Санкт-Петербургский государственный университет

Кафедра информационно-аналитических систем

Группа 23.Б10-мм

Реализация валидатора условных функциональных зависимостей в проекте Desbordante

Федосеев Дмитрий Алексеевич

Отчёт по учебной практике
в форме «Решение»

Научный руководитель:
ассистент кафедры ИАС Чернышев Г. А.

Санкт-Петербург
2025

Оглавление

Введение

В современном мире количество данных, накопленных компаниями, с каждым годом стремительно увеличивается, что делает их анализ мощным инструментом для улучшения качества услуг или принятия стратегических решений. Например, в розничной торговле анализ данных о покупках позволяет выявить предпочтения клиентов и спрогнозировать их будущие потребности, что помогает повысить уровень обслуживания. Анализ данных приобретает все большее значение, поскольку компании стремятся извлекать полезную информацию для поддержания конкурентоспособности.

Одним из методов анализа данных является профилирование, которое представляет собой процесс извлечения метаданных, то есть сведений о самих данных, из их массивов. Оно включает в себя и так называемое наукоемкое профилирование данных — процесс выявления скрытых взаимосвязей и шаблонов, особенно значимый при анализе табличных данных. Для описания таких закономерностей используются примитивы — математически заданные правила, действующие над определенной частью данных. Примером такого примитива является функциональная зависимость (Functional Dependency, FD), при которой одинаковые значения одного атрибута таблицы неизменно соответствуют одинаковым значениям другого. Условные функциональные зависимости (Conditional Functional Dependency, CFD) являются более сложной версией, где зависимость фиксируется только на определенных подмножествах данных, что открывает новые возможности для детального анализа.

В данном исследовании будет рассмотрена проблема валидации условных функциональных зависимостей в данных с помощью Desbordante [2] — платформы для высокопроизводительного профилирования данных. Основная цель настоящей работы — разработка и внедрение алгоритма, обеспечивающего валидацию CFD, создание Python интерфейса, а также написание примера использования.

1. Постановка задачи

Целью работы является внедрение алгоритма валидации CFD в систему Desbordante, а также добавление наглядных примеров его использования.

1. Выполнить обзор предметной области, произвести анализ существующих решений в проекте;
2. Реализовать алгоритм валидации CFD на C++;
3. Реализовать Python-привязки для алгоритма валидации;
4. Создать пример использования.

2. Обзор предметной области

2.1. Основные определения

Все определения в данном разделе были взяты из статьи [4].

Определение 1. Множество всех атрибутов в таблице обозначим буквой A, каждый атрибут B $\in A$ имеет свою область определения, обозначаемую как dom(B).

Определение 2. Кортежем t называется упорядоченное множество пар вида $(B, t[B])$, где $B \in A$ и $\forall B$ выполнено либо $t[B] \in dom(A)$, либо $t[B]$ является безымянной переменной, обозначаемой "_".

Определение 3. Обозначим подмножество множества всех кортежей как D.

Определение 4. Условная функциональная зависимость φ это пара вида $(X \to Y, t_p)$, определенная на D, где X — это подмножество атрибутов из A, Y — атрибут из A, который не принадлежит X, а t_p — шаблонный кортеж *(Pattern Tuple)*, в котором ключами являются атрибуты из $X \cup Y$. CFD $\varphi = (X \to Y, t_p)$, в котором $t_p[Y] =$ "_", называется *переменной* (Variable), в противном случае — φ *константна* (Constant).

Определение 5. Кортеж t_p поддерживает t, если для любого атрибута A выполнено $t_p[A] = t[A]$ или $t_p[A] =$ "_".

Определение 6. Поддержка (Support) условной функциональной зависимости φ — это количество кортежей из D, которые поддерживаются t_p, обозначается $supp(t_p, D)$.

Определение 7. Будем говорить, что CFD φ вида $(X \to Y, t_p)$ удерживается на D с уверенностью (Confidence) 1, если на множестве поддерживаемых кортежей выполняется функциональная зависимость $X \to Y$.

Определение 8. Если для CFD φ вида $(X \to Y, t_p)$ не выполняется функциональная зависимость $X \to Y$ на множестве поддерживаемых кортежей, она всё ещё может удерживаться на некотором подмножестве. Минимальное количество кортежей, которое необходимо убрать, чтобы функциональная зависимость удерживалась, обозначим как $I(\varphi, D)$. Тогда уверенность (Confidence) φ рассчитывается по формуле $\dfrac{|supp(t_p, D)| - I(\varphi, D)}{|supp(t_p, D)|}$.

2.2. Примеры

Таблица 1: Пример данных клиентов

CC	AC	PN	NM	STR	CT	ZIP
01	908	111111	Mike	Tree Ave.	MH	07974
01	908	111111	Rick	Tree Ave.	MH	07974
01	212	22222	Joe	5th Ave.	NYC	01202
01	908	22222	Jim	Elm Str.	MH	07974
44	131	33333	Ben	High St.	EDI	EH4 1DT
44	131	44444	Ian	High St.	EDI	EH4 1DT
44	908	44444	Ian	Port PI	MH	W1B 1JH
01	131	22222	Sean	3rd Str.	UN	01202

Пример 1. Пример взят из [1]. Рассмотрим таблицу 1, содержащую информацию о клиентах, включая их код страны (CC), телефонный код региона (AC), персональный номер (PN), имя (NM), улицу (STR), город (CT) и почтовый индекс (ZIP).

На этой таблице удерживается CFD φ вида: $\{(CC,\ 44), (ZIP,\ \_\_)\} \to (STR,\ \_\_)$. Строки $5-7$ подтверждают данную зависимость, а её уверенность составляет 1. Данная CFD устанавливает, что для клиентов из Великобритании ($CC = 44$) почтовый индекс (ZIP) однозначно определяет улицу (STR).

Пример 2. Пример взят из [4]. На данных таблицы 2 можно сформулировать CFD φ: $\{(Windy,\ false), (Outlook,\ \_)\} \to (Play,\ \_)$. Это означает, что если значение $Windy = false$, то результат (Play) определяется исключительно параметром $Outlook$. Для проверки рассмотрим

Таблица 2: Play tennis

id	Outlook	Temperature	Humidity	Windy	Play
1	sunny	hot	high	false	false
2	sunny	hot	high	true	false
3	overcast	hot	high	false	true
4	rain	mild	high	false	true
5	rain	cool	normal	false	true
6	rain	cool	normal	true	false
7	overcast	cool	normal	true	true
8	sunny	mild	high	false	false
9	sunny	cool	normal	false	true
10	rain	mild	normal	false	true
11	sunny	mild	normal	true	true
12	overcast	mild	high	true	true
13	overcast	hot	normal	false	true
14	rain	mild	high	true	false

строки, в которых $Windy = false$: $1, 3, 4, 5, 8, 9, 10, 13$. На этом множестве функциональная зависимость $(Outlook, \_) \to (Play, \_)$ не удерживается, при этом $I(\varphi, D) = 1$. Если исключить строку с номером 9, зависимость начнет удерживаться. В этом случае уверенность φ будет равна $\frac{8-1}{8} = 0.875$.

2.3. Анализ существующего решения

Исходный код для валидации CFD написан на Python и использует библиотеку Desbordante[1]. Он был создан в рамках курсовой работы [6] для демонстрации примера работы алгоритма, а не как основная задача проекта. Код выполняет обнаружение CFD в табличных данных, проверяет их выполнение и визуализирует результаты. Однако его архитектура не оптимальна для расширения и интеграции новых алгоритмов. Подход, использованный в коде, больше относится к «shallow integration», так как большая часть вычислений выполняется в Python, а не переносится на ядро C++. Это значительно снижает скорость работы алгоритма, а также доставляет неудобство конечному пользователю.

[1] https://github.com/Desbordante/desbordante-core/pull/349

В текущей реализации на Python отсутствует чёткое разделение между этапами работы алгоритма. Валидация CFD выполняется в функции `validate_cfd(...)`, которая одновременно обрабатывает данные и определяет строки, удовлетворяющие зависимости. Визуализация `visualize_cfd(...))` также совмещает несколько задач, включая анализ таблицы и форматирование вывода. В результате изменения в одном из компонентов требуют модификации других частей кода, что усложняет сопровождение и расширение системы. В процессе работы алгоритма создаются вспомогательные структуры данных, такие как `lhs_to_row_nums`, где строки группируются по значениям левой части CFD. Эта операция реализована с использованием `defaultdict(list)`, что при работе с крупными таблицами может приводить к значительному расходу памяти. Также в коде используются операции с `pandas.Series`, выполняемые в циклах без оптимизации, что негативно сказывается на производительности.

Новый код на C++ с Python-привязками позволит устранить эти проблемы, значительно повысив производительность и снизив требования к памяти, что особенно важно при работе с большими объемами данных.

3. Описание решения

3.1. Описание алгоритма

Опишем исходный алгоритм, представленный в работе [6]. Валидация некоторой CFD $\varphi = (X \to Y, t_p)$, выполняется следующим образом:

- Создание маски поддерживаемых кортежей;
- Группировка строк по значениям LHS;
- Определение наиболее частого значения RHS для каждой LHS;
- Вычисление поддержки и уверенности валидируемой CFD.

Алгоритм перебирает кортежи множества D и проверяет их соответствие условиям CFD φ. Кортежи, не удовлетворяющие условиям, исключаются из множества поддерживаемых CFD. После фильтрации данных кортежи группируются по значениям атрибутов множества X, определяющих LHS. Далее для каждой группы вычисляется наиболее часто встречающееся значение RHS.

Затем рассматриваются только те кортежи, которые входят в маску поддерживаемых кортежей. Оставляя в выборке только кортежи, содержащие наиболее частое значение RHS для соответствующего LHS, алгоритм гарантирует выполнение функциональной зависимости $(X \to Y)$ на оставшемся множестве. При этом отбрасывается минимально возможное количество кортежей $I(\varphi, D)$, что обеспечивает максимальное значение уверенности CFD.

3.2. Интеграция алгоритма

Рассмотрим теперь результат настоящей работы — новую версию алгоритма, внесенную в ядро Desbordante.

Реализация алгоритма разделена на два основных класса: `CFDVerifier` и `CFDStatsCalculator`. Их структура представлена

на диаграмме классов (см. рис. 1). Класс `CFDVerifier`, унаследованный от абстрактного класса Algorithm, отвечает за верификацию CFD. Он выполняет:

- Загрузку данных (`LoadDataInternal()`) — преобразует входную таблицу `input_table_` в объект `CFDRelationData`. Если таблица пуста, выбрасывается исключение;
- Обработку CFD-правил — строковые правила `string_rule_left_` и `string_rule_right_` преобразуются в числовые идентификаторы через `extract_item_ids()`;
- Выполнение проверки CFD (`ExecuteInternal()`) — выполняет последовательные этапы:
  1. Преобразование входных правил в числовые индексы;
  2. Запуск процедуры верификации CFD (`VerifyCFD()`);
  3. Вычисление статистик (`CalculateStatistics()`).

Для интеграции с Desbordante в `CFDVerifier` реализованы методы `MakeExecuteOptsAvailable()` и `RegisterOptions()`, обеспечивающие регистрацию и доступность параметров (`kTableOpt`, `kCFDRuleLeft`, `kCFDRuleRight`, `kMinimumSupport`, `kMinimumConfidence`).

Класс `CFDStatsCalculator` отвечает за вычисление статистик, необходимых для проверки CFD. В частности, он вычисляет:

- Поддержку (support) — количество строк, удовлетворяющих левой части правил;
- Уверенность (confidence) — долю корректных соответствий в выборке;
- Списки строк, удовлетворяющих или нарушающих CFD.

При создании экземпляра класса в него передаются `relation` и само CFD-правило. Основной метод `CFDVerifier::CalculateStatistics()` инициирует выполнение всех необходимых расчетов, вызывая методы

CFDStatsCalculator для формирования итоговых метрик. Алгоритм вычисления этих метрик был описан выше.

Дополнительно, в процессе верификации используется класс Highlight, который отвечает за группировку нарушающих кортежей в кластеры на основе значений LHS. Этот класс содержит:

- cluster_ — объект model::PLI::Cluster, представляющий кластер строк, нарушающих CFD;
- violating_rows_ — список индексов строк, входящих в кластер.

Кластеры формируются по совпадающим значениям атрибутов левой части зависимости, что позволяет выявлять систематические нарушения. Класс Highlight используется методами CFDVerifier для анализа нарушений CFD и облегчения интерпретации результатов валидации.

3.3. Python-привязки

Для интеграции алгоритма с Python используются привязки, реализованные с помощью pybind11 [3]. Функция BindCFDVerification() создает подмодуль cfd_verification в основном Python-модуле и привязывает ключевые методы класса CFDVerifier, а также класс Highlight, предназначенный для работы с кластерами нарушений:

- cfd_holds() — проверяет выполнение CFD;
- get_real_support() — возвращает поддержку;
- get_real_confidence() — возвращает уверенность;
- get_highlights() — возвращает список кластеров нарушений;
- get_num_clusters_violating_cfd() — возвращает количество кластеров нарушений;
- get_num_rows_violating_cfd() — возвращает количество строк, нарушающих CFD.

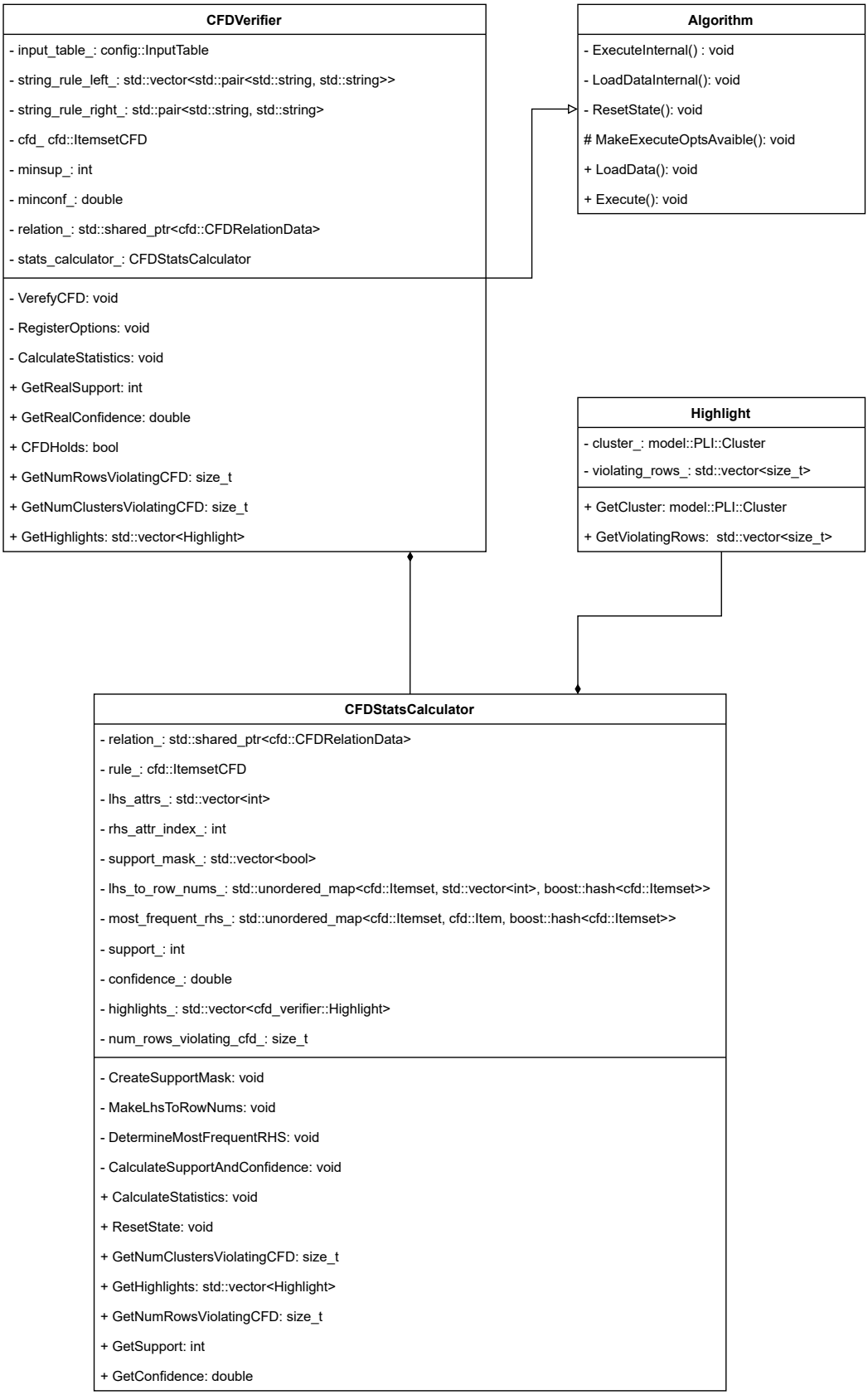

Рис. 1: Диаграмма классов

Кроме того, в модуле определен класс `Highlight`, который предоставляет доступ к информации о кластерах нарушающих кортежей:

- `cluster` — возвращает список индексов строк, входящих в кластер;
- `get_violating_rows()` — возвращает список индексов строк, нарушающих условия в кластере.

Привязка реализуется через вызов `BindPrimitiveNoBase<CFDVerifier>()`, обеспечивая доступ к методам в коде на Python. Это позволяет анализировать нарушения CFD и выявлять закономерности в ошибках прямо в среде Python.

3.4. Пример использования

В листинге 1 приведён пример использования библиотеки `desbordante` для проверки CFD на табличных данных с недвижимостью в различных городах. Код загружает данные, выполняет верификацию CFD и визуализирует результаты.

В этом примере проверяется, есть ли у зданий, которые стоили дорого при постройке, какая-либо зависимость от города и типа здания на основе данных из таблицы 3. `Support` в данном примере показывает, сколько раз в таблице встречаются здания с одинаковым городом и типом. `Confidence` отражает, насколько часто такие здания имеют стоимость “high”. Функция `scenario_incorrect_data` загружает данные, создает экземпляр алгоритма CFD-верификации и выполняет проверку с заданным порогом уверенности. Функция `print_results` выводит результаты проверки CFD. На **18-й строке** вызывается `verifier.get_highlights()`, получая список кластеров. На **20–23 строках** выполняется итерация по элементам `highlight.cluster`, извлекаются значения `lhs` и `rhs`, затем они выводятся (см. рис. 2).

Листинг 1: Пример проверки CFD

```python
import pandas
import desbordante

TABLE_PATH = 'examples/datasets/cfd_verification_datasets/city.csv'

def print_results(data, verifier, lhs, rhs):
    print(f"CFD: {lhs} -> {rhs}")
    print(f"CFD holds: {verifier.cfd_holds()}")
    print(f"Support: {verifier.get_real_support()}")
    print(f"Confidence: {verifier.get_real_confidence():.2f}")
    lhs_columns = [data[col] for col, _ in lhs]
    rhs_columns = [data[rhs[0]]]

    for i, highlight in enumerate(verifier.get_highlights(), start=1):
        print(f"{'\033[1;34m'}#{i} cluster: {'\033[0m'}")
        for el in highlight.cluster:
            lhs_values = [col.iloc[el] for col in lhs_columns]
            rhs_values = [col.iloc[el] for col in rhs_columns]
            print(f"{el}: {lhs_values} -> {rhs_values}")

def scenario_incorrect_data():
    table = pandas.read_csv(TABLE_PATH)
    algo = desbordante.cfd_verification.algorithms.Default()
    algo.load_data(table=table)

    lhs, rhs = [("City", "_"), ("BuildingType", "_")], ("BuildingCost", "high")
    algo.execute(cfd_rule_left=lhs, cfd_rule_right=rhs, minconf = 1)
    print_results(table, algo, lhs, rhs)
```

```
CFD: [('City', '_'), ('BuildingType', '_')] -> ('BuildingCost', 'high')
CFD holds: False
Support: 15
Confidence: 0.67
#1 cluster:
4: ['Chicago', 'Office'] -> ['high']
13: ['Chicago', 'Office'] -> ['medium']
#2 cluster:
1: ['Chicago', 'Apartment'] -> ['medium']
7: ['Chicago', 'Apartment'] -> ['low']
10: ['Chicago', 'Apartment'] -> ['medium']
#3 cluster:
0: ['Los Angeles', 'Apartment'] -> ['high']
6: ['Los Angeles', 'Apartment'] -> ['low']
9: ['Los Angeles', 'Apartment'] -> ['high']
```

Рис. 2: Пример вывода

Таблица 3: Данные о недвижимости

City	**Street**	**PostalCode**	**BuildingType**	**BuildingCost**
Los Angeles	Hollywood Blvd	90029	Apartment	high
Chicago	State Street	60601	Apartment	medium
New York	Broadway	10002	Apartment	high
Los Angeles	Sunset Blvd	90001	House	high
Chicago	Michigan Ave	60611	Office	high
New York	Wall Street	10005	Office	high
Los Angeles	Hollywood Blvd	90028	Apartment	low
Chicago	State Street	60602	Apartment	low
New York	Broadway	10001	Apartment	high
Los Angeles	Hollywood Blvd	90028	Apartment	high
Chicago	State Street	60601	Apartment	medium
New York	Broadway	10001	Apartment	high
Los Angeles	Sunset Blvd	90001	House	high
Chicago	Michigan Ave	60611	Office	medium
New York	Wall Street	10005	Office	high

4. Эксперимент

Основная задача эксперимента — оценить производительность алгоритма на больших объёмах данных. В ходе исследования проводится анализ времени верификации CFD в зависимости от характеристик входных данных, а также сравнение эффективности работы алгоритмов верификации для FD и CFD.
Для тестирования были выбраны два набора данных:

- `iowa1kk.csv` — 1 000 000 строк и 24 столбца (219.7 MB;
- `EpicMeds.csv` — 1 281 731 строк и 10 столбцов (56.8 MB).

Тестирование проводилось на следующей вычислительной системе:

- Операционная система: Linux 6.11.10-2-MANJARO;
- Процессор: AMD Ryzen 7 6800H;
- Оперативная память: 16 GiB.

Так как алгоритмы поиска CFD не могут эффективно работать на столь больших наборах данных, используемые правила были получены с помощью алгоритмов поиска FD. В исходных зависимостях (CFD) для каждого атрибута в pattern tuple содержится либо "_", либо конкретное значение. Однако алгоритм верификации FD работает только с функциональными зависимостями между наборами атрибутов и не учитывает шаблоны. Поэтому для тестирования на больших данных FD-правила были преобразованы в CFD следующим образом: во всех правилах CFD для всех атрибутов было установлено значение "_".

Время выполнения алгоритмов измерялось с учётом загрузки данных, так как при загрузке данных в `fd_verification` выполнялась предварительная обработка данных с использованием PLI. Это позволяло алгоритму FD-верификации значительно ускорять последующую проверку зависимостей за счёт PLI. В отличие от него, алгоритм CFD-верификации применяет другой подход с использованием хеш-таблицы, что могло привести к различиям во времени выполнения.

Для каждого теста проводилось 100 запусков, после вычислялись:

- Среднее время выполнения;
- Среднеквадратичное отклонение (std dev), отражающее разброс значений вокруг среднего.

Результаты экспериментов, представленные в таблицах 4, 5 и 6, позволяют сделать следующие выводы:

- Алгоритмы проверки CFD работают значительно быстрее, чем FD. В среднем время выполнения CFD в 5-6 раз меньше;
- Разброс значений (std dev) у FD выше, что говорит о менее стабильном времени выполнения.

Следует отметить, что в нашем случае скорость работы алгоритма верификации FD оказывается значительно ниже из-за предварительного вычисления PLI [5] для всей таблицы перед началом проверки. Это ускоряет последующую верификацию большого набора правил, так как позволяет повторно использовать предобработанные данные. Однако, если требуется проверить всего одно правило, построение PLI для всей таблицы может оказаться более затратным по времени по сравнению с подходом, применяемым в валидаторе CFD. Этот вопрос требует более детального изучения, чтобы оценить влияние PLI на производительность в различных сценариях.

Таблица 4: Таблица временных характеристик CFD на iowa1kk.csv

Условная функциональная зависимость	**Время, с**	**std dev, с**
{(Item Number, _), (State Bottle Cost, _), (Sale (Dollars), _)} → (State Bottle Retail, _)	4.14	0.08
{(Store Number, _), (Item Description, _), (Sale (Dollars), _), (Date, _), (Bottle Volume (ml), _)} → (Volume Sold (Liters), _)	5.11	0.06
{(Store Number, _), (Item Description, _), (Pack, _), (State Bottle Retail, _), (Sale (Dollars), _), (Volume Sold (Liters), _)} → (Bottle Volume (ml), _)	5.30	0.06

Таблица 5: Сравнение временных характеристик CFD и FD на iowa1kk.csv

Правило	**CFD**		**FD**	
	Время, с	**std dev, с**	**Время, с**	**std dev, с**
{(State Bottle Cost, _), (Item Number, _), (Sale (Dollars), _)} → (State Bottle Retail, _)	4.98	0.62	27.66	1.51
{(Date, _), (Store Number, _), (Item Number, _), (State Bottle Cost, _), (Sale (Dollars), _)} → (Bottles Sold, _)	5.37	0.06	28.09	1.99

Таблица 6: Сравнение временных характеристик CFD и FD на EpicMeds.csv

Правило	**CFD**		**FD**	
	Время, с	**std dev, с**	**Время, с**	**std dev, с**
(B, _), (C, _), (D, _), (E, _) → (H, _)	4.60	0.13	16.14	0.71
(A, _), (C, _), (E, _) → (G, _)	4.10	0.12	16.12	0.64

Заключение

В рамках данной работы были выполнены следующие задачи:

1. Проведён обзор предметной области, выполнен анализ существующих решений в проекте;
2. Реализован алгоритм валидации CFD на C++;
3. Разработаны Python-привязки для алгоритма валидации;
4. Создан пример использования, демонстрирующий работу алгоритма.

Реализация вошла в проект Desbordante. Pull request #544 доступен на GitHub[2].

[2] https://github.com/Desbordante/desbordante-core/pull/544

Список литературы

[1] Conditional Functional Dependencies for Data Cleaning / Philip Bohannon, Wenfei Fan, Floris Geerts et al. // 2007 IEEE 23rd International Conference on Data Engineering. — 2007. — P. 746–755.

[2] 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.

[3] Jakob Wenzel, Rhinelander Jason, Moldovan Dean. pybind11 — Seamless operability between C++11 and Python. — 2017. — URL: https://github.com/pybind/pybind11.

[4] Rammelaere Joeri, Geerts Floris. Revisiting Conditional Functional Dependency Discovery: Splitting the "C" from the "FD": European Conference, ECML PKDD 2018, Dublin, Ireland, September 10–14, 2018, Proceedings, Part II. — 2019. — 01. — P. 552–568. — ISBN: 978-3-030-10927-1.

[5] 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.

[6] Реализация интерфейсов и создание примера использования для инструмента поиска условных функциональных зависимостей в профайлере Desbordante : Rep. / Saint Petersburg State University ; Executor: Иван Волгушев : 2024. — URL: https://github.com/Desbordante/desbordante-core/blob/main/docs/papers/Bindings%2C%20example%20and%20CLI%20for%20CFD%20-%20Ivan%20Volgushev%20-%20autumn.pdf.