2017

SEIM

Second Conference on

Software Engineering and Information Management

FULL PAPERS

Second Conference on Software Engineering and Information Management (SEIM-2017)

(full papers)

Saint Petersburg, April 21, 2017

George Chernishev, Marat Akhin, Boris Novikov, Vladimir Itsykson (editors)

Saint Petersburg — 2017

Second Conference on Software Engineering and Information Management (SEIM-2017)
Saint Petersburg, April 21, 2017

George Chernishev, Marat Akhin, Boris Novikov, Vladimir Itsykson (editors)

This volume contains eight selected papers originally presented at the Second Conference on Software Engineering and Information Management (SEIM-2017), which was held in Saint Petersburg, Russia, on April 21, 2017. These papers were selected in thorough single-blind reviewing process.

ISBN 978-5-9906498-3-5

Originally published online by CEUR Workshop Proceedings
(CEUR-WS.org, ISSN 1613-0073)

Формат 215,9x279,4. Печать цифровая.
Тираж 70 экз. Заказ №205568.
Отпечатано в ООО "Цифровая фабрика "Быстрый Цвет".
196066, Санкт-Петербург, ул. Варшавская, 98.
(812) 644-40-44

Table of Contents

Message from the Editors

The Second Conference on Software Engineering and Information Management (SEIM-2017) aims to bring together students, researchers and practitioners in different areas of software engineering and information management. We consider SEIM-2017 to be a stepping stone for young researchers, which should help them familiarize with the conference workflow, practice writing academic papers, gather valuable feedback about their research and expand their research network. The conference welcomes submissions on a wide range of topics, including but not limited to:

- Algorithms and data structures
- Cloud systems
- Coding theory
- Compilers
- Crowdsourcing
- Data storage and processing
- Development management
- Digital signal processing
- Distributed systems
- E-commerce / e-government
- Empirical software engineering
- High-performance computing
- Information retrieval
- Information security
- Intelligent data analysis
- Internet of Things
- Machine learning
- Mobile systems
- Modelling
- Natural language processing
- Networks and telecommunications
- (Non-)relational databases
- Operating systems
- Programming languages
- Recommendation systems
- Robotics
- Semantic web
- Social networks
- Software analysis
- Software testing
- Software verification
- Software virtualization
- Software-defined networks
- Theoretical computer science

In total, we received 35 papers, each reviewed by at least 3 members of the Program Committee, of which 8 were selected for publication in CEUR-WS.org, 8 — for indexing in RSCI, and 4 were accepted as talk-only to allow the young authors to experience the process of a scientific conference. We would like to thank the members of our Program Committee for their great work and contribution to the success of our conference!

These proceedings include the SEIM-2017 papers, which were selected by the Program Committee for publication in CEUR-WS.org. These papers passed not only the original review procedure, but also an additional round of post-review with the conference feedback. We thank the authors for their submissions to SEIM 2017 and hope to see them in the future!

Furthermore, we would also like to thank Tatiana Mironova and Sergey Zherevchuk for their great help in organizing the conference, Computer Science Center for hosting the event, and JetBrains Research for their overall support of this endeavour! The additional information about the SEIM conference series can be found on the conference website at: `http://2017.seim-conf.org/`

George Chernishev, Marat Akhin, Boris Novikov, Vladimir Itsykson
Editors

SEIM 2017 Organization

The conference was organized jointly with Computer Science Center and supported by JetBrains Research.

Steering Committee

Dmitry Bulychev / St. Petersburg State University
Vladimir Itsykson / St. Petersburg Polytechnic University
Andrey Ivanov / JetBrains
Iakov Kirilenko / St. Petersburg State University
Kirill Krinkin / St. Petersburg Electrotechnical University
Boris Novikov / St. Petersburg State University

Program Committee Chairs

George Chernishev / St. Petersburg State University
Marat Akhin / St. Petersburg Polytechnic University

Program Committee

Marat Akhin / St. Petersburg Polytechnic University
Dmitry Barashev / St. Petersburg Academic University
Alexander Batyukov / St. Petersburg State University
Mikhail Belyaev / St. Petersburg Polytechnic University
Natalia Bogach / St. Petersburg Polytechnic University
Dmitry Boulytchev / St. Petersburg State University
Kirill Cherednik / St. Petersburg State University
George Chernishev / St. Petersburg State University
Mikhail Glukhikh / JetBrains
Oleg Granichin / St. Petersburg State University
Natalia Grafeeva / St. Petersburg State University
Dmitry Grigoriev / St. Petersburg State University
Semyon Grigorev / St. Petersburg State University
Vladimir Itsykson / St. Petersburg Polytechnic University
Aleksandr Ivanov / DSX Technologies
Iakov Kirilenko / St. Petersburg State University
Kirill Krinkin / Open Source and Linux Lab
Svetlana Lazareva / Raidix
Yurii Litvinov / St. Petersburg State University
Dmitry Luciv / St. Petersburg State University
Igor Malyshev / St. Petersburg Polytechnic University
Elena Mikhailova / St. Petersburg State University
Mikhail Moiseev / Intel Corporation
Slava Nesterov / Dell EMC
Boris Novikov / St. Petersburg State University
Gennady Pekhimenko / Microsoft Research
Oleg Pliss / Oracle
Sergey Salishev / St. Petersburg State University
Alexander Shalimov / Lomonosov Moscow State University
Elena Sivogolovko / Arcadia
Kirill Smirnov / St. Petersburg State University
Sergey Stupnikov / Institute of Informatics Problems RAS
Maksim Tkatchenko / Singapore Management University
Peter Trifonov / St. Petersburg Polytechnic University
Andrey Vlasovskikh / JetBrains
Anna Yarygina / St. Petersburg State University
Mikhail Zymbler / South Ural State University

DeepAPI#: Синтез цепочки вызовов API CLR/C# по текстовому запросу

Александр Чебыкин
Санкт-Петербургский
Государственный Университет
Email: a.e.chebykin@gmail.com

Михаил Кита
Санкт-Петербургский
Государственный Университет
Email: mihail.kita@gmail.com

Яков Кириленко
Санкт-Петербургский
Государственный Университет
Email: jake.kirilenko@gmail.com

Аннотация—Разработчики часто ищут способы реализовать стандартную функциональность с помощью библиотечных функций (например, создать кнопку в UI или получить данные из формата JSON). Обычно источником такой информации является Интернет. Альтернатива - различные статистические инструменты, которые, обучившись на большом объеме кода, по текстовому запросу могут предоставлять пользователю набор вызовов функций, решающих задачу.

Мы рассматриваем один из таких инструментов — DeepAPI. Этот современный алгоритм основывается на глубоком обучении, и, согласно его авторам, работает лучше аналогов. Мы пытаемся воспроизвести этот результат, взяв целевым языком не Java, как оригинальный DeepAPI, а C#. В данной статье мы рассказываем о возникающих проблемах сбора данных для обучения, сложностях в построения и обучения модели, а также обсуждаем возможные модификации алгоритма.

I. Введение

Последние несколько десятков лет предпринимаются попытки анализировать исходный код для разных целей: генерация имени метода по его телу [1], анализ качества кода [2], генерация кода по текстовому запросу [3], [4]. Последнее особенно интересно, поскольку позволяет разработчикам избежать долгого изучения программных библиотек в поисках нужной функциональности, что является значимой проблемой [5].

Обычно разработчики ищут способы реализовать стандартную функциональность с помощью поисковых движков в сети Интернет, для чего те не оптимизированы [6]. Исследователями предлагались разные замены им, в том числе инструменты, основывающиеся на статистическом анализе большой кодовой базы. Яркие представители:

- MAPO[4] генерирует по имени функции наиболее релевантные шаблоны её использования;
- UP-Miner [7] — улучшенная версия MAPO, достигающая лучших результатов в генерации благодаря более сложному алгоритму;
- SWIM[3] генерирует код по текстовому запросу.

Одним из самых свежих алгоритмов в этой области является DeepAPI [8] — алгоритм из статьи строит последовательность вызовов функций по текстовому запросу, достигая лучших результатов, чем SWIM, за счёт использования моделей глубокого машинного обучения из области обработки естественных языков. Глубокое обучение успешно используется для перевода между натуральными языками, DeepAPI пользуется результатами последних исследований в этой области. Задача генерации кода по тексту здесь рассматривается как задача перевода с английского языка на язык вызовов функций (слова в таком языке — функции языка программирования, предложения — упорядоченные наборы слов).

Результаты, представленные в статье об алгоритме DeepAPI, привлекли наше внимание, и мы решили повторить этот опыт и расширить его. В результате работы оригинального алгоритма DeepAPI получается линейная последовательность вызовов функций, код с их применением программисту приходится писать самостоятельно. Хотелось бы облегчить ему этот этап, предлагая не список функций, а сниппет кода — то есть отрывок кода на языке программирования, в котором нужно модифицировать минимальное количество деталей для того, чтобы он заработал. Для реализации этого может оказаться полезным алгоритм T2API [9], который так же является достаточно свежим исследованием возможности генерации кода по текстовому запросу. В первой части алгоритма по текстовому запросу выбираются некоторые релевантные элементы API, во второй — из них генерируются граф управления и код. Теоретически, замена первой части алгоритма на DeepAPI позволит сделать итоговые результаты более точными, так как модель, используемая на первом шаге T2API, более старая, а также не упорядочивает элементы API на выходе.

На текущий момент мы пытаемся повторить эксперимент из статьи DeepAPI и реализовать алгоритм из неё на базе другого языка программирования. В оригинальной статье авторы работают с элементами API и кодовой базой языка Java и упоминают, что одна из угроз действительности эксперимента — работа только с одним языком. Мы желаем нивелировать эту угрозу, а потому работаем с языком C#.

Наш текущий вклад в исследования можно описать следующими тезисами:

- собран набор тренировочных данных, реализован

инструментарий для сбора данных;
- проведено исследование использования в качестве тренировочных данных данные, отличные от рассмотренных в оригинальной статье;
- построена аппроксимация алгоритма DeepAPI;
- поставлен частично успешный первичный эксперимент.

II. DeepAPI

Модель DeepAPI основывается на современных техниках глубокого обучения и машинного перевода. Ключевая техника — обучение Sequence-to-Sequence [10], суть которой — по входной строке генерировать выходную. При этом обычно первая строка принадлежит одному естественному языку, вторая — другому. В случае DeepAPI исходный язык — английский, выходной — язык элементов API.

Техника Sequence-to-Sequence включает в себя две рекуррентные нейронные сети (рекуррентная нейронная сеть (RNN) — один из подвидов моделей глубокого обучения). Кодер — первая из этих сетей — поэлементно считывает вход и обновляет свои параметры в текущем скрытом состоянии. После окончания считывания последнее состояние — предполагается, что в нём заключена суть входного предложения — берётся в качестве вектора контекста, и передаётся в качестве входа второй RNN — декодеру. Декодер на каждом шаге, сверяясь с вектором контекста, генерирует очередной элемент выходной последовательности, и обновляет своё состояние. Работа декодера прекращается, когда он генерирует слово, означающее окончание строки — <EOS>.

Пример работы модели представлен на рисунке 2. Входной поток — “generate random number”, выходной — “Random.new Random.nextInt”.

На рисунке для простоты понимания состояния расширены по времени: поскольку нейронная сеть рекуррентная, переход из состояния должен вести в него же. То есть здесь h1, h2, h3 — одно и то же состояние в моменты времени 1, 2, 3.

В модели DeepAPI также используется улучшение техники Sequence-to-Sequence: механизм внимания. Вместо использования лишь последнего состояния кодера в качестве вектора контекста для генерации целого предложения, на каждом шаге работы декодера используется свой вектор контекста, получаемый как взвешенная сумма всех состояний кодера

$$c_j = \sum_{t=1}^{T} a_{jt} h_t$$

где c_j — вектор контекста для шага j, h_t — исторические состояния кодера, a_{jt} — веса этих состояний для шага j (моделируются при помощи отдельной заранее натренированной нейронной сети).

Для увеличения точности работы алгоритма вводится еще одно улучшения — новая функция потерь, включающая в себя наказание за использование чересчур частых элементов API. Ведь если функция часто используется, то скорее всего, она не связана с реализацией конкретной функциональности, а потому считается шумом. Оценка важности элемента API основывается на tf–idf [11], и выглядит так:

$$w_{idf}(y_t) = log(N/n_{y_t})$$

где N — общее число экземпляров в тренировочном наборе, n_{y_t} — число экземпляров, в которых встречается y_t.

Так как DeepAPI должен переводить запросы на английском языке в цепочки вызовов функций, она должна тренироваться на примерах того, какие английские предложения каким цепочкам вызовов функций соответствуют. Такие примеры авторы оригинальной статьи извлекают из открытого исходного кода, а именно из методов с комментариями. Так как целевым языком программирования выбрана Java, имеющая зафиксированную структуру комментария для документации Javadoc, авторы пользуются этим, и берут первое предложение комментария в качестве описания метода на английском языке (по определению Javadoc, именно это написано в первом предложении). Чтобы получить список вызовов функций, код метода анализируется компилятором Eclipse JDT. Выводятся классы каждой переменной, вызовы методов записываются вместе с именем класса, которому этот метод принадлежит. Пример вычленения информации из кода представлен на рисунке 1.

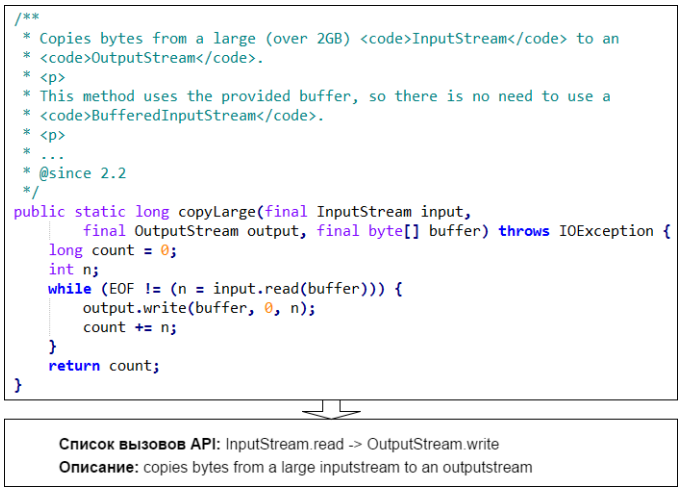

Рис. 1: Пример получения данных из кода

После обучения, во время генерации результата по запросу, авторами используется лучевой поиск — вместо того, чтобы генерировать самое вероятное слово на каждом шаге, поддерживается список из n текущих самых вероятных слов, и генерация продолжается для

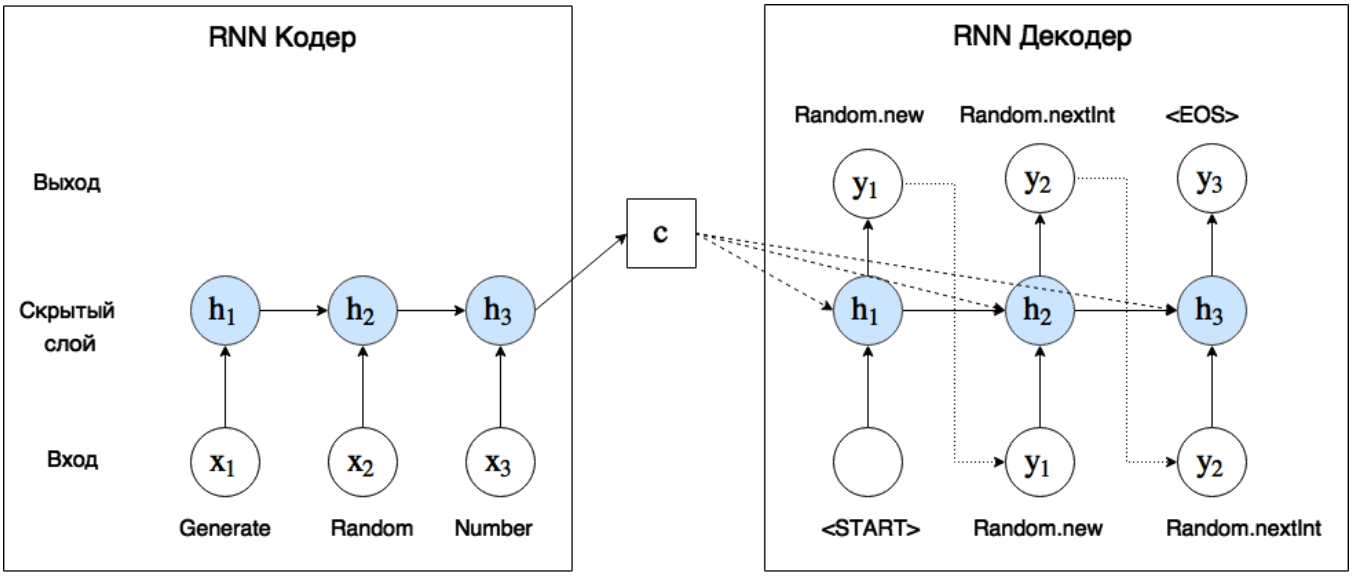

Рис. 2: Пример работы RNN Кодер-Декодера

каждого из них. Другие результаты отсекаются. Цель подобной практики — предотвратить случаи, когда наиболее точная цепочка отбрасывается из-за того, что её первое слово оказалось не самым подходящим. В итоге для каждого запроса получаем список из нескольких возможных ответов.

Для человеконезависимой оценки качества работы алгоритма используется метрика BLEU. BLEU часто применяется для оценки точности машинного перевода, она измеряет, насколько сгенерированная последовательность близка к той, что была написана человеком и использовалась во время обучения. Шкала оценок идёт от 0 до 100.

По представленным в статье оценкам, DeepAPI превосходит предыдущие техники. Поскольку не у всех из них была подсчитана оценка BLEU, авторы DeepAPI реализовали и оценили прошлые алгоритмы самостоятельно. В итоге UP-Miner получил оценку 11.97, SWIM — 19.90, DeepAPI — 54.42.

Имеется также рабочий прототип [12], относительно правдоподобно отвечающий на запросы.

III. Получение данных

Отдельное внимание стоит уделить процессу сбора данных для исследования и тренировки модели. Для получения большего количества данных, мы использовали не только скачивание кода открытых проектов с Github, но и обработку скомпилированных сборок проектов, полученных из репозитория NuGet. В следующих двух секциях мы рассматриваем эти способы подробнее.

A. NuGet

Первый подход заключался в извлечении нужных данных из скомпилированных библиотек, которые в большом количестве доступны в репозитории NuGet [13]. Для улучшения качества данных пакеты рассматривались в порядке убывания популярности: мы исходили из предположения, что наиболее популярные пакеты содержат более качественный код. Пакеты скачивались в автоматическом режиме, после чего содержащиеся в них библиотеки собирались в одну папку, чтобы обеспечить разрешение зависимостей. В случае возникновения коллизии имён предпочтение отдавалось более поздней версии библиотеки. Помимо самих библиотек также собирались и все доступные xml-файлы, содержащие комментарии к методам.

Далее следовал процесс обработки собранных файлов: для каждого извлечённого метода генерировалось синтаксическое дерево, а затем производился его разбор. Описанные действия позволили получить исчерпывающую информацию о каждом вызове метода в коде, включая тип объекта, имя метода и параметры. Эта информация использовалась для построения последовательности вызовов, которая затем сохранялась на диск для дальнейшего использования.

Помимо последовательностей вызовов необходимо было также собрать комментарии к методам. На этом этапе мы столкнулись с проблемой: количество методов с комментариями в скачанных нами пакетах составляет лишь 11.21% от числа всех методов. Были собраны все доступные комментарии из xml-файлов, а для оставшихся методов мы генерировали описания, используя имя данного метода и имя содержащего его класса.

Как показала практика, описанный метод сбора данных оказался достаточно эффективным. Было скачано 4,100 пакетов, содержащих в общей сложности 4,278 библиотек. Итоговое количество собранных методов составило 611,945, из них 68,585 — с комментариями.

B. Github

GitHub [14] — популярный хостинг проектов с открытым исходным кодом. Именно отсюда авторы оригинальной статьи скачали 442,928 проектов на Java. Для фильтрации проектов авторы выставили условие, что у каждого из них должна быть хотя бы одна звезда.

По подобному запросу для языка C# выдаётся 121,672 репозитория (https://github.com/search?utf8=%E2%9C%93&q=language%3AC%23+stars%3A%3E0&type=Repositories&ref=searchresults).

Таким образом, по сравнению с Java, репозиториев, с которыми можно потенциально работать, примерно в 4 раза меньше. Однако судя по имеющимся у нас данным, кажется возможным собрать количество данных того же порядка. Авторы в итоге получили 7,519,907 пар "описание на естественном языке — список вызовов API". После обработки 48,789 репозиториев мы получаем 870,008 пар.

Помимо количества репозиториев, возникает проблема из-за выбранного целевого языка. В Java из исходного кода можно легко получить тип (или надтип) переменной из её объявления. Однако в языке C#, начиная с версии 3.0, переменные могут иметь неявный тип var, поэтому для выведения явного типа таких переменных требуется компиляция всего проекта. Это ограничивает количество репозиториев, которые мы можем обработать. Кроме этого, мы сталкиваемся и с другими сложностями со стороны языка, например, динамический тип данных dynamic, о котором во время компиляции мы ничего не знаем. Но такие проблемы по сравнению с проблемой неявных типов несущественны: мы провели небольшой эксперимент, запустив поиск в исходном коде 470 случайно выбранных репозиториев слова "dynamic"и "var и получив всего 4,856 результатов для первого, и 176,561 результатов для второго. Поэтому пока что мы не исследуем способы решения подобных неприоритетных проблем, оставляя это на будущее.

В качестве компилятора мы используем Roslyn — компилятор C# с открытым исходным кодом, разработанный Microsoft. Чтобы собирать проекты в автоматическом режиме, необходимо, чтобы:

1) не надо было предпринимать никаких дополнительных действий (например, запускать индивидуальные для каждого проекта скрипты);
2) проект содержал файл с расширением .sln — именно с ним может работать Roslyn.

Из 100 самых популярных результатов поиска таким ограничениям удовлетворяет 51 проект, в среднем по всем результатам поиска — 11.2%.

Код из подходящих репозиториев мы обрабатываем аналогично оригинальной статье.

IV. Эксперимент

На текущий момент мы не занимались реализацией упомянутой ранее модификации функции потерь, наказывающую генерацию слишком частых функций. Это изменение не было приоритетным, т.к. влияет на оценку BLEU не больше, чем на 2 очка, увеличивая её с 52.49 до 54.42, а для нас более интересно получить первичный результат примерно того же порядка, что и 52.49.

A. Первый эксперимент

В качестве функции потерь в модели DeepAPI берётся условная логарифмическая функция правдоподобия. Модель тренируется максимизировать её с помощью стохастического градиентного спуска [15] в комбинации с оптимизатором AdaDelta [16]. Авторы реализуют модель на базе библиотеки GroundHog. Однако он устарел и более не поддерживается, поэтому мы реализуем нашу первую модель на основе TensorFlow — популярной библиотеки для машинного обучения с открытым кодом.

Надо упомянуть, что обучение рекуррентных нейронных сетей очень ресурсоёмко, а потому ведётся на видеокартах. В то время как авторы для тренировки пользовались видеокартой Nvidia K20, у нас имеется гораздо менее мощная Nvidia GTX 660. В связи с этим нам пришлось уменьшить параметры тренировки до менее оптимальных. Например, авторы обнаружили, что оптимальное количество нейронов в скрытом слое кодера и декодера — 1000. Мы выставляем их количество в 700. Авторы не обсуждают влияние размера пакета (batch) на модель, выставляя его значение в 200. Мы делаем его размер равным 32.

Мы вводим дополнительное улучшение в виде bucketing — разделения исходных предложений на классы по длине, и тренировки внутри этих классов. Дело в том, что для быстрого обучения нам нужны вектора фиксированной длины — а предложения имеют длину разную. Можно их все дополнять до максимальной длины (с помощью специального слова), а можно разделить на категории вида длина < 10, длина < 20, и т.д., и дополнять до максимума в этой категории, тренируя категории отдельно. Чем меньше дополняем пустыми словами, тем меньше шума, и лучше (теоретически) результат.

Мы тренируем модель на 605,146 парах данных в течение 40,000 итераций, ограничив словарь самыми популярными 10,000 слов в исходном и выходном языках.

Итоговая модель умеет верно отвечать на некоторые запросы, например 'generate random number — System.Random.new System.Random.Next' или 'replace part of string with other string — System.String.Replace', но на большинстве входов выдаёт нерелевантный результат.

B. Второй эксперимент

Так как хороших результатов получить на модели с урезанными параметрами не удалось, мы создаём

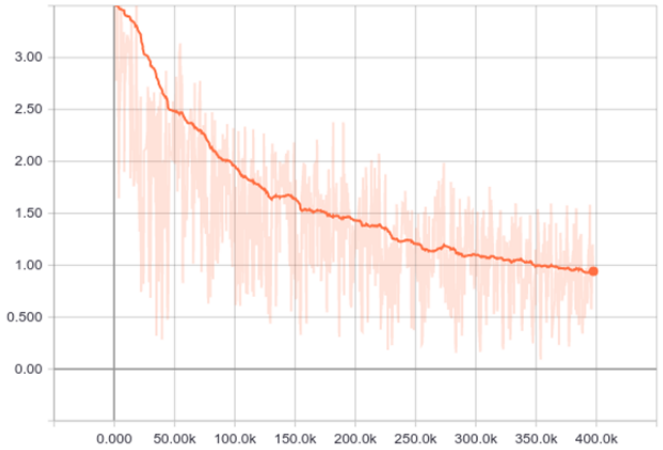

Рис. 3: График изменения функции потерь во время обучения

модель на основе нового фрэймворка от Google - tf-seq2seq, специализирующегося на Sequence-to-Sequence моделях. С его помощью мы обучаем модель на тех же значениях параметров, что и в оригинальной статье, при этом для этого хватает производительности нашей не мощной видеокарты. Также во время декодирования мы используем лучевой поиск - улучшение, предложенное авторами оригинальной статьи и описанное в секции II; в предыдущей модели мы этот механизм не реализовывали.

Модель мы тренируем на 924,593 парах данных в течение 390,000 итераций, ограничивая словарь в исходном и выходном языках самыми популярными 10,000 слов.

Итоговая модель может не только отвечать на одиночные запросы, как старая, но и показывает ненулевые результаты на тестовом множестве из 14,000 пар данных. Надо упомянуть, что пары из тестового множества не используются во время тренировки, поэтому положительные результаты на них означают, что модель не просто запомнила данные ей в тренировочном множестве примеры, а действительно научилась обобщать знания. Оценка модели по метрике BLEU растёт от 0.48 после 158,000 итераций до 1.95 после 390,000 итераций. Помимо этого, искусственная функция потерь, которую модель минимизирует, стабильно уменьшается (см. рисунок 3).

Таким образом, из общей положительной динамики мы можем заключить, что техника имеет право на существование. Мы считаем, что тренировка на большем количестве данных позволит получить сравнимый с оригинальным результат.

V. Дальнейшая работа

Мы планируем продолжать исследование данного подхода. В ближайшей перспективе перед нами стоят следующие цели:

1) собрать набор данных, сопоставимый по размеру с набором данных в оригинальной статье;
2) поэкспериментировать с параметрами обучения, и найти оптимальные;
3) исследовать возможность тренировки на не комментированных методах, используя имена и классы параметров, имя метода;
4) найти способы решения специфичных для целевого языка проблем;
5) реализовать алгоритм T2API;
6) объединить алгоритмы DeepAPI и T2API в один, проанализировать получившийся результат.

VI. Заключение

В этой статье мы описали нашу частично успешную попытку повторить результат статьи DeepAPI. Используя современные методы глубокого обучения и машинного перевода, мы достигли небольших успехов в задаче генерации списка вызовов функций по текстовому запросу.

Мы считаем, что модель DeepAPI действительно может быть реализована на базе другого языка, а также улучшена, что мы и собираемся продемонстрировать по результатам дальнейшей работы.

Список литературы

[1] M. Allamanis, H. Peng, and C. Sutton, “A convolutional attention network for extreme summarization of source code.”
[2] V. Barstad, M. Goodwin, and T. Gjøsæter, “Predicting source code quality with static analysis and machine learning.” in NIK, 2014.
[3] M. Raghothaman, Y. Wei, and Y. Hamadi, “Swim: synthesizing what i mean: code search and idiomatic snippet synthesis,” in Proceedings of the 38th International Conference on Software Engineering. ACM, 2016, pp. 357–367.
[4] T. Xie and J. Pei, “Mapo: Mining api usages from open source repositories,” in Proceedings of the 2006 international workshop on Mining software repositories. ACM, 2006, pp. 54–57.
[5] M. P. Robillard and R. Deline, “A field study of api learning obstacles,” Empirical Software Engineering, vol. 16, no. 6, pp. 703–732, 2011.
[6] J. Stylos and B. A. Myers, “Mica: A web-search tool for finding api components and examples,” in Visual Languages and Human-Centric Computing, 2006. VL/HCC 2006. IEEE Symposium on. IEEE, 2006, pp. 195–202.
[7] J. Wang, Y. Dang, H. Zhang, K. Chen, T. Xie, and D. Zhang, “Mining succinct and high-coverage api usage patterns from source code,” in Proceedings of the 10th Working Conference on Mining Software Repositories. IEEE Press, 2013, pp. 319–328.
[8] X. Gu, H. Zhang, D. Zhang, and S. Kim, “Deep api learning,” in Proceedings of the 2016 24th ACM SIGSOFT International Symposium on Foundations of Software Engineering. ACM, 2016, pp. 631–642.
[9] T. Nguyen, P. C. Rigby, A. T. Nguyen, M. Karanfil, and T. N. Nguyen, “T2api: synthesizing api code usage templates from english texts with statistical translation,” in Proceedings of the 2016 24th ACM SIGSOFT International Symposium on Foundations of Software Engineering. ACM, 2016, pp. 1013–1017.
[10] I. Sutskever, O. Vinyals, and Q. V. Le, “Sequence to sequence learning with neural networks,” in Advances in neural information processing systems, 2014, pp. 3104–3112.
[11] G. Salton and M. J. McGill, “Introduction to modern information retrieval,” 1986.

[12] H. K. U. of Science and Technology. (2017) Deepapi. [Online]. Available: http://www.cse.ust.hk/~xguaa/deepapi/
[13] Nuget gallery. [Online]. Available: http://www.nuget.org
[14] Github. [Online]. Available: https://github.com
[15] L. Bottou, "Large-scale machine learning with stochastic gradient descent," in Proceedings of COMPSTAT'2010. Springer, 2010, pp. 177–186.
[16] M. D. Zeiler, "Adadelta: an adaptive learning rate method," arXiv preprint arXiv:1212.5701, 2012.

DeepAPI#: CLR/C# call sequence synthesis from text query

Alexander Chebykin, Mikhail Kita, Iakov Kirilenko

Developers often search for an implementation of typical features via libraries (for example, how to create a UI button control, extract data from a JSON-formatted file, etc.). The Internet is the usual source of the information on the topic. However, various statistical tools provide an alternative: after processing large amounts of source code and learning common patterns, they can convert a user request to a set of relevant function calls.

We examine one of those tools — DeepAPI. This fresh deep learning based algorithm outperforms all others (according to its authors). We attempt to reproduce this result using different target programming language — C# — instead of Java used in the original DeepAPI. In this paper we report arising problems in the data gathering for training, difficulties in the model construction and training, and finally discuss possible modifications of the algorithm.

Implementing Common Table Expressions for MariaDB

Galina Shalygina
Mathematics and Mechanics Department
Saint Petersburg State University
Saint-Petersburg, Russia
galashalygina@gmail.com

Boris Novikov*
Mathematics and Mechanics Department
Saint Petersburg State University
Saint-Petersburg, Russia
b.novikov@spbu.ru

***Abstract*—Common Table Expressions (CTEs), introduced in the SQL Standard 1999, are similar to subroutines in programming languages: they can be referenced from multiple places in a query and may refer to themselves, providing recursive queries. Many RDBMSs implemented this feature, yet not in full. Until recently MariaDB did not have this feature either. This article describes CTE overall and looks through some interesting cases that are implemented in MariaDB only, like non-linear recursion and mutual recursion. It also compares optimizations for non-recursive CTEs across different RDBMSs. Finally, the results of experiments comparing computation of recursive CTEs for MariaDB and PostgreSQL are presented.**

Keywords—common table expressions; optimization; MariaDB;

I. INTRODUCTION

In spite of the fact that first SQL Standard appeared in 1986, there was lack of standard recursive constructions for over a decade from this period. Nevertheless, as it was essential to write hierarchical queries in some cases, several DBMSs introduced their own recursive constructions. One of the most popular is CONNECT BY presented by Oracle in 1980's [6]. And still now, even after standard recursive construction common table expression (CTE) was officially introduced, there are many researches on presentation of recursive queries, especially on using Object-Relational Mapping to make recursion [9], [10], [11], [12].

The first attempt of CTE was in the SQL Standard 1999 [3] and from this version of Standard the CTE specification didn't change in the latest versions (see SQL:2008 [4]). The first implementation of CTEs dates back to 1997, to RDBMS DB2. Later there was a period of stagnation and only in 2003-2005 other RDBMSs started implementing CTE [5]. Nowadays CTE are supported by most major RDBMSs [7], but still none of them support all CTE features that are described in the SQL Standard 1999 [3]. Not so long ago MariaDB introduced its own implementation of CTE on which one of the authors has worked with other MariaDB developers [1], [2]. This implementation includes features that have never been introduced by other RDBMSs, like mutual recursion or non-linear recursion. A little later MySQL included an implementation of CTE in their code as well.

As soon as a CTE is defined in a query, this CTE may be used several times in the same query, thus the role of CTEs is similar to that of subroutines in imperative programming languages.

The SQL Standard requires that results of the query execution are the same as if CTEs were executed only once during the query processing. This requirement suggests a straightforward implementation of CTEs that computes a CTE as a separate query and materializes results in a temporary table. However, this implementation might result in a poor performance. For example, consider a CTE defining a large number of rows, say, extracting a table with millions rows. Such CTE may be invoked in a query with additional selection criteria restricting the size of output to a single row. If the same table expression was placed directly into FROM clause, an optimizer, most likely, would evaluate the condition first, avoiding calculation and materialization of unneeded rows.

An efficient implementation of CTE, even for non-recursive queries, is a non-trivial task.

These days more and more benchmarks include CTE usage. To compare, TPC-H 1999 Benchmark has no tests using CTE, while in TPC-H 2011 38 of 100 queries contain CTE [8]. To provide fast and low-cost CTE computation a lot of attention was given to researches on optimization techniques for CTE. Articles [13], [14], [15] provide such techniques for recursive CTEs. MariaDB also introduced its own optimization technique for non-recursive CTEs, that concerns non-mergeable views and derived tables.

In this article we will discuss the implementation of CTE that takes into account MariaDB Server special characteristics. We will also overview different optimization techniques for non-recursive CTEs. Non-recursive CTEs are handled as derived tables, but recursive ones are a much more difficult case. Moreover, they are computed in a different way.

Firstly, it should be checked if recursion is linear. Secondly, all mutual recursive CTEs are detected, if there are

*The work of the second author is supported by Academy of Finland and Russian Foundation for Basic research grant 16-57-48001

any. At the same time all Standard restrictions on CTEs are checked. As regards optimization techniques for non-recursive CTEs, we describe most popular of them with proper examples and compare different RDBMSs approaches.

The contributions of this work include:

- Techniques for efficient implementation of several special cases of CTEs
- Techniques for implementation of mutual recursion of CTEs
- An implementation of both non-recursive and recursive CTEs in MariaDB

This paper is organized as follows: Section 2 discusses non-recursive CTEs in general and Section 3 describes recursive ones. Also Section 3 shows how computation of recursive CTEs in MariaDB goes, looks through different recursion cases and demonstrates how recursion can be stopped. Section 4 compares optimizations in different RDBMSs and Section 5 presents the results of experiments on two RDBMSs MariaDB and PostgreSQL. In Section 6 a conclusion is made.

II. Non-recursive CTE

CTE can be introduced as a temporary result set that is defined within the scope of a single SELECT, INSERT, UPDATE, DELETE or CREATE VIEW statement. In MariaDB CTE can be defined only in SELECT or/and CREATE VIEW statements.

Each definition of non-recursive CTE consists of obligatory *WITH* keyword, the CTE name, an optional column list and a query specifying this CTE.

```
WITH expression_name [( column_name [,...] )]
AS <CTE_query_specification>
```

Non-recursive CTEs can be called 'query-local views' and are similar to derived tables. As well as derived tables they aren't stored in the database. This means that CTEs live only for the duration of the query where they are defined. However, unlike derived tables, they can be referenced multiple times in the same query. Instead of redefining the same derived table every time CTE can be used with the aim of making query more readable.

In MariaDB after CTE identification all references to non-recursive CTEs are formed as references to derived tables. On further phases of semantical analysis, optimization and execution non-recursive CTEs are handled as derived tables. The only thing that should be checked is if there are any renamed columns in the CTE definition because derived table doesn't have such an option.

III. Recursive CTE

The greatest advantage that using CTE provides is that it can reference to itself so a recursive query can be specified.

Each definition of a recursive CTE consists of obligatory *WITH RECURSIVE* keyword, the CTE name, an optional column list and seed and recursive parts specifying this CTE. Both seed part and recursive part can be defined by several SELECT statements and they should be joined by *UNION* or *UNION ALL* operations.

```
WITH RECURSIVE
expression_name [( column_name [,...] )]
AS ( [<seed_part>] UNION [ALL]
     <recursive_part> ) [,...]
```

A. Computation

At the first step all the components of the seed part are computed. At all further steps the recursive part is computing using the records produced by the previous steps. The Standard requires that only linear recursion can be used. This means that at each step of recursion only those records that are produced by the latest step can be used. The process of computation of recursive CTE stops when there are no new records produced.

In MariaDB computation of recursive CTE goes according to the following scenario:

At the preparatory stage the dependency matrix for all CTEs used in the query is built to detect recursive CTEs. Also at this stage it is checked if there are enough seed parts for recursive CTEs.

At the stage of the semantical analysis the structure of temporary tables where results will be saved is defined using the seed part of the query. Several temporary tables are created: the table where the final result records are accumulated, the table for the records produced by the latest step and the tables for each reference to the recursive CTE. Further all Standard restrictions are checked at this stage.

Lastly, at the execution stage CTE is executed in cycle using temporary tables defined at the previous stage. At the beginning these temporary tables contain the result of execution of the seed part. On each iteration the content of the table for new records is used as the entry for the recursive part. The result of the recursive part execution is added to the table where the final result is stored and the new produced records are written in the table for the new records. If there is no data in the table for the new records, the process stops.

As stated before, there is one temporary table for each reference to the recursive CTE. All these tables are similar to each other, but storing them all is caused by the current limitations of the MariaDB server. Later an optimization which solves this problem will be added to the server.

Also it must be mentioned that any recursive CTE is computed only once for the query.

B. Non-linear recursion

As it was said before, a non-linear recursion is forbidden by the Standard. However, MariaDB supports this feature.

A non-linear recursion can be useful when some restrictions of the Standard on recursive CTEs have to be lifted. So, non-linear recursion can be used when:

- there is more than one reference to recursive CTEs in a FROM clause of *recursive_part* of CTE;
- there are some references to recursive CTEs in the right part of LEFT JOIN or in the left part of RIGHT JOIN;
- there are some references to recursive CTEs in a subquery that is defined in a WHERE clause of some recursive CTE;

The main difference in computation of non-linear recursion from linear one is that on each iteration not only the records produced by the latest recursive step but all records produced so far for the CTE are used as the entries for the recursive part. So, in MariaDB implementation of CTEs when the restrictions are lifted, new records are just added to the tables created for the recursive references from the specification of CTE. The recursion stops when no new records produced.

Whereas in some cases a query looks cleaner and executing converges faster, usage of non-linear recursion can be very helpful in many cases. For example, it can be effectively used to stop the recursive process.

If the user wants to use non-linear recursion in MariaDB, he can set *@@standard_compliant_cte=0* and work with it.

C. Mutual recursion

Mutual recursion is said to be one of the most interesting forms of recursion. It is such a form where two or more CTEs refer to each other.

In MariaDB all mutual recursive CTEs are detected at the preparatory stage. For every *recursive_part* of CTE a search is made for all *recursive_parts* mutually recursive with the CTE where it was defined. It must be said that recursive CTE is mutually recursive with itself. All found mutually recursive CTEs are linked into a ring chain. Further, it is checked if there are enough *seed_parts* for a mutually recursive group. There should be at least one anchor for the mutually recursive group.

Mutual recursion is allowed by the Standard, but it required that any mutually recursive group of CTEs can be transformed into a single recursive CTE. An example of the non-restricted case of mutual recursion can be as follows: when there are two recursive CTEs, where on each iteration one CTE waits until the second one ends computation with the content of the first CTE as the entry, and only after that goes to the next step. MariaDB supports only mutual recursion as specified in the Standard. It also must be said that MariaDB is the first RDBMS that implemented mutual recursion and the only one who did it at the time of writing.

D. How recursion can be stopped

In MariaDB using linear recursion for traversal of a tree or a directed acyclic graph the execution is guaranteed to stop. However, in many other cases the user has to add some conditions to prevent loops.

When a transitive closure is computed, in the definition of recursive CTE only UNION can be used to join recursive and seed parts. For instance, when the user needs to find all cities that he can reach from some place by bus, there can be more than one bus route with the same point of arrival. If there are such routes and the user applies UNION ALL, some destinations will be added repeatedly and the routes will be added again and again. This will lead to an infinite process.

In the case when the paths over the graph with the loops need to be computed, for example, all paths consist of cities that can be reached by bus from some place, there might be a special condition written into WHERE in the recursive part of CTE definition to stop indefinitely increasing paths computing. As well as in the previous case there can be some bus routes with the same destination. Adding a city that already exists in the path will lead to an infinite process, that's why a condition that checks if the city exists in the path needs to be added.

Also in MariaDB there is a safety measure – the special variable *@@max_recursive_iteration*s that controls the count of the completed iterations during computation of a recursive CTE. The user can change it himself if needed.

IV. Optimization techniques

The basic algorithm of execution of a CTE stores results of CTE in a temporary table. When the query where the CTE was defined calls for the CTE results, the result records are taken from this temporary table. Although this algorithm always works, in most cases it is not optimal. Some optimization techniques on non-recursive CTEs are discussed and a comparison between different RDBMSs approaches is made below.

A. CTE merging

With this optimization applied the CTE is merged into parent JOIN so that parts of the CTE definition replace corresponding parts of the parent query. There are some restrictions on a CTE in order it could be merged: GROUP BY, DISTINCT, etc. can't be used in CTE definition.

This optimization technique is the same as ALGORITHM=MERGE for views in MySQL.

On the Fig. 1 the example of how this technique works is shown. The upper listing shows the initial query and the low shows how the optimizer will transform it.

```
WITH engineers AS (
 SELECT *
 FROM employees
 WHERE dept = 'Development')
SELECT *
FROM engineers E, support_cases SC
WHERE E.name = SC.assignee AND
      SC.created = '2016-09-30' AND
      E.location = 'Amsterdam'
```

```
SELECT *
FROM employees E, support_cases SC
WHERE E.dept = 'Development' AND
      E.name = SC.assignee AND
      SC.created = '2016-09-30' AND
      E.location = 'Amsterdam'
```

Fig. 1. Example of CTE merging

B. Condition pushdown

The condition pushdown is used when merging is not possible, for example when CTE has GROUP BY. Conditions in the WHERE clause of a query that depend only on the columns of the CTE are pushed into the query defining this CTE. In the general case conditions can be pushed only in the HAVING clause of the CTE, but on some conditions it makes sense to push them into the WHERE clause. As a result, a temporary table is made smaller.

Besides CTEs this optimization works for derived tables and non-mergeable views.

On the Fig. 2 the example of how this technique works is shown. The upper listing shows the initial query and the low shows how optimizer will transform it.

```
WITH sales_per_year AS (
 SELECT year(order.date) AS years,
        sum(order.amount) AS sales
 FROM order
 GROUP BY year)
SELECT *
FROM sales_per_year
WHERE year IN ('2015','2016')
```

```
WITH sales_per_year AS (
 SELECT year(order.date) AS years,
        sum(order.amount) AS sales
 FROM order
 WHERE year IN ('2015','2016')
 GROUP BY year)
SELECT *
FROM sales_per_year
```

Fig. 2. Example of condition pushdown

C. CTE reuse

The main idea of this method is to fill the CTE once and then use it multiple times. It works with condition pushdown. Yet if different conditions are to be pushed for different CTE instances than the disjunction of these conditions has to be pushed into CTE.

D. Comparison of optimizations in MariaDB, PostgreSQL, MS SQL Server, MySQL 8.0.0-labs

MariaDB as MS SQL Server supports merging and condition pushdown. PostgreSQL supports reuse only. MySQL 8.0.0-labs supports both merging and reuse and it works in such way: it tries merging otherwise makes reuse.

TABLE I. EXISTENCE OF OPTIMIZATION TECHNIQUES IN DIFFERENT RDBMSS

DBMS	Optimization technique exists		
	CTE merge	*Condition pushdown*	*CTE reuse*
MariaDB 10.2	yes	yes	no
MS SQL Server	yes	yes	no
PostgreSQL	no	no	yes
MySQL 8.0.0-Labs	yes	no	yes

V. THE RESULTS OF EXPERIMENTS ON MARIADB AND POSTGRESQL

Some tests have been conducted on the computer with processor Intel(R) Core(TM) i7-4710HQ CPU, 2.50GHz, 8 GB RAM on Opensuse 13.2 operating system. We tested PostgreSQL 9.3 and MariaDB 10.2 database systems, and gave the same amount of 16 Mb memory for temporary tables in both systems. It was relevant to use PostgreSQL 9.3 because CTEs implementation didn't change in further versions.

The experiments were made in a database containing the information about domestic flights in the USA during 2008. Database schema consists of the following relations:

- tab_2008(month, dayofmonth, dep_time, arrtime, flightnum, airtime, origin, dest, dist);
- airports(names);

We wanted to find multi-destination itineraries. So, we decided to find the shortest way between the airports of interest by plane. The table *airports* shows which airports should be visited. None of the airports can be visited twice. Besides, the plane should leave for the next destination a day or more after the previous plane.

The following query *Q1* for MariaDB is shown on Fig. 3. The script for PostgreSQL has a difference in functions *cast(origin as char(32))* and *locate(tab_2008.dest, s_planes.path)*. The analogue of *locate* function in PostgreSQL is *position* function and its return type is *text*, that's why the result of *cast* function in PostgreSQL in this query will be not *char(32)*, but *text*.

This query starts from 'IAD' airport in *seed_part* and looks through the table *tab_2008* to find flights with 'IAD' as origin

and one of the airports from table *airports* as destination. As the needed destination is found it is checked if it has already been visited on the route to prevent repeats. From the received data we take only those paths that involve all airports from table *airports* and have the smallest overall distance.

```
WITH RECURSIVE s_planes (path, dest, dayofmonth,
dist, it) AS (
 SELECT cast(origin as char(30)), origin,
        dayofmonth, 0, 1
 FROM tab_2008
 WHERE dayofmonth = 3 AND origin = 'IAD' AND
       flightnum = 3231
 UNION
 SELECT
  concat(s_planes.path,',',tab_2008.dest),
  tab_2008.dest, tab_2008.dayofmonth,
  s_planes.dist+tab_2008.dist, it+1
 FROM tab_2008, airports, s_planes
 WHERE
  tab_2008.origin = s_planes.dest AND
  locate(tab_2008.dest, s_planes.path)=0 AND
  tab_2008.dest = airports.name AND
  tab_2008.dayofmonth > s_planes.dayofmonth)
SELECT *
FROM s_planes
WHERE it = 8 AND
      dist = (SELECT min(dist)
              FROM s_planes
              WHERE it = 8);
```

Fig. 3. Query Q1 for MariaDB

TABLE II. THE RESULTS OF THE QUERY Q1 (OVERALL RESULT)

DBMS	Records count		
	587130	*1134055*	*6858079*
MariaDB	16.72 sec	31.97 sec	3 min 9 sec
PostgreSQL	60.29 sec	1 min 91 sec	11 min 50 sec

TABLE III. THE RESULTS OF THE EXPERIMENTS DURING AIRPORTS COUNT MINIMIZATION (OVERALL RESULT)

DBMS	Airports count				
	4	*5*	*6*	*7*	*8*
MariaDB	2.28 sec	3.49 sec	5.93 sec	14.45 sec	31.97 sec
PostgreSQL	1.13 sec	2.97 sec	6.74 sec	38.66 sec	1 min 91 sec

We've made a query *Q1* on table *tab_2008* consists of different number of records: 587130, 1134055 and 6858079. The results of the experiments are shown in TABLE II.

What can be seen from the table is that the results of the tests in MariaDB are more than three times better than in PostgreSQL.

Also query *Q1* was made on the same table with 587130 records but with the index on *origin* column. The results improved only for an overall value of 1 second in both database systems, so it was decided to continue experiments on the tables without indexes.

Although, we decided to make some other experiments and minimize the number of searched airports. So, fewer steps of recursion were made. We made these experiments only on the table with 1134055 records. The results are shown in TABLE III.

When the number of searched airports is 8, MariaDB has much better results than PostgreSQL. But during the minimization of the number of the airports PostgreSQL results become closer and closer to MariaDB results. When the number of the airports is fewer than 5 PostgreSQL results become better than MariaDB ones and this trend continues.

We compared query execution plans in both database systems and found that they were absolutely the same. However, operation of HASH JOIN made in a recursive part of the query requires 3-4 times longer period in PostgreSQL than in MariaDB on the big number of recursive iterations. We don't know the reason of this yet, and in our further researches we will focus on this theme.

VI. CONCLUSION

In this paper we presented a number of techniques for execution of CTEs and provided an implementation of these techniques for MariaDB. We described in details recursive CTE computation and mutual recursive CTE computation. Also we discussed in which cases non-linear recursion can be used. We compared existence of optimization techniques for non-recursive CTE in different databases.

We also performed some experiments using flights table on PostgreSQL and MariaDB. They showed that PostgreSQL has better results only when few steps of recursion are needed. For the long recursive process on a huge amount of data MariaDB is a better choice.

The authors want to express gratitude to MariaDB developers Igor Babaev and Sergey Petrunya for their participation in the work on MariaDB CTE implementation and their help in writing this article.

REFERENCES

[1] Code of the implementation of non-recursive CTEs for MariaDB, URL: https://github.com/MariaDB/server/pull/134

[2] Code of the implementation of recursive CTEs for MariaDB, URL: https://github.com/MariaDB/server/pull/182

[3] SQL/Foundation ISO/IEC 9075-2:1999

[4] SQL/Foundation ISO/IEC 9075-2:2008

[5] P. Przymus, A. Boniewicz, M. Burza´nska, and K. Stencel. Recursive query facilities in relational databases: a survey. In DTA and BSBT, pages 89–99. Springer, 2010

[6] Oracle Database Online Documentation, 10g Release 2 (10.2)

[7] D. Stuparu and M. Petrescu. Common Table Expression: Different Database Systems Approach. Journal of Communication and Computer, 6(3):9–15, 2009

[8] TPC Benchmark™DS (TPC-DS): The New Decision Support Benchmark Standard

[9] Boniewicz, A., Stencel, K., Wiśniewski, P.: Unrolling SQL:1999 recursive queries. In Kim, T.h., Ma, J., Fang, W.c., Zhang, Y., Cuzzocrea, A., eds.: Computer Applications for Database, Education, and Ubiquitous Computing. Volume 352 of Communications in Computer and Information Science. Springer Berlin Heidelberg (2012) 345–354

[10] Szumowska, A., Burzańska, M., Wiśniewski, P., Stencel, K.: Efficient Implementation of Recursive Queries in Major Object Relational Mapping Systems. In FGIT 2011 78-89

[11] Burzanska, M., Stencel, K., Suchomska, P., Szumowska, A., Wisniewski, P.: Recursive queries using object relational mapping. In Kim, T.H., Lee, Y.H., Kang, B.H., Slezak, D., eds.: FGIT. Volume 6485 of Lecture Notes in Computer Science., Springer (2010) 42–50

[12] Wiśniewski, P., Szumowska, A., Burzańska, M., Boniewicz, A.: Hibernate the recursive queries - defining the recursive queries using Hibernate ORM. In Eder, J., Bielikov´a, M., Tjoa, A.M., eds.: ADBIS(2). Volume 789 of CEUR Workshop Proceedings., CEUR-WS.org (2011) 190–199

[13] Ghazal, A., Crolotte, A., Seid, D.Y.: Recursive sql query optimization with k-iteration lookahead. In Bressan, S., K¨ung, J., Wagner, R., eds.: DEXA. Volume 4080 of Lecture Notes in Computer Science., Springer (2006) 348–357

[14] Ordonez, C.: Optimization of linear recursive queries in sql. IEEE Trans. Knowl. Data Eng. 22 (2010) 264–277

[15] Burzanska, M., Stencel, K., Wisniewski, P.: Pushing predicates into recursive sql common table expressions. In Grundspenkis, J., Morzy, T., Vossen, G., eds.: ADBIS. Volume 5739 of Lecture Notes in Computer Science., Springer (2009) 194–205

Разработка аспектно-ориентированного расширения для языка Kotlin

Борис Скрипаль
Санкт-Петербургский политехнический
университет Петра Великого
Email: skripal@kspt.icc.spbstu.ru

Владимир Ицыксон
Санкт-Петербургский политехнический
университет Петра Великого
Email: vlad@icc.spbstu.ru

***Аннотация*—В статье описывается разработка аспектно-ориентированного расширения для языка Kotlin. Kotlin — молодой мультипарадигменный язык программирования, быстро набирающий популярность. Однако для полноценной конкуренции с существующими языками необходима реализация многих возможностей, уже существующих для традиционных языков. Одна из таких возможностей — аспектно-ориентированное программирование. В статье на основе анализа существующих аспектно-ориентированных расширений для различных объектно-ориентированных языков разрабатывается программный прототип, позволяющий использовать аспектно-ориентированный подход при написании программ на языке Kotlin. Для этого проектируется язык описания аспектов, реализуются методы внедрения сквозной функциональности. Созданный прототип протестирован на серии примеров. Тестирование показало корректность предложенного подхода и работоспособность прототипа.**

I. Введение

Аспектно-ориентированный подход (АОП) был предложен как решение проблемы описания сквозной функциональности в объектно-ориентированных программах. Впервые подход был представлен в 1997 году Грегором Кичалесом в работе «Aspect-oriented programming» [1]. В предложенном подходе сквозная функциональность описывается отдельно от объектно-ориентированного кода программы и внедряется на этапе компиляции. Такое разделение позволяет не только компактно описывать сквозную функциональность, но и делать её внедрение прозрачным для программиста.

Парадигма АОП предложила элегантное решение для ряда задач, как, например, протоколирование и трассировка программ, работа с транзакциями, обработка ошибок, а также некоторых других. АОП стал интенсивно развиваться и, в настоящий момент, аспектно-ориентированные расширения созданы для большинства объектно-ориентированных языков программирования, как, например, C++ [2], C# [3], Java [4], [5], Python [6] и т.п.

В данной статье мы представляем аспектно-ориентированное расширение для нового языка программирования Kotlin. Язык Kotlin — молодой мультипарадигменный язык программирования, разрабатываемый компанией JetBrains. Основная цель языка Kotlin — быть компактной, выразительной и надежной заменой языка Java. При этом язык обеспечивает полную совместимость с программами на языке Java.

Наличие существенного числа новых конструкций не позволяет использовать уже готовые АОП-расширения для JVM-совместимых языков, поэтому аспектно-ориентированное расширение для языка Kotlin необходимо разрабатывать, специально ориентируясь на особенности языка Kotlin.

Оставшаяся часть статьи организована следующим образом. В втором разделе приведено общее описание аспектно-ориентированной парадигмы, в третьем разделе представлены актуальные реализации аспектно-ориентированных расширений. Четвертый раздел посвящен описанию разрабатываемого расширения для языка Kotlin. В пятом разделе проводится тестирование расширения на реальных проектах. В заключении приводятся направления дальнейшего развития.

II. Аспектно-ориентированный подход

При использовании АОП, сквозная функциональность инкапсулируется в отдельные сущности, называемые *аспектами* (aspects). Каждый аспект состоит из *советов* (advices) и *срезов* (pointcuts). Совет — сущность, содержащая в себе непосредственно сквозную функциональность (код совета), а также способ её внедрения в программный код. Срез — описание множества *точек внедрения* советов. Типовыми способами внедрения сквозной функциональности являются:

- вставка совета до точки внедрения;
- вставка совета после точки внедрения;
- вставка совета после возникновения исключительной ситуации;
- вставка совета вместо точки внедрения.

Вставка кода совета может производиться статически и динамически. При статическом способе внедрения советов, сквозная функциональность внедряется или на этапе сборки (модификация исходных кодов программы и модификация промежуточного представления в процессе компиляции), или же, сразу после сборки, например, путем модификации байт-кода (для программ, работающих поверх JVM). Основным преимуществом данного подхода является отсутствие затрат на внедрение советов во время выполнения программы [7], однако, при изменении одного совета, будет необходима повторная сборка всего проекта. В связи с этим, статический способ внедрения используется как правило в

тех случаях, когда сквозная функциональность необходима для корректной работы программы.

При динамическом внедрении сквозной функциональности код советов применяется непосредственно в процессе выполнения программы. Одним из способов динамического внедрения советов является создание прокси-объектов [8], которые перехватывают исполнение программы, позволяя выполнять код совета до, после или вместо точки внедрения. Такой способ внедрения аспектов не требует повторной сборки всего проекта после изменения аспектов, однако имеет меньшую производительность.

III. Обзор существующих аспектно-ориентированных расширений

Перед разработкой АОП-расширения для языка Kotlin рассмотрим существующие на рынке АОП-решения и их основные особенности.

A. AspectJ

AspectJ — первое аспектно-ориентированное расширение для языка Java, оно было представлено в 2003 году [9] и развивается до сих пор[1]. AspectJ расширяет грамматику языка Java, предоставляя дополнительные языковые конструкции для описания аспектов. В отличие от классов описание аспекта в AspectJ начинается с ключевого слова *aspect*. Сам аспект имеет уникальное имя и состоит из тела, которое содержит описание срезов и советов, а также может использовать переменные и методы языка Java. Описания срезов могут задаваться как отдельно от совета (в таком случае им необходимо задавать идентификатор, уникальный в рамках аспекта), так и, непосредственно, при описании совета. Способ внедрения кода совета относительно точки внедрения задается при описании совета с помощью ряда ключевых слов: *before* (после точки внедрения), *after* (перед точкой внедрения) и т.д. Также в AspectJ существует возможность описания аспектов при помощи специальных аннотаций над классами языка Java. При таком подходе в качестве аспекта выступает класс, снабженный аннотацией *@Aspect*, а срезами и советами выступают методы данного класса, имеющие аннотации *@Pointcut* или же специальными аннотациями советов (например, *@Before*) соответственно. Описание срезов также задается внутри аннотаций.

AspectJ поддерживает статический и динамический способы внедрения аспектов [4]. Статическое внедрение аспектов может производиться, как на уровне исходных кодов, так и сразу после компиляции программы (внедрение кода советов в байт-код скомпилированной программы). Динамическое внедрение аспектов производится непосредственно перед тем, как JVM загружает файл класса, что позволяет избежать временных затрат на внедрение аспектов во время выполнения программы.

[1]На момент написания статьи последняя доступная версия 1.8.10 датирована 12 декабря 2016 г.

B. SpringAOP

Другим популярным АОП-расширением для языка Java является SpringAOP — расширение, входящее в состав фреймворка «Spring Framework». Первая версия данного расширения была представлена в 2005 году, последняя (5.0.0.M5) — в феврале 2017 года. Для описания аспектов в SpringAOP используются специальные аннотации, размечающие классы и их методы, как аспекты, советы и срезы [5]. Также, как и в AspectJ, аспект может содержать не только описание срезов или советов, но и обычные в понимании языка Java переменные и методы.

SpringAOP поддерживает только динамическое внедрение аспектов в код. Основным способом применение аспектов в SpringAOP является связывание при помощи прокси-объектов.

Начиная с версии 5.0, Spring Framework добавил поддержку языка Kotlin [10], что позволяет учитывать execution функции, а также проверку на nullability при использовании SpringAOP.

C. AspectC++

Третьим рассматриваемым АОП-расширением является AspectC++ [2] — АОП-расширение для программ на языке C++. Первая реализация AspectC++ была представлена в 2001 году, последняя — в марте 2017 года. В качестве срезов, в AspectC++ могут выступать как статические сущности (классы, структуры, функции и т.д.), так и динамические сущности (вызов и выполнение функций, создание и удаление объектов) [11]. В данном расширении, аспект выступает как отдельная сущность, обозначающая ключевым словом *aspect* и содержащая в себе описание советов и именованных срезов. Основным способом внедрения аспектов, используемым в AspectC++ является статическое внедрение аспектов на уровне исходных кодов [12].

D. PostSharp

Еще одной реализацией АОП является PostSharp — АОП-фреймворк для программ, написанных на языке C#. PostSharp было представлен в 2004 году [3], как свободная библиотека для языка C#, однако в 2009 году данное расширение стало проприетарным. На момент написания статьи последняя стабильная версия расширения 4.3.29 была представлена в феврале 2017 года. Для описания аспектов в PostSharp существует ряд классов, обеспечивающих внедрения кода совета в программу. Каждый из таких классов определяет некоторый набор способов включения сквозной функциональности относительно точки внедрения [13]. Для реализации аспектов, необходимо переопределить соответствующие методы этого класса, а также прописать правила формирования среза внутри специальных аннотаций данного класса.

PostSharp поддерживает как статический, так и динамический способы внедрения аспектов. Для динамического применения аспектов могут использоваться как прокси-объекты, так и перехват момента загрузки файла в память,

после чего аспекты применяются непосредственно к бинарному файлу. Статическое внедрение аспектов может производиться как на уровне исходных кодов, так и во время компиляции. При применении аспектов во время компиляции используется MSIL — промежуточное представление, создаваемое в процессе компиляции программ для платформы .NET. MSIL является независимым от процессора набором инструкций, который впоследствии преобразуется в набор кодов для конкретного процессора. Такой подход позволяет удобно анализировать целевую программу, а также избавляет от необходимости поддерживать корректность исходных кодов программы после внедрения аспектов, что необходимо делать при модификации на уровне исходных кодов.

По результатам анализа ряда популярных АОП-расширений можно сделать вывод о том, что не смотря на схожую функциональность, расширения сильно отличаются как способами описания аспектов, так и способами их внедрения. Данные различия обусловлены не только особенностями языка программирования, для которого предназначено расширение, но и стремлением сделать синтаксис описания аспектов как можно более удобным и понятным для разработчика.

IV. Разработка аспектно-ориентированного расширения для языка Kotlin

При реализации аспектно-ориентированного расширения необходимо сформировать синтаксис аспектов и определить механизмы внедрения аспектов.

A. Разработка синтаксиса описания аспектов

Описание аспектов является расширением языка программирования и должно позволять в удобной и понятной форме задавать все сущности АОП: аспекты, советы, срезы и т.п. В этой работе мы приняли решение не разрабатывать специализированный синтаксис аспектов для языка Kotlin, а воспользоваться синтаксисом описания аспектов, используемый в AspectJ для языка Java, расширив его необходимыми для языка Kotlin конструкциями. Такой выбор был сделан по нескольким причинам:

- синтаксис описания аспектов, используемый в AspectJ, является очень удобным для программиста;
- грамматика AspectJ, описанная на языке ANTLR4, находится в свободном доступе и может быть использована для разработки языка аспектов для языка Kotlin;
- AspectJ является одним из самых популярных аспектно-ориентированных расширений для языка Java [14] и знаком многим разработчикам.

Учитывая, что в AspectJ описание аспектов может производится двумя способами: при помощи специальных аннотаций над классами и как отдельные сущности, было решено использовать второй способ. Такой выбор был сделан ввиду того, что такой способ описания аспектов, по мнению авторов, является менее избыточным, по сравнению с аннотациями, а также нагляднее отделяет объектно-ориентированную функциональность от сквозной.

Для адаптации AspectJ к языку Kotlin в синтаксис AspectJ были внесены следующие изменения:

- способ описания методов был изменен в соответствии с правилами языка Kotlin;
- стандартные типы языка Java были заменены на стандартные типы языка Kotlin;
- в соответствии с правилами языка Kotlin изменены модификаторы полей и методов;
- добавлена возможность задавать атрибуты аргументов методов;
- добавлена поддержка функций-расширений (extension functions) и подставляемых функций (inline functions).

Описание аспекта начинается с ключевого слова *aspect* и состоит из идентификатора аспекта и тела аспекта. Тело аспекта может содержать в себе описание срезов и советов. Описание среза начинается с ключевого слова *pointcut* и состоит из идентификатора среза и правила, в соответствии с которым производится формирования среза.

Правило формирования среза по сути является описанием множества точек внедрения, для задания которого используются логические операции и специальные ключевые слова, например, *execution* или *call*, указывающие на определенную группу мест в программном коде. Допускается описывать одни срезы с использованием других.

Для указания множества методов, входящих в срез, используются следующие механизмы:

- модификаторы видимости с логическими операторами (например, можно выделить все не public методы);
- класс, к которому принадлежит метод;
- имя метода или его часть;
- список аргументов метода с атрибутами;
- тип возвращаемого значения.

Описание совета состоит из задания способа применения совета (например, *before* или *after*), правила формирования среза и кода совета.

Пример описания аспекта приведен в листинге 1.

```
aspect A {
    pointcut fooPC(): execution(fun Foo.*())
    pointcut printPC(): call(public fun kotlin.io.
    pri*(!Any))

    after(): fooPC() && printPC() {
        println("Hello after!!")
    }

    before(): fooPC() && printPC() {
        println("Hello before!!")
    }
}
```

Листинг 1. Пример описания аспекта

Приведенный в листинге аспект содержит описание двух срезов: fooPC и printPC. Срез fooPC включает в себя все места вызовов функций и инициализации переменных внутри всех методов класса Foo. Срез printPC включает в себя все места вызовов публичных методов пакета kotlin.io, имя

которых начинается с pri, принимающих один аргумент, тип которого отличается от Any. Далее описываются два совета: первый вставляет код совета (вызов метода println) до всех точек внедрения, которые принадлежат одновременно к двум срезам fooPC и printPC, второй — после. После применения совета к программе во всех местах вызовов методов, удовлетворяющих следующим условиям:

- вызов происходит внутри методов, принадлежащих классу Foo;
- вызываемый метод имеет модификатор public и принадлежит к пакету kotlin.io;
- имя вызываемой метода начинается с pri;
- метод принимает единственный аргумент, тип которого отличается от «Any».

до вызова метода будет выведена на печать строка «Hello after!!», а после — «Hello before!!».

На момент написания статьи реализованы следующие возможности, существующие в AspectJ:

- Структуры, используемые при описании срезов:
  - *call* — вызов заданного метода;
  - *execution* — выполнение заданного метода;
  - *target* — вызов метода у экземпляра указанного класса;
- Способы внедрения советов:
  - *before* — вставка кода совета перед точкой внедрения;
  - *after* — вставка кода совета после точки внедрения;
  - *around* — вставка кода совета до и после точки внедрения.

B. Внедрение аспектов

Язык Kotlin имеет ряд оригинальных языковых конструкций (extension функции, специфичные лямбда-функции и т.п.), при этом основной целевой платформой компиляции для языка Kotlin является JVM. Как результат — все специальные конструкции Kotlin преобразуются в стандартный байт-код, который имеет одинаковую структуру и для Java-, и для Kotlin-программ. Из-за этого поиск некоторых структур языка Kotlin в байт-коде становится затруднительным и, как следствие, динамическое внедрение кода советов при загрузке файлов в JVM становится практически невозможным. По той же причине статическое внедрение советов в байт-код программы также является очень сложной задачей.

Таким образом, единственным разумным способом внедрения аспектов является внедрение аспектов в исходный код программы или в модель программы во время компиляции.

Разработчики языка Kotlin предусмотрели специальную структуру данных для работы с программным кодом — Program Structure Interface (PSI). В нашем проекте мы используем PSI для внедрения объектов на этапе компиляции проекта.

Внедрение аспектов происходит в несколько этапов, схематически изображенных на рисунке 1.

Первым этапом является построение PSI — промежуточного представления проекта, состоящего из набора виртуальных файлов, соответствующих исходными файлам, а также из прочей информации о проекте (конфигурация, путь до JDK и т.п.). Каждый из этих файлов содержит дерево разбора программного кода и информацию о файле. Каждый элемент дерева разбора содержит в себе текст соответствующего элемента, ссылки на потомков и различную сопровождающую информацию. Одним из таких полей является «userMap» типа *KeyFMap* — структура в, которую могут быть записаны различные пользовательские данные. На рисунке 1 схематически представлен процесс внедрения аспектов в программный код.

Вторым этапом является чтение файлов с описаниями аспектов и формирование модели аспектов, состоящих из срезов и советов. Каждый экземпляр совета и среза содержит:

- уникальный идентификатор, используемый для разметки PSI;
- дерево разбора логического выражения, используемое при анализе принадлежности точки срезу.

Также экземпляр совета содержит в себе код совета, приведенный к виду промежуточного представления для более удобной модификации PSI. Дерево разбора в качестве нетерминальных узлов содержит в себе логические операции «и», «или», «не». Терминальными же узлами выступают или сигнатуры, используемые для описания срезов, или же идентификаторы других срезов.

Формирование набора точек внедрения также происходит в несколько шагов. На первом шаге, каждый узел PSI проверяется на соответствие каждому из отдельно описанных внутри совета срезов. В случае, если узел подходит данному срезу, то в структуру userMap добавляется идентификатор этого среза. На втором шаге, после того, как дерево было размечено идентификаторами срезов, для каждого совета проверяется, можно ли его применить к данному узлу дерева. Если можно, то происходит внедрение кода совета в соответствующую позицию относительно точки внедрения.

При внедрении в PSI код советов оборачивается в лямбда функцию *run*, что позволяет разрешать сложные случаи, например, при последовательном вызове нескольких функций без создания большого числа дополнительных промежуточных переменных. Для наглядности, рассмотрим участок кода в листинге 2.

```
...
val res = A.foo().bar().baz()
...
```

Листинг 2. Пример целевой точки внедрения

При необходимости применить совет непосредственно после вызова функции *bar()* целевая функция оборачивается в метод *run*, значение, возвращаемое функцией присваивается некоторой промежуточной переменной, после чего вставляется код совета и затем из блока возвращается перемен-

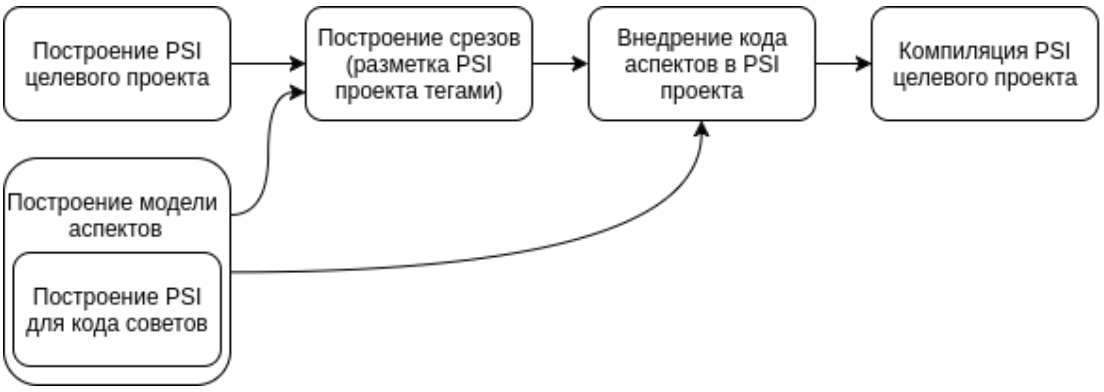

Рис. 1. Процесс внедрения аспектов в программный код при компиляции

ная, содержащая результат, возвращенный функцией *bar*, к которой был применен совет. Результат преобразования представлен в листинге 3.

```
...
val res = A.foo().run{
        val buf = bar()
        //advice code
        buf
    }.baz()
...
```

Листинг 3. Пример внедрения кода с использованием функции run

Данный подход решает проблему разнесения вложенного вызова методов, при котором пришлось бы заводить множество временных переменных, а также контролировать возвращаемые результаты методов. К недостаткам данного подхода можно отнести то, что при изменении аспекта, требуется полная перекомпиляция проекта. Однако, при таком подходе, нет необходимости производить дополнительные действия при запуске для корректной работы полученного jar-файла.

V. Тестирование разработанного прототипа

Для того чтобы убедиться в работоспособности изложенного выше подхода, был реализован прототип АОП-расширения для языка Kotlin. Проверка работоспособности разработанного подхода проводится с помощью тестирования. Сначала проверяется корректность реализации отдельных функций, затем проводится проверка приложений, к которым применили аспекты.

Проверка корректности реализации отдельных функций включает в себя:

- тестирование разбора логического выражения, описывающего срез;
- проверка правильности выделения точек внедрения;
- проверка внедрения кода советов в PSI целевой программы.

Из-за того, что сама по себе проверка PSI является сложной задачей, было решено проверить корректность работы на уровне функций следующим образом:

1) составить набор аспектов, которые будут применяться к целевой программе;
2) составить набор эталонных файлов, представляющих из себя исходные файлы целевой программы, к которым вручную применены описанные выше аспекты;
3) модифицировать PSI целевой программы, а именно произвести составление модели аспектов, разметку PSI и применить их к PSI проекта;
4) сформировать на основе модифицированных виртуальные файлов исходные файлы на языке Kotlin;
5) сравнить на уровне токенов полученные файлы с эталонными исходными файлами, в которых аспекты применены вручную.

После успешной проверки корректности работы на уровне функций, необходимо убедиться в том, что после применения аспектов программа остается корректной и работает согласно нашим ожиданиям. Это можно сделать, например, поместив в код совета вывод отладочных сообщений, и после выполнения программы необходимо сравнить результат работы программы с ожидаемым.

На начальных этапах работы, для тестирования были созданы простые примеры программ (несколько классов с 2-3 функциями в каждом из них, суммарный объем — несколько десятков строк кода). Впоследствии была проведена проверка на программах студентов, изучающих язык Kotlin (суммарный размер каждого проекта достигает несколько сотен строк кода). Время внедрения аспектов к программам размером в несколько сотен строк кода составило около 1-2 секунд.

В ходе тестирования ошибок обнаружено не было, что свидетельствует о работоспособности подхода и прототипа.

Рассмотрим пример протоколирования программы при помощи разработанного прототипа. В качестве целевой программы было выбрано приложение, вычисляющее значение арифметического выражения, заданного в файле. В качестве входных данных приложение принимает имя файла, в котором записано выражение, а также список значений параметра x, используемые при вычислении выражения. Результатом работы является ассоциативный массив в котором ключем является значение параметра, а значением — результат вы-

числения выражения при подстановке данного значения в качестве значения параметра.

Программа работает по следующему алгоритму: первым вызывается метод *pExpr*, принимающий имя файла в котором записано выражение и список значений параметра. Внутри метода *pExpr* в цикле вызывается метод *calc*, принимающий значение параметра и возвращающий вычисленное значение выражения. После чего значение параметра и результат вычисления записывается в ассоциативный массив, возвращаемый функцией *pExpr* в качестве результата. Для того, чтобы зафиксировать в журнале время начала и завершения работы метода *calc* воспользуемся аспектом, представленным в листинге 4.

```
aspect A {
  pointcut pExprPC() : execution(fun pExpr(String,
    List<Int>): Map<Int, Int> )
  pointcut calcPC() : call(fun Expression.calc(Int):
     Int)

  before(): pExprPC() && calcPC() {
    val l = java.util.logging.Logger.getLogger("ALog
    ")
    l.info("Start ${System.currentTimeMillis()}")
  }

  after(): pExprPC() && calcPC() {
    val l = java.util.logging.Logger.getLogger("ALog
    ")
    l.info("Finish ${System.currentTimeMillis()}")
  }
}
```

Листинг 4. Пример аспекта, используемого для логирования вызова метода calc

После применения аспекта, место вызова метода *calc*, представленное в листинге 5 будет преобразовано к виду, приведенному в листинге 6.

```
for (value in values) {
  result[value] = expr.calc(value)
}
```

Листинг 5. Место вызова метода calc до применения аспекта

```
for (value in values) {
  result[value] = expr.run{
    val ____a = run{
      val l = java.util.logging.Logger.getLogger("
    ALog")
      l.info("Start ${System.currentTimeMillis()}")
      calc(value)}
  val l = java.util.logging.Logger.getLogger("ALog")
  l.info("Finish ${System.currentTimeMillis()}")
  ____a}
}
```

Листинг 6. Место вызова метода calc после применения аспекта

Как видно из листинга 6, вызов метода был помещен внутрь лямбда функции *run* согласно ожиданиям. После запуска, в журнал выводятся сообщения о времени начала и окончания работы метода *calc*, что соответствует ожиданиям.

VI. Заключение

В ходе данной работы был разработан подход, позволяющий использовать аспектно-ориентированную парадигму при написании программ на языке Kotlin. На основе AspectJ было реализовано расширение, которое, используя статическое внедрение советов, модифицирует модель PSI. Работоспособность подхода была показана на ряде примеров.

Направления дальнейшей работы:

- расширение синтаксиса аспектов для поддержки всех возможностей языка Kotlin;
- оптимизация процедур внедрения аспектов;
- более глубокое тестирование разработанного программного обеспечения.

Список литературы

[1] G. Kiczales, J. Lamping, A. Mendhekar, C. Maeda, C. Lopes, J. Loingtier и J. Irwin, “Aspect-oriented programming,” *European Conference on Object-Oriented Programming (ECOOP)*, 1997.

[2] (2017). The home of aspectc++. англ., url: http://www.aspectc.org/Home.php.

[3] (2017). Postsharp documentation. англ., url: http://doc.postsharp.net/.

[4] (2017). Aspectj documentation. англ., url: http://www.eclipse.org/aspectj/docs.php.

[5] (2017). Aspect oriented programming with spring. англ., url: https://docs.spring.io/spring/docs/current/spring-framework-reference/html/aop.html.

[6] (2017). Aspect oriented programming — spring python. англ., url: http://docs.spring.io/spring-python/1.2.x/sphinx/html/aop.html.

[7] M. Forgac и J. Kollar, “Static and dynamic approaches to weaving,” *5th Slovakian-Hungarian Joint Symposium on Applied Machine Intelligence and Informatics*, 2007.

[8] W. Gilani и O. Spinczyk, “A family of aspect dynamic weavers,” *Proceedings of the 2003 Dynamic Aspect Workshop (DAW04 2003)*, 2003.

[9] G. Kiczales, E. Hilsdale, J. Hugunin, J. Kersten M. Palm и W. Griswold, “An overview of aspectj,” *European Conference on Object-Oriented Programming (ECOOP)*, 2001.

[10] (2017). Introducing kotlin support in spring framework 5.0. англ., url: https://kotlin.link/articles/Introducing-Kotlin-support-in-Spring-Framework-5-0.html.

[11] O. Spinczyk, “Documentation: Aspectc++ language reference,” *Pure-systems*, 2017.

[12] ——, “Aspectc++ — a language overview,” *Friedrich-Alexander University Erlangen-Nuremberg Computer Science 4*, 2005.

[13] (2017). Postsharp. aspects namespace. англ., url: http://doc.postsharp.net/n_postsharp_aspects.

[14] H. Kurdi, “Review on aspect oriented programming,” *International Journal of Advanced Computer Science and Applications*, 2013.

Aspect-oriented extension for the Kotlin programming language

Boris Skripal, Vladimir Itsykson

This paper present an aspect-oriented extension for the Kotlin programming language. Kotlin is a new multi-paradigm programming language that is rapidly gaining popularity. However, to be competitive with the existing programming languages it is necessary to implement many extensions already existing for the traditional programming languages. One of such extensions is an aspect-oriented approach. In the paper, after the analysis of the aspect-oriented extensions for the different object-oriented languages, we have developed a program prototype, which allows using the aspect-oriented approach with Kotlin programming language. We have developed a grammar for describing aspects and have created extension of Kotlin compiler for weaving cross-cutting functionality. The developed extension has been successfully verified on a test suite.

Automatic Creation of Stock Trading Rules on the Basis of Decision Trees

Dmitry Iskrich, Dmitry Grigoriev
St. Petersburg State University
St. Petersburg, Russia
st034749@student.spbu.ru, d.a.grigoriev@spbu.ru

***Abstract*—The main task of any trader is the selection of profitable strategies for trading a financial instrument. One of the simplest ways to represent trading rules is binary decision trees based on the comparison of current values of technical indicators with some absolute values. This approach simplifies the creation of trading systems but their validity is limited to short-term intervals of trade. This paper represents an approach for generating more universal trade rules in the form of binary decision trees based on the comparison of the current values of technical indicators with relative levels. The ranges of these levels are recalculated on a daily basis which allows the trading rule to remain relevant within a long term. The proposed system analyzes the historical price data for a long period and using the genetic algorithm causes trading strategies optimized relatively to the Sharpe ratio.**

I. Introduction

A strategy of a trader concluded in the formalization of rules for opening and closing a position. Quite often these rules are based on methods of the technical analysis and processing of such data as the current price of the instrument, the volume of trading, the maximum / minimum price for a certain period, the price at the time of opening / closing of the trading session. Based on the series of these data, so-called technical indicators are generated. In accordance with postulates of the technical analysis technical indicators allow to predict the probable trend direction in a given security. Thus, the combination of the indicator and its value can serve as a basis for the trading strategy [1], [2], [3].

The discovery of profitable patterns is a non-trivial task. Typically a trader manually programmes his trading system and then optimizes its key parameters by exhaustive search or by heuristic algorithms [4]. In this paper we describe a method for automatic building the rules of a trading system without human intervention. Obviously, its use will allow us to discover a greater number of effective patterns of price behavior.

One of the simplest representations of trade strategy is binary decision trees [5], [6], [7]. In this case nodes contain various assumptions about the state of a technical indicator or another characteristic of the current market situation, and its truth or falsity is determined by both branches. Methods, which use this data structure have a wide area of application [8], [9].

In our work, a similar approach is used, which was first presented in [10]. Its specificity is the use of the genetic algorithm for the generation and selection of optimal decision trees. This algorithm relies on the daily historical data of indicators and generates the most "trained" trading decision tree. Unlike in [7] and [11], the values of technical indicators in the tree are compared not with the real value, but with one of the levels relative to the dynamics in the past. This increases the time of the rule's relevance, as the ranges of levels are recalculated on a daily basis and thus they reflect the latest state of the market.

It should be noted that similar systems use BUY/SELL signals allowing opening short positions and working with them symmetrically to the long ones. In practice, however, there are limitations: the adequate estimation of the effectiveness of short positions is difficult because of the necessity to pay for the borrowed securities; the period of their retention should be as short as possible; there are temporary or permanent prohibitions on their opening.

There are systems based on the markup of historical data with instructions to take a trade action. This stage is necessary for training some classifiers the essence of which is to find the relationship between a particular class and the values of technical indicators. However, at the moment there is no optimal way of such a markup.

So, for example, in paper [8] for the arrangement of BUY, SELL and HOLD signals for a certain day the trading indicators of the next two days were used. After that using algorithm ID3 with these data a decision tree was created. At the same time, according to allegations [12] for the determination of the trade action it may be insufficient to have several subsequent values. The general trend of price changes plays an important role, and its duration time is not always obvious. In system [13] the arrangement of BUY and SELL signals is shown on the basis of a fall or rise in prices for 2 years by 10 or 15 percent respectively. Data with the assigned marks are used by algorithm C4.5 to create the optimal decision tree. The same algorithm is used in paper [14], however, the markup of the data for training is based on the real actions of the most successful trader on a specific day.

Thus, the abundance of all possible variants of the markups of historical data per trading actions led to the decision to use the genetic algorithm as a classifier. This approach requires only the existence of the objective function from the data for the decision tree, thus eliminating the need to subjectively mark out historical data.

The contribution of this paper is as follows:

1) In trade rules only long positions are allowed.

2) A new optimization algorithm for the decision tree is presented which is obtained after the application of crossover and mutation operators.

3) The principle of calculating the levels of indicators is described.

A more detailed description of the tree structure is given in Section II. Section III contains a description of the genetic algorithm including the initialization process and the classical evolutionary steps. Principles of testing the best systems and the analysis of their effectiveness are presented in Section IV.

II. Trading decision tree

A decision tree is a connected acyclic graph with the leaves represented by decisions, in our case these can be signals for opening a long position (LONG) and exit from it (CLOSE LONG). The process of determining decisions starts from the root, then, depending on the node value, the transition into either the right or the left node takes place. As a result, the leaf at the end is taken, i.e. an order to buy a stock and open a long position (LONG) or sell out the purchased volume (CLOSE LONG). The system starts with the CLOSE LONG state allowing the first purchase with the LONG signal. The position size is determined by the maximum possible number of shares that can be purchased at the current balance. Thus, all the received income is reinvested.

A tree node is a predicate defining a transition to one of the two children. In every non-terminal node, the value of the technical indicator is compared to a certain level to see whether they match.

A. Calculation of the indicator level

Most often indicators have real values, but their current state can be associated with a certain level, this approach proved itself in [15]. In other words, at a given time, a technical indicator can have one value from the enumeration: {*Very Low, Low, Medium, High, Very High*}.

To calculate this value, the system uses n previous values of the indicator. In this system, n equals to the duration of the training period. After that the calculation of the ranges for each level takes place:

1) The maximum (Max) and minimum (Min) values of the indicator for the previous n days are computed.
2) On the basis of these indicators, an interval [Min, Max] is generated and divided into 5 equal segments.
3) The current level is determined by the segment into which the real value of the indicator falls.

If the value exceeds Max, the indicator level is assigned $VeryHigh$. Similarly, if the value is less than Min the indicator level will be $VeryLow$.

It should also be noted that the current level of indicator is determined after closing the trading session and it is valid for the whole next trading day.

In the simplest form, the decision tree is shown in Fig. 1.

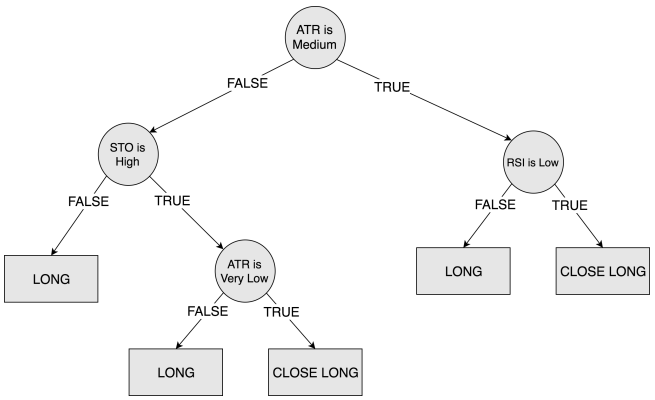

Figure 1. Example of an trading decision tree

III. Genetic algorithm

To identify the best tree the genetic algorithm is used. All steps of the classic evolution are involved.

A. Initialization

The initial population is initialized with a preset number of random generated trees, with each node created by selecting a random indicator and associating it with a random level. The height of the tree is determined by the number of indicators available to the system (in our case 3). After this height has been reached, the generation of non-terminal nodes ceases and random LONG/CLOSE LONG leaves are created.

B. Selection

For the further tree modification with the genetic operators the individuals that fit most are selected and thus the next generation is formed. The selection is done by the roulette wheel method. For each tree the probability of being chosen is:

$$p_i = \frac{f_i}{\sum_{j=1}^{n} f_j} \tag{1}$$

where f_i is the fitness function of tree T_i

C. Fitness function

For the evaluation of the tree the Sharpe ratio (Sharpe Ratio), an indicator of the effectiveness of the strategy, introduced by W.F. Sharpe [16] was chosen. It determines the relationship between the profit and the possible risk. It is calculated as:

$$f = \frac{r_p - R_f}{\delta(x)}, \tag{2}$$

where r_p is the average income, R_f is the income with a zero risk, $\delta(x)$ is the standard deviation of income. As a risk-free rate, the value of 2.5% was taken as an analogue of the 10-year US Treasuries.

D. Genetic operators

Genetic operators are constructed by analogy with paper [11] and include crossover and mutation operators.

Due to these two operations, the principles of inheritance and self-adaptivity can be achieved. Taking into account the specifics of the tree-shape data structure, the basis for the modification consists of manipulations with the terminal and non-terminal nodes.

In the process of crossover, two individuals (in the form of decision trees) are selected for crossing. The probability of selection of each one is consistent with the roulette method and it is computed using formula (1). After that copies of the both trees exchange their randomly selected subtrees and get included into the population (together with the original trees). An example of the operator's action is shown in Fig. 2.

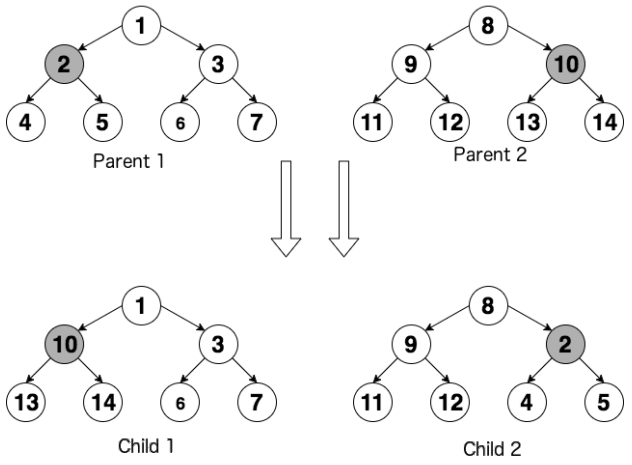

Figure 2. Crossover Operator

The mutation operator is a unary operator, whose purpose is to prevent from premature convergence of the generation to each other. It is based on the same principle as the crossover - two nodes are randomly selected and swapped in the tree.

After these operators are applied, an important step is needed to remove inconsistent nodes. In Fig. 3 a typical case of this kind of errors is presented. As it can be seen, when visiting node *IND1 is LOW* it is already known that this predicate is false, since on the way from it to the root there is node *IND1 is HIGH*. The process of removing excess nodes goes by raising their respective child. In this example, if *IND1 is HIGH* is true, there will be a transition to *IND3 is MEDIUM* (i.e. to the subtree, to which would be a transition in case of falsity of node *IND1 is LOW*).

After creating a new generation, an excess node removal algorithm (Algorithm 1) is run for each tree that had been modified by crossover or mutation. As a result, a tree either retains its structure, or, if an unnecessary predicate is found in a node, the nodes are replaced according to the algorithm suggested below. The output is an optimized tree ($optimizedTree$) that works exactly the same way as the original one.

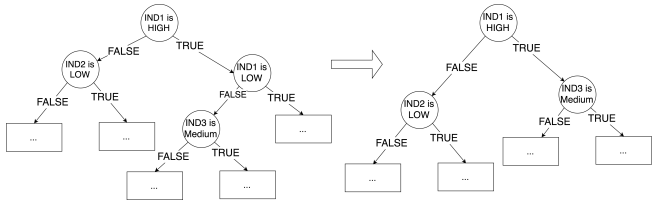

Figure 3. Optimization of tree

As the input the algorithm accepts two nodes which are swapped. In this case the nodes can belong to both different trees and to one tree. So, it can be used both in case of a crossover and a mutation. In this algorithm $replacedNode1$, $replacedNode2$ are the nodes that are swapped around; $RightTree$ is the subtree node in the case of truth of the predicate, $LeftTree$ is the subtree in case of falsity; $Name$ - the name of the indicator in the predicate, $Value$ is its amount; *parent* is the parent of the node.

The estimation of computational complexity is $\mathcal{O}(n \log n)$, where n is the number of nodes in the tree. This complexity allows, in principle, to use a large number of indicators to build a trading system.

E. General algorithm

The general algorithm consists of the steps of the classical genetic algorithm. Iteratively (in 25 steps) new generations (200 individuals) are created, after which their fitness function is calculated and the selection is carried out by means of the roulette method. To a certain number of winners the operators of crossover (35%) and mutation (5%) are applied. The trees that underwent these permutations are optimized by Algorithm 1. The result will be the best tree (by the fitness function value) in the latest generation.

All parameters are chosen empirically to ensure a balance of quality and running time. The algorithm scheme, its parameters and time characteristics are described more in detail in our work [10]. Sometimes sustainability of the result can be required from the genetic algorithm. That is, subsequent start-ups of the system with identical parameters must bring the identical decision tree. In case of three technical indicators, this will require increasing the number of individuals to 800 and for generations to 50.

```
Input  : replacedNode1, replacedNode2
Output: optimizedTree
// Traversal of the children of the changed nodes
TraversalOptimize(replacedNode1);
TraversalOptimize(replacedNode2);
Procedure TraversalOptimize(tree)
    if tree.Left ≠ Null then
        Traversal(tree.Left);
    end
    if tree.Right ≠ Null then
        Traversal(tree.Right);
    end
    FixRepetitions (tree);
/* /* Procedure for checking and removing excess
   nodes                                          */
Procedure FixRepetitions (node)
    /* Remember the true and false branch of the
       verified node                              */
    trueWay ⟵ node.RightTree;
    falseWay ⟵ node.LeftTree;
    currentNode ⟵ node.parent;
    /* Going up to the root and checking
       currentNode with each node on the path     */
    while currentNode ≠ Null do
        if currentNode.Name = node.Name then
            if currentNode.Value = node.Value then
                // An identical predicate is found
                if node in currentNode.LeftTree
                 then
                    /* If the node is in the left
                       descendants subtree of the
                       current node, replace it with the
                       false branch.                */
                    node ⟵ falseWay;
                else
                    /* Otherwise, for the true one  */
                    node ⟵ trueWay
                end
            else
                /* If only the names of the indicators
                   coincide, it is sufficient to check
                   whether the node is in the right
                   descendants subtree of the current
                   one. In this case, the node is
                   obviously false, since the values
                   differ.                          */
                if node in currentNode.RightTree
                 then
                    node ⟵ falseWay;
                end
            end
        end
        /* Going one level up                       */
        currentNode ⟵ currentNode.parent
    end
```

Algorithm 1: Removing excess node

IV. Testing

A. Metrics

In addition to the objective function – the Sharpe ratio, for the analysis of the results, additional metrics were used.

1) Return is a profit rate in percent and it is calculated by trade simulations using historical data. The higher it is, the better.
$$Return = (\frac{FinalBalance}{StartBalance} - 1) * 100, \qquad (3)$$
where $FinalBalance$ is the amount of money after the simulation and $StartBalance$ is the amount of money at the beginning.
2) Max drawdown is the maximum reduction of the account balance value from its local high in percent. It allows to estimate the maximum loss of capital when using this strategy. The lower it is, the better.
3) Trades count is the number of open positions for the period under review. The index is proportional to the possible value of the commission costs.
4) Buy and Hold represents the stock return, which is provided by the market. The essence of this strategy is in the purchase of the instrument at the very beginning and its retention until the end of the given trading period. For a comparison, the profit indices and the Sharpe ratio (B&H Returns, B&H Sharpe) were taken.

B. Indicators

Below there are indicators [1], on which the system builds up its trade recommendations:

1) Relative strength index (RSI), this oscillator compares the number of recent asset price rises with the number of its falls.
2) Stochastic oscillator (STO), shows the position of the current price relative to the price range for a certain period.
3) Average True Range (ATR) is an indicator of the current market volatility.

A greater number of indicators increases the sensitivity of the trading system, which increases the number of transactions.

C. Results of testing

As a testing object, five large IT companies were selected: Citrix Systems (CTXS), Electronic Arts (EA), eBay Inc (EBAY), Intel (INTC), Oracle (ORCL). Consistently for each year a selection of the best trading decision tree and its verification in the next one is carried out. One year has 251 trading signals, each of which corresponds to a certain stock market day.

It should be noted that the time characteristics of our algorithm allow to apply it to the analysis of 5-minute timeframes and higher. However, for liquid shares the greater the timeframe is, the lower the influence of the "noise" component of the market is. This component is

Table I
CTXS results

Year	Sharpe	B&H Sharpe	Return	B&H returns	Max drawdown	Trades Count
2007 Train	2.73	1.26	59.64%	36.16%	6.92%	42
2008 Test	-0.82	-0.73	-20.23%	-35.72%	36.66%	28
2008 Train	1.60	-0.73	26.88%	-35.72%	6.70%	26
2009 Test	-0.05	1.52	1.90%	67.20%	6.57%	16
2009 Train	3.42	1.52	102.75%	67.20%	7.96%	40
2010 Test	1.64	1.34	19.64%	54.37%	4.08%	18
2010 Train	1.82	1.34	64.59%	54.37%	10.23%	15
2011 Test	0.29	-0.07	6.66%	-9.04%	28.50%	14
2011 Train	1.53	-0.07	40.58%	-9.04%	8.58%	14
2012 Test	0.65	0.24	11.06%	5.26%	9.25%	21
2012 Train	1.57	0.24	12.60%	5.26%	2.76%	15
2013 Test	-0.78	-0.32	-3.27%	-8.77%	9.68%	18
2013 Train	2.67	-0.32	37.12%	-8.77%	3.56%	30
2014 Test	1.85	0.16	21.80%	3.66%	2.99%	34
2014 Train	3.84	0.16	46.11%	3.66%	1.51%	37
2015 Test	3.00	0.69	38.48%	19.96%	4.37%	34
2015 Train	1.69	0.69	18.37%	19.96%	1.82%	18
2016 Test	1.16	0.73	10.98%	18.66%	3.88%	35

Table II
EA results

Year	Sharpe	B&H Sharpe	Return	B&H returns	Max drawdown	Trades Count
2007 Train	2.74	0.45	54.25%	11.56%	9.78%	70
2008 Test	-1.30	-1.94	-33.88%	-65.56%	44.81%	38
2008 Train	3.32	-1.94	132.34%	-65.56%	7.19%	54
2009 Test	1.87	0.19	28.07%	1.50%	4.35%	30
2009 Train	2.72	0.19	71.93%	1.50%	8.78%	29
2010 Test	0.54	-0.25	10.76%	-8.73%	9.25%	22
2010 Train	2.78	-0.25	43.72%	-8.73%	4.86%	52
2011 Test	-0.05	0.67	0.11%	23.08%	13.33%	34
2011 Train	1.32	0.67	46.63%	23.08%	20.28%	14
2012 Test	-0.58	-0.95	-16.38%	-27.89%	33.46%	12
2012 Train	1.46	-0.95	24.23%	-27.89%	7.94%	39
2013 Test	1.38	1.30	24.36%	53.26%	4.34%	44
2013 Train	3.62	1.30	58.90%	53.26%	5.01%	57
2014 Test	2.53	2.22	26.16%	97.53%	3.33%	46
2014 Train	5.10	2.22	112.49%	97.53%	4.00%	36
2015 Test	1.46	1.33	32.30%	44.03%	11.97%	41
2015 Train	4.59	1.33	95.09%	44.03%	4.00%	74
2016 Test	0.44	0.71	8.71%	19.65%	12.76%	52

caused by unpredictable actions of large players, as well as by a random combination of actions of many trading robots and traders. Vulnerability of the trading system to such a random component leads to premature signals. Therefore, similarly to works [7] and [11], testing is carried out on daily historical data.

The results are shown in Tables I - V. The lines with the better system performance than the B&H strategy are highlighted in grey.

V. Future work

Obviously, the efficiency of the constructed trading strategies substantially depends on the set of technical indicators and their parameters. In this regard, it is necessary to investigate the applicability of other technical indicators and to automate the selection of optimal parameters of the indicators. Also in the future it is planned to expand the set of nodal predicates. At the moment indicator values are only able to belong to the level but later it is planned to add the relationships between several indicators within a predicate. An important area is the study of applicability of the approach for BUY/SELL/HOLD signals.

Table III
EBAY results

Year	Sharpe	B&H Sharpe	Return	B&H returns	Max drawdown	Trades Count
2007 Train	1.90	0.36	47.86%	9.01%	9.78%	39
2008 Test	-0.77	-1.42	-26.94%	-52.55%	40.55%	24
2008 Train	2.70	-1.42	74.23%	-52.55%	6.64%	33
2009 Test	3.38	1.24	123.44%	55.25%	8.52%	37
2009 Train	2.92	1.24	78.65%	55.25%	8.14%	36
2010 Test	0.83	0.62	11.11%	16.91%	6.39%	12
2010 Train	1.49	0.62	24.08%	16.91%	6.39%	27
2011 Test	0.85	0.24	12.15%	4.68%	7.84%	7
2011 Train	1.45	0.24	25.98%	4.68%	8.23%	30
2012 Test	1.72	1.73	21.82%	58.61%	4.96%	20
2012 Train	2.23	1.73	55.22%	58.61%	7.50%	55
2013 Test	-1.57	0.07	-21.31%	1.61%	27.20%	31
2013 Train	1.36	0.07	15.80%	1.61%	2.86%	39
2014 Test	0.68	0.20	8.71%	4.62%	6.12%	37
2014 Train	3.20	0.20	28.06%	4.62%	3.05%	28
2015 Test	-0.38	-0.69	-28.73%	-45.02%	48.15%	25
2015 Train	3.71	-0.69	55.73%	-45.02%	2.98%	48
2016 Test	0.14	0.45	3.32%	11.61%	18.19%	16

Table IV
INTC results

Year	Sharpe	B&H Sharpe	Return	B&H returns	Max drawdown	Trades Count
2007 Train	1.49	1.00	35.18%	26.54%	11.73%	36
2008 Test	-0.57	-0.93	-24.27%	-39.51%	38.93%	19
2008 Train	3.28	-0.93	68.39%	-39.51%	2.63%	45
2009 Test	1.35	0.90	18.79%	30.67%	5.21%	49
2009 Train	4.09	0.90	94.73%	30.67%	3.22%	36
2010 Test	-1.11	0.01	-7.79%	0.04%	13.79%	11
2010 Train	1.55	0.01	20.00%	0.04%	4.01%	12
2011 Test	0.66	0.55	6.92%	13.98%	1.16%	7
2011 Train	1.23	0.55	24.10%	13.98%	9.28%	41
2012 Test	0.62	-0.86	8.90%	-14.49%	8.15%	30
2012 Train	0.64	-0.86	7.47%	-14.49%	6.22%	47
2013 Test	0.48	0.85	6.76%	17.98%	6.00%	30
2013 Train	2.73	0.85	31.64%	17.98%	4.69%	30
2014 Test	3.53	1.59	51.25%	39.15%	4.03%	38
2014 Train	3.53	1.59	51.25%	39.15%	4.03%	38
2015 Test	3.11	-0.10	45.35%	-2.03%	2.43%	42
2015 Train	3.11	-0.10	45.35%	-2.03%	2.43%	42
2016 Test	-0.35	0.30	-0.91%	6.75%	7.26%	17

Table V
ORCL results

Year	Sharpe	B&H Sharpe	Return	B&H returns	Max drawdown	Trades Count
2007 Train	2.38	0.96	35.94%	26.42%	4.23%	48
2008 Test	0.12	-0.38	2.86%	-20.45%	15.15%	42
2008 Train	2.74	-0.38	46.03%	-20.45%	6.35%	43
2009 Test	4.25	1.04	94.30%	32.90%	2.49%	46
2009 Train	4.30	1.04	118.95%	32.90%	3.29%	68
2010 Test	1.82	0.99	29.85%	24.37%	5.00%	58
2010 Train	1.58	0.99	21.62%	24.37%	3.68%	34
2011 Test	-0.27	-0.50	0.40%	-16.88%	6.75%	12
2011 Train	1.30	-0.50	22.14%	-16.88%	14.05%	40
2012 Test	0.73	1.19	8.22%	25.65%	4.98%	35
2012 Train	1.94	1.19	34.01%	25.65%	8.55%	32
2013 Test	-0.38	0.39	-5.24%	8.61%	17.23%	30
2013 Train	4.10	0.39	35.41%	8.61%	1.92%	43
2014 Test	3.89	0.89	34.14%	18.79%	1.98%	39
2014 Train	4.22	0.89	42.84%	18.79%	2.19%	49
2015 Test	2.19	-0.85	25.82%	-14.37%	5.39%	42
2015 Train	1.97	-0.85	15.50%	-14.37%	2.16%	29
2016 Test	-1.48	0.33	-8.11%	6.92%	9.41%	16

VI. Conclusion

As it can be seen from the presented examples, on average in 6 out of 9 cases the integral Sharpe ratio was better with our system than with the market. At the same time, the maximum loss of capital was 48% against 65% for a simple investment. This means that there are stocks and periods for which this approach can be effective. The number of entries into the long position varies from 58 to 7, which indicates a low sensitivity of the rules received. This may be due to the fact that a period of 1 year was taken as a sliding interval to determine the level ranges. Reducing this interval leads to an increase in the speed of the system reaction, but more false signals emerge.

It should be noted that during the training this trade system is adjusted to a certain ratio of trend and consolidation movements of the share price. If in the next period this ratio changes, then the trader can expect whether profit or loss. This fact is inherent in most algorithmic trading strategies based on methods of the technical analysis.

References

[1] M.J John, "Technical analysis of the financial markets: a comprehensive guide to trading methods and applications (2nd ed.)", New York Institute of Finance, 1999.

[2] R. Morta, E. Dadios "Proposed system for predicting Buy, Hold and Sell recommendations for a publicly listed Philippine company using computational intelligence", *TENCON 2015-2015 IEEE Region 10 Conference*, 2015, pp. 1-8.

[3] S. Yodphet, S. Rimcharoen, and Leelathakul, "LARG: Loss avoidance technical trading rules using genetic algorithm. In Knowledge and Smart" *Technology (KST), 2016 8th International Conference*, 2016, pp. 33-38.

[4] F. Giacomel, R. Galante, and A. Pereira. "An Algorithmic Trading Agent based on a Neural Network Ensemble: a Case of Study in North American and Brazilian Stock Markets.", *In Web Intelligence and Intelligent Agent Technology (WI-IAT)*, 2015, IEEE/WIC/ACM International Conference on (Vol. 2, pp. 230-233). IEEE.

[5] B.A. Sheldon, "Binary Decision Diagrams", *IEEE Transactions on Computers*, pp. 509–516, 1978.

[6] L. Rokach and O. Maimon, "Data mining with decision trees: theory and applications", 2014.

[7] F. X. Satriyo D. Nugroho, T. Bharata Adji, S. Fauziati, "Decision support system for stock trading using multiple indicators decision tree", *The 1st International Conference on Information Technology, Computer, and Electrical Engineering*, 2014.

[8] C. N. Ochotorena, C. A. Yap , E. Dadios and E. Sybingco, "Robust stock trading using fuzzy decision trees", *Computational Intelligence for Financial Engineering and Economics (CIFEr)*, pp. 1-8, 2012.

[9] R. Dash and P.K. Dash, "A hybrid stock trading framework integrating technical analysis with machine learning techniques", *The Journal of Finance and Data Science*, pp. 42-57, 2016.

[10] D. Iskrich, D. Grigoriev "Generating Long-Term Trading System Rules Using a Genetic Algorithm Based on Analyzing Historical Data", *Proceedings of the International FRUCT Conference on Intelligence, Social Media and Web*, 2017

[11] J. Potvin, P. Soriano, M. Vallée "Generating trading rules on the stock markets with genetic programming", *Computers & Operations Research*, 2004.

[12] Iba, Hitoshi, and Claus C. Aranha "Practical Applications of Evolutionary Computation to Financial Engineering", Springer, 2012.

[13] Nur Aini Rakhmawati, Erma Suryani "DECISION FOR BUYING AND SELLING STOCK WITH DECISION TREE ALGORITHM", Sepuluh Nopember Institute of Technology.

[14] Al-Radaideh, A. Qasem, Aa Assaf, and Eman Alnagi. "Predicting stock prices using data mining techniques.", *The International Arab Conference on Information Technology* (ACIT'2013), 2013.

[15] A. Ghandar, Z. Michalewicz, M. Schmidt, T. To, and R. Zurbrugg, "Computational Intelligence for Evolving Trading Rules", *IEEE TRANSACTIONS ON EVOLUTIONARY COMPUTATION 1*, pp. 71-86, 2009.

[16] W.F. Sharpe, "The sharpe ratio", *The journal of portfolio management*, 1994, pp. 49-58.

Smoothing Algorithm for Magnetometric Data

Mariya PASHKOVA, Sofya ILINYKH,
Saint-Petersburg State University
St. Petersburg, Russia
Pashkova.Mariya@rambler.ru, Ilinykh-Sofya@mail.ru

Natalia GRAFEEVA
Saint-Petersburg State University
St. Petersburg, Russia
N.Grafeeva@spbu.ru

***Abstract*— This article presents the problem of searching useful magnetic anomalies by using the magnetometric survey methods. Metal debris (spot anomalies) often create local interference that makes it difficult to find vast spatial anomalies caused by the remnants of ancient buildings, such as walls of houses, wells, dugouts, etc. The main purpose of this study is to develop an algorithm that eliminates such interference. We explore the methods of working with magnetometric data and design an algorithm which is based on seeking spot anomalies and using the arithmetic method of smoothing spatial data. We test the final algorithm on many real datasets, obtained during excavations in the Crimea, and it shows good results.**

Keywords— magnetometric, spatial data smoothing, arithmetic mean of spatial data

I. INTRODUCTION

In archaeology there are different methods of determining the location of future excavation sites. One of those methods, *magnetometric survey*, is based on magnetic properties of hidden underground objects. The main advantage of this method is a relatively low cost.

The process of magnetometric survey is described in [6]. Magnetometric survey is usually performed on a square surface area of proposed excavation site with the size ranging from 50 by 50 meters to 100 by 100 meters**.** That square area is called a *quadrangle*. At first, a modulus of magnetic induction of the geomagnetic field is measured at previously determined observation points. Then, collected data is preprocessed (e.g. to take into account global geomagnetic field changes). Then, researchers build a map of a quadrangle, estimate the probability of finding objects of archaeological value. Finally, a conclusion on conducting excavations is made. Sometimes, anomalies (deviation from the average value of magnetic field), created by past human activity, can be spotted on a map.

Often, small metal junk creates interferences represented in numerical data by sharp value drops causing difficulties in searching for vast spatial anomalies created by ancient buildings (houses, wells and burial mounds). The main objective of this study is developing and testing an algorithm which will allow smoothing out sharp data drops thus eliminating interferences caused by metal junk.

II. DATA DESCRIPTION

Provided data is a result of a magnetometric survey that was conducted on excavation sites in Crimea [3]. Magnetometric data is an example of spatial data: each data set consists of 15000 three-dimensional points. The first two components are coordinates; the third one is the difference between value of magnetic field at a given point and the average value of magnetic field at the quadrangle rounded to the nearest whole (measured in nT). The point coordinates are measured with 0.5 meters increments. Such spatial data can be represented as a two-dimensional image where the third component plays the role of color.

The visualization of this data can be created in RStudio IDE which is also capable of providing the data statistics useful for some algorithms implementation. Table I-II and Fig.1 provide examples of input data, visualization and statistics calculation in RStudio IDE.

TABLE I. AN EXAMPLE OF INITIAL DATA

X	0	0	0	0	0	0	0	0	…
Y	49.5	50	50.5	51	51.5	52	52.5	53	…
value	-3	3	156	-3	4	0	-6	-11	…

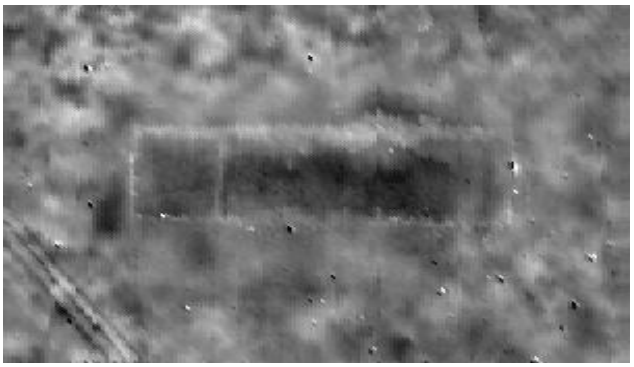

Fig. 1. Dataset visualization.

TABLE II. DATA STATISTICS

	x	y	Value
Min	0.0	49.0	-322.000
1st Qu.	12.5	62.0	-6.000
Median	25.0	75.0	-1.000
Mean.	25.0	75.0	-1.754
3rd Qu.	37.5	88.0	2.000
Max	50.0	101.5	285.000

III. OVERVIEW OF METHODS FOR WORKING WITH MAGNITOMETRIC DATA

There are three groups of methods are usually used to process magnetometric data.

A. Working with data as an image

Magnetometric data can be easily visualized [3]. For example, magnetic value can serve as a color of the black-to-white gradient. This way researchers can process data as an image, apply filtration [10], stretching, etc. Some of these methods are presented in [3].

There are two different ways of processing images within the context of the task:

- Methods based on processing pixels directly. For example, logarithmic and power transformations, application of Gaussian filter [8] and Sobel operator [9], etc.
- Methods, based on modification of the Fourier spectrum of the image [9]

B. Working with numerical data, based on its nature

Considering the nature of data, it can be processed as numbers. In each point x of quadrangle input data is represented as a sum

$$B(x) = R(x) + P(x) + A(x)$$

where R is a level of regional background magnetic field, P – interference, A – local anomalies.

Applying certain algorithms (analytical continuation of the field into the upper half plane, Kalman filter, etc.) the researcher is trying to distinguish and deduct the background and the interference to find important local anomalies. The main source of information about those methods is [2].

C. Other approaches

Lately, a variety of methods (clusterization [7], patterns recognition [11], etc.), that can be applied to magnetometric data, but do not take in consideration the nature of the data, have been suggested.

Often image processing methods cannot provide the desired results. Application of those methods sometimes causes strong distortion of the image (reasons include aforementioned metal objects), which hinders the search for useful spatial anomalies. Methods, based on the nature of the data, are interesting and sometimes useful, but are costly. At the same time simple disposal of interference, created by metal objects, can drastically change the visual image of the analyzed data. With that in mind in this study attention was focused on finding an algorithm, which processes data to eliminate interference that hinders the search of vast space anomalies.

IV. CHOOSING A METHOD FOR SMOOTHING MAGNETOMETRIC DATA

In Fig. 2 black dots surrounded by white areas are clearly visible (in numerical data they are represented by large deviations from average value of the quadrangle). These are the point anomalies, caused by metal junk or instrumentation errors. Usually, they cover no more than 1-2% of the whole area. They are not always objects of archaeological research, but they can obstruct analysis of more significant objects.

So, the idea is to smooth the values in such points. Then, small meaningful deviations will be more noticeable when visualized.

This can be achieved, for example, by a simple method of arithmetic mean, but instead of time series, space surrounding the interference points can be used.

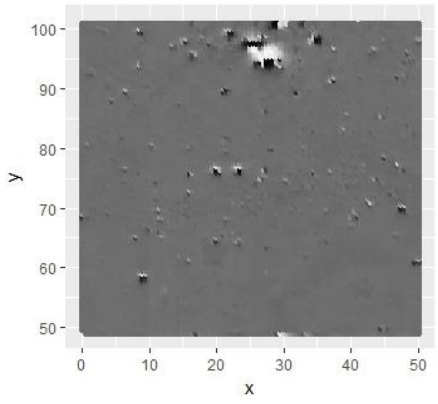

Fig. 2. Example of input data with point anomalies

V. DEVELOPMENT OF ALGORITHM

During data analysis it was observed that in places of metal junk concentration of large deviations occur, both positive and negative. At the same time, points where such situation happens make up a small portion of all points in the quadrangle. There are two approaches for finding point anomalies based on these ideas.

A. Frequency-based approach

First approach is based on sorting magnetic value by frequency of appearance on a set area. Afterwards points, that appear rarely enough (i.e. make up no more than M percent of the area), are selected. Table II shows a table of magnetic values and their corresponding frequencies with values, selected to be smoothed, highlighted. In this example M equals 2%. Fig. 3 shows values of the examined area with selected values marked in dark grey.

B. Deviation-based approach

In the second approach points are considered to be interference when their value deviates from the average by more than parameter δ.

Fig. 4 shows selected points with $\delta = 9$. The average value is -3.5.

After the interference points have been found, their values should be smoothed. Following solution has been considered: points are assigned an average value of some neighborhood that does not include point anomalies.

Firstly, two extreme cases have been implemented. In the first case an interference point was assigned a value of the closest (using the Euclidean distance) point that is not a subject to smoothing. In the second case, the average value of the quadrangle is assigned to all interference points. As can be seen in Fig. 5-7, results of applying these extreme cases are not what we are looking for: smoothing is too rough and inaccurate, distortions are still visible and local anomalies become blurred, thus it does not solve our problem. In both cases the interference points were found using deviation method.

TABLE III. MAGNETIC VALUES AND A PERCENT OF AREA COVERED BY EACH VALUE

Magnetic values	-14	-13	-9	3	4	-16	-12	-10	-6	-3	5	6	-8	2	7	-1	-4	-5	0	-2
Percent of area	2%	2%	2%	2%	2%	3%	3%	3%	3%	3%	3%	5%	5%	5%	6%	6%	8%	9%	13%	16%

2	2	5	6	6	0	-7	-5
-3	-1	0	5	4	2	-1	-2
-5	-2	-2	3	-2	0	0	0
-2	-5	-4	0	-8	-4	-1	-3
-2	-6	-7	-2	-10	-10	-4	-6
-5	-5	-12	-5	-8	-16	-9	-4
-1	-2	-7	-4	-8	-14	-13	0
1	0	-2	-2	-5	-12	-16	-7

Fig. 3. Example of finding interference point using a frequency method, parameter M = 2%

2	2	5	6	6	0	-7	-5
-3	-1	0	5	4	2	-1	-2
-5	-2	-2	3	-2	0	0	0
-2	-5	-4	0	-8	-4	-1	-3
-2	-6	-7	-2	-10	-10	-4	-6
-5	-5	-12	-5	-8	-16	-9	-4
-1	-2	-7	-4	-8	-14	-13	0
1	0	-2	-2	-5	-12	-16	-7

Fig. 4. Example of finding interference point using a deviation method, parameter δ = 9

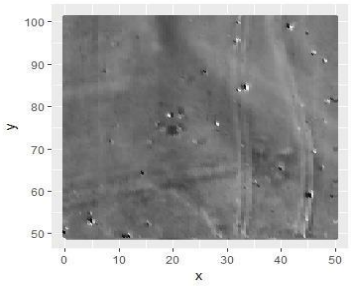

Fig. 5. Input data

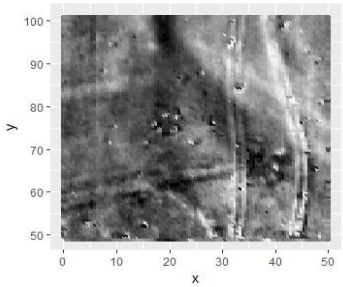

Fig. 6. Single-point smoothing result

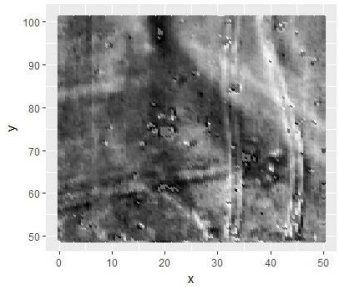

Fig. 7. Smoothing using all quadrangle

2	2	5	6	6	0	-7	-5	-5	-5	-6
-3	-1	0	5	4	2	-1	-2	-5	-3	-4
-5	-2	-2	3	-2	0	0	0	-2	-1	-3
-2	-5	-4	0	-8	-4	-1	-3	-3	-1	2
-2	-6	-7	-2	-10	-10	-4	-6	-3	-3	3
-5	-5	-12	-5	-8	-16	-9	-4	1	-1	2
-1	-2	-7	-4	-8	-14	-13	0	1	-2	2
1	0	-2	-2	-5	-12	-16	-7	-5	-3	-1
1	-1	-2	-2	-3	-7	-16	-12	-11	-12	-9
-1	-2	-5	1	-3	-4	-13	-18	-11	-11	-6
-1	-3	-1	0	1	-2	-9	-16	-12	-7	-7
-1	-3	-2	2	3	0	-4	-11	-12	-4	3
-2	-2	2	1	-1	-2	2	-2	-9	-9	2
-2	-2	0	3	3	0	0	-1	-12	-13	1

Fig. 8. Example of located interference points and their neighbourhoods.

During the work it was found that in most cases 8 is an optimal number of points in a neighborhood. Fig. 8 shows interference points (marked in gray) and their corresponding neighborhoods (marked in light gray). The interference points were found using the deviation method, where parameter M is equal 9.

Fig. 9-10 present results of the resulting algorithm with different values of parameters δ and M. It can be concluded that frequency method deforms the initial image less, but it is less effective in getting rid of interference points

For the next step of the work it was decided to divide the initial quadrangle into smaller parts and then apply aforementioned approaches to each of them. Figures 11-12 show the results of dividing the quadrangle into 25 parts.

Results are improved and the frequency method performs better than the deviation method. Thus, by dividing initial data into separate parts and smoothing rarely occurring values, desired results can be achieved.

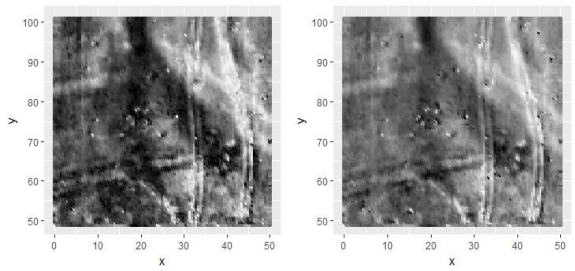

Fig. 9. Frequency method, M = 0.45% and M = 4%

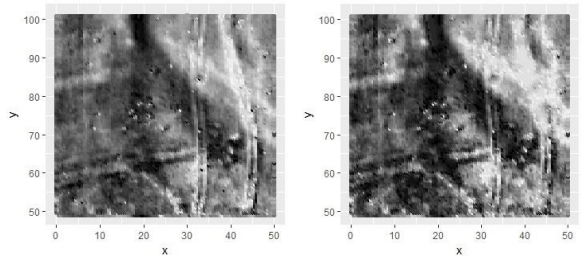

Fig. 10. Deviation method, δ = 13 and δ = 9

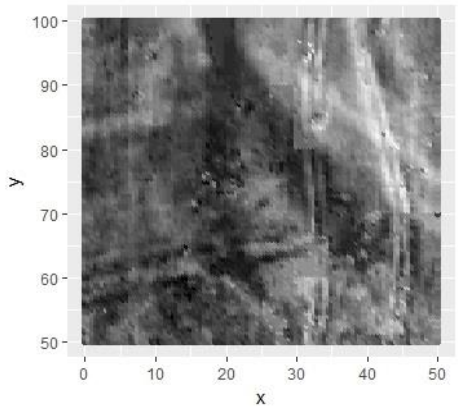

Fig. 11. Deviation method, δ = 8

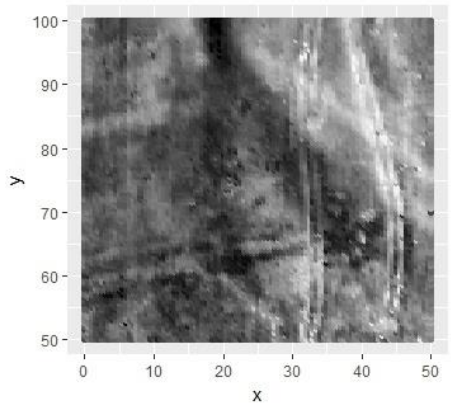

Fig. 12. Frequency method, M = 7%

VI. Conclusion

During the work spatial magnetometric data was smoothed using arithmetic average values helping in identifying local magnetic anomalies.

As can be seen in Fig. 13-14, developed algorithm processes initial data in such way, that an archaeology specialist would have no trouble determining the contours of the object and deciding if the excavation is advisable at the sector in question.

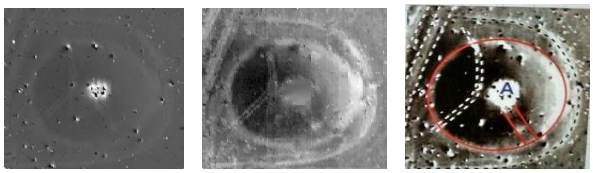

Fig. 13. Initial data, the result of the algorithm and interpretation of archeologists[1]

An advantage of the algorithm is that it transforms numbers, not images. Therefore it can be used not only by itself, but also as preprosessing step for other methods, for example, clustering[4],[5] or filtering[10].

Sample data

Result of Algorithm

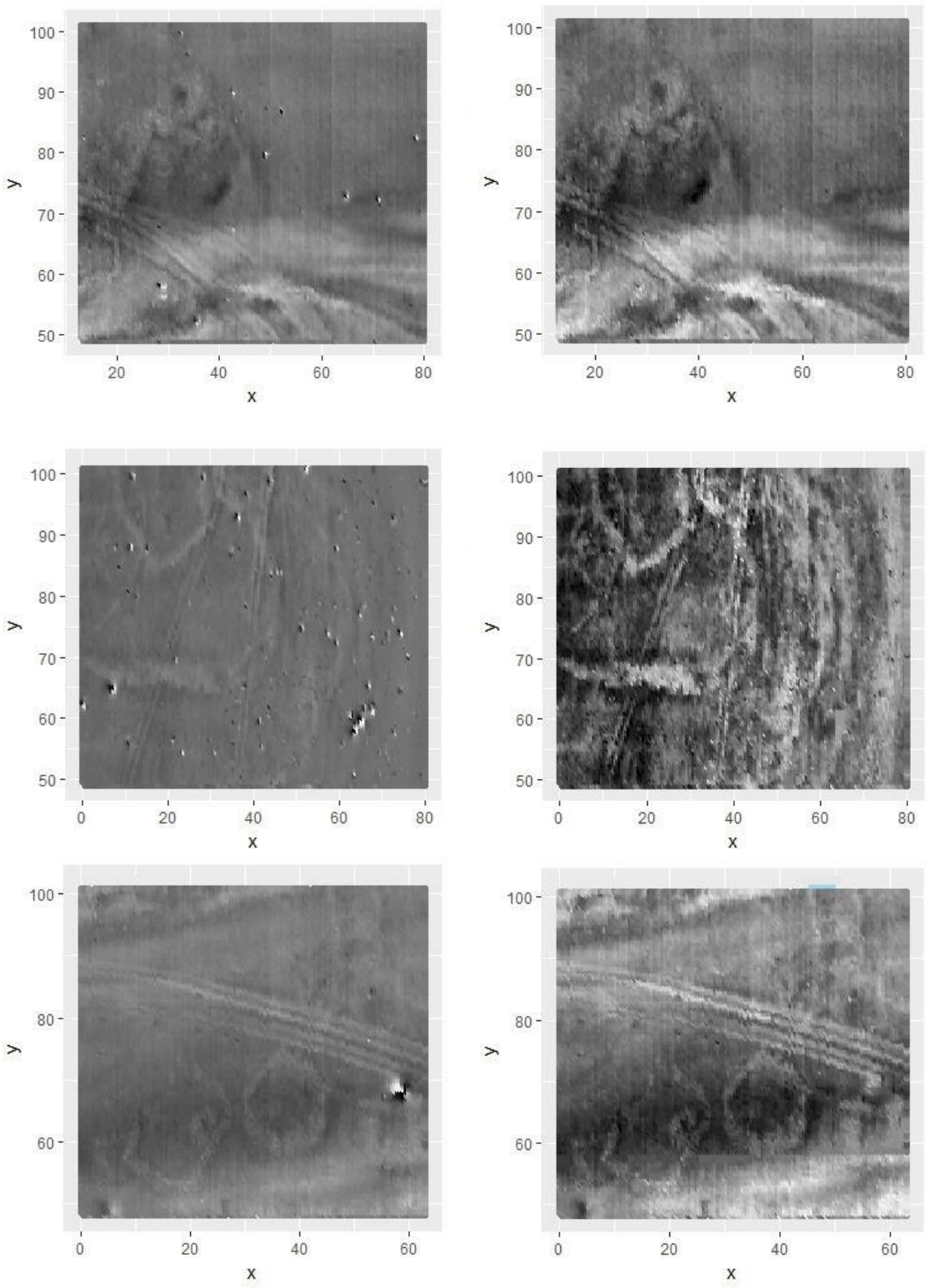

Fig. 14. Another results of testing algorithm on real data

ACKNOWLEDGMENT

The authors would like to thank the Low Field Magnetic Resonance Laboratory in Saint Petersburg State University for provided data, which were obtained as a result of excavations in the Crimea.

REFERENCES

[1] V.A.Kutajsov, T.N.Smekajlov, Materials for archaeological map of Crimea: History and archaeology of Northwestern Crimea, Simferopol: Phoenix Company, 2014. [in Russian]

[2] Dudkin V.P. and Koshelev I.N., Method for complex interpreting the results of magnetometric survey of archaeological landmarks, Vostochnoevropeisk. Arkheol. Zh., No. 3(6), 2002.

[3] V. S. Mikhailova, N. G. Grafeeva, E. G. Mikhailova, A. V. Chudin, Magnetometry Data Processing to Detect Archaeological Sites, Pattern Recognition and Image Analysis, 2016, Vol. 26, No. 4, pp. 789–799. ©Pleiades Publishing, Ltd., 2016.

[4] S. V. Belim, P. V. Kutlunin, Boundary Extraction in Images Using A Clustering Algorithm, Computer Optic, Vol. 39, No. 1, pp. 119–124, 2015. [in Russian]

[5] Elena Volzhina ; Andrei Chudin ; Boris Novikov ; Natalia Grafeeva ; Elena Mikhailova, Discovering geo-magnetic anomalies: a clustering-based approach, 2016, Page(s):1–7 Intelligence, Social Media and Web (ISMW FRUCT), 2016 International FRUCT Conference

[6] V. K. Khmelevskoi, Geophysical Methods for Investigating the Earth Crust, Dubna: Dubna Univ., 1999. [in Russian]

[7] Barsegjan A.A., Kuprijanov M.S., Holod I.I., Tess M.D. and Elizarov S.I., Data and process analysis, St. Petersburg: BHV-Petersburg, 2009. [in Russian]

[8] Duda and P. Hart, Pattern Classification and Scene Analysis, New York: John Wiley and Sons, 1973.

[9] Jian-Lei Liu Da-Zheng Feng, "Two-dimensional multi-pixel anisotropic Gaussian filter for edge-line segment", Image and Vision Computing archive, vol. 32, Jan. 2014, pp. 37–53.

[10] R. Gonzalez and R. Woods, Digital Image Processing, New Jersey: Prentice Hall, 2008.

[11] Mesteckiy L. M., Mathematical methods of pattern recognition. Lecture course, Moscow: Moscow State University, Faculty of Mathematic and Cybernetics, 2002. [in Russian]

A study of PosDB Performance in a Distributed Environment

George Chernishev
Saint-Petersburg State University,
JetBrains Research
Email: chernishev@gmail.com

Viacheslav Galaktionov
Saint-Petersburg State University
Email: viacheslav.galaktionov@gmail.com

Valentin Grigorev
Saint-Petersburg State University
Email: valentin.d.grigorev@gmail.com

Evgeniy Klyuchikov
Saint-Petersburg State University
Email: evgeniy.klyuchikov@gmail.com

Kirill Smirnov
Saint-Petersburg State University
Email: kirill.k.smirnov@math.spbu.ru

***Abstract*—PosDB is a new disk-based distributed column-store relational engine aimed for research purposes. It uses the Volcano pull-based model and late materialization for query processing, and join indexes for internal data representation. In its current state PosDB is capable of both local and distributed processing of all SSB (Star Schema Benchmark) queries.**

Data, as well as query plans, can be distributed among network nodes in our system. Data distribution is performed by horizontal partitioning.

In this paper we experimentally evaluate the performance of our system in a distributed environment. We analyze system performance and report a number of metrics, such as speedup and scaleup. For our evaluation we use the standard benchmark — the SSB.

I. Introduction

Column-stores have been actively investigated for the last ten years. Many open-source [1], [2], [3], [4], [5] and commercial [6], [7], [8] products with different features and aims have been developed. The core design issues such as compression [9], [10], materialization strategy [11], [12] and result reuse [13] got significant attention. Nevertheless, distribution of data and control in disk-based column-store systems was not studied at all.

The reason for this is that none of open-source systems are truly distributed, although some of them [5], [14] support limited distribution. Several commercial systems, such as Vertica [6], are distributed but closed-source. To the best of our knowledge, no investigation of distribution aspects in column-stores has been conducted.

To address this problem, we are developing a disk-based distributed relational column-store engine — PosDB. In its current state it is based on the Volcano pull-based model [15] and late materialization. Data distribution is supported in the form of horizontal per-table partitioning. Each fragment can be additionally replicated on an arbitrary number of nodes, i.e. our system is partially replicated [16]. Control (query) distribution is also supported: parts of a query plan can be sent to a remote node for execution.

In our earlier studies [17], [18] we have described opportunities offered by such a system and sketched its design. Later, an initial version of our system, PosDB, was presented and its high-level features were described [19].

In this paper, we present the results of first distributed experiments with PosDB. We evaluate system performance by studying several performance metrics, namely speedup and scaleup. For evaluation we use a standard OLAP benchmark — the Star Schema Benchmark [20].

The paper is structured as follows. The architecture of the system is described in detail in section II. A short survey of distributed technology in databases is presented in section I. In section III we discuss used metrics (scaleup and speedup). The experimental evaluation and its results are presented in section IV.

II. Related Work

There is a shortage of distribution-related studies for relational column-oriented databases [18]. The main reasons are the scarcity of research prototypes and the drawbacks in the existing ones.

Two research prototypes of distributed column-store systems are known to the authors — [5], [14]. Both studies use an in-memory DBMS, MonetDB, some of whose parts were rewritten to add distribution-related functionality. This approach cannot be considered "true" distribution, because, in general, it restricts the pool of available distributed processing techniques. Developers have to take into account the architecture of the underlying centralized DBMS in order to employ it. Unfortunately, the degree of these restrictions is unclear for the aforementioned systems.

Another distributed column-store, the ClickHouse system, is an industrial open-source disk-based system. However, there are two issues with this system. Firstly, it was open-sourced only recently, in 2016, and there are no research papers based on this system, known to the authors. Secondly, it has several serious architectural drawbacks: a very restricted partitioning [21] and issues with distributed joins [22].

At the same time, there are hundreds, if not thousands, of papers on the subject in application to row-stores [16], [23].

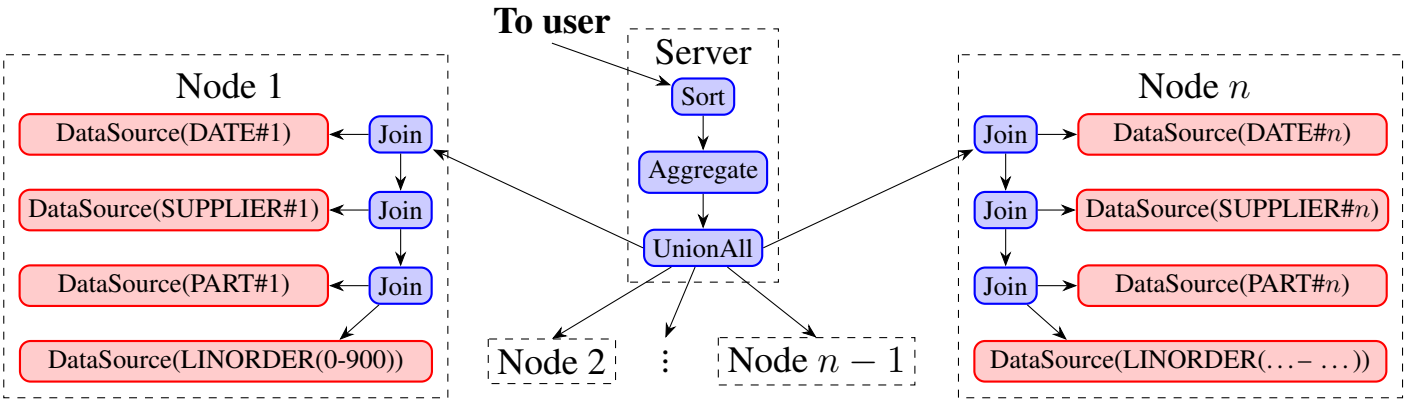

Fig. 1. Example of distributed query plan — distributed query 2.1 from SSB

III. PosDB: Architecture

PosDB uses the Volcano pull-based model [15], so each query plan is represented as a tree with operators as vertexes and data flows as edges. All operators support the "open()–next()–close()" interface and can be divided into two groups:

- Operators that produce blocks of positions.
- Operators that produce individual tuples.

PosDB relies on late materialization, so operators of the second type are always deployed on the top of a query tree. They are used to build tuples from position blocks and to perform aggregation. The whole tree below the materialization point consists of operators which return blocks.

Each position block stores several position vectors of equal length, one per table. This structure is essentially a join index [24], [25], which we use to process a chain of join operators.

Currently we have the following operators that produce join indexes:

- `DataSource`, `FilteredDataSource`, operators for creating initial position streams. The former generates a list of contiguous positions without incurring any disk I/O, while the latter conducts a full column scan and produces a stream of positions whose corresponding values satisfy a given predicate. These operators are the only possible leaves of a query tree in our system;
- `GeneralPosAnd`, `SortedPosAnd`, binary operators that intersect two streams of positions belonging to a single table. `SortedPosAnd` expects both streams to be sorted, while `GeneralPosAnd` does not.
- `NestedLoopJoin`, `MergeJoin`, `HashJoin`, binary operators, which implement the join operation using different classic algorithms;
- `UnionAll` — an n-ary operator that processes its subtrees in separate threads and merges their output into a single stream in an arbitrary order;
- `ReceivePos` — an ancillary unary operator that sends a query plan subtree to a remote node, receives join indexes from it and returns them to the ancestor;
- `Asynchronizer` — an ancillary unary operator that processes its child operator in a separate thread and stores the results in the internal fixed-size buffer;

and the following that produce tuples:

- `Select`, for tuple reconstruction;
- `Aggregate`, for simple aggregation without grouping;
- `SGAggregate`, `HashSGAggregate`, for complex aggregation with grouping and sorting;
- `SparseTupleSorter`, for tuple sorting.

As can be seen, query distribution is maintained on the operator level using two ancillary operators: `ReceivePos` and `UnionAll`. It should be emphasized, that a multithreaded implementation of UnionAll is essential here, because sequential execution would definitely incur severe waiting penalties, completely negating the benefits of a distributed environment. Figure 1 presents an example of a distributed query plan for the query 2.1 from the SSB which is as follows:

```
select sum(lo_revenue), d_year, p_brand1
from lineorder, date, part, supplier
where lo_orderdate = d_datekey
  and lo_partkey = p_partkey
  and lo_suppkey = s_suppkey
  and p_category = 'MFGR#12'
  and s_region = 'AMERICA'
group by d_year, p_brand1
order by d_year, p_brand1;
```

Also, there is a notion of data readers in our system. Data reader is a special entity used for reading attribute values corresponding to the position stream. Currently, we support the following hierarchy of readers:

- `ContinuousReader` and `NetworkReader`, basic readers for accessing a local or remote partition respectively;
- `PartitionedDataReader`, an advanced reader for accessing the whole column, whose partitions are stored on one or several machines. For each partition it creates a corresponding basic reader to read values from either a local drive or a remote server. Then, using information

from the catalog, a `PartitionedDataReader` automatically determines which reader to use for a position in a join index;
- `SyncReader`, an advanced reader responsible for synchronous reading of multiple attributes. This reader maintains a `PartitionedDataReader` for each column.

Initially, a query plan does not contain readers. Each operator creates readers on demand and feeds them positions to receive necessary data. Operators that materialize tuples use `SyncReader`, others usually employ `PartitionedDataReader`. Using these advanced readers allows operators to be unaware of data distribution.

IV. General Considerations and Used Metrics

Distributing the DBMS has two important goals [16]: improving performance and ensuring easy system expansion. These goals are usually evaluated using two metrics [26]: scaleup and speedup.

Speedup reflects the dependency of system performance on the number of processing nodes under the fixed workload. Thus, it shows the performance improvement that can be achieved by using additional equipment and without system redesign.

Linear speedup is highly desired but rarely can be achieved in practice. Superlinear speed points out an unaccounted distributed system resources or poor algorithm. So, a good system should try to approximate linear dependency as well as it can.

Scaleup is a similar metric that reflects how easy it is to sustain the achieved performance level under an increased workload. The number of processing nodes and a size of the workload are increased by the same number of times. An ideal system achieves linear scaleup, but again, it is rarely achievable in practice.

Workload can be increased either by increasing the number of queries or the amount of data. The former is the transactional scaleup and the latter is the data scaleup. We do not investigate transactional scaleup, because PosDB is oriented towards OLAP processing — a kind of processing that implies long-running queries. Taniar et al. [26] argue that transactional scaleup is relevant in transaction processing systems where the transactions are small queries. On the other hand, data scaleup is very important for our system because the amount of data in OLAP environments can exhibit feasible growth.

V. Experiments

In order to conduct the experiments, we selected the following setup of data and query distributions. We designate one processing node as a server and assign it several worker nodes. The server only processes user requests, while the data is stored on worker nodes (see Figure 2). Each worker node stores a horizontal partition of the fact table (LINEORDER) along with the replicas of all other (dimension) tables. Dimension tables are always tiny compared to the fact table, so their replication incurs almost no storage overhead.

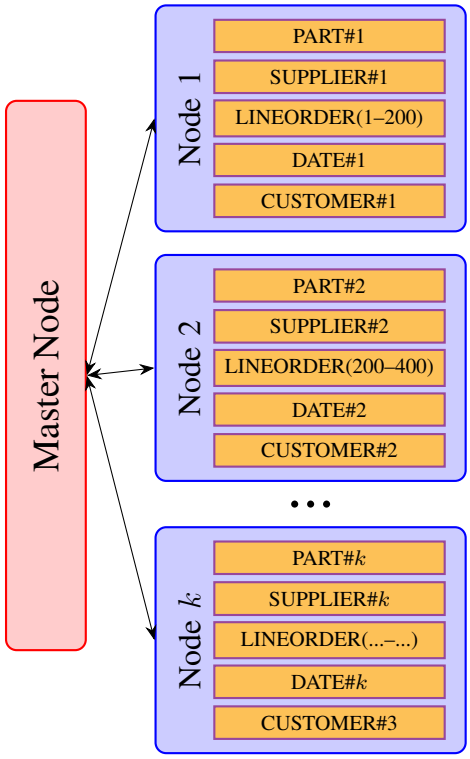

Fig. 2. Data distribution scheme in PosDB

Figure 1 shows the distributed query for query 2.1 from the workload. It illustrates the general approach which we follow in this paper for each query. The server is responsible for receiving data from worker nodes and for aggregation. Note that all queries in this benchmark can be distributed in such a manner that no inter node (worker node) communication is required.

A. Description of Experiments, Hardware and Software Experimental Setups

We consider three different experiments, all using the SSB workload:

1) The dependency of PosDB performance on SSB scale factor in a local (one node) case.
2) The speedup of PosDB, i.e. the dependency of the performance for a fixed workload (scale factor 50) on the number of nodes. The number of nodes includes server and $1, 2, 4, 6, 8$ worker nodes.
3) The scaleup of PosDB, i.e. the performance on $k = 1, 2, 4, 6, 8$ nodes for scale factor $10 * k$ workload.

These experiments are conducted on a cluster of ten machines connected by 1GB local network. Each machine has the following characteristics: Intel(R) Core(TM) i5-2310 CPU @ 2.90GHz (4 cores total), 4 GB RAM. The software used is Ubuntu Linux 16.04.1 (64 bit), GCC 5.4.0, JSON for Modern C++ 2.1.0.

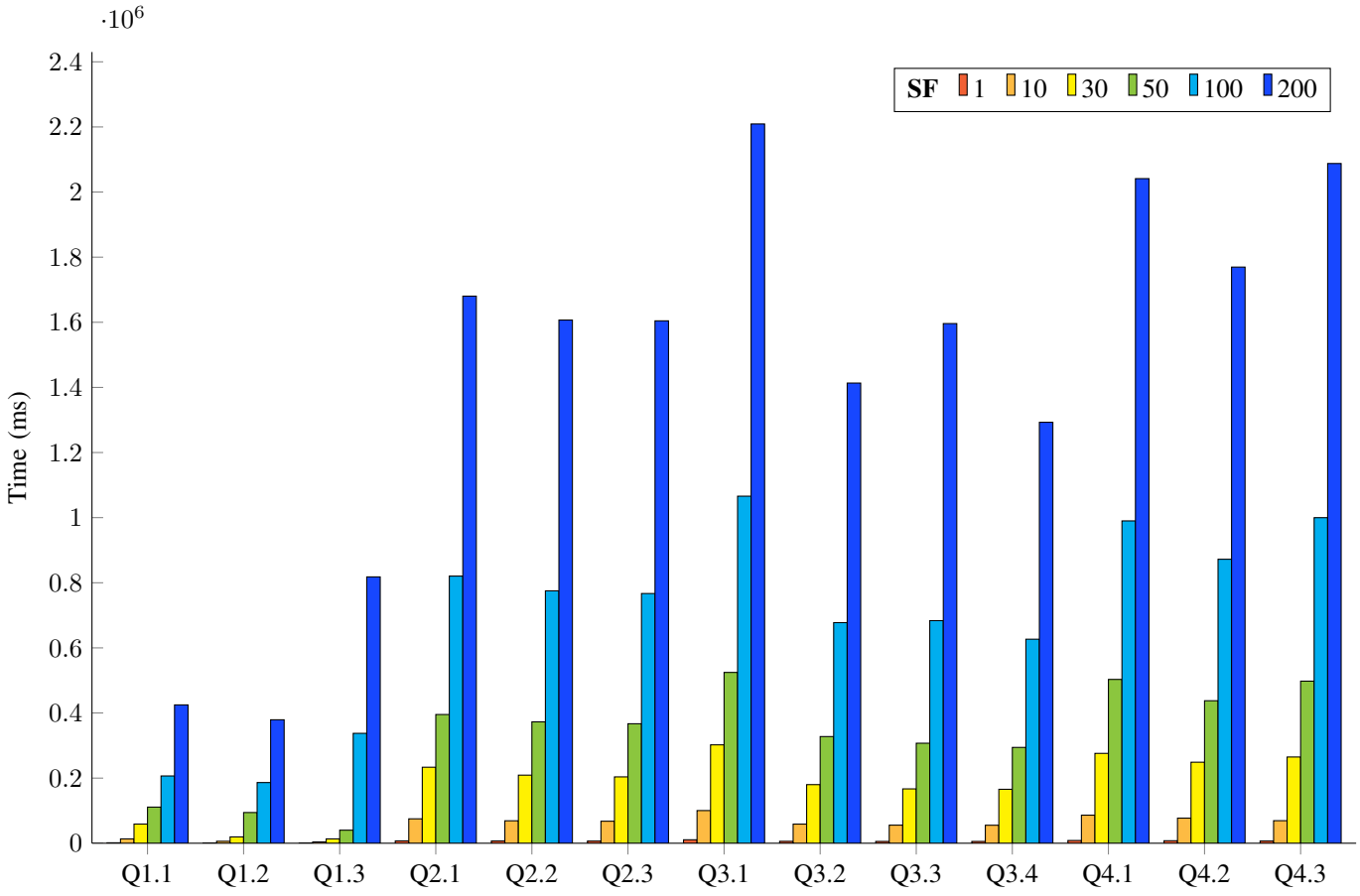

Fig. 3. Query performance from scale factor dependency in local case

B. Experiment 1

In this experiment we study PosDB behavior in a local case under the full SSB workload. We have chosen six different scale factors: $1, 10, 30, 50, 100, 200$. The results of this experiment are presented in Figure 3. It should be emphasized that logarithmic y-axis is used here.

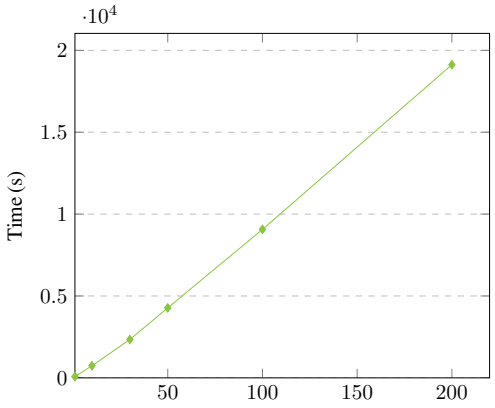

Fig. 4. Dependency of total SSB execution time from scale factor

After careful analysis, several interesting conclusions can be drawn:

- Although query 1.3 has a higher selectivity, its execution time is higher than that of the other queries of its flight. Perhaps it is due to a more expensive aggregation.
- Execution time of the queries from the second flight decreases with the increase in selectivity, as is to be expected.
- Query flight 3 reveals two interesting points. Query 3.1 is much more expensive than the others, because its first join operator returns a significantly higher number of records, thus loading the rest of the query tree. With high scale factors, query 3.3 become much more expensive than others. We suppose that it is due to intensive disk usage.
- Queries 4.1 and 4.2 behave in a very similar way, however the last join in the query 4.1 produces more results. This is the reason for the slightly extended run times for the whole query. Also, there is an anomaly in query 4.3 which still has to be explained. We plan to explore it in our further studies.

The total time of the whole workload is presented in Figure 4. In order to obtain this graph we summed up the run times of all queries described in the SSB. Essentially, this graph is just another representation of the information presented in Figure 3.

C. Experiment 2

You can see the results of the second experiment in Figure 5. Starting with 1, the number of nodes is increased by 2 with each step. The contents of the LINEORDER table (about 11 GBs) are evenly partitioned and distributed across them. Other tables are fully replicated. The red line shows how much faster the queries are executed when the number of nodes increases. The green line represents the "ideal" case, where the speedup grows linearly.

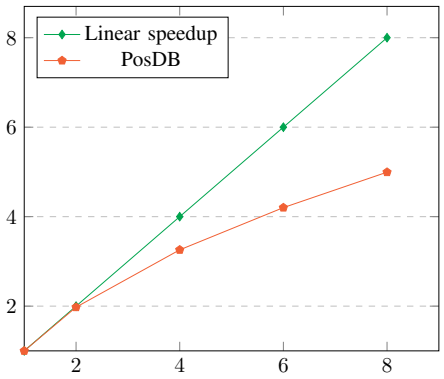

Fig. 5. Speedup from number of servers dependency in PosDB

As you can see, PosDB's performance increases when new nodes are added, although not linearly, which is because our system is yet in its infancy. We believe that such high overhead can be written off on the lack of a proper buffer manager, which means that the same data may be transferred over the network many times.

D. Experiment 3

In this experiment we measured PosDB data scaleup under scale factors $10, 20, 40, 60, 80$ on $1, 2, 4, 6, 8$ nodes. Data and query for each test configuration are distributed similar to the experiment 2. LINEORDER is partitioned between nodes, other tables are fully replicated. Then, parts of query plan that lie below aggregation (or tuple construction) are sent to different nodes, each with a DataSource operator for the corresponding LINEORDER partition. See Figures 2 and 1 for more details.

The following formula is used to calculate the scaleup metric:

$$scaleup(k) = \frac{(server + 1\, machine) execution time}{(server + k\, machines) execution time}.$$

We present the results of our experiments in Figure 6. To estimate the PosDB scaleup, we also plotted the "linear scaleup" and "no scaleup" cases. The former is a situation when scaleup is constant (ideal value) during all experiments. In the "no scaleup" case we assume that the amount of data grows linearly, but the computing power remains constant, so scaleup is calculated as follows:

$$scaleup(k) = \frac{1}{k}.$$

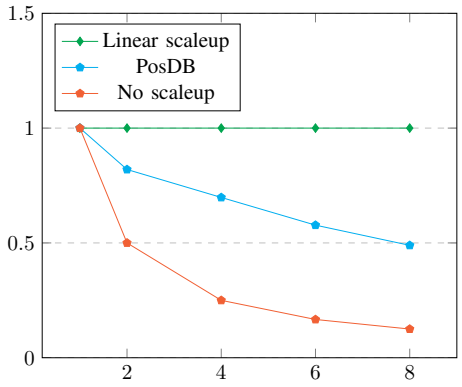

Fig. 6. Data scaleup from number of servers dependency in PosDB

We can see that PosDB scaleup is in $[0.5, 0.75]$ boundaries, slowly decreasing with the number of servers growing. Thus, comparing to the case "no scaleup", we can conclude that our system can offer a good scaleup.

VI. Conclusion

In this paper we presented an evaluation of PosDB, our distributed column-store query engine. We used the Star Schema Benchmark — a standard benchmark used for evaluation of OLAP systems. We studied several performance metrics, such as speedup and scaleup. In our experiments we were able to achieve scale factor 200 on a single machine, our system demonstrated sublinear speedup and a good data scaleup. The evaluation also allowed us to discover some anomalies and bottlenecks in our system. They are the subject of our future research.

References

[1] M. Stonebraker, D. J. Abadi, A. Batkin, X. Chen, M. Cherniack, M. Ferreira, E. Lau, A. Lin, S. Madden, E. O'Neil, P. O'Neil, A. Rasin, N. Tran, and S. Zdonik, "C-store: A Column-oriented DBMS," in *Proceedings of the 31st International Conference on Very Large Data Bases*, ser. VLDB '05. VLDB Endowment, 2005, pp. 553–564. [Online]. Available: http://dl.acm.org/citation.cfm?id=1083592.1083658

[2] S. Idreos, F. Groffen, N. Nes, S. Manegold, K. S. Mullender, and M. L. Kersten, "MonetDB: Two Decades of Research in Column-oriented Database Architectures," *IEEE Data Eng. Bull.*, vol. 35, no. 1, pp. 40–45, 2012. [Online]. Available: http://sites.computer.org/debull/A12mar/monetdb.pdf

[3] "Google. Supersonic library," https://code.google.com/archive/p/supersonic/, 2017, acessed: 12/02/2017.

[4] J. Arulraj, A. Pavlo, and P. Menon, "Bridging the Archipelago Between Row-Stores and Column-Stores for Hybrid Workloads," in *Proceedings of the 2016 International Conference on Management of Data*, ser. SIGMOD '16, 2016, pp. 583–598. [Online]. Available: http://db.cs.cmu.edu/papers/2016/arulraj-sigmod2016.pdf

[5] Y. Zhang, Y. Xiao, Z. Wang, X. Ji, Y. Huang, and S. Wang, *ScaMMDB: Facing Challenge of Mass Data Processing with MMDB*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2009, pp. 1–12. [Online]. Available: http://dx.doi.org/10.1007/978-3-642-03996-6_1

[6] A. Lamb, M. Fuller, R. Varadarajan, N. Tran, B. Vandiver, L. Doshi, and C. Bear, "The Vertica Analytic Database: C-store 7 Years Later," *Proc. VLDB Endow.*, vol. 5, no. 12, pp. 1790–1801, Aug. 2012. [Online]. Available: http://dx.doi.org/10.14778/2367502.2367518

[7] M. Zukowski and P. Boncz, "From x100 to Vectorwise: Opportunities, Challenges and Things Most Researchers Do Not Think About," in *Proceedings of the 2012 ACM SIGMOD International Conference on Management of Data*, ser. SIGMOD '12. New York, NY, USA: ACM, 2012, pp. 861–862. [Online]. Available: http://doi.acm.org/10.1145/2213836.2213967

[8] D. Abadi, P. Boncz, and S. Harizopoulos, *The Design and Implementation of Modern Column-Oriented Database Systems*. Hanover, MA, USA: Now Publishers Inc., 2013.

[9] D. Abadi, S. Madden, and M. Ferreira, "Integrating Compression and Execution in Column-oriented Database Systems," in *Proceedings of the 2006 ACM SIGMOD International Conference on Management of Data*, ser. SIGMOD '06. New York, NY, USA: ACM, 2006, pp. 671–682. [Online]. Available: http://doi.acm.org/10.1145/1142473.1142548

[10] V. Raman, G. Attaluri, R. Barber, N. Chainani, D. Kalmuk, V. KulandaiSamy, J. Leenstra, S. Lightstone, S. Liu, G. M. Lohman, T. Malkemus, R. Mueller, I. Pandis, B. Schiefer, D. Sharpe, R. Sidle, A. Storm, and L. Zhang, "DB2 with BLU Acceleration: So Much More Than Just a Column Store," *Proc. VLDB Endow.*, vol. 6, no. 11, pp. 1080–1091, Aug. 2013. [Online]. Available: http://dx.doi.org/10.14778/2536222.2536233

[11] D. J. Abadi, D. S. Myers, D. J. DeWitt, and S. Madden, "Materialization Strategies in a Column-Oriented DBMS," in *Proceedings of the 23rd International Conference on Data Engineering, ICDE 2007, The Marmara Hotel, Istanbul, Turkey, April 15-20, 2007*, 2007, pp. 466–475. [Online]. Available: http://dx.doi.org/10.1109/ICDE.2007.367892

[12] L. Shrinivas, S. Bodagala, R. Varadarajan, A. Cary, V. Bharathan, and C. Bear, "Materialization strategies in the Vertica analytic database: Lessons learned," in *2013 IEEE 29th International Conference on Data Engineering (ICDE)*, April 2013, pp. 1196–1207.

[13] M. G. Ivanova, M. L. Kersten, N. J. Nes, and R. A. Gonçalves, "An Architecture for Recycling Intermediates in a Column-store," in *Proceedings of the 2009 ACM SIGMOD International Conference on Management of Data*, ser. SIGMOD '09. New York, NY, USA: ACM, 2009, pp. 309–320. [Online]. Available: http://doi.acm.org/10.1145/1559845.1559879

[14] Y. Liu, F. Cao, M. Mortazavi, M. Chen, N. Yan, C. Ku, A. Adnaik, S. Morgan, G. Shi, Y. Wang, and F. Fang, *DCODE: A Distributed Column-Oriented Database Engine for Big Data Analytics*. Cham: Springer International Publishing, 2015, pp. 289–299. [Online]. Available: http://dx.doi.org/10.1007/978-3-319-24315-3_30

[15] G. Graefe, "Query Evaluation Techniques for Large Databases," *ACM Comput. Surv.*, vol. 25, no. 2, pp. 73–169, Jun. 1993. [Online]. Available: http://doi.acm.org/10.1145/152610.152611

[16] M. T. Ozsu, *Principles of Distributed Database Systems*, 3rd ed. Upper Saddle River, NJ, USA: Prentice Hall Press, 2007.

[17] G. Chernishev, *New Trends in Databases and Information Systems: ADBIS 2015 Short Papers and Workshops, BigDap, DCSA, GID, MEBIS, OAIS, SW4CH, WISARD, Poitiers, France, September 8-11, 2015. Proceedings*. Cham: Springer International Publishing, 2015, ch. Towards Self-management in a Distributed Column-Store System, pp. 97–107. [Online]. Available: http://dx.doi.org/10.1007/978-3-319-23201-0_12

[18] ——, "The design of an adaptive column-store system," *Journal of Big Data*, vol. 4, no. 1, p. 21, 2017. [Online]. Available: http://dx.doi.org/10.1186/s40537-017-0069-4

[19] G. Chernishev, V. Grigorev, V. Galaktionov, E. Klyuchikov, and K. Smirnov, *Perspectives of System Informatics: 11th International Ershov Informatics Conference, PSI 2017*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2017, ch. PosDB: a Distributed Column-Store Engine (paper accepted).

[20] "P. E. ONeil, E. J. ONeil and X. Chen. The Star Schema Benchmark (SSB)." http://www.cs.umb.edu/~poneil/StarSchemaB.PDF, 2009, acessed: 20/07/2012.

[21] "A migration Yandex ClickHouse. A transcript of a talk at Highload++ 2016, http://www.highload.ru/2016/abstracts/2297.html," https://habrahabr.ru/post/322620/, 2017, acessed: 30/04/2017.

[22] "A Comparison of In-memory Databases." http://www.exasol.com/site/assets/files/3147/a_comparison_of_in-memory_databases.pdf, 2017, acessed: 30/04/2017.

[23] D. Kossmann, "The State of the Art in Distributed Query Processing," *ACM Comput. Surv.*, vol. 32, no. 4, pp. 422–469, Dec. 2000. [Online]. Available: http://doi.acm.org/10.1145/371578.371598

[24] Z. Li and K. A. Ross, "Fast Joins Using Join Indices," *The VLDB Journal*, vol. 8, no. 1, pp. 1–24, Apr. 1999. [Online]. Available: http://dx.doi.org/10.1007/s007780050071

[25] D. Tsirogiannis, S. Harizopoulos, M. A. Shah, J. L. Wiener, and G. Graefe, "Query Processing Techniques for Solid State Drives," in *Proceedings of the 2009 ACM SIGMOD International Conference on Management of Data*, ser. SIGMOD '09. New York, NY, USA: ACM, 2009, pp. 59–72. [Online]. Available: http://doi.acm.org/10.1145/1559845.1559854

[26] D. Taniar, C. H. C. Leung, W. Rahayu, and S. Goel, *High-Performance Parallel Database Processing and Grid Databases*, A. Zomaya, Ed. Wiley Series on Parallel and Distributed Computing, 2008.

A Survey of Database Dependency Concepts

Nikita Bobrov
Saint Petersburg State University
Email: nv.mm61@gmail.com

Anastasia Birillo
Saint Petersburg State University
Email: anastasia.i.birillo@gmail.com

George Chernishev
Saint Petersburg State University,
JetBrains Research
Email: g.chernyshev@spbu.ru

***Abstract*—A database dependency is a formal concept which is used to describe patterns in data. These patterns are employed during data analysis, data cleansing, and schema normalization. There is about dozen of major dependency types.**

In this paper we survey these types of database dependencies employed in the relational databases. We start from the earliest ones — functional dependencies and conclude with the state-of-the-art findings. For each type we provide both formal and non-formal definitions and present an example and counterexample. We also briefly discuss extraction algorithms and possible use cases.

I. Introduction

The era of big data not only presents new opportunities, but also poses new challenges for both industry and academy. The challenges come from the three "Vs", which are frequently used to describe the properties of big data: volume, velocity, and variety. Volume refers to the ever-increasing amounts of data that need to be processed. Velocity usually implies rapid arrival speeds and variety reflects that different types of data involved.

To benefit from this data, one has to be able to process, store, and query it efficiently. This requires that data semantics — patterns, properties, and purposes must be known to the user. Unfortunately, most of the time this is not the case for big data: it is hard to obtain this knowledge because of these three properties, especially given the large volumes.

There is a strong demand for methods, tools, and approaches for knowledge extraction. One of such tools is database dependencies, a concept known since the early 70s. This approach is very promising because big data is closely related to the NoSQL movement, which relies on very wide "tables". Thus, knowledge extraction employing these dependencies becomes very relevant in this environment.

A database dependency is a formal concept that can be used to describe patterns in data. Initially, the dependencies were employed for schema normalization and data cleansing. Currently, one of the most popular contemporary applications is data analysis.

A database dependency can be described as a rule which has left and right hand sides (LHS and RHS). This rule guarantees some properties of the data and involves LHS and RHS. For example, functional dependency guarantees that for any pair of tuples with equal LHS their RHS would be equal also.

Database dependencies are quite an old concept. The first kind — functional dependencies were introduced in 1971. Since then, the area grew rapidly, and many novel types and sub-types were proposed. Currently, there is a dozen of major dependency types.

In this paper we survey database dependency concepts employed in the relational databases. We survey them starting from the earliest ones and conclude with the state-of-the-art findings. We provide both formal and non-formal definitions and present an example for each type. We also briefly discuss extraction algorithms and possible use cases. Our goal is to produce a broad survey that involves major types of database dependencies without delving into details.

II. Motivation

The idea of this paper arrived during our development of a data-driven tool for automatic database physical design. Our approach was to rely on data properties instead of workload knowledge. During this study an extensive exploration of dependency concept literature was performed. We have discovered many classes of such concepts, not only textbook examples like functional dependencies. However, there were no single study which would present a broad overview of this field.

Thus, we decided to convert our expertise into a survey which would cover the major classes of dependencies, including the recent ones. This survey may be useful to beginners and to researchers who are interested in discovery of knowledge hidden in the data and who do not require solid theoretical understanding.

III. Related Work

There are several surveys on database dependency concepts. However, they are not entirely comparable to this study due to a number of reasons. First, there is a couple of studies that survey different kinds of dependencies, but which are more than 20 years old [1]. This fact renders them unusable for learning up-to-date dependency types.

On the other hand, the recent studies pursue goals that differ from the ones of our survey.

Reference [2] is a classification of types of relaxed functional dependencies. In this survey, the authors consider 35 types of imprecise functional dependencies, build a classification and provide a list of application domains for each class. However, the authors deliberately left inclusion dependencies and multivalued dependencies out of scope of this work.

There is another type of surveys that deal with dependency discovery methods. Reference [3] contains a Related Work

Patient	Gender	Doctor	Hospital	Area	Phone
Benson	M	Morgan	St. Mark's	North	89084140683
Ellis	F	Robin	St. George	South	89608401913
Graham	M	Robin	St. Mark's	North	89084140683
Harris	M	Jean	St. Mungo's	West	89607968712
Joy	F	Smith	St. Thomas'	East	89685290066

TABLE I
FD EXAMPLE

section that lists a number of methods for discovery of functional dependencies. Another functional dependency discovery survey is presented in reference [4]. This work features not only a survey, but also an experimental evaluation.

A survey of discovery methods for different types of dependencies is presented in reference [5]. This survey includes methods for conditional, inclusion, approximate, and XML dependency discovery.

Thus, to the best of our knowledge, there is no broad survey which lists known types of database dependencies.

IV. Dependency Types

A. Functional Dependencies

The notion of the functional dependency (FD) was originally introduced by E.F. Codd in the early 1970s in his paper "Further Normalization of the Data Base Relational Model" [6]. In this paper, Codd proposed to use FDs for database schema design. Now, the range of FD application is much wider. For example, study [7] describes query optimization methods that are based on the FD notion.

Study [8] contains research results on the topic of FDs that are used for query optimization. Besides, the usefulness of FDs is shown in relational database systems as well as in non-relational database environments.

The first formal definition of an FD was given in W.W. Armstrong's work [9].

Definition 1: A relation R satisfies the FD $X \to Y$ (where $X, Y \subseteq R$), if and only if for all $t_1, t_2 \in R$ holds: if $t_1[X] = t_2[X]$, then $t_1[Y] = t_2[Y]$.

Thus, an FD is essentially a "many to one" relation between the value sets of attributes participating in LHS and RHS.

Consider relation depicted in Table I. The following FDs are present:

- {Area, Phone} → {Hospital}
- {Patient} → {Gender}

As we can see from Table I, any value from the set {Patient, Doctor} is related exactly to one value from the set {Area}. The second FD can be explained in a similar manner.

Further, we can list dependencies which do not hold (violate the FD condition) in the relation:

- {Doctor} → {Hospital, Area}
- {Hospital} → {Patient}

The first one is not an FD since two different values in the RHS attribute set (Hospital and Area) present for a fixed value of the Doctor attribute. For example doctor Robin maps simultaneously to $\{St.George, South\}$, $\{St.Mark's, North\}$ that contradicts the meaning of FD. The issue with the second dependency can be explained in a similar way.

Nowadays, relaxed FDs (RFDs) [2] are one of the most active sub-types of FDs. These dependencies relax one or more constraints of the canonical FD. For example, RFD has to hold for most tuples, not for all of them like in FD case.

B. Inclusion Dependencies

Inclusion dependency (IND) was first formalized by R. Fagin [10], but have also been used by J.M. Smith and D.C.P. Smith in [11]. Since then INDs have become just as popular as any other traditional dependency type (FD, MVD, JD).

By introducing this concept, a new kind of a normal form was defined — the domain-key normal form (DK/NF). This form requires every constraint on the relation to be a logical consequence of key constraints and domain constraints. A relation is in DK/NF if it is guaranteed that no insertion and deletion anomalies are present, as it is stated in Fagin's theorem 3.13 ("a satisfiable 1NF relation schema is in DK/NF if and only if it has no insertion or deletion anomalies" [10]). We say that an insertion anomaly occurs if after insertion of a tuple one of relation constraints (either key or domain) is violated. Similarly, a deletion anomaly occurs when deletion of a single tuple results in constraint violation. According to paper [12], we define IND as:

Definition 2: Let $R = (R_1 \dots R_k)$ and $S = (S_1 \dots S_m)$ be two relational tables. Further, $\hat{R} = R_{i_1} \dots R_{i_n}$ and $\hat{S} = S_{i_1} \dots S_{i_n}$ be n-ary column combinations of distinct columns. We say that IND $\hat{R} \subseteq \hat{S}$ holds, if for every tuple $t_R \in R$, there is a tuple $t_S \in S$, such that $t_R[\hat{R}] = t_S[\hat{S}]$. $\hat{R}$ is called the dependent column combination and $\hat{S}$ – the referenced column combination.

In other words, we can say that all tuples of the attribute combination $\hat{R}$ in the relation R must be also contained in tuples of the attribute combination $\hat{S}$ of the relation S.

Figure IV-B depicts an example of IND {DLN} ⊆ {DLID}. This dependency information can help us in discovery of a foreign key (which is DLN in the example).

UID	Name	Gender	DLN
1	Sofia	F	21
2	Leonard	M	35
3	Shavkat	M	10
4	Mary	F	65
5	Andrew	M	10

DLID	Country
21	Romania
35	Spain
10	Germany
65	USA

TABLE II
IND EXAMPLE

Let us construct an example where the dependency presented in Table IV-B is violated. In order to violate {DLN} ⊆ {DLID}, the DLN should stop being the foreign key for the corresponding table. Thus, we have to substitute any value of the DLN with the value absent in the $DLID$. For example, changing the third tuple to $(3, Shavkat, M, 11)$ is enough to break this dependency.

Name	Disease	Doctor
Wilson	Arthritis	Lewis
Taylor	Insomnia	Jackson
Kimberly	Pancreatitis	Brooks
Ellis	Schizophrenia	Brooks

TABLE III
JD EXAMPLE

Name	Doctor
Wilson	Lewis
Taylor	Jackson
Kimberly	Brooks
Ellis	Brooks

Name	Disease
Wilson	Arthritis
Taylor	Insomnia
Kimberly	Pancreatitis
Ellis	Somnambulism

TABLE IV
JD EXAMPLE

Name	Doctor
Wilson	Lewis
Taylor	Jackson
Kimberly	Brooks
Ellis	Brooks

Doctor	Disease
Lewis	Arthritis
Jackson	Insomnia
Brooks	Pancreatitis
Brooks	Somnambulism

TABLE V
JD COUNTEREXAMPLE

IND is one of the most important dependencies for many tasks such as data integration, integrity checking, schema design, and any other where we may need additional metadata for understanding significant aspects or structure of an unknown database. Consider a data source that comes only with superficial information, such as relation or attributes names with no interrelational constraints. In this case, detected INDs between relations may be considered as a precondition for a foreign key constraint. In fact, being a basis of the interrelational constraint is the most prominent application of the IND.

There are further state of the art approaches for exact and approximate inference of INDs: [13], [14], [12].

C. Join Dependencies

Join dependencies (JDs) were first mentioned at the end of the 1970s in the works [15], [16]. The notion of JDs ensures lossless reconstruction of the data that was decomposed into a number of relations. The data is reconstructed using the join relational operation.

Paper [17] contains the formal definition of JD:

Definition 3: A JD defined for a relation $R[U]$ (U — is a set of attributes) is an expression of the form $\bowtie [X_1, \cdots, X_n]$, where $X_1 \cup \cdots \cup X_n = U$. An instance I of $R[U]$ satisfies $\bowtie [X_1, \cdots, X_n]$, if $I = \pi_{X_1}(I) \bowtie \cdots \bowtie \pi_{X_n}(I)$.

Consider the relation presented in Table III. Here, the following FD exists: $\{\text{Name}\} \rightarrow \{\text{Disease}\}$. This FD implies that the JD $\bowtie [\{\text{Name}, \text{Doctor}\}, \{\text{Name}, \text{Disease}\}]$ is satisfied, thus lossless decomposition is available (see Table IV).

Let us study the violation of JD constraint in the same table III. The dependency $\bowtie [\{\text{Name}, \text{Doctor}\}, \{\text{Doctor}, \text{Disease}\}]$ is not a valid JD because there is no opportunity to restore the original table. In this case the join of these two tables would lead to appearance of two phantom records, absent in the original table. The key "Brooks" would produce four items instead of two. This is explained by the fact that the dependency $\{\text{Hospital}\} \rightarrow \{\text{Doctor}\}$ is not an FD. Also, there may be the opposite case when the records would be lost.

Currently, several algorithms are being actively developed for efficient testing of existing JDs. For example, the algorithm [18] is aimed for an I/O-efficient testing in the external memory model.

Paper [19] contains the finite axiomatization of the implication problem for inclusion and conditional independence atoms (dependencies) in the dependence logic context. The general axiomatization approach is described, which extend to such types of dependencies as inclusion and join dependencies.

FlightID	Aircraft	Date *(DD/MM/YY)*	Price
1	A320	01.01.17	300
2	A320	04.01.17	340
3	Boeing 747	13.03.17	210
4	A380	15.03.17	410
5	Boeing 747	17.03.17	270
6	A320	23.07.17	450

TABLE VI
DD EXAMPLE

D. Differential Dependencies

Differential dependencies (DD) are the newest concept we review in this paper. They were first presented in 2011 [20] and are defined as follows:

Definition 4: Let $\phi_{LHS}[X]$, $\phi_{RHS}[Y]$ be differential functions specifying constraints on distances over attributes X and Y of relation R. A differential dependency $\phi_{LHS}[X] \rightarrow \phi_{RHS}[Y]$ holds on relation R if for any pair of tuples for which the difference on attributes X satisfies the constraints specified by $\phi_{LHS}[X]$, the difference on Y also satisfies the constraints specified by $\phi_{RHS}[Y]$.

Speaking less formally, we say that if a DD $\phi_{LHS}[X] \rightarrow \phi_{RHS}[Y]$ holds, then for any two tuples on attribute combination X whose distance belongs to the range specified by ϕ_{LHS}, the distance of the same two tuples on Y is in the range specified by ϕ_{RHS}. Differential function on attributes may be specified by operators $\{=, <, >, \leq, \geq\}$. In this case, DD can be reduced to other dependency concepts in the following way: (i) if a differential function for LHS represents a similarity constraint $(= 0)$, then DD becomes matching dependency (MD concept, [21]), (ii) if all differential constraints are $(= 0)$, then DD subsumes FD [20].

Consider an example presented in Figure VI — a table containing flight information. The following DD $[Aircraft(= 0) \wedge Date(\leq 4)] \rightarrow [Price(\geq 40, \leq 60)]$ holds. It means that for the same type of aircraft ($Aircraft(= 0)$) the price difference for flights in any range of four days ($Date(\leq 4)$) must be ≥ 40 and ≤ 60.

If we take away the date from the dependency, the dependency will be violated: $[Aircraft(= 0)] \rightarrow [Price(\geq 40, \leq 60)]$. Let us consider the $A320$ value. There is the following set of $Price$ attribute values corresponding to

$A320$: $\{300, 340, 450\}$. Here, the difference between prices for $FlightID = 1$ and $FlightID = 6$ is outside of the range specified by the dependency. This illustrates the dependency violation.

Since DDs represent a special kind of FD and MD, their application domains overlap. DD has a rather broad application: the data quality problem, integrity constraints checking, and query optimization. Furthermore, the idea of a differential key (a new type of constraint that is based on deduced rules related to attribute distance) provides insight into the data partitioning task.

DD is a more general concept than FD, and its definition is based on distances between attribute values, so FD inference algorithms are not fully capable of DD discovery — they find special cases only. Nevertheless, in the original work [20] DD row- and column-based inference approaches are described.

E. Multivalued Dependencies

Multivalued dependencies (MVD) were introduced independently by R. Fagin and C.A. Zaniolo in 1976 [22], [23]. The modern MVD definition is as follows [24]:

Definition 5: A relation R satisfies the MVD $X \twoheadrightarrow Y$ (where $X, Y \subseteq R$, $X \cap Y = \emptyset$), if and only if for all $t_1,\ t_2 \in R$ holds: if $t_1[XY] = t_2[XY]$ then there is $t \in R$ such that $t[XY] = t_1[XY]$ and $t[X(R - XY)] = t_2[X(R - XY)]$.

That means if we have two tuples of R that agree on X, then their values for Y may be swapped, and the result will be two tuples that are also in the relation R.

In subsequent studies, the MVD definition was slightly modified: LHS and RHS of dependency are no longer necessarily disjoint [25]. A number of MVD inference rules were presented as well.

Traditionally, MVDs have been considered as the necessary and sufficient condition of a relation to be decomposed into two of its projections without loss of information [22]. This means that an MVD $X \twoheadrightarrow Y$ holds if and only if $R = R[XY] \bowtie R[X(R - XY)]$ [24]. Such information on relation decomposability may be used in a schema (re-)design task as it reveals internal structure of a data source.

Lately, MVDs were generalized by bringing in a new concept of full hierarchical dependency, see Section IV-F. Another remarkable fact is the relationship between FDs and MVDs: we may say for $X \cap Y = \emptyset$, that if an FD $X \to Y$ holds in a relation, then an MVD $X \twoheadrightarrow Y$ also holds. Thus, each FD is also an MVD, but not vice versa [25].

Let us consider a relation presented in Figure VII. Attribute $CourseID$ determines a set of values $StudentName$ and $RecommendedBook$. The latter does not depend on attribute $StudentName$ (that means there is no connectivity between these two attributes), and we may say that the MVDs {CourseID} $\twoheadrightarrow$ {StudentName} and {CourseID} $\twoheadrightarrow$ {RecommendedBook} hold in relation.

To show the violation of the dependence, it is sufficient to replace the value of one field in the table VII. The result of the changes is shown in the table VIII, from which we can see that changing the value of attribute RecommendedBook in the third row of the table violates both MVDs. In order to keep the dependencies there should be Course_book_math for Lena and Course_book_optic for Michael.

CourseID	StudentName	RecommendedBook
10	Michael	Course_book_math
10	Michael	Course_book_mechanic
10	Lena	Course_book_math
10	Lena	Course_book_mechanic
106	Michael	Course_book_cs
106	Michael	Course_book_mlearn
9	John	Course_book_reading
9	John	Course_book_grammar

TABLE VII
MVD EXAMPLE

CourseID	StudentName	RecommendedBook
10	Michael	Course_book_math
10	Michael	Course_book_mechanic
10	Lena	**Course_book_optic**
10	Lena	Course_book_mechanic
106	Michael	Course_book_cs
106	Michael	Course_book_mlearn
9	John	Course_book_reading
9	John	Course_book_grammar

TABLE VIII
MVD COUNTEREXAMPLE

For complete understanding of MVDs and their place among other dependency concepts, the following papers are recommended [26], [24].

F. Full Hierarchical Dependencies

As it was already mentioned, a full hierarchical dependency (FHD) is an attempt to generalize the MVD concept. The first notion and formalization of FHD was presented in [27]. Nowadays, the following definition is used [24]:

Definition 6: Let $X \subseteq R$ and S is a non-empty set of pairwise disjoint subsets of R that are also disjoint from X. $S \neq \emptyset$, for all $Y \in S$ we have $Y \subseteq S$ and for all $Y,\ Z \in S \cup \{X\}$ we have $Y \cap Z = \emptyset$. Relation R satisfies FHD $X : Y_1, \ldots, Y_k$, if and only if for all $t_1, \ldots, t_{k+1} \in R$ the following condition is satisfied: if $t_i[X] = t_j[X]$ for all $1 \leq i,\ j \leq k + 1$ then there is some $t \in R$ such that $t[XY_i] = t_i[XY_i]$ for all $i = 1, \ldots, k$ and $t[X(R - XY_1 \ldots Y_k)] = t_{k+1}[X(R - XY_1 \ldots Y_k)]$.

As we can see, in case of $k = 1$ this is the definition of MVD. Moreover, if for each $k = 1, \ldots, n$ MVD $X \twoheadrightarrow Y_k$ holds over R, then R satisfies the FHD $X : \{Y_1, \ldots, Y_n\}$. Therefore, the area of FHD application is the same as the one of MVDs — schema design in terms of relation decomposition. The result of such decomposition is called generalized hierarchical decomposition (GHD), and it may be represented as a tree structure [27].

Since we show in the MVD section that {CourseID} $\twoheadrightarrow$ {StudentName} and {CourseID} $\twoheadrightarrow$ {RecommendedBook} hold, then the FHD {CourseID} : {StudentName, RecommendedBook} also holds.

Category	Weight	Cost	Distance
Portable	$0-5$ kg	150$	$0-10$ km
Portable	$0-5$ kg	200$	$11-20$ km
Portable	$0-5$ kg	250$	$21-30$ km
Midsize	$6-10$ kg	250$	$0-10$ km
Midsize	$6-10$ kg	350$	$11-20$ km
Midsize	$6-10$ kg	450$	$21-30$ km
Largesize	$11-15$ kg	350$	$0-10$ km
Largesize	$11-15$ kg	500$	$11-20$ km
Largesize	$11-15$ kg	650$	$21-30$ km

TABLE IX
OD EXAMPLE

Due to the aforementioned relationships between concepts, FHD mainly appears in studies on combining dependency classes (e.g. [24], where non-trivial FD and FHD connectivity is studied).

G. Order Dependencies

The term "order dependencies" (OD) first appeared in the study [28]. An OD allows to describe the value of some attribute using the information about the order relation on the given value set. For example, we need to transfer an item from point A to point B. OD gives us an opportunity to estimate that in this case delivery by a heavy vehicle will cost at least as much as the delivery by a car as both have to travel the same distance (we ignore real-life aspects of shipping).

The formal definition ODs is as follows [29]:

Definition 7: Call $X \mapsto Y$ an order dependency (where $X, Y \subseteq R$) over the relation R if, for every pair of admissible tuples $t_1, t_2 \in R$, $t_1[X] \preceq t_2[X]$ implies $t_1[Y] \preceq t_2[Y]$.

The analysis of these ODs can be used for improving the efficiency of database query processing. However, arrangement of sets (on which the model of OD is based) is a challenging task. Currently, development of algorithms for fast sets arrangement is a very active area of research. For example, in study [29] an efficient algorithm for lexicographical set arrangement has been introduced. Also, a large number of inference rules that are used for query transformation during the optimization process is presented.

For example, Table IX shows the following OD: {Category, Weight, Distance} $\mapsto$ {Cost}. So, according to the commodity (that is characterized by the weight of the shipment) we can specify whether the delivery costs more or less for the shipping to the same distances. As we have already mentioned, we suppose that portable shipping will cost less than the large size shipping on the same distances.

Consider the dependence {Distance} $\mapsto$ {Cost}. It is not a valid OD since the order relation for the shipment price for different distances will be violated. For example, shipment of $Largesize$ for 0–10 km is cheaper than shipment of $Midsize$ for 21–30 km.

H. Conditional Functional Dependencies

One of the newest FD types is the Conditional Functional Dependencies (CFDs). The first mention of CFD appears in the study by Wenfei Fan et al. [30]. CFDs aim at capturing the consistency of data by enforcing bindings of semantically related values.

SC	NB	City	Client	Rate
110	Bank1	Moscow	Wood	10%
110	Bank2	Bremen	Martin	7%
220	Bank1	Stockholm	King	6%
110	Bank2	St. Petersburg	King	7%
220	Bank1	Hamburg	Turner	6%

TABLE X
CFD EXAMPLE, SOURCE DATA

SC	NB	Rate
220	Bank1	—
110	Bank2	—

TABLE XI
CFD EXAMPLE, PATTERN TABLEU

We frequently encounter CFDs in our everyday life. For example, the position of a worker determines his salary only in some departments but not in the whole organization. CFDs are extensively used in the data integration domain. It is justified by the fact that the patterns existing in the individual data sources would also present in the combined data set.

CFDs extend the FDs using the pattern table that provides the bound of semantically related values. It is important to note that it is necessary to apply CFDs only to tuples that meet the values from the pattern table, but not to all tuples.

The formal definition CFDs is as follows [31]:

Definition 8: A conditional functional dependency (CFD) ϕ on S is a pair $(X \to Y, T_p)$, where $X \to Y$ is a standard FD, referred to as the embedded FD; and T_p is a "pattern tableau" that defines over which rows of the table the embedded FD applies. Each entry $t_p \in T_p$ specifies a pattern over $X \cup Y$, so for each attribute in $A \in X \cup Y$, either $t_p[A] = \alpha$, where α is a value in the domain of A, or else $t_p[A] = \_\_$, for the special wild-card symbol __. A row r_i satisfies an entry t_p of tableau T_p for attributes A, denoted by $r_i[A] \asymp t_p[A]$, if either $r_i[A] = t_p[A]$, or else $t_p[A] = \_\_$. The CFD ϕ holds if

$$\forall i, j, p. r_i[X] = r_j[X] \asymp t_p[X] \Rightarrow r_i[Y] = r_j[Y] \asymp t_p[Y].$$

Consider the Table X that contains client base of various banks. We introduce the following abbreviation: subdivision code – SC, name of the bank – NB. Table X illustrates the following CFD: {Subdivision code, Name of the bank} $\longrightarrow$ {Rate}. The Table XI shows that the banking sector values with an equal subdivision code will probably have an identical interest rate. Although it is important to understand that this condition holds for the majority of tuples in source data, but not for all. For example, Bank2 in St. Petersburg with subdivision code of 110 has above rate than Bank2 in Bremen, which has the same subdivision code.

In order to demonstrate the violation of CFD we can modify Table XI (pattern tableu) in the following way.substitute bank2 entry with the wild-card. In this case values corresponding to $SC = 110$ are not equal: $7, 7, 10$. Thus, the dependency does not hold anymore.

Dependency Concept	Notation	Use-case	Inference Rules and Algorithms
FD	$X \rightarrow Y$	Data cleansing, Query optimization [8]	TANE [32], FDEP [33], DFD [34], FDmine [35]
IND	$X \subseteq Y$	Data integration, Schema design, Integrity checking [36]	Binder [13], Faida [12], Mind [37]
JD	$\bowtie [X_1, \ldots, X_n]$	Schema design [15]	Hu et.al [18]
DD	$\phi_{LHS}[X] \rightarrow \phi_{RHS}[Y]$	Data partition, Query Optimization, Integrity Constraints [20]	Song and Chen [20], Liu et.al [38]
MVD	$X \twoheadrightarrow Y$	Data partition, Schema design [26]	Savnik and Flach [26]
FHD	$X : \{Y_1, \ldots, Y_n\}$	Data partition, Schema design [27]	Biskup and Link [24]
OD	$X \mapsto Y$	Query optimization [28]	Szlichta et.al [29]
CFD	$X \rightarrow Y, T_p$	Consistency checking, Data cleansing [30]	Li et.al [39]

TABLE XII
CUMULATIVE TABLE

An important problem is efficient estimation of CFD confidence using a small number of data passes and a small amount of space. The solution to this problem is described in [31]. An improvement of the algorithm for the minimum CFD set construction has been proposed in the work [39]. The described algorithm works in linear time and is one of the most efficient contemporary algorithms.

V. CONCLUSION

In this paper we surveyed several database dependency concepts. For each type we provided both formal and non-formal definitions and presented an example. We also briefly discussed extraction algorithms and possible use cases. A short summary of considered dependency types is presented in Table XII.

REFERENCES

[1] R. Fagin and M. Y. Vardi, "The theory of data dependencies — an overview," in *Proceedings of the 11th Colloquium on Automata, Languages and Programming*. London, UK, UK: Springer-Verlag, 1984, pp. 1–22. [Online]. Available: http://dl.acm.org/citation.cfm?id=646238.683349

[2] L. Caruccio, V. Deufemia, and G. Polese, "Relaxed functional dependencies — a survey of approaches," *IEEE Transactions on Knowledge and Data Engineering*, vol. 28, no. 1, pp. 147–165, Jan 2016.

[3] ——, "On the discovery of relaxed functional dependencies," in *Proceedings of the 20th International Database Engineering & Applications Symposium*, ser. IDEAS '16. New York, NY, USA: ACM, 2016, pp. 53–61. [Online]. Available: http://doi.acm.org/10.1145/2938503.2938519

[4] T. Papenbrock, J. Ehrlich, J. Marten, T. Neubert, J.-P. Rudolph, M. Schönberg, J. Zwiener, and F. Naumann, "Functional dependency discovery: An experimental evaluation of seven algorithms," *Proceedings of the VLDB Endowment*, vol. 8, no. 10, pp. 1082–1093, June 2015. [Online]. Available: http://dx.doi.org/10.14778/2794367.2794377

[5] J. Liu, J. Li, C. Liu, and Y. Chen, "Discover dependencies from data—a review," *IEEE Trans. on Knowl. and Data Eng.*, vol. 24, no. 2, pp. 251–264, Feb. 2012. [Online]. Available: http://dx.doi.org/10.1109/TKDE.2010.197

[6] E. F. Codd, "Further normalization of the data base relational model," *Data base systems*, pp. 33–64, 1972.

[7] H. Darwen and C. Date, "The role of functional dependencies in query decomposition," *Relational Database Writings*, vol. 1991, pp. 133–154, 1989.

[8] G. N. Paulley, "Exploiting functional dependence in query optimization," Ph.D. dissertation, Citeseer, 2001.

[9] W. W. Armstrong, "Dependency structures of data base relationships." in *IFIP congress*, vol. 74. Geneva, Switzerland, 1974, pp. 580–583.

[10] R. Fagin, "A normal form for relational databases that is based on domains and keys," *ACM Trans. Database Syst.*, vol. 6, no. 3, pp. 387–415, Sep. 1981. [Online]. Available: http://doi.acm.org/10.1145/319587.319592

[11] J. M. Smith and D. C. P. Smith, "Database abstractions: Aggregation," *Commun. ACM*, vol. 20, no. 6, pp. 405–413, Jun. 1977. [Online]. Available: http://doi.acm.org/10.1145/359605.359620

[12] K. Sebastian, P. Thorsten, D. Christian, F. Moritz, H. Manuel, Z. Martin, Z. Christian, and N. Felix, "Fast approximate discovery of inclusion dependencies," in *Proceedings of the conference on Database Systems for Business, Technology, and Web (BTW)*, 0 2017.

[13] T. Papenbrock, S. Kruse, J.-A. Quiané-Ruiz, and F. Naumann, "Divide conquer-based inclusion dependency discovery," *Proc. VLDB Endow.*, vol. 8, no. 7, pp. 774–785, Feb. 2015. [Online]. Available: http://dx.doi.org/10.14778/2752939.2752946

[14] A. Koeller and E. A. Rundensteiner, *Heuristic Strategies for the Discovery of Inclusion Dependencies and Other Patterns*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2006, pp. 185–210.

[15] A. V. Aho, C. Beeri, and J. D. Ullman, "The theory of joins in relational databases," *ACM Transactions on Database Systems (TODS)*, vol. 4, no. 3, pp. 297–314, 1979.

[16] J. Rissanen, "Relations with functional and join dependencies and their representation by independent components," *Unpublished manuscript, IBM Research Lab., San Jose, Calif*, 1978.

[17] ——, "Theory of relations for databasesa tutorial survey," in *International Symposium on Mathematical Foundations of Computer Science*. Springer, 1978, pp. 536–551.

[18] X. Hu, M. Qiao, and Y. Tao, "Join dependency testing, loomis-whitney join, and triangle enumeration," in *Proceedings of the 34th ACM SIGMOD-SIGACT-SIGAI Symposium on Principles of Database Systems*. ACM, 2015, pp. 291–301.

[19] M. Hannula and J. Kontinen, "A finite axiomatization of conditional independence and inclusion dependencies," *Information and Computation*, vol. 249, pp. 121–137, 2016.

[20] S. Song and L. Chen, "Differential dependencies: Reasoning and discovery," *ACM Trans. Database Syst.*, vol. 36, no. 3, pp. 16:1–16:41, Aug. 2011. [Online]. Available: http://doi.acm.org/10.1145/2000824.2000826

[21] W. Fan, "Dependencies revisited for improving data quality," in *Proceedings of the Twenty-seventh ACM SIGMOD-SIGACT-SIGART Symposium on Principles of Database Systems*, ser. PODS '08. New York, NY, USA: ACM, 2008, pp. 159–170. [Online]. Available: http://doi.acm.org/10.1145/1376916.1376940

[22] R. Fagin, "Multivalued dependencies and a new normal form for relational databases," *ACM Trans. Database Syst.*, vol. 2, no. 3, pp. 262–278, Sep. 1977. [Online]. Available: http://doi.acm.org/10.1145/320557.320571

[23] C. A. Zaniolo, "Analysis and design of relational schemata for database systems." Ph.D. dissertation, 1976, aAI7622226.

[24] J. Biskup and S. Link, "Appropriate inferences of data dependencies in relational databases," *Annals of Mathematics and Artificial Intelligence*,

vol. 63, no. 3-4, pp. 213–255, Dec. 2011. [Online]. Available: http://dx.doi.org/10.1007/s10472-012-9275-0

[25] C. Beeri, R. Fagin, and J. H. Howard, "A complete axiomatization for functional and multivalued dependencies in database relations," in *Proceedings of the 1977 ACM SIGMOD International Conference on Management of Data*, ser. SIGMOD '77. New York, NY, USA: ACM, 1977, pp. 47–61. [Online]. Available: http://doi.acm.org/10.1145/509404.509414

[26] I. Savnik and P. A. Flach, "Discovery of multivalued dependencies from relations," *Intell. Data Anal.*, vol. 4, no. 3,4, pp. 195–211, Sep. 2000. [Online]. Available: http://dl.acm.org/citation.cfm?id=1294171.1294173

[27] C. Delobel, "Normalization and hierarchical dependencies in the relational data model," *ACM Trans. Database Syst.*, vol. 3, no. 3, pp. 201–222, Sep. 1978. [Online]. Available: http://doi.acm.org/10.1145/320263.320271

[28] S. Ginsburg and R. Hull, "Order dependency in the relational model," *Theoretical computer science*, vol. 26, no. 1-2, pp. 149–195, 1983.

[29] J. Szlichta, P. Godfrey, and J. Gryz, "Fundamentals of order dependencies," *Proceedings of the VLDB Endowment*, vol. 5, no. 11, pp. 1220–1231, 2012.

[30] W. Fan, F. Geerts, X. Jia, and A. Kementsietsidis, "Conditional functional dependencies for capturing data inconsistencies," *ACM Transactions on Database Systems (TODS)*, vol. 33, no. 2, p. 6, 2008.

[31] G. Cormode, L. Golab, K. Flip, A. McGregor, D. Srivastava, and X. Zhang, "Estimating the confidence of conditional functional dependencies," in *Proceedings of the 2009 ACM SIGMOD International Conference on Management of data*. ACM, 2009, pp. 469–482.

[32] Y. Huhtala, J. Kärkkäinen, P. Porkka, and H. Toivonen, "Tane: An efficient algorithm for discovering functional and approximate dependencies," *The Computer Journal*, pp. 100–111, 1992.

[33] P. A. Flach and I. Savnik, "Database dependency discovery: A machine learning approach," *AI Commun.*, vol. 12, no. 3, pp. 139–160, Aug. 1999. [Online]. Available: http://dl.acm.org/citation.cfm?id=1216155.1216159

[34] Z. Abedjan, P. Schulze, and F. Naumann, "DFD: Efficient functional dependency discovery," in *Proceedings of the 23rd ACM International Conference on Conference on Information and Knowledge Management*, ser. CIKM '14. New York, NY, USA: ACM, 2014, pp. 949–958. [Online]. Available: http://doi.acm.org/10.1145/2661829.2661884

[35] H. Yao, H. J. Hamilton, and C. J. Butz, "Fd_mine: Discovering functional dependencies in a database using equivalences," in *Proceedings of the 2002 IEEE International Conference on Data Mining*, ser. ICDM '02. Washington, DC, USA: IEEE Computer Society, 2002, pp. 729–732. [Online]. Available: http://dl.acm.org/citation.cfm?id=844380.844800

[36] J. Bauckmann, U. Leser, and F. Naumann, *Efficient and exact computation of inclusion dependencies for data integration*, 2010.

[37] F. D. Marchi, S. Lopes, and J.-M. Petit, "Unary and n-ary inclusion dependency discovery in relational databases," *Journal of Intelligent Information Systems*, vol. 32, no. 1, pp. 53–73, 2009. [Online]. Available: http://dx.doi.org/10.1007/s10844-007-0048-x

[38] L. Jixue, K. Selasi, L. Jiuyong, Y. Feiyue, and W. V. Millist, "Discovery of approximate differential dependencies," *CoRR*, vol. abs/1309.3733, 2013. [Online]. Available: http://arxiv.org/abs/1309.3733

[39] J. Li, J. Liu, H. Toivonen, and J. Yong, "Effective pruning for the discovery of conditional functional dependencies," *The Computer Journal*, vol. 56, no. 3, pp. 378–392, 2013.

Обзор методов обнаружения аномалий в потоках данных

В.П. Шкодырев, К.И. Ягафаров, В.А. Баштовенко, Е.Э. Ильина

***Аннотация*— Данная статья посвящена исследованию различных подходов к идентификации аномалий во временных рядах, которая заключается в обнаружении и обработке отклонений в потоках данных, получаемых во время проведения технологических процессов. Выявление аномалий в поведении системы позволяет не только повысить качество таких процессов, но и предотвращать нештатные ситуации и аварии на ранних этапах. Все это указывает на актуальность проведения исследований в данной области.**

В работе приведен обзор существующих методов и алгоритмов обнаружения аномалий с целью структуризации имеющихся данных и последующего отбора средств для разработки системы идентификации аномалий в потоках больших данных.

***Ключевые слова* — Поиск аномалий, Анализ данных, Потоки данных**

I. Введение

Интеллектуальный анализ данных, называемый также Data mining, используется для выделения новой значимой информации из большого объема данных. В условиях постоянного увеличения этих объемов, а также возрастающей значимости результатов их анализа вопрос идентификации имеющихся в них аномалий стоит особенно остро. Результаты анализа без предварительного исключения аномальных экземпляров данных могут быть значительно искажены.

Обнаружение аномалий относится к поиску непредвиденных значений (паттернов) в потоках данных. Аномалия (выброс, ошибка, отклонение или исключение) – это отклонение поведения системы от стандартного (ожидаемого). В данной статье эти термины являются эквивалентными. Они могут возникать в данных самой различной природы и структуры в результате технических сбоев, аварий, преднамеренных взломов и т.д. В настоящее время разработано множество методов и алгоритмов поиска аномалий для различных типов данных. Целью данной статьи является обзор наиболее универсальных из них.

II. Виды аномалий

Аномалии в данных могут быть отнесены к одному из трех основных типов [3].

Точечные аномалии возникают в ситуации, когда отдельный экземпляр данных может рассматриваться как аномальный по отношению к остальным данным. На рисунке 1а экземпляр A1, а также группа экземпляров A2 являются аномальными при нормальных экземплярах в группах C1 и C2. Данный вид аномалий является наиболее легко распознаваемым, большинство существующих методов создано для распознавания точечных аномалий.

Контекстуальные аномалии наблюдаются, если экземпляр данных является аномальным лишь в определенном контексте, (данный вид аномалий также называется условным). Для определения аномалий этого типа основным является выделение контекстуальных и поведенческих атрибутов.

- Контекстуальные атрибуты используются для определения контекста (или окружения) для каждого экземпляра. Во временных рядах контекстуальным

Рис.1 Виды аномалий

атрибутом является время, которое определяет положение экземпляра в целой последовательности. Контекстуальным атрибутом также может быть положение в пространстве или более сложные комбинации свойств.

- Поведенческие атрибуты определяют не контекстуальные характеристики, относящиеся к конкретному экземпляру данных.

Аномальное поведение определяется посредством значений поведенческих атрибутов исходя из конкретного контекста. Таким образом, экземпляр данных может быть контекстуальной аномалией при данных условиях, но при таких же поведенческих атрибутах считаться нормальным в другом контексте. Так, на рисунке 1б в точке A наблюдается аномалия, в отличие от точек N1 – N5, имеющих аналогичное значение. При обнаружении контекстуальных аномалий это свойство является ключевым в разделении контекстуальных и поведенческих атрибутов.

Коллективные аномалии возникают, когда последовательность связанных экземпляров данных (например, участок временного ряда) является аномальной по отношению к целому набору данных. Отдельный экземпляр данных в такой последовательности может не являться отклонением, однако совместное появление таких экземпляров является коллективной аномалией. На рисунке 1в участок A является коллективной аномалией.

Кроме того, в то время как точечные или контекстуальные аномалии могут наблюдаться в любом наборе данных, коллективные наблюдаются только в тех, где данные связаны между собой.

Стоит так же отметить, что точечные или коллективные аномалии могут в то же время являться и контекстуальными.

III. Методы обнаружения точечных аномалий

Существует несколько вариантов классификации существующих методик поиска аномалий [3]. В данной работе буду рассмотрены два вида деления: по режиму распознавания и по способу реализации.

В зависимости от применяемого алгоритма результатом работы системы идентификации аномалий может быть либо метка экземпляра данных как аномального, либо оценка степени вероятности того, что экземпляр является аномальным.

Процесс выявления аномалий может проводиться для данных различного формата:

- поток данных (работа в реальном времени);
- архив данных.

A. *Режимы распознавания аномалий*

Часто для решения задачи поиска аномалий требуется набор данных, описывающих систему. Каждый экземпляр в нем описывается меткой, указывающей, является ли он нормальным или аномальным. Таким образом, множество экземпляров с одинаковой меткой формируют соответствующий класс.

Создание подобной промаркированной выборки обычно проводится вручную и является трудоемким и дорогостоящим процессом. В некоторых случаях получить экземпляры аномального класса невозможно в силу отсутствия данных о возможных отклонениях в системе, в других могут отсутствовать метки обоих классов. В зависимости от того, какие классы данных используются для реализации алгоритма, методы поиска аномалий могут выполняться в одном из трех перечисленных ниже режимов:

1) Supervised anomaly detection (режим распознавания с учителем)

Данная методика требует наличия обучающей выборки, полноценно представляющей систему и включающей экземпляры данных нормального и аномального классов. Работа алгоритма происходит в два этапа: обучение и распознавание. На первом этапе строится модель, с которой в последствие сравниваются экземпляры, не имеющие метки. В большинстве случаев предполагается, что данные не меняют свои статистические характеристики, иначе возникает необходимость изменять классификатор [11].

Основной сложностью алгоритмов, работающих в режиме распознавания с учителем, является формирование данных для обучения. Часто аномальный класс представлен значительно меньшим числом экземпляров, чем нормальный, что может приводить к неточностям в полученной модели. В таких случаях применяется искусственная генерация аномалий.

2) Semi-Supervised anomaly detection (режим распознавания частично с учителем)

Исходные данные при этом подходе представляют только нормальный класс. Обучившись на одном классе, система может определять принадлежность новых данных к нему, таким образом, определяя противоположный.

Алгоритмы, работающие в режиме распознавания частично с учителем, не требуют информации об аномальном классе экземпляров, вследствие чего они шире применимы и позволяют распознавать отклонения в отсутствие заранее определенной информации о них.

3) Unsupervised anomaly detection (режим распознавания без учителя)

Применяется при отсутствии априорной информации о данных. Алгоритмы распознавания в режиме без учителя базируются на предположении о том, что аномальные экземпляры встречаются гораздо реже нормальных. Данные обрабатываются, наиболее отдаленные определяются как аномалии. Для применения этой методики должен быть доступен весь набор данных, т.е. она не может применяться в режиме реального времени.

B. *Методы распознавания аномалий*

1) Классификация

Реализация данного метода основана на предположении о том, что нормальное поведение системы может определяться одним или несколькими классами. Таким образом, экземпляр, не принадлежащий ни к одному из

классов, является отклонением. Поиск аномалий проходит в два этапа: обучение и распознавание. Классификатор обучается на массиве маркированных данных, далее определяется принадлежность к одному из известных классов. В противном случае экземпляр помечается, как аномалия.

Наиболее широко применяемыми механизмами реализации распознавания аномалий с помощью классификации являются: нейронные сети, Байесовы сети, метод опорных векторов и метод на основе правил.

• Метод обнаружения аномалий на основе нейронных сетей включает два этапа. Первый: нейронная сеть обучается распознаванию классов нормального поведения на тренировочной выборке. Второй: каждый экземпляр поступает в качестве входного сигнала нейронной сети. Система, основанная на нейронных сетях, может распознавать как один, так и несколько классов нормального поведения.

Для нахождения аномалий посредством распознавания только одного класса используются репликативные нейронные сети [7]. Получившая широкое распространение технология нейронных сетей глубинного обучения (Deep Learning) также успешно применяется для решения данной задачи [31].

• Байесовской сетью является графическая модель, отображающая вероятностные зависимости множества переменных и позволяющая проводить вероятностный вывод с помощью этих переменных. Она состоит из двух основных частей: графическая структура, которая определяет набор зависимостей и независимостей во множестве случайных величин, представляющих субъекты предметной области, и набор вероятностных распределений, определяющих силу отношений зависимости, закодированных в графической структуре. Таким образом, применение Байесовской сети при идентификации аномалий заключается в оценке вероятности наблюдения одного из нормальных или аномальных классов.

Наиболее простой реализацией данного подхода является Наивный байесовский подход (Naive Bayes Approach). Описание алгоритма его работы приведено в [37].

• Метод опорных векторов (Support Vector Machine) применяется для поиска аномалий в системах, где нормальное поведение представляется только одним классом. Данный метод определяет границу региона, в котором находятся экземпляры нормальных данных. Для каждого исследуемого экземпляра определяется, находится ли он в определенном регионе. Если экземпляр оказывается вне региона, он определяется как аномальный. Описание работы метода опорных векторов приведено в [37].

• Последний метод основывается на генерации правил, которые соответствуют нормальному поведению системы. Экземпляр, который не соответствует этим правилам, распознается как аномальный. Алгоритм состоит из двух шагов. Первый: обучение правил из выборки с помощью одного из алгоритмов, таких как RIPPER, Decision Trees и т.д. Каждому правилу присваивается свое значение, которое пропорционально соотношению между числом обучающих экземпляров, классифицируемых, как правило, и общим числом обучающих экземпляров, покрываемых этим правилом. Второй шаг: поиск для каждого тестируемого экземпляра правила, которое наилучшим образом подходит к данному экземпляру. Система может распознавать как один, так и несколько классов поведения

Одним из подвидов систем на основе правил являются системы нечеткой логики. Они применяется, когда граница между нормальным и аномальным поведением системы является размытой. Каждый экземпляр является аномалией в некоторой степени удаленности от центра масс нормального интервала. Описание применения данного подхода к задаче поиска аномалий приведено в [40].

2) Кластеризация

Данная методика предполагает группировку похожих экземпляров в кластеры и не требует знаний о свойствах возможных отклонений. Выявление аномалий может строиться на следующем предположении:

– Нормальные экземпляры данных относятся к кластеру данных, в то время как аномалии не принадлежат ни к одному из кластеров.

Однако при такой формулировке может возникнуть проблема определения точных границ кластеров. Отсюда следует другое предположение:

– Нормальные данные ближе к центру кластера, а аномальные – значительно дальше.

В случае, когда аномальные экземпляры не являются единичными, они также могут образовывать кластеры. Таким образом, их выявление строится на следующем предположении:

– Нормальные данные образуют большие плотные кластеры, а аномальные – маленькие и разрозненные.

Одной из простейших реализаций подхода на основе кластеризации является алгоритм k-means, описанный в работе [39]. Методология применения подхода на основе кластеризации также приведена в [25].

3) Статистический анализ

При использовании этого подхода исследуется процесс, строится его профиль (модель), который затем сравнивается с реальным поведением. Если разница в реальном и предполагаем поведении системы, определяемая заданной функцией аномальности, выше установленного порога, делается вывод о наличии отклонений. Применяется предположение том, что нормальное поведение системы будет находиться в зоне высокой вероятности, в то время как выбросы – в зоне низкой.

Данный класс методов удобен тем, что не требует заранее определенных знаний о виде аномалии. Однако сложности могут возникать в определении точного статистического распределения и порога [2].

Методы статистического анализа подразделяются на

две основные группы:

• Параметрические методы. Предполагают, что нормальные данные генерируются параметрическим распределением с параметрами θ и функцией плотности вероятности P(x, θ), где x – наблюдение. Аномалия является обратной функцией распределения. Эти методы часто основываются на Гауссовой или регрессионной модели, а также их комбинации. Подробное описание параметрических методов приведено в [30].

• Не параметрические методы. Предполагается, что структура модели не определена априорно, вместо этого она определяется из предоставленных данных. Включает методы на основе гистограмм или функций ядра.

Базовый алгоритм поиска аномалий с применением гистограмм включает два этапа. На первом этапе происходит построение гистограммы на основе различных значений выбранной характеристики для экземпляров тренировочных данных. На втором этапе для каждого из исследуемых экземпляров определяется принадлежность к одному из столбцов гистограммы. Не принадлежащие ни к одному из столбцов экземпляры помечаются как аномальные. Подробный алгоритм, основанный на применении гистограмм, описан в [13].

Распознавание аномалий на основе функции ядра происходит аналогично параметрическим методам за исключением способа оценки плотности вероятности. Сравнение результатов работы данного метода с параметрическим методом на основе Гауссовой модели приведено в [16].

4) Алгоритм ближайшего соседа

Для использования данной методики необходимо определить понятие расстояния (меры похожести) между объектами. Примером может быть Евклидово расстояние.

Два основных подхода основываются на следующих предположениях:

• Расстояние до k-го ближайшего соседа. Для реализации этого подхода расстояние до ближайшего объекта определяется для каждого тестируемого экземпляра класса. Экземпляр, являющийся выбросом, наиболее отдален от ближайшего соседа.

• Использование относительной плотности основано на оценке плотности окрестности каждого экземпляра данных. Экземпляр, который находится в окрестности с низкой плотностью, оценивается как аномальный, в то время как экземпляр в окрестности с высокой плотностью оценивается как нормальный. Для данного экземпляра данных расстояние до его k-го ближайшего соседа эквивалентно радиусу гиперсферы с центром в данном экземпляре и содержащей k остальных экземпляров.

5) Спектральные методы

Спектральные методы находят аппроксимацию данных, используя комбинацию атрибутов, которые передают большую часть вариативности в данных.

Эта методика основана на следующем предположении: данные могут быть вложены в подпространство меньшей размерности, в котором нормальное состояние и аномалии проявляются иначе. Спектральные методы часто применяются совместно с другими алгоритмами для предобработки данных.

Исследование модификаций спектрального метода приведено в [24].

6) Гибридные методы

Гибридные методики распознавания аномалий, позволяют сочетать преимущества различных подходов. При этом различные техники могут применяться как последовательно, так и параллельно для достижения усредненных результатов.

Примерами гибридных систем распознавания аномалий могут служить следующие исследования:

• Совмещение кластеризации и алгоритма ближайшего соседа в работе [20].

• Параллельное использование совмещенных алгоритмов Байесовых сетей и решающих деревьев, а также алгоритма ближайшего соседа с классификацией на основе правил в работе [22].

• Совмещение метода опорных векторов и нейронной сети глубинного обучения в работе [6].

Обзор публикаций с описанием конкретных алгоритмов, реализующих рассмотренные выше методы, приведен в таблице 1.

Сравнительный анализ методов приведен в таблице 2.

ТАБЛИЦА I. ОБЗОР ПУБЛИКАЦИЙ

Метод	**Публикации**
Классификация на основе репликационных нейронных сетей	Dau, Ciesielski [2014]
Классификация на основе нейроных сетей глубинного обучения	Xu et al. [2015], Yan et al. [2015]
Классифкация на основе Байесовых сетей	Hill et al. [2009], Heard [2010]
Классификация на основе правил	Li et al.[2007], Nasr et al. [2016],
Классификация на основе систем нечеткой логики	Ghosh et al. [2017]
Классификация на основе метода опорных векторов	Amer et al. [2013], Zhang et al. [2015]
Кластеризация	Portnoy et al.[2001], Кокорева с соавт.[2015], Kiss et al. [2014]
Параметрические методы статистического анализа	Thatte et al. [2010]
Статистический анализ на основе гистограмм	Kind et al. [2009]
Статистический анализ на основе функции ядра	Latecki et al. [2007], Zhang et al. [2015], Sharma et al. [2016]
Алгоритм ближайшго соседа	Liao, Vemuri [2002], Su [2011]
Спектральные методы	Денисова, Мясников [2014],

ТАБЛИЦА II. СРАВНЕНИЕ МЕТОДОВ

Метод	Результат	Режим распознавания	Определение класса аномалий	Работа без предварительного обучения
Классификация	Метка	Supervised, semi-supervised	Да	Нет
Кластеризация	Метка	Unsupervised, semi-supervised	Нет	Нет
Статистический анализ	Степень	Semi-supervised	Нет	Нет
Алгоритм ближайшего соседа	Степень	Unsupervised	Нет	Да
Спектральные методы	Метка	Unsupervised, Semi-supervised	Нет	Да

C. *Распознавание аномалий в потоках данных*

Выявление аномалий в режиме реального времени может потребовать дополнительной модификации методов. Наиболее простым в реализации является алгоритм скользящего окна.

Данная методика используется для временных рядов, которые разбивается на некоторое число подпоследовательностей – окон (рис.2). Необходимо выбрать окно фиксированной длины, меньшей чем длина самого ряда, чтобы захватить аномалию в процессе скольжения. Поиск аномальной подпоследовательности осуществляется при помощи скольжения окна по всему ряду с шагом, меньшим длины окна.

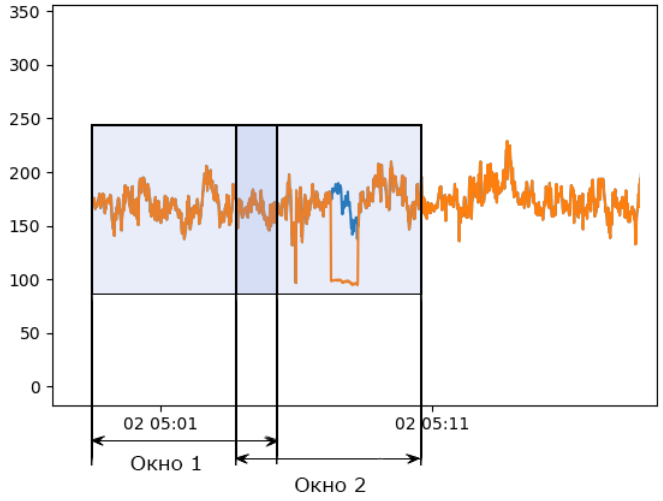

Рис.2 Применение алгоритма скользящего окна

Стоит отметить, что для методов, требующих наличия всего объема данных (функционирующих в режиме распознавания без учителя) применение данной техники может привести к повышенной неточности результатов, так как вычисления будут проводиться только для экземпляров в пределах окна.

В случае применения алгоритмов, основанных на предварительном построении модели с помощью классификации, существенных модификаций системы не требуется, поскольку на этапе распознавания каждый экземпляр обрабатывается отдельно.

Варианты построения систем распознавания аномалий в потоках данных приведены в [17], [28], [34].

IV. Способы выявления контекстуальных и коллективных аномалий

Все описанные выше методики применяются для поиска точечных аномалий, однако они также могут быть применены для распознавания коллективных и контекстуальных аномалий, при сведении их к точечным.

При поиске контекстуальных аномалий данный подход реализуется с помощью определения контекстуальных атрибутов и преобразования данных на их основе [8]. После этого к преобразованным данным можно применить один из методов идентификации точечных аномалий. Альтернативой данному методу является моделирование временных рядов на основе авторегрессии (например, построение модели ARIMA) [23] или преобразование их к символьным последовательностям [26], [35].

При поиске коллективных аномалий возможно определение подпоследовательностей фиксированной длины, как единичных объектов, однако при этом делается предположение, что все участки, являющиеся коллективными аномалиями, имеют одинаковую длину.

Описание комплексной методики выявления коллективных контекстуальных аномалий приведено в исследовании [12].

V. Заключение

Данная работа посвящена рассмотрению видов аномалий в потоках данных, а также обзору существующих методов и подходов к их поиску. Были проведены классификация и сравнение наиболее распространенных групп методов по основным критериям, приведены краткие описания алгоритмов. Кроме того, был осуществлен обзор конкретных реализаций и модификаций данных методов в публикациях последних лет.

Литература

[1] S. Agrawal, J. Agrawal, “Survey on Anomaly Detection using Data Mining Techniques”, Procedia Computer Science, vol. 60, 2015, pp. 708-713.

[2] M. Amer, M. Goldstein, S. Abdennadher, “Enhancing one-class support vector machines for unsupervised anomaly detection”, CM SIGKDD Workshop on Outlier Detection and Description, 2013, pp. 8-15

[3] V. Chandola, A. Banerjee, V. Kumar, “Anomaly detection: A survey”, ACM Computing Surveys, vol. 41(3), 2009, pp. 1–58.

[4] H. Dau, V. Ciesielski, A. Song, “Anomaly Detection Using Replicator Neural Networks Trained on Examples of One Class”, Simulated Evolution and Learning. Lecture Notes in Computer Science, vol 8886, 2014.

[5] S. Ghosh, A. Pal, A. Nag, S. Sadhu and R. Pati, “Network anomaly detection using a fuzzy rule-based classifier”, Computer, Communication and Electrical Technology, 2017 , pp. 61 -65.

[6] S. Erfani, Sarah, M. Baktashmotlagh, S. Rajasegarar, S. Karunasekera and C. Leckie, “A randomised nonlinear approach to large-scale

anomaly detection", 29th AAAI Conference on Artificial Intelligence, 2015, pp. 25–30, Hyatt Regency in Austin, Texas.
[7] S. Hawkins, H. He, G. J. Williams and R. A. Baxter, "Outlier detection using replicator neural networks", 4th International Conference on Data Warehousing and Knowledge Discovery. Springer-Verlag, 2002, pp. 170 – 180.
[8] M. Hayes, M.Capretz, "Contextual anomaly detection framework for big sensor data", Journal of Big Data, vol. 2(2), 2015.
[9] N.A. Heard, D.J. Weston, K. Platanioti, D.J. Hand, "Bayesian anomaly detection methods for social networks", Ann. Appl. Stat. 4 vol. 2, 2010, pp. 645 – 662.
[10] D.J. Hill, B. S. Minsker, and E. Amir, "Real-time Bayesian anomaly detection in streaming environmental data", Water Resour. Res., 45, 2009.
[11] H. Huang, "Rank Based Anomaly Detection Algorithms" Electrical Engineering and Computer Science – Dissertations, 2013, 331.
[12] Y. Jiang, C. Zeng, J. Xu and T. Li. "Real time contextual collective anomaly detection over multiple data streams", 2014.
[13] A. Kind, M. P. Stoecklin and X. Dimitropoulos, "Histogram-based traffic anomaly detection," IEEE Transactions on Network and Service Management, vol. 6(2), 2009, pp. 110-121.
[14] I. Kiss, B. Genge, P. Haller, G. Sebestyén, "Data clustering-based anomaly detection in industrial control systems", IEEE 10th International Conference on Intelligent Computer Communication and Processing (ICCP), 2014, pp. 275-281.
[15] L.J. Latecki, A. Lazarevic, D. Pokrajac, "Outlier Detection with Kernel Density Functions", Machine Learning and Data Mining in Pattern Recognition. Lecture Notes in Computer Science, vol 4571. Springer, Berlin, Heidelberg, 2007.
[16] R. Laxhammar, G. Falkman and E. Sviestins, "Anomaly detection in sea traffic - A comparison of the Gaussian Mixture Model and the Kernel Density Estimator," 12th International Conference on Information Fusion, Seattle, WA, 2009, pp. 756-763.
[17] W. Lee, S. J. Stolfo, P. K. Chan, E. Eskin, W. Fan, et al., "Real Time Data Mining-based Intrusion Detection", DARPA Information Survivability Conference & Exposition II, vol. 1, 2001.
[18] X. Li , J. Han , S. Kim , H. Gonzalez, "Roam: Rule- and motif-based anomaly detection in massive moving object data sets", 7th SIAM International Conference on Data Mining, 2007.
[19] Y. Liao, , V.R, Vemuri, "Use of K-Nearest Neighbor classifier for intrusion detection", Computers & Security, vol. 21(5), 2002, pp. 439-448
[20] W-C. Lin, S-W. Ke, C-F. Tsai, "An intrusion detection system based on combining cluster centers and nearest neighbors", Knowledge-Based Systems, vol. 78, 2015, pp. 13-21.
[21] A. A. Nasr, M. Z. Abdulmaged, "A Learnable Anomaly Detection System using Attributional Rules", International Journal of Computer Network and Information Security, vol. 8(11), 2016.
[22] M. Panda, A. Abraham, M. Patra, "Hybrid intelligent systems for detecting network intrusions". Security Comm. Networks, vol. 8, 2012, pp. 2741–2749
[23] E. H. M. Pena, M. V. O. de Assis and M. L. Proença, "Anomaly Detection Using Forecasting Methods ARIMA and HWDS," *2013* 32nd International Conference of the Chilean Computer Science Society (SCCC), Temuco, 2013, pp. 63-66.
[24] J. Piñeyro, A. Klempnow, V. Lescano, "Effectiveness of new spectral tools in the anomaly detection of rolling element bearings", Journal of Alloys and Compounds, vol. 310(1–2), 2000, pp. 276-279.
[25] L. Portnoy, E. Eskin, S. J. Stolfo, "Intrusion Detection with Unlabeled Data Using Clustering", Columbia University, New York, 2001.
[26] S. Sarkar, K. G Lore, S. Sarkar, V. Ramanan, S. R Chakravarthyet al., "Early Detection of Combustion Instability from Hi-speed Flame Images via Deep Learning and Symbolic Time Series Analysis", Annual Conference of the Prognostics and Health Management Society, vol.6, 2015.
[27] M. Sharma, K. Das, M. Bilgic, B. Matthews, D. Nielsen, N. Oza, "Active Learning with Rationales for Identifying Operationally Significant Anomalies in Aviation", Machine Learning and Knowledge Discovery in Databases. Lecture Notes in Computer Science, vol 9853, 2016
[28] M-Y. Su, "Real-time anomaly detection systems for Denial-of-Service attacks by weighted k-nearest-neighbor classifiers", Expert Systems with Applications, vol. 38(4), 2011, pp. 3492-3498.
[29] S. T. Teoh, K. Zhang, S-M. Tseng, K-L. Ma, S. F. Wu, "Combining visual and automated data mining for near-real-time anomaly detection and analysis in BGP" ACM workshop on Visualization and data mining for computer security. ACM, New York, NY, USA, 2004, pp. 35-44.
[30] G. Thatte, U. Mitra and J. Heidemann, "Parametric Methods for Anomaly Detection in Aggregate Traffic," IEEE/ACM Transactions on Networking, vol. 19(2), 2011, pp. 512-525.
[31] W. Yan and L. Yu, "On Accurate and Reliable Anomaly Detection for Gas Turbine Combustors: A Deep Learning Approach", Annual Conference of the Prognostics and Health Management Society, vol. 6, 2015.
[32] L. Zhang, J. Lin, and R. Karim, 'Adaptive Kernel Density-based Anomaly Detection for Nonlinear Systems', 2016.
[33] M. Zhang, B. Xu and J. Gong, "An Anomaly Detection Model Based on One-Class SVM to Detect Network Intrusions," 11th International Conference on Mobile Ad-hoc and Sensor Networks (MSN), Shenzhen, 2015, pp. 102-107.
[34] S. Zhao, M. Chandrashekar, Y. Lee and D. Medhi, "Real-time network anomaly detection system using machine learning," 11th International Conference on the Design of Reliable Communication Networks (DRCN), Kansas City, MO, 2015, pp. 267-270.
[35] С. Антипов, М. Фомина, «Проблема обнаружения аномалий в наборах временных рядов», Программные продукты и системы № 2, 2012, с. 78 – 82.
[36] Д. Заварзин "К вопросу поиска аномалий во временных рядах", Инновации в науке: сб. ст. по матер. XXIX междунар. науч.-практ. конф. № 1(26). – Новосибирск: СибАК, 2014.
[37] Е. В. Зубков, В. М. Белов, «Методы интеллектуального анализа данных и обнаружение вторжений», Вестник СибГУТИ № 1, 2016.
[38] А. Денисова, В. Мясников, «Обнаружение аномалий на гиперспектральных изображениях», КО. №2, 2014.
[39] Я. Кокорева, А. Макаров, «Поэтапный процесс кластерного анализа данных на основе алгоритма кластеризации k-means», Молодой ученый, №13, 2015. с. 126-128.
[40] А. Суханов, «Интеллектуальные методы обнаружения и прогнозирования аномальных событий в темпоральных данных», Диссертация на соискание ученой степени кандидата технических наук, РГУПС, Ростов-на-Дону, 2015.

The Overview Of Anomaly Detection Methods in Data Streams

Viacheslav P. Shkodyrev, Kamil I. Yagafarov, Valentina A. Bashtovenko, Ekaterina E. Ilyina

This article is devoted to the research of different approaches to the anomaly detection in time-series data, which includes identification and processing of deviations in data streams obtained from technological process. Detection of anomalies in system behavior helps not only to increase quality of these processes, but also to avoid emergency situations and accidents at the early stages. All of these demonstrates the relevance of the topic.

Existing methods and algorithms of anomaly detection are reviewed in this paper. The aim of research lies in structuring of available techniques and providing a subsequent selection of methods for system of anomaly detection development for Big Data streams.