Санкт-Петербургский государственный университет

Кафедра информационно-аналитических систем

Группа 23М04-мм

Реализация алгоритмов поиска и верификации AOD в платформе Desbordante

Полынцов Михаил Александрович

Отчёт по учебной практике
в форме «Решение»

Научный руководитель:
ассистент кафедры информационно-аналитических систем Г. А. Чернышев

Санкт-Петербург
2024

Оглавление

Введение

Профилирование данных представляет собой процесс анализа, целью которого является извлечение метаданных [1]. Этот процесс можно подразделить на два основных типа: ненаукоемкий и наукоемкий [7]. Ненаукоемкий тип профилирования направлен на извлечение базовых метаданных, таких как дата создания данных и их авторы. Наукоемкий тип включает в себя поиск более глубокой информации — закономерностей в данных.

Данные, подвергаемые анализу, можно классифицировать по логической модели, к которой они относятся: графовые, табличные, транзакционные, пространственные и т.д. [2]. Модель данных во многом определяет методы, применяемые в алгоритмах для обнаружения закономерностей. В рамках данной работы мы будем рассматривать только табличные данные.

Существует множество способов формального определения закономерностей в табличных данных [4,5]. Одним из самых распространённых примеров такой формализации являются функциональные зависимости [12]. В данной работе эти формальные описания мы будем называть примитивами или зависимостями. Информация о закономерностях используется для решения различных прикладных задач, таких как очистка данных (data cleaning), исследование данных (data exploration), реверс-инжиниринг баз данных (database reverse engineering) и оптимизация запросов в системах управления базами данных [1, 13]. В связи с этим активно разрабатываются алгоритмы автоматизированного поиска зависимостей.

Зависимости порядка (order dependencies, OD), обобщающие функциональные зависимости, также являются примером формализации закономерностей. Неформально говоря, зависимость порядка определяет, что что отсортировав данные по одному набору атрибутов, они также окажутся отсортированными по некоторому другом набору атрибутов. Информация о том, какие OD удерживаются над конкретной таблицей, может быть использована, например, для очистки данных [10, 11, 15].

Реальные данные часто имеют ошибки или какие-то неточности, из-за которых зависимость может не удерживаться точно. Поэтому на практике часто используются приближенные варианты зависимостей, которые являются толерантными к некоторому значению ошибки. Соответственно зависимости порядка, которые удерживаются на данных, но с какими-то исключениями, называются приближенными (approximate order dependencies, AOD). Поиск таких зависимостей над заданной таблицей является важной функциональностью наукоемкого профилировщика данных.

Desbordante [6] представляет собой платформу для наукоемкого профилирования данных с открытым исходным кодом[1]. В ней реализованы алгоритмы для поиска различных примитивов, а также интерфейсы для работы с ними, что позволяет решать прикладные задачи анализа данных. Desbordante активно развивается и в данный момент не поддерживает приближенные зависимости порядка. Это стало мотивацией для темы данной работы.

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

1 Постановка задачи

Целью данной работы является реализация алгоритма верификации приближенных зависимостей порядка, а также реализация алгоритма их поиска в Desbordante. Для достижения этой цели были поставлены следующие задачи.

1. Выполнить обзор предметной области.
2. Реализовать алгоритм поиск AOD в платформе Desbordante.
3. Реализовать алгоритм верификации AOD в платформе Desbordante.

2 Обзор

Для того чтобы формально определить зависимости порядка, введём некоторые обозначения.

Отношения. Через R будем обозначать *отношение* (схему), а через r будем обозначать *экземпляр отношения*. Через A, B и C будем обозначать одиночные *атрибуты*, через s и t — *кортежи*, а через t_A будем обозначать значение атрибута A в кортеже t.

Множества. Через $\mathcal{X}$ и $\mathcal{Y}$ будем обозначать *множества атрибутов*. Тогда $t_\mathcal{X}$ обозначает *проекцию* кортежа t на $\mathcal{X}$. $\mathcal{XY}$ используется как сокращение для $\mathcal{X} \cup \mathcal{Y}$. Пустое множество атрибутов обозначается как $\{\}$.

Списки. X, Y и Z обозначают *списки атрибутов*. X может являться пустым списком, явно пустой список будем обозначать как $[]$. $[A, B, C]$ обозначает список из атрибутов A, B и C. $[A \mid T]$ обозначает список с первым атрибутом A и оставшимся "хвостом" T, то есть оставшейся частью списка после удаления первого элемента. Через XY будем обозначать конкатенацию списков X и Y. Обозначим через $\mathcal{X}$ множество элементов в списке X. Через X' обозначим некоторую перестановку элементов списка X.

Определение 1. Пусть X — это список атрибутов. Будем говорить, что выполнено $t \preceq_X s$ для двух кортежей t и s, $\mathcal{X} \in R$, если:

- $X = []$; или
- $X = [A \mid T]$ и $t_A < s_A$; или
- $X = [A \mid T]$, $t_A = s_A$, и $t \preceq_T s$.

Определение 2. Пусть X и Y — списки атрибутов. Тогда $X \mapsto Y$ обозначает зависимость порядка. Зависимость $X \mapsto Y$ удерживается, если $\forall t, s, \in r : t \preceq_X \Rightarrow t \preceq_Y$.

Зависимость $X \mapsto Y$ означает, что если отсортировать таблицу по атрибутам X, она также окажется отсортированной по атрибуту Y. Например, в таблице 1 удерживаются зависимости $sal \mapsto taxGrp$ и $pos, exp \mapsto bonus$.

#	pos	exp	sal	taxGrp	tax	bonus
t1	acc	1	20	A	2K	1K
t2	acc	3	25	A	2.5K	1.5K
t3	dev	1	30	A	0.3K	3K
t4	acc	5	40	B	12K	2K
t5	dev	3	50	B	1.5K	4K
t6	dev	5	55	B	16.5K	4K
t7	dev	5	60	B	1.8K	4K

Таблица 1: Пример таблицы с зависимостями порядка

У зависимостей порядка есть два важных свойства:

- $X \mapsto Y \not\Rightarrow Y \mapsto X$.
- $XA \mapsto Y \not\Rightarrow AX \mapsto Y$.

Из этих свойств следует необходимость в алгоритм поиска зависимостей порядка в худшем случае проверять все перестановки, что приводит к факториальной сложности алгоритма от числа атрибутов в таблице. На практике это может пагубно влиять на производительность алгоритма.

Для того чтобы избежать необходимости проверять перестановки, в работе [8] была представлена *set-based аксиоматизация* и алгоритм FastOD для поиска зависимостей в set-based аксиоматизации. Выше описана *list-based аксиоматизации*, алгоритм Order, ищущий зависимости такого вида, представлен в работе [14].

Определение 3. Будем говорить, что атрибут A является *константой* в каждом классе эквивалентности относительно $\mathcal{X}$, что обозначается как $\mathcal{X} : [] \mapsto_{\text{const}} A$, если $X' \to XA$ для любой перестановки X' списка X.

$\mathcal{X} : [] \to_{\text{const}} A$ эквивалентна функциональной зависимости $X \to A$. В таблице 1 удерживаются $sal : [] \mapsto_{\text{const}} exp$ и $bonus : [] \mapsto_{\text{const}} tax$.

Определение 4. Списки атрибутов X и Y являются "совместимыми по порядку" (*order compatible*), что будем обозначать как $X \sim Y$, тогда и только тогда, когда $XY \leftrightarrow YX$. Пустой список атрибутов совместим по порядку с любым другим списком атрибутов.

Неформально совместимость по порядку двух атрибутов A и B обозначает, что в таблице существует такой порядок кортежей, что они одновременно отсортированы и по A, и по B. Совместимость по порядку требование более слабое, чем зависимость порядка. Например, в таблице 1 удерживается $pos \sim exp$, но не $pos \mapsto exp$ или $exp \mapsto pos$.

Определение 5. Два атрибута A и B являются совместимыми по порядку в каждом классе эквивалентности $\mathcal{X}$ (*order compatible within each equivalence class with respect to* $\mathcal{X}$), что будем обозначать как $\mathcal{X} : A \sim B$, если $X'A \sim X'B$.

В таблице 1 удерживается $pos : exp \sim bonus$, но не $exp \sim bonus$.

Зависимости из определения 4 и определения 5 так же называют каноничными (*canonical order dependencies*). Существует [8] отображение зависимостей, определенных в list-based аксиоматизации на зависимости, определенные в set-based аксиоматизации.

Определение 6. Пусть дана функция ошибки g и порог ошибки e, тогда зависимость порядка γ является приближенной зависимостью порядка (*approximate order dependency, AOD*), если и только если $g(\gamma, e)$.

В работе [3] определяется две функции ошибки для list-based аксиоматизации: g_1 и g_3.

$$g_1(X \to Y, r) = g_{\text{split}}(X \to Y, r) + 2 \times g_{\text{swap}}(X \to Y, r),$$

где

$$g_{\text{split}}(X \to Y, r) = \frac{|\{(t,s) \in r^2 \mid t_X = s_X \wedge t_Y \neq s_Y\}|}{|r^2| - |r|}$$

и

$$g_{\text{swap}}(X \to Y, r) = \frac{|\{(t,s) \in r^2 \mid t_X = s_X \wedge t \preceq_Y s \wedge s \preceq_Y t\}|}{|r^2| - |r|}.$$

g_1 по сути является отношением количества пар кортежей, на которых нарушается зависимость порядка, к общему числу пар кортежей в таблице.

$$g_3(\gamma, r) = \frac{|r| - \max(|r'| \mid r' \subseteq r,\ r' \text{ satisfies } \gamma)}{|r|}$$

g_3 в качестве ошибки возвращает отношение количества кортежей,

которые нужно убрать из таблицы, чтобы удерживалась точная зависимость, к общему числу кортежей. Для зависимостей set-based аксиоматизации определение g_3 остается таким же.

В данный момент известно два алгоритма поиска AOD:

- Алгоритм FastOD [9]:
  - ищет зависимости в set-based аксиоматизации;
  - поиск точных зависимостей уже реализован в Desbordante;
  - использует функцию ошибки g_3.
- Алгоритм DisOD [3]:
  - ищет зависимости в list-based аксиоматизации;
  - не реализован в Desbordante, никакой реализации нет в открытом доступе, только псевдокод в исходной работе;
  - может использовать как g_1, так и g_3 функцию ошибки.

Оба алгоритма были опубликованы в 2021 году примерно в одно и то же время, поэтому в работах, их описывающих, нет упоминаний друг друга и экспериментальных сравнений. Для реализации в весеннем семестре была выбрана set-based аксиоматизация и FastOD, так как:

- точный FastOD уже реализован и оптимизирован в Desbordante, что ускорит реализацию;
- FastOD в теории должен быть значительно быстрее засчет использования set-based аксиоматизации.

3 Реализация

3.1 Верификация AOD

Задача верификации может быть сформулирована следующим образом. На вход алгоритму подается таблица и OD, необходимо проверить, удерживается ли OD точно и если не удерживается, то вернуть значение ошибки, при котором будет удерживаться соответствующая AOD. Пользователю также могут быть интересны конкретные строки, из-за которых точная OD не удерживается, потому что именно такие строки могут содержать потенциальные ошибки. Так как в качестве функции ошибки была реализована g_3, такими строками является весь набор кортежей, который из таблицы нужно убрать, чтобы зависимость удерживалась точно. Сама верификация в сущности заключается в том, чтобы вычислить такой минимальный набор кортежей, называемый *removal set*. Если removal set пуст, то зависимость удерживается точно, иначе с порогом ошибки $|removal\ set|/|r|$.

Для верификации зависимостей в set-based аксиоматизации необходимо уметь верифицировать два вида зависимостей: canonical OC и OFD.

Проверить, что OFD $\mathcal{X} : [] \mapsto A$ не удерживается эквивалентно тому, чтобы проверить, что не удерживается функциональная зависимость $X \to A$. Функциональная зависимость $X \to A$ не удерживается, если есть пара кортежей, равные в X, но не равные в A. Построим *removal set* следующим образом. Будем итерироваться по множеству классов эквивалентности $\Pi_{\mathcal{X}}$. Для каждого класса $\mathcal{E}$ известно, что кортежи в нём равны в X. Вычислим самое распространенное значение среди этих кортежей по атрибуту A. Все остальные кортежи, которые имеют в A значение отличное от самого распространенного, составляют *removal set*. Понятно, что если эти кортежи убрать, то зависимость начнет удерживаться и что такой набор минимален, потому что если убрать не все кортежи, то любой оставшийся будет нарушать инвариант функциональной зависимости. Псевдокод алгоритма представлен на

листинге 1.

Algorithm 1 Алгоритм вычисления *removal set* для OFD $\mathcal{X} : [] \mapsto A$

Require: Table r, OFD $\mathcal{X} : [] \mapsto A$
Ensure: Removal set s
1: $s = \{\}$
2: **for** all $\mathcal{E} \in \Pi_{\mathcal{X}}$ **do**
3: $\quad t = \mathcal{E}$
4: $\quad L = \text{computeTuplesWithTheMostCommonValue}(t_A)$
5: $\quad s = s \cup (t \setminus L)$
6: **end for**
7: **return** s

Проверить, что ОС $\mathcal{X} : A \sim B$ не удерживается немного сложнее. Алгоритм, предложенный в работе [9], также заключается в том, чтобы вычислить наибольшее по включению множество кортежей, на котором ОС выполняется, а потом вычесть это множество из всего множества кортежей. Наибольшее по включению множество кортежей конкретного класса эквивалентности вычисляется следующим образом. Сначала отсортируем кортежи по возрастанию по атрибутам A и B, используя лексикографический порядок (аналогично выражению `ORDER BY A, B` в SQL). Из полученного упорядоченного списка кортежей возьмём значения по атрибуту B. Далее вычислим наибольшую неубывающую подпоследовательность (*longest non-decreasing subsequence*) в последовательности данных значений. Она и будет являться искомым наибольшим по включению множеством, на котором удерживается заданная ОС в текущем классе $\mathcal{E}$. Псевдокод алгоритма представлен на листинге 1.

3.2 Поиск AOD с помощью FastOD

Алгоритм FastOD [8] состоит из нескольких основных шагов, они представлены на Рис. 1. Первый шаг парсинга данных получает на вход csv таблицу и строит по ней внутреннее представление таблицы, кодируя значения в таблице числами с сохранением порядка. Далее алгоритм генерирует кандидаты на AOD. Часть кандидатов отсекается с помощью правил отсечения, основанных на аксиомах set-based аксиома-

Algorithm 2 Алгоритм вычисления *removal set* для ОС $\mathcal{X} : A \sim B$

Require: Table r, ОС $\mathcal{X} : A \sim B$
Ensure: Removal set s

```
s = {}
for all 𝓔 ∈ Π_𝒳 do
    t = order 𝓔 by [A ASC, B ASC]
    L = computeLNDS(t_B)
    s = s ∪ (t_B \ L)
end for
return s
```

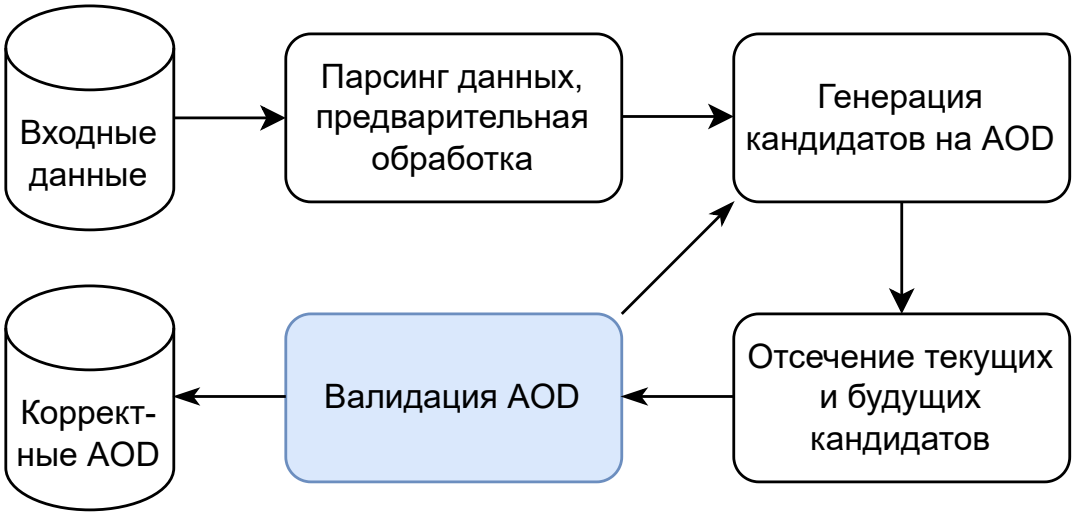

Рис. 1: Шаги алгоритма FastOD.

тизации. Неотсеченные кандидаты попадают в шаг валидации, который для каждого конкретного кандидата возвращает ответ: удерживается данная AOD на всей таблице или нет. Удерживающиеся зависимости сохраняются в результат алгоритм, неудерживающиеся передаются шагу генерации кандидатов, который с учетом новой информации генерирует следующих кандидатов. FastOD, ищущий точные зависимости, отличается от версии, ищущей приближенные, только шагом валидации (отмечен голубым на Рис. 1). Для реализации приближенного алгоритма FastOD, достаточно заменить шаг валидации, чтобы он возвращал положительный результат (зависимость удерживается) не только для точных зависимостей, но и для приближенных. Используем те же алгоритмы,

что использовали для верификации OFD и OC (на вход шагу валидации подается или OC, или OFD).

Важным замечанием здесь является то, что валидация приближенных зависимостей требует вычисление полного *removal set*. Для валидации точных зависимостей нам достаточно проверить, что есть хотя бы одно нарушение, что эквивалентно проверке на непустоту *removal set*. Поиск одного нарушения завершается гораздо быстрее, чем вычисление полного *removal set* (при условии, что хотя бы одно нарушение есть). Поэтому если пользователь указывает ошибку равной нулю, будем откатываться к имеющейся логики верификации точной зависимости, что значительно ускорит выполнение алгоритма.

Заключение

1. Выполнен обзор существующих подходов к поиску AOD, для реализации выбрана set-based аксиоматизация и алгоритм FastOD.
2. В существующую реализацию алгоритма FastOD в Desbordante (C++) добавлена поддержка поиска приближенных зависимостей порядка.
3. Реализован алгоритм верификации приближенных зависимостей порядка в set-based аксиоматизации, а именно:
   - Реализован алгоритм на языке C++ в ядре платформы Desbordante.
   - Реализация покрыта тестами.
   - Реализован интерфейс к алгоритму верификации для взаимодействия через Python.
4. Код реализации доступен в репозитории[2].

[2]https://github.com/Desbordante/desbordante-core/pull/468

Список литературы

[1] Abedjan Ziawasch, Golab Lukasz, Naumann Felix. Profiling Relational Data: A Survey // The VLDB Journal. — 2015. — aug. — Vol. 24, no. 4. — P. 557–581. — URL: https://doi.org/10.1007/s00778-015-0389-y.

[2] Aggarwal Charu C. Data Mining: The Textbook. — Springer Publishing Company, Incorporated, 2015. — ISBN: 3319141414.

[3] Approximate Order Dependency Discovery / Yifeng Jin, Zijing Tan, Weijun Zeng, Shuai Ma // 2021 IEEE 37th International Conference on Data Engineering (ICDE). — 2021. — P. 25–36.

[4] Caruccio Loredana, Deufemia Vincenzo, Polese Giuseppe. Relaxed Functional Dependencies—A Survey of Approaches // IEEE Transactions on Knowledge and Data Engineering. — 2016. — Vol. 28, no. 1. — P. 147–165.

[5] Data Dependencies Extended for Variety and Veracity: A Family Tree / Shaoxu Song, Fei Gao, Ruihong Huang, Chaokun Wang // IEEE Transactions on Knowledge and Data Engineering. — 2022. — Vol. 34, no. 10. — P. 4717–4736.

[6] Desbordante: a Framework for Exploring Limits of Dependency Discovery Algorithms / Maxim Strutovskiy, Nikita Bobrov, Kirill Smirnov, George A. Chernishev // 29th Conference of Open Innovations Association, FRUCT 2021, Tampere, Finland, May 12-14, 2021. — IEEE, 2021. — P. 344–354. — URL: https://doi.org/10.23919/FRUCT52173.2021.9435469.

[7] Chernishev George, Polyntsov Michael, Chizhov Anton et al. Desbordante: from benchmarking suite to high-performance science-intensive data profiler (preprint). — 2023. — 2301.05965.

[8] Effective and complete discovery of order dependencies via set-based axiomatization / Jaroslaw Szlichta, Parke Godfrey, Lukasz Golab et al. //

Proc. VLDB Endow. — 2017. — mar. — Vol. 10, no. 7. — P. 721–732. — URL: https://doi.org/10.14778/3067421.3067422.

[9] Karegar Reza, Godfrey Parke, Golab Lukasz et al. Efficient Discovery of Approximate Order Dependencies. — 2021. — 2101.02174.

[10] Expressiveness and complexity of order dependencies / Jaroslaw Szlichta, Parke Godfrey, Jarek Gryz, Calisto Zuzarte // Proc. VLDB Endow. — 2013. — sep. — Vol. 6, no. 14. — P. 1858–1869. — URL: https://doi.org/10.14778/2556549.2556568.

[11] FASTOD: Bringing Order to Data / Alexandar Mihaylov, Parke Godfrey, Lukasz Golab et al. // 2018 IEEE 34th International Conference on Data Engineering (ICDE). — 2018. — P. 1561–1564.

[12] 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.

[13] Ilyas Ihab F., Chu Xu. Data Cleaning. — New York, NY, USA : Association for Computing Machinery, 2019. — ISBN: 978-1-4503-7152-0.

[14] Langer Philipp, Naumann Felix. Efficient order dependency detection // The VLDB Journal. — 2016. — apr. — Vol. 25, no. 2. — P. 223–241. — URL: https://doi.org/10.1007/s00778-015-0412-3.

[15] Szlichta Jaroslaw, Godfrey Parke, Gryz Jarek. Fundamentals of order dependencies // Proc. VLDB Endow. — 2012. — jul. — Vol. 5, no. 11. — P. 1220–1231. — URL: https://doi.org/10.14778/2350229.2350241.