Санкт-Петербургский государственный университет

Кафедра информационно-аналитических систем

Группа 20.Б10-мм

Рахмукова Анна Игоревна

Реализация алгоритма поиска функциональных зависимостей AID-FD

Отчёт по учебной практике
в форме «Решение»

Научный руководитель:
ассистент кафедры ИАС Чернышев Г.А.

Санкт-Петербург
2022

Оглавление

Введение

Функциональные зависимости (далее ФЗ) описывают связь между атрибутами заданного реляционного отношения. Эта информация о данных бывает полезна при очистке данных, нормализации данных, оптимизации запросов, интеграции данных и других задачах. В настоящий момент существуют алгоритмы, решающие проблему поиска указанных закономерностей в массивах данных.

Приложением, в котором собраны различные алгоритмы для поиска ФЗ и предоставлена функциональность для работы с ними, стала платформа Metanome [2]. Проект реализован на языке программирования Java, что вызывает проблемы, связанные с производительностью и низкоуровневой оптимизацией, подробно описанные в статье [4]. Для устранения этих недостатков ведется работа над проектом Desbordante [4] — платформы для профилирования данных, написанной на языке C++.

Независимо от реализации, по-прежнему остаётся актуальной проблема производительности алгоритмов поиска ФЗ. Алгоритмам необходимо обрабатывать большие массивы данных, и даже самые эффективные из них тратят много времени на извлечение всех ФЗ рассматриваемого отношения.

Описанную проблему можно частично решить, если использовать приближённые алгоритмы поиска ФЗ наряду с точными. Алгоритмы, которые найдут часть закономерностей в наборах данных или добавят в результат невалидные зависимости, могут обладать лучшей скоростью. Примером приближённого алгоритма служит AID-FD (Approximate Iterative Discovery of FDs) [1]. В данной работе будет рассмотрена реализация этого алгоритма на языке C++ в рамках проекта Desbordante.

1. Постановка задачи

Целью данной работы является реализация алгоритма поиска функциональных зависимостей AID-FD на платформе Desbordante на языке программирования C++. Для её выполнения были поставлены следующие задачи:

- Провести обзор алгоритма AID-FD;
- Реализовать алгоритм;
- Выполнить анализ производительности алгоритма.

2. Обзор

2.1. Основные определения

Далее даны несколько определений, необходимых для понимания работы алгоритма.

1. **Функциональная зависимость (ФЗ)**: Рассмотрим отношение со множеством атрибутов R и множеством кортежей r, $X \subseteq R, A \in R$. Говорят, что выполняется функциональная зависимость $X \to A$, если $\forall t_1, t_2 \in r\ t_1[X] = t_2[X] \Rightarrow t_1[A] = t_2[A]$;

2. **Левая сторона ФЗ (Left hand side)**: X в $X \to A$;

3. **Правая сторона ФЗ (Right hand side)**: A в $X \to A$;

4. **Минимальность ФЗ**: $X \to A$ минимальна, если $\nexists X' \subset X$ такого, что $X' \to A$ — ФЗ;

5. **Нетривиальность ФЗ**: $X \to A$ нетривиальна, если $A \notin X$;

6. **Золотой стандарт (Gold) [1]**: множество всех минимальных нетривиальных ФЗ заданного отношения;

7. **Набор согласованности (Agree Set) кортежей** t_1, t_2: максимальное по включению множество $Y \subseteq R$ такое, что $t_1[Y] = t_2[Y]$. Обозначение: $ag(t_1, t_2)$;

8. **Негативное покрытие**: множество наборов согласованности [1];

9. **Позитивное покрытие**: множество минимальных нетривиальных ФЗ.

2.2. Описание алгоритма

Рассматриваемый алгоритм состоит из трёх этапов: индексация атрибутов, создание негативного покрытия и его инвертирование.

2.2.1. Этап 1: индексация атрибутов

На первом этапе алгоритм разбивает каждый столбец таблицы на кластеры — упорядоченные множества индексов строк, для которых значения элементов в данном столбце совпадают. При этом создается таблица, в которой для каждой ячейки исходной таблицы хранится её индекс в кластере.

Помимо этого, на данном этапе алгоритма выделяются константные столбцы, которые могут быть исключены из рассмотрения на следующих этапах.

Algorithm 1: Build negative cover.

```
Data: A relation schema R and a relation instance r
Result: A negative cover 𝒩
𝒩 = ∅
i ← 1
while Termination criterion not met do
    for t ∈ r do
        make_ith_checks(t, i)
    i ← i + 1
return 𝒩

Function make_ith_checks(t, i)
    for a ∈ R do
        cluster ← clusters[t, a]
        index ← indices[t, a]
        if i ≤ index then
            other_cluster_index ← prp(index, i)
            t' ← cluster[other_cluster_index]
            𝒩 ← 𝒩 ∪ ag(t, t')
```

Рис. 1: Построение негативного покрытия, рисунок взят из статьи [1]

2.2.2. Этап 2: построение негативного покрытия

Псевдокод данного этапа алгоритма приведён на Рис. 1. В начале создаётся пустое негативное покрытие, и далее перебираются пары строк, для которых вычисляется набор согласованности и добавляет-

ся во множество. Процесс завершается, когда встречается критерий завершения. В статье [1] в качестве возможных критериев завершения построения были предложены фиксированное время и достижение фиксированных темпов роста негативного покрытия на последней или нескольких последних итерациях.

Рассмотрим процесс подбора пар строк. На итерации i для кортежа t в каждом из столбцов подбирается другой кортеж, находящийся в том же кластере. Для выбора псевдослучайного индекса используется функция $prp(index, i) = (i \cdot prime)\ mod\ index$, где $index$ — индекс строки в кластере, а $prime$ — большое простое число. Данная функция для каждого фиксированного индекса с ростом итераций перебирает все индексы, меньшие данного. Когда номер итерации превышает номер индекса, текущий столбец может быть пропущен, поскольку все меньшие индексы уже перебраны (строка 13).

Algorithm 2: Phase 2: Negative cover inversion.

Data: negative FD cover $\mathcal{N}$, set of attributes R, set of constant columns S
Result: the set of minimal, non-trivial FDs Ω

```
1  Ω ← {∅ → const | const ∈ S}
2  R ← R \ S
3  N ← sort(N)
4  for rhs ∈ R do
5      P ← {{attr} | attr ∈ R, attr ≠ rhs}
6      for N ∈ N do
7          if rhs ∉ N then
8              handleNonFD (N, P, R \ {rhs})
9      Ω ← Ω ∪ {L → rhs | L ∈ P}
10 return Ω
11
12 Function handleNonFD(N, P, X)
13     S ← {P ∈ P | P ⊆ N}
14     P ← P \ S
15     for L ∈ S do
16         for add ∈ (X \ N) do
17             if ∀P ∈ P: P ⊈ (L ∪ {add}) then
18                 P ← P ∪ {L ∪ {add}}
```

Рис. 2: Инвертирование негативного покрытия, рисунок взят из статьи [1]

2.2.3. Этап 3: инвертирование негативного покрытия

На этапе инвертирования негативного покрытия для каждого не константного столбца rhs подбирается множество возможных левых сторон L таких, что $L \to rhs$ — минимальная нетривиальная ФЗ. Объединение данных множеств по всем правым сторонам и является результатом работы алгоритма. Псевдокод представлен на Рис. 2.

Чтобы понять процесс выбора левых сторон L, сперва рассмотрим следующее свойство. Пусть N — элемент негативного покрытия (набор согласованности некоторых кортежей), тогда если $X \subseteq N, A \notin N$, то $X \to A$ — не ФЗ.

Чтобы гарантировать нетривиальность ФЗ, обрабатываются только те элементы негативного покрытия, которые не содержат данную правую сторону. Из всех подмножеств негативного покрытия, принадлежащих позитивному покрытию, образуем множество S. Далее из множества потенциальных левых сторон удаляются все элементы S, которые не могут быть ФЗ, согласно описанному выше свойству. В качестве кандидата на левую сторону рассматривается $L \cup add$, где $L \in S$, $add \notin N$ и $add \neq rhs$. Данное объединение уже не содержится в N, а значит может быть левой стороной ФЗ. Проверка того, что в текущем позитивном покрытии отсутствуют подмножества добавляемой левой стороны, обеспечивает минимальность данной ФЗ (строка 17).

2.3. Метрики оценивания алгоритма

Поскольку AID-FD предлагает приближённый результат, нужно определить, каким образом оценивать точность рассматриваемого алгоритма. Авторами статьи [1] для этой цели были предложены следующие метрики:

1. **Полнота (Completeness):** Отношение числа найденных алгоритмом минимальных ФЗ к общему числу минимальных ФЗ

$$\frac{|Out \cap Gold|}{|Gold|}$$

2. **Корректность (Correctness):** Отношение числа найденных алгоритмом ФЗ к общему числу найденных ФЗ

$$\frac{|Out \cap Gold^+|}{|Out|}$$

3. **Минимальность (Minimality):** Отношение числа найденных алгоритмом минимальных ФЗ к общему числу найденных алгоритмом ФЗ

$$\frac{|Out \cap Gold|}{|Out \cap Gold^+|}$$

$Gold^+$ — обозначение для множества всех ФЗ, которые могут быть получены из Gold с помощью правил вывода Армстронга [7].

2.4. Реализация в Metanome

В реализации алгоритма AID-FD в Metanome допускается использование различных критериев завершения инвертирования негативного покрытия, упомянутых ранее, и также фиксированное количество итераций.

Также, не описывается авторами статьи [1], но представлено в реализации алгоритма: осуществление проверки полученных ФЗ на корректность; использование фильтра Блума [8] на втором этапе работы алгоритма для того, чтобы избежать повторного рассмотрения пар кортежей [6].

3. Реализация

3.1. Особенности реализации алгоритма

При реализации алгоритма было принято решение придерживаться псевдокода оригинальной статьи, приведённого выше. Алгоритм представлен классом Aid, наследующимся от существующего класса FDAlgorithm [3]. Также реализована структура данных SearchTree, подробно описанная ниже. Код написан на языке C++ стандарта 17, в упомянутой структуре данных используются умные указатели из стандартной библиотеки, множества атрибутов в коде представлены с помощью динамических битсетов из библиотеки boost.

В настоящее время в качестве критерия завершения второго этапа в работе алгоритма используется достижение фиксированного роста негативного покрытия на десяти последних итерациях. Порог роста негативного покрытия при этом составляет 0.01. Данные значения были выбраны, основываясь на результатах нескольких сравнений скорости и точности алгоритма на различных наборах данных. В дальнейшем планируется расширить множество возможных критериев завершения и предоставить пользователю возможность самостоятельно устанавливать значения каждого из параметров.

3.2. Используемые структуры

Для оптимизации поиска подмножеств и надмножеств заданного множества атрибутов используется структура данных SearchTree, упомянутая авторами статьи [1]. Данная структура представляет бинарное дерево, обладающее следующими характеристиками:

1. В листьях дерева находятся хранимые множества атрибутов;

2. Внутренние узлы хранят указатели номер атрибута, по которому происходит разделение множеств на две части. Множества, находящиеся в правом поддереве узла с заданным номером атрибута,

содержат данный атрибут, множества в левом поддереве — не содержат;

3. Для каждого внутреннего узла, кроме корня, выполняется следующее свойство: номер разделяющего атрибута родителя строго больше номера разделяющего атрибута, хранящегося в рассматриваемом узле;

4. Кроме того, в каждом внутреннем узле хранятся объединение и пересечение всех множеств, находящихся в поддереве, корнем которого является данный узел. Это помогает отсекать заведомо неподходящие ветви при поиске множеств в дереве (например, при поиске подмножеств некоторого множества A пересечение всех множеств в данном поддереве должно содержаться в A).

Перед построением данной структуры данных в алгоритме атрибуты сортируются в порядке возрастания частоты появления каждого в негативном покрытии. Согласно результатам, приведённым в статье, этот приём уменьшает время инвертирования негативного покрытия (Рис. 3).

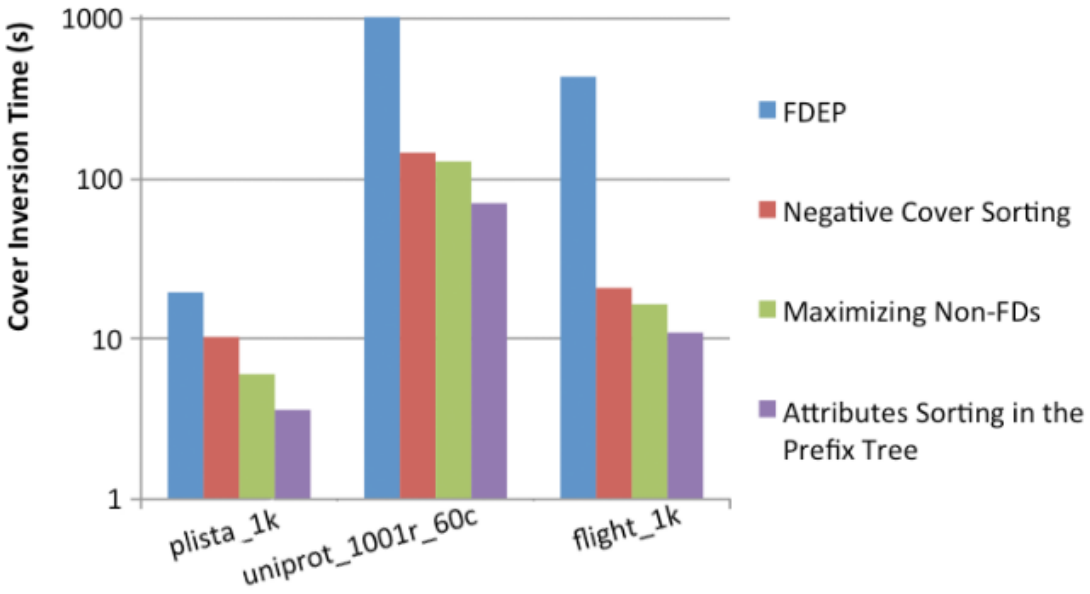

Рис. 3: Время инвертирования негативного покрытия, рисунок взят из статьи [1]

Стоит упомянуть и другие оптимизации, представленные на графике выше: сортировка элементов негативного покрытия в убывающем порядке относительно мощностей элементов и максимизация невалидных ФЗ. Обе из них также использованы в реализации алгоритма.

Кроме того, в отличие от реализации в Metanome, каждый узел дерева, кроме корня, хранит указатель на родительский узел, что позволяет реализовать часть операций в дереве с использованием итераций вместо рекурсии.

4. Эксперименты

4.1. Условия эксперимента

Характеристики системы, на которой проводились эксперименты: AMD Ryzen 7 3700U CPU @ 2.30GHz × 4, 8 GiB RAM, Ubuntu 20.04.2 LTS. Metanome запускался на OpenJDK версии 11.0.16. При сравнении алгоритмов было проведено 10 запусков каждого из них на различных наборах данных. Эксперименты проводились на незагруженной системе для того, чтобы уменьшить влияние операционной системы на результаты.

Ниже представлена информация о наборах данных, которые использовались в ходе эксперимента.

Данные	Атрибуты	Строки	Размер [KB]	Число ФЗ
iris	5	150	5	4
balance-scale	5	625	7	1
chess	7	28,056	519	1
abalone	9	4177	187	137
nursery	9	12960	1024	1
breast-cancer-wisconsin	11	699	20	46
bridges	13	108	6	142
echocardiogram	13	132	6	538
adult	14	48842	3528	78
letter	16	20000	695	61
ncvoter_1001r_19c	19	1000	151	758
hepatitis	20	155	8	8296
horse	27	300	25	128695

4.2. Сравнение с точным алгоритмом

Для сравнения был выбран алгоритм поиска ФЗ FDep [5, 9]. Данный алгоритм относится к группе индуктивных алгоритмов, рост времени работы которых, как и AID-FD, больше зависит от числа кортежей, нежели числа атрибутов. По этой причине сравниваемые алгоритмы могут вести себя похоже на наборах данных с определенным коли-

чеством строк и столбцов. Подробнее о типах алгоритмов для поиска ФЗ можно прочитать в статье [4].

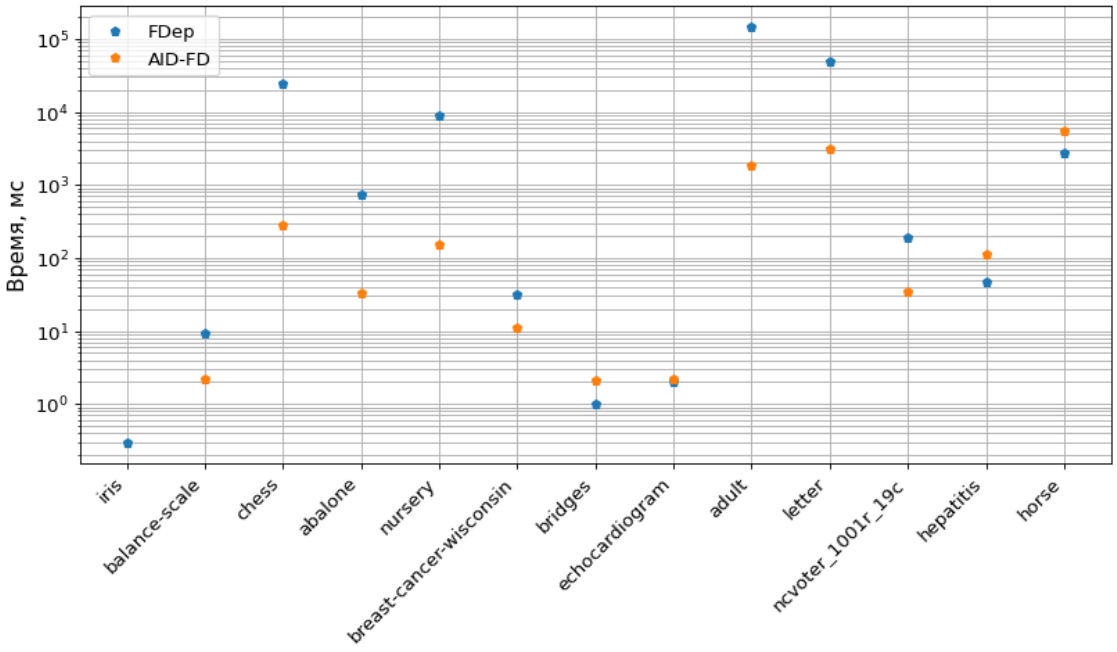

Рис. 4: Скорость работы AID-FD и FDep

Результаты сравнения двух алгоритмов представлены на Рис.4. На 10 из 13 наборов данных AID-FD оказался быстрее FDep. Оставшиеся наборы данных объединяет большое количество ФЗ при сравнительно небольшом количестве строк.

4.3. Сравнение с Metanome

Для проведения эксперимента в реализации алгоритма AID-FD в Metanome были установлены те же параметры, что используются в реализации в Desbordante. Кроме того, из кода были удалены все выводы промежуточных результатов на консоль, так как это дорогостоящая операция, которая может повлиять на время выполнения алгоритма.

Результаты представлены на Рис. 5. На данном этапе работы реализация алгоритма в Metanome показывает лучшие результаты, чем реализация в Desbordante.

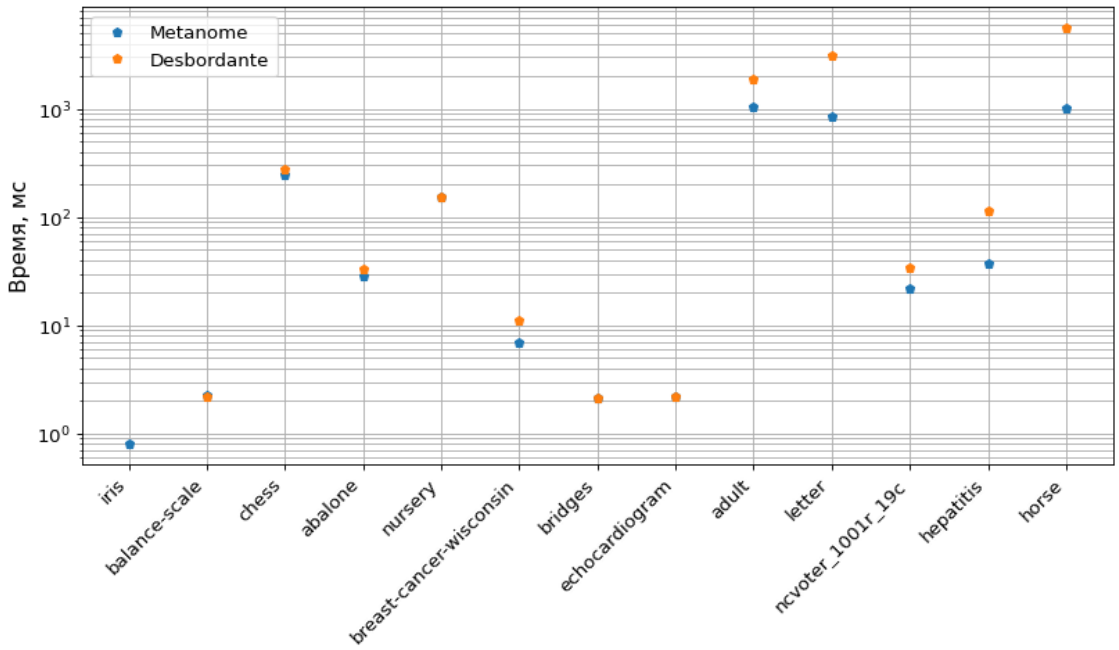

Рис. 5: Скорость работы реализаций Desbordante и Metanome

4.4. Анализ точности

Из трех приведённых выше метрик были выбраны полнота и корректность для оценивания точности полученного алгоритма.

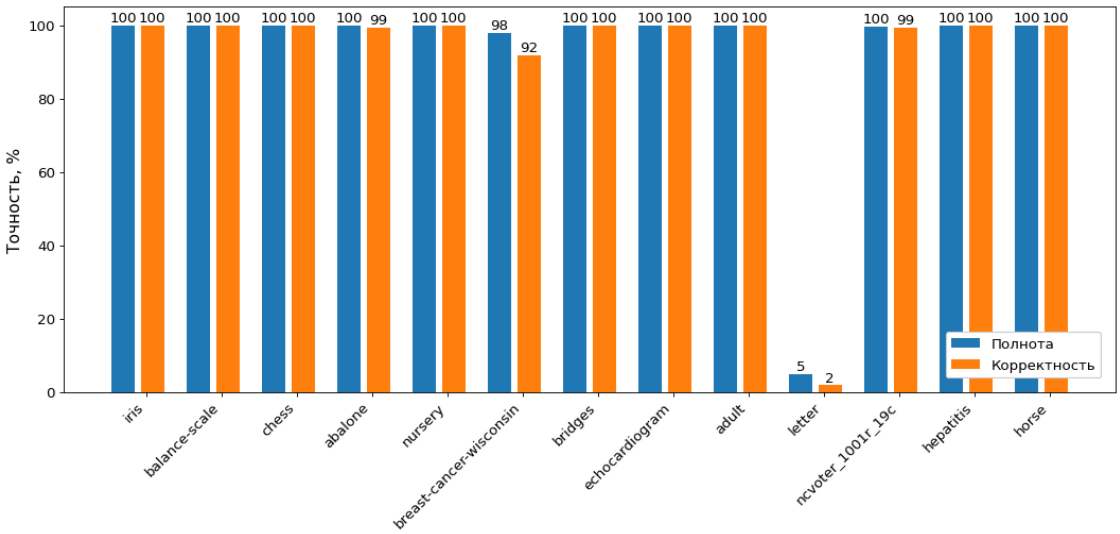

Рис. 6: Полнота и корректность

На Рис. 6 видно, что на всех наборах данных, кроме letter, алгоритм демонстрирует достаточно высокую точность (не менее 98% для полноты и 92% для корректности).

Заключение

В результате работы были решены следующие задачи:

- Проведен обзор алгоритма AID-FD;
- Реализован алгоритм на языке C++ в рамках платформы Desbordante (исходный код доступен на GitHub [3], пользователь rakhmukova);
- Выполнен анализ производительности алгоритма.

В дальнейшей работе планируется оптимизация реализованного алгоритма для достижения лучшей производительности. Также, как указывалось ранее, будет расширен список возможных критериев завершения второго этапа работы AID-FD, что обеспечит большую гибкость в использовании алгоритма.

Список литературы

[1] Approximate Discovery of Functional Dependencies for Large Datasets / Tobias Bleifuß, Susanne Bülow, Johannes Frohnhofen et al. // Proceedings of the 25th ACM International on Conference on Information and Knowledge Management. — CIKM '16. — New York, NY, USA : Association for Computing Machinery, 2016. — P. 1803–1812. — URL: https://doi.org/10.1145/2983323.2983781.

[2] Data Profiling with Metanome / Thorsten Papenbrock, Tanja Bergmann, Moritz Finke et al. // Proc. VLDB Endow. — 2015. — aug. — Vol. 8, no. 12. — P. 1860–1863. — URL: https://doi.org/10.14778/2824032.2824086.

[3] Desbordante. — https://github.com/Mstrutov/Desbordante.

[4] Desbordante: a Framework for Exploring Limits of Dependency Discovery Algorithms / Maxim Strutovskiy, Nikita Bobrov, Kirill Smirnov, George Chernishev // 2021 29th Conference of Open Innovations Association (FRUCT). — 2021. — P. 344–354.

[5] Flach Peter A., Savnik Iztok. Database Dependency Discovery: A Machine Learning Approach // AI Commun. — 1999. — aug. — Vol. 12, no. 3. — P. 139–160.

[6] Metanome Algorithms. — https://github.com/HPI-Information-Systems/metanome-algorithms.

[7] Silberschatz Abraham, Korth Henry F, Sudarshan S. Database Systems Concepts. — 5 edition. — Maidenhead, England : McGraw Hill Higher Education, 2005.

[8] Wikipedia contributors. Bloom filter — Wikipedia, The Free Encyclopedia. — https://en.wikipedia.org/w/index.php?title=Bloom_filter&oldid=1104716059. — 2022. — [Online; accessed 25-October-2022].

[9] Кристофович Салью Артур. Высокопроизводительный поиск функциональных зависимостей в данных. — 2022. — URL: https://oops.math.spbu.ru/SE/YearlyProjects/spring-2021/uchebnye-praktiki/pi-2/Saliou-report.pdf/view (online; accessed: 2022-10-25).