Санкт-Петербургский государственный университет

Кафедра информационно-аналитических систем

Группа 22.Б08-мм

Реализация алгоритма поиска дифференциальных зависимостей в рамках платформы Desbordante

Синельников Михаил Алексеевич

Отчёт по учебной практике
в форме «Решение»

Научный руководитель:
ассистент кафедры ИАС Г. А. Чернышев

Санкт-Петербург
2024

Оглавление

Введение **3**

1. Постановка задачи **5**

2. Обзор **6**

2.1. Основные определения 6

2.2. Алгоритм SPLIT 9

3. Метод **13**

3.1. Используемые структуры 13

3.2. Особенности реализации 14

4. Эксперимент **16**

Заключение **18**

Список литературы **19**

Введение

Профилирование данных — процесс извлечения метаданных из некоторого набора данных. Метаданные — это информация о данных. Например, метаданными для файла являются его название, его размер и дата его создания. Кроме того, метаданными могут являться различные нетривиальные закономерности в данных. Наукоёмкое профилирование занимается поиском этих неочевидных зависимостей.

В качестве примера закономерностей в табличных данных можно привести функциональные зависимости (FD). Пусть X, Y — два множества атрибутов в таблице. Функциональная зависимость $X \to Y$ существует, если для любой пары строк в таблице из равенства их значений на атрибутах X следует равенство их значений на атрибутах Y.

Desbordante[1] — высокопроизводительный профилировщик данных с открытым исходным кодом, позволяющий пользователю искать наукоёмкие паттерны в своих данных. На данный момент Desbordante поддерживает поиск и валидацию функциональных зависимостей, а также некоторых других видов зависимостей.

Другим примером нетривиальных закономерностей являются дифференциальные зависимости (DD). Данный вид закономерностей является обобщением функциональных зависимостей на случай, когда значения не в точности равны, а отличаются в определённых пределах. Условия для существования дифференциальных зависимостей более мягкие, чем для функциональных, что позволяет найти большее число интересных закономерностей в данных.

Существующий алгоритм поиска дифференциальных зависимостей, описанный в статье [2], не имеет открытой реализации, что делает невозможным обычному пользователю исследовать данные с их помощью. Таким образом, существует необходимость в создании открытой реализации алгоритма для поиска дифференциальных зависимостей. Для улучшения взаимодействия с пользователем было решено интегрировать данный алгоритм в платформу Desbordante. Кроме вышеуказанно-

[1] https://github.com/Mstrutov/Desbordante

го преимущества, интеграция позволит расширить функциональность профилировщика и повысить его полезность для пользователя.

1. Постановка задачи

Целью работы является реализация алгоритма SPLIT поиска дифференциальных зависимостей. Для достижения цели были поставлены следующие задачи:

- провести обзор предметной области и алгоритма SPLIT;
- реализовать алгоритм на языке программирования C++ в рамках платформы Desbordante;
- произвести сравнение производительности разных версий алгоритма.

2. Обзор

2.1. Основные определения

Все определения в данном подразделе взяты из работы [1], так как они более просты для понимания и удобны в использовании, чем определения, данные в работе [2].

Определение 2.1. Обозначим буквой R множество всех атрибутов в таблице, домен атрибута $A_i \in R$ обозначим как $dom(A_i)$.

Определение 2.2. Метрика d_A — функция, определённая на атрибуте $A \in R$, которая любой паре значений a_1, a_2 из атрибута сопоставляет расстояние между ними. При этом d_A обязана удовлетворять следующим условиям:

1. $d_A(a_1, a_2) \geq 0$;
2. $d_A(a_1, a_2) = 0 \Leftrightarrow a_1 = a_2$;
3. $d_A(a_1, a_2) = d_A(a_2, a_1)$,

где $a_1, a_2 \in dom(A)$.

Определение 2.3. Дифференциальная функция $A[\omega]$ ($\omega = [x, y], 0 \leq x \leq y$) — функция, определённая на атрибуте A с метрикой d_A и возвращающая булево значение, показывающее, выполняется ли утверждение $\forall a_1, a_2 \in dom(A)\ x \leq d_A(a_1, a_2) \leq y$.

Определение 2.4. Дифференциальной функцией $X[W_X]$ на множестве атрибутов $X = \{A_1, \ldots, A_m\}$ называется логическое умножение дифференциальных функций для каждого атрибута:

$$X[W_X] = A_1[\omega_1] \wedge A_2[\omega_2] \wedge \ldots \wedge A_m[\omega_m].$$

Если для пары строк таблицы значение дифференциальной функции $X[W_X]$ истинно, то говорят, что данная пара строк удовлетворяет $X[W_X]$.

Определение 2.5. Дифференциальная функция $X[W_X]$ включает в себя дифференциальную функцию $Y[W_Y]$ ($X[W_X] \succeq Y[W_Y]$), если:

$$\forall A_i[\omega_i] \in X[W_X] \; \exists \; A_i[\omega_i'] \in Y[W_Y] : \omega_i' \subseteq \omega_i.$$

Заметим, что для того, чтобы $X[W_X] \succeq Y[W_Y]$, необходимо, чтобы $X \subseteq Y$. В самом деле, если $\exists A_i : A_i \in X, A_i \notin Y$, то для $A_i[\omega_i] \in X[W_X] \nexists A_i[\omega_i'] \in Y[W_Y]$. Также заметим, что если $X[W_X] \succeq Y[W_Y]$, то любая пара строк таблицы, удовлетворяющая $Y[W_Y]$, также удовлетворяет и $X[W_X]$.

Определение 2.6. Дифференциальная зависимость (DD) — утверждение $X[W_L] \to Y[W_R]$ между двумя дифференциальными функциями. Дифференциальная зависимость удерживается в таблице, если для любой пары строк таблицы из того, что она удовлетворяет $X[W_L]$, следует то, что она удовлетворяет $Y[W_R]$.

Определение DD похоже на определение включения в себя. Однако, если для того, чтобы $X[W_L] \succeq Y[W_R]$, необходимо, чтобы $L \subseteq R$, для того, чтобы DD удерживалась, это не требуется. Более того, наиболее интересными для рассмотрения являются зависимости с непересекающимися областями определения левой и правой дифференциальных функций ($L \cap R = \emptyset$).

Определение 2.7. Множество DD Σ_2 является логическим следствием (импликацией) множества DD Σ_1, если для любой таблицы из удерживания всех DD из Σ_1 следует удерживание всех DD из Σ_2.

Определение 2.8. Пусть Σ — множество дифференциальных зависимостей. DD $X[W_L] \to Y[W_R] \in \Sigma$ называется минимальной, если выполняются следующие условия:

1. $\nexists X'[W_X] \to Y[W_R] \in \Sigma : X'[W_X] \succeq X[W_L]$;
2. $\nexists X[W_L] \to Y'[W_Y] \in \Sigma : Y[W_R] \succeq Y'[W_Y]$.

Определение 2.9. Множество DD Σ' называется покрытием для множества DD Σ, если любая DD из Σ лежит в Σ' или является логическим следствием дифференциальных зависимостей из Σ'.

Определение 2.10. Покрытие Σ_c множества Σ называется минимальным, если не существует покрытия $\Sigma' \subseteq \Sigma_c$.

Таблица 1: Пример таблицы (взят из работы [1])

TID	**YoS**	**Gen**	**Edu**	**Sal**
1	25	2	4	15
2	25	1	3	18
3	28	1	3	15
4	32	2	2	12
5	30	1	1	15

Пример 2.1. В качестве примера рассмотрим таблицу 1. Возьмём в качестве метрики для каждой колонки $A \in R$ $d_A(x, y) = |x - y|$. Дифференциальная зависимость $YoS[0, 0] \to Edu[1, 1]$ удерживается, так как для любой пары строк, удовлетворяющих $YoS[0, 0]$ (то есть для пары строк $\{1, 2\}$), эта пара строк удовлетворяет и $Edu[1, 1]$. DD $YoS[2, 2] \to Sal[0, 0]$ не удерживается, так как пара строк $\{4, 5\}$ удовлетворяет $YoS[2, 2]$, но не удовлетворяет $Sal[0, 0]$.

Теперь возьмём в качестве множества дифференциальных зависимостей

$$\begin{aligned}\Sigma = \{&YoS[0,5]Gen[0,0] \to Edu[0,0], YoS[0,5] \to Edu[0,2],\\ &YoS[0,7] \to Edu[0,0], Gen[0,0]Sal[0,0] \to YoS[3,5],\\ &Gen[0,0]Sal[0,0] \to YoS[3,7]\}.\end{aligned} \tag{1}$$

Множество DD

$$\Sigma_c = \{YoS[0,7] \to Edu[0,0], Gen[0,0]Sal[0,0] \to YoS[3,5]\}$$

является минимальным покрытием Σ. Действительно, любая DD из Σ является логическим следствием DD из Σ_c. Например, если удерживается DD $YoS[0, 7] \to Edu[0, 0]$, то будет выполняться следующая цепочка

следствий: пара строк удовлетворяет $YoS[0,5]Gen[0,0] \Rightarrow$ она удовлетворяет $YoS[0,7] \Rightarrow$ она удовлетворяет $Edu[0,0]$, значит, удерживается DD $YoS[0,5]Gen[0,0] \rightarrow Edu[0,0]$. Аналогично доказывается удерживание остальных DD из Σ. Также очевидно, что не существует покрытия меньшего по включению.

Пример 2.2. Пусть $DD_1 = d[1,6] \rightarrow a[0,150]$, $DD_2 = d[6,7] \rightarrow a[0,100]$, $DD_3 = d[1,7] \rightarrow a[0,150]$, $\Sigma = \{DD_1, DD_2, DD_3\}$. $\Sigma_c = \{DD_1, DD_2\}$ является минимальным покрытием Σ, так как DD_3 логически выводится из $\{DD_1, DD_2\}$, и, если убрать из Σ_c хотя бы одну из зависимостей, Σ_c перестанет быть покрытием.

Теперь заметим, что DD_1 не является минимальной DD, так как она не удовлетворяет условию 1 определения 2.8. Действительно, $DD_3 \in \Sigma$ и $d[1,7] \succeq d[1,6]$. Таким образом, если заменить DD_1 на DD_3 в Σ_c, множество $\Sigma'_c = \{DD_2, DD_3\}$ будет минимальным покрытием Σ, состоящим только из минимальных DD.

Одной из наиболее значимых задач поиска зависимостей в данных является поиск минимального покрытия всех зависимостей, удерживающихся в таблице. Минимальное покрытие удобно тем, что оно позволяет значительно уменьшить огромное по размеру множество удерживающихся зависимостей до достаточно небольшого множества, из которого можно логически вывести все остальные зависимости. При поиске дифференциальных зависимостей даже минимальное покрытие является очень большим по размеру, вследствие чего рассматривается задача поиска минимального покрытия, состоящего только из минимальных зависимостей.

2.2. Алгоритм SPLIT

Алгоритм SPLIT предназначен для поиска минимального покрытия всех DD, удерживающихся в таблице, состоящего только из минимальных DD. Алгоритм имеет три основных стадии:

1. Построение пространства поиска;

2. Отсечение зависимостей, не минимальных по левой части (условие 1 определения 2.8);

3. Отсечение зависимостей, не минимальных по правой части (условие 2 определения 2.8), и зависимостей, логически выводящихся из остальных.

Первая составляющая алгоритма — построение пространства поиска дифференциальных функций. В работе [2] указаны некоторые возможные способы это сделать. Например,

$$\forall A_i \in R\ \Phi(A_i) = \{A_i[u, v] \mid 0 \leq u \leq v \leq D\},$$

где D — максимальное расстояние между значениями атрибута A_i. Соответственно, для нескольких атрибутов $X = \{A_1, \ldots, A_m\}$ пространством поиска является $\Phi(X) = \Phi(A_1) \times \ldots \Phi(A_m), A_i \in X$.

Вторая составляющая алгоритма — отсечение зависимостей, не минимальных по левой части. Более формально задача звучит так: из данного пространства поиска дифференциальных функций $\Phi(X)$, являющихся левыми частями DD с фиксированной правой частью $\phi_R[Y]$, выделить такое подмножество $\Phi'(X)$, что все DD $\phi_R[X] \rightarrow \phi_R[Y]$, где $\phi_R[X] \in \Phi'(X)$, удерживаются в таблице и $\nexists \phi'_R[X] \rightarrow \phi_R[Y] : \phi'_R[X] \succeq \phi_R[X]$ и $\phi'_R[X] \rightarrow \phi_R[Y]$ удерживается в таблице.

Наивная реализация данной части алгоритма — валидация каждой DD из пространства поиска и дальнейшее отсечение из получившегося множества удерживающихся зависимостей тех, которые не являются минимальными по левой части. Для валидации DD требуется $O(n^2)$ времени, где n — число строк в таблице, так как необходимо рассмотреть все пары строк и вычислить расстояние между значениями в этих строках. Если размер таблицы большой, то валидация DD становится самой затратной по времени частью алгоритма. Дальнейшие варианты алгоритма SPLIT представляют собой различные способы уменьшить число валидаций.

Основные варианты алгоритма:

1. Негативное отсечение;

2. Позитивное отсечение;

3. Гибридное отсечение;

4. Отсечение пар строк.

Негативное отсечение основывается на простом утверждении: если DD $\phi_1[X] \rightarrow \phi_R[Y]$ не удерживается в таблице, то и DD $\phi_2[X'] \rightarrow \phi_R[Y]$, где $\phi_2[X'] \succeq \phi_1[X]$, также не удерживается. Таким образом, если $\phi_1[X] \rightarrow \phi_R[Y]$ не удерживается в таблице, можно сразу отсечь некоторое множество зависимостей, которые не нужно валидировать.

ALGORITHM 1: Bottom-Up REDUCE(I, $\Phi(X)$, $\phi_R[Y]$)

Input: An instance I, a search space $\Phi(X)$ of $\phi_L[X]$ and a target $\phi_R[Y]$
Output: A minimal set Σ for all DDs determining $\phi_R[Y]$ under I

$\Sigma := \emptyset$
$\phi_p[Z] :=$ the last element removed from $\Phi(X)$
$\Phi_3(X) := \{\phi_v[V] \in \Phi(X) \mid V \subseteq Z, \phi_v[V] \succeq \phi_p[V]\}$
$\Phi_4(X) := \Phi(X) \setminus \Phi_3(X)$
if $I \not\models \phi_p[Z] \rightarrow \phi_R[Y]$ **then**
    $\Sigma := \Sigma \uplus$ REDUCE(I, $\Phi_4(X)$, $\phi_R[Y]$)
else
    $\Sigma := \Sigma \uplus \{\phi_p[Z] \rightarrow \phi_R[Y]\}$
    $\Sigma := \Sigma \uplus$ REDUCE(I, $\Phi_3(X)$, $\phi_R[Y]$)
    $\Sigma := \Sigma \uplus$ REDUCE(I, $\Phi_4(X)$, $\phi_R[Y]$)
end if
return Σ

Рис. 1: Негативное отсечение (взято из работы [2])

Позитивное отсечение основывается на аналогичном утверждении, позволяя отсечь некоторое множество зависимостей, если $\phi_1[X] \rightarrow \phi_R[Y]$ удерживается в таблице.

Гибридное отсечение комбинирует негативное и позитивное отсечение, позволяя алгоритму работать быстрее как в случае, когда большинство DD из пространства поиска удерживается, так и в обратном случае.

Отсечение пар строк основывается на том факте, что множество пар строк, удовлетворяющих $\phi_2[X]$ является подмножеством множества пар строк, удовлетворяющих $\phi_1[X']$, если $\phi_1[X] \succeq \phi_2[X']$. Таким образом можно отсекать некоторые пары строк, что позволяет сократить время валидации DD.

Третья составляющая алгоритма — отсечение зависимостей, не минимальных по правой части, и зависимостей, логически выводящихся из остальных. После предыдущего этапа работы алгоритма для каждого атрибута Y и для каждой дифференциальной функции $\phi_R[Y] \in \Phi(Y)$ получено множество DD, минимальных по левой части. Для получения минимального покрытия достаточно удалить из множества DD, логически выводящиеся из остальных, то есть удалить такие DD $\phi_1[W] \rightarrow \phi_2[V]$, что $\exists \phi_L[X] \rightarrow \phi_R[Y] : \phi_L[X] \succeq \phi_1[W], \phi_2[V] \succeq \phi_R[Y]$.

ALGORITHM 3: Minimal COVER(R, I)

Input: A relation schema R, and an instance I of R.
Output: A minimal cover Σ for all DDs under I.

1: $\Sigma := \emptyset$
2: **for** each attribute $Y \in R$ **do**
3: $X := R \setminus \{Y\}$
4: **for** each $\phi_R[Y] \in \Phi(Y)$ **do**
5: $\Sigma := \Sigma \cup$ REDUCE(I, $\Phi(X)$, $\phi_R[Y]$) {reduce step}
6: **end for**
7: **end for**
8: **repeat**
9: **if** exist $\phi_L[X] \rightarrow \phi_R[Y], \phi_w[W] \rightarrow \phi_v[V] \in \Sigma$ such that $X \subseteq W$, $\phi_L[X] \succeq \phi_w[X]$, $V \subseteq Y$ and $\phi_v[V] \succeq \phi_R[V]$ **then**
10: remove $\phi_w[W] \rightarrow \phi_v[V]$ from Σ
11: **end if**
12: **until** no changes on Σ
12: **return** Σ

Рис. 2: Поиск минимального покрытия (взято из работы [2])

3. Метод

Ввиду того, что открытой реализации алгоритма SPLIT не было найдено, для начала был реализован алгоритм, по возможности похожий на тот, что был описан в работе [2].

3.1. Используемые структуры

Для реализации алгоритма в платформе Desbordante были добавлены следующие классы и структуры:

- Класс `Split`;
- Тип `DF`;
- Структура `DD`;
- Структура `DFConstraint`;
- Структура `DDString`.

Класс `Split` — наследник класса `Algorithm` (базового класса для всех алгоритмов в платформе Desbordante). Простое переопределение нескольких методов класса позволяет удобно получать входные данные для алгоритма, что способствует улучшению взаимодействия с пользователем.

Тип `DF` — тип, предназначенный для хранения дифференциальной функции, определён как `std::vector<std::pair<double,double>>` — вектор, указывающий для каждой колонки таблицы нижнюю и верхнюю границы расстояний. Длина этого вектора всегда должна быть равна числу колонок в таблице, а значения расстояний больше или равны 0. В том случае, если для некоторого атрибута не определяется ограничение на расстояния, по умолчанию в соответствующий элемент вектора устанавливается пара из минимального и максимального значений расстояний на атрибуте таблицы.

Структура `DD` предназначена для хранения дифференциальной зависимости в виде пары дифференциальных функций (`DF`), обозначающих левую и правую часть зависимости.

Структура `DFConstraint` — упорядоченная тройка, состоящая из названия атрибута и нижней и верхней границ расстояний.

Структура `DDString` — это пара `std::list<DFConstraint>`. Структура предназначена для хранения DD в виде, удобном для тестирования алгоритма, и используется исключительно в тестах.

Кроме того, при реализации были использованы производные от данных типов: `std::list<DF>` для хранения пространства поиска дифференциальных функций и `std::list<DD>` для хранения минимального покрытия.

3.2. Особенности реализации

Стадия 1 алгоритма SPLIT предоставляет наибольшее пространство для возможных вариантов реализации. В работе [2] не было написано, каким в точности должно быть пространство поиска, однако в секции экспериментов было указано, что для их проведения были использованы пространства поиска, состоящие из пяти дифференциальных функций на каждый атрибут.

При реализации собственной версии алгоритма было решено оставить выбор пространства поиска конечному пользователю: в качестве параметра запуска алгоритма была добавлена специальная CSV-таблица, в которой пользователю предлагается для каждой колонки таблицы, в которой будет проводиться поиск DD, указать список дифференциальных функций, задаваемых нижней и верхней границами возможных расстояний. Пример такой таблицы указан ниже (таблица 2). Для построения пространства поиска был реализован метод `SearchSpace` класса `Split`.

Стадия 2 алгоритма SPLIT была реализована по описанию из работы [2]. В секции экспериментов (рисунок 3) работы сравнивались 6 различных вариантов данного этапа алгоритма. Для реализации в настоя-

Таблица 2: Пример таблицы со списками дифференциальных функций

Col1	**Col2**	**Col3**
[0;0]	[4;4]	[5;5]
[0;1]	[4;8]	[5;7]
[0;2]	[8;8]	[5;12]
[1;1]	—	[7;7]
[1;2]	—	[7;12]
[2;2]	—	[12;12]

щей работе из них были выбраны три: негативное отсечение [1] (`BU-Ne`), гибридное отсечение [3] (`Hybrid`) и отсечение пар строк [4] (`IE-Hybrid`).

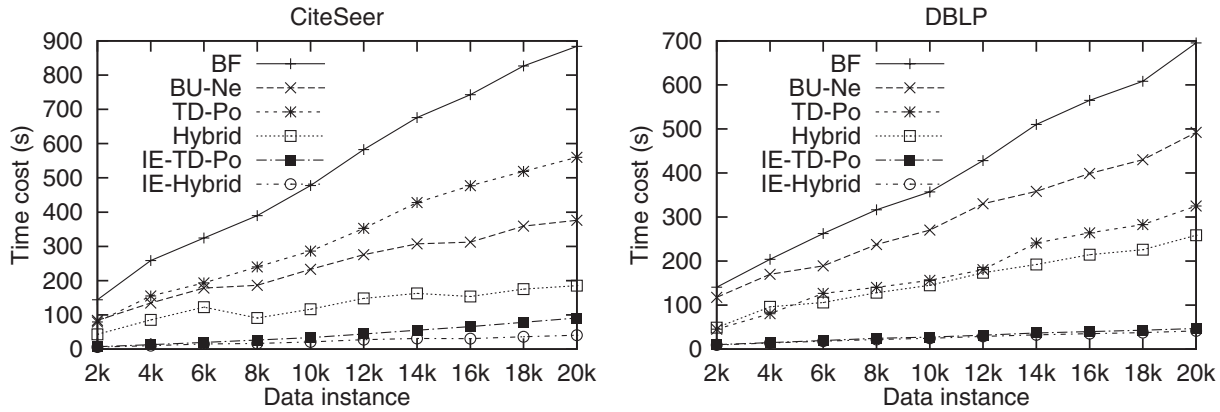

Рис. 3: Сравнение времени работы различных версий алгоритма (взято из работы [2])

Было замечено, что существенную часть времени работы данного этапа алгоритма занимает подсчёт расстояний между строковыми значениями. Для оптимизации данного подсчёта был реализован подсчёт всех расстояний в таблице до начала работы остальной части алгоритма (в методе `CalculateAllDistances` класса `Split`). Также этот подсчёт позволяет определить минимальное и максимальное расстояние между значениями для каждого атрибута, которые являются значениями по умолчанию для типа `DF`.

Стадия 3 алгоритма SPLIT была реализована в соответствии с описанием в работе [2] (рисунок 2).

4. Эксперимент

Для тестирования алгоритма на корректность была использована библиотека GoogleTest. Для тестов были придуманы несколько простых случаев, в которых минимальное покрытие возможно вычислить вручную.

Ввиду того, что написанную версию алгоритма невозможно сравнить по производительности с оригинальной версией из работы [2], было решено провести схожие замеры производительности для разных версий алгоритма (как на рисунке 3).

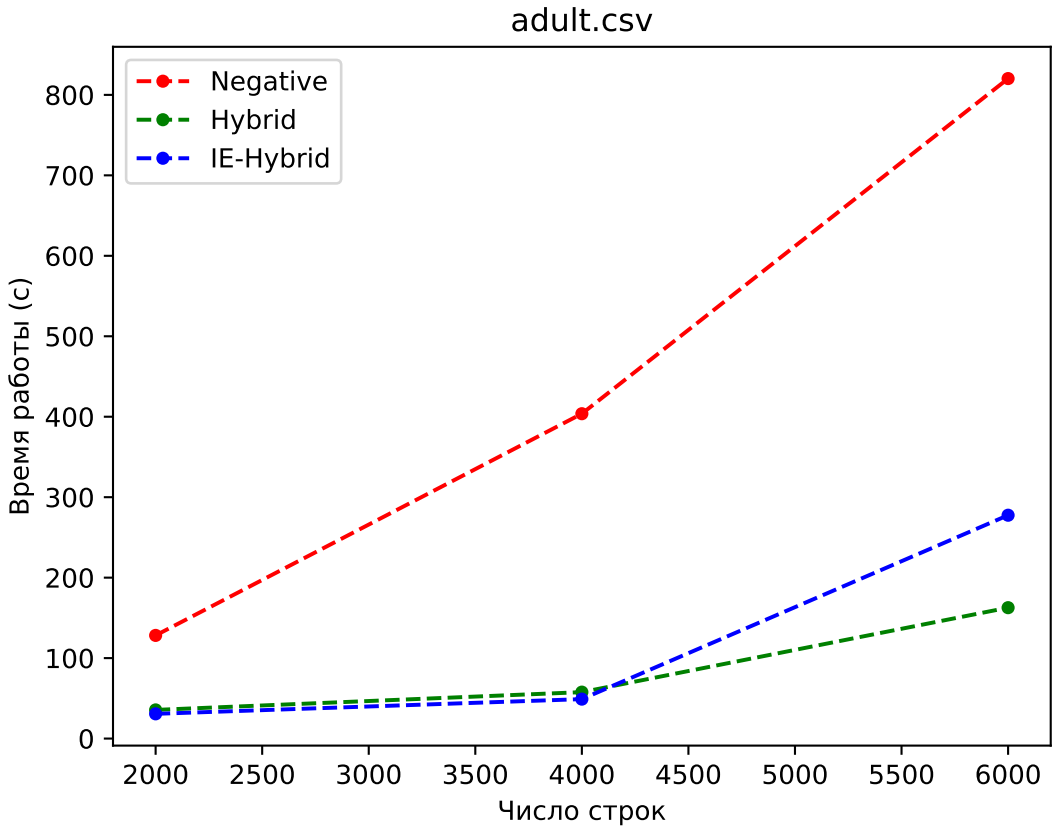

Рис. 4: Сравнение времени работы различных версий алгоритма

Замеры производительности проводились на датасете adult.csv. Для экспериментов были взяты первые 5 его атрибутов и первые 2000, 4000 и 6000 строк. Как и в оригинальной работе [2], для каждого атрибута были взяты по 5 дифференциальных функций для формирования пространства поиска. На рисунке 4 показаны результаты замеров

времени работы реализованных в платформе Desbordante версий алгоритма SPLIT. Как и в экспериментах работы [2] (рисунок 3), реализованные версии гибридного отсечения (`Hybrid`) и отсечения пар строк (`IE-Hybrid`) работают значительно быстрее негативного отсечения (`Negative`). Однако, с ростом числа строк версия `IE-Hybrid` начинает работать медленнее `Hybrid`, в то время как в экспериментах работы [2] (рисунок 3) `IE-Hybrid` остаётся самым быстрым вариантом алгоритма. Это может быть связано с нерациональным использованием памяти в реализации данной версии. Данная проблема, возможно, будет рассмотрена в дальнейшем.

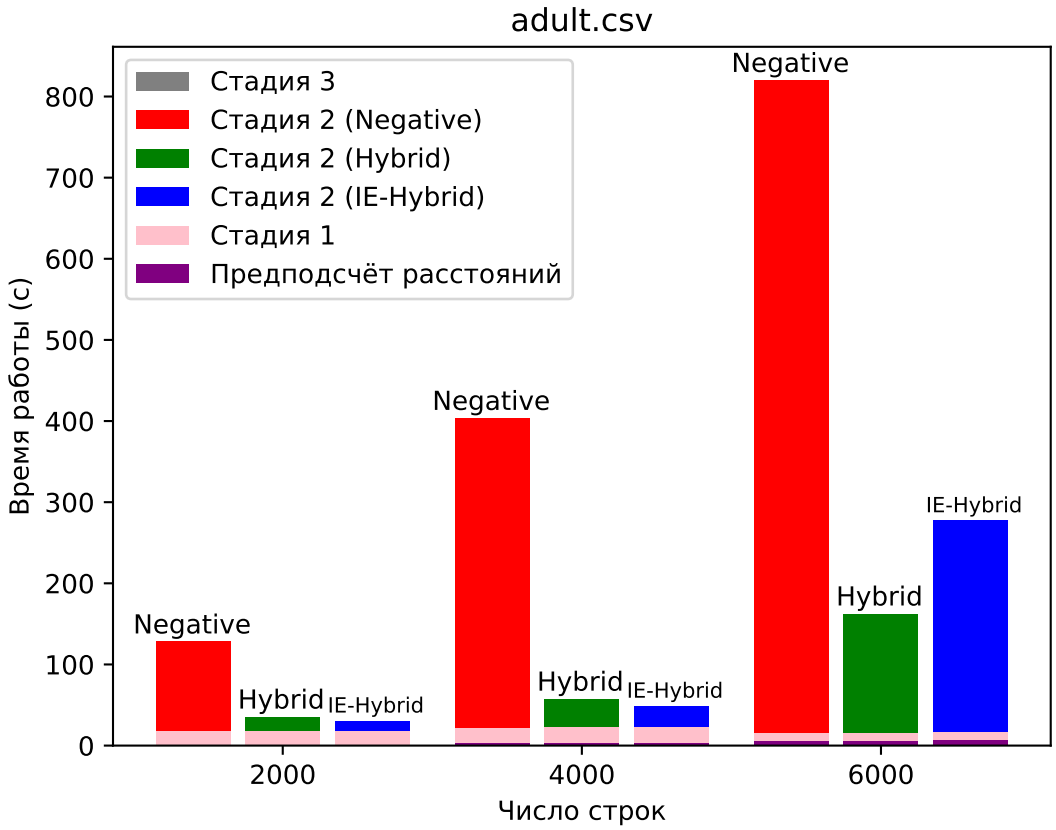

Рис. 5: Время работы различных стадий алгоритма

На рисунке 5 показано время работы различных стадий реализованного алгоритма. Из рисунка видно, что стадия 3 и предподсчёт расстояний в таблице почти не занимают время, а основную часть времени работы алгоритма занимает стадия 2. Именно эту часть алгоритма SPLIT оптимизируют различные его версии.

Заключение

В ходе работы были выполнены следующие задачи:

- сделан обзор предметной области и алгоритма SPLIT;
- реализованы различные версии алгоритма;
- проведено сравнение производительности реализованных версий алгоритма.

Исходный код алгоритма доступен на GitHub[2].

[2] https://github.com/MichaelS239/Desbordante/tree/dd (дата обращения: 2 января 2024 г.).

Список литературы

[1] Efficient Discovery of Differential Dependencies Through Association Rules Mining / Selasi Kwashie, Jixue Liu, Jiuyong Li, Feiyue Ye // Databases Theory and Applications / Ed. by Mohamed A. Sharaf, Muhammad Aamir Cheema, Jianzhong Qi. — Cham : Springer International Publishing, 2015. — P. 3–15.

[2] Song Shaoxu, Chen Lei. Differential Dependencies: Reasoning and Discovery // ACM Trans. Database Syst. — 2011. — aug. — Vol. 36, no. 3. — 41 p. — URL: https://doi.org/10.1145/2000824.2000826.