Санкт-Петербургский государственный университет

Кафедра информационно-аналитических систем

Группа 23.Б08-мм

Обзор алгоритма поиска условных функциональных зависимостей CFDFinder

Кожуков Иван Сергеевич

Отчёт по учебной практике
в форме «Теоретическое исследование»

Научный руководитель:
ассистент кафедры ИАС Г.А. Чернышев

Санкт-Петербург
2025

Оглавление

1. Введение

Профилирование данных подразумевает процесс изучения их структуры, содержания и взаимосвязей. Одной из задач профилирования является поиск зависимостей между атрибутами данных, так как это позволяет выявить скрытые связи, улучшить их качество и устранить их избыточность (data cleaning) [2].

Одной из концепций, которой можно описать взаимосвязи, являются функциональные зависимости (functional dependencies, далее — ФЗ). ФЗ выражает отношение между атрибутами, в котором значения одного атрибута (или набора атрибутов) однозначно определяют значение другого атрибута. Например, в таблице с информацией о клиентах номер паспорта однозначно определяет имя клиента, что отражает связь между этими атрибутами.

Однако, в реальных данных, особенно в условиях больших объемов, часто встречаются ситуации, когда зависимости выполняются не для всех записей отношения, а лишь для их некоторого подмножества. Для описания таких связей была введена концепция условных функциональных зависимостей (conditional functional dependencies, далее — УФЗ). УФЗ представляют собой обобщение ФЗ, которое включает дополнительные условия, определяющие подмножества данных, на которых зависимость допустима. Например, зависимость между ценой товара и его категорией может быть справедлива только для определённого региона или периода времени. Такой подход позволяет значительно расширить возможности анализа данных, обеспечивая его большую точность и гибкость.

В данной работе проводится изучение алгоритма поиска УФЗ CFDFinder [5], а также новых подходов, которые он использует, для последующей реализации этого алгоритма на C++ в рамках платформы Desbordante [3].

2. Постановка задачи

Целью работы является изучение алгоритма CFDFinder. Для её достижения были поставлены следующие задачи:

1. Провести обзор предметной области;
2. Провести обзор алгоритма CFDFinder и принципа его работы;
3. Рассмотреть стратегии отсечения и расширения шаблонов, представленные автором;
4. Рассмотреть оценку производительности и масштабируемости алгоритма.

3. Обзор предметной области

В этом разделе приведены определения, описанные в работах [6, 5, 7], а также используется отношение CUST, представленное в [5], которое моделирует данные клиентов. Оно включает такие атрибуты, как номер телефона (состоящий из кода страны CC, кода города AC и номера телефона PN), имя клиента (NM) и адрес, состоящий из улицы (STR), города (CT) и почтового индекса (ZIP).

Таблица 1: Пример отношения CUST.

	CC	**AC**	**PN**	**NM**	**STR**	**CT**	**ZIP**
t_1	01	908	1111111	Mike	Tree Ave.	NYC	07974
t_2	01	908	2222222	Rick	Tree Ave.	NYC	07974
t_3	01	212	3333333	Joe	Elm Str.	NYC	01202
t_4	01	212	4444444	Jim	Elm Str.	NYC	01202
t_5	01	215	5555555	Ben	Oak Ave.	PHI	02394
t_6	44	131	6666666	Ian	High St.	EDI	EH4 1DT

3.1. Основные понятия

Пусть задано отношение r и схема R, определенная над множеством атрибутов $\mathcal{A}$.

Определение 1. ФЗ $f\colon X \to A$, *где* $X \subseteq R$ *и* $A \in R$, допустима *над* r, *если* $\forall\ t_i, t_j \in r : t_i[X] = t_j[X] \Longrightarrow t_i[A] = t_j[A]$. *Будем называть* X — левой частью или LHS (left-hand side), *а* A — правой частью или RHS (right-hand side).

Определение 2. ФЗ $f\colon X \to A$, *является* обобщением *другой ФЗ* $g\colon Y \to A$, *если* $X \subset Y$, *и является* специализацией, *если* $Y \subset X$.

Определение 3. ФЗ $f\colon X \to A$ минимальна, *если не существует такого* B, *что* $X \setminus B \to A$ *является допустимой ФЗ, то есть если не существует допустимого обобщения.*

Пример. Рассмотрим отношение, представленное в Таблице 1, в котором можно выделить следующие ФЗ:

$$f_1 : [CC, AC] \to CT$$

$$f_2 : [CC, AC, PN] \to [STR, CT, ZIP]$$

Зависимость f_1 отражает, что комбинация кода страны CC и кода региона AC однозначно определяет город CT. Зависимость f_2 означает, что, имея информацию о телефонном номере (составленном из CC, AC и PN), можно точно определить местоположение абонента, включая его улицу STR, город CT и почтовый индекс ZIP.

В отличие от этих зависимостей, ФЗ $f_3 : CT \nrightarrow ZIP$ недопустима, поскольку для кортежей t_2 и t_3 : $t_2[CT] = t_3[CT]$, но $t_2[ZIP] \neq t_3[ZIP]$. Это означает, что пара кортежей t_2, t_3 нарушает ФЗ f_3.

Определение 4. УФЗ φ : — *это пара* $(f : X \to Y,\ T_p)$*, где* $f : X \to Y$ — *функциональная зависимость, которая называется* встроенной в УФЗ φ*, и* T_p — *таблица из атрибутов* X *и* Y*, которая называется* таблицей шаблонов *(pattern tableau) и для которой выполнено:* $\forall A \in X \cup Y$ *и* $\forall t \in T_p : t[A] = a$*, где* $a \in dom(A)$*, или* $t[A] =$ "_", *где* "_" — переменная привязки *(variable binding), которая выражает, что для выполнения условия подходит любое значение из* $dom(A)$*. Кортеж* $t \in T_p$ *будем называть* шаблонным кортежем или шаблоном.

Определение 5. *УФЗ, у которой таблица шаблонов состоит только из констант, называется* константной, *иначе* — переменной.

Любую ФЗ можно представить в виде УФЗ, добавив к ней таблицу, которая состоит из одного шаблона с переменными привязки для каждого атрибута. Например, ФЗ f_1 можно расширить до УФЗ φ_{f_1}:

$$f_1 \equiv \varphi_{f_1} : [CC,\ AC] \to CT,\ T_p : \begin{array}{c|c||c} CC & AC & CT \\ \hline - & - & - \end{array}$$

Также можно выявить другие УФЗ, например:

$$\varphi_1 : [CC, ZIP] \to STR,\ T_{\varphi_1} : \begin{array}{c|c\|c} CC & ZIP & STR \\ \hline 44 & - & - \end{array}$$

УФЗ может уточнять ФЗ путем привязки (binding) констант. Например, уточним f_1:

$$\varphi_2 : [CC, AC] \to CT,\ T_{\varphi_2} : \begin{array}{c|c\|c} CC & AC & CT \\ \hline 01 & 908 & NYC \\ \hline 01 & 212 & NYC \end{array}$$

Определение 6. *Кортеж* $t \in r$ соответствует *шаблону* $t_p \in T_p$ *на множестве атрибутов* $S \subset R$, *обозначается* $t \asymp t_p$, *если* $\forall B \in S : t_p =$ "_" *или* $t_p[B] = t[B]$.

Определение 7. *Отношение* r удовлетворяет УФЗ φ $(R : X \to Y, T_p)$, *если* $\forall t_i, t_j \in r$ *и* $\forall t_p \in T_p : t_i[X] = t_j[X] \asymp t_p[X] \Longrightarrow t_i[Y] = t_j[Y] \asymp t_p[Y]$.

В отличие от ФЗ, которая может быть нарушена парой кортежей (multi-tuple violation, MTV), УФЗ может нарушиться одним кортежем (single-tuple violation, STV). То есть, STV происходит, если $\exists t \in r$ и $\exists t_p \in T_p$: $t[X] \asymp t_p[X]$, но $t[Y] \not\asymp t_p[Y]$, а MTV происходит, если $\exists t_i, t_j \in r$ и $\exists t_p \in T_p : t_i[X] = t_j[X] \asymp t_p[X]$, но $t_i[Y] = t_j[Y] \not\asymp t_p[Y]$. Например, при добавлении в отношение записей, представленных в Таблице 2, возникают нарушения УФЗ.

Таблица 2: Добавленные кортежи в отношение CUST.

	CC	**AC**	**PN**	**NM**	**STR**	**CT**	**ZIP**
t_7	44	131	7777777	Owen	Green St.	EDI	EH4 1DT
t_8	01	908	8888888	Jack	Central Ave.	New Providence	07974

Кортеж t_7 соответствует первому шаблону в T_{φ_1}, однако пара кортежей t_6 и t_7 нарушает УФЗ φ_1, поскольку в t_6 с тем же LHS встречается другой RHS (например, Green St.). Кроме того, нарушение УФЗ φ_2 можно наблюдать в кортеже t_8, который совпадает с LHS первого шаблона T_{φ_2}, но не удовлетворяет его RHS.

3.2. Показатели качества УФЗ

Сравнение УФЗ с точки зрения минимальности встроенной в них ФЗ имеет смысл только, если они охватывают одни и те же записи в отношении. Для сравнения УФЗ, охватывающих различные части, предлагаются следующие метрики качества.

Определение 8. Покрытием (cover) шаблона p — *множество кортежей, соответствующих p т.е.,*

$$cover(p) = \{t \in r : t[X] \asymp p[X]\}.$$

Определение 9. Локальная поддержка p (local support)*:*

$$local\_support(p) = \frac{|cover(p)|}{|r|}.$$

Определение 10. Глобальная поддержка T_p (global support)*:*

$$global\_support(T_p) = \frac{|\bigcup_{p \in T_p} cover(p)|}{|r|}.$$

Будем называть *поддержкой УФЗ* φ глобальную поддержу таблицы шаблонов φ.

Определение 11. Хранители (keepers) шаблона p — *множество кортежей, которые покрываются p и не приводят к нарушению встроенной ФЗ и нарушениям типа single-tuple violation.*

Определение 12. Локальная уверенность *p (local confidence):*

$$local\_confidence(p) = \frac{|keepers(p)|}{|cover(p)|}.$$

Определение 13. Глобальная уверенность T_p (global confidence)*:*

$$global\_confidence(T_p) = \frac{|\bigcup_{p \in T_p} keepers(p)|}{|\bigcup_{p \in T_p} cover(p)|}.$$

Будем называть *уверенностью УФЗ* φ глобальную уверенность таблицы шаблонов φ.

Определение 14. *УФЗ с уверенностью 1 называется* точной, *иначе —* приближенной.

3.3. Поиск УФЗ

Несмотря на то что обнаружение ФЗ является сложной задачей [2], этот процесс оказывается проще, чем поиск УФЗ. Это обусловлено меньшим размером пространства поиска и наличием хорошо изученных методов его сокращения. Например, алгоритм обнаружения ФЗ HyFD, предложенный в [7], работает на несколько порядков быстрее, чем наиболее эффективный алгоритм обнаружения УФЗ, описанный в работе [4], при использовании тех же наборов данных. Таким образом, обнаружение ФЗ может служить предварительным этапом для алгоритмов поиска УФЗ, если допустимые ФЗ для рассматриваемого набора данных заранее не известны. CFDFinder использует эту возможность и сначала вычисляет все минимальные ФЗ, используя алгоритм поиска HyFD, чтобы создать начальный набор кандидатов УФЗ.

Алгоритм HyFD использует фазу попарного сравнения кортежей для построения приближённого множества недопустимых ФЗ, которое называется *отрицательным покрытием* и которое может быть использовано для эффективного отсечения кандидатов ФЗ в фазах поиска по уровням и проверки (validation).

УФЗ, которые не относятся к стандартным ФЗ, могут быть сформированы только на основе кандидатов из отрицательного покрытия. В соответствии с аксиомами Армстронга, перечислить все наиболее специализированные (максимальные) недопустимые ФЗ (не-ФЗ) можно, начиная с нахождения отрицательного покрытия, и затем рекурсивно обобщая их. Наивным способом нахождения этого покрытия является обобщение всех ФЗ и проверка их специализаций относительно допустимых ФЗ. Однако этот процесс сопряжен с двумя проблемами: во-первых, требуется перечисление всех допустимых ФЗ, что может быть крайне трудоемким, и, во-вторых, в результате будет сгенерировано значительное количество не максимальных не-ФЗ, которые потребуют

дополнительной фильтрации. Автор алгоритма предлагает двухступенчатый подход извлечения отрицательного покрытия, который интегрируется в структуру HyFD.

4. Алгоритм

CFDFinder представлен как гибкий алгоритм поиска интересных УФЗ. Автор называет *интересными* точные УФЗ с лаконичной (concise) [5] таблицей шаблонов, то есть содержащей только несколько шаблонных кортежей и в достаточной степени поддерживающейся данными. Так же интересными считаются другие УФЗ, которые буду рассмотрены в подразделе 4.4. Можно выделить три основные компоненты, из которых состоит алгоритм:

1. Предварительная обработка. На этом шаге алгоритм обрабатывает входное отношение и строит необходимые структуры данных, а также выполняется поиск всех минимальных ФЗ с помощью алгоритма HyFD и проводится отбор кандидатов.

2. Обход пространства поиска. На данном этапе происходит обход множества кандидатов УФЗ по уровням.

3. Генерация таблицы шаблонов. Эта компонента генерирует для данного кандидата УФЗ таблицу шаблонов в соответствии с выбранными стратегиями отсечения и расширения шаблонов.

4.1. Предварительная обработка

На этапе предварительной обработки исходные данные преобразуются в две компактные структуры, необходимые для работы HyFD: $plis$ и $pliRecords$. Первая структура $plis$ представляет собой массив *списков позиций индексов* (position list index, pli). Каждый такой список для атрибута X формирует группы кортежей в эквивалентные классы на основе значений атрибутов из набора X. Таким образом, два кортежа t_1 и t_2 относятся к одному классу эквивалентности, если выполняется условие: $\forall A \in X : t_1[A] = t_2[A]$. Эти классы называются *кластерами*, так как они объединяют записи с одинаковыми значениями. Чтобы уменьшить объем хранимой информации, кластеры, содержащие только один элемент, в pli не записываются.

С помощью структуры *plis* создаётся сжатое представление записей в виде объекта *pliRecords*. Каждая сжатая запись представлена массивом идентификаторов кластеров, где каждое поле указывает на кластер соответствующей записи для атрибута $A \in [0, numAttrs]$. Эти представления извлекаются из *plis*, которые уже связывают идентификаторы кластеров с идентификаторами записей для каждого атрибута. Сжатые записи используются на этапе отбора (sampling) для обнаружения нарушений ФЗ и на этапе проверки (validation) для определения идентификаторов LHS и RHS кластеров для конкретных записей.

Затем эти структуры используются для извлечения отрицательного покрытия, которое используется для перечисления всех кандидатов УФЗ. Для этого CFDFinder использует фазу индукции (induction) кандидатов HyFD, которая заключается в преобразовании не-ФЗ, найденных в процессе отбора, в соответствующие кандидаты на минимальные ФЗ. Во время работы фазы индукции собираются те найденные не-ФЗ, которые не могут быть обобщены далее на данном этапе из-за уже построенного множества допустимых ФЗ. Также собираются не-ФЗ, которые представляют собой прямые обобщения минимальных ФЗ. Эти кандидаты сохраняются в префиксном дереве, при этом те из них, которые имеют специализацию, уже включенную в набор кандидатов, удаляются. Такой метод позволяет избежать необходимости полного перебора множества всех допустимых ФЗ.

4.2. Обход пространства поиска

Алгоритм 1 описывает процесс обхода пространства поиска, реализуемый CFDFinder. Для организации кандидатов на УФЗ используется структура уровней, основанная на количестве атрибутов в их LHS (строки 2–3). На начальном этапе на соответствующий уровень добавляются все максимальные не-ФЗ (строки 4–6). После этого алгоритм начинает обработку, начиная с верхнего уровня (строка 7), и пытается создать допустимую таблицу шаблонов для каждого кандидата на основе заданных параметров с помощью функции *GenerateTableau* (строка 10).

В случае успешной генерации таблицы (строка 11) на следующий уровень добавляются новые кандидаты УФЗ, формируемые из всех непустых подмножеств LHS и атрибута в RHS, при условии их отсутствия среди уже существующих кандидатов (строка 12). После завершения обработки текущего уровня алгоритм переходит к следующему (строка 14), продолжая до тех пор, пока не будут обработаны все уровни.

Алгоритм 1: Обход множества кандидатов УФЗ [5]

Ввод: Множество всех максимальных не-ФЗ N, отношение R

```
1  Function TraverseLattice(N, R):
2      for 1 ≤ i ≤ |R| do
3          L_i ← ∅;
4      for n = (X_n, A_n) ∈ N do
5          l ← |X_n|;
6          L_l ← L_l ∪ {n};
7      p ← |R|;
8      while p > 0 do
9          for c = (X_c, A_c) ∈ L_p do
10             T ← GenerateTableau(c);
11             if T ≠ ∅ then
12                 S ← {(X', A_c) | X' ⊆ X_c ∧ |X_c \ X'| = 1};
13                 L_{p−1} ← L_{p−1} ∪ S;
14         p ← p − 1;
```

4.3. Генерация таблицы шаблонов

Построение таблицы шаблонов для кандидата УФЗ не зависит от метода, который используется для получения этого кандидата. В алгоритме CFDFinder был доработан и модифицирован жадный алгоритм, предложенный в работе [6], который мы будем называть *оригинальным подходом*. Этот алгоритм создает таблицу шаблонов, близкую к оптимальной, для данного кандидата УФЗ с учётом заданных глобальных порогов поддержки s' и уверенности $\hat{c}$. Здесь под оптимальностью подразумевается максимальность глобальных показателей поддержки и уверенности. Авторы доказывают, что задача максимизации глобаль-

ных порогов является NP-полной и что для нее не существует приближенного решения. Однако смещение фокуса на оптимизацию локальной, а не глобальной уверенности позволяет придумать жадное приближение за полиномиальное время. Это возможно благодаря утверждению, что таблица, содержащая только шаблоны с локальной уверенностью, превышающей порог глобальной уверенности, также будет иметь глобальную уверенность, превышающую порог.

Алгоритм CFDFinder значительно расширяет оригинальный подход, позволяя использовать модульные стратегии отсечения и расширения для нескольких типов шаблонов, что делает его гибким и адаптируемым для разных задач и наборов данных.

Для начала рассмотрим несколько определений, которые ввели авторы оригинального подхода. Рассмотрим два шаблона: $p_1 = (a, b, \_, d)$ и $p_2 = (a, \_, \_, \_)$, заданные для атрибутов (A, B, C, D). Шаблон p_2 *более общий* по сравнению с p_1, так как вместо сопоставления конкретной константы $b \in dom(B)$ он допускает любое значение из $dom(B)$. Аналогично, шаблон p_2 более общий по атрибуту D, поскольку не накладывает на него ограничений. В то же время p_1 можно считать *более специализированным*, так как он ограничивает значения в $dom(B)$ и $dom(D)$, используя привязку конкретных значений. Отношения обобщения и специализации шаблонов напрямую влияют на их покрытие. Так например, если множество атрибутов, связанных с константой, в p_2 является подмножеством аналогичного множества в p_1, то $cover(p_2) \subseteq cover(p_1)$. Будем рассматривать шаблоны, которые отличаются только одним связыванием, так как они предоставляют минимальные изменения в покрытии.

Определение 15. *Шаблон p_p , который содержит ровно на одну привязанную константу больше, чем шаблон p_c, называется* родительским *шаблоном для p_c. Аналогично, шаблон p_c, содержащий ровно на одну привязанную константу меньше, чем шаблон p_p, называется* дочерним *шаблоном для p_p. Один шаблон может иметь несколько родительских и дочерних шаблонов.*

Определение 16. *Шаблон, который содержит только переменные*

привязки "_" для всех атрибутов, называется нулевым шаблоном.

Алгоритм 2: Генерация таблицы шаблонов [5]

Ввод : кандидат УФЗ d, экземпляр отношения r, стратегии отсечения ps и расширения es

Вывод: Таблица шаблонов T_p

```
1  Function GenerateTableau(d, r, ps, es):
2      T_p ← ∅; F ← ∅; p_0 ← es.GenerateNullPattern(d);
3      foreach t ∈ r do
4          cover(p_0) ← cover(p_0) ∪ {t};
5          kp(p_0) ← kp(p_0) + kp(t);
6          ct(p_0) ← ct(p_0) + ct(t);
7      margSupp(p_0) ← ct(p_0);
8      conf(p_0) ← kp(p_0) \ ct(p_0);
9      F ← F ∪ {p_0};
10     while F ≠ ∅ and ps.Continue(T_p) do
11         p ← arg max_{x∈F} margSupp(x);
12         F ← F \ {p};
13         if ps.AddPattern(p) then
14             T_p ← T_p ∪ {p};
15             foreach p' ∈ F do
16                 cover(p') ← cover(p') \ cover(p);
17                 margSupport(p') ← Σ_{t∈cover(p')} ct(t);
18                 if not ps.ConsiderPattern(p') then
19                     F ← F \ {p'};
20         else
21             foreach c ∈ es.expand(p) do
22                 if ps.Valid(c) then
23                     cover(c) ← DetermineCover(cover(p), c);
24                     margSupport(c) ← |cover(c)|;
25                     if ps.ConsiderPattern(c) then
26                         F ← F ∪ {c};
27     return T_p;
```

Путём итеративной специализации нулевого шаблона можно сгенерировать все возможные шаблоны для экземпляра отношения. Алгоритм 2 представляет собой изменённую версию оригинального подхо-

да с дополнениями, выделенными серым цветом. Выбранные стратегии отсечения и расширения шаблонов обозначаются как ps и es соответственно.

На начальном этапе алгоритм формирует нулевой шаблон из атрибутов LHS кандидата УФЗ (строки 3–6). Важным отличием от оригинального подхода является то, что процесс создания нулевого шаблона теперь делегирован выбранной стратегии расширения. Сначала все уникальные кортежи отношения добавляются в покрытие нулевого шаблона, так как по определению оно совпадает со всеми кортежами. Обозначим размер набора хранителей как $kp(p)$ и размер покрытия как $ct(p)$. В данном контексте $ct(t)$ обозначает количество точных дубликатов кортежа t в исходном отношении, или просто 1 для уникальных кортежей. $kp(t)$ равно числу хранителей, связанных с данным кортежем, то есть, если t не нарушает встроенную ФЗ, тогда $kp(t) = ct(t)$, в противном случае $kp(t) = 0$. После этого рассчитываются поддержка и уверенность для нулевого шаблона и он добавляется в *границу* (frontier) F (строки 7–9). Она представляет собой множество шаблонов-кандидатов, которое сортируется по *маржинальной поддержке* (marginal support), то есть дополнительной поддержке, которую этот шаблон предоставляет таблице. Это позволяет жадному алгоритму всегда выбирать лучший шаблон с точки зрения маржинальной поддержки.

Далее функция $continue(T_p)$, реализованная стратегией отсечения ps, определяет, можно ли улучшить таблицу T_p в соответствии со стратегией, т.е. может ли алгоритм завершиться раньше. Например, в оригинальном подходе эта функция представлена проверкой того, что совокупная поддержка всех шаблонов таблицы не превышает заданный порог поддержки. Если улучшение возможно, алгоритм продолжает обрабатывать шаблон-кандидат с наибольшей маржинальной поддержкой, содержащийся в границе (строки 11–12). Если шаблон подходит для добавления в таблицу, то есть удовлетворяет заданным ограничениям, то он добавляется в неё, а его покрытие удаляется из покрытий всех остальных шаблонов-кандидатов в границе, гарантируя, что маржинальная поддержка шаблонов, покрывающие те же кортежи, уменьша-

ется (строки 15–19). Этот шаг гарантирует, что перекрытие шаблонов в таблице минимально, а глобальная поддержка таблицы увеличивается на маржинальную поддержку каждого добавленного шаблона. Также для оставшихся шаблонов-кандидатов с помощью функции стратегии отсечения $considerPattern(p')$ проверяется, следует ли сохранить шаблон $p' \in F$ для добавления в таблицу на позднем этапе, то есть имеет ли смысл сохранять этот шаблон в границе, и, если не имеет, p' удаляется из неё. Например, шаблоны, которые не покрывают никакие дополнительные кортежи в таблице, могут быть исключены из границы, чтобы повысить производительность.

Если выбранный шаблон не подходит для добавления, то он специализируется, то есть для него вычисляются все дочерние шаблоны p, и затем добавляются в границу (строка 26), если шаблон подходит по параметрам для рассмотрения. Так как дочерние шаблоны могут иметь несколько родителей, автор добавляет условие на строке 22, которое в зависимости от стратегии гарантирует, что все родительские шаблоны дочернего должны быть рассмотрены до того, как он будет добавлен в границу. Покрытие дочерних шаблонов может быть вычислено из покрытия их родительского шаблона, что помогает снизить вычислительные затраты на вычисление покрытия по всему экземпляру отношения (строка 23).

В конечном итоге алгоритм либо возвращает таблицу шаблонов для кандидата, либо завершает выполнение без результата, если глобальный порог поддержки не удаётся достичь до того, как все шаблоны-кандидаты будут использованы.

Важно отметить, что адаптированная версия алгоритма при выборе соответствующей стратегии, которая называется *унаследованной* (legacy pruning strategy), ведет себя так же, как и оригинальный подход.

4.4. Отсечение кандидатов

В качестве основной стратегии отсечения автор предлагает комбинацию двух идей, описанных далее.

Отсечение по снижению поддержки (support drop pruning) Найденные УФЗ, аналогично кандидатам на ФЗ, могут быть упорядочены в виде решетки на основе левой части их встроенных ФЗ. CFDFinder добавляет к критериям интересности УФЗ следующее условие: если обобщение УФЗ приводит к значительному снижению поддержки или уверенности, то УФЗ над этим снижением представляет интерес.

На рисунке 1 представлен фрагмент решетки УФЗ, построенной для примерного отношения $R = (A, B, C, D, E, \ldots)$. Каждый узел этой решетки обозначает УФЗ и включает встроенную ФЗ, соответствующую таблицу шаблонов, а также показатели качества. Автор отмечает, что данный подход применим для любых метрик качества, которые монотонно зависят от размера LHS. Для CFDFinder используется глобальное значение поддержки, так как глобальная уверенность равна единице для каждого результата.

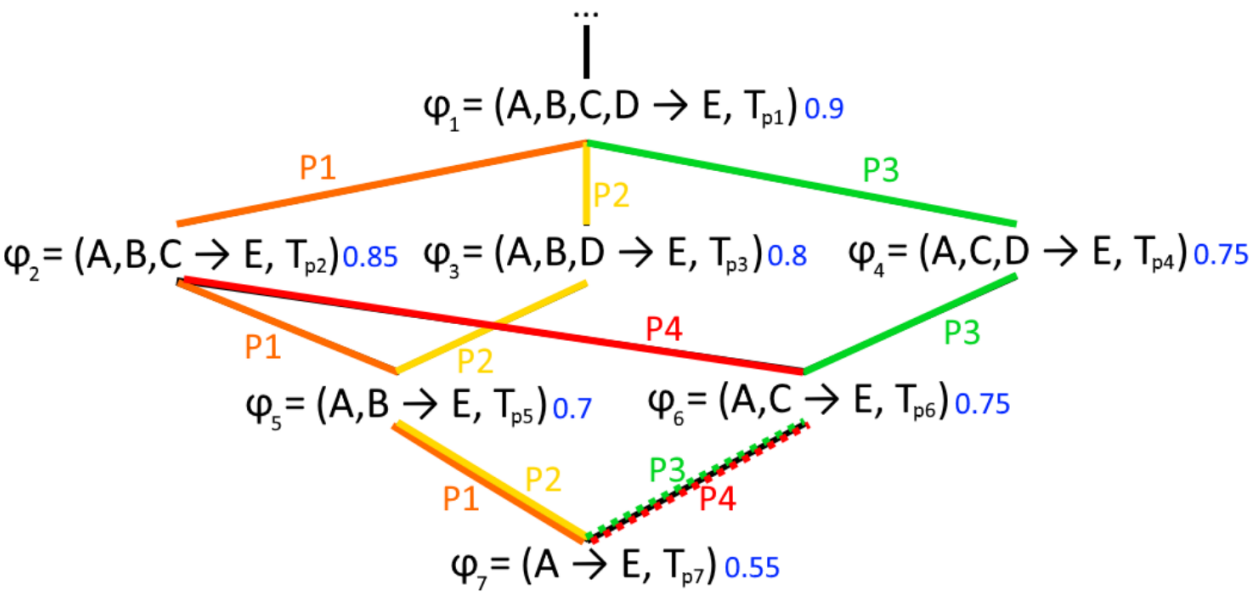

Рис. 1: Фрагмент решетки УФЗ [5]

Благодаря обходу решетки сверху вниз более специализированные результаты всегда вычисляются до их общих потомков. Таким образом,

можно прокладывать пути через решетку результатов, наблюдать изменения в глобальной поддержке и отсекать пути, которые заведомо не соответствуют установленным ограничениям качества, сокращая таким образом пространство поиска.

Рассмотрим четыре пути в решетке результатов на рисунке 1. Путь P_1 представляет последовательное удаление атрибутов D, C и B из левой части их встроенной ФЗ, что снижает общую поддержку УФЗ. P_2 начинается с того же узла, что и P_1, но удаляет атрибуты в другом порядке, посещая узел φ_3. Однако пути снова объединяются на их третьем узле.

Следует отметить, что для каждого кандидата создается только одна таблица шаблонов, независимо от того, какие предки он имеет. Это означает, что независимо от порядка удаления атрибутов два пути, соединяющиеся в одном узле, всегда будут иметь одинаковую поддержку в этом узле.

Путь P_3 последовательно удаляет атрибуты B,D,C, снова соединяясь с P_1 и P_2 в φ_8. Однако он демонстрирует другое поведение результата по сравнению с другими путями. В то время как P_1 и P_2 никогда не теряют более 0.15 поддержки на каждом удалении, обобщение φ_6 до φ_7 показывает снижение на 0.2. Аналогичное снижение показывает путь P_4.

В зависимости от задачи этот компромисс между размером LHS и поддержкой может обрабатываться по-разному. Автор предлагает отсекать пути на ребрах, где снижение поддержки превышает заданный порог. В примере с порогом $\delta_{max} = 0.15$ каждое удаление допускает снижение поддержки не более чем на 0.15. Очевидно, что ребро между φ_6 и φ_7 нарушает этот порог. Таким образом, пути P_3 и P_4 обрываются на этом ребре, и CFDFinder объявляет φ_6 как интересную УФЗ. Поскольку φ_7 не может быть обобщена дальше и не имеет потомков в решетке результатов, она также считается интересной. Все остальные УФЗ исключаются при использовании этого подхода.

Кратко, данный метод представляет собой следующее действие: для заданного порога δ_{max} отсечение по снижению поддержки строит пути

обобщения УФЗ из набора результатов, обрывает ребра, где снижение поддержки превышает порог, и сообщает только результаты на концах путей.

Снижение глобальной поддержки можно обнаружить во время обхода решетки кандидатов (подраздел 4.2). Если таблица шаблонов, созданная для кандидата, не соответствует порогу относительно его родительского УФЗ, потомки кандидата не добавляются на следующий уровень, что сокращает поисковое пространство. Однако, если потомок кандидата доступен через другой путь с другим поведением поддержки, он может быть повторно добавлен в набор кандидатов.

Установка порога снижения поддержки $\delta_{max} = 0$ позволяет находить УФЗ, для которых удаление атрибутов из LHS не снижает глобальную поддержку. Ослабление ограничений за счет увеличения порога позволяет CFDFinder находить УФЗ с более низкой поддержкой по сравнению с корневым узлом пути. При $\delta_{max} = 1$ фактически отключается отсечение путей и минимизируется размер LHS результата, сохраняя при этом ограничение на глобальную уверенность. Стоит отметить, что порог снижения поддержки не влияет на абсолютные значения поддержки результата.

Отсечение по минимальному приросту поддержки (support-gain pruning)

Автор выделяет два недостатка оригинального подхода, первым из которых является использование глобальных порогов для поддержки и уверенности, так как выбор оптимального порога для поиска интересных УФЗ часто сводится к методу проб и ошибок. Завершение алгоритма генерации таблицы только после проверки всех возможных кандидатов или достижения глобального порога поддержки может привести к ситуации, когда в таблицу добавляется большое количество шаблонов с минимальной поддержкой. Это вызывает переобучение (overfitting) данных, что значительно ухудшает как время работы алгоритма, так и качество итоговой таблицы шаблонов.

Если порог поддержки выбран слишком высоким, алгоритм может завершиться, так и не достигнув порога, что делает выполненные вы-

числения бесполезными. С другой стороны, если порог поддержки установить слишком низким, жадный алгоритм завершит работу раньше, хотя в таблицу еще можно было бы добавить интересные шаблоны.

Дополнительной проблемой является определение критериев “интересности” УФЗ. Например, в работе [6] предлагается использовать пороги поддержки в диапазоне от 0.3 до 0.9, однако как на примере показывает автор, некоторые интересные УФЗ все же могут иметь низкую поддержку и при таких ограничениях не будут обнаружены.

Метод отсечения по минимальному приросту поддержки позволяет избежать этих проблем, предлагая компромисс между глобальной поддержкой, краткостью таблиц и интересностью шаблонов. Этот метод задает нижнюю границу локальной поддержки, при нарушении которой шаблоны-кандидаты и их дочерние шаблоны исключаются. В соответствии с определением интересных УФЗ порог уверенности равен единице. Это означает, что таблица шаблонов должна эффективно устранять все нарушения с помощью отбора, а не допускать их. Также такой подход позволяет использовать идею отсечения по падению поддержки, что позволяет избежать необходимость выбора между поддержкой и уверенностью. Также для ограничения размера таблицы CFDFinder предоставляет возможность задания верхней границы количества шаблонов в таблице.

Такое отсечение помогает избежать переобучения и при этом позволяет находить УФЗ с низкой поддержкой. Установка нижнего порога локальной поддержки в один кортеж фактически отключает отсечение и при этом максимизирует глобальную поддержку при минимальном количестве шаблонов, что по утверждению автора невозможно в других алгоритмах обнаружения УФЗ без точного знания порога результата. С другой стороны, если нижний порог установить равным $|r|$, алгоритм обнаруживает только такие УФЗ, которые содержат один шаблон, покрывающий всё отношение. Это возможно только для функциональных зависимостей или эквивалентным им константных УФЗ.

4.5. Расширение шаблонов

На этапе расширения в алгоритме генерации таблицы рассматриваются шаблоны, которые не соответствуют установленным ограничениям качества, но всё ещё не могут быть отсечены. Эти шаблоны обладают достаточной локальной поддержкой, но недостаточной локальной уверенностью для добавления в таблицу. Такие шаблоны дорабатываются путём специализации в новые кандидаты-шаблоны, которые затем добавляются в границу. При этом не имеет значения, каким образом расширяется шаблон при условии, что покрытие дочернего шаблона является подмножеством покрытия родительского. Стратегия расширения рассматривается как метод перечисления всех возможных шаблонов в наборе данных, начиная с тех, что имеют наибольшую локальную поддержку, и заканчивая минимальной поддержкой. CFDFinder использует три стратегии расширения для различных типов условий, которые были предложены в работе [6]:

1. Расширение условий через добавление констант;
2. Расширение для отрицательных условий;
3. Расширение для условий с диапазоном.

Расширение через добавление констант. Данная стратегия применима для классических условий из раздела 3. Если шаблон не обладает достаточной уверенностью, его покрытие используется для создания новых шаблонов. В этом случае каждое значение атрибута, которое не связано с константой в данном шаблоне, преобразуется в новую константу из домена этого атрибута. Такой метод позволяет перебрать все возможные комбинации привязок констант. Алгоритм 3 описывает этот процесс.

Количество шаблонов-кандидатов, добавляемых в границу, при таком расширении возрастает на значительную величину. В худшем случае, если все значения $v \in V$ различны, где $V(t_p) = \{A \in R \mid t_p[A] = \text{“\_”}\}$, общее число дочерних шаблонов будет равно $|V| \cdot |r|$. Однако в

Алгоритм 3: Расширение через добавление констант [5]

```
   Ввод  : шаблон p
   Вывод : все дочерние шаблоны children
 1 Function expand(p):
 2     children ← ∅;
 3     foreach t ∈ cover(p) do
 4         foreach A ∈ attr(p) do
 5             if p[A] = "_" then
 6                 c ← p;
 7                 c[A] ← t[A];
 8                 children ← children ∪ {c};
 9             end
10         end
11     end
12     return children;
```

такой ситуации каждый из этих шаблонов охватывает только одну запись, что приводит к их исключению при использовании отсечения на основе минимального прироста поддержки. Быстрый рост количества новых шаблонов является причиной того, что расширение происходит только, когда шаблон выглядит перспективным [6].

Расширение для отрицательных условий.

Чтобы дополнительно реализовать отрицательные условия, необходимо изменить строку 7 в алгоритме 3. Помимо положительного условия, например $A = 4$, создается дополнительное условие $A \neq 4$. Это ведет к удвоению пространства поиска шаблонов и существенным изменениям в процессе отсечения. Если обозначить покрытие родительского шаблона p как $cover(p)$, тогда $cover(c_{[A]})$, то есть покрытие дочернего шаблона с положительной привязкой в атрибуте A, равно $cover(p) - cover(c_{[\neg A]})$. В случае низкой поддержки $c_{[A]}$, шаблон с условием $c_{[A]}$ будет иметь высокую поддержку, и наоборот. Это приводит к тому, что при отсечении возможно исключить не больше половины шаблонов-кандидатов. На практике, по замечанию автора, покрытия с положительными привязками обычно меньше, чем их обратное с отрицательными, и поэтому отсечение сказывается еще более негативно.

Расширение для условий с диапазоном. Эта стратегия используется следующее представление переменной привязки. В таблице шаблонов T_p для УФЗ φ: $(X \longrightarrow Y,\ T_p)$ для каждого шаблона $t_p \in T_p$ и для каждого атрибута $A \in X \cup Y$ $t_p[A] = [a_l, a_r]$, где $a_l \in dom(A)$ — нижняя граница, а $a_r \in dom(A)$ — верхняя граница и $a_l < a_r$. Кортеж t соответствует этому условию, если $a_l \leq t[A] \leq a_r$. Таким образом, привязки с диапазоном обобщают переменные привязки при $a_l = min(dom(A)) \wedge a_r = max(dom(A))$, а также константные привязки при $a_l = a_r = a \in dom(A)$.

Для этого типа привязок нулевой шаблон p_0 инициализируется диапазонными условиями, охватывающими весь домен каждого атрибута. При расширении для каждого атрибута A, у которого $p[A] = [a_l, a_r]$, создаются два новых дочерних шаблона: $[a_l + 1, a_r]$ и $[a_l, a_r - 1]$ при условии $a_l < a_r$. Таким образом, мы удаляем по одному значению с каждой стороны диапазона, чтобы получить два новых диапазона для двух новых дочерних шаблонов и перечислить все смежные диапазоны. Вычисление родительского шаблона происходит аналогично: диапазон $[a_l, a_r]$ обобщается на $[a_l - 1, a_r]$ и $[a_l, a_r + 1]$, если $min(dom(A)) \leq a_l \leq a_r \leq max(dom(A))$.

Условия с диапазоном значительно увеличивают количество рассматриваемых шаблонов-кандидатов, особенно если допускаются произвольные поддиапазоны и отношение содержит много атрибутов. Поэтому автор рекомендует использовать данный тип условий только для атрибутов с небольшим размером домена.

4.6. Адаптации алгоритма

CFDFinder поддерживает несколько расширений, которые являются интересными сами по себе. В этом подразделе автор описывает два таких расширения.

Условия на RHS. В своей базовой версии CFDFinder строит таблицы шаблонов с условиями только для LHS встроенной ФЗ, так как условия для RHS не влияют ни на локальную, ни на глобальную поддерж-

ку. При добавлении RHS условий увеличивается количество нарушений, что приводит к вырождению этапа генерации таблицы в перечисление гораздо большего количества шаблонов-кандидатов, что снижает производительность. Поиск условий для RHS, однако, всё же может полезен для построения константных УФЗ.

Поиск таких условий может быть выполнен аналогично поиску условий LHS. При этом необходимо адаптировать только генерацию нулевого шаблона и обнаружение нарушений. Нулевой шаблон должен дополнительно включать атрибут RHS в качестве еще одной переменной привязки. При сопоставлении шаблонов с кластером должны сопоставляться только условия LHS. Однако теперь при поиске нарушений необходимо учитывать условие атрибута RHS, сопоставляя его со значением атрибута записи, чтобы учесть нарушения вида STV.

Также поиск константы привязки может быть выполнен на этапе постобработки результатов. Для этого сначала генерируется таблица с единственной переменной привязки в RHS. Затем для каждого шаблона мы пытаемся специализировать константные привязки для RHS в соответствии со стратегией расширения, сохраняя при этом порог уверенности.

Поиск минимальных частичных ФЗ. CFDFinder может быть адаптирован для поиска *частичных ФЗ*. Они эквивалентны УФЗ, которые охватывают весь экземпляр отношения, то есть имеют поддержку равную единице, но могут иметь уверенность менее единицы.

Данное расширение реализуется как стратегия отсечения, представленная в алгоритме 4. В этой стратегии $conf(p)$ определяет уверенность шаблона p, однако возможно использование других метрик. Один из вариантов, который предлагает автор — метрика g_1, которая делит количество нарушенных пар кортежей на количество возможных пар кортежей. Параметр $\hat{c}$ представляет собой порог уверенности. В строках 1–4 алгоритму запрещается рассматривать любой другой шаблон, кроме нулевого, который на самом деле представляет собой частичную ФЗ. Если уверенность нулевого шаблона достаточна, мы добавляем его в таблицу и тем самым принимаем эту частичную ФЗ, в строке 5. Никакой другой

шаблон не может быть сгенерирован из-за функции *considerPattern*. Поэтому, если нулевой шаблон не удовлетворяет порогу уверенности, набор шаблонов-кандидатов пуст и алгоритм генерации таблиц завершается безрезультатно, сигнализируя, что кандидат не производит корректную частичную ФЗ. Чтобы ввести минимальность частичных ФЗ, повторно используется отсечение по снижению поддержки, описанное в разделе 4.4, с порогом $\delta_{max} = 0$. Поскольку значение поддержки для всех результатов равно единице, все результаты, кроме тех, которые не имеют подмножеств в решетке, то есть минимальные частичные ФЗ, отсекаются.

Алгоритм 4: Адаптация CFDFinder для поиска частичных ФЗ [5]

Ввод: таблица шаблонов T_p / шаблон p

Function `ConsiderPattern`(p)**:**
    return false;

Function `Valid`(p)**:**
    return false;

Function `AddPattern`(p)**:**
    return $conf(p) \geq \hat{c}$;

Function `Continue`(T_p)**:**
    return $|T_p| < 1$;

5. Эксперимент

В данном разделе автором работы [5] проводится экспериментальная оценка алгоритма CFDFinder на основе нескольких критериев. Вначале анализируется производительность на различных наборах данных и оценивается масштабируемость алгоритма путем изменения количества строк и столбцов в тех же наборах. Дополнительно изучается влияние параметров минимального прироста s' и максимального падения поддержки δ_{max} на время выполнения, структуру и качество результатов, а также роль отсечения по падению поддержки при формировании результата. Все изображения и таблицы в этом разделе были взяты из работы [5].

Условия эксперимента. CFDFinder реализован на языке Java 1.7 на платформе Metanome[1]. Эксперименты были проведены на OpenJDK 64-Bit Server VM 1.7.0.25 и системе CentOS 6.4. На используемом устройстве было установлено 128 ГБ оперативной памяти и два ЦПУ Intel Xeon E5-2650 частотой 2 ГГц.

Стандартная конфигурация алгоритма. Если не указано иное, для исследования используется отсечение по приросту поддержки с минимальным приростом s', равным 5% размера набора данных, отсечение по уменьшению поддержки с δ_{max}, равным 10%, фиксированная уверенность $\hat{c}$ на уровне 100%, и максимальный размер таблицы шаблонов в 2000 шаблонов. Выбор таких параметров отсечения объясняется в 5.3.

Наборы данных. Оценка CFDFinder проходит на различных синтетических и реальных наборах данных. Большинство из них взято из UCI Machine Learning Repository[2]. Кроме того, используются наборы данных *ncvoter*[3] и *Wine Reviews*[4]. Все наборы данных были преобразованы в CSV-файлы для использования с Metanome.

[1]Metanome Framework. www.metanome.de

[2]Lichman. M. UCI machine learning repository. https://archive.ics.uci.edu/datasets

[3]North Carolina State Board of Elections & Ethics Enforcement. https://dl.ncsbe.gov/

[4]Wine Reviews Dataset by zackthoutt. https://www.kaggle.com/datasets/zynicide/wine-reviews

5.1. Производительность

Автор подчёркивает, что прямое сравнение производительности CFDFinder с другими алгоритмами является затруднительным из-за принципиальных различий в их выходных данных. Попытки модифицировать CFDFinder и аналогичные алгоритмы из работ [1, 4] таким образом, чтобы они выявляли идентичные УФЗ, приводят к искажению их работы, что делает корректную оценку производительности невозможной. Сравнение их эффективности при выявлении различных УФЗ также оказывается малоинформативным, поскольку все методы демонстрируют высокую чувствительность к настройкам параметров. Это приводит к тому, что результаты могут варьироваться от крайне низких до превосходных в зависимости от конфигурации. С учётом этих ограничений, в данном подразделе представлена оценка производительности исключительно алгоритма CFDFinder на различных наборах данных.

В таблице 3 приведены характеристики наборов данных, а также количество найденных УФЗ и время выполнения алгоритма для каждого набора.

Таблица 3: Наборы данных и результаты

Набор	**Столбцы [#]**	**Строки [#]**	**Размер [КБ]**	**УФЗ**	**Время работы [сек]**
iris	5	150	5	10	0.687
balance-scale	5	625	7	4	0.417
liver-disorder	7	345	7	39	1.346
chess	7	28,056	519	4	88.599
abalone	9	4,177	187	184	279.122
nursery	9	12,960	1,024	9	151.182
Wine Reviews	11	150,930	17,540	$> 40^{\dagger}$	$> 43,200.000$
WBC	11	699	20	179	294.027
Bridges	13	108	6	443	56.288
Echocardiogram	13	132	6	491	30.841
1994 US Adult	14	48,842	3,528	$> 100^{\dagger}$	$> 43,200.000$
ncvoter	19	1,000	151	$> 20^{\dagger}$	$> 43,200.000$

Для наборов данных Wine Reviews, 1994 US Adult и ncvoter

CFDFinder не может получить результаты за разумное время при ипользовании заданных параметров, поэтому эти эксперименты были прерваны после 12 часов работы (отмечены как †). Долгое время выполнения алгоритма на большинстве наборов можно объяснить, основываясь на данных, представленных на рисунке 2.

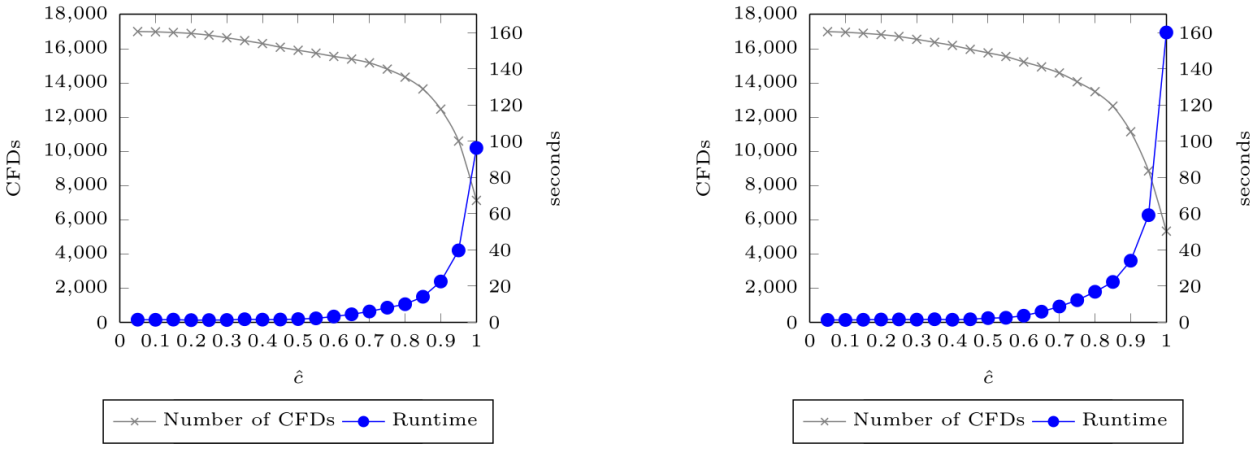

Рис. 2: Количество найденных УФЗ и время выполнения в зависимости от минимальной уверенности для оригинального подхода: Bridges $s' = 0.15$ и Bridges $s' = 0.85$ соответственно.

Этот рисунок иллюстрирует время выполнения и размер результирующего набора при использовании оригинального подхода отсечения для наборов данных WDC и Bridges в двух конфигурациях. Как показано в предыдущих работах, использование высокого порога уверенности значительно увеличивает время выполнения алгоритма, одновременно снижая количество найденных CFD до минимума [1, 4]. CFDFinder работает исключительно с порогом уверенности $\hat{c} = 1$, что не позволяет алгоритму завершаться раньше из-за очень общих шаблонов, уже удовлетворяющих встроенным ФЗ.

5.2. Масштабируемость

На время работы CFDFinder влияют два свойства входных данных: количество строк и количество столбцов. С ростом объёма, как правило, эти показатели увеличиваются. Поэтому автор исследуют время работы CFDFinder при увеличении каждого из них на наборах данных Nursery,

Wine Reviews и ncvoter.

Масштабируемость по строкам. На рисунке 3 синим цветом показано время работы алгоритма, а серый график представляет количество найденных УФЗ. CFDFinder обрабатывает набор данных Nursery примерно за 2.5 минуты. При этом можно заметить, что в этом наборе время выполнения алгоритма линейно зависит от количества записей. Максимальное число результатов 135 УФЗ достигается при объеме данных в 1500 строк и остается неизменным при дальнейших измерениях. Проанализировав полученные результаты, автор делает вывод, что большинство УФЗ обладают схожей структурой. Они формируются на основе условий для *атрибута решения* (decision attribute) — атрибута, имеющему всего три отдельных класса и, следовательно, характеризуются высокой поддержкой. После того как такое условие попадает в таблицу шаблонов, последующие кандидаты с подмножествами в LHS либо не должны иметь меньшую поддержку (меньше, чем на δ_{max}), либо должны исключаться.

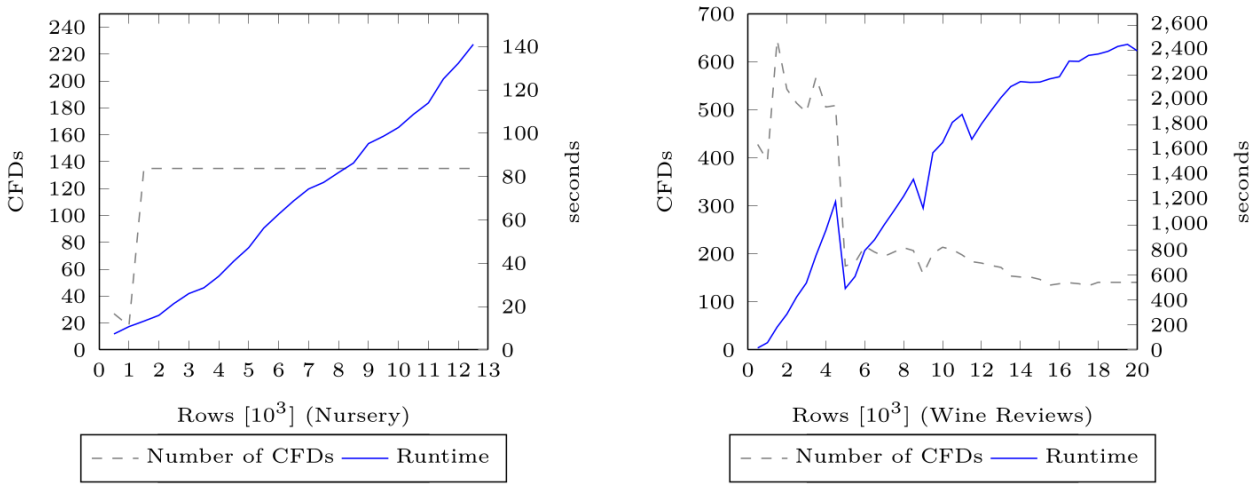

Рис. 3: Масштабируемость CFDFinder в зависимости от количества строк.

При работе с набором данных Wine Reviews наблюдается аналогичная зависимость времени выполнения алгоритма от объема записей. Из-за большего числа атрибутов CFDFinder требуется около 40 минут для обработки первых 20,000 строк. Также можно заметить, что число результатов влияет на время выполнения: при объеме данных в 5000 строк

наблюдается значительное уменьшение числа результатов. Это происходит из-за того, что на ранних этапах могут быть построены более качественные таблицы шаблонов, которые, в свою очередь, приводят к отсечению более поздних кандидатов с помощью отсечения по падению поддержки. Эти изменения отражаются и на графике времени выполнения, где видны три ключевые точки.

Таким образом, хотя объем записей напрямую влияет на время работы алгоритма, основной фактор определяется структурой данных. При увеличении числа строк вероятность уникальности значений атрибутов снижается, что приводит к увеличению числа нарушений типа MTV, что усложняет алгоритму построение таблиц шаблонов с высокой степенью уверенности. Для небольших объемов данных в результате часто встречаются случайные зависимости, однако при увеличении количества строк УФЗ становятся более точными с точки зрения реальных корреляций и размер результирующего набор со временем стабилизируется.

Масштабируемость по столбцам. В этом эксперименте проводится оценка производительности CFDFinder на наборах данных Nursery (первые 5000 записей) и ncvoter при изменении числа столбцов. На Рисунке 4 представлены результаты: синяя линия показывает время выполнения алгоритма для обоих наборов данных, а серая линия отражает количество найденных УФЗ. Серия измерений завершается, если время выполнения алгоритма превышает шесть часов.

Алгоритм демонстрирует высокую скорость работы с наборами данных небольшого, и среднего размера: при числе столбцов менее 10 обработка занимает менее трех минут. Однако при увеличении числа столбцов до диапазона 11–16 наблюдается резкий рост времени выполнения, хотя результат все еще можно получить за менее чем шесть часов. Если число столбцов превышает 16, результаты получить не удается из-за ограничений настроек и предела времени выполнения.

Как и предполагал автор, время работы CFDFinder увеличивается экспоненциально с ростом числа столбцов. Это связано с тем, что увеличение числа столбцов расширяет размер решётки кандидатов УФЗ и

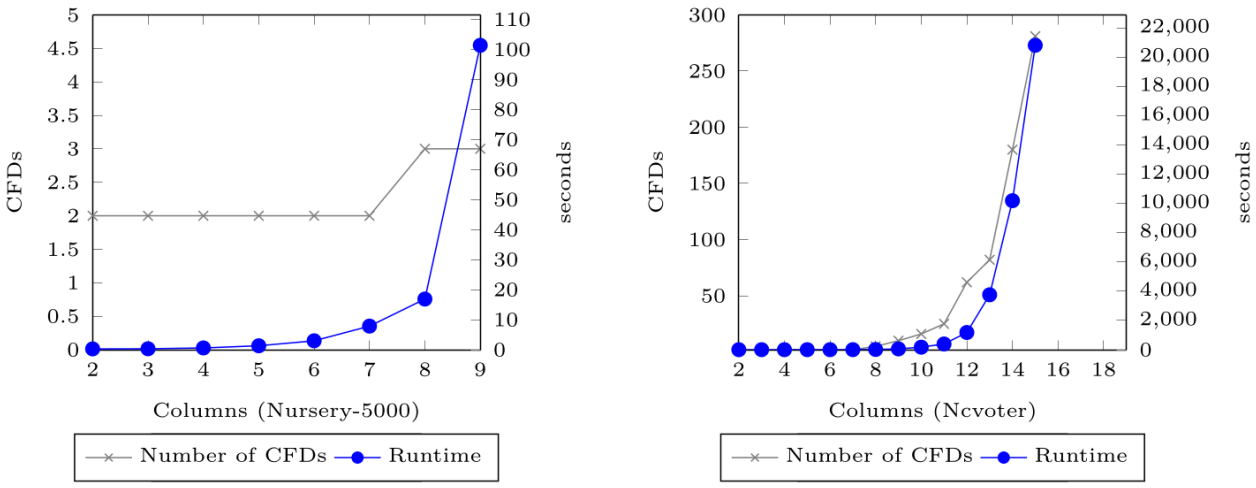

Рис. 4: Масштабируемость CFDFinder в зависимости от количества столбцов.

количество возможных шаблонов-кандидатов. Интересным наблюдением является то, что в наборе данных ncvoter наблюдается рост количества найденных УФЗ из-за указанных причин, в то время как в наборе Nursery количество УФЗ остается практически неизменным при всех измерениях.

5.3. Параметры отсечения

В этом подразделе автор изучает влияние параметров s' и δ_{max} на время выполнения, размер результирующего набора и качество результатов. Кроме того, рассматривается влияние отсечения по падению поддержки на структуру результирующего набора и даются рекомендации для настройки по умолчанию.

Размер результирующего набора и время выполнения. На рисунке 5 показан размер результирующего набора при различных конфигурациях s' и δ_{max} для набора данных Wisconsin Breast Cancer (WBC). Цвет каждой ячейки определяется по линейной шкале между минимальным значением (красный) и максимальным (зеленый) и свидетельствует только о количестве результатов для соответствующей пары значений параметров, но не о качестве результатов.

Видно, что количество результатов монотонно растет с увеличением δ_{max}, что связано с ослаблением ограничений на качество результатов с

δ_{max} \ s'	0	0,05	0,1	0,15	0,2	0,25	0,3	0,35	0,4	0,45	0,5	0,55	0,6	0,65	0,7
0	1207	760	874	838	1291	1905	1917	127	127	127	127	124	34	20	0
0,05	3980	907	1008	909	1312	1925	1937	147	147	147	147	140	34	20	0
0,1	4668	1603	1042	938	1587	1960	1972	182	182	181	166	152	34	20	0
0,15	5220	2757	1382	1165	1673	2025	2037	247	247	228	186	152	34	20	0
0,2	5655	3681	2544	2194	1695	2042	2054	264	263	239	186	152	34	20	0
0,25	5850	4206	3283	2517	2225	2044	2082	266	265	239	186	152	34	20	0
0,3	5958	4539	3524	2725	2243	2056	2082	266	265	239	186	152	34	20	0
0,35	6041	4924	3655	2994	2574	2056	2082	266	265	239	186	152	34	20	0
0,4	6127	5348	3715	3083	2612	2056	2082	266	265	239	186	152	34	20	0
0,45	6279	5515	4037	3490	2769	2056	2082	266	265	239	186	152	34	20	0
0,5	6575	5711	4402	3653	2778	2056	2082	266	265	239	186	152	34	20	0
0,55	7019	5884	4441	3687	2798	2082	2082	266	265	239	186	152	34	20	0
0,6	7475	6074	4453	3702	2798	2082	2082	266	265	239	186	152	34	20	0
0,65	7931	6204	4473	3704	2798	2082	2082	266	265	239	186	152	34	20	0
0,7	8164	6290	4517	3704	2798	2082	2082	266	265	239	186	152	34	20	0
0,75	8432	6397	4517	3704	2798	2082	2082	266	265	239	186	152	34	20	0
0,8	8731	6410	4517	3704	2798	2082	2082	266	265	239	186	152	34	20	0
0,85	8914	6428	4517	3704	2798	2082	2082	266	265	239	186	152	34	20	0
0,9	9006	6433	4517	3704	2798	2082	2082	266	265	239	186	152	34	20	0
0,95	9338	6433	4517	3704	2798	2082	2082	266	265	239	186	152	34	20	0
1	9453	6433	4517	3704	2798	2082	2082	266	265	239	186	152	34	20	0

Рис. 5: Размер результирующего набора при различных s' и δ_{max} для набора данных WBC.

каждым уровнем решетки. Таким образом, при фиксированном s' количество результатов может только увеличиваться. Наибольший размер результирующего набора достигается при $s' = 0$, $\delta_{max} = 1$, поскольку эта конфигурация максимизирует все таблицы шаблонов независимо от прироста поддержки и принимает любое падение поддержки на последующих уровнях. По сути, алгоритм в такой конфигурации находит почти оптимальную таблицу шаблонов для каждой возможной УФЗ в наборе данных, независимо от качества, и завершает работу раньше времени только в том случае, если для кандидата не может быть построена таблица шаблонов, то есть если ни один из возможных шаблонов не имеет уверенность равной единице.

Интересно, что s' не оказывает одинакового влияния на размер результирующего набора во всех конфигурациях. Можно было бы предположить, что более низкий минимальный порог прироста поддержки

позволит алгоритму найти больше подходящих УФЗ, однако автор замечает, что для низких значений δ_{max} максимальный размер результирующего набора достигается при $s' = 0.3$ на этом наборе данных. При более высоких значениях δ_{max} этот эффект уменьшается, что намекает на взаимодействие этих двух параметров на уровне данных. При $s' = 0.3$ алгоритм находит одни и те же УФЗ на нескольких уровнях (и, следовательно, без падения глобальной поддержки) и сообщает обо всех из них, в то время как при меньшем s' алгоритм находит УФЗ с более высокой глобальной поддержкой на ранних этапах, что приводит к отсечению следующих уровней, если глобальная поддержка не может быть сохранена на том же уровне. Также, те же эффекты автор наблюдает в случае времени выполнения, то есть время выполнения обычно увеличивается с ростом δ_{max} и уменьшается с ростом s' , за исключением низких пороговых значений падения поддержки.

Влияние на качество результатов. Параметры s' и δ_{max} поразному влияют на качество результата. В то время как s' уменьшает шум и размер таблицы шаблонов, δ_{max} стремиться уменьшить размер LHS встроенной ФЗ, смягчая ограничение на качество результата. На рисунке 6 представлен график зависимости среднего размера LHS (обозначен как $|X|$) от значения параметра δ_{max} для набора данных WDC до и после удаления *неинтересных* УФЗ, то есть тех, которые имеют допустимые обобщения в результирующем наборе.

Как и ожидал автор, средний размер LHS уменьшается с увеличением δ_{max}, пока не стабилизируется при $\delta_{max} > 0.6$, поскольку такая конфигурация максимизирует размер результирующего набора, как показано на рисунке 5. Удаление неинтересных УФЗ уменьшает средний размер LHS результирующего набора. Этот эффект обусловлен расположением падений поддержки в решетке результатов и, таким образом, зависит от набора данных. Например, для набора данных Bridges удаление неинтересных УФЗ уменьшает средний размер LHS примерно на 1.23 атрибута. Стоит отметить, что при удалении неинтересных специализаций влияние δ_{max} ослабевает до значения 0.2. В диапазоне от 0.2 до 0.6 автор замечает небольшие колебания среднего размера LHS. Это

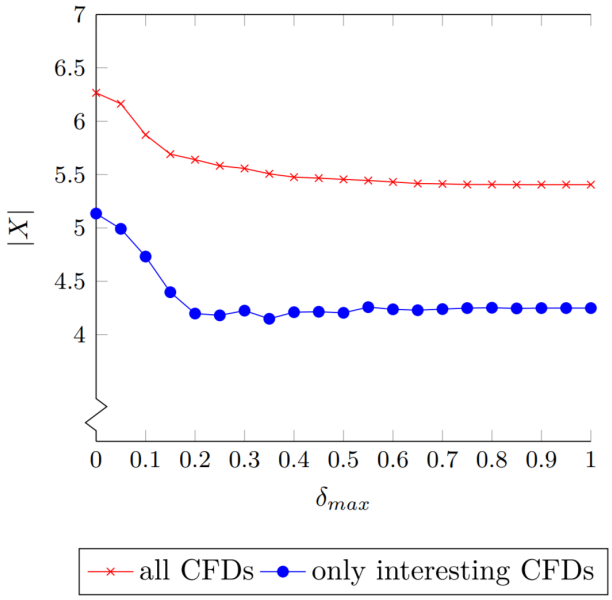

Рис. 6: Средний размер LHS при изменении δ_{max} с удалением и без удаления неинтересных УФЗ для набора WBC, $s' = 0.05$.

связано с тем, что с ростом δ_{max} в решетке становятся доступны больше путей, что увеличивает долю результатов с большими LHS. Однако, при больших значениях δ_{max} эти эффекты исчезают, так как все больше путей становятся доступны, а количество падений поддержки стремится к нулю.

Структура результирующего набора. Чтобы наглядно продемонстрировать влияние отсечения по падению поддержки на результирующий набор, рассмотрим рисунок 7, который показывает обнаруженные УФЗ для набора данных WDC до и после удаления допустимых обобщений из результирующего набора.

Автор показывает, как алгоритм эффективно сокращает количество результатов и средний размер LHS, сохраняя при этом ключевые характеристики неизменными. В частности, минимальные и максимальные значения глобальной поддержки остаются неизменными. Конфигурацию с $\delta_{max} = 0.1$ можно рассматривать как компромисс, обменивающий максимум 10 % поддержки на каждый удаленный атрибут LHS. Этот компромисс автор также заметил в результатах с низкой поддержкой, полученных при $\delta_{max} = 0.1$ в том же эксперименте, о которых не сообщается в результатах в конфигурации $\delta_{max} = 0$. Однако, как видно

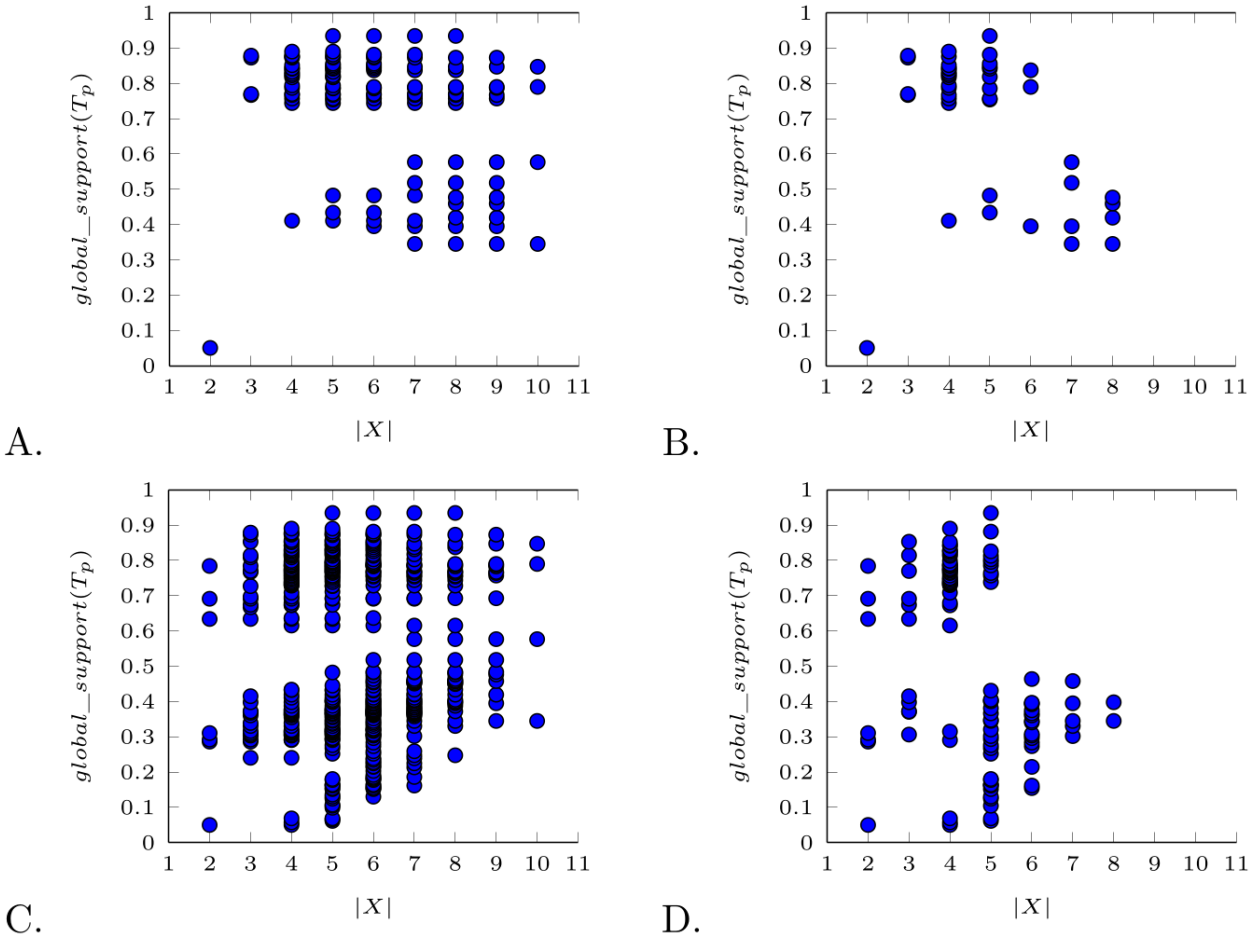

Рис. 7: Найденные УФЗ с соответствующей глобальной поддержкой таблицы шаблонов T_p и размером LHS $|X|$ для набора данных WBC при $s' = 0.05$ и A) $\delta_{max} = 0$, B) $\delta_{max} = 0$ (неинтересные результаты удалены), C) $\delta_{max} = 0.1$, D) $\delta_{max} = 0.1$ (неинтересные результаты удалены).

из диаграмм B и D, увеличение порога падения поддержки приводит к обнаружению трех результатов с размером LHS равным двум и глобальной поддержкой от 0.6 до 0.8, которые конфигурация $\delta_{max} = 0$ не может найти. Таким образом, δ_{max} расширяет результирующий набор с точки зрения поддержки и эффективно снижает средний размер LHS найденных УФЗ.

Влияние ограничения на максимальный размер таблицы шаблонов. В последнем эксперименте автор варьирует верхнюю границу максимального размера таблицы шаблонов $|T_p|$. На рисунке 8 показаны количество найденных УФЗ, средняя глобальная поддержка и время работы алгоритма при $s' = 0$ и $\delta_{max} = 0.1$ на наборе данных Bridges.

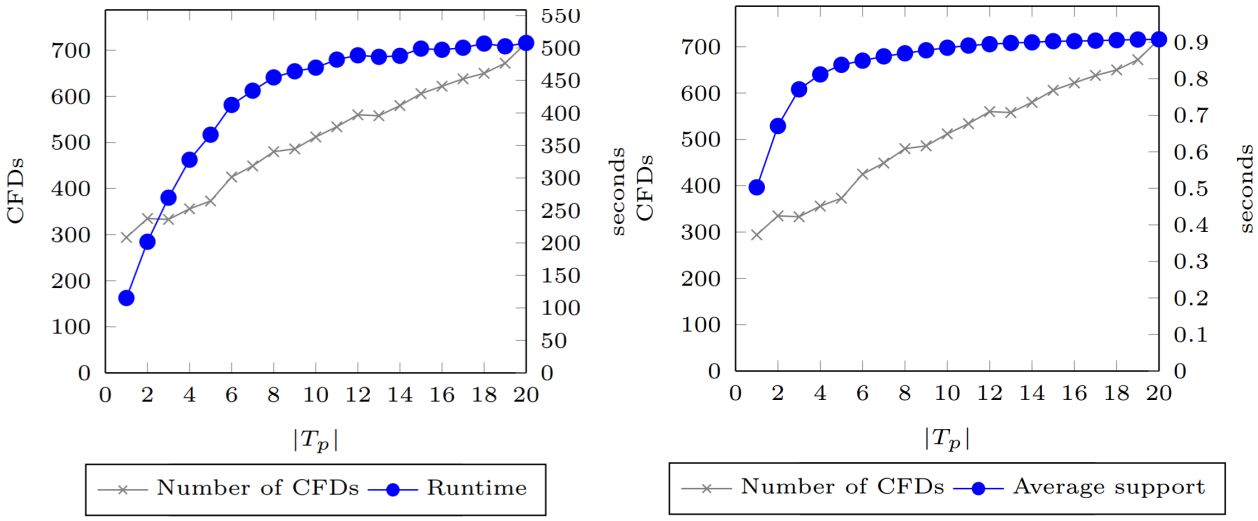

Рис. 8: Размер результирующего набора, время выполнения и средняя поддержка при изменении максимального количества шаблонов для набора данных Bridges при $s' = 0$ и $\delta_{max} = 0.1$.

Интуитивно понятно, что ограничение на максимальный размер таблицы шаблонов влияет на время работы и размер результирующего набора из-за большей гибкости. В эксперименте автор убеждается в этом, однако он замечает уменьшение влияния на время работы алгоритма с увеличением верхней границы. Автор связывает это с уменьшением влияния каждого дополнительного условия на таблицу шаблонов. При высокой верхней границе все шаблоны с высокой поддержкой уже добавлены в каждую таблицу шаблонов, средняя поддержка высока, и приходится рассматривать много путей в решетке. В этом случае дополнительные шаблоны не имеют большого значения. При очень высокой границе размер результирующего набора и время работы стабилизируются, когда все таблицы шаблонов насыщены и больше не существует подходящих для добавления условий.

Рекомендованные настройки. По итогам проведенных экспериментов, автор рекомендует устанавливать параметры s' и δ_{max} в диапазоне от 0 до 0.3. Ограничение на максимальный размер таблицы шаблонов влияет на производительность алгоритма на больших наборах данных и должно быть установлено исходя из требований конкретной задачи или быть больше, чем $\frac{1}{s'}$, чтобы можно было в алгоритме сохранять таблицы шаблонов любого размера.

6. Заключение

По итогам работы был проведен обзор магистерской диссертации “Discovering Interesting Conditional Functional Dependencies” за авторством Maximilian Grundke:

1. Выполнен обзор предметной области;
2. Выполнен обзор алгоритма CFDFinder и принципа его работы;
3. Рассмотрены стратегии сокращения пространства поиска и расширения шаблона, представленные автором;
4. Рассмотрена оценка производительности и масштабируемости алгоритма.

В дальнейшем планируется реализовать рассмотренный алгоритм на C++ в рамках платформы *Desbordante*.

Список литературы

[1] Chiang Fei, Miller Renée J. Discovering data quality rules // Proc. VLDB Endow. — 2008. — Aug. — Vol. 1, no. 1. — P. 1166–1177.

[2] 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.

[3] 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.

[4] Discovering Conditional Functional Dependencies / Wenfei Fan, Floris Geerts, Laks V. S. Lakshmanan, Ming Xiong // 2009 IEEE 25th International Conference on Data Engineering. — 2009. — P. 1231–1234.

[5] Grundke Maximilian. Discovering Interesting Conditional Functional Dependencies. — 2018. — Information Systems Chair Hasso-Plattner-Institute, Potsdam.

[6] On generating near-optimal tableaux for conditional functional dependencies / Lukasz Golab, Howard Karloff, Flip Korn et al. // Proc. VLDB Endow. — 2008. — Aug. — Vol. 1, no. 1. — P. 376–390.

[7] Papenbrock Thorsten, Naumann Felix. A Hybrid Approach to Functional Dependency Discovery // Proceedings of the 2016 International Conference on Management of Data. — SIGMOD '16. — New York, NY, USA : Association for Computing Machinery, 2016. — P. 821–833.