Санкт-Петербургский государственный университет

Кафедра информационно-аналитических систем

Струтовский Максим Андреевич

Реализация алгоритмов поиска функциональных зависимостей в базах данных

Курсовая работа

Научный руководитель:
ассистент Чернышев Г.А.

Санкт-Петербург
2021

Оглавление

Введение **3**

Постановка задачи **4**

1. Обзор **5**

1.1. Основные понятия 5

1.2. Классификация алгоритмов 6

1.2.1. Решётчатые алгоритмы 6

1.2.2. Построчные алгоритмы 7

1.2.3. Индуктивные алгоритмы 7

2. Реализация классических алгоритмов **9**

3. Распараллеливание Pyro **10**

4. Эксперименты **11**

4.1. Классические алгоритмы 12

4.2. Многопоточный Pyro 14

Заключение **15**

Благодарности **17**

Список литературы **18**

Введение

Зависимости в реляционных базах данных представляют собой правила в данных, применимые в широком спектре задач: например, при очистке данных [4], нормализации схемы [11] и исследовании данных [1]. В данной работе рассматривается отдельный подвид зависимостей — функциональные зависимости.

Поскольку функциональные зависимости можно выявить, обладая достаточными знаниями о предметной области, задачей их обнаружения может заниматься эксперт, однако в некоторых случаях, например, при отсутствии подобной информации или наличии только тела таблицы, используются алгоритмы автоматического поиска зависимостей. Сложность задачи для таблицы с m атрибутов и n кортежей оценивается в $\mathcal{O}(n^2m^2 2^m)$ операций, и было предложено множество алгоритмов [6] с уникальными подходами и эвристиками, призванными обеспечить оптимальную производительность на реальных данных.

Время работы каждого алгоритма сильно зависит от параметров таблицы и самих данных, поэтому получить понимание границ их применимости можно лишь обладая их реализациями в рамках одной высокопроизводительной платформы — с этой целью была разработана платформа Desbordante. Эксперименты показали, что реализованный на её основе алгоритм Pyro превосходит по скорости работы существовавшую реализацию Metanome [5]. Следующим шагом в развитии Desbordante стала реализация и сравнение остальных алгоритмов поиска функциональных зависимостей.

Постановка задачи

Целью данной работы является реализация и экспериментальное изучение алгоритмов поиска функциональных зависимостей на основе платформы Desbordante. Для достижения данной цели были поставлены следующие задачи:

1. Реализовать шесть классических алгоритмов поиска ФЗ и провести эксперимент по сравнению их быстродействия.

2. Распараллелить уже реализованный алгоритм Pyro.

1. Обзор

1.1. Основные понятия

Определение 1. Функциональная зависимость (ФЗ) определяет связь между атрибутами отношения [8]. Говорят, что между двумя множествами атрибутов X и Y в таблице данных имеет место функциональная зависимость $X \to Y$, если любые два кортежа, совпадающие на атрибутах X, совпадают на атрибутах Y. В этом случае, X называется левой частью функциональной зависимости, а Y — правой.

Определение 2. Если Y не зависит функционально ни от одного подмножества X, функциональную зависимость $X \to Y$ называют минимальной [8].

Поиск всех функциональных зависимостей в таблице сводится к нахождению минимальных зависимостей с правой частью, состоящей из одного атрибута, так как остальные функциональные зависимости можно вывести с помощью правил вывода Армстронга [3].

Недостаток такого определения функциональной зависимости заключается в том, что артефакты в реальных данных, такие как опечатки, отсутствующие значения и грязные данные, могут привести к ситуации, когда ФЗ не может быть выведена, хотя семантически она должна удерживаться. В качестве решения этой проблемы было сформулировано понятие приближённой функциональной зависимости, где под “приближённостью” подразумевается то, что небольшая часть кортежей может различаться на правой части при совпадении на левой. Таким образом, алгоритмы поиска приближённых ФЗ параметризуются значением максимальной погрешности e_{max}, служащим формальным критерием того, что не совпадает только “небольшая” часть кортежей, и зависимость-кандидат можно записать как приближённую ФЗ ($e(X \to Y) \leq e_{max}$). Стоит отметить, что приближённые ФЗ являются обобщением точных ФЗ: действительно, если взять $e_{max} = 0$, ни одна пара совпадающих на левой части кортежей не может различаться на правой части.

Определение 3. Пусть r — отношение, и $X \to Y$ —потенциальная приближённая ФЗ. Тогда погрешность вычисляется как [9]:

$$e(X \to Y, r) = \frac{|\{(t_1, t_2) \in r^2 | t_1[X] = t_2[X] \wedge t_1[Y] \neq t_2[Y]\}|}{|r|^2 - |r|}$$

Приближённая ФЗ $X \to Y$ удерживается на r, если $e(X \to Y, r) \leq e_{max}$.

1.2. Классификация алгоритмов

В литературе представлена масса алгоритмов поиска ФЗ, однако в качестве наиболее эффективных и изученных принято выделять следующие: Tane [12], FD_Mine [14], Dfd [2], Dep-Miner [10], FastFDs [13] и FDep [7]. Все алгоритмы так или иначе решают задачу поиска ФЗ перебором всевозможных кандидатов, однако в их основе лежат разные подходы к генерации и проверке кандидатов. При описании алгоритмов обычно используется следующая классификация, основанная на особенностях этих подходов.

1.2.1. Решётчатые алгоритмы

Представители данного класса (Tane, FD_Mine, Dfd) ставят перед собой задачу перебрать ФЗ со всеми возможными комбинациями атрибутов. Пространство перебора моделируется как многоуровневый ориентированный граф, где вершина представляет собой набор колонок, а ребро между вершинами XY и YZ соответствует зависимости $X \to Z$. Вершины объединяются в уровни по количеству колонок и, поскольку разыскиваются только ФЗ с одноколоночной правой частью, ребра проводятся только между соседними уровнями. Алгоритмы Tane, FD_Mine рассматривают уровни последовательно, начиная с нулевого, проверяя всевозможные ФЗ, представленные рёбрами, исходящими из вершин рассматриваемого уровня, тогда как Dfd использует обход графа в глубину. Валидация кандидатов происходит через пересечение партиций. Не каждый кандидат проходит трудоёмкий процесс валидации: имея набор уже обнаруженных ФЗ, можно заранее исключить

из рассмотрения некоторые кандидаты, следуя определённым правилам, выведенным из критериев минимальности ФЗ — базовый набор правил был предложен авторами TANE, тогда как авторы FD_Mine и Dfd расширили его. Так как размер графа, а, значит, и потенциальное количество посещённых вершин, зависит экспоненциально от количества атрибутов, алгоритмы данного типа обладают лучшей производительностью на "длинных" таблицах, то есть на отношениях с большим числом кортежей.

1.2.2. Построчные алгоритмы

DepMiner и FastFDs отталкиваются от такой структуры, как набор согласованности. Набором согласованности нескольких кортежей является набор атрибутов, по которым эти кортежи равны. Интуитивно, набор согласованности содержит информацию о том, какие ФЗ могут удерживаться на отношений, а какие явно не удерживаются: например, ФЗ с левой частью, входящей в набор согласованности, и не входящей в него правой однозначно нарушаются на рассматриваемом отношении. Оба алгоритма начинают работу с построения наборов согласованности для всех пар кортежей, после чего, применив к построенному множеству ряд преобразований, выводят набор минимальных ФЗ. Поскольку построение наборов согласованности зависит квадратично от количества кортежей и линейно от количества атрибутов, принято считать, что DepMiner и FastFDs достигают лучших результатов на "широких" таблицах, то есть на отношениях с большим числом атрибутов.

1.2.3. Индуктивные алгоритмы

Алгоритмы этого класса, представленные алгоритмом FDep, также используют попарное сравнение строк, но имеют отличную идею вывода искомого множества ФЗ. Алгоритм начинает с набора наиболее общих ФЗ, после чего, сравнивая строки попарно, заменяет нарушающиеся ФЗ более специализированными, то есть, например, если в таблице из трёх атрибутов ABC найдётся два кортежа, различающихся

на A, ФЗ $\emptyset \rightarrow A$ будет заменена на $B \rightarrow A$, $C \rightarrow A$. Это единственный класс алгоритмов, не использующий пересечение партиций. FDep обладает схожими качествами с построчными алгоритмами: поскольку необходимо сравнить все пары кортежей, FDep масштабируется лучше при увеличении числа атрибутов, чем числа кортежей.

2. Реализация классических алгоритмов

Desbordante — открытая платформа для высокопроизводительного поиска функциональных зависимостей в реляционных базах данных, написанная на языке C++17 с активным использованием библиотек boost и GoogleTest и применением системы сборки CMake, находящаяся в открытом доступе на GitHub[1]. В рамках данной работы было решено дополнить набор алгоритмов, уже реализованных на платформе Desbordante, алгоритмами FDep, DFD, DepMiner, Fd_Mine, FastFDs. Для достижения этой цели был налажен процесс командной разработки, заключающийся в том, что изучение, реализация и отладка каждого отдельного алгоритма выполнялись в рамках работы студента младшего курса, тогда как перед автором данной работы стояли следующие задачи:

- осуществление коммуникации между членами команды;
- инспектирование кода;
- решение архитектурных вопросов, таких как: определение и вынесение общего функционала в единый интерфейс, отслеживание общих структур данных и оценка возможности их переиспользования;
- реализация автоматической тестовой системы для единообразной валидации и измерения производительности алгоритмов;
- исправление программных ошибок, относящихся к общим компонентам.

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

3. Распараллеливание Pyro

Помимо добавления к функциональности Desbordante классических алгоритмов велась работа по улучшению уже реализованного Pyro. Будучи современным алгоритмом поиска приближённых ФЗ, являющихся обобщением ФЗ и обладающих более широким спектром применимости, Pyro является наиболее востребованным и, зачастую, эффективным, поэтому его ускорение является приоритетной задачей [9]. Один из способов ускорения работы программы за счёт использования имеющихся аппаратных мощностей — распараллеливание, а принцип работы Pyro позволяет распределить исполняемые инструкции по нескольким потокам довольно естественным образом и получить ощутимое ускорение, не прибегая к на порядок более сложным структурам данных и контролю за параллельным доступом к общим данным.

В Pyro для каждого атрибута строится собственное, имеющее свои структуры данных пространство перебора, в котором ведётся дальнейший перебор кандидатов. Исследование поисковых пространств осуществляется практически независимо: из общих структур данных запись происходит только в кеш партиций. Таким образом, Pyro был распараллелен следующим образом:

1. Формируется пул потоков фиксированного размера, задаваемого пользователем, либо определяемого количеством доступных процессоров в системе, после чего ему отдаётся на выполнение набор задач, каждая из которых состоит в исследовании поискового пространства, соответствующего одному из атрибутов.

2. Структура данных, представляющая кеш партиций, модифицирована для корректного параллельного доступа посредством блокировки потока при одновременном доступе. Реализация неблокирующей структуры данных была сочтена излишней, поскольку исследование поискового пространства и пересечение партиций являются гораздо более трудоёмкими операциями, чем чтение и запись партиций из кеша.

4. Эксперименты

Для оценки быстродействия алгоритмов и сравнения разных алгоритмов в рамках одной платформы, а также разных реализаций одного алгоритма, были проведены эксперименты по общей схеме. Для каждого алгоритма обе его реализации были запущены на отобранном наборе отношений. Если платформа требовала при исполнении алгоритма более 8GB оперативной памяти или более 12 минут, результатом эксперимента считался Memory Limit и Time Limit, соответственно. Достижение одного из пределов отслеживалось при помощи утилит Linux `ulimit` и `timeout`. В случае успешного завершения работы алгоритма он был запущен ещё девять раз, и в таблицу результатов записывался доверительный интервал, построенный по десяти запускам, с уровнем доверия 0.95.

Таблица	Атрибуты	Строки	Размер, КБ	#ФЗ
iris	5	150	5	4
balance-scale	5	625	7	1
chess	7	28,056	519	1
abalone	9	4,177	187	137
nursery	9	12,960	1,024	1
breast-cancer	11	699	20	46
bridges	13	108	6	142
echocardiogram	13	132	6	538
adult	14	48,842	3,528	78
letter	17	20,000	695	61
ncvoter	19	1,000	151	758
hepatitis	20	155	8	8,250
horse	27	368	25	128,726
fd-reduced-30	30	250,000	69,581	89,571
plista	63	1,000	568	178,152
flight	109	1,000	575	982,631
uniprot	223	1,000	2,439	unknown

Таблица 1: Обзор отобранных таблиц

В качестве датасета для проведения экспериментов был выбран набор из [8], характеристики отношений из этого набора представлены в таблице 1. В данной работе было проведено экспериментальное сравнение семи алгоритмов поиска ФЗ на Metanome. Реализация классических алгоритмов в рамках текущей работы позволяет воспроизвести эксперимент, пользуясь Desbordante — платформой, написанной на другом языке программирования и нацеленной на достижение пределов производительности алгоритмов.

В качестве платформы для исполнения кода был использован настольный ПК с операционной системой Pop!_OS 21.04 64-bit, компилятором C++ gcc 10.2.0, виртуальной машиной OpenJDK 64-bit Server VM 11.0.9.1 и комплектующими: Intel(R) Core(TM) i5-7600K CPU (4 ядра) @ 3.80GHz, 16GB DDR4 2133MHz RAM, 2TB HDD WD20EZAZ. Алгоритмы, реализованные на Metanome, версии 1.2, были взяты из официального репозитория[2], а для проведения экспериментов был использован Skeleton версии 1.2 — легковесная среда, имитирующая Metanome[3]. Наконец, при исполнении обеих платформ была обеспечена минимально возможная дополнительная нагрузка операционной системы, то есть не исполнялись другие тяжеловесные процессы, а тактовая частота ядер процессора была зафиксирована утилитой `cpufreq-set` на уровне 4.0GHz.

4.1. Классические алгоритмы

Результаты запусков реализаций алгоритмов на обеих платформах занесены в таблицу 2. Помимо алгоритмов Tane, FD_Mine, Dfd, DepMiner, FastFDs и FDep, в эксперимент был включён и алгоритм Pyro с двумя разными значениями максимально допустимой погрешности ФЗ: 0 и 0.01.

При сравнении производительности алгоритмов на Desbordante прослеживаются практически те же закономерности, что и в работе [8]. На

[2]https://github.com/HPI-Information-Systems/metanome-algorithms

[3]https://hpi.de/naumann/projects/data-profiling-and-analytics/metanome-data-profiling/algorithms.html

Таблица	Платформа	depminer	dfd	fastfds	fdep	fdmine	tane	pyro_0	pyro_0.01
BALANCE	D	24 ± 0	0 ± 0	24 ± 0	9 ± 0	1 ± 0	0 ± 0	2 ± 0	2 ± 0
	M	61 ± 0	TL	72 ± 0	31 ± 0	35 ± 0	38 ± 0	47 ± 1	46 ± 0
BREAST	D	114 ± 0	18 ± 0	110 ± 0	36 ± 0	205 ± 0	29 ± 0	56 ± 0	47 ± 1
	M	346 ± 0	TL	373 ± 0	62 ± 0	850 ± 9	185 ± 3	192 ± 4	193 ± 6
BRIDGES	D	131 ± 0	35 ± 0	7 ± 0	2 ± 0	355 ± 0	17 ± 0	61 ± 0	55 ± 1
	M	46 ± 0	TL	99 ± 0	17 ± 0	1975 ± 48	111 ± 1	191 ± 1	192 ± 2
IRIS	D	0 ± 0	0 ± 0	0 ± 0	0 ± 0	0 ± 0	0 ± 0	1 ± 0	1 ± 0
	M	13 ± 0	TL	25 ± 0	7 ± 0	18 ± 0	31 ± 1	39 ± 0	39 ± 0
ABALONE	D	851 ± 1	26 ± 0	842 ± 1	672 ± 1	420 ± 0	17 ± 0	62 ± 0	57 ± 0
	M	1224 ± 29	TL	1286 ± 27	730 ± 20	651 ± 6	159 ± 5	102 ± 1	100 ± 1
CHESS	D	52023 ± 27	52 ± 1	51890 ± 18	26743 ± 94	205 ± 0	99 ± 0	117 ± 0	127 ± 1
	M	108458 ± 384	621 ± 29	110494 ± 431	28309 ± 1353	1798 ± 20	497 ± 24	253 ± 19	239 ± 14
NURSERY	D	26965 ± 18	40 ± 0	27068 ± 56	9546 ± 19	286 ± 0	144 ± 0	69 ± 0	73 ± 0
	M	65439 ± 619	570 ± 25	68605 ± 945	10583 ± 748	3470 ± 20	663 ± 29	414 ± 21	425 ± 12
ECHODIAGRAM	D	83 ± 0	104 ± 0	6 ± 0	2 ± 0	14331 ± 21	6 ± 0	136 ± 0	131 ± 0
	M	38 ± 0	TL	61 ± 0	20 ± 0	49907 ± 606	73 ± 1	132 ± 1	133 ± 1
ADULT	D	544456 ± 648	553 ± 10	537845 ± 294	171407 ± 358	49349 ± 51	17730 ± 21	417 ± 0	353 ± 12
	M	TL	TL	TL	163474 ± 2390	ML	TL	6334 ± 135	6368 ± 83
LETTER	D	145600 ± 205	1137 ± 43	86928 ± 62	52190 ± 70	ML	79026 ± 25	1438 ± 0	1374 ± 13
	M	451857 ± 18291	TL	436696 ± 2897	58767 ± 1655	TL	ML	3027 ± 22	3072 ± 32
NCVOTER_1K	D	29136 ± 6	583 ± 5	109 ± 0	237 ± 0	TL	357 ± 0	479 ± 0	399 ± 15
	M	532 ± 56	TL	720 ± 3	289 ± 2	ML	750 ± 51	388 ± 15	365 ± 9
HEPATITIS	D	TL	19266 ± 169	855 ± 0	52 ± 0	TL	3832 ± 3	11050 ± 6	6781 ± 794
	M	369 ± 3	TL	4633 ± 10	141 ± 1	TL	4020 ± 36	6010 ± 64	6013 ± 75
HORSE	D	TL	TL	48321 ± 150	2375 ± 2	TL	ML	206315 ± 168	126453 ± 14876
	M	9779 ± 75	TL	237247 ± 1591	3731 ± 18	TL	ML	81456 ± 1564	81120 ± 2622
FD_REDUCED	D	TL	491814 ± 2225	TL	TL	TL	8067 ± 6	32810 ± 27	29679 ± 599
	M	20835 ± 90	TL	TL	108138 ± 281	TL	TL	TL	TL
PLISTA	D	TL	TL	698129 ± 304	11694 ± 14	ML	ML	TL	TL
	M	ML	TL	ML	ML	ML	ML	TL	TL
FLIGHT	D	ML	TL	104063 ± 52	780 ± 10	ML	ML	TL	TL
	M	20847 ± 78	TL	TL	108285 ± 368	ML	TL	TL	TL
UNIPROT_1K	D	TL	TL	TL	ML	ML	ML	TL	TL
	M	TL	TL	TL	ML	TL	TL	TL	TL

Таблица 2: Измерение времени работы алгоритмов Desbordante(D) и Metanome(M), мс. ML, TL означают выход за пределы по времени и памяти, соответственно.

большинстве таблиц лучшие результаты показывает FDep — впрочем, некоторые таблицы, такие как chess, adult и letter, обладают большим количеством кортежей, и его производительность заметно ухудшается, значительно уступая Dfd и Tane. Стоит отметить, что Pyro редко оказывается лидирующим по производительности алгоритмом, хоть и показывая приемлемые результаты на подавляющей части таблиц. Это можно объяснить тем, что поиск приближённых ФЗ является более трудоёмкой задачей даже для современного алгоритма, причём при поиске точных ФЗ, то есть при пороге погрешности 0, Pyro демонстрирует те же результаты, что и при пороге 0.01.

При попарном сравнении алгоритмов, реализованных в Metanome и Desbordante, можно выделить следующие результаты:

- Алгоритмы Tane, FD_Mine и FastFDs на Desbordante показывают меньшее время исполнения на всех таблицах: среднее ускорение

достигает 5.0, 5.9 и 3.8 раза, соответственно.

- В алгоритме Dfd на Metanome была обнаружена программная ошибка, приводящая к бесконечному зацикливанию на всех таблицах, кроме chess, echodiagram и nursery.

- Алгоритм Fdep показывает неоднозначные результаты: на одной части таблиц Metanome значительно уступает Desbordante, на другой наблюдаются обратные результаты. Таким образом, обе реализации FDep требуют дальнейшего анализа и отладки.

4.2. Многопоточный Pyro

На Рис. 1 нанесены доверительные интервалы, полученные в результате исполнения Pyro на Metanome с четырьмя рабочими потоками и на Desbordante — с четырьмя и одним. По графику можно сделать следующие выводы:

- Многопоточной версии Pyro на Desbordante требуется меньшее время для успешного завершения работы, по сравнению с версией Metanome — в среднем в 4.27 раза. Исключением является датасет Horse — на нём Desbordante работает ощутимо медленнее. Впрочем, это качество было изначально присуще реализации Desbordante, ещё до распараллеливания, следовательно, выяснение причин остаётся вне рамок данного эксперимента.

- Распараллеливание, что вполне ожидаемо, дало прирост производительности лишь на больших датасетах — требующих более 200мс. для обработки одним потоком — это связано с дополнительными затратами на создание потоков, синхронизацию и переключение контекста, окупающимися лишь при достаточно длительной работе. Прирост в среднем составил 2.0 раза.

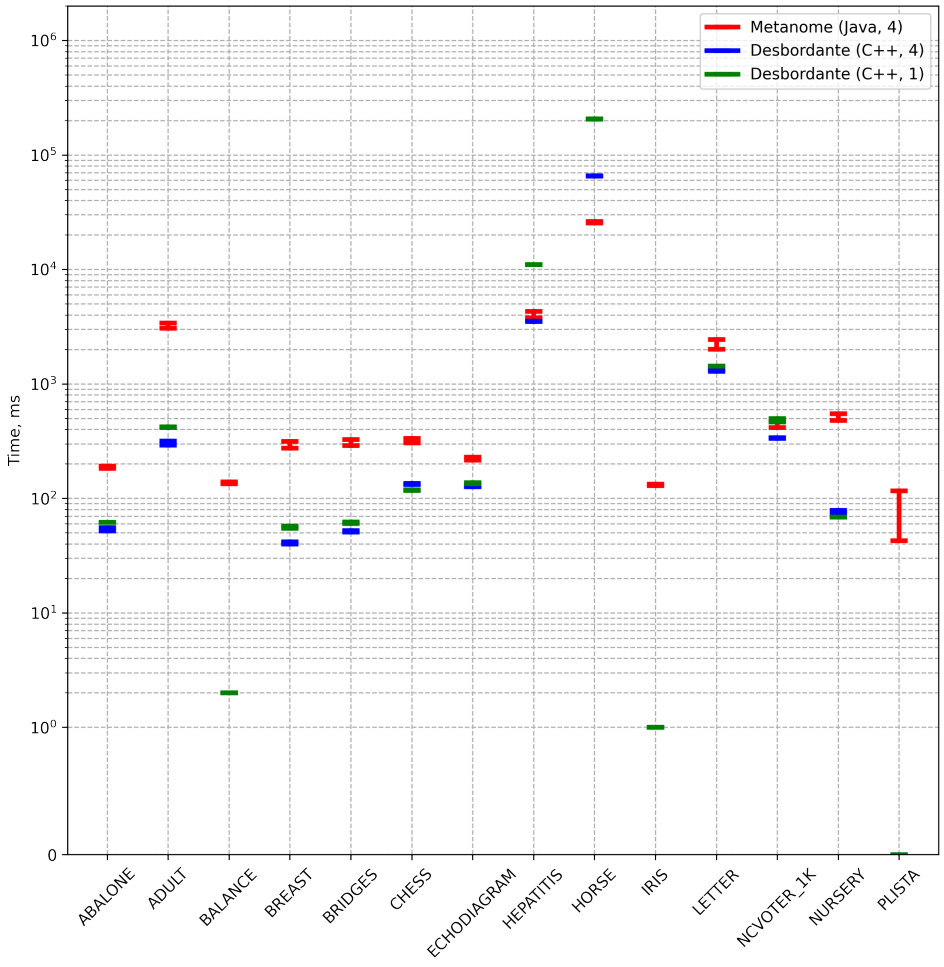

Рис. 1: Измерение производительности распараллеленного Pyro

Заключение

В рамках данной работы были достигнуты следующие результаты:

1. Реализованы и экспериментально исследованы шесть классических алгоритмов поиска ФЗ.

   - выполнен обзор и рассмотрены принципы работы алгоритмов поиска функциональных зависимостей;
   - в результате командной разработки алгоритмы были реализованы на основе платформы Desbordante;

- проведены эксперименты, подтвердившие выдвинутые в литературе закономерности между параметрами таблиц и быстродействием алгоритмов, а также показавшие более высокую производительность реализованных алгоритмов по сравнению с алгоритмами, реализованными на Metanome.

2. Распараллелен алгоритм Pyro.

   - описан и реализован механизм распределения работы алгоритма по нескольким потокам;
   - проведён эксперимент, показавший, что распараллевание ощутимо уменьшает время работы алгоритма на достаточно объёмных датасетах.

Благодарности

Автор выражает отдельную признательность коллегам, занимавшимся непосредственной реализацией алгоритмов:

- Эдуарду Гайсину — за реализацию Dep-Miner;
- Михаилу Полынцову — за реализацию FastFDs и активное участие в улучшении Desbordante;
- Артуру Салью — за реализацию FDep;
- Александру Смирнову — за реализацию Dfd и активное участие в улучшении Desbordante;
- Илье Щукину — за реализацию FD_Mine.

Список литературы

[1] Abedjan Ziawasch, Golab Lukasz, and Naumann Felix. Profiling relational data: a survey // The VLDB Journal. — 2015. — Vol. 24. — P. 557–581.

[2] Abedjan Ziawasch, Schulze Patrick, and Naumann Felix. DFD: Efficient Functional Dependency Discovery // Proceedings of the 23rd ACM International Conference on Conference on Information and Knowledge Management. — New York, NY, USA : Association for Computing Machinery. — 2014. — CIKM '14. — P. 949–958. — Access mode: https://doi.org/10.1145/2661829.2661884.

[3] Armstrong W. W. Dependency Structures of Data Base Relationships // IFIP Congress. — 1974.

[4] Khayyat Zuhair, Ilyas Ihab, Jindal Alekh, Madden Samuel, Ouzzani Mourad, Papotti Paolo, Quian'E-ruiz Jorge-Arnulfo, Tang Nan, and Yin Si. BigDansing: A System for Big Data Cleansing. — 2015. — 06.

[5] Strutovskiy Maxim, Bobrov Nikita, Smirnov Kirill, and Chernishev George. Desbordante: a Framework for Exploring Limits of Dependency Discovery Algorithms (paper accepted) // FRUCT'29: Proceedings of the 29th Conference of Open Innovations Association FRUCT. — fruct.org. — 2021.

[6] Liu Jixue, Li Jiuyong, Liu Chengfei, and Chen Yongfeng. Discover Dependencies from Data—A Review // Knowledge and Data Engineering, IEEE Transactions on. — 2012. — 03. — Vol. 24. — P. 251 – 264.

[7] Flach Peter A. and Savnik Iztok. Database Dependency Discovery: A Machine Learning Approach // AI Commun. — 1999. — Aug. — Vol. 12, no. 3. — P. 139–160.

[8] Papenbrock Thorsten, Ehrlich J., Marten J., Neubert T., Rudolph J.-P, Schönberg M., Zwiener J., and Naumann Felix. Functional dependency discovery: An experimental evaluation of seven algorithms. — 2015. — 01. — P. 1082–1093.

[9] Kruse Sebastian and Naumann Felix. Efficient Discovery of Approximate Dependencies // Proceedings of the VLDB Endowment. — 2018. — 03. — Vol. 11.

[10] Lopes Stéphane, Petit Jean-Marc, and Lakhal Lotfi. Efficient Discovery of Functional Dependencies and Armstrong Relations. — 2000. — 03. — Vol. 1777. — P. 350–364.

[11] Papenbrock Thorsten and Naumann Felix. Data-driven Schema Normalization // EDBT. — 2017.

[12] Huhtala Ykä, Kärkkäinen Juha, Porkka Pasi, and Toivonen Hannu. TANE: An Efficient Algorithm for Discovering Functional and Approximate Dependencies. // Comput. J. — 1999. — 02. — Vol. 42. — P. 100–111.

[13] Wyss Catharine, Giannella Chris, and Robertson Edward. FastFDs: A Heuristic-Driven, Depth-First Algorithm for Mining Functional Dependencies from Relation Instances Extended Abstract. — 2001. — 01. — P. 101–110.

[14] Yao Hong, Hamilton Howard J., and Butz Cory J. FD_Mine: Discovering Functional Dependencies in a Database Using Equivalences // Proceedings of the 2002 IEEE International Conference on Data Mining. — USA : IEEE Computer Society. — 2002. — ICDM '02. — P. 729.