883800023.

[13] Shaabani Nuhad, Meinel Christoph. Scalable Inclusion Dependency Discovery // Database Systems for Advanced Applications /

Ed. by Matthias Renz, Cyrus Shahabi, Xiaofang Zhou, Muhammad Aamir Cheema. — Cham : Springer International Publishing, 2015. — P. 425–440.

[14] Shaabani Nuhad, Meinel Christoph. Detecting Maximum Inclusion Dependencies without Candidate Generation // Database and Expert Systems Applications / Ed. by Sven Hartmann, Hui Ma. — Cham : Springer International Publishing, 2016. — P. 118–133.

[15] Tschirschnitz Fabian, Papenbrock Thorsten, Naumann Felix. Detecting Inclusion Dependencies on Very Many Tables // ACM Trans. Database Syst. — 2017. — jul. — Vol. 42, no. 3. — 29 p. — URL: https://doi.org/10.1145/3105959.