Санкт-Петербургский государственный университет

Кафедра информационно-аналитических систем

Группа 22.Б08-мм

Обзор алгоритма FastDD поиска дифференциальных зависимостей

Синельников Михаил Алексеевич

Отчёт по учебной практике
в форме «Теоретическое исследование»

Научный руководитель:
ассистент кафедры ИАС Г. А. Чернышев

Санкт-Петербург
2025

Оглавление

Введение

В современном быстро развивающемся мире всё большую роль начинают играть данные. Их количество стремительно растёт, поэтому умение эффективно извлекать полезную информацию из данных становится всё более важным.

Профилирование данных — одна из важных задач, связанных с обработкой данных. Профилирование данных заключается в извлечении метаданных, то есть некоторой информации о данных, из заданного набора данных. Количество строк в файле или его название являются самыми простыми примерами метаданных. Кроме того, метаданными являются и более нетривиальные зависимости, полученные из данных.

Наиболее известный тип таких неочевидных зависимостей, которые можно извлечь из табличных данных — это функциональные зависимости (FD) [5]. Функциональная зависимость имеет форму $X \to Y$, где X, Y — множества атрибутов таблицы. Говорят, что функциональная зависимость удерживается в таблице, если любому уникальному значению на атрибутах X соответствует ровно одно уникальное значение на атрибутах Y. Основная сфера использования функциональных зависимостей — управление базами данных, улучшение их качества.

Реальные данные часто содержат различные неточности: ошибочные записи, опечатки и разные форматы записи данных определённого типа. Перечисленные свойства данных затрудняют использование функциональных зависимостей на практике, поскольку из-за таких ошибок записи, обозначающие одну сущность, не будут в точности равны между собою, что требуется для выполнения функциональных зависимостей [7]. Для решения вышеуказанной проблемы были предложены обобщения функциональных зависимостей, учитывающие неоднородность данных.

Дифференциальные зависимости (DD) [10] – это обобщение функциональных зависимостей, использующее расстояние между значениями для определения степени их различия. Расстояние определяется с помощью метрики, вводящейся для каждого атрибута в зависимости от

его типа данных. Например, для числовых данных в качестве метрики может быть использован модуль разности значений, а для строковых данных — расстояние Левенштейна [8].

Desbordante[1] [1] — наукоёмкий профилировщик данных с открытым исходным кодом, предназначенный для поиска и валидации различных типов зависимостей в табличных данных. На данный момент в Desbordante поддерживается 28 различных видов закономерностей [2], включая функциональные и дифференциальные зависимости.

Мною в рамках Desbordante был реализован[2] [12] и оптимизирован[3,4] [13] алгоритм SPLIT поиска дифференциальных зависимостей, описанный в статье [10]. Однако в 2024 году была опубликована статья [3], авторы которой предложили новый алгоритм поиска DD под названием FastDD[5], а также реализовали алгоритм SPLIT[6] для сравнения с новым алгоритмом. Эксперименты, проведённые в работе, показали, что новый алгоритм решает задачу поиска DD значительно быстрее оригинального. Таким образом, возникла необходимость провести обзор алгоритма FastDD и исследовать целесообразность его реализации в рамках платформы Desbordante с целью повышения производительности поиска DD.

[1] https://github.com/Desbordante/desbordante-core
[2] https://github.com/Desbordante/desbordante-core/pull/374
[3] https://github.com/Desbordante/desbordante-core/pull/439
[4] https://github.com/Desbordante/desbordante-core/pull/506
[5] https://github.com/TristonK/FastDD
[6] https://github.com/TristonK/FastDD-Exp/blob/main/Exp-1/IE.zip

1. Постановка задачи

Целью работы является обзор алгоритма FastDD поиска дифференциальных зависимостей и исследование целесообразности его реализации в рамках платформы Desbordante. Для достижения цели были поставлены следующие задачи:

- провести обзор дифференциальных зависимостей;
- детально рассмотреть принципы работы алгоритма FastDD;
- произвести сравнение производительности открытой реализации алгоритма FastDD и реализации алгоритма SPLIT в Desbordante.

2. Дифференциальные зависимости

2.1. Общие определения

Все определения в данном подразделе повторяют определения из соответствующего подраздела моего отчёта [12] за осень 2023 года.

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 $a_1, a_2 \in dom(A)$.

2.2. Оригинальные определения

Все определения в данном подразделе взяты из оригинальной работы [10], впервые вводящей понятие дифференциальных зависимостей.

Определение 2.3. Дифференциальная функция $\varphi[A]$ — это функция, определённая на атрибуте A с метрикой d_A и возвращающая булево значение, показывающее, выполняется ли утверждение $\forall a_1, a_2 \in dom(A)$ $d_A(a_1, a_2)$ op x, где $op \in \{=, <, >, \leq, \geq\}$, x — численное ограничение.

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

$$\varphi[X] = \varphi[A_1] \wedge \varphi[A_2] \wedge \ldots \wedge \varphi[A_m].$$

Если для пары строк таблицы (t, s) значение дифференциальной функции $\varphi[X]$ истинно, то говорят, что данная пара строк удовлетворяет $\varphi[X]$ ($(t, s) \asymp \varphi[X]$).

Определение 2.5. Пересечением дифференциальных функций $\varphi_1[A], \varphi_2[A]$, определённых на одном атрибуте, называется их логическое умножение: $\varphi_3[A] = \varphi_1[A] \wedge \varphi_2[A]$.

Таблица 1: Пример таблицы (взят из работы [4])

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 $A \in R$ $d_A(x, y) = |x - y|$. Расстояния между строками 1 и 2 на атрибутах YoS, Gen, Sal равны 0, 1 и 3 соответственно. Таким образом, пара строк $\{1, 2\}$ удовлетворяет дифференциальным функциям $\varphi_1[YoS] = [YoS(\leq 0)], \varphi_2[Gen] = [Gen(= 1)], \varphi_3[Sal] = [Sal(> 2)]$. Также ясно, что эта пара строк не удовлетворяет дифференциальным функциям $\varphi_1[YoS] = [YoS(\geq 1)], \varphi_2[Gen] = [Gen(> 1, < 5)], \varphi_3[Sal] = [Sal(\geq 1, \leq 2)]$.

Определение 2.6. Дифференциальная функция $\varphi_1[X]$ включает в себя дифференциальную функцию $\varphi_2[X]$ ($\varphi_1[X] \succeq \varphi_2[X]$), если любая пара строк, удовлетворяющая $\varphi_2[X]$, также удовлетворяет и $\varphi_1[X]$.

Определение 2.7. Дифференциальная зависимость (DD) — это утверждение $\varphi_L[X] \rightarrow \varphi_R[Y]$ между двумя дифференциальными функциями ($X \subset R, Y \subset R$). Дифференциальная зависимость удерживается в таблице, если для любой пары строк таблицы из того, что она удовлетворяет $\varphi_L[X]$, следует то, что она удовлетворяет $\varphi_R[Y]$.

2.3. Альтернативные определения

Все определения в данном подразделе повторяют определения из соответствующего подраздела моего отчёта [12] за осень 2023 года и взяты из работы [4]. Данные определения отличаются от определений, введённых в оригинальной работе [10], однако эти определения имеют много общего.

Определение 2.8. 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 2.9. 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 (t, s) значение дифференциальной функции $X[W_X]$ истинно, то говорят, что данная пара строк удовлетворяет $X[W_X]$ ($(t, s) \asymp X[W_X]$).

Как указано в работе [10], домен каждого атрибута $dom(A)$ считается конечным. Таким образом, множество всевозможных расстояний между значениями из $dom(A)$ также является конечным. Принимая во внимание этот факт, покажем, что любой дифференциальной функции в смысле определения из подраздела 2.2, можно сопоставить дифференциальную функцию в смысле определения из подраздела 2.3.

Пусть $D = \{d_A(a_1, a_2) | a_1 \in dom(A), a_2 \in dom(A)\}$ — множество всех возможных расстояний на атрибуте A. Тогда любой дифференциальной функции вида $[A(< a)]$ соответствует дифференциальная функция вида $[A(\leq max(\{d \in D | d < a\})]$. Аналогично, любой дифференциальной функции вида $[A(> a)]$ соответствует дифференциальная функция вида $[A(\geq min(\{d \in D | d > a\})]$. Таким образом, любую дифференциальную

функцию в смысле определения из подраздела 2.2 можно однозначно записать без использования ограничений $<$ и $>$.

Любой дифференциальной функции вида $[A(\leq a)]$ в смысле оригинального определения соответствует дифференциальная функция вида $A[0, a]$ в смысле альтернативного определения. Любой дифференциальной функции вида $[A(\geq a)]$ соответствует дифференциальная функция вида $A[a, max(\{d|d \in D\})]$. Любой дифференциальной функции вида $[A(= a)]$ соответствует дифференциальная функция вида $A[a, a]$. Любое пересечение дифференциальных функций в смысле оригинального определения, если оно не пусто, представляется ограничением с некоторым отрезком: $A(\geq a, \leq b)$. Такой дифференциальной функции соответствует дифференциальная функция $A[a, b]$.

Таким образом, любой дифференциальной функции в смысле оригинального определения можно однозначно сопоставить дифференциальную функцию в смысле альтернативного определения, поэтому оригинальное и альтернативное определения дифференциальной функции можно считать эквивалентными.

Определение 2.10. 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 $X[W_X]$. Таким образом, данное определение включения в себя можно считать эквивалентным соответствующему определению 2.14, введённому в оригинальной статье [10].

Определение 2.11. Дифференциальная зависимость (DD) — утверждение $X[W_L] \rightarrow Y[W_R]$ между двумя дифференциальными функциями ($X \subset R, Y \subset R$). 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 ($L \cap R = \emptyset$).

Поскольку оригинальное и альтернативное определения дифференциальной функции можно считать эквивалентными, то вытекающие из них определения дифференциальной зависимости также можно считать эквивалентными. Так как альтернативное определение дифференциальной зависимости оказалось более простым для реализации, именно оно было использовано в предыдущих работах для реализации алгоритма SPLIT [12, 13].

2.4. Новые определения

Все определения в данном подразделе взяты из работы [3], предлагающей алгоритм FastDD поиска дифференциальных зависимостей.

Определение 2.12. Дифференциальная функция $\varphi[A]$ — это функция, определённая на атрибуте A с метрикой d_A и возвращающая булево значение, показывающее, выполняется ли утверждение $\forall a_1, a_2 \in dom(A)$ $d_A(a_1, a_2)$ op x, где $op \in \{\leq, >\}$, x — численное ограничение.

Определение 2.13. Дифференциальной функцией $\varphi[X]$ на множестве атрибутов $X = \{A_1, \ldots, A_m\}$ называется логическое умножение дифференциальных функций для каждого атрибута:

$$\varphi[X] = \varphi[A_1] \wedge \varphi[A_2] \wedge \ldots \wedge \varphi[A_m].$$

Если для пары строк таблицы (t, s) значение дифференциальной функции $\varphi[X]$ истинно, то говорят, что данная пара строк удовлетворяет $\varphi[X]$ ($(t, s) \asymp \varphi[X]$).

Определение 2.14. Дифференциальная функция $\varphi_1[X]$ включает в себя дифференциальную функцию $\varphi_2[X]$ ($\varphi_1[X] \succeq \varphi_2[X]$), если любая пара строк, удовлетворяющая $\varphi_2[X]$, также удовлетворяет и $\varphi_1[X]$.

Определение 2.15. Дифференциальная зависимость (DD) — утверждение $\varphi_L[X] \to \varphi_R[A]$ между двумя дифференциальными функциями ($X \subset R, A \in R$). Дифференциальная зависимость удерживается в таблице, если для любой пары строк таблицы из того, что она удовлетворяет $\varphi_L[X]$, следует то, что она удовлетворяет $\varphi_R[A]$.

Определения дифференциальной функции и дифференциальной зависимости, данные в статье [3], имеют существенные отличия от определений, данных в оригинальной статье [10]. В новом определении в правой части дифференциальной зависимости фигурирует дифференциальная функция, определённая на одном атрибуте, в то время как в оригинальном определении соответствующая дифференциальная функция определена на нескольких атрибутах. Однако данное отличие не влияет на эквивалентность определений, так как справедливо следующее утверждение: дифференциальная зависимость $\varphi_L[X] \to \varphi_R[Y]$ ($Y = \{A_1, \ldots, A_n\}$) удерживается тогда и только тогда, когда удерживаются дифференциальные зависимости $\varphi_L[X] \to \varphi_R[A_i]$ $\forall i = 1..n$.

Отличие нового определения дифференциальной функции от оригинального заключается в том, что для дифференциальных функций не определяется их пересечение. Из данного факта следует, что на одном атрибуте возможны ограничения вида $[A(\leq a)]$ и $[A > a]$, но невозможны ограничения вида $[A(= a)]$ и $[A(\geq a, \leq b)]$. Таким образом, область возможных дифференциальных функций сужается, и новое определение дифференциальной функции, а значит, и дифференциальной зависимости, нельзя считать эквивалентным оригинальному. Таблица 2 показывает эквивалентные ограничения для трёх различных определений дифференциальной функции.

Таблица 2: Эквивалентные ограничения для различных определений дифференциальных функций

Оригинальное определение (2.2)	Альтернативное определение (2.3)	Новое определение (2.4)
$[A(\leq a)]$	$A[0, a]$	$[A(\leq a)]$
$[A(\geq a)]$	$A[a, max(\{d \mid d \in D\})]$	$[A(> max(\{d \in D \mid d < a\}))]$
$[A(< a)]$	$A[0, max(\{d \in D \mid d < a\})]$	$[A(\leq max(\{d \in D \mid d < a\}))]$
$[A(> a)]$	$A[min(\{d \in D \mid d > a\}), max(\{d \mid d \in D\})]$	$[A(> a)]$
$[A(= a)]$	$A[a, a]$	—
$[A(\geq a, \leq b)]$	$A[a, b]$	—

2.5. Дополнительные определения

Все определения в данном подразделе повторяют определения из соответствующего подраздела моего отчёта [12] за осень 2023 года и взяты из работы [4]. Несмотря на то, что определения дифференциальных функций и дифференциальных зависимостей отличаются, основанные на них определения данного подраздела являются общими. Для удобства для определений используются обозначения и определения из подраздела 2.3.

Определение 2.16. 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 2.17. Пусть Σ — множество дифференциальных зависимостей. DD $X[W_L] \rightarrow Y[W_R] \in \Sigma$ называется минимальной, если выполняются следующие условия:

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

Определение 2.18. 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 2.19. Покрытие Σ_c множества Σ называется минимальным, если не существует покрытия $\Sigma' \subseteq \Sigma_c$.

Пример 2.2. В качестве примера вновь рассмотрим таблицу 1. Возьмём в качестве метрики для каждой колонки $A \in R$ $d_A(x, y) = |x - y|$. Дифференциальная зависимость $YoS[0, 0] \rightarrow Edu[1, 1]$ 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 $YoS[2, 2] \rightarrow Sal[0, 0]$ 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

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

Множество DD

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

является минимальным покрытием Σ. Действительно, любая DD из Σ является логическим следствием DD из Σ_c. Например, если удерживается DD $YoS[0, 7] \rightarrow Edu[0, 0]$, 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 2.3. 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 2.17. 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.

3. Алгоритм FastDD

В 2024 году в статье [3] был представлен алгоритм FastDD поиска дифференциальных зависимостей. Судя по результатам экспериментов, проведённых авторами статьи, алгоритм FastDD является более производительным, чем алгоритм SPLIT. В данном разделе будет проведён подробный обзор алгоритма FastDD.

3.1. Определения, используемые в алгоритме

В данном подразделе представлены определения, которые активно используются в описании алгоритма FastDD.

Определение 3.1. Пусть Ψ — множество дифференциальных функций $\varphi[A], A \in R$. Множеством различий для пары строк (t, s) называется множество $D(t, s) = \{\varphi[A] \in \Psi | (t, s) \not\asymp \varphi[A]\}$, то есть подмножество множества дифференциальных функций Ψ, которым не удовлетворяет пара строк (t, s).

Определение 3.2. Множеством различий для экземпляра отношения r называется множество $D_r = \{D(t, s) | t \in r, s \in r, D(t, s) \neq \emptyset\}$, то есть множество непустых множеств различий для пар строк таблицы.

Определение 3.3. Пусть $\varphi[A] \in \Psi$. Обозначим за $D_r(\varphi[A])$ множество множеств различий, содержащих в себе $\varphi[A]$: $D_r(\varphi[A]) = \{U | U \in D_r \wedge \varphi[A] \in U\}$.

Утверждение 3.1. DD $\varphi_L[X] \rightarrow \varphi[A]$, где $\varphi[X] = \bigwedge_{A_i \in X} \varphi_i[A_i], A \notin X$, удерживается в таблице (то есть экземпляре отношения r) тогда и только тогда, когда $\forall U \in D_r(\varphi[A])\ \exists \varphi_i[A_i] : \varphi_i[A_i] \in U$.

Утверждение 3.1 позволяет связать задачу поиска дифференциальных зависимостей с задачей поиска вершинного покрытия. Следующие определения, связанные с понятием вершинного покрытия взяты из работ [6] и [9] и повторяют секцию обзора моего отчёта [11] за весну 2024 года.

Определение 3.4. Гиперграф $\mathcal{F}$ — множество подмножеств множества вершин V ($F = \{F_1, \ldots, F_m\}, F_i \subseteq V$). F_i — рёбра гиперграфа. $|F|$ — число рёбер гиперграфа.

Определение 3.5. Вершинное покрытие гиперграфа $\mathcal{F}$ (transversal, hitting set) — подмножество V, имеющее непустое пересечение с каждым ребром $\mathcal{F}$.

Определение 3.6. Вершинное покрытие S гиперграфа $\mathcal{F}$ называется минимальным, если не существует вершинного покрытия $S_1 : S_1 \subset S$.

Таким образом, в силу утверждения 3.1, задача поиска дифференциальных зависимостей с правой частью $\varphi[A]$, удерживающихся в таблице, эквивалентна задаче поиска вершинного покрытия гиперграфа $D_r(\varphi[A])$. Однако задача поиска минимального покрытия всех удерживающихся DD не эквивалентна задаче поиска минимального вершинного покрытия в гиперграфе, так как, помимо минимальности по включению в смысле множеств, для DD также требуется минимальность по включению в себя (2.14). Для того, чтобы результаты поиска были верными, необходимы дополнительные проверки на минимальность, то есть стандартный алгоритм поиска минимального вершинного покрытия должен быть модифицирован. Более подробно модифицированный алгоритм будет описан в следующих подразделах.

3.2. Описание алгоритма

Алгоритм FastDD поиска дифференциальных зависимостей состоит из нескольких стадий:

1. Препроцессинг
2. Построение множества различий
3. Поиск и минимизация вершинных покрытий

Схема работы алгоритма представлена на рисунке 1. На этапе препроцессинга считывается входная таблица и определяется пространство

поиска дифференциальных зависимостей. На этапе построения множества различий из построенного пространства поиска строится множество различий. На этапе поиска вершинного покрытия для полученного множества различий строятся вершинные покрытия, состоящие из удерживающихся в таблице DD. На этапе минимизации вершинных покрытий из полученных DD отсеиваются не минимальные. В результате работы алгоритма получается минимальное покрытие дифференциальных зависимостей, удерживающихся в таблице. Каждая стадия алгоритма будет описана более подробно в следующих подразделах.

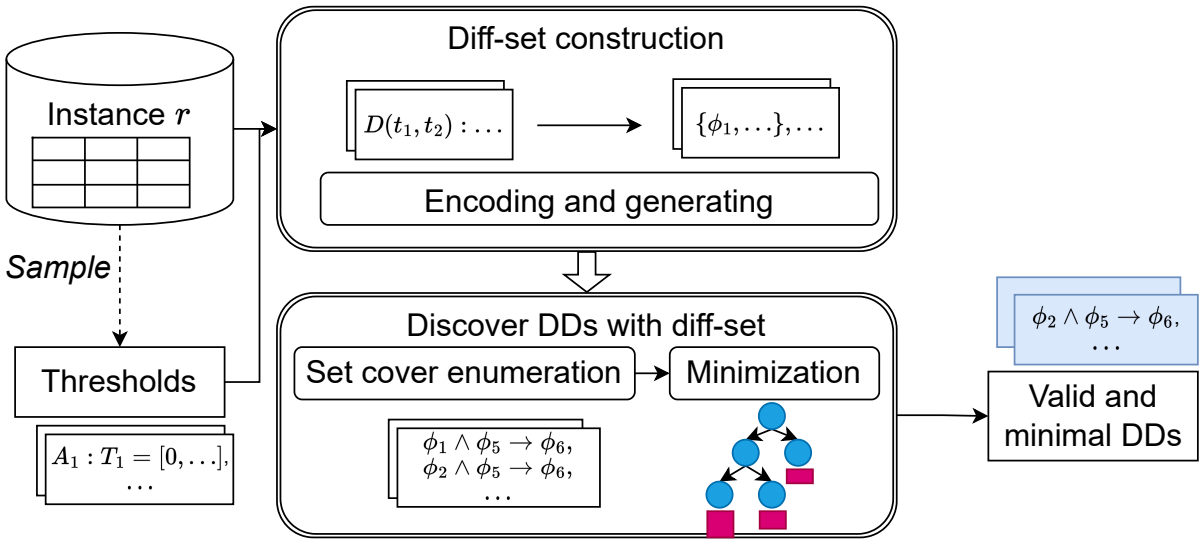

Рис. 1: Схема алгоритма FastDD (взято из работы [3])

3.2.1. Препроцессинг

На этапе препроцессинга алгоритм считывает входную таблицу и определяет пространство поиска. В работе [3] описываются несколько способов построения пространства поиска, которые относятся к двум типам: автоматическое определение ограничений, участвующих в дифференциальных функциях, и предоставление ограничений самим пользователем. В открытой реализации алгоритма FastDD[7] от авторов статьи [3] могут быть использованы оба способа задания пространства поиска.

[7] https://github.com/TristonK/FastDD

3.2.2. Построение множества различий

На этапе построения множества различий по множеству дифференциальных функций Ψ, полученному на этапе препроцессинга, строится множество различий D_r. При построении используется специальная сокращённая запись множеств различий для каждой пары строк с целью экономии потребляемой памяти.

Пусть для атрибута $A_i \in R$ было определено $|T_i|$ численных ограничений (T_i — множество ограничений, отсортированное по возрастанию). Данные ограничения порождают $|T_i|$ отрезков разбиения, выглядящих следующим образом: $(T_i[0], T_i[1]], \ldots, (T_i[|T_i| - 1], \infty)$. Любое расстояние между строками таблицы на атрибуте A_i попадает в один из этих отрезков разбиения, поэтому расстоянию $dist$ можно однозначно сопоставить номер соответствующего отрезка разбиения $\#_{A_i}(dist)$, принимающий значения от 0 до $|T_i|$.

Для того, чтобы представить $D(t, s)$ в компактной форме, используется следующая формула: $D(t, s) = a_1 + a_2 * S_1 + \cdots + a_{|R|} * S_{|R|-1}$, где $a_i = \#_{A_i}(d_{A_i}(t_{A_i}, s_{A_i})), S_i = \prod_{k=1}^{i}(|T_k| + 1)$. При известных T_i по номеру, полученному по данной формуле, можно однозначно определить номера a_i, а с помощью a_i для любой дифференциальной функции $\varphi_i[A_i]$ можно однозначно определить, удовлетворяет ли пара строк этой дифференциальной функции.

Для ускорения построения множества различий используются различные техники. Расстояния подсчитываются не между каждой парой строк, а между каждой парой кластеров равных значений, которых обычно значительно меньше. Для эффективного хранения этих кластеров используются индексы списков позиций (Position List Index, PLI) [5]. Кроме того, для атрибутов, значения которых могут быть отсортированы, данное свойство используется для ускорения подсчёта расстояний. Для больших датасетов используется техника шардирования, заключающаяся в разбиении таблицы на несколько блоков, вычислении множеств различий для каждого блока и последующем объединении множеств по всем блокам. Данная техника позволяет ускорить

подсчёт расстояний, а также делает возможным параллельные вычисления для разных блоков, что позволяет добиться значительного ускорения на современных многоядерных процессорах.

3.2.3. Поиск и минимизация вершинных покрытий

На этапе поиска и минимизации вершинных покрытий для каждой дифференциальной функции $\varphi[A_i] \in \Psi$ находится минимальное вершинное покрытие для гиперграфа $D_r(\varphi[A_i])$, а затем дополнительно минимизируется с учётом включения в себя. Листинг данного этапа алгоритма представлен на рисунке 2.

Перед поиском множество Ψ сортируется так, что дифференциальная функция $\varphi'[A_i]$ находится перед дифференциальной функцией $\varphi[A_i]$, если $\varphi[A_i] \succeq \varphi'[A_i]$. Данная сортировка проводится с целью ускорения процедуры минимизации.

Для каждой дифференциальной функции $\varphi[A_i] \in \Psi$ по полученному на предыдущей стадии множеству различий D_r строится множество $D_r(\varphi[A_i])$ и фиксируется множество $\Psi' = \{\varphi[A_j] \in \Psi | i \neq j\}$. $\varphi[A_i]$ является правой частью конструируемых DD, а левая часть будет получена с помощью пересечения дифференциальных функций из Ψ'.

Как было отмечено в подразделе 3.1, задача поиска DD с правой частью $\varphi[A_i]$, удерживающихся в таблице, эквивалентна задаче поиска вершинного покрытия гиперграфа $D_r(\varphi[A_i])$. Для поиска формируется множество Γ возможных кандидатов. Перед поиском $\Gamma = \{\{\varphi\} | \varphi \in \Psi'\}$, то есть кандидаты на роль левой части зависимости состоят из одной дифференциальной функции.

В процессе поиска перебирается каждое ребро гиперграфа ($U \in D_r(\varphi[A_i])$) и для него проверяется, есть ли пересечение с кандидатами. Если для $\gamma \in \Gamma$ $\gamma \cap U \neq \emptyset$, то данные кандидаты остаются в Γ. Если $\gamma \cap U = \emptyset$, то данный кандидат не является вершинным покрытием. Каждый такой кандидат γ удаляется из Γ, и на его основе генерируется новое множество кандидатов $\{\gamma \cup \varphi'[A_j] | \varphi'[A_j] \in U \setminus \varphi[A_i]\}$, которые добавляются в Γ, если они минимальны по включению и γ не содержит дифференциальной функции, определённой на A_j. Таким образом, кан-

Algorithm 2: DD discovery based on D_r (GenDD)

Input: the diff-set D_r of r, and the set Ψ of singleton differential functions $\phi[A_i]$, $\forall A_i \in R$

Output: the set Σ of minimal and valid DDs on r

$\Sigma \leftarrow \emptyset$

sort Ψ based on a partial order, such that $\forall \phi'[A_i], \phi[A_i] \in \Psi$, $\phi'[A_i]$ is before $\phi[A_i]$ if $\phi[A_i] \succ \phi'[A_i]$

foreach $\phi[A_i] \in \Psi$ **do**

  $\Psi' \leftarrow \{\phi'[A_j] \in \Psi \mid i \neq j\}$

  $\Gamma \leftarrow \text{Cover}(\Psi', D_r(\phi[A_i]))$

  $\Gamma \leftarrow \text{Minimize}(\Sigma, \Gamma, \phi[A_i])$

  foreach $\phi_L[X] \in \Gamma$ **do**

    $\Sigma \leftarrow \Sigma \cup \{\ \phi_L[X] \rightarrow \phi[A_i]\ \}$

Function Cover(Ψ', $D_r(\phi[A_i])$)

  $\Gamma \leftarrow \{\{\phi\} \mid \phi \in \Psi'\}$

  foreach $U \in D_r(\phi[A_i])$ **do**

    $\Gamma^- \leftarrow \{\gamma \in \Gamma \mid \gamma \cap U = \emptyset\}$ `// no intersection with` U

    $\Gamma \leftarrow \Gamma \backslash \Gamma^-$

    foreach $\gamma \in \Gamma^-$ **do**

      foreach $\phi'[A_j] \in U \backslash \{\phi[A_i]\}$ **do**

        if $\exists \phi''[A_j] \in \gamma$ **then**

          continue `// already a function on` A_j

        if $\nexists \gamma' \in \Gamma$ *such that* $\gamma' \subseteq (\gamma \cup \{\ \phi'[A_j]\})$ **then**

          $\Gamma \leftarrow \Gamma \cup \{\ \gamma \cup \{\ \phi'[A_j]\}\}$

  return Γ

Function Minimize(Σ, Γ, $\phi[A_i]$)

  $\Gamma_{full} \leftarrow \{\ \phi_L[X] \mid \phi_L[X] \rightarrow \phi'[A_i] \in \Sigma \wedge \phi[A_i] \succ \phi'[A_i]\ \}$

  sort Γ based on a partial order, such that $\forall \phi_L[X], \phi'_L[X'] \in \Gamma$, $\phi'_L[X']$ is before $\phi_L[X]$ if $\phi'_L[X'] \succ \phi_L[X]$

  $\Gamma_{new} \leftarrow \emptyset$

  foreach $\phi_L[X] \in \Gamma$ **do**

    if $\nexists \phi'_L[X'] \in \Gamma_{full}$ *such that* $\phi'_L[X'] \succeq \phi_L[X]$ **then**

      $\Gamma_{full} \leftarrow \Gamma_{full} \cup \phi_L[X]$

      $\Gamma_{new} \leftarrow \Gamma_{new} \cup \phi_L[X]$

  return Γ_{new}

Рис. 2: Листинг этапа 3 алгоритма FastDD (взято из работы [3])

дидаты, оставшиеся в Γ после обработки каждого ребра $U \in D_r(\varphi[A_i])$, имеют непустое пересечение с каждым из рёбер гиперграфа $D_r(\varphi[A_i])$, поэтому Γ является минимальным вершинным покрытием $D_r(\varphi[A_i])$.

Полученное покрытие состоит из дифференциальных функций, минимальных по включению, но не по включению в себя. Таким образом, для получения минимальных DD требуется дополнительная минимизация вершинного покрытия. Для этого полученное множество Γ сортируется так, что дифференциальная функция $\varphi'_L[X] \in \Gamma$ находится перед дифференциальной функцией $\varphi_L[X'] \in \Gamma$, если $\varphi'_L[X'] \succeq \varphi_L[X]$. Также на основе уже построенных DD формируется множество $\Gamma_{full} = \{\varphi_L[X] | \varphi_L[X] \to \varphi'[A_i] \in \Sigma \wedge \varphi[A_i] \succeq \varphi'[A_i]\}$, где Σ — текущее множество построенных DD. Далее для каждого кандидата $\varphi_L[X] \in \Gamma$ проверяется, есть ли среди уже построенных DD их левая часть $\varphi'_L[X]$ такая, что $\varphi'_L[X'] \succeq \varphi_L[X]$. Если такая дифференциальная функция $\varphi'_L[X]$ найдётся, то кандидат $\varphi_L[X] \to \varphi[A_i]$ не является минимальной DD. Действительно, в таком случае эта DD следует из DD $\varphi'_L[X] \to \varphi'[A_i]$ следующим образом: $(t, s) \asymp \varphi_L[X] \Rightarrow (t, s) \asymp \varphi'_L[X] \Rightarrow (t, s) \asymp \varphi'[A_i] \Rightarrow (t, s) \asymp \varphi[A_i]$. Первое следствие получается из факта $\varphi'_L[X'] \succeq \varphi_L[X]$, второе — из факта $\varphi'_L[X] \to \varphi'[A_i] \in \Sigma$, третье — из факта $\varphi[A_i] \succeq \varphi'[A_i]$. Если такой дифференциальной функции $\varphi'_L[X]$ не найдётся, то кандидат $\varphi_L[X] \to \varphi[A_i]$ является минимальной DD и добавляется в Σ. Таким образом, в результате получается множество Σ, являющееся минимальным покрытием DD, удерживающихся в таблице.

4. Эксперимент

В секции экспериментов статьи [3] было проведено сравнение производительности алгоритма FastDD и алгоритма SPLIT. Для сравнения были использованы открытые реализации алгоритмов на языке Java от авторов статьи. Эксперименты показали, что алгоритм FastDD значительно производительнее алгоритма SPLIT.

Для исследования целесообразности реализации алгоритма FastDD в рамках Desbordante было решено сравнить скорость работы текущей реализации алгоритма SPLIT в Desbordante и открытой реализации алгоритма FastDD. Эксперименты проводились на ноутбуке с установленной операционной системой Linux Ubuntu 24.04, 8 ГБ оперативной памяти и 8 процессорами Intel(R) Core(TM) i5-9300H CPU @ 2.40GHz 2.40 GHz (максимальная частота — 4.1 GHz). При проведении экспериментов был выключен файл подкачки, сброшены кэши и зафиксирована максимальная частота процессоров. Все представленные показатели времени работы алгоритмов являются усреднёнными значениями по пяти замерам. Для каждого атрибута использовалось по 5 дифференциальных функций для формирования пространства поиска.

В таблице 3 представлено время работы реализации алгоритма SPLIT в Desbordante и реализации алгоритма FastDD от авторов статьи [3] на разных датасетах. ML означает, что исполнение прервалось из-за недостатка оперативной памяти.

Из таблицы можно увидеть, что реализация алгоритма FastDD превосходит реализацию алгоритма SPLIT в Desbordante как по времени работы, так и по затрачиваемой памяти. Среднее ускорение работы — 1.58, однако на больших датасетах оно становится больше и достигает 4.48 (на датасете adult с 9 колонками). Стоит отметить, что класс зависимостей, которые находит новый алгоритм, несколько уже, что было показано в секции обзора. Однако производительность данного алгоритма позволяет обрабатывать большие по размеру датасеты, что является ключевым преимуществом алгоритма FastDD перед алгоритмом SPLIT. Таким образом, можно сделать вывод, что алгоритм FastDD

Таблица 3: Время работы (с)

Датасет	#колонок	#строк	#DD	SPLIT	FastDD
adult	5	2000	9	0.272	0.404
adult	5	4000	10	1.031	1.000
adult	5	6000	11	2.343	1.960
adult	5	8000	11	4.081	3.272
adult	5	10000	9	6.357	5.030
adult	5	32561	5	68.731	36.598
adult	6	32561	8	80.156	39.028
adult	7	32561	14	92.526	41.727
adult	8	32561	19	107.621	44.778
adult	9	32561	37	213.793	47.681
adult	10	32561	72	ML	48.694
abalone	5	4177	9	0.801	0.956
abalone	6	4177	18	0.977	1.166
abalone	9	4177	117	1.750	1.568
digits	6	1797	0	0.336	0.406
digits	9	1797	16	2.317	0.997
flights	10	1000	17	0.065	0.228
flights	15	1000	61	46.745	0.324
neighbors10k	7	10000	12	5.117	5.793

имеет смысл реализовать в рамках платформы Desbordante с целью повышения скорости решения задачи поиска дифференциальных зависимостей в Desbordante.

Заключение

В ходе работы были выполнены следующие задачи:

- сделан обзор дифференциальных зависимостей и их различных определений;
- проведён детальный обзор алгоритма FastDD;
- проведено сравнение производительности открытой реализации алгоритма FastDD и реализации алгоритма SPLIT в Desbordante.

В результате экспериментов была показана целесообразность реализации алгоритма FastDD в рамках платформы Desbordante. Реализовать данный алгоритм планируется в следующем семестре.

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] Desbordante: from benchmarking suite to high-performance science-intensive data profiler / George Chernishev, Michael Polyntsov, Anton Chizhov et al. // Proceedings of the 8th International Conference on Data Science and Management of Data (12th ACM IKDD CODS and 30th COMAD). — CODS-COMAD '24. — New York, NY, USA : Association for Computing Machinery, 2025. — P. 234–243. — URL: https://doi.org/10.1145/3703323.3703725.

[3] Efficient Differential Dependency Discovery / Shulei Kuang, Honghui Yang, Zijing Tan, Shuai Ma // Proc. VLDB Endow. — 2024. — may. — Vol. 17, no. 7. — P. 1552–1564. — URL: https://doi.org/10.14778/3654621.3654624.

[4] 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.

[5] Functional dependency discovery: an experimental evaluation of seven algorithms / Thorsten Papenbrock, Jens Ehrlich, Jannik Marten et al. // Proc. VLDB Endow. — 2015. — jun. — Vol. 8, no. 10. — P. 1082–1093. — URL: https://doi.org/10.14778/2794367.2794377.

[6] Hitting set enumeration with partial information for unique column combination discovery / Johann Birnick, Thomas Bläsius, Tobias Friedrich et al. // Proc. VLDB Endow. — 2020. — jul. —

Vol. 13, no. 12. — P. 2270–2283. — URL: https://doi.org/10.14778/3407790.3407824.

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

[8] Levenshtein Vladimir I et al. Binary codes capable of correcting deletions, insertions, and reversals // Soviet physics doklady / Soviet Union. — Vol. 10. — 1966. — P. 707–710.

[9] Murakami Keisuke, Uno Takeaki. Efficient algorithms for dualizing large-scale hypergraphs // Discrete Appl. Math. — 2014. — jun. — Vol. 170. — P. 83–94. — URL: https://doi.org/10.1016/j.dam.2014.01.012.

[10] 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.

[11] Синельников Михаил. Интеграция алгоритма поиска UCC в рамках платформы Desbordante. — 2024. — URL: https://github.com/Desbordante/desbordante-core/blob/main/docs/papers/HPIValidintegration-MichaelSinelnikov-2024spring.pdf.

[12] Синельников Михаил. Реализация алгоритма поиска дифференциальных зависимостей в рамках платформы Desbordante. — 2024. — URL: https://github.com/Desbordante/desbordante-core/blob/main/docs/papers/SPLIT-MichaelSinelnikov-2023autumn.pdf.

[13] Синельников Михаил. Оптимизация алгоритма SPLIT поиска дифференциальных зависимостей в рамках платформы Desbordante. — 2025. — URL: https://github.com/Desbordante/desbordante-core/blob/main/docs/papers/SPLIToptimization-MichaelSinelnikov-2024autumn.pdf.