Санкт-Петербургский государственный университет

Кафедра информационно-аналитических систем

Группа 23.Б10-мм

Улучшение поддержки 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

Современные организации накапливают огромные объемы данных, и их эффективное использование становится ключевым фактором конкурентоспособности. Системы аналитики позволяют не только улучшать качество обслуживания клиентов, но и принимать более обоснованные стратегические решения. В этом контексте особую ценность приобретает профилирование данных — процесс изучения их структуры и свойств, направленный на выявление закономерностей и скрытых зависимостей.

Одним из фундаментальных инструментов профилирования являются зависимости между атрибутами данных. Функциональные зависимости (Functional Dependency, FD) описывают жесткие правила сопоставления значений атрибутов и широко применяются в задачах нормализации и контроля качества данных. Их обобщением служат условные функциональные зависимости (Conditional Functional Dependency, CFD), которые фиксируются только для определённых подмножеств записей. Такой подход позволяет выявлять более гибкие и практико-ориентированные закономерности, что делает CFD особенно востребованными при анализе табличных данных.

Несмотря на значимость CFD в задачах анализа и очистки данных, их извлечение и проверка остаются вычислительно сложными задачами, требующими как эффективных алгоритмов, так и удобных инструментов для практического применения. В проекте Desbordante [1] уже присутствует поддержка CFD, однако её возможности были ограничены: отсутствовало единое внутреннее представление зависимостей, интерфейсы модулей майнинга и валидации оставались разрозненными. Цель данной работы заключается в устранении указанных недостатков и формировании единой, целостной инфраструктуры, обеспечивающей более удобный и надёжный процесс майнинга и валидации CFD в рамках платформы Desbordante.

1. Постановка задачи

Целью работы является улучшение поддержки условных функциональных зависимостей (CFD) в проекте Desbordante за счёт устранения архитектурных ограничений и создания единой инфраструктуры для их майнинга и валидации.

1. проанализировать существующую архитектуру и выявить ошибки и недочёты, затруднявшие использование CFD;
2. интегрировать алгоритмы майнинга и валидации CFD в систему Python-биндингов, обеспечить единое внутреннее представление зависимостей в ядре C++;
3. переработать примеры использования CFD на Python, сделав их полноценными учебными и демонстрационными сценариями.

2. Обзор

Все определения в данном разделе были взяты из статьи [3] и приведены в предыдущей работе [5].

2.1. Обзор предметной области

Определение 1. Пусть A — множество атрибутов таблицы, каждый атрибут $B \in A$ имеет область значений $dom(B)$. Кортеж t определяется как набор пар $(B, t[B])$, где $t[B] \in dom(B)$ либо $t[B] =$"_". Под D будем понимать множество всех кортежей.

Определение 2. 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). Также, символ "_" называется шаблоном (wildcard).

Определение 3. Кортеж t_p поддерживает t, если для любого атрибута A выполнено $t_p[A] = t[A]$ или $t_p[A] =$ "_".

Определение 4. Поддержка (Support) условной функциональной зависимости φ — это количество кортежей из D, которые поддерживаются t_p, обозначается $supp(t_p, D)$.

Определение 5. 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 6. 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: Данные о недвижимости

№	City	Street	PostalCode	BuildingType	BuildingCost
1	Los Angeles	Hollywood Blvd	90029	Apartment	high
2	Chicago	State Street	60601	Apartment	medium
3	New York	Broadway	10002	Apartment	high
4	Los Angeles	Sunset Blvd	90001	House	high
5	Chicago	Michigan Ave	60611	Office	high
6	New York	Wall Street	10005	Office	high
7	Los Angeles	Hollywood Blvd	90028	Apartment	low
8	Chicago	State Street	60602	Apartment	low
9	New York	Broadway	10001	Apartment	high
10	Los Angeles	Hollywood Blvd	90028	Apartment	high
11	Chicago	State Street	60601	Apartment	medium
12	New York	Broadway	10001	Apartment	high
13	Los Angeles	Sunset Blvd	90001	House	high
14	Chicago	Michigan Ave	60611	Office	medium
15	New York	Wall Street	10005	Office	high

Пример 1. Рассмотрим таблицу 1, содержащую информацию о недвижимости. Сформулируем CFD: $(City,\ Los\ Angeles) \rightarrow (BuildingCost,\ high)$. Она отражает гипотезу о том, что все объекты недвижимости в Лос-Анджелесе должны иметь высокую стоимость строительства.

Для проверки рассмотрим все строки, где `City = Los Angeles`, это строки 1, 4, 7, 10, 13. Можем заметить: в седьмой строке значение `BuildingCost = low`, тогда как правило требует `high`. Таким образом, левую часть зависимости поддерживают пять записей, но одна запись не удовлетворяет правой части. Это даёт уверенность:

$$Conf(\varphi, D) = \frac{5-1}{5} = 0.80.$$

2.3. Анализ изначального кода

Изначальный код системы был разработан в рамках работ [4, 5] и имел ряд ограничений, затруднявших совместное использование модулей майнинга и валидации.

На тот момент в проекте существовала проблема несовместимости представлений условных функциональных зависимостей в различных компонентах. Алгоритм майнинга CFD использовал собственные внутренние структуры, но при этом возвращал наружу объекты типа `RawCFD`, предназначенные для строкового представления зависимостей. Валидатор не принимал такие объекты и работал исключительно с парами `CFDAttributeValuePair`, определёнными как `std::pair<std::string, std::string>`. В итоге результаты, полученные в майнере, нельзя было напрямую передавать в валидатор, что затрудняло работу с зависимостями.

Структура класса RawCFD в изначальной версии имела ограниченные возможности. Она позволяла хранить левую и правую части правила, а также выводить их в строковом виде, однако не содержала ряда методов, необходимых для полноценной работы с объектами. В результате RawCFD использовался в основном как вспомогательный класс для представления результата, но не подходил для интеграции с другими компонентами.

Валидатор CFDVerifier также имел ряд ограничений. Для задания правил CFD он ожидал раздельный ввод левой части `cfd_rule_left` и правой части `cfd_rule_right`, что было неудобно для пользователя.

В совокупности это приводило к избыточной зависимости от строковых представлений, отсутствию единого обменного формата для CFD и дублированию логики преобразований между компонентами. В результате интеграция майнинга и валидации была неудобной, а расширение системы — затруднено.

3. Описание решения

В ходе работы была переработана архитектура представления условных функциональных зависимостей CFD и унифицированы интерфейсы между модулями майнинга и верификации. Также существенно расширены Python-примеры и привязки, что сделало API последовательным и удобным.

3.1. Унификация представления CFD

Как уже было описано выше, ключевой проблемой была фрагментация представлений условных функциональных зависимостей и отсутствие универсального внутреннего формата. В новой версии был переработан класс `RawCFD`. Теперь он используется не только для отображения правил, но и как универсальный контейнер для CFD. Внесённые изменения включают:

- добавление конструктора по умолчанию (устранило проблему невозможности создавать пустые объекты);
- реализацию метода `ToString` для `RawItem`;
- перегрузку оператора равенства для `RawItem` и `RawCFD`.

В результате `RawCFD` стал единым внутренним представлением условных функциональных зависимостей, пригодным как для майнинга, так и для валидации.

3.2. Изменения в модуле CFDVerifier

В модуле `CFDVerifier` устаревшие параметры `string_rule_left_` и `string_rule_right_` были удалены, а их место занял единый объект `raw_cfd_rule_` типа `RawCFD`. Это позволило:

- устранить избыточность строковых представлений;
- передавать результат майнинга напрямую в валидатор;

- реализовать проверку входных данных через `RawItem` с поддержкой wildcard.

3.3. Python-привязки

Модуль привязок на базе `pybind11` [2] был существенно расширен. В файле `bind_cfd.cpp` реализованы следующие изменения:

- для `RawCFD::RawItem` добавлены:
  - конструктор `(attribute, value)`, где `value` может быть `None` (wildcard);
  - строковое представление (`__str__`/`__repr__`);
  - оператор равенства `__eq__` и поддержка хэширования `__hash__`;
- для `RawCFD` добавлены:
  - конструкторы (пустой; из `RawItems + RawItem`; из Python-структур `list[tuple], tuple`);
  - оператор равенства и `__hash__` (обеспечивает использование в множествах/словарях).
- добавлена вспомогательная функция `CreateRawCFDFromTuples` для создания `RawCFD` из Python-структур.

Эти изменения позволили использовать объекты `RawCFD` в Python в привычном стиле: создавать их из нативных структур данных, сравнивать и включать в коллекции.

В результате Python-привязки получили полноценную поддержку работы с условными функциональными зависимостями. Теперь объекты `RawCFD`, найденные майнером, могут напрямую передаваться в `CFDVerifier`.

3.4. Примеры использования

Важным дополнением к архитектурным изменениям стало создание двух крупных примеров на Python, иллюстрирующих процесс майнинга и валидации условных функциональных зависимостей. Ранее проект содержал только упрощённые демонстрации, но новые примеры превратились в полноценные учебные сценарии, поясняющие концепцию CFD и практические аспекты их применения.

Пример майнинга (mining_cfd.py). Скрипт `mining_cfd.py` был полностью переписан. Теперь он представляет собой руководство, включающее:

- представление концепции CFD и формальных определения из статьи [3];
- объяснение параметров майнинга;
- обзор доступных алгоритмов в библиотеке Desbordante;
- анализ датасета сотрудников (`employees.csv`), где выявляются ожидаемые закономерности: связь должности и уровня зарплаты, зависимости «отдел → локация», а также примеры нарушений;
- проверку найденных зависимостей с помощью модуля верификации, включая вычисление метрик (support, confidence) и визуализацию нарушений;
- раздел, посвящённый экспериментальной настройке параметров майнинга и объясняющий стратегию поиска «правильных» зависимостей.

Таким образом, данный пример не только демонстрирует работу алгоритма, но и обучает принципам применения CFD для анализа данных.

Пример валидации (verifying_cfd.py). Файл `verifying_cfd.py` также был переработан. Вместо минималистичного кода с ручными проверками теперь реализованы несколько сценариев:

- представление концепции CFD и формальных определения из статьи [3];
- создание и работа с объектами `CFD` в Python: строковое представление, сравнение, операции над множествами;
- объяснение датасета недвижимости и формулирование бизнес-гипотез о ценообразовании;
- сценарий проверки правил на примере города Лос-Анджелес: выявление нарушения, его объяснение и последующая коррекция данных;
- анализ альтернативных гипотез (например, все дома должны быть дорогими, все квартиры в Нью-Йорке должны быть высокобюджетными) с пошаговым выводом метрик и нарушений;
- заключительный раздел о том, как экспериментировать с валидацией CFD.

Оба примера используют цветовую подсветку вывода в консоли, что облегчает восприятие результатов.

3.5. Диаграмма классов

Диаграмма на рисунке 1 наглядно показывает эволюцию архитектуры после внесённых изменений. Видно, что класс `RawCFD` получил новые методы и операторы, модуль `CFDVerifier` был упрощён за счёт перехода к единому объекту `raw_cfd_rule_`, а в Python-привязках появились полноценные конструкторы, методы сравнения и хэширования.

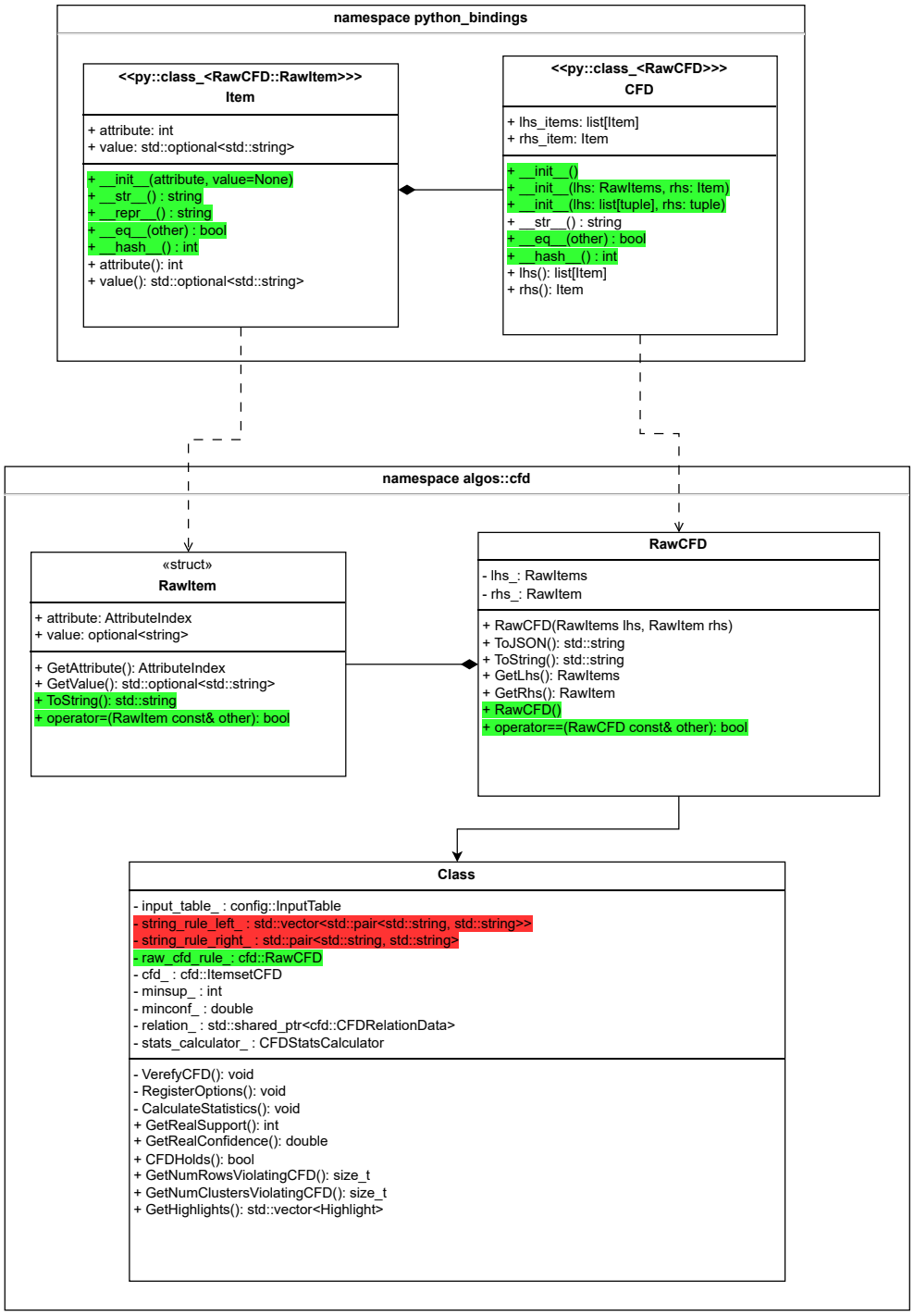

Рис. 1: Диаграмма классов интегрированного кода алгоритма

Добавленный метод

Удаленный метод

4. Апробация

Для демонстрации разработанных улучшений был подготовлен пример на языке Python, включающий как этап майнинга, так и верификации условных функциональных зависимостей. Следует отметить, что в рамках пул-реквеста #593 в основной репозиторий проекта были добавлены другие два сценария — `mining_cfd.py` и `verifying_cfd.py`, они подробно раскрывают концепцию CFD, включая интерактивные пояснения и визуализацию нарушений. Поскольку данные скрипты содержат несколько сотен строк, в рамках отчёта представлен сокращённый пример, сохраняющий основную логику взаимодействия модулей.

4.1. Описание примера

В листинге 1 представлен фрагмент скрипта `cfd_example.py`, демонстрирующего применение новых возможностей библиотеки Desbordante. Скрипт включает два логических этапа: **(1)** майнинг зависимостей и **(2)** их последующую проверку.

Листинг 1: Фрагмент скрипта cfd_example.py

```
def demo_mining():
    df = create_dataset()
    algo = desbordante.cfd.algorithms.Default()
    algo.load_data(table=df)
    algo.execute(cfd_minsup=6, cfd_minconf=0.75, cfd_max_lhs=2)
    cfds = algo.get_cfds()
    print(f"Найдено {len(cfds)} CFD:")
    for i, cfd in enumerate(cfds, 1):
        print(f"{i}. {cfd}")

def demo_verification(df):
    verifier = desbordante.cfd_verification.algorithms.Default()
    verifier.load_data(table=df)
    cfd_valid = desbordante.cfd.CFD(lhs=[(1, "Manager")], rhs=(3, "High"))
    verifier.execute(cfd_rule=cfd_valid)
    print(f"CFD выполняется: {verifier.cfd_holds()}")
    cfd_bad = desbordante.cfd.CFD(lhs=[(0, "IT")], rhs=(2, "NYC"))
    verifier.execute(cfd_rule=cfd_bad)
    print(f"Нарушений: {verifier.get_num_rows_violating_cfd()}")
```

4.2. Результаты выполнения

На рисунке 2 приведён вывод программы при запуске примера. Слева показан этап майнинга, где выявлены девять CFD, а справа — этап верификации, включающий проверку как корректных, так и нарушенных зависимостей.

```
======================================================================
                          CFD MINING
======================================================================

1.1. Исходные данные
  Department    Position Location  Salary
0         IT     Manager      NYC    High
1         IT   Developer      NYC  Medium
2         IT   Developer      NYC  Medium
3         HR     Manager       LA    High
4         HR  Specialist       LA     Low
5    Finance     Manager  Chicago    High
6    Finance     Analyst  Chicago  Medium
7         IT   Developer   Boston  Medium

1.2. Параметры майнинга
* Минимальная поддержка (cfd_minsup): 6
* Минимальная уверенность (cfd_minconf): 0.75
* Максимальный размер LHS (cfd_max_lhs): 2

1.3. Запуск CFD-майнера

Найдено 9 CFD:
  1. {(1, _)} -> (3, _)
  2. {(3, _)} -> (1, _)
  3. {(3, _),(2, _)} -> (1, _)
  4. {(1, _)} -> (0, _)
  5. {(3, _),(0, _)} -> (1, _)
  6. {(0, _)} -> (2, _)
  7. {(2, _)} -> (0, _)
  8. {(3, _),(0, _)} -> (2, _)
  9. {(1, _),(0, _)} -> (2, _)


1.4. Анализ структуры CFD
CFD: {(1, _)} -> (3, _)
  * Левая часть (LHS): [(1, _)]
  * Правая часть (RHS): (3, _)
  * Количество условий в LHS: 1
```

(a) Майнинг

```
======================================================================
                        CFD VERIFICATION
======================================================================

2.1. Проверка валидного CFD
CFD: {(1, Manager)} -> (3, High)

Результаты проверки:
  * CFD выполняется: True
  * Поддержка: 3
  * Уверенность: 1.00
  * Нарушений: 0

2.2. Проверка CFD с нарушениями
CFD: {(0, IT)} -> (2, NYC)

Результаты проверки:
  * CFD выполняется: True
  * Поддержка: 4
  * Уверенность: 0.75
  * Нарушений: 1
  * Кластеров с нарушениями: 1

2.3. Анализ нарушений
Найдено кластеров с нарушениями: 1

Кластер #1:
  Условие кластера: Department=IT
  Нарушающие строки: [7]
    Row 7: ['Developer'] -> Medium

2.4. Сравнение и хеширование CFD
CFD1: {(0, IT)} -> (2, NYC)
CFD2: {(0, IT)} -> (2, NYC)
CFD3: {(0, HR)} -> (2, LA)

CFD1 == CFD2: True
CFD1 == CFD3: False

Множество {['{(0, HR)} -> (2, LA)', '{(0, IT)} -> (2, NYC)']}
содержит 2 уникальных CFD.
```

(b) Валидация

Рис. 2: Вывод программы cfd_example.py

На этапе **майнинга** анализируется набор данных о сотрудниках, содержащий атрибуты `Department`, `Position`, `Location` и `Salary`. На этапе **верификации** каждая гипотеза проверяется на реальных данных. Для корректного правила `[(1, "Manager")] → (3, "High")` система сообщает об отсутствии нарушений, а для зависимостей, противоречащих данным, отображаются кластеры строк, нарушающих CFD. Это подтверждает, что разработанные изменения позволили упростить создание и передачу CFD между модулями и повысить наглядность результатов.

Заключение

Целью работы было улучшение поддержки условных функциональных зависимостей (CFD) в проекте Desbordante, для этого были выполнены следующие задачи:

- В результате анализа существующей архитектуры были выявлены ключевые проблемы: отсутствие единого внутреннего представления CFD, избыточная зависимость от строковых структур и несогласованность форматов между алгоритмами майнинга и валидации.
- В ходе архитектурных изменений реализована унификация представления CFD.
- Выполнена интеграция модулей CFD в систему Python-биндингов на базе `pybind11`.
- Переработаны примеры использования CFD на Python. Созданы два полноценных демонстрационных сценария (`mining_cfd.py` и `verifying_cfd.py`).

Реализация вошла в проект Desbordante. Pull request #593 доступен на GitHub.

Список литературы

[1] 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.

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

[3] 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.

[4] Волгушев Иван. Реализация интерфейсов и создание примера использования для инструмента поиска условных функциональных зависимостей в профайлере Desbordante. — 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.

[5] Федосеев Дмитрий. Реализация валидатора условных функциональных зависимостей в проекте Desbordante. — 2025. — URL: https://github.com/Desbordante/desbordante-core/blob/main/docs/papers/CFD%20Verification%20-%20Dmitry%20Fedoseev%20-%202024%20autumn.pdf.