Санкт-Петербургский государственный университет

Баруткин Илья Дмитриевич

Выпускная квалификационная работа

Реализация алгоритма проверки вероятностных функциональных зависимостей в профилировщике данных Desbordante

Уровень образования: бакалавриат

Направление *02.03.03 «Математическое обеспечение и администрирование информационных систем»*

Основная образовательная программа *СВ.5162.2020 «Технологии программирования»*

Научный руководитель:
доцент кафедры информационно-аналитических систем, к.ф.-м.н. Михайлова Е. Г.

Консультант:
ассистент кафедры информационно-аналитических систем Чернышев Г. А.

Рецензент:
программист-разработчик ООО «В Контакте» Слесарев А. Г.

Санкт-Петербург
2024

Saint Petersburg State University

Ilia Barutkin

Bachelor's Thesis

Implementation of probabilistic functional dependency validation algorithm in the Desbordante data profiler

Education level: bachelor

Speciality *02.03.03 "Software and Administration of Information Systems"*

Programme *CB.5162.2020 "Programming Technologies"*

Scientific supervisor:
C.Sc., docent E. G. Mikhailova

Consultant:
Assistant G. A. Chernishev

Reviewer:
Software developer at «V Kontakte» A. G. Slesarev

Saint Petersburg
2024

Оглавление

Введение

Профилирование данных [4] направлено на извлечение метаданных из данных. Самым известным примером метаданных является простая статистика: количество строк и пропущенных значений, дисперсия и математическое ожидание. Существует множество инструментов поиска простых статистик [20, 26]. С другой стороны, среди метаданных есть и более сложные закономерности, представленные структурами, которые называются примитивами [3]. Профилирование, нацеленное на поиск вхождений этих примитивов, называется наукоемким.

Примерами примитивов являются различные зависимости в базах данных (функциональные зависимости [11], зависимости включения [14]), ассоциативные правила [1], алгебраические ограничения [2] и другие. Такие примитивы имеют множество применений. Например, можно использовать примитивы для очистки данных от ошибок, поиска неточных дубликатов и аномалий [16]. Некоторые примитивы получили приложения в задаче нормализации схем в базах данных [12]. Наконец, в научных данных найденные вхождения примитивов могут указывать на наличие некоторой закономерности [9], которая может способствовать формулированию гипотезы.

Одним из широко известных примитивов является функциональная зависимость. Отношение R удовлетворяет функциональной зависимости $X \to Y \iff \forall t_1, t_2 \in R \quad t_1[X] = t_2[X] \implies t_1[Y] = t_2[Y]$, где X, Y это наборы столбцов. Иначе говоря, функциональная зависимость означает, что если две записи таблицы равны по атрибутам X, то они должны быть равны и по атрибутам Y.

Однако реальные данные часто содержат несоответствия, пропущенные значения и другие дефекты. Следовательно, точные зависимости редко сохраняются в таких данных, и алгоритм поиска не сможет их обнаружить. Поэтому были предложены различные подходы к ослаблению определения функциональной зависимости.

Для функциональных зависимостей существует несколько приближенных вариантов, допускающих некоторое количество строк, наруша-

ющих зависимость. Такие определения функциональной зависимости включают в себя различные формулы ошибки, построенные на семействе метрик g_1, g_2, g_3 [15]. Наиболее известным вариантом являются приближенные функциональные зависимости [17], основанные на модифицированной метрике g_1. Одной из альтернатив являются вероятностные функциональные зависимости [12, 7], которые применялись в задачах очистки данных и нормализации схем в системах интеграции данных.

Извлечение и проверка функциональных зависимостей требует больших вычислительных затрат. Поэтому востребованы сложные алгоритмы и их эффективные реализации. В настоящее время существуют два наукоемких профилировщика данных — Metanome [5] и Desbordante.

Desbordante [3] — это *наукоемкий*, *высокопроизводительный* инструмент профилирования данных с *открытым исходным кодом*, реализованный на языке C++. Он способен обнаруживать и проверять множество примитивов, включая точные и приближенные (на основе модифицированной метрики g_1) функциональные зависимости.

Проект Desbordante активно развивается и стремится расширить набор поддерживаемых примитивов, среди которых и вероятностные функциональные зависимости. Это послужило мотивацией к задаче по их исследованию и добавлению поддержки их поиска в Desbordante, выполненной в ходе настоящей работы.

1. Постановка задачи

Целью данной дипломной работы является расширение профилировщика данных Desbordante поддержкой поиска вероятностных функциональных зависимостей. Для достижения этой цели были сформулированы следующие задачи:

- провести обзор предметной области;
- исследовать примитив вероятностных функциональных зависимостей;
- реализовать компоненту поиска вероятностных функциональных зависимостей;
- интегрировать компоненту в инструмент Desbordante;
- экспериментально исследовать производительность реализации.

2. Предварительные сведения

2.1. Функциональные зависимости

Пусть дана схема R, тогда функциональная зависимость [24] — это выражение, обозначаемое как $X \to A$, где $X \subseteq R$ и $A \in R$. В настоящей работе множество X также называется левой частью (Left-Hand Side, LHS), и атрибут A правой частью (Right-Hand Side, RHS) зависимости. Зависимость удерживается на отношении r, если для всех пар кортежей $t, u \in r$ выполнено: $\forall B \in X(t[B] = u[B]) \implies t[A] = u[A]$.

На практике не требуется явно обнаруживать все удерживающиеся зависимости, поэтому полезны следующие определения. Функциональная зависимость $X \to A$ называется минимальной [24], если для любого собственного подмножества $X' \subset X$ зависимость $X' \to A$ не удерживается. Далее, функциональная зависимость $X \to A$ называется тривиальной [24], если $A \in X$.

2.2. Ослабленные функциональные зависимости

Существуют различные подходы для смягчения ограничений, налагаемых на функциональные зависимости. Более того, в связанных работах используются одинаковые наименования для различных примитивов: например в определении приближенной функциональной зависимости могут использоваться различные метрики ошибки [24, 17] или же оно может вводиться аксиоматически [13].

В настоящей работе приближенной функциональной зависимостью называется примитив, определенный в соответствующей работе 2018 года “Efficient Discovery of Approximate Dependencies” [17], он широко распространен и используется в инструментах для профилирования данных. Под вероятностной функциональной зависимостью имеется в виду примитив, впервые предложенный в статье 2008 года “Functional Dependency Generation and Applications in Pay-As-You-Go Data Integration Systems” [12].

Приближенная функциональная зависимость (approximate functio-

nal dependency, AFD) обозначается как $X \to A$, где X — множество атрибутов, A — один атрибут в схеме R. Для данной зависимости и некоторого отношения r определяется значение ошибки. В настоящей работе предполагается следующая метрика ошибки, основанная на g_1, если не сказано иное:

$$e(X \to A, r) = \frac{|\{(t_1, t_2) \in r^2 \mid t_1[X] = t_2[X] \wedge t_1[Y] \neq t_2[Y]\}|}{|r|^2 - |r|}.$$

Специализациями данной зависимости называются все зависимости вида $X' \to A$, где $X \subset X'$. Похожим образом, обобщениями называются все зависимости вида $X' \to A$, где $X' \subset X$.

Теперь зафиксируем некоторый порог ϵ, который принимает значения от 0 до 1. Зависимость называется минимальной, если ее значение ошибки не больше ϵ, но для всех ее обобщений значение ошибки превышает этот порог. Используемая в определении AFD метрика e обладает свойством монотонности, то есть, $e(X \to A, r) \geq e(XY \to A, r)$ [17]. Иначе говоря, значение ошибки данной зависимости всегда не больше значения ошибки ее специализаций.

Вероятностная функциональная зависимость (probabilistic functional dependency, pFD) обозначается как $X \xrightarrow{p} A$, где p — вероятность удержания зависимости, которая определяется по следующим правилам. Пусть даны набор атрибутов X, отношение r, и атрибут $A \notin X$, причем, согласно авторам [12], среди значений атрибутов X нет значения *null*. Обозначим класс кортежей с данными значениями X_1 атрибутов из X и не-null значениями атрибута A как $V_{X_1} = \{t \in r \mid t[X] = X_1 \wedge t[A] \neq null\}$. Множество всех не-null значений атрибута X обозначим как D_X.

Дальше, для данного значения X_1 обозначим множество кортежей

$$(V_A, V_{X_1}) = \{t \in r \mid t[X] = X_1 \wedge t[A] = \underset{A_k \in D_A}{argmax}\{|V_{X_1} \cap V_{A_k}|\}\},$$

то есть для кортежей с данным значением атрибута X найдем кортежи с наиболее популярным и не равным null значением атрибута A.

Тогда вероятность зависимости для одного набора значений X_1 ат-

рибутов из X определяется как $Pr(X \to A, V_{X_1}) = \frac{|V_A, V_{X_1}|}{|V_{X_1}|}$.

Вероятность функциональной зависимости между атрибутами X и атрибутом A на отношении r определяется двумя метриками, а именно PerValue и PerTuple [12]:

$$Pr(X \to A, r)_{PerValue} = \frac{\sum_{X \in D_X} Pr(X \to A, V_X)}{|D_X|},$$

$$Pr(X \to A, r)_{PerTuple} = \frac{\sum_{X \in D_X} |V_A, V_X|}{\sum_{X \in D_X} |V_X|}.$$

Зафиксируем теперь порог вероятности τ. Мы говорим, что pFD $X \xrightarrow{p} Y$ является минимальной, если для любого собственного подмножества $X' \subset X$ зависимость $X' \xrightarrow{p} Y$ не удерживается (то есть ее вероятность меньше данного порога τ). pFD называется тривиальной, если $Y \in X$. Для того, чтобы сравнивать в ходе данной работы данный примитив с другим, мы называем значением ошибки вероятностной функциональной зависимости выражение $1 - P(X \to A, r)$, где P — метрика PerValue или PerTuple.

Отметим свойства, которые могут оказаться важны при выборе алгоритма поиска вхождений данного примитива.

Метрика PerTuple, как и e, обладает свойством монотонности, то есть $Pr(X \to A, r)_{PerTuple} \leq Pr(XY \to A, r)_{PerTuple}$. В тоже время, для метрики PerValue это утверждение не выполняется как показывает пример в Таблице 1: $Pr(A \to C, r)_{PerValue} = 0.75$, но $Pr(AB \to C, r)_{PerValue} = 0.(6)$.

Наконец, допустим, что в отношении r нет null-значений. Тогда метрика PerTuple — это практически метрика g_3, только определенная в терминах вероятности: $Pr(X \to A, r)_{PerTuple} = 1 - g_3(X \to A, r)$. Если же null-значение присутствует в правой части зависимости, то метрики вероятности не учитывают их, как видно из таблицы 2: вероятность $Pr(X \to A, r)_{PerTuple} = 1$, хотя зависимость не удерживается как точная.

Таблица 1: Пример

A	B	C
a	d	1
a	c	1
a	d	0
a	c	0
b	d	2
b	d	2
b	d	2

Таблица 2: Пример

X	A
a	1
a	1
a	null
a	null
b	2
b	2
b	2

3. Обзор

3.1. Алгоритмы поиска функциональных зависимостей

3.1.1. Алгоритм TANE

Алгоритм TANE позволяет находить все нетривиальные минимальные приближенные функциональные зависимости (с метрикой g_3), которые удерживаются в данном отношении. Он был представлен в 1999 году в статье "TANE: An Efficient Algorithm for Discovering Functional and Approximate Dependencies" [24]. По нашим данным, почерпнутым в ходе научного поиска, алгоритм TANE стал первым алгоритмом для поиска приближенных функциональных зависимостей.

Пространство поиска алгоритма можно представить в виде решетки (см. Рисунок 1). Элемент решетки представляет собой набор атрибутов $X \subset R$ данной схемы R и соответствует функциональным зависимостям вида $X \backslash \{A\} \to A$. Решетка разделена на уровни, причем i-ый уровень содержит только комбинации атрибутов размера i. Для каждой такой комбинации $X \subset R$ алгоритм хранит множество, определяемое как $C^+(X) = \{A \in R \mid \forall B \in X : X \backslash \{A, B\} \to B \textit{ не удерживается}\}$, элементы которого называются rhs^+-кандидатами. Данное множество используется в различных правилах отсечения и влияет на генерацию последующих уровней решетки.

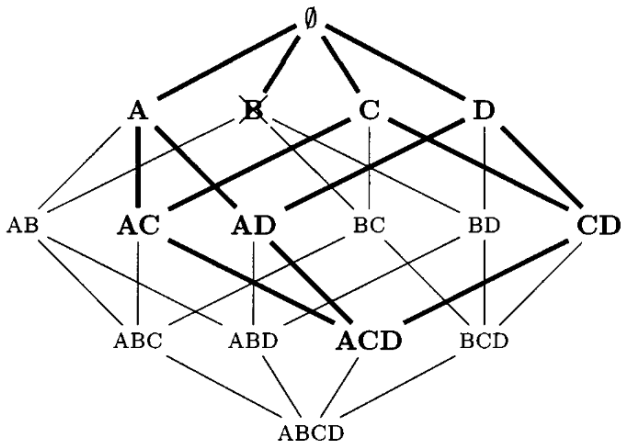


Рис. 1: Пространство поиска алгоритма TANE (взят из оригинальной работы [24])

Для проверки всех потенциальных функциональных зависимостей, алгоритм TANE использует структуру данных, называемую сокращенной партицией (Stripped Partition, SP). Для множества атрибутов X сокращенная партиция $\hat{\pi}_X$ определяется как множество классов эквивалентности кортежей по отношению равенства значений атрибутов X из которого были удалены все одноэлементные множества. В алгоритме используется тот факт, что точная функциональная зависимость $X \to A$ удерживается тогда и только тогда, когда $\hat{\pi}_X$ уточняет $\hat{\pi}_{X \cup \{A\}}$, то есть когда любой класс эквивалентности из $\hat{\pi}_X$ является подмножеством некоторого класса из $\hat{\pi}_{X \cup \{A\}}$. Алгоритм выполняет эту проверку, вычисляя по данной сокращенной партиции $\hat{\pi}_X$ так называемую оценку ошибки $e(X)$, которая определяется как минимальная доля кортежей, которые нужно удалить, чтобы X стал ключем. В итоге, чтобы показать, что точная зависимость $X \to A$ удерживается, достаточно проверить, что $e(X) = e(X \cup \{A\})$. Значение ошибки собственно функциональной зависимости $e(X \to A)$ вычисляется с помощью отдельной процедуры, которая использует сокращенные партиции.

В алгоритме используется три правила, которые позволяют сократить пространство поиска, то есть удалить часть потенциальных зависимостей из рассмотрения. Первое правило позволяет отбрасывать часть rhs^+-кандидатов на основании неминимальности: если функциональная зависимость $X \backslash \{A\} \to A$ удерживается, то из множества $C^+(X)$ можно

удалить атрибут A и все атрибуты $B \in C^+(X) \backslash X$. Второе правило позволяет удалить элемент решетки X, если $C^+(X) = \emptyset$. Наконец, третье правило позволяет удалить элементы решетки, если они соответствуют ключу или суперключу.

В алгоритме TANE правила отсечения приводят к устранению избыточных проверок функциональных зависимостей, которые не являются минимальными по отношению к уже найденным, и только таких. Поэтому корректность этой процедуры алгоритма не зависит от используемой метрики ошибки функциональной зависимости.

3.1.2. Алгоритм Pyro

Как и предыдущий алгоритм, алгоритм Pyro [17] ищет все минимальные нетривиальные приближенные функциональные зависимости, но использует другую метрику ошибки, основанную на метрике g_1. Алгоритм Pyro представлен в 2018 году на конференции по сверхбольшим базам данных (VLDB), спустя почти 20 лет после алгоритма TANE.

Ключевой особенностью алгоритма является его способность оценивать позицию минимальной зависимости в решетке, а также приблизительно оценивать значение ошибки функциональной зависимости без необходимости всегда вычислять ее явно. Более того, для ускорения вычислений алгоритм использует новые, относительно алгоритма TANE, техники и структуры данных. Для эффективного переиспользования сокращенных партиций алгоритм использует кэш на основе префиксного дерева (trie). Для оценки значения ошибки зависимости-кандидата алгоритм вводит структуру данных, называемую выборкой согласованного множества (agree set sample, AS sample).

В алгоритме Pyro общее пространство поиска разделено на множество пространств по числу атрибутов. Каждое пространство поиска соответствует всем функциональным зависимостям-кандидатам с фиксированной правой частью E и может быть представлено в виде решетки. Каждый элемент решетки соответствует одному набору атрибутов $X \subset R$ и функциональной зависимости $X \to E$.

Шаг алгоритма состоит в обходе пространства поиска. Обход мож-

но рассматривать как поочередное движение вверх и вниз по решетке, представляющей данное пространство. Он состоит из нескольких раундов и начинается с основания решетки, где располагаются зависимости с одним атрибутом в левой части.

В примере, представленном на Рисунке 2, алгоритм выбирает вершину A в качестве стартовой точки и поднимается в пространстве поиска до тех пор, пока не обнаружит зависимость $ABCD \rightarrow E$. От этой зависимости, называемой пиком (Peak), алгоритм движется вниз и оценивает положение всех минимальных зависимостей, обобщающих пик. Выбрав, путем такой оценки, элемент CD, он затем пытается подтвердить минимальность соответствующей зависимости. Для этого алгоритм проверяет оставшихся кандидатов, а именно $ABD \rightarrow E$, отсекая часть вершин.

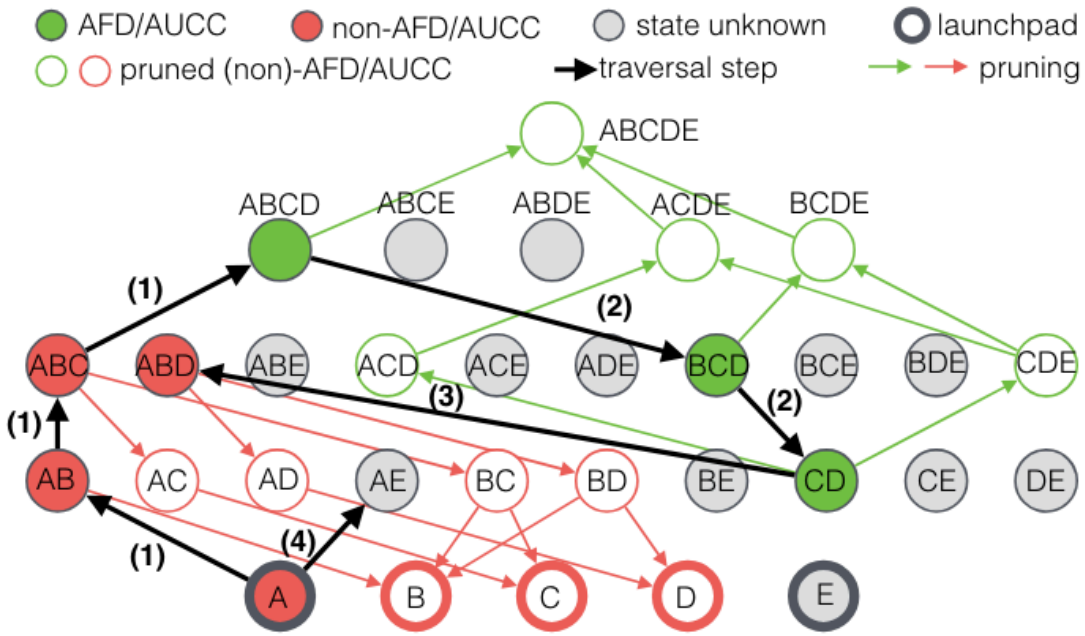


Рис. 2: Пример обхода пространства поиска в алгоритме Pyro (взят из оригинальной работы [17])

В отличие от алгоритма TANE, данный алгоритм отсекает не только специализации найденных функциональных зависимостей на основании заведомой неминимальности этих специализаций, но также удаляет из пространства поиска и обобщения потенциальных зависимостей, у которых посчитанная ошибка оказалась меньше установленного порога. На Рисунке 2 отсечение первого типа отмечено зеленым цветом, а

второго типа красным.

В описанных процедурах алгоритм серьезно полагается на используемую метрику ошибки. С одной стороны, авторами предложен эффективный метод оценки значения ошибки зависимости-кандидата без ее собственно вычисления. С другой стороны, корректность процедуры отсечения основана на свойстве монотонности используемой метрики. В примере на Рисунке 2, алгоритм, определив, что зависимость $ABD \to E$ не удерживается, может отсечь все *обобщения* данной зависимости, поскольку их ошибка не может быть меньше ошибки *специализации*.

3.2. Инструменты для профилирования данных

3.2.1. Metanome

Metanome [6] — это открытая платформа для профилирования данных, реализованная на языке Java. Она включает различные алгоритмы профилирования наиболее современного уровня, в том числе алгоритмы поиска функциональных зависимостей. Платформа позволяет запускать эти алгоритмы на наборах данных как при помощи встроенного web-интерфейса, так и из консоли.

Цель проекта Metanome — предоставить современные алгоритмы профилирования, проводить сравнительные оценки и поддерживать разработчиков в создании и тестировании алгоритмов.

В проекте Metanome реализован алгоритм TANE [23] на языке Java, авторами описаны внесенные модификации [17].

3.2.2. Desbordante

Desbordante [22] — это профилировщик данных с открытым исходным кодом, реализованный на языке C++. Он создан с упором на промышленное применение: он эффективный, масштабируемый и устойчивый к сбоям. Инструмент предоставляет web-интерфейс, консольный интерфейс и библиотеку для использования в программах на языке Python.

Среди большинства существующих систем профилирования данных, ориентированных на поиск сложных метаданных, инструмент Desbordante выделяется следующими характеристиками. Во-первых, он стремится обеспечить более тесную интеграцию с инструментами, использующимися специалистами по анализу данных. Во-вторых, Desbordante ориентируется на нагрузки промышленного уровня: ряд реализованных в Desbordante алгоритмов до десяти раз быстрее чем java-реализации [19, 10] из проекта Metanome и требуют до трех раз меньше памяти. Наконец, он преследуют цель дать пользователю объяснения, почему тот или иной шаблон не найден.

Наконец, в проекте Desbordante реализован алгоритм TANE на языке C++, который поддерживает поиск AFD по модифицированной метрике g_1.

3.3. Вывод

В ходе обзора были описаны особенности и преимущества двух алгоритмов для поиска неточных зависимостей.

Рассмотренные алгоритмы изначально разработаны для поиска вхождений определенного варианта неточной функциональной зависимости. Примитив вероятностной функциональной зависимости отличается от них метрикой ошибки (вероятности). Поэтому, выбирая алгоритм для поиска нового примитива, следует удостовериться, что этот алгоритм возможно корректно модифицировать.

Для ускорения, каждый алгоритм использует процедуру отсечения части пространства поиска, то есть удаляет из рассмотрения некоторые зависимости. Ввиду решаемой задачи, алгоритмы удаляют из рассмотрения заведомо неминимальные или неудерживающиеся зависимости. В случае алгоритма Pyro, обнаружение неудерживающийся зависимости позволяет удалить из рассмотрения все ее обобщения, которые по свойству используемой метрики тоже не будут удерживаться.

Однако метрика pFD PerValue допускает, что обобщения могут иметь ошибку меньше исходной зависимости. Поэтому алгоритм Pyro

в оригинальном виде не применим для поиска вероятностных функциональных зависимостей. Процедура эффективного отсечения — это существенная часть алгоритма Pyro, так что даже если ее модифицировать, получится другой алгоритм.

Напротив, в результате изучения алгоритма TANE выявлено, что его процедура отсечения не полагается на свойства используемой метрики. Более того, алгоритм TANE считается классическим и является широко распространенным. Будучи достаточно общим, он был использован для поиска различных примитивов [17, 24, 12, 25]. Авторы примитива pFD адаптировали именного его для поиска pFD, однако, насколько нам известно, они не представили исходный код своей реализации [12]. Данные наблюдения послужили доводом к выбору алгоритма TANE для реализации поиска вероятностных функциональных зависимостей.

В инструменте Desbordante, который планируется расширять, на момент начала работы присутствовала реализация алгоритма TANE, которая может послужить основой для поддержки нового примитива.

4. Сравнение определений функциональных зависимостей

Готовясь расширять инструмент Desbordante, важно исследовать и оценить примитив вероятностных функциональных зависимостей. С одной стороны, это позволит мотивировать новую функциональность. С другой стороны, оценка поможет подготовить примеры использования нового примитива, что является неотъемлемой частью предлагаемого решения и поможет пользователям инструмента выбрать наиболее подходящий для их данных примитив.

На момент начала работы, в инструменте Desbordante поддерживался поиск одного примитива ослабленных функциональных зависимостей, а именно AFD. Поэтому естественно было провести сравнение с ним нового примитива pFD. Так, мы оценим поведение соответствующих им метрик ошибки на синтетических примерах, а также проведем сравнение найденных множеств вхождений pFD и AFD в реальном наборе данных.

Для начала рассмотрим упрощенный набор данных r, представленный в таблице 4 и зависимость $X \to Y$. Метрика pFD PerValue не зависит от частоты X. Действительно, рассмотрим $|V_0| \to \infty$. В этом случае $Pr(X \to Y, r)_{PerValue}$ стремится к $\frac{6}{7}$ и соответствующая ей ошибка $1 - Pr(X \to Y, r)_{PerValue}$ стремится к $\frac{1}{7}$. В то же время $1 - Pr(X \to Y, r)_{PerTuple}$ и $e(X \to Y, r)$ стремятся к 1.

Таким образом, эта метрика может учитывать «неисправные» LHS, если их не слишком много. Это позволяет иметь много нарушающих зависимость кортежей, если они соответствуют относительно небольшому количеству различных значений LHS. Обращаясь к примеру из реального мира, можно представить набор из большого количества датчиков, которые производят последовательные измерения. Если датчики исправны, то можно ожидать, что будет удерживаться точная зависимость $X \to A$. Однако даже если один-два датчика начали сообщать много некорректных данных, то вероятностная функциональная зависимость с метрикой PerValue будет скорее обнаружена, чем AFD, при

фиксированном порогом ошибки. Более конкретный пример был разработан во время интеграции новой компоненты в Desbordante и представлен в главе 5.2.

Теперь рассмотрим данные, представленные в таблице 5, и ту же зависимость. Зависимость pFD PerValue с меньшей вероятностью удерживается: $Pr(X \to Y, r)_{PerValue} = \frac{5}{8} = 0.625$ и $1 - Pr(X \to Y, r)_{PerValue} = 0.375$. При этом $1 - Pr(X \to Y, r)_{PerTuple} = \frac{3}{11} = 0.27$, а $e(X \to Y, r) = 0.05$.

Таким образом, метрика pFD PerValue сообщает о большей ошибке, когда имеется много уникальных значений LHS, для которых зависимость не сохраняется. Она игнорирует положительный вклад значения «0», независимо от его количества.

Для аналитика данных такое поведение может быть нежелательным и привести к упущению ценных фактов. Предположим, что в этой таблице один миллион таких «0», а остальные шесть записей остались прежними. В этой таблице зависимость будет иметь значение ошибки PerValue, равное 0.375, что довольно велико, и поэтому закономерность будем проигнорирована. Однако более вероятная интерпретация следующая: один миллион записей верен (поскольку их один миллион), а эти шесть значений являются аномалиями, которые следует удалить. Именно AFD окажется более точным при такой интерпретации.

Чтобы показать существенное различие примитивов на практике, обратимся к Таблице 3. Она содержит результаты поиска трех различных типов зависимостей в наборе данных Monkeypox.csv: AFD, pFD с PerValue (PV) и pFD с PerTuple (PT). Таблица показывает множества различных минимальных нетривиальных зависимостей, их разность и пересечение, а также множество pFD, которые не выводятся из минимальных AFD путем добавления атрибутов к LHS зависимости. Таблица показывает, что AFD не может найти некоторые pFD при запуске с определенными порогами ошибки, несмотря на то, что набор данных содержит сопоставимое количество минимальных нетривиальных AFD.

Подводя итог, можно сказать, что pFD имеют свои сильные стороны и отличаются от AFD. В частности, при фиксированном пороге ошибки, pFD не являются простым подмножеством AFD, и наоборот.

Таблица 3: Минимальные нетривиальные pFD и AFD, найденные в наборе данных monkeypox.csv

	Всего			$\lvert pFDs \setminus AFDs \rvert$		Невыводимые pFDs		$\lvert AFDs \setminus pFDs \rvert$		$\lvert pFDs \cap AFDs \rvert$	
Ошибка	AFD	pFD PV	pFD PT	PV	PT	PV	PT	PV	PT	PV	PT
0.01	126	142	134	133	124	3	1	117	116	9	10
0.05	73	69	71	62	64	2	2	66	66	7	7
0.1	55	81	64	72	55	2	0	46	46	9	9
0.2	69	168	70	153	60	29	0	54	59	15	10
0.3	63	70	51	61	42	30	2	54	54	9	9

Таблица 4: Пример 1

X	Y
0	1
0	2
0	3
0	4
0	5
...	...
1	1
2	2
3	3
4	4
5	5
6	6

Таблица 5: Пример 2

X	Y
0	1
0	1
0	1
0	1
0	1
1	1
1	2
2	3
2	4
3	5
3	6

5. Решение

Особенность разработки и поддержки проекта Desbordante в том, что он — сложный проект с обширной кодовой базой на нескольких языках программирования, включающий в себя более десятка алгоритмов, различные структуры данных, автоматические тесты и многое другое. Для того, чтобы справляться с возрастающей сложностью проекта, его разработчики следуют выработанным в проекте архитектурным решениям, касающихся стандартизации пользовательского интерфейса, формата входных и результирующих данных, тестирования, подхода к определению параметров алгоритмов. Также в системе выделены логические модули, связанные с собственно реализациями алгоритмов и их опциями, которые будут затронуты в ходе *реализации* новой функциональности. Кроме того, выделяются модули, связанные с тестами, python-библиотекой и консольным интерфейсом, которые затрагивает *интеграция* разработанной компоненты в систему.

5.1. Особенности реализации

В проекте Desbordante все алгоритмы представлены классами, наследующими класс *Algorithm*. Данный абстрактный класс содержит стандартные методы для загрузки данных и взаимодействия с индикатором прогресса. Алгоритмы для поиска функциональных зависимостей могут также быть унаследованы от класса *FDAlgorithm*, который содержит стандартные методы для регистрации найденных зависимостей, преобразования результата в формат JSON, подсчет хеш-значения и прочие. В каждом алгоритме можно зарегистрировать используемые опции в методе RegisterOptions и определить логику алгоритма в методе ExecuteInternal.

На рисунке 3 представлена диаграмма классов, которая показывает место разработанной компоненты в иерархии наследования классов. Оранжевым цветом отмечен класс новой компоненты, разработанный в рамках настоящей работы. Классы, иерархия наследования которых была изменена и в которые были внесены мелкие исправления в ходе

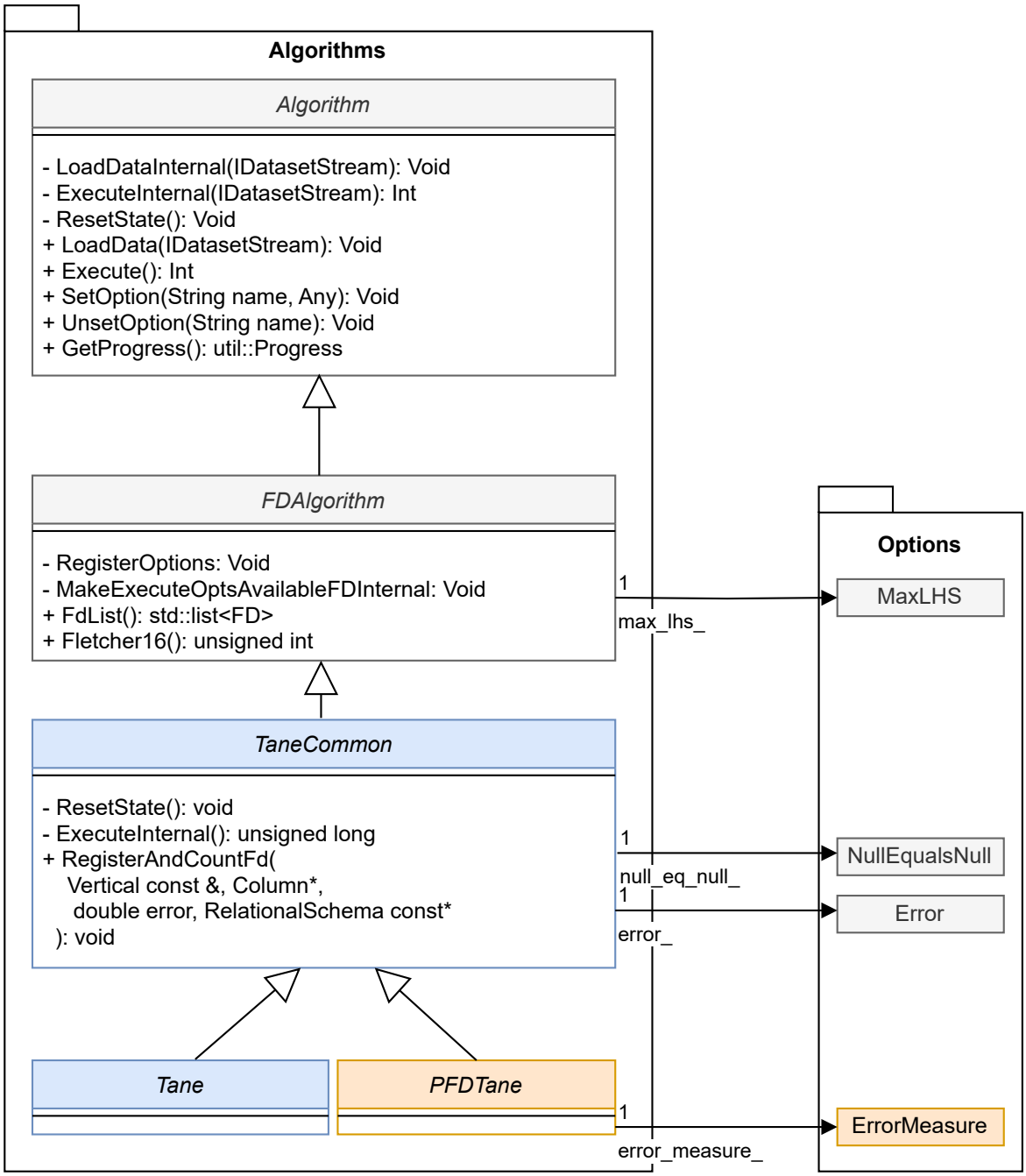


Рис. 3: Иерархия классов алгоритмов поиска FD

работы, отмечены синим цветом.

На момент начала работы в проекте Desbordante уже была реализация алгоритма TANE для поиска приближенных функциональных зависимостей, представленная в классе *Tane* (в дальнейшем называется также *AFDTane*). В Desbordante принято архитектурное решение о том, что каждый алгоритм поиска вхождений определенного примитива должен быть представлен отдельным классом. Поэтому поиск вероятност-

ных функциональных зависимостей тоже представлен отдельным классом PFDTane. Для того того, чтобы переиспользовать код имеющейся реализации TANE, был создан абстрактный класс *TaneCommon*, содержащий основную логику алгоритма. В ходе работы был также произведен рефакторинг базовой реализации TANE, выделены в отдельные методы процедуры вычисления зависимостей (ComputeDependencies) и отсечения (Prune), а также исправлено отсечение вершин решетки по правилу ключей и суперключей.

На рисунке 4 представлена диаграмма классов, связанных с алгоритмом TANE. Алгоритм поиска AFD представлен классом *Tane*, а алгоритм поиска pFD классом *PFDTane*. В классе *TaneCommon* объявлены виртуальные методы подсчета ошибки *CalculateZeroAryFdError* и *CalculateFdError*. После завершения работы по изменению иерархии наследования было экспериментально подтверждено, что производительность алгоритма TANE не была снижена [21] из-за использования виртуальных методов.

Процедура подсчета вероятности зависимости в сущности является единственной частью, которую необходимо было изменить в алгоритме TANE, чтобы приспособить его к поиску pFD. Процедура реализована в классе PFDTane и алгоритм представлен в листинге 1. В отличие от алгоритма из оригинальной статьи [12], представленного в листинге 2 на его вход принимаются сокращенные партиции для наборов атрибутов X и XA.

5.2. Интеграция в инструмент Desbordante

Разработанная компонента может быть использована программистами в программах на языке C++, однако проект Desbordante нацелен на то, чтобы предоставить и более простые способы использования алгоритмов, например, из консоли или посредством python-библиотеки. Данная задача, а также задача по разработке примеров использования и автоматических тестов решена в ходе интеграции компоненты в систему.

Input: Отношение R без null-значений, партиции $\hat{\pi}_X, \hat{\pi}_{XA}$
Output: Значение метрики PerValue для зависимости $X \to A$

```
1  SORT(π̂_XA)
2  S ← 0;
3  t̄' ← next in π̂_XA
4  for each t̄ ∈ π̂_X do
5  |   c ← 1
6  |   for each t ∈ t̄ do
7  |   |   if not next in π̂_XA then
8  |   |   |   break
9  |   |   end
10 |   |   if t == t̄'_1 then
11 |   |   |   c ← max(|t̄'|, c)
12 |   |   |   t̄' ← next in π̂_XA
13 |   |   end
14 |   end
15 |   S ← S + c/|t̄|
16 end
17 return (S+|R|−|π̂_X|) / |π̂_X|
```

Algorithm 1: Вычисление вероятности PerValue

Input: Отношение R, атрибуты X и A
Output: Значение метрики PerValue для зависимости $X \to A$

```
1  SORT(R, {X, A})
2  c ← t_1(X);
3  |π(X)| ← 1;
4  count(c) ← 0
5  c' ← t_1(X, A);
6  count(c') ← 0;
7  maxCount(c) ← 0
8  sum ← 0
9  for each t ∈ R do
10     if t(X) == c then
11         count(c) ← count(c) + 1
12         if t(X, A) == c' then
13             count(c') ← count(c') + 1
14         end
15         else
16             if maxCount(c) < count(c') then
17                 maxCount(c) ← count(c')
18             end
19             c' ← t(X, A);
20             count(c') ← 0
21         end
22     end
23     else
24         sum ← sum + maxCount(c)/count(c)
25         c ← t(X);
26         |π(X)| ← |π(X)| + 1
27         count(c) ← 0
28         maxCount(c) ← 0
29     end
30 end
31 return sum/|π(X)|
```

Algorithm 2: Вычисление вероятности PerValue [12]

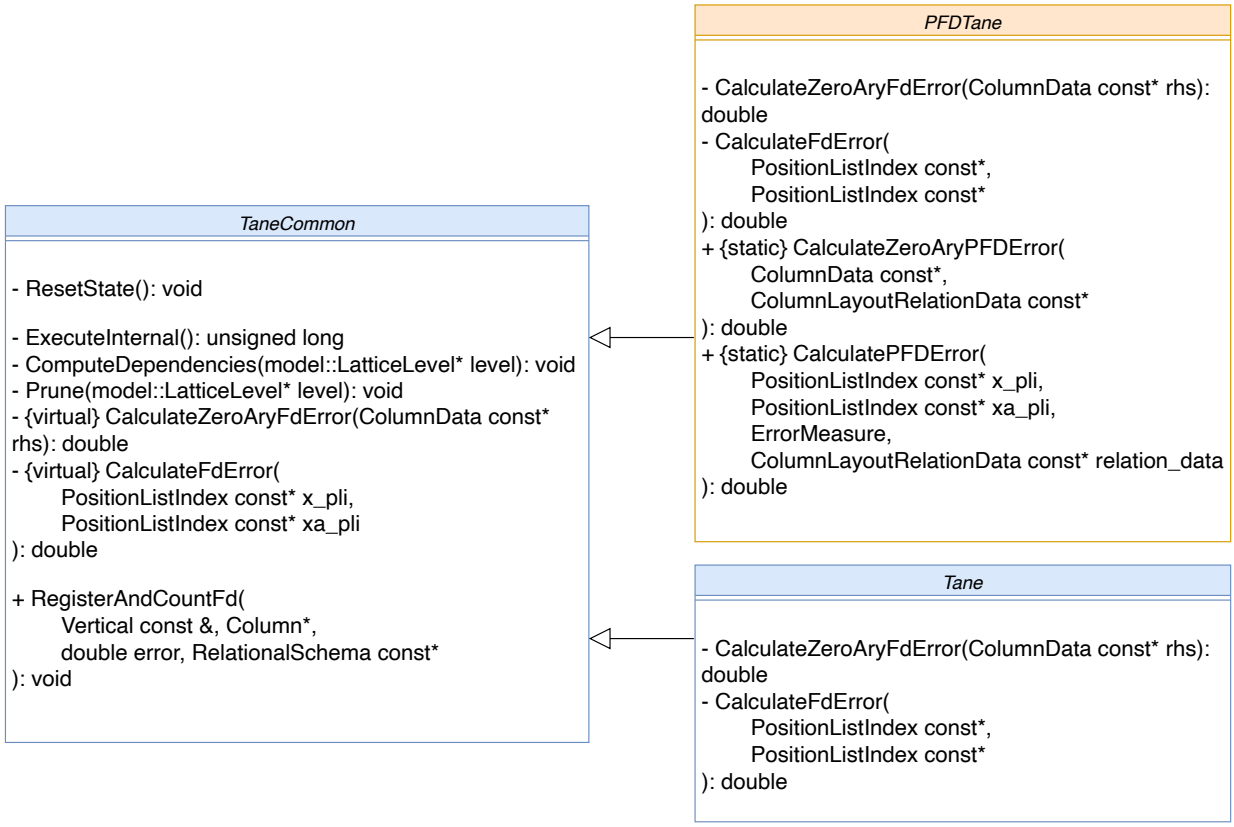


Рис. 4: Иерархия классов алгоритмов, основанных на TANE

На рисунке 5 представлена компонентная диаграмма системы. В python-библиотеке определяются модули *fd.algorithms*, *afd.algorithms*, *pfd.algorithms*, которые содержат python-классы алгоритмов для соответствующих примитивов. Python-классы определяют методы, такие как Execute, которые связываются со своей реализацией на C++ в конкретных алгоритмах. Модуль консольного интерфейса (CLI) используют внутри себя python-библиотеку.

Библиотека для языка python собирается с помощью инструментария pybind11. Он позволяет определять модули, классы и методы, которые станут доступны в python-скриптах при подключении python-библиотеки, связывая их с конкретной реализацией на C++. Для того чтобы алгоритм PFDTane стал доступен из python-скриптов, был зарегистрирован новый модуль «pfd» в функции *python_ bindings::BindFd*. Чтобы в python-скриптах можно было использовать опции алгоритма, были определены соответствующие им python-типы в функции *python_ bindings::GetPyType*.

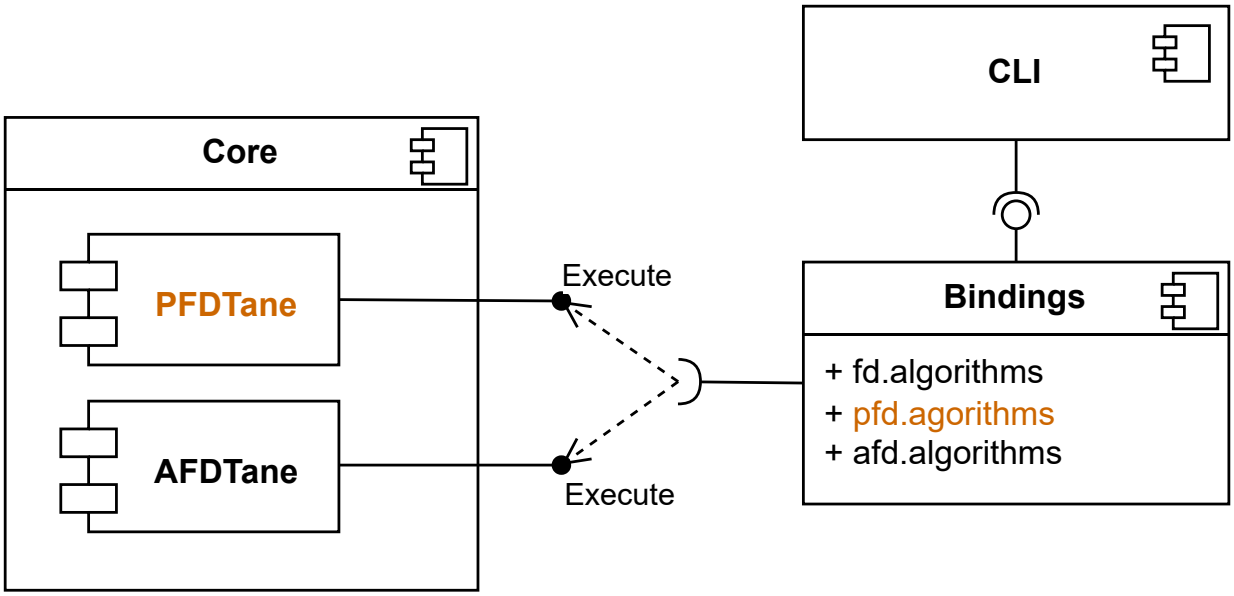


Рис. 5: Диаграмма компонентов системы

Консольный интерфейс в проекте Desbordante реализован на языке python в файле cli.py. Для поддержки нового примитива, был определен новый тип задачи «pfd» в словаре *TASK_INFO*, а также алгоритм «pfdtane» в словаре *ALGOS*, который соответствует доступному в python-библиотеке классу *pfd.algorithms.PFDTane*. Пример использования консольного интерфейса приведен ниже.

```
python3 cli.py --task=pfd --algo=pfdtane --error=0.01 \
    --error_measure=per_value --table data.csv , True
```

На следующем этапе интеграции были разработаны автоматические тесты алгоритма поиска pFD. Во-первых, разработанный алгоритм был добавлен в список тестируемых алгоритмов поиска точных функциональных зависимостей. Дополнительно были разработаны тесты вычисления вероятности по метрикам PerValue и PerTuple и тесты для проверки результатов полного поиска функциональных зависимостей. Сравнение результатов алгоритма с эталонными происходит с помощью вычисления хеш-значения.

Интеграция новой функциональности в Desbordante обязательно сопровождается разработкой документации. Дополнительную помощь пользователю предоставляют также примеры использования, которые

Таблица 6: Пример с неисправным датчиком

id	DeviceId	Data
1	D-1	1001
2	D-1	1002
3	D-1	1003
4	D-1	999
5	D-1	1010
6	D-1	300
7	D-1	301
8	D-2	1000
9	D-2	1000
10	D-3	500
11	D-4	3000
12	D-4	3000
13	D-4	3000
14	D-5	300

не только показывают технические аспекты использования примитивов, но и описывают приближенные к реальным сценарии использования.

Для нового алгоритма были разработаны примеры, которые показывают особенности метрик PerValue и PerTuple, а также демонстрируют сценарий использования pFD в случае данных с одним неисправным датчиком. Первый пример, представленный в листинге 1 показывает самый базовый пример использования python-библиотеки: загрузка данных происходит на четвертой строке, запуск алгоритма — на шестой. Другой пример проясняет особенность метрики PerValue, состоящую в том, что метрика не принимает во внимание частоту отдельных значений атрибутов из левой части зависимости среди всех кортежей: например, в таблице 6 значение ошибки $1 - PerValue([DeviceId] \to Data) = 0.1714$ меньше, чем значение ошибки AFD $e([DeviceId] \to Data) = 0.2307$. В данном сценарии число ошибочных записей, поступивших от неисправного датчика, лишь немного влияет на метрику PerValue, таким образом позволяя обнаружить функциональную зависимость измерений «Data» от устройства «DeviceId».

Листинг 1: Пример использования реализованного алгоритма посредством python-библиотеки

```
import desbordante
ERROR_1 = 0.02777777778 # threshold of the Y->X dependency
algo = desbordante.pfd.algorithms.PFDTane()
algo.load_data(table=('examples/datasets/pfd.csv',',',True))
for ERROR_MEASURE in ['per_value', 'per_tuple']:
    algo.execute(error=ERROR_1, error_measure=ERROR_MEASURE)
    result = algo.get_fds()
    print(ERROR_MEASURE, 'pFDs:')
    for fd in result:   print(fd)
```

6. Эксперименты

6.1. Методология

Для исследования производительности поиска pFD были поставлены следующие исследовательские вопросы (ИВ):

ИВ1 Каковы затраты времени и памяти алгоритма поиска pFD по сравнению с AFD?

ИВ2 Как значение порога ошибки влияет на время выполнения и потребление памяти алгоритма поиска pFD?

ИВ3 Насколько затратна в вычислительном отношении процедура подсчета значения ошибки pFD по сравнению с AFD?

Эксперименты проводились с использованием следующей аппаратной и программной конфигурации. Аппаратное обеспечение: процессор AMD® Ryzen 5 7600X с частотой 5,453 ГГц (6 ядер), 32 ГБ ОЗУ. Программное обеспечение: Ubuntu 22.04 LTS, ядро 6.5.0-15-generic (64-разрядное). Для проведения экспериментального исследования использовались наборы данных с разными характеристиками, представленные в Таблице 11. Ссылки на наборы данных доступны в GitHub репозитории [18].

Таблица 7: Отношение времени выполнения алгоритмов AFDTane и pFDTane PerValue

Набор данных \ Порог ошибки	0.025	0.05	0.075	0.1	0.15	0.2	0.25	0.3	0.4	0.5
BKB_WaterQualityData_2020084	2.529	2.199	2.010	1.763	1.432	1.267	1.128	1.076	1.050	1.027
EpicVitals	2.671	2.637	2.623	2.589	2.145	1.952	1.679	1.636	1.548	1.547
jena_climate_2009_2016	1.455	1.475	1.470	1.359	1.387	1.316	1.308	1.322	1.320	1.360
measures_v2	0.962	0.972	0.951	0.921	0.946	0.953	0.992	0.985	0.949	0.915
nuclear_explosions	2.160	1.862	1.668	1.466	1.220	1.101	1.070	1.048	1.024	1.015
parking_citations	3.358	3.261	3.177	3.079	2.882	2.513	2.073	1.550	1.351	1.231
SEA	1.745	1.815	1.767	1.805	1.752	1.821	1.794	1.773	1.615	1.674
games	1.785	1.587	1.517	1.446	1.361	1.299	1.277	1.217	1.212	1.162

Таблица 8: Отношение времени выполнения алгоритмов AFDTane и pFDTane PerTuple

Набор данных \ Порог ошибки	0.025	0.05	0.075	0.1	0.15	0.2	0.25	0.3	0.4	0.5
BKB_WaterQualityData_2020084	2.685	2.415	2.320	2.262	2.127	2.007	1.908	1.804	1.578	1.417
EpicVitals	2.642	2.636	2.677	2.638	2.283	2.087	2.086	2.031	2.017	1.974
jena_climate_2009_2016	1.465	1.479	1.514	1.487	1.410	1.343	1.326	1.396	1.316	1.368
measures_v2	0.894	0.952	0.904	0.892	0.928	0.948	0.961	0.965	0.943	0.914
nuclear_explosions	2.475	2.316	2.251	2.153	2.010	1.802	1.701	1.542	1.419	1.264
parking_citations	3.386	3.373	3.344	3.289	3.197	2.987	2.740	2.681	2.557	2.318
SEA	1.742	1.800	1.747	1.780	1.796	1.819	1.777	1.776	1.626	1.636
games	2.064	1.810	1.723	1.630	1.529	1.503	1.451	1.399	1.356	1.300

Таблица 9: Отношение потребляемой памяти алгоритмов AFDTane и pFDTane PerValue

Набор данных \ Порог ошибки	0.025	0.05	0.075	0.1	0.15	0.2	0.25	0.3	0.4	0.5
BKB_WaterQualityData_2020084	0.931	0.932	0.922	0.911	0.894	0.885	0.873	0.871	0.874	0.873
EpicVitals	0.542	0.542	0.542	0.542	0.521	0.521	0.486	0.486	0.486	0.486
jena_climate_2009_2016	0.356	0.356	0.356	0.356	0.356	0.356	0.356	0.356	0.356	0.356
measures_v2	0.985	0.985	0.985	0.985	0.985	0.985	0.985	0.985	0.985	0.985
nuclear_explosions	0.127	0.128	0.127	0.127	0.127	0.127	0.127	0.127	0.127	0.127
parking_citations	0.926	0.926	0.926	0.926	0.926	0.926	0.926	0.925	0.925	0.925
SEA	1.029	1.030	1.029	1.030	1.030	1.030	1.030	1.030	1.030	1.046
games	0.170	0.162	0.160	0.160	0.159	0.159	0.159	0.159	0.159	0.159

Таблица 10: Отношение потребляемой памяти алгоритмов AFDTane и pFDTane PerTuple

Набор данных \ Порог ошибки	0.025	0.05	0.075	0.1	0.15	0.2	0.25	0.3	0.4	0.5
BKB_WaterQualityData_2020084	0.942	0.940	0.935	0.934	0.927	0.918	0.915	0.911	0.901	0.890
EpicVitals	0.542	0.542	0.542	0.542	0.543	0.542	0.542	0.521	0.521	0.521
jena_climate_2009_2016	0.356	0.356	0.356	0.356	0.356	0.356	0.356	0.356	0.356	0.356
measures_v2	0.985	0.985	0.985	0.985	0.985	0.985	0.985	0.985	0.985	0.985
nuclear_explosions	0.127	0.128	0.127	0.127	0.127	0.126	0.126	0.126	0.127	0.127
parking_citations	0.926	0.926	0.926	0.926	0.926	0.926	0.926	0.926	0.926	0.926
SEA	1.029	1.030	1.029	1.030	1.030	1.030	1.030	1.030	1.030	1.046
games	0.184	0.169	0.165	0.163	0.161	0.160	0.160	0.160	0.159	0.159

Таблица 11: Наборы данных для экспериментов

Набор данных	Строки	Атрибуты	Размер	Источник	pFD PT	pFD PV	AFD
EpicVitals.csv	1246303	7	33MB	EPF	10	13	21
BKB_WaterQualityData_2020084.csv	2370	17	180KB	U.S. FWS	3389	3712	901
games.csv	20058	16	7.67MB	kaggle	2264	1810	266
jena_climate_2009_2016.csv	420550	15	43.16MB	kaggle	3003	3148	210
measures_v2.csv	1330816	13	300.06MB	kaggle	642	573	144
nuclear_explosions.csv	2046	16	220KB	tidytudesday repository	2795	3619	1459
parking_citations.csv	95433	13	10MB	norfolk opendata	224	269	565
SEA.csv	1000000	4	33MB	openml.com	3	3	9
monkeypox.csv	5875	14	516KB	who.int	134	142	126

Таблица 12: Производительность поиска точных FD

Набор данных	Время (с)			Память (MB)		
	pFDTane PV	pFDTane PT	AFDTane	pFDTane PV	pFDTane PT	AFDTane
BKB_WaterQualityData_2020084	2.013	2.018	0.935	216	216	260
EpicVitals	8.609	8.583	4.266	807	807	1490
jena_climate_2009_2016	12.711	12.720	6.205	530	530	1490
measures_v2	16.245	16.334	15.278	1758	1758	1785
nuclear_explosions	2.198	2.203	0.795	189	189	1490
parking_citations	25.908	25.953	7.360	1381	1381	1491
SEA	1.963	1.956	1.193	279	279	270
games	3.207	3.222	1.492	399	399	1490

6.2. Сравнение производительности алгоритмов AFDTane и PFDTane

Для ответа на первый исследовательский вопрос реализации алгоритмов были протестированы с разными порогами ошибки. Результаты представлены в таблицах 7–8 по времени выполнения и в таблицах 9–10 по потреблению памяти соответственно. Каждая ячейка содержит соотношение соответствующего измерения алгоритма pFDTane к AFDTane.

Почти на всех наборах данных заметно, что AFDTane является в два-три раза более быстрым алгоритмом, чем pFDTane, за исключением measures_v2.csv. Результаты свидетельствуют об уменьшении разницы в производительности при пороге ошибки, превышающем 0.1.

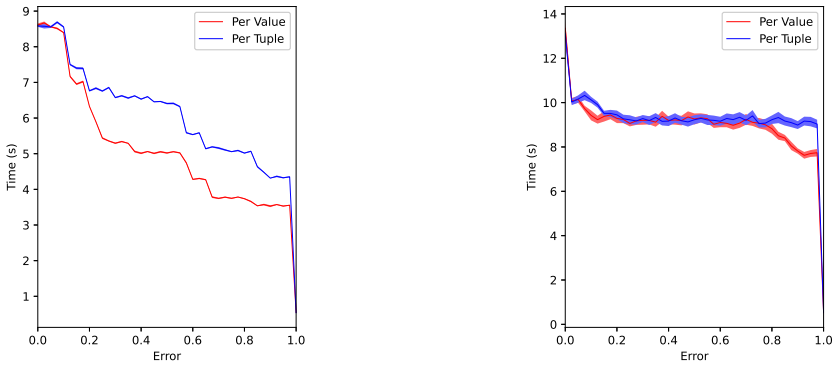
Кроме того, было замечено, что потребление памяти для pFDTane ниже для всех наборов данных, кроме SEA.csv. В отличие от времени выполнения, потребление памяти для pFDTane и AFDTane практически одинаково для каждого из порога ошибки.

6.3. Влияние порога ошибки на производительность

Чтобы ответить на второй исследовательский вопрос, поиск функциональных зависимостей был выполнен в различных наборах данных и с разным значением порога ошибки, принимающим значение от нуля до единицы. Рисунки 6a и 6b показывают два различных паттерна поведения реализации. Когда алгоритм pFDTane использовался для поиска зависимостей в наборе данных jena_climate_2009_2016.csv, время работы не демонстрировало заметное отличие при пороге ошибки больше 0.25. Наоборот, когда на вход алгоритма подавался набор данных EpicVitals.csv, время работы стремительно уменьшалось при увеличении порога ошибки. Алгоритм работал подобным образом на шести из восьми наборов данных.

В первом случае поведение алгоритма можно объяснить принципом работы алгоритма TANE: чем больше зависимостей алгоритм найдет на первых уровнях решетки, тем меньше всего проверок будет проведено.

При этом, как видно из анализа соответствующего исследовательского вопроса, процедура проверки является довольно затратной. Соответственно, при увеличении порога ошибки, алгоритму приходится проводить меньше вычислительно дорогих валидаций зависимостей на части датасетов, что уменьшает общее время поиска.



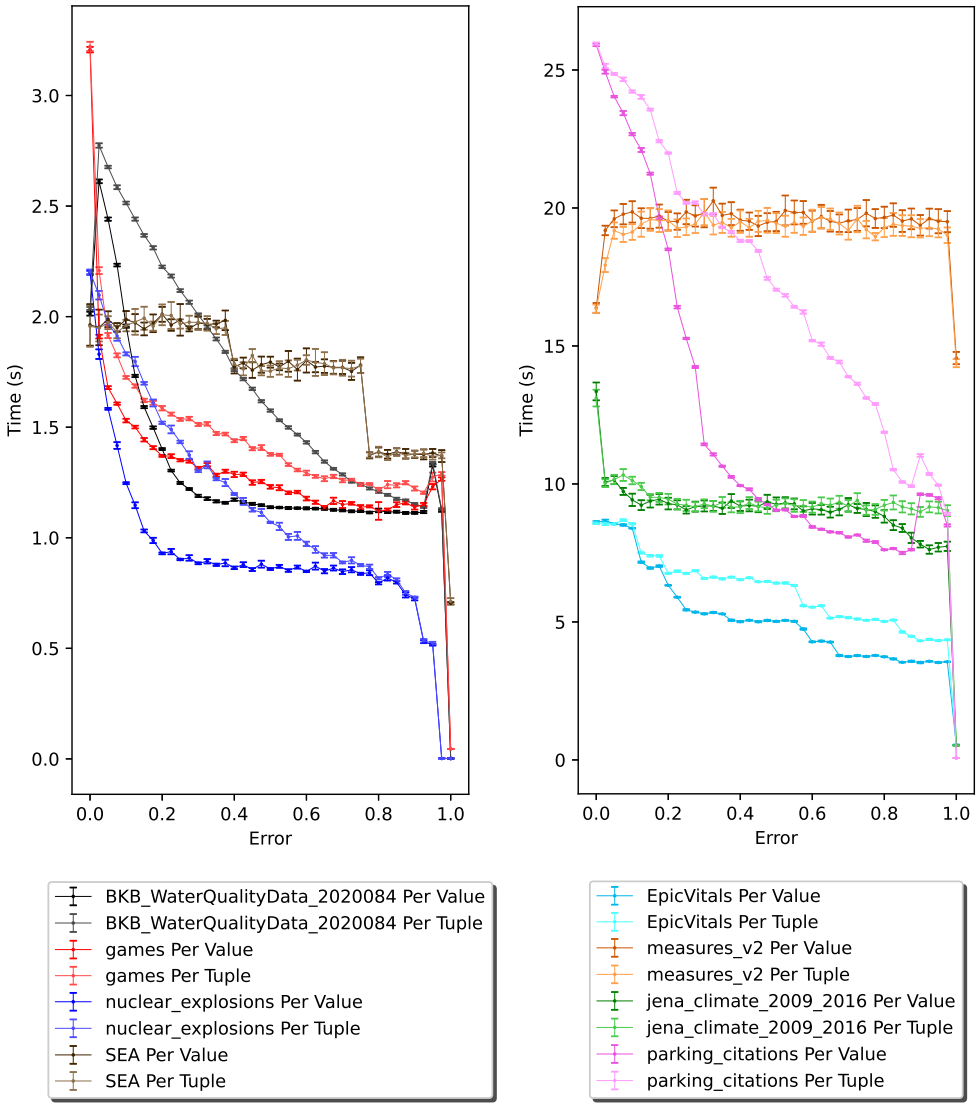
(a) EpicVitals (b) jena_climate

Рис. 6: Время работы алгоритма pFDTane

Наконец, можно сравнить поведение алгоритма при использовании различных метрик вероятности. Как видно из Рисунка 7, поиск с метрикой PerValue показывает лучшие результаты в смысле затрачиваемого времени почти на всех датасетах, кроме трех. Замеры на SEA и jena_climate и measures_v2 не позволяют однозначно судить о преимуществе той или иной метрики. Наконец, наблюдается различное поведение алгоритма при приближении порога ошибки к единице: иногда наблюдается внезапное увеличение времени выполнения, как в случае с parking_citations. Ответы на вопросы о причинах данного поведения может дать дальнейшее профилирование программы. Но стоит отметить, что порог ошибки, больший 0.8 на практике не используется.

6.4. Производительность процедуры подсчета значения ошибки

Чтобы сравнить время выполнения и потребление памяти функций подсчета ошибки pFDTane и AFDTane, соответствующие алгоритмы были запущены с ошибкой, установленной в 0. Этот параметр гаранти-



(a) Первая группа (b) Вторая группа

Рис. 7: Время работы в зависимости от порога ошибки

рует, что все алгоритмы проходят одну и ту же часть решетки и, таким образом, находятся на равных условиях. Результаты, представленные в таблице 12, демонстрируют тот факт, что и PerValue, и PerTuple работают медленнее, чем проверка с помощью метрики AFD. С другой стороны, PerValue и PerTuple не демонстрируют существенной разницы ни во времени выполнения, ни в потреблении памяти.

Можно также отметить, что почти для всех наборов данных pFDTane потребляет меньше памяти по сравнению с AFDTane. Исключением был набор данных SEA.csv, который использовал примерно тот же объем памяти.

Заключение

В ходе данной работы были выполнены следующие задачи:

- исследованы алгоритмы из предметной области и обоснован выбор алгоритма TANE;
- исследованы вероятностные функциональные зависимости и показаны их отличия от приближенных;
- реализована компонента поиска вероятностных функциональных зависимостей на основе существующей реализации алгоритма TANE на языке C++;
- выполнена интеграция компоненты в платформу Desbordante, добавлена поддержка консольного интерфейса и языка Python, разработаны автоматические тесты и примеры использования;
- выполнено экспериментальное исследование реализации:
  - на 75% наборах данных алгоритм начинает работать быстрее при увеличении порога ошибки, но на остальных время работы почти не меняется,
  - поиск вероятностных функциональных зависимостей работает до двух-трех раз медленнее, чем поиск приближенных функциональных зависимостей,
  - подсчет значения ошибки вероятностной зависимости медленнее, чем для приближенной и приводит к двукратному-трехкратному замедлению алгоритма поиска.

Основная разработанная функциональность принята в основной репозиторий, PR #300. Ссылка на репозиторий https://github.com/desbordante/desbordante-core (имя пользователя — iliya-b).

По итогам работы написана статья “Extending Desbordante with Probabilistic Functional Dependency Discovery Support” [8]. Статья принята и доложена на конференции “Open Innovations Association (FRUCT)”, материалы которой публикуются в IEEE и индексируются в Scopus.

Благодарности

Автор выражает признательность Георгию Чернышеву за курирование всех аспектов работы, от постановки задачи до научной публикации, а также коллегам, помогавшим в ходе настоящей работы: Сергею Белоконному за помощь в написании скриптов для экспериментов, Владиславу Макееву за возможность запуска экспериментов на его компьютере, Антону Чижову за ценные замечания по написанному коду.

Список литературы

[1] Aggarwal Charu C., Han Jiawei. Frequent Pattern Mining. — Springer Publishing Company, Incorporated, 2014. — ISBN: 3319078208.

[2] Brown Paul G., Hass Peter J. BHUNT: Automatic Discovery of Fuzzy Algebraic Constraints in Relational Data // Proceedings of the 29th International Conference on Very Large Data Bases - Volume 29. — VLDB '03. — VLDB Endowment, 2003. — P. 668–679.

[3] Chernishev G. et al. Desbordante: from benchmarking suite to high-performance science-intensive data profiler // CoRR. — 2023. — Vol. abs/2301.05965.

[4] Data Profiling / Ziawasch Abedjan, Lukasz Golab, Felix Naumann, Thorsten Papenbrock. — Morgan & Claypool Publishers, 2018. — ISBN: 1681734486.

[5] 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.

[6] 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: http://dx.doi.org/10.14778/2824032.2824086.

[7] Discovering Functional Dependencies in Pay-As-You-Go Data Integration Systems : Rep. : UCB/EECS-2009-119 ; Executor: Daisy Zhe Wang, Michael Franklin, Luna Dong et al. : 2009. — URL: https://www2.eecs.berkeley.edu/Pubs/TechRpts/2009/EECS-2009-119.pdf.

[8] Extending Desbordante with Probabilistic Functional Dependency Discovery Support / Ilia Barutkin, Maxim Fofanov, Sergey Belokonny

et al. // 2024 35th Conference of Open Innovations Association (FRUCT). — 2024. — P. 158–169. — Paper accepted.

[9] FARM: Hierarchical Association Rule Mining and Visualization Method / Petr Tsurinov, Oleg Shpynov, Nina Lukashina et al. // Proceedings of the 12th ACM Conference on Bioinformatics, Computational Biology, and Health Informatics. — BCB '21. — New York, NY, USA : Association for Computing Machinery, 2021. — 1 p. — URL: https://doi.org/10.1145/3459930.3469499.

[10] Fast Discovery of Inclusion Dependencies with Desbordante / Alexander Smirnov, Anton Chizhov, Ilya Shchuckin et al. // 2023 33rd Conference of Open Innovations Association (FRUCT). — 2023. — P. 264–275.

[11] Functional Dependency Discovery: An Experimental Evaluation of Seven Algorithms / Thorsten Papenbrock, Jens Ehrlich, Jannik Marten et al. // Proc. VLDB Endow. — 2015. — jun. — Vol. 8, no. 10. — P. 1082–1093. — URL: https://doi.org/10.14778/2794367.2794377.

[12] Functional Dependency Generation and Applications in Pay-As-You-Go Data Integration Systems / Daisy Zhe Wang, Xin Luna Dong, Anish Das Sarma et al. // 12th International Workshop on the Web and Databases, WebDB 2009, Providence, Rhode Island, USA, June 28, 2009. — 2009. — URL: http://webdb09.cse.buffalo.edu/papers/Paper18/webdb09.pdf.

[13] Giannella Chris, Robertson Edward. On approximation measures for functional dependencies // Inf. Syst. — 2004. — aug. — Vol. 29, no. 6. — P. 483–507. — URL: https://doi.org/10.1016/j.is.2003.10.006.

[14] 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.

[15] 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.

[16] Koumarelas loannis, Papenbrock Thorsten, Naumann Felix. MDedup: duplicate detection with matching dependencies // Proc. VLDB Endow. — 2020. — jan. — Vol. 13, no. 5. — P. 712–725. — URL: https://doi.org/10.14778/3377369.3377379.

[17] Kruse Sebastian, Naumann Felix. Efficient discovery of approximate dependencies // Proc. VLDB Endow. — 2018. — mar. — Vol. 11, no. 7. — P. 759–772. — URL: https://doi.org/10.14778/3192965.3192968.

[18] Links to the datasets used in experiments. — URL: https://gist.github.com/iliya-b/f67cf0aa8397ec0a5ab7849376ec8a31 (дата обращения: 1 мая 2024 г.).

[19] Order in Desbordante: Techniques for Efficient Implementation of Order Dependency Discovery Algorithms / Yakov Kuzin, Dmitriy Shcheka, Michael Polyntsov et al. // 2024 35th Conference of Open Innovations Association (FRUCT). — 2024. — P. 413–424. — Paper accepted.

[20] Pandas repository. — URL: https://github.com/pandas-dev/pandas (дата обращения: 1 мая 2024 г.).

[21] Pull Request notes on Github. — URL: https://github.com/Desbordante/desbordante-core/pull/378#issue-2203421159 (дата обращения: 1 мая 2024 г.).

[22] Solving Data Quality Problems with Desbordante: a Demo / George A. Chernishev, Michael Polyntsov, Anton Chizhov et al. // CoRR. — 2023. — Vol. abs/2307.14935. — arXiv : 2307.14935.

[23] TANE implementation in the Metanome project. — URL: https://github.com/HPI-Information-Systems/pyro/blob/master/pyro-metanome/src/main/java/de/hpi/isg/pyro/algorithms/TaneX.java (дата обращения: 1 мая 2024 г.).

[24] Tane: An Efficient Algorithm for Discovering Functional and Approximate Dependencies / Yká Huhtala, Juha Kärkkäinen, Pasi Porkka, Hannu Toivonen // The Computer Journal. — 1999. — Vol. 42, no. 2. — P. 100–111.

[25] Wang Xue, Chen Guoqing. Discovering Fuzzy Functional Dependencies as Semantic Knowledge in Large Databases // The Fourth International Conference on Electronic Business - Shaping Business Strategy in a Networked World / Ed. by Jian Chen. — Academic Publishers/-World Publishing Corporation, 2004. — P. 1136–1139.

[26] Ydata repository. — URL: https://github.com/ydataai/ydata-profiling (дата обращения: 1 мая 2024 г.).