Санкт-Петербургский государственный университет

Кафедра информационно-аналитических систем

Струтовский Максим Андреевич

Реализация алгоритмов поиска функциональных зависимостей на языке программирования C++

Курсовая работа

Научный руководитель:
ассистент кафедры ИАС Чернышев Г.А.

Санкт-Петербург
2020

Оглавление

1. Введение

Функциональные зависимости — полезный инструмент при работе с базами данных и анализе данных. Данная работа посвящена алгоритмам поиска функциональных зависимостей.

Алгоритмы поиска функциональных зависимостей обладают высокой стоимостью как в требуемой памяти, так и в затрачиваемом времени. Например, в таблице с 12 атрибутами всего 24564 возможных зависимостей, а количество кортежей может превосходить $1*10^6$, вследствие чего, чтобы найти все функциональные зависимости по определению, необходимо совершить порядка 10^{16} операций. Таким образом, становится актуальной задача реализации алгоритмов поиска зависимостей на эффективном языке программирования, позволяющем минимизировать время и память, требуемые алгоритмами, а также позволяющих точное измерение этих показателей. Существующее решение (Metanome) не решает эту задачу в полной мере из-за следующих недостатков языка Java:

- улучшение изначально не самой высокой производительности программы, написанной на Java, требует тонкой настройки виртуальной машины Java, выбора оптимального алгоритма сборки мусора и прочих инструментов [6];
- используемые при работе кода, написанного на Java, технологии наподобие динамической компиляции и сборки мусора вносят неопределённость в скорость работы программы, что усложняет задачу оценки производительности ё[5].

1.1. Постановка задачи

Целью данной работы является исследование производительности алгоритмов поиска функциональных зависимостей на языке C++. Для достижения этой цели в данной работе были сформулированы следующие задачи:

- обзор предметной области, приложений и технологий поиска функциональных зависимостей;
- обзор существующих инструментов для работы с Функциональными зависимостями;
- реализация алгоритма TANE на языке программирования C++;
- сравнение производительности имплементации на C++ с существующим решением, написанным на Java.

2. Функциональные зависимости

2.1. Основные понятия

Определение 1. Функциональная зависимость определяет связь между атрибутами отношения. 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.

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.

Знания о наличии функциональных зависимостей в таблице используются в следующих областях:

- разбиение таблицы;
- нормализация базы данных;
- очистка данных [1];
- оптимизации запросов [9];
- анализ данных;
- восстановление данных [11].

Таким образом, функциональные зависимости — важный инструмент при работе с большим количеством данных. Один из способов их поиска — привлечение эксперта в предметной области, знакомого с закономерностями и правилами, однако, если это невозможно, прибегают

ко второму способу — использованию алгоритмов для обнаружения зависимостей [4]. Такие алгоритмы в основном используют одну из трёх стратегий.

1. *Исследование частично упорядоченного множества возможных зависимостей.* Алгоритмы этого типа представляют множество всех возможных функциональных зависимостей в виде решётки, которую исследуют, отбрасывая неминимальные зависимости и исключая её части, исследование которых не приведёт к обнаружению новых зависимостей.

2. *Обработка множеств согласованности и разности кортежей.* Алгоритм попарно сравнивает кортежи, формируя пространство для поиска, после чего генерирует удерживающиеся зависимости.

3. *Индукция.* Алгоритм начинает со множества наиболее общих зависимостей (каждый атрибут определяет все остальные), затем, попарно сравнивая кортежи, уточняет это множество.

В связи с необходимостью получения знаний о функциональных зависимостях при анализе данных, существует множество алгоритмов и некоторое количество поддерживающих их платформ, предназначенных для их обнаружения. В данной работе будет рассмотрен один подобный проект, занимающий лидирующие позиции по современности и количеству реализованных алгоритмов.

2.2. Metanome

Metanome — открытая платформа, на уровне бэкэнда реализованная на языке программирования Java, основной целью которой является анализ метаданных [2]. Подобная информация собирается инструментом с помощью множества алгоритмов поиска функциональных зависимостей, зависимостей включения, порядковых зависимостей и ключей, а далее анализируется различными методами визуализации и позволяет пользователю сделать выводы о хранимых данных.

3. Реализация алгоритма TANE

3.1. Описание алгоритма

TANE — алгоритм поиска функциональных зависимостей, в основе которого лежит исследование решётки, являющейся частично упорядоченным множеством возможных зависимостей [10]. Решётка является графом, чьи вершины представляют собой все возможные множества атрибутов рассматриваемой таблицы, причём каждая вершина с множеством атрибутов X соединена с вершинами $X \cup A$, где A — атрибут, отсутствующий в X. Таким образом, решётку можно рассматривать как набор уровней 0:N, где N — количество атрибутов в таблице, где между собой соединены только вершины соседних уровней, и каждое ребро, соединяющее X и $X \cup A$, обозначает функциональную зависимость вида $X \to A$.

Основная идея TANE — последовательный анализ уровней, проверка функциональных зависимостей между i-м и $i + 1$-м уровнем и исключение вершин решётки из рассмотрения, что позволяет значительно уменьшить количество посещаемых вершин. Например, на рисунке 1 представлена ситуация, в которой была найдена зависимость $\emptyset \to B$, то есть B является ключом, поэтому очевидно, что любое множество атрибутов, содержащее B, функционально определяет любой атрибут, соответственно, нет смысла посещать вершины, содержащие B. Наконец, такая концепция обеспечивает нахождение минимальных зависимостей с одним атрибутом в правой части, так как в первую очередь проверяются вершины с меньшим количеством атрибутов.

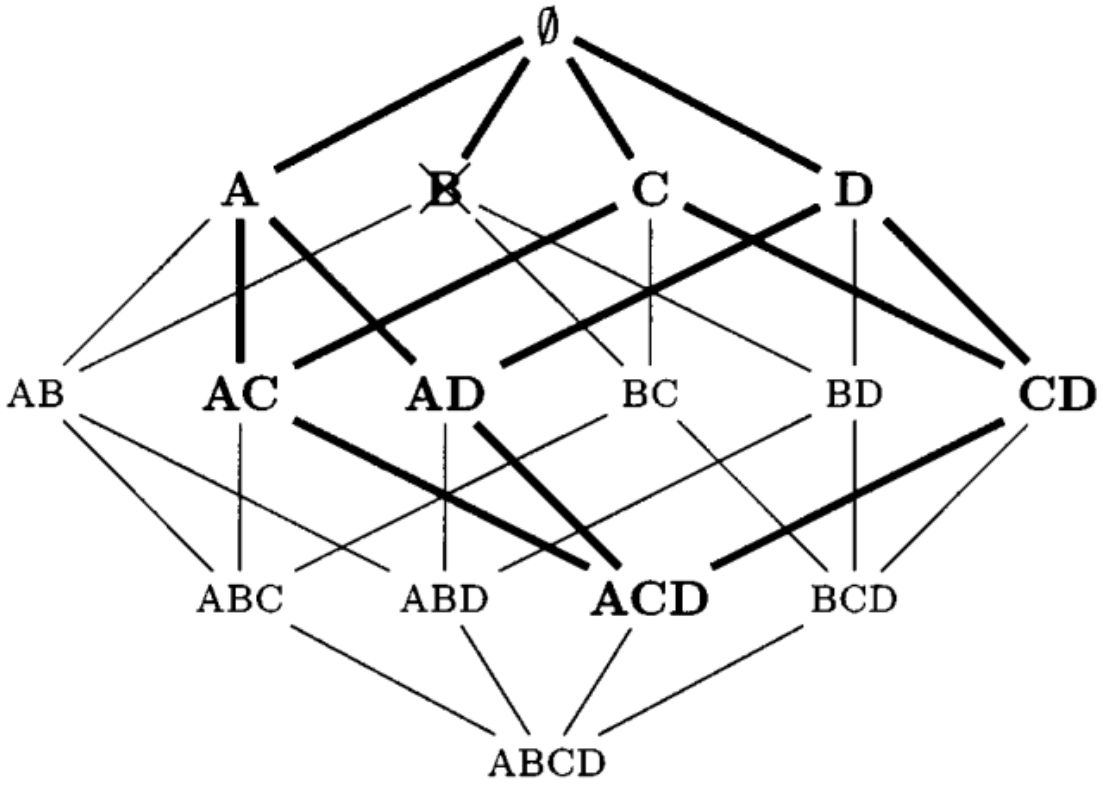

Рис. 1: Алгоритм посетит только выделенные вершины.

Реализация алгоритма на языке C++ опирается на описание, данное в соответствующей работе, и реализацию, используемую в Metanome. Сам алгоритм представлен тремя классами: `Tane`, `LatticeLevel`, `LatticeVertex`. Tane реализует основную логику алгоритма, `LatticeLevel` — уровень решётки, `LatticeVertex` — её вершина, набор атрибутов которой представлен через `boost::dynamic_bitset<>`, как и её прямые надмножества. Для доступа к её прямым подмножествам хранится вектор указателей на соответствующие вершины. Далее, два булевских поля дают информацию о необходимости посещения данной вершины. Наконец, хранится указатель на усечённую партицию — набор классов эквивалентности строк (две строки эквивалентны, если они равны на данном множестве атрибутов), реализованный как `std::vector<std::vector<int> >`. `LatticeLevel` состоит из `std::map<boost::dynamic_bitset, *LatticeVertex>`.

3.2. Эксперименты

Для тестирования производительности были выбраны 9 наборов данных, и на каждом из них по 10 раз подряд были запущены две версии TANE: написанная на языке Java, и на языке C++, после чего выведено среднее значение времени, прошедшего с окончания считывания программой таблицы до завершения работы алгоритма и относительное среднеквадратическое отклонение этого значения. Эксперименты проводились на системе Ubuntu 18.04.3 LTS, 64-bit, Intel Core i7-4700HQ 2.4GHz * 8, 8 GiB ОЗУ.

Таблица 1: Измерение производительности TANE

Набор данных	Размерность	C++, мс	Java, мс	Ускорение
neighbors100k	7, 10^5	$75.4 \pm 5\%$	$386.4 \pm 35\%$	5.1
neighbors50k	7, 0.5×10^5	$12.6 \pm 11\%$	$101.1 \pm 7\%$	8
LegacyPayors	4, 10^6	$162.7 \pm 3\%$	$1651 \pm 7\%$	10.2
FamHx	4, 10^5	$53.5 \pm 10\%$	$194.6 \pm 16\%$	3.6
PatientDemographics	18, 0.4×10^5	$16774.1 \pm 2\%$	$27635 \pm 11\%$	1.6
CIPublicHighway100k	18, 10^5	$2062.4 \pm 1\%$	$5315 \pm 3\%$	2.6
LegacyOPMeds	7, 0.7×10^6	$878.1 \pm 1\%$	$2040 \pm 2\%$	2.3
LegacyIPMeds	8, 0.6×10^6	$1520 \pm 1\%$	$9079 \pm 1\%$	6
EpicVitals	7, 10^6	$6231.6 \pm 1\%$	$11567.3 \pm 5\%$	1.9

В таблице 1 показаны результаты описанного эксперимента. Под размерностью набора данных имеется в виду совокупность двух показателей: количество атрибутов в таблице и количество кортежей, соответственно. Далее приведено среднее время работы алгоритма, написанного на конкретном языке, в миллисекундах, с относительным среднеквадратическое отклонением. Наконец, в столбце "Ускорение" выведено время работы имплементации на Java относительно C++.

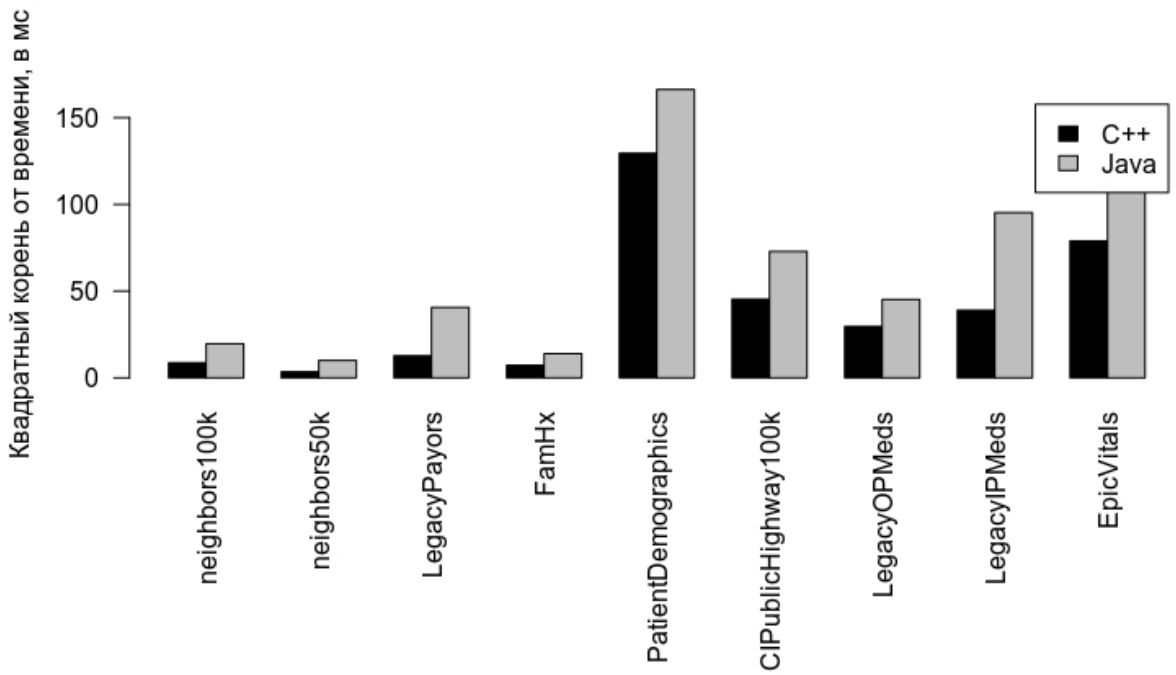

Рис. 2: Графическое представление производительности двух реализаций на приведённых наборах данных

По результатам экспериментов можно заключить, что реализация на C++ превосходит реализацию на Java по относительной дисперсии практически на всех наборах, а по среднему времени выполнения на объёмных наборах данных — почти в два раза, причём на меньших этот разрыв увеличивается.

4. Заключение

В ходе данной работы были достигнуты следующие результаты:

- проведён краткий обзор предметной области;
- выделены слабые стороны существующих решений;
- разобран и реализован на языке C++ алгоритм TANE;
- проведены эксперименты, выявившие более эффективную и стабильную работу реализации на C++ на выборке таблиц относительно реализации на Java.

4.1. Направления продолжения работы

- реализация более современных алгоритмов поиска функциональных зависимостей [7, 8];
- сравнение требуемого различными реализациями объёма памяти;
- сравнение с другими существующими решениями, например, FDTool [3];
- переписывание часто вызываемых методов с помощью AVX инструкций.

Список литературы

[1] BigDansing: A System for Big Data Cleansing / Zuhair Khayyat, Ihab Ilyas, Alekh Jindal et al. — 2015. — 06.

[2] Data profiling with metanome / Thorsten Papenbrock, Tanja Bergmann, Moritz Finke et al. // Proceedings of the VLDB Endowment. — 2015. — 08. — Vol. 8. — P. 1860–1863.

[3] FDTool: a Python application to mine for functional dependencies and candidate keys in tabular data / Matt Buranosky, Elmar Stellnberger, Emily Pfaff et al. // F1000Research. — 2019. — 06. — Vol. 7. — P. 1667.

[4] Functional dependency discovery: An experimental evaluation of seven algorithms / Thorsten Papenbrock, J. Ehrlich, J. Marten et al. — 2015. — 01. — P. 1082–1093.

[5] Georges Andy, Buytaert Dries, Eeckhout Lieven. Statistically Rigorous Java Performance Evaluation. — Vol. 42. — 2007. — 10.

[6] Group JUG Ru. Диалоги о Java Performance // Коллективный блог Хабр. — 2016. — URL: `https://habr.com/en/company/jugru/blog/280420/` (дата обращения: 29.03.2016).

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

[8] Papenbrock Thorsten, Naumann Felix. A Hybrid Approach to Functional Dependency Discovery. — 2016. — 06. — P. 821–833.

[9] Paulley Glenn. Exploiting Functional Dependence in Query Optimization. — 2000. — 01.

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

[11] UGuide: User-Guided Discovery of FD-Detectable Errors / Saravanan Thirumuruganathan, Laure Berti-Equille, Mourad Ouzzani et al. — 2017. — 05. — P. 1385–1397.