Санкт-Петербургский государственный университет

Кафедра системного программирования

Группа 21.Б11-мм

Интеграция алгоритма по поиску условных функциональных зависимостей в “Desbordante”

Выродов Михаил Владимирович

Отчёт по учебной практике
в форме «Решение»

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

Санкт-Петербург
2023

Оглавление

Введение

В современном мире количество полезных данных растёт с неимоверной скоростью. Сейчас практически каждая IT-компания накапливает у себя терабайты данных, чтобы после их анализа сделать вывод, который поможет улучшить продукты компании. Например, большинству веб-разработчиков интересно, как пользователь взаимодействует с их сайтом. По поведению мышки пользователя можно понять, как надо расположить кнопки на сайте, чтобы среднестатистический посетитель чувствовал себя комфортно и не выходил с сайта как можно дольше. Также операторам мобильной связи важно знать, что послужило причиной ухода их клиентов к конкурентам. Для этого они анализируют огромную базу всех своих пользователей и ищут закономерности между конкретной информацией о клиенте и тем, ушёл он к другому оператору связи или остался.

Для таких задач существует профилирование данных. Согласно определению, это процесс извлечения метаданных из данных. Возникает вопрос, что такое метаданные и причём тут поиск закономерностей? Метаданные — это данные о данных. Например, если мы рассматриваем файл, то его метаданными являются название, имя автора, время создания и размер. Но, на самом деле, мы можем понимать это определение чуть шире. Оно ещё включает в себя разного рода зависимости, которые могут быть не очевидны при первом взгляде на данные. Профилирование данных, которое ищет такие "скрытые" отношения между данными называют наукоёмким профилированием.

Рассмотрим табличные данные. Закономерности между табличными атрибутами могут описываться совершенно разными способами. Мы будем описывать их с помощью примитивов. Примитив — это описание правил, заданных над какой-то частью данных с помощью математических методов. Такой способ описания позволяет задать отношение максимально точно и ясно. В качестве примера можно привести функциональные зависимости (FD). Пусть X, Y это множества атрибутов в таблице. Функциональная зависимость $X \to Y$ существует, если для

каждых двух строк в таблице, если равны значения атрибутов X, то и равны значения атрибутов Y.

Примитивы могут автоматически находиться с помощью различных алгоритмов. Они могут оказаться довольно полезными для разного рода аналитиков и учёных. Найденная закономерность может помочь аналитикам лучше понять, как взаимосвязаны данные, над которыми они работают, а учёным подать новую идею для эксперемента.

Инструмент для профилирования данных Desbordante[1] предназначен для продвижения использования примитивов в массы и предоставления возможности людям изучать свои данные с их помощью.

Из множества примитивов можно выделить условные функциональные зависимости (CFD). Это функциональные зависимости, которые удовлетворяют некоторому шаблону данных (условию) и которые могут иметь определённую неточность (некоторое количество ошибок). В этом определении шаблон данных — это заранее заданные значения для подмножеств X и Y. Такого рода закономерности позволяют видеть зависимости на конкретном подмножестве таблицы, что может быть крайне полезно. В Desbordante уже реализован алгоритм по поиску CFD, но существует ещё более быстрые варианты [4] для их майнинга. Моя задача заключается в интеграции этих более эффективных алгоритмов в проект.

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

1. Постановка задачи

Целью работы является интеграция наиболее эффективного алгоритма по майнингу CFD, описанного в статье [5]. Для её выполнения были поставлены следующие задачи:

1. Выделить самый эффективный алгоритм из [5] и описать принцип его работы;
2. Интегрировать код алгоритма [4] в Desbordante, по возможности используя уже написанные структуры данных из проекта;
3. Протестировать интегрированный код, сравнив его результаты с результатами оригинального кода.

2. Обзор

Все определения в данном разделе взяты из работы [5], а примеры из работ [2, 5].

Определение 2.1. Обозначим множество всех атрибутов в таблице буквой A, домен атрибута B обозначим как $dom(B)$.

Определение 2.2. Кортежем (tuple) t называется упорядоченное множество пар вида $(B, t[B])$, где $B \in A$ и $\forall B$ либо значение $t[B] \in dom(B)$, либо $t[B]$ является безымянной переменной, которая обозначается "_". Проекция t на множество атрибутов X обозначается как $t[X]$. Подмножество множества всех возможных кортежей обозначим за D.

Определение 2.3. Условная функциональная зависимость φ это пара вида $(X \to Y, t_p)$, заданная на D, где $X \subset A$, $Y \in A$ и $Y \notin X$, и $X \to Y$ — это классическая функциональная зависимость, а t_p — шаблонный кортеж (pattern tuple), т.е. кортеж, где ключами являются атрибуты из $X \cup Y$. CFD φ, для которой $t[Y] =$ "_", называют переменной, в других случаях φ называют константной. Множество атрибутов $X \cup Y$ обозначается как $attrs(\varphi)$. Заметим, что если $\forall B \in attrs(\varphi)$ $t[B] \neq$ "_", то такая CFD φ полностью соответствует FD $X \to Y$.

Определение 2.4. Пусть кортеж $t \in D$ соответствует шаблонному кортежу t_p на множестве атрибутов X, если $\forall B \in X$ либо $t_p[B] =$ "_", либо $t[B] = t_p[B]$. Обозначим как $t \sim t_p$. Кортеж t нарушает переменную CFD φ, если $t[X] \sim t_p[X]$ и существует другой кортеж t', что $t[X] = t'[X]$ и $t[Y] \neq t'[Y]$. Кортеж t нарушает константную CFD φ, если $t[X] = t_p[X]$ и $t[Y] \neq t_p[Y]$. Множество кортежей из D, нарушающих φ, обозначается как $VIO(\varphi, D)$. Если $VIO(\varphi, D) = \emptyset$, то говорят, что D удовлетворяет φ. Это обозначается как $D \models \varphi$.

Пример 1. Рассмотрим пример, который был взят из [2]. Датасет на рисунке 1 был взят из [1]. Он хранит данные о пользователях оператора сотовой связи, а именно о номере телефона пользователя: атрибуты CC

(country code), AC (area code) и PN (phone number), об имени пользователя: атрибут NM (name), и об адресе пользователя: атрибуты STR (street), CT (city), ZIP (zip code). На приведённых кортежах есть две функциональные зависимости:

1. $f_1 : [CC, AC] \to CT$;
2. $f_2 : [CC, AC, PN] \to STR$.

f_1 значит, что два пользователя из датасета, у которых одинаковые коды стран и коды областей, живут в одном городе. Похожий смысл имеет и f_2. Приведём примеры некоторых CFD на этом датасете:

1. $\varphi_1 : [CC, ZIP] \to STR$, (44, _ || _);
2. $\varphi_2 : [CC, AC] \to CT$, (_, _ || _);
3. $\varphi_3 : [CC, AC] \to CT$, (01, 908 || MH);
4. $\varphi_4 : [CC, AC] \to CT$, (44, 131 || EDI).

φ_1 отражает новую закономерность, которая выполняется только на тех кортежах t, для которых $t[CC] = 44$. φ_2 идентична f_1, а φ_3 и φ_4 выражают ту же зависимость, что и φ_2, но на конкретном подмножестве кортежей, соответствующих шаблонным кортежам φ_3 и φ_4.

CC	AC	PN	NM	STR	CT	ZIP
01	908	1111111	Mike	Tree Ave.	MH	07974
01	908	1111111	Rick	Tree Ave.	MH	07974
01	212	2222222	Joe	5th Ave	NYC	01202
01	908	2222222	Jim	Elm Str.	MH	07974
44	131	3333333	Ben	High St.	EDI	EH4 1DT
44	131	4444444	Ian	High St.	EDI	EH4 1DT
44	908	4444444	Ian	Port PI	MH	W1B 1JH
01	131	2222222	Sean	3rd Str.	UN	01202

Рис. 1: Датасет для примера 1

Определение 2.5. Пара (B, v), где $B \in A$ и $v \in dom(B)$, поддерживается кортежем t, если $t[B] = v$. Пара (B, "_") поддерживается любым кортежем. Пара вида (B, v) или (B, "_") называется элементом (item), а

их множество — набором элементов (itemset). Кортеж t поддерживает itemset I, если он поддерживает все элементы из I. Количество кортежей в D, которые поддерживают itemset I, обозначается как $supp(I, D)$. Множество уникальных идентификаторов этих кортежей обозначается как $cov(I, D)$. Теперь CFD $\varphi = (X \to Y, t_p)$ рассматривается как ассоциативное правило (association rule) $I \to j$ между itemset'ом I и item'ом j, где $I = \bigcup_{B \in X}\{(B,\, t_p[B])\}$ и $j = (Y, t_p[Y])$.

Определение 2.6. По аналогии с приближёнными функциональными зависимостями, было введено понятие уверенности (confidence) для CFD. Уверенность CFD ($\varphi = I \to j$) — это $conf(\varphi, D) = 1 - \frac{|D^{'}|}{supp(I,D)}$, где $|D^{'}| \subset D$ — это минимальное подмножество, для которого $D \setminus D^{'} \models \varphi$. Для константной CFD $|D^{'}| = |VIO(\varphi, D)|$, поэтому можем переписать формулу так — $conf(\varphi, D) = \frac{supp(I,D) - |VIO(\varphi,D)|}{supp(I,D)} = \frac{supp(I \cup \{j\},D)}{supp(I,D)}$, а это уже стандартная уверенность для ассоциативных правил. CFD φ называется точной на D, если $conf(\varphi, D) = 1$.

Определение 2.7. Пусть два кортежа t_1 и t_2 эквивалентны по отношению к I, если $\forall (B, v) \in I, t_1[B] = t_2[B] \sim v$. Все классы эквивалентности по отношению к I обозначаются как $\Pi(I)$. Для одной константной пары (B, v), $\Pi(B, v) = \{cov((B, v), D)\}$, т.е. это множество уникальных идентификаторов кортежей, поддерживающих (B, v). Для одной переменной пары (B, "_") $\Pi(B,$ "_") $= \{cov(B, v) \mid v \in dom(B)\}$. Для itemset'а I, $\Pi(I) = \bigcap_{i \in I} \Pi(i)$, где все классы эквивалентности попарно пересекаются. Обозначим за $|\Pi(I)|$ количество классов эквивалентности в нём, а за $||\Pi(I)||$ — количество элементов во всех классах эквивалентности, иначе — $supp(I, D)$.

Пример 2. Рассмотрим пример из [5]. Использован датасет из [3]. Одной из приближённых CFD φ на нём является $\{(Windy,\, false), (Outlook,\, \_)\} \to (Play,\, \_)$. Пусть $I = \{(Windy, false), (Outlook, \_),\ (Play, \_)\}$, а $j = (Play, \_)$. $\Pi(I \setminus \{j\}) = \{\{1, 8, 9\}, \{3, 13\}, \{4, 5, 10\}\}$ и $\Pi(I) = \{\{1, 8\},\ \{9\}, \{3, 13\},\ \{4, 5, 10\}\}$. Заметим, что $|\Pi(I \setminus \{j\})| = 3$ и $|\Pi(I)| = 4$ и оба отношения имеют поддержку $||\Pi(I \setminus \{j\})|| = ||\Pi(I)|| = 8$. Поддерживаемые кортежи, где $t[Windy] =$ false, окрашены в оттенки се-

рого. Каждому оттенку соответствует класс эквивалентности в $\Pi(I)$. φ может быть точной при удалении кортежа с номером девять, чтобы $\Pi(I \setminus \{j\}) = \Pi(I)$. Следовательно, $conf(\varphi, D) = 1 - \frac{|D^{'}|}{||\Pi(I)||} = 1 - \frac{1}{8} = 0.875$. Наконец, $VIO(\varphi, D) = \{t_1, t_8, t_9\}$.

tid	Outlook	Temperature	Humidity	Windy	Play
1	sunny	hot	high	false	dont
2	sunny	hot	high	true	dont
3	overcast	hot	high	false	play
4	rain	mild	high	false	play
5	rain	cool	normal	false	play
6	rain	cool	normal	true	dont
7	overcast	cool	normal	true	play
8	sunny	mild	high	false	dont
9	sunny	cool	normal	false	play
10	rain	mild	normal	false	play
11	sunny	mild	normal	true	play
12	overcast	mild	high	true	play
13	overcast	hot	normal	false	play
14	rain	mild	high	true	dont

Рис. 2: Датасет для примера 2

В Desbordante уже реализован алгоритм для поиска CFD — CTANE[2], но в статье [5] было экспериментально показано, что алгоритм FD-First в большинстве случаев в несколько раз быстрее, чем CTANE.

[2]https://github.com/vs9h/Desbordante/tree/ctane/src/algorithms/c_tane

3. Метод

Мною был разобран алгоритм FD-First [5], а так же интегрирована его версия FD-First-DFS [4].

3.1. Описание алгоритма

Алгоритмы FD-First и Itemset-First предназначены для нахождения CFD на заданном множестве кортежей. Входными параметрами алгоритмов являются минимальные значения поддержки и уверенности найденных CFD и максимальное количество атрибутов в левой части каждой CFD φ.

В статье [5] было экспериментально показано, что в большинстве случаев алгоритм FD-First намного эффективнее, чем Itemset-First.

Далее приведено описание алгоритма FD-First. Сначала на заданном множестве кортежей D находятся все функциональные зависимости. Если уверенность найденной зависимости больше минимального коэффициента уверенности, то она добавляются в множество найденных CFD (FD $\varphi = I \setminus \{j\} \to j$ является CFD вида (φ, t_p), где $t_p = \{(B,$ "_") $\mid B \in attrs(I)\}$). Однако, если её уверенность меньше, чем единица, то с помощью функции `MinePatterns`, подбираются константные шаблонные кортежи, при которых CFD, заданная с помощью этой FD и подобранного шаблонного кортежа, имеет больший коэффициент уверенности, чем минимально заданный. Такие CFD добавляются в множество найденных CFD. В алгоритме Fd-First CFD представляются через ассоциативные правила, т.е. как $(I \setminus \{j\} \to j)$. Таким образом, результатом алгоритма является множество Σ, в котором хранятся все удовлетворяющие параметрам найденные CFD.

Ключом к эффективности этого алгоритма является тот факт, что поддержка и уверенность рассматриваемой CFD $I \setminus \{j\} \to j$ могут быть быстро посчитаны, используя $\Pi(I)$. Действительно, каждый класс эквивалентности из $\Pi(I)$ соответствует уникальному константному шаблонному кортежу на атрибутах $attrs(I)$. Присвоим каждому классу эквивалентности уникальный номер. Определим покрытие itemset'а (item'а)

J на отношении $\Pi(I)$ как множество номеров классов эквивалентности $\Pi(I)$, в которых itemset (item) J встречается. Обозначим это как $cov(J, \Pi(I))$. Так как обычно $|cov(J, \Pi(I))| \ll |cov(J, D)|$, то эффективность поиска кортежей, соответствующих шаблонному кортежу, повысилась.

Пример 3. Датасет для примера был взят из [3]. Рассмотрим FD $\{(Windy, \_), (Outlook,)\} \rightarrow (Play, \_)$, соответствующую itemset'у $I = \{(Windy, \_), (Outlook, \_), (Play, \_)\}$ с множеством классов эквивалентности $\Pi(I) = \{\{1, 8\}, \{2\}, \{3, 13\}, \{4, 5, 10\}, \{6, 14\}, \{7, 12\}, \{9\}, \{11\}\}$. Константный кортеж $(Windy, false)$ может быть представлен его pidlist'ом, т.е. $cov((Windy, false), \Pi(I)) = \{1, 3, 4, 7\}$. Т.к. $supp((Windy, false), D) = 8$, мы уменьшили объём покрытия этого кортежа вдвое.

Таблица 1: Датасет для примера 3

tid	Outlook	Temperature	Humidity	Windy	Play
1	sunny	hot	high	false	dont
2	sunny	hot	high	true	dont
3	overcast	hot	high	false	play
4	rain	mild	high	false	play
5	rain	cool	normal	false	play
6	rain	cool	normal	true	dont
7	overcast	cool	normal	true	play
8	sunny	mild	high	false	dont
9	sunny	cool	normal	false	play
10	rain	mild	normal	false	play
11	sunny	mild	normal	true	play
12	overcast	mild	high	true	play
13	overcast	hot	normal	false	play
14	rain	mild	high	true	dont

Первым шагом алгоритма является инициализация поисковой решётки. Для этого сначала инициализируется множество $\Lambda = \{(B, \text{“\_”}) \mid B \in A\}$. После вычисляется $\Pi(\{i\}, D)$ для каждого $i \in \Lambda$. Наконец, инициализируется поисковая решётка E, изначально равная Λ, и создаётся множество Σ для хранения найденных CFD.

Вторым шагом является обход решётки. Пока решётка не пустая, из неё берётся itemset I, при этом удаляясь из самой решётки. Далее для каждого $j \in I$ рассматривается FD вида $(I \setminus \{j\} \to j)$. Если выполняется неравенство $conf(I \setminus \{j\} \to j, D) \geq 1 - \varepsilon$, то в Σ добавляется $\{I \setminus \{j\} \to j\}$.

Если рассматриваемая FD имеет уверенность меньше единицы, то начинается поиск всех константных кортежей, на которых CFD φ, заданная с помощью этой FD и найденного кортежа, имеет подходящую уверенность. Это сделано так, потому что если уверенность найденной FD равна единице, то φ имеет единичную уверенность на любом шаблонном кортеже, поэтому нет смысла для такой FD искать константные кортежи. Для поиска подходящих кортежей сначала инициализируется решётка константных пар $\Lambda^{pat} = \{(B, v) \mid B \in attrs(I), v \in dom(A)\}$. Для каждой пары её pidlist считается из классов эквивалентности в $\Pi(I)$. Функция `MinePatterns()` проходится по этой решётке, генерируя pidlist'ы новых itemset'ов, пересекая pidlist'ы двух его родителей в решётке. Поддержка itemset'а M может быть легко посчитана с помощью его pidlist'а — $supp(M, \Pi(I)) = \sum_{pid \in cov(M, \Pi(I))} |\Pi(i)[pid]|$, где $\Pi(i)[pid]$ — класс эквивалентности с номером pid. Рассматриваются только шаблонные itemset'ы M с $supp(M, \Pi(I))$ больше заданного минимального значения. Для всех таких M itemset $I \setminus \{j\}$ заменяется на $W = M \cup \{(B,$ "_") $\in I \setminus \{j\} \mid B \notin attrs(M)\}$, т.е. W содержит все элементы M и все элементы $I \setminus \{j\}$, в которых атрибут не из M. Далее расматривается CFD на множестве W и считается её уверенность. Если она равна единице, то мы добавляем эту CFD в Σ.

3.2. Интеграция алгоритма

В readme файле кода алгоритмов [4] сказано, что версия FD-First-DFS в общем случае более эффективна, чем FD-First-BFS, поэтому было принято решение интегрировать именно FD-First-DFS версию.

В Desbordante классы представления данных наследуются от класса `AbstractRelationData`. Этот класс шаблонный, в качестве шаблонного параметра имеет класс представления данных в одной колонке. В

AbstractRelationData реализован основной интерфейс работы с табличными данными. Например, доступ к столбцам по ссылке, доступ к имени таблицы, к её атрибутам и к значениям в конкретном столбце. В свою очередь, класс представления данных в одной колонке надо наследовать от класса AbstractColumnData, в котором хранятся имя и индекс колонки, а также в нем представлен интерфейс, который должны реализовывать наследники класса.

В оригинальном коде алгоритмов тоже есть класс представления данных — Database. Некоторые поля и методы этого класса были отредактированы так, чтобы он наследовался от AbstractRelationData и использовал его методы. Для этого были заменены несколько методов в Database на уже существующие, аналогичные методы в AbstractRelationData. Интегрированный класс представления данных называется CFDRelationData.

От класса Database в оригинальном коде наследуется класс DatabaseReader, методы которого служат для чтения табличных данных из файла и для создания объекта класса Database из этих данных (из std::ifstream либо из std::string). Также в самом классе Database тоже были методы, служащие только для создания объекта этого класса, т.е. предназначенные только для методов класса DatabaseReader. Это несоответствие принципу S в SOLID.

В Desbordante иная стратегия создания объекта класса представления данных. В каждом классе представления данных есть статический метод CreateFrom(), в который передаются параметры, нужные для создания объекта этого класса. Возвращает этот метод unique_ptr на созданный объект класса представления данных. Также, в этот метод как параметр передаётся ссылка на объект класса, который поддерживает интерфейс IDatasetStream. Этот интерфейс заменяет работу с файловым вводом напрямую. Он поддерживает проверку на наличие header'а в таблице, проверку на конец содержимого в файле, предоставляет доступ к названию столбца и таблицы и к списку элементов в строке таблицы.

В методах FromTable() и FromTablePart() класса DatabaseReader

сначала создаётся объект класса `Database`, поля которого постепенно заполняются на основе данных из файла. Однако, так как `CreateFrom()` находится непосредственно в `CFDRelationData`, то нельзя в этом методе создавать объект `CFDRelationData`. Методы `FromTable()` и `FromTablePart()` из `DatabaseReader` были отредактированы так, чтобы они:

1. Являлись двумя перегрузками метода `CreateFrom()`, который находится в `CFDRelationData`;

2. Использовали объект класса `IDatasetStream` вместо работы с потоком файла напрямую;

3. Не создавали объект класса `CFDRelationData` внутри метода, а создавали отдельно все нужные объекты для создания этого класса и в конце через `std::make_unique<CFDRelationData>()` возвращали `unique_ptr` на заполненный объект класса `CFDRelationData`. Для этого также пришлось сделать конструктор в классе `CFDRelationData` от всех нужных этому классу параметров.

В оригинальном коде параметры алгоритма передаются через консоль. В `main()` сначала создаётся объект класса `CFDDiscovery` с помощью конструктора, который принимает файл таблицы. Далее параметры алгоритма проходят проверку на правильность и передаются напрямую в нужный метод алгоритма внутри созданного объекта `CFDDiscovery`. После завершения работы алгоритма найденные CFD накапливаются в отведённом поле объекта `CFDDiscovery`.

В Desbordante все алгоритмы унаследованы от абстрактного класса `Primitive`. В нём представлен основной интерфейс, который должен поддерживать каждый алгоритм. В нём описан способ получения параметров, нужных алгоритму, и последовательность выполнения алгоритма. Есть методы `FitInternal()` и `ExecuteInternal()`, которые служат для заполнения объекта класса представления данных и для запуска алгоритма соответственно. Есть методы `SetOption()`, `RegisterOption()`, `MakeOptionsAvailable()` и т.д., служащие для получения параметров алгоритма извне и управления доступом к ним.

В Desbordante более сложная работа с входными параметрами алгоритмов. Каждому входному параметру алгоритма соответствует своя опция. Опция представляет собой класс, который хранит строковое описание параметра, его строковое название и само значение параметра. Есть обязательные и необязательные опции. Также есть опции, общие для всех алгоритмов (например, путь к файлу таблицы и название зависимости, которую надо найти).

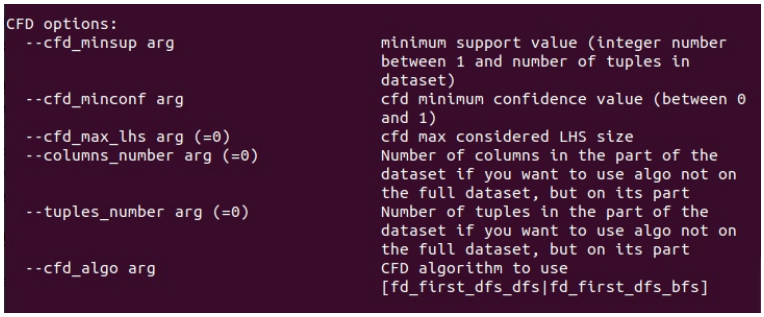

Рис. 3: Имена и описания опций для интегрированного алгоритма

В `main()` описаны все возможные опции для всех алгоритмов, которые пользователь может ввести через консоль. После ввода нужных параметров опции выбранного алгоритма заполняются этими введёнными значениями и передаются в класс нужного алгоритма.

В интегрированном классе `CFDDiscovery` тоже были созданы все нужные для алгоритма опции. Они заполняются с помощью метода `RegisterOptions()`, который использует метод класса `Primitive` — `RegisterOption()`. Далее были переопределены методы `FitInternal()` и `ExecuteInternal()`. В методе `FitInternal()` сначала идёт проверка на существование файла таблицы. Если файл существует, то вызывается функция `CreateFrom()` для объекта класса `CFDRelationData`. В методе `ExecuteInternal()` находится запуск алгоритма `FD-First-DFS`. Также был создан метод `CheckForIncorrectInput()`, который проверяет полученные параметры на правильность и вызывается прямо перед запуском алгоритма в функции `ExecuteInternal()`. На следующей странице приведена диаграмма классов, связанных с интегрированным алгоритмом. Интегрированный алгоритм можно увидеть по ссылке https://github.com/Mstrutov/Desbordante/tree/edbt/src/algorithms/cfd.

Рис. 4: Диаграмма классов интегрированного алгоритма

- Созданный класс/метод/поле с нуля
- Используемый класс из Desbordante без изменений
- Используемый класс из [4] без изменений
- Отредактированный метод/поле из [4]. Может иметь другое название

4. Эксперимент

Чтобы проверить правильность интеграции алгоритма нужно было сравнить результаты работы оригинального Fd-First алгоритма и его интегрированной версии в Desbordante, а также их время работы и убедиться, что оно не сильно отличается.

Для этого было решено написать модульные (unit) тесты для интегрированного кода при разных входных параметрах. В качестве ожидаемых значений были взяты результаты оригинального алгоритма [4] на тех же параметрах.

Ожидается, что интегрированный алгоритм:

- При одинаковых входных параметрах имеет те же самые результаты, что и оригинальный.
- Выдаёт результат за примерно то же время, что и оригинальный.

Каждый алгоритм после завершения работы выводил в консоль количество затраченных на работу миллисекунд, которое и использовалось при сравнении быстродействия алгоритмов.

Результаты интегрированного и оригинального алгоритмов полностью совпадают на датасетах разной величины.

В Таблице 2 приведены результаты тестирования. Integrated в колонке Algo значит, что тест проводился на интегрированном коде алгоритма, Original — на оригинальном коде алгоритма. Размер датасета mushroom ≈374КБ. Размер датасета adult ≈5МБ.

Таблица 2: Результаты тестирования оригинального и интегрированного алгоритмов

Dataset	Min supp	Min Conf	Max lhs	Diff, %
mushroom	8	0.85	3	0.1
adult	8	0.85	3	3.1
adult	8	0.9	4	2.5

Из таблицы видно, что интегрированный и оригинальный алгоритмы отличаются во времени работы в среднем не больше, чем на три процента. Заметно, что время работы алгоритма на датасете mushroom в три раза больше, чем время на датасете adult, хотя размер датасета adult практически в 14 раз больше, чем mushroom. Это из-за того, что в mushroom в два раза больше атрибутов, чем в adult, и поэтому есть в несколько раз больше вариантов перестановок атрибутов в потенциальной CFD. По той же причине время работы алгоритма сильно возросло на датасете adult при увеличении параметра `max_lhs` на единицу.

Заключение

Алгоритм FD-First-DFS из [4] был интегрирован в проект Desbordante. К нему были написаны модульные тесты, где проводилось сравнение интегрированного алгоритма с оригинальным.

Результаты:

1. В ходе изучения литературы было выяснено, что алгоритм FD-First демонстрирует наибольшую эффективность в общем случае. Он был изучен автором настоящей работы и было произведено описание принципа его функционирования.

2. Код алгоритма FD-First-DFS был полностью интегрирован так, чтобы он был завязан на используемых в проекте абстрактных классах и чтобы можно было запускать алгоритм через консоль. Класс алгоритма наследуется от `Primitive`, а класс представления данных наследуется от `AbstractRelationData`.

3. Были написаны модульные тесты для интегрированного кода. Результаты оригинального и интегрированного алгоритмов совпадали на одинаковых входных параметрах и отличались во времени не более, чем на три процента.

Код интегрированного алгоритма доступен по ссылке https://github.com/Mstrutov/Desbordante/tree/edbt/src/algorithms/cfd.

Список литературы

[1] Conditional functional dependencies for capturing data inconsistencies / Wenfei Fan, Floris Geerts, Xibei Jia, Anastasios Kementsietsidis // ACM Trans. Database Syst. — 2008. — 06. — Vol. 33.

[2] Discovering Conditional Functional Dependencies / Wenfei Fan, Floris Geerts, Jianzhong Li, Ming Xiong // IEEE Transactions on Knowledge and Data Engineering. — 2011. — Vol. 23, no. 5. — P. 683–698.

[3] Quinlan J. R. Induction of decision trees // Machine Learning. — 1986. — Mar. — Vol. 1, no. 1. — P. 81–106. — URL: https://doi.org/10.1007/BF00116251.

[4] Rammelaere Joeri. Implementation code of algorithms in [5]. — 2018. — URL: https://codeocean.com/capsule/6146641/tree/v1.

[5] Rammelaere Joeri, Geerts Floris. Revisiting Conditional Functional Dependency Discovery: Splitting the "C" from the "FD" // Machine Learning and Knowledge Discovery in Databases / Ed. by Michele Berlingerio, Francesco Bonchi, Thomas Gärtner et al. — Cham : Springer International Publishing, 2019. — P. 552–568.