Санкт-Петербургский государственный университет

Кафедра системного программирования
Программная инженерия

Салью Артур Кристофович

Высокопроизводительный поиск функциональных зависимостей в данных

Отчёт по учебной практике

Научный руководитель:
асс. кафедры ИАС Чернышев Г. А.

Санкт-Петербург
2022

Оглавление

1. Введение

Представим, что у нас имеются некоторые данные, в которых указаны имя, фамилия и марка автомобиля. Наиболее вероятно, что в имеющейся информации по имени и фамилии мы сможем однозначно определить марку автомобиля владельца. Подобное соответствие между элементами принято называть *функциональными зависимостями*.

Функциональные зависимости (далее *ФЗ*) являются важной характеристикой баз данных (*БД*). Например, с их помощью могут быть найдены ошибки, несоответствия в таблице или же, наоборот, какие-то полезные закономерности. С использованием ЭВМ задача поиска ФЗ в данных может быть успешно решена, однако же она требует большой вычислительной мощности устройства.

Первые серьёзные результаты в реализации программного обеспечения для выполнения оговорённой выше задачи были получены командой немецких исследователей [4]. Их платформа — Metanome объединила в себе все наиболее известные алгоритмы поиска функциональных зависимостей. Тем не менее, существенный её недостаток кроется в выбранном для реализации языке программирования — Java.

Данный язык подходит для быстрого проектирования приложений, но в то же время в ряде случаев Java-программы обладают сравнительно невысокой скоростью работы, а также существенно потребляют память устройства.

В целях устранения данных недостатков в настоящее время ведётся работа над проектом Desbordante — высокопроизводительной платформой для поиска функциональных зависимостей, целиком написанной на языке программирования C++.

В данном отчёте будет подробно рассмотрена реализация одного из алгоритмов поиска — FDep [3] на языке C++ для Desbordante [2].

2. Постановка задачи

Цель работы — реализация начальной версии алгоритма поиска функциональных зависимостей FDep на языке программирования C++ и исследование её производительности.
Для достижения поставленной цели были выделены следующие задачи:

- провести обзор предметной области;
- выполнить обзор алгоритма FDep;
- реализовать алгоритм на языке C++;
- проанализировать производительность программы.

3. Обзор предметной области

3.1. Функциональные зависимости

Функциональные зависимости рассматриваются в реляционной модели данных. Введем некоторые определения [1, 7]:

Определение 1 *Кортеж — множество упорядоченных троек вида* $\langle A_i, T_i, v_i \rangle$*, где* A_i *— имя атрибута,* T_i *— имя типа атрибута,* v_i *— значение типа* T_i*. Так как, вообще говоря, атрибуты являются уникальными, то неформально элементы кортежа описываются парами* $\langle A_i, v_i \rangle$

Определение 2 *Отношением* R *называют подмножество декартового произведения множеств* $T_1, T_2, ..., T_n$*, являющихся типами атрибутов. Отношение* R *состоит из схемы отношения — множества уникальных атрибутов и тела — множества кортежей* t*.*

Определение 3 *Пусть* R *— схема отношений, а* $X, Y \subseteq R$*. Выражение вида* $F : X \to Y$ *называется функциональной зависимостью.*

Определение 4 *Говорят, что отношение* R *удовлетворяет ФЗ:* $X \to Y$ *тогда и только тогда, когда для любых кортежей* t_1, t_2 *справедливо:* $t_1[X] = t_2[X] \Rightarrow t_1[Y] = t_2[Y]$*. Такую функциональную зависимость далее будем называть валидной.*

Пример:

Отношение, представленное на Рис. 1, содержит много информации: на всём отношении выполняется зависимость *Имя&Фамилия* $\to$ *Цвет автомобиля*, то есть на заданном отношении по имени и фамилии человека однозначно можно определить цвет его автомобиля. С другой стороны, на кортежах t_3, t_5 нарушается зависимость *Имя&Фамилия* $\to$ *Марка автомобиля*, что, вероятней всего, свидетельствует об ошибке в данных.

Отношение R			
Имя	Фамилия	Марка автомобиля	Цвет автомобиля
Александр	Петров	Desbordauto	Navy blue
Андрей	Серый	Desbordauto	Snakeskin Green
Василиса	Шавел	Fdepus	Ghost White
Дмитрий	Петров	Desbordauto	Navy blue
Василиса	Шавел	FdepuZ	Ghost White

Рис. 1: Пример функциональных зависимостей

Заметим, что в примере выполняются зависимости *Имя* $\rightarrow$ *Цвет автомобиля* и *Фамилия* $\rightarrow$ *Цвет автомобиля*. Данные ФЗ несут в себе больше всего информации. Очевидно, что ФЗ: *Имя*&*Фамилия* $\rightarrow$ *Цвет автомобиля* является лишь уточнением (*специализацией*) предыдущих. Сформулируем это в виде важного свойства — *наиболее информативными являются ФЗ с минимальной левой частью.*

Функциональные зависимости имеют достаточно широкую область применения. Они используются при нормализации БД [1, 7], для восстановления данных [3], а также для поиска в них различных ошибок и дубликатов.

3.2. Существующее решение

Задача поиска функциональных зависимостей в базе данных является достаточно требовательной к ресурсам ЭВМ. Так, верхняя оценка алгоритмической сложности [4] нахождения всех ФЗ — $O(n^2(\frac{m}{2})^2 2^m)$, где m есть мощность схемы, а n — количество кортежей в схеме. Существующее приложение, Metanome [4], целиком написано на языке Java, что влечёт за собой ряд трудностей:

1. Как правило, нативные приложения, написанные на C++, являются более производительными, чем Java-приложения. Причиной

тому — необходимость тонкой настройки на уровне JVM [6];

2. Производительность Java-приложений непредсказуема [5].
   Это вызвано JIT компилятором, автоматическим сборщиком мусора, оптимизацией, происходящей в JVM;

3. Программы на языке Java, как правило, имеют более высокие накладные расходы [8] на память, чем программы, написанные на C++.

4. Реализация алгоритма FDep

4.1. Описание

Алгоритм FDep предназначен для поиска функциональных зависимостей на заданном отношении. FDep относится к так называемым индукционным алгоритмам. В его основе лежит построение множества зависимостей, имея на начальном этапе лишь множество наиболее общих зависимостей. В работе [3] такой подход называется *bottom-up induction*. Далее сформулируем некоторые определения из статьи [3].

Определение 5 *Зависимость D будем называть минимальной, если множество* $DEP(r) \setminus \{D\}$ *уже не содержит всех ФЗ. Здесь DEP(r) — множество всевозможных зависимостей на отношении r.*

Определение 6 *Покрытием* $DEP(r)$ *назовем такое подмножество зависимостей* $COV \subseteq DEP(r)$*, состоящее лишь из минимальных зависимостей, с помощью которых можно уточнить информацию об остальных.*

Определение 7 *Будем говорить, что* $V \to W$ *следует из* $X \to Y$ *тогда и только тогда, когда* $\exists Z : V = XZ \ \& \ Y - Z \subseteq W \subseteq Y$*, где* $V, W, X, Y, Z \subseteq R$*. Здесь R — схема отношения.*

По исходным данным сложно построить множество валидных зависимостей, так как если ФЗ соблюдается на каком-то подмножстве тела отношения, то это не является гарантией валидности на всём отношении. Однако возможно действие от противного. Далее опишем основные шаги алгоритма, представленные на Рис. 2.

Первым шагом является инициализация множества наиболее общих зависимостей $DEPS = \{\emptyset \to A_i | A_i \in R\}$, где R — схема отношения.

Следующим важным шагом работы FDep является построение *отрицательного покрытия* $NCOV$ — множества невалидных зависимостей с минимальной левой частью. NCOV строится за один проход по данному отношению путём попарного сравнения всех кортежей.

Конечным ключевым шагом алгоритма является инвертирование NCOV — построение PCOV (положительного покрытия), множества всех минимальных, удовлетворяющих заданное отношение, зависимостей. Здесь происходит переход от общих начальных зависимостей к покрытию необходимых валидных ФЗ с учетом информации, имеющейся в NCOV.

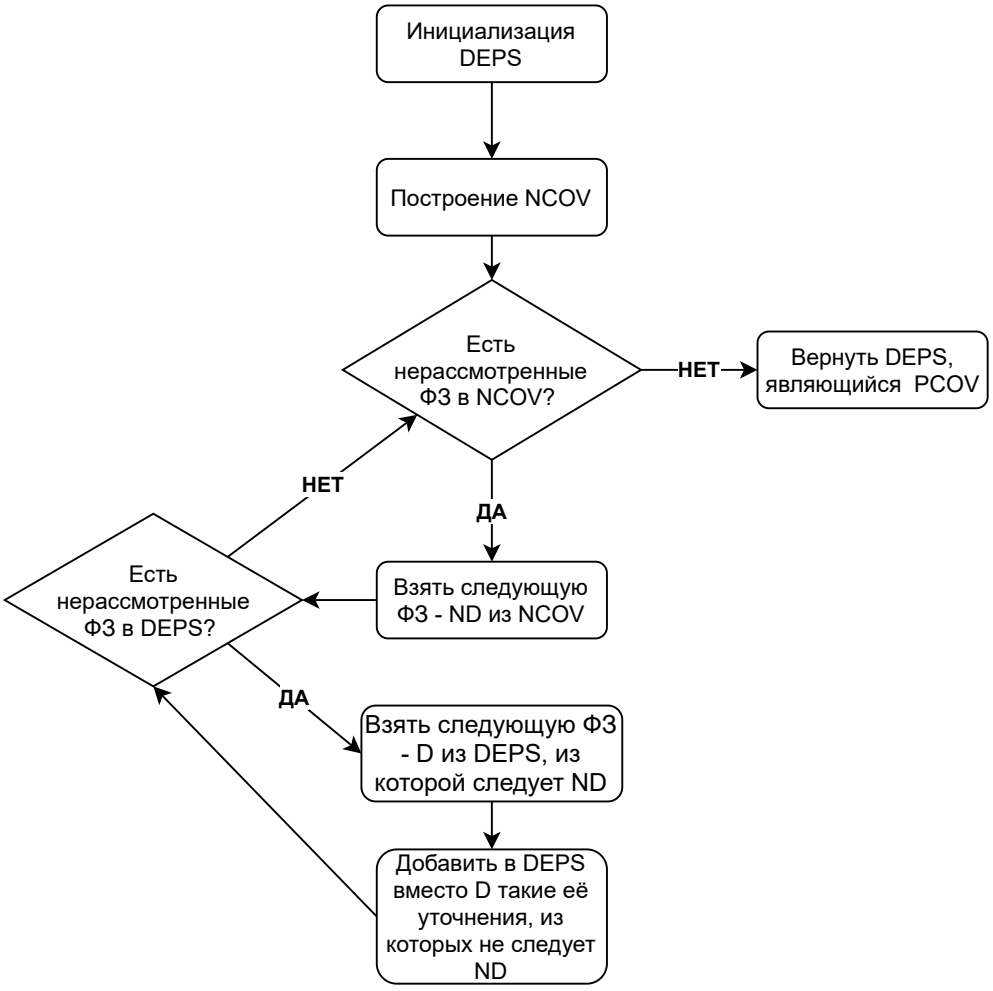

Рис. 2: Блок-схема алгоритма FDep

4.2. Используемые структуры

Для эффективных операций над зависимостями, которые повсеместно применяются при построении $NCOV$, а также $PCOV$ авторами статьи [3] вводится структура под названием *FD-Tree*. Данная структура представляет собой дерево, узлы которой являются атрибутами из

схемы отношений R. При этом должны быть выполнены следующие свойства:

- на атрибутах схемы задано отношение строгого порядка;
- дети узла больше самого узла;
- для каждой ФЗ: $X \to B$ из заданного множества ФЗ в дереве существует путь представляющий X, где $X, B \subseteq R$. Узлы на этом пути последовательно отмечены атрибутами B.

Пример:
Пусть задано множество зависимостей:

$$F = \{AB \to C, ACE \to B, AD \to C\}$$

Тогда FD-Tree, построенное на данном множестве представлено ниже:

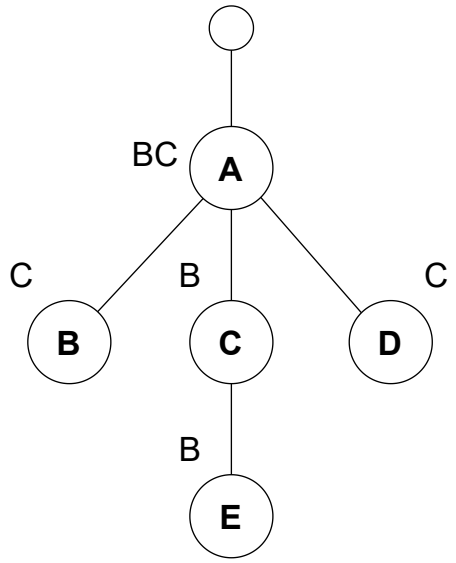

Рис. 3: Дерево функциональных зависимостей

Заметим, что наиболее общие зависимости расположены рядом с корнем дерева. Для того чтобы специализировать какую-то зависимость (рассмотреть ФЗ с расширенной левой частью), достаточно от её узла спуститься вглубь дерева. Например, у нас имеется зависимость $A \to BC$, из которой следуют $AC \to B$, $ACE \to B$, $AB \to C$, $AD \to C$. Функции специализации и обобщения (рассмотрения таких ФЗ, левая

часть которых является подмножеством исходной левой части) представлены в оригинальной статье [3] в виде рекурсивных функций.

Таким образом, необходимые функциональные зависимости представлены путями от корня до листьев дерева.

4.3. Детали реализации алгоритма

Алгоритм FDep был реализован на языке программирования C++ стандарта 17. FDTree представлен классом *FDTreeElement*, внутри которого активно используются динамические битсеты из библиотеки boost и умные указатели из стандартной библиотеки. Исходный код алгоритма доступен на GitHub[1].

[1] `https://github.com/ArthurChains/Desbordante`

5. Анализ производительности

5.1. Задача исследования

Главной задачей является анализ работы первой версии алгоритма на языке C++. Данное исследование должно выявить слабые и сильные стороны собственной реализации и задать вектор для дальнейшего её улучшения.

Таблица 1: Используемые при исследовании датасеты

Данные	Строки	Атрибуты	Размер[KB]	Число ФЗ
abalone	4177	9	187	137
balance-scale	625	5	7	4
breast-c-w	699	11	20	46
bridges	108	13	6	142
hepatitis	155	20	8	8250
horse	368	27	25	128726
ncvoter-1001	1000	19	151	758

На таблице выше представлены датасеты, взятые с uci[2] для анализа производительности собственной реализации FDep.

Эксперименты проводились на платформе следующей программной конфигурацией: операционная система Linux Ubuntu 20.04 64-bit, компилятор C++ gcc 9.3.0, виртуальная машина Java: OpenJDK 64-bit Server VM 11.0.11. Аппаратная конфигурация: Intel(R) Core(TM) i5-8300H CPU (4 ядра) @ 2.3GHz, 16GB DDR4 2666MHz RAM, 1TB HDD WD Elements SE.

При выполнении замеров с помощью системной утилиты cpufreq фиксировалась максимальная тактовая частота процессора — 4.0GHz.

[2]https://archive.ics.uci.edu/ml/datasets.php

5.2. Сравнение с Metanome

Для проведения экспериментов на рабочей платформе были развёрнуты Metanome, путём сборки и исполнения через OpenJDK 11, и Desbordante — сборка через gcc с уровнем оптимизации "-O3". Время измерялось от окончания обработки файла с данными, до получения набора ФЗ — построения PCOV.

Результаты отображены в таблице ниже.

Таблица 2: Сравнение времени исполнения (мс) Desbordante и Metanome

Платформа	abalone	balance-scale	breast-c-w	bridges	hepatitis	horse	ncvoter-1001
Desbordante	1191±10	19±0	61±0	4±0	87±0	4238±11	391±0
Metanome	1010±35	67±3	106±3	29±3	189±12	3811±20	334±5

По данным результатам видно, что реализованная первая версия FDep на языке C++ проигрывает на некоторых датасетах оригинальной имплементации от Metanome.

Стоит отметить, что алгоритм на C++ имеет большие возможности для дальнейшей оптимизации путём большего использования структур данных библиотеки boost или имплементации собственных более эффективных структур.

Заключение

В ходе данной работы были достигнуты следующие результаты:

- проведён обзор предметной области;
- проведён обзор работы алгоритма FDep и используемых структур;
- реализован алгоритм FDep на языке C++;
- выполнен анализ производительности программы.

Не на всех рассматриваемых данных реализация алгоритма FDep на C++ смогла продемонстрировать результаты лучшие, чем таковые у оригинальной имплементации. Можно выделить следующие направления продолжения работы:

- анализ высоких временных затрат реализованного алгоритма на некоторых таблицах;
- сравнительный анализ потребляемой памяти FDep Desbordante и Metanome;
- реализация более эффективной версии алгоритма.

Список литературы

[1] Date C.J. An Introduction to Database Systems. — 8 edition. — USA : Addison-Wesley Longman Publishing Co., Inc., 2003. — ISBN: 0321197844.

[2] 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.

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

[4] 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. — Access mode: https://doi.org/10.14778/2794367.2794377.

[5] Georges Andy, Buytaert Dries, Eeckhout Lieven. Statistically Rigorous Java Performance Evaluation // SIGPLAN Not. — 2007. — Oct. — Vol. 42, no. 10. — P. 57–76. — Access mode: https://doi.org/10.1145/1297105.1297033.

[6] Group JUG Ru. Диалоги о Java Performance // Хабр. — 2016. — Режим доступа: https://habr.com/ru/company/jugru/blog/280420/ (дата обращения: 16.03.2021).

[7] JetBrains Образовательные проекты. Введение в функциональные зависимости // Хабр. — 2019. — Режим доступа: https://habr.com/ru/company/JetBrains-education/blog/473882/ (дата обращения: 14.03.2021).

[8] Wikipedia. Comparison of Java and C — Wikipedia, The Free Encyclopedia. — https://en.wikipedia.org/wiki/Comparison_of_Java_and_C++. — 2021. — (Online accessed: 16.03.2021).