Санкт-Петербургский государственный университет

Кафедра информационно-аналитических систем

Группа 20Б.08-мм

Щека Дмитрий Вадимович

Реализация алгоритма BHUNT для поиска нечётких алгебраических ограничений

Отчёт по учебной практике
в форме «Производственное задание»

Научный руководитель:
Ассистент кафедры ИАС Чернышев Г.А.

Санкт-Петербург
2023

Оглавление

Введение

Производительность систем управления базами данных все время улучшается. Для этого есть множество различных методов: как программных, так и аппаратных. Алгебраические ограничения могут позволить улучшить производительность таких систем путем оптимизации запросов данных [2]. Алгебраические ограничения нужны для выявления предикатов ограничения, которые могут потенциально ускорить запросы по конкретным парам атрибутов отношения. Так же их можно использовать не только как автоматическую систему оптимизации, но и для изменения физической организации данных.

Алгебраические ограничения могут быть интересны и сами по себе, поскольку позволяют найти новые закономерности в имеющихся данных. По алгебраическим ограничениям можно строить предикаты, которые показывают для пары колонок, как значения одной находятся в интервалах, получающихся из значений второй колонки применением к ним арифметических операций.

1. Постановка задачи

Целью работы является реализация алгоритма BHUNT для поиска тривиальных алгебраических ограничений. Для достижения цели были поставлены следующие задачи:

- провести обзор алгоритма BHUNT;
- реализовать алгоритм на языке программирования C++;
- произвести тестирование алгоритма.

2. Краткое описание

2.1. Упрощенное описание алгоритма

Пользователь передает алгоритму таблицу в виде .csv файла. Для нахождения тривиальных алгебраических ограничений нужно, чтобы в таблице было хотя бы две колонки данных, значения которых имеют одинаковый числовой тип. Пусть такие колонки нашлись, тогда берутся всевозможные пары этих колонок. Далее к паре значений, стоящих в одной строке, применяется некоторая бинарная арифметическая операция. Полученные результаты операции группируются (отдельно для каждой пары колонок) на непересекающиеся интервалы или отрезки. Результатом работы алгоритма являются эти наборы интервалов, с помощью которых можно построить предикат. Этот предикат демонстрирует, как значения первой колонки зависят от значений второй.

Это упрощенное описание, поэтому в нем опущены процесс разбиения на интервалы с помощью сегментирования и процесс выбора строк таблицы.

2.2. Влияние параметров алгоритма

В зависимости от переданных алгоритму параметров можно получать различные наборы интервалов.

Параметр **weight** является значением из диапазона $(0, 1)$. Он влияет на размер и количество интервалов: чем ближе к 1, тем интервалов меньше и тем они крупнее (поэтому, начиная с некоторого места при приближении к единице, результатом будет всего один интервал, захватывающий все значения), чем ближе к 0, тем интервалов больше и тем они меньше.

Параметры **fuzziness** и **p_fuzz** влияют на точность алгоритма. Параметры принимают значения из диапазона $(0, 1)$. Чем ближе **fuzziness** к 0, а **p_fuzz** к 1, тем больше строк таблицы будет выбрано. Например, при **p_fuzz** $= 1$ значение арифметической операции будет вычислено во всех строках, поэтому в интервалы попадут значения для каждой

строки без исключений. Чем ближе **fuzziness** к 1, а **p_fuzz** к 0, тем меньше строк таблицы будет выбрано, поэтому многие значения не попадут в интервалы.

Число получаемых интервалов можно ограничить при помощи параметра **bumps_limit**. Конкретное число интервалов для всех пар колонок можно получить, передав нужное значение в **bumps_limit**, а затем подобрав достаточно малый **weight**.

Параметр **iterations_limit** принимает значения натурального числа. При передаче значения близкого к 1 может уменьшиться точность алгоритма.

2.3. Простой пример

Предположим у нас таблица доставки заказов из двух колонок. В первой колонке находится день отправки заказа, во второй — день его прибытия до места доставки. Запустив алгоритм с арифметической операцией вычитания, в каждой строке таблицы будет вычислена разность между днем отправки и прибытия. Эти разности можно представить в виде гистограммы Рис. 1.

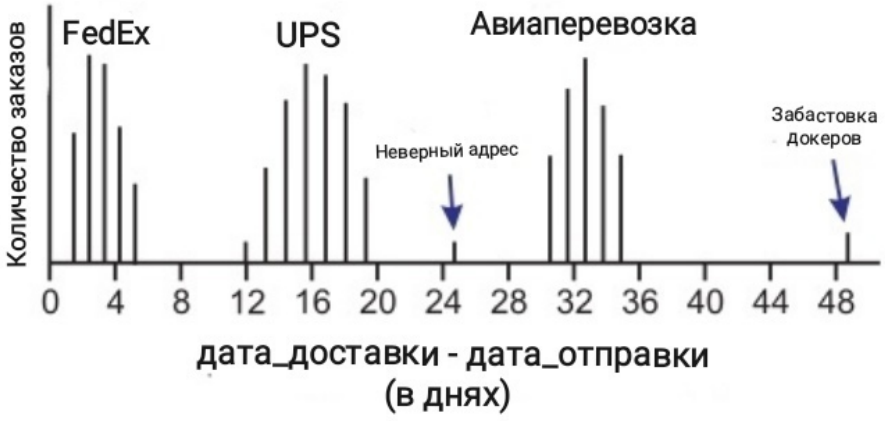

Рис. 1: гистограмма разностей, рисунок адаптирован из работы [2].

Тогда, в зависимости от параметров алгоритма, мы получим различное число интервалов, в которые попадает наибольшее число заказов. Исключительные случаи (например, как неверный адрес) тоже

могут быть не включены в интервалы, при выборе некоторых значений **fuzziness** и **p_fuzz**.

Если ограничить число получаемых интервалов тремя и выбрать достаточно малый **weight**, то интервалы, вероятнее всего (в процессе выбора строк используется генератор случайных чисел), будут представлять из себя границы трех «кочек» на гистограмме. Таким образом, можно по найденным интервалам построить предикат, демонстрирующий зависимость между двумя колонками:

```
    (дата_доставки BETWEEN дата_отправки + 2 DAYS
    AND дата_отправки + 5 DAYS)
OR (дата_доставки BETWEEN дата_отправки + 12 DAYS
    AND дата_отправки + 19 DAYS)
OR (дата_доставки BETWEEN дата_отправки + 31 DAYS
    AND дата_отправки + 35 DAYS)
```

Большая часть данных удовлетворяет данному предикату, поэтому те данные, что не удовлетворяют, считаются исключениями.

3. Обзор алгоритма BHUNT

3.1. Основные понятия

Алгоритм носит название BHUNT [5] (Bump HUNTer, где дословный перевод: «охотник за кочками»), потому что при гистограммировании интервалы, которые ищет алгоритм, можно определить по кочкам на графике.

Алгебраическое ограничение мы будем записывать как

$$AC = (a_1, a_2, P, \oplus, I_1, ..., I_k)$$

Где a_1, a_2 — это атрибуты отношения. P — правило, которое диктует как элементы одного атрибута образуют пары с элементами второго, то есть задает некоторое биективное отображение. Зачастую это JOIN предикат либо тривиальное правило, где каждому элементу одного атрибута соответствует элемент второго из той же строки. $\oplus$ — бинарная операция, которую можно использовать между элементами атрибутов. I_i — промежутки действительных чисел.

3.2. Краткое описание работы алгоритма BHUNT

1. Найти пары атрибутов, на элементах которых возможно применять арифметические операции;

2. Выбрать правила разбиения на пары. На этом этапе мы получаем набор объектов вида $(a_1, a_2, P, \oplus)$, который называется *кандидатом*;

3. В зависимости от начальных данных алгоритма выбрать размер образца отношения, где размер — это количество пар элементов атрибутов кандидата (либо же просто количество строк, если считать, что JOIN уже был выполнен), к которым будет применена операция $\oplus$;

4. Результаты применения операции $\oplus$ (их обозначать будем за x_i)

разбиваются на интервалы с помощью одного из следующих способов: кластеризация, сегментирование, гистограммирование;

5. Исключения запоминаются;

6. Полученные интервалы можно использовать для оптимизации запросов.

3.3. Дополнительные детали

3.3.1. Атрибуты

В отношении берутся атрибуты, имеющие типы, над которыми определены бинарные арифметические операции. При разбиении на пары рассматриваются атрибуты одного типа. Можно было бы рассматривать различные типы, если один приводим к другому, но это сильно ухудшает производительность алгоритма.

3.3.2. Параметры алгоритма

k_{max} — максимальное количество интервалов, предоставляемых алгоритмом (если количество интервалов превосходит k_{max}, тогда соединяем интервалы с помощью жадного алгоритма, пока не станет ровно k_{max}).

f — дробь, представляющая какая часть значений окажется исключением.

p — вероятность, с которой исключений будет максимум f.

i_{max} — максимально количество итераций для вычисления размера образца.

w — вес, необходимый для регулировки мелкости разбиения при сегментировании. Принимает действительные значения в диапазоне $(0, 1)$. Ближе к нулю будет много коротких интервалов, ближе к единице — мало длинных интервалов.

3.3.3. Вычисление размера образца

Вычисление размера образца выполняется по следующей формуле:

$$n(k) \approx \frac{\chi^2_{1-p}(2-f)}{4f} + \frac{k}{2}$$

Где χ^2_{1-p} считается по формулам 26.2.23 и 26.4.17 в [3], а степень свободы принять равной $2(k+1)$.

Подсчет размера происходит по следующему алгоритму:

1. Инициализация: $i = 1$ и $k = 1$;
2. Подсчитать размер образца для начальных значений $n = n(k)$;
3. Вычислить алгебраическое ограничение по данному размеру образца (n строк выбираются с помощью семплирования Бернулли). Получить k' интервалов;
4. Если $n(k') \leq n$ или $i = i_{max}$, тогда закончить. Иначе $k = k'$ и $i = i + 1$, и вернуться к шагу 2.

3.3.4. Разбиение на интервалы

Примером метода разбиения на интервалы может быть сегментирование. Два раздельных значения должны попасть в один интервал, если $x_{t+1} - x_t < d$, где $d = \Delta(w/(1-w))$.

Δ вычисляется как $max|x_t - x_l| \ \forall \ t, l$.

w описан выше в разделе «Параметры алгоритма».

4. Реализация алгоритма

Алгоритм реализовывался в высокопроизводительной системе профилирования данных Desbordante [6], где основным языком разработки является C++ из-за его высокого быстродействия.

Использовались уже реализованные [1] классы для хранения и получения данных .csv файлов. А именно ColumnLayoutTypedRelationData, позволяющий получать тип атрибутов и значения элементов этих атрибутов.

Поддерживаемыми типами являются numeric типы, определенные в types/Types.h.

В качестве P используется тривиальное правило разбиения на пары. Атрибуты, имеющие элементы со значением NULL игнорируются. Приведение типов не применяется.

На момент первоначальной версии алгоритм разбит на 4 функции, находящиеся в классе ACAlgorithm.

- CalculateSampleSize: С помощью функций из стандартной библиотеки <cmath> и упомянутых ранее формул вычисляет размер образца;

- Sampling: использует функцию bernoulli_distribution [4] из стандартной библиотеки <random> для получения количества пар значений атрибутов, равного вычисленному размеру образца, проводит заданную арифметическую операцию между значениями. Записывает результаты и сортирует их. После нахождения интервалов и проверки, что размер образца подходит, возвращает полученные интервалы. Если размер образца не подходит, то, в соответствии с алгоритмом вычисления размера образца, пересчитывает размер образца и повторяет запуск поиска интервалов;

- ConstructDisjunctiveRanges: при помощи сегментирования разбивает результаты применения арифметической операции на интервалы;

- RestrictRangesAmount: проверяет, что количество интервалов не больше k_{max}. Если больше, то жадно соединяет интервалы, пока не станет k_{max} интервалов.

5. Тестирование

Было написано 7 тестов для алгоритма поиска алгебраических ограничений. Таким образом, был протестирован метод расчета размера образца, был протестирован неточный поиск промежутков на небольшой таблице.

Также были протестированы арифметические операции. Были использованы параметры **fuzziness** $= 0$ и **p_fuzz** $= 1$, чтобы значения арифметической операции были вычислены для всех строк и промежутки включали в себя все значения. Был взят знаменитый набор данных iris. Например для операции сложения и колонок таблицы с индексами 2 и 3, которые обозначают длину (petal_length) и ширину (petal_width) лепестка соответственно, будет гистограмма Рис. 2.

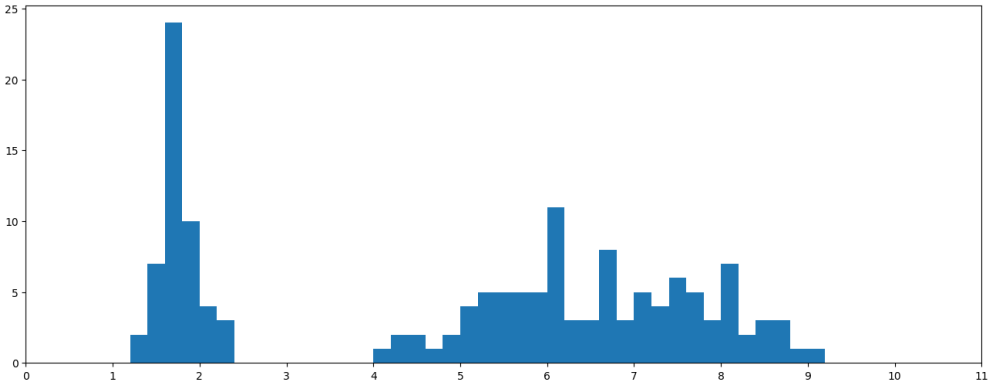

Рис. 2: гистограмма petal_length + petal_width

И результатом работы алгоритма при значении **weight** $= 0.05$ являются промежутки $[1.2, 2.3]$ и $[4.1, 9.2]$, крайние точки которых совпадают с точными границами «кочек» на гистограмме Рис. 2. С помощью этих промежутков можно выразить зависимость между длиной и шириной лепестка.

Для первого промежутка: $(petal\_length + petal\_width) \in [1.2,\ 2.3] \Rightarrow$
$\Rightarrow petal\_length \in [1.2 - petal\_width,\ 2.3 - petal\_width]$

Заключение

В ходе работы были выполнены следующие задачи:

- был написан обзор алгоритма BHUNT;
- был реализован алгоритм нахождения тривиальных алгебраических ограничений;
- было произведено тестирование алгоритма.

Можно выделить следующие направления продолжения работы:

- Увеличение количества поддерживаемых типов данных;
- Реализация более эффективной версии алгоритма;
- Реализация других методов группирования на интервалы.

Код алгоритма доступен на Github[1].

[1] https://github.com/Mstrutov/Desbordante/pull/124

Список литературы

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

[2] IBM. B-HUNT: Automatic Discovery of Fuzzy Algebraic Constraints in Relational Data. — 2003. — URL: https://people.cs.umass.edu/~phaas/files/vldb2003.pdf.

[3] M.Abramowitz, I.A.Stegun. Handbook of mathematical functions. — 1970.

[4] Microsoft. Справочник по стандартной библиотеке C++. — URL: https://docs.microsoft.com/ru-ru/cpp/standard-library/.

[5] Paul G. Brown Peter J. Haas. BHUNT: Automatic Discovery of Fuzzy Algebraic Constraints in Relational Data. — 2003. — URL: https://vldb.org/conf/2003/papers/S20P03.pdf.

[6] Unidata. Data profiling, и с чем его едят. — 2022. — URL: https://habr.com/ru/company/unidata/blog/667636/.