Санкт-Петербургский государственный университет

Кафедра информационно-аналитических систем

Группа 21.Б08-мм

Реализация алгоритмов проверки метрических функциональных зависимостей

МАНАННИКОВ Степан Дмитриевич

Отчёт по учебной практике
в форме «Производственное задание»

Научный руководитель:
ассистент кафедры ИАС Г. А. Чернышев

Санкт-Петербург
2023

Оглавление

Введение

Профилирование данных — совокупность действий и процессов для определения метаданных в некотором наборе данных [1]. Профилирование данных позволяет получать общую информацию о наборе данных, например количество рядов или колонок, а также находить закономерности или ошибки в данных.

Функциональные зависимости (ФЗ) — метаданные, определяющие связь между множествами атрибутов таблицы. ФЗ позволяют определить точную зависимость одних атрибутов от других. Но у традиционных ФЗ есть недостатки, к примеру, они учитывают лишь строгое равенство между значениями, которые могут быть не равны, но очень близки друг к другу, из-за чего теряются возможные связи между атрибутами.

В отличии от традиционных ФЗ, метрические функциональные зависимости (МФЗ) учитывают погрешность или ошибки в данных. МФЗ основаны на вычислении расстояния между значениями в наборе данных и определении того, превышает ли это расстояние заданный параметр.

Desbordante [3] — высокопроизводительный наукоемкий профилировщик данных. На данный момент в Desbordante уже реализованы алгоритмы нахождения различных видов ФЗ, и имеется необходимость расширять функциональность платформы путем реализации новых примитивов, обеспечивающих продвинутое профилирование данных.

1. Постановка задачи

Целью работы является реализация на языке C++ примитива, обеспечивающего проверку МФЗ в рамках платформы Desbordante. Для достижения этой цели были поставлены следующие задачи:

- Произвести обзор предметной области;
- Реализовать алгоритмы поиска МФЗ, работающие в евклидовой метрике в $\mathbb{R}^n$;
- Обеспечить поддержку различных строковых метрик;
- Осуществить подсветку данных, максимальные расстояния между которыми превышают заданный параметр.

2. Обзор

2.1. Основные понятия

Определения взяты из статьи [6].

Обозначим область определения атрибута X как $\mathrm{dom}(X)$. Если X состоит из набора атрибутов $X = A_1A_2...A_k$, то

$$\mathrm{dom}(X) = \mathrm{dom}(A_1) \times \mathrm{dom}(A_2) \times ... \times \mathrm{dom}(A_k)$$

Определение 1 *Пусть* $d : \mathrm{dom}(Y) \times \mathrm{dom}(Y) \to \mathbb{R}$ *— метрика, определённая на области* Y. *Диаметр* $\Delta_d(S)$ *набора точек* S *в метрическом пространстве — максимальное расстояние между любой парой точек*: $\Delta_d(S) = \max_{p,q \in S} d(p, q)$.

Для кортежа t и атрибутов X, класс эквивалентности t, обозначаемый $[t]_X(r)$ — набор всех кортежей в r, совпадающих с t на X: $[t]_X(r) = \{u \in r | u[X] = t[X]\}$.

Обозначим партицию отношения r относительно X как множество классов эквивалентности $\pi_X(r) = \{[t]_X(r)\}$. Классы эквивалентности также называют кластерами.

Для набора кортежей $T \subseteq r$ обозначим проекцию T на набор атрибутов Y: $T[Y] = \{t[Y] | t \in T\}$.

Определение 2 *При заданном отношении* r, *определённом на схеме отношения* R, *наборах атрибутов* $X, Y \subseteq R$, *метрике* d *над* Y *и параметре* $\delta \geq 0$, *говорят, что метрическая функциональная зависимость* $X \xrightarrow{\delta} Y$ *удерживается, если*

$$\max_{T \in \pi_X(r)} \Delta_d(T[Y]) \leq \delta$$

2.2. Метрики

Для проверки МФЗ были использованы следующие метрики:

- Евклидова метрика для числовых типов в $\mathbb{R}^n$;
- Расстояние Левенштейна для строковых типов [9];
- Метрика, основанная на косинусном сходстве строк, представленных в виде векторов q-gram. Для вычисления расстояния между

строками строятся векторы, координаты которых — количество вхождений каждой подстроки размера q в соответствующую строку. Расстояние между векторами рассчитывается по формуле:

$$\mathrm{dist}(x, y) = 1 - \cos(\theta) = 1 - \frac{x \cdot y}{|x||y|}$$

Пример:

Пусть q = 2. Рассмотрим строки “abc” и “bcd”. Первая строка имеет по одному вхождению подстрок “ab” и “bc” и ноль вхождений “cd”. Вторая строка имеет ноль вхождений “ab” и по одному вхождению “bc” и “cd”. Следовательно, q-gram векторы строк “abc” и “bcd” будут, соответственно, (1, 1, 0) и (0, 1, 1). Косинусное расстояние этих двух строк равняется:

$$1 - \frac{1 \cdot 0 + 1 \cdot 1 + 0 \cdot 1}{\sqrt{1^2+1^2+0^2}\sqrt{0^2+1^2+1^2}} = 0.5$$

2.3. Алгоритмы

Для евклидовой метрики в $\mathbb{R}^1$ достаточно посчитать расстояние между максимальным и минимальным значением точек в кластере и сравнить его с параметром.

Для евклидовой метрики в $\mathbb{R}^2$ используется алгоритм *rotating calipers* [7], с помощью которого можно вычислить диаметр точек на плоскости за $O(n \log n)$ [8]. Данный алгоритм обходит выпуклую оболочку изначального набора точек и выдаёт пары точек, расстояние между которыми может быть максимальным.

Для строковых метрик и евклидовой метрики в $\mathbb{R}^n, n \geq 2$ используются алгоритм «грубой силы», рассчитывающий расстояние между всеми возможными парами точек за $O(n^2)$, а также линейный 2-приближённый алгоритм, делающий предположение, основываясь на том, что для любой точки найдется такая точка, что расстояние между ними — по крайней мере половина диаметра [5]. Для проверки МФЗ с помощью 2-приближённого алгоритма достаточно в каждом кластере сравнить с параметром удвоенное максимальное расстояние от любой из точек.

3. Реализация

3.1. Подсветка

Для получения подробной информации при нарушении МФЗ была реализована подсветка значений. Подсветка разделена на кластеры, для каждого из которых выводится набор записей, соответствующих точкам в данном кластере. Выводятся только те кластеры, в которых нарушается МФЗ; Если МФЗ удерживается, то вычисления подсветки не происходит. Подсветка также не вычисляется, если для проверки МФЗ используется 2-приближённый алгоритм. По умолчанию записи подсветки сортируются по убыванию относительно расстояния между соответствующей точкой и её наиболее удалённым соседом, но так же имеется возможность сортировать по индексу точки и по индексу самой удалённой от неё. Также при сортировки группируются вместе NULL и пустые значения. Запись позволяют пользователю получить следующую информацию о точке:

- Расстояние до наиболее удаленной точки и её индекс;
- Индикатор, показывающий, превышает ли данное расстояние заданный параметр;
- Индекс и значение точки.

Кроме того, в подсветке для каждого кластера выводится LHS значение, по которому совпадают кортежи.

```
Metric fd does not hold.
---------------------------------------- LHS value: 1
[X] | max dist: 5.468734       | index: 0     | value: (1, -0.132423) | furthest point index: 3      | furthest point value: (4, 4.440000)
[X] | max dist: 5.468734       | index: 3     | value: (4, 4.440000)  | furthest point index: 0      | furthest point value: (1, -0.132423)
[√] | max dist: 4.667236       | index: 1     | value: (2, 0.223000)  | furthest point index: 3      | furthest point value: (4, 4.440000)
[√] | max dist: 4.223408       | index: 4     | value: (5, 1.223000)  | furthest point index: 0      | furthest point value: (1, -0.132423)
[√] | max dist: 3.663899       | index: 5     | value: (4, 0.776101)  | furthest point index: 3      | furthest point value: (4, 4.440000)
[√] | max dist: 3.362157       | index: 2     | value: (3, 1.230000)  | furthest point index: 3      | furthest point value: (4, 4.440000)
---------------------------------------- LHS value: 2
[X] | max dist: 6.429352       | index: 11    | value: (10, 2.310000) | furthest point index: 9      | furthest point value: (4, -0.000100)
[X] | max dist: 6.429352       | index: 9     | value: (4, -0.000100) | furthest point index: 11     | furthest point value: (10, 2.310000)
[X] | max dist: 5.099628       | index: 10    | value: (9, 1.003000)  | furthest point index: 9      | furthest point value: (4, -0.000100)
[X] | max dist: 5.066826       | index: 8     | value: (8, 3.110000)  | furthest point index: 9      | furthest point value: (4, -0.000100)
[√] | max dist: 4.501766       | index: 6     | value: (6, 4.033000)  | furthest point index: 9      | furthest point value: (4, -0.000100)
[√] | max dist: 3.799214       | index: 7     | value: (7, 2.331000)  | furthest point index: 9      | furthest point value: (4, -0.000100)
```

Рис. 1: Пример вывода подсветки

Пример вывода подсветки представлен на Рис. 1. Для выбранных колонок в наборе данных есть два кластера, в каждом из которых шесть точек. В данном примере производилась проверка МФЗ $X \xrightarrow{5} Y$ для евклидовой метрики в $\mathbb{R}^2$, при этом МФЗ не удерживается. Записи отсортированы по убыванию относительно максимального расстояния, в записях, где максимальное расстояние больше пяти индикатор показывает «[X]», в записях, где максимальное расстояние меньше либо равно пяти индикатор показывает «[✓]».

3.2. Общая архитектура

Иерархия классов алгоритмов представлена на Рис. 2. Некоторые детали реализации опущены. Классы и функции, реализованные автором данной работы, отмечены зелёным. Синим отмечены классы и функции, уже имеющиеся в системе Desbordante.

В начале работы класс **MetricVerifier**, в зависимости от введённых пользователем параметров, определяет функцию **ClusterFunction**, которая отвечает за генерацию точек, на которых будет работать алгоритм проверки МФЗ, запуск алгоритма на данном наборе точек и вычисление подсветки. Эта функция затем вызывается на каждом кластере. Методы класса **MetricVerifier** отвечают за работу алгоритмов проверки МФЗ и за определение **ClusterFunction**. Методы вычисления точек для алгоритмов проверки МФЗ и для алгоритмов подсветки вынесены в отдельный класс **PointsCalculator**. Методы, отвечающие за вычисление и сортировку подсветки вынесены в класс **HighlightCalculator**.

Реализация векторов q-gram представлена в классе **QGramVector**. Вектор реализован с помощью контейнера *unordered_map*, который сопоставляет подстроке количество вхождений этой подстроки в строку.

Функции **CalculateConvexHull** и **GetAntipodalPairs** отвечают за вычисление выпуклой оболочки и за работу алгоритма *rotating calipers* соответственно.

Функция, вычисляющая расстояние Левенштейна, была уже реа-

лизована в Desbordante, далее она была использована автором данной работы в алгоритмах проверки МФЗ.

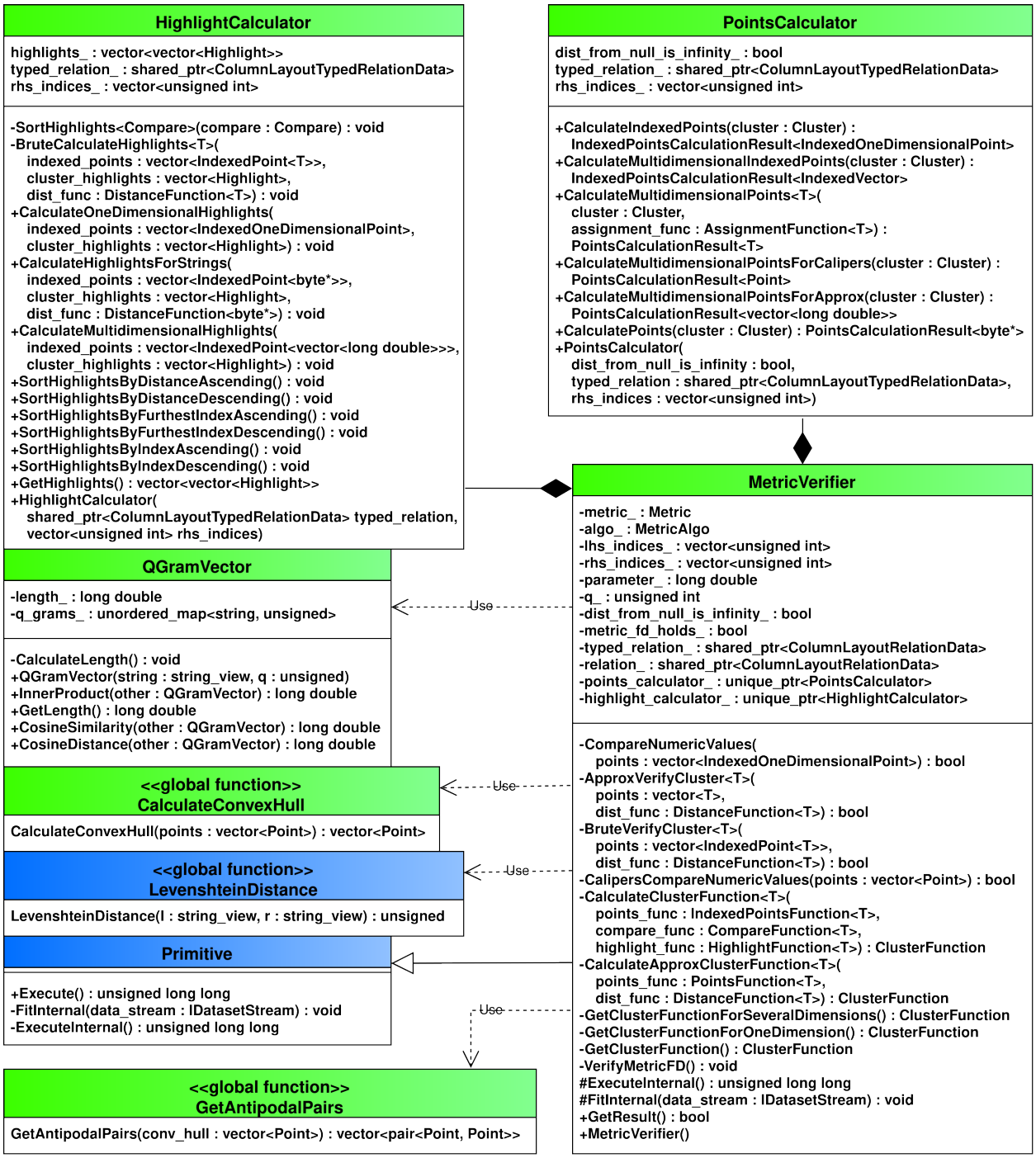

Рис. 2: Иерархия Классов

3.3. Особенности реализации

Для построения выпуклой оболочки был реализован алгоритм *monotone chain* [2]. Кроме того, алгоритм *rotating calipers* был немного модифицирован: при обработке параллельных линий он выдаёт две пары точек вместо четырёх, что позволяет сократить количество сравнений при проверке МФЗ.

Расстояние от пустого значения считается равным нулю. Расстояние от NULL значения равно нулю либо бесконечности в зависимости от заданного параметра.

4. Эксперимент

Для алгоритмов проверки МФЗ было осуществлено тестирование на корректность, а также были выполнены замеры производительности.

4.1. Тестирование

Тестирование производилось с помощью библиотеки GoogleTest [4]. Результаты работы алгоритмов проверки МФЗ, а также вычисленная примитивом подсветка сравнивались с ожидаемыми результатами. Всего было написано 27 тестов для проверки корректности работы алгоритмов МФЗ и 18 тестов для проверки информации, полученной в результате вычисления подсветки.

Тестирование помогло выявить некоторые ошибки во время реализации, которые затем были исправлены. На данный момент, все тесты успешно выполняются.

4.2. Измерение времени работы

Характеристики системы, на которой проводились эксперименты: Intel Core i7-4500U @ 3.0GHz, 8 GiB RAM, Arch Linux, Kernel 6.1.1-arch1-1, компилятор C++ gcc 12.2.0. Компиляция Desbordante производилась с флагом оптимизации -O3.

Алгоритмы проверки МФЗ запускались на наборе данных *iowa*, на отдельных столбцах для строковых метрик и для евклидовой метрики в $\mathbb{R}^2$. Набор данных был разделён на несколько частей с разным количеством строк. Все эксперименты проводились с заданным параметром δ, при котором МФЗ $X \xrightarrow{\delta} Y$ удерживается, благодаря чему алгоритмы вычисления подсветки не запускались, иначе это могло бы повлиять на общее время работы. Косинусное расстояния высчитывалось на 2-gram векторах.

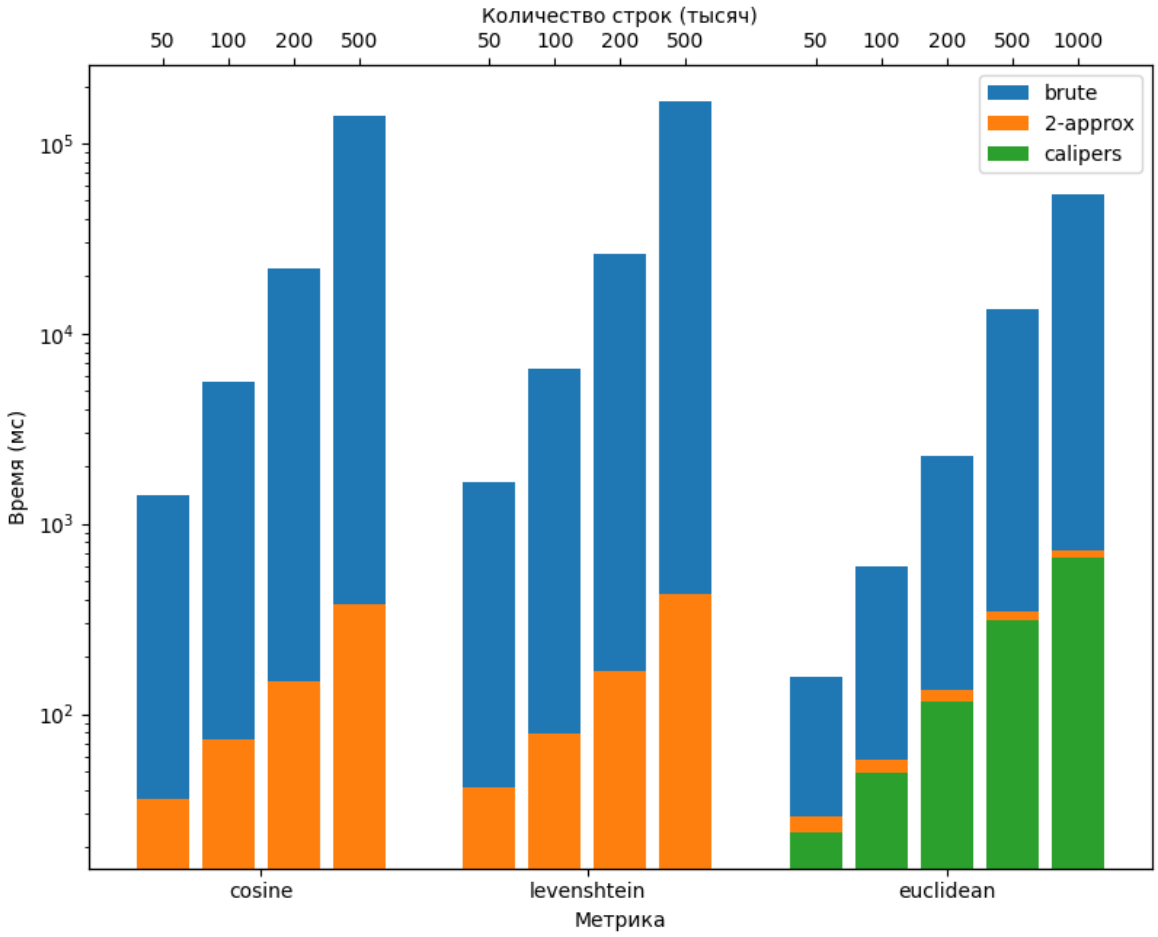

Рис. 3: Время работы алгоритмов

4.3. Результаты

Результаты представлены на Рис. 3. Время работы на графике не включает в себя время, затраченное системой Desbordante на обработку таблицы. Цветом отмечены алгоритмы проверки МФЗ.

Исходя из информации на графике, можно сделать вывод, что реализованные алгоритмы, в целом, соответствуют своей асимптотике. Также, на данном наборе данных время работы 2-приближённого алгоритма слегка превосходит время работы алгоритма *rotating calipers*. Кроме того, вычисление косинусного расстояния происходит немного быстрее, чем вычисление расстояния Левенштейна. Соотношение времени работы алгоритма «грубой силы» и 2-приближённого алгоритма совпадает с соотношением времени работы данных алгоритмов, полученным в результате экспериментов в статье [6].

Заключение

В ходе данной работы были достигнуты следующие результаты:

- Произведён обзор предметной области;
- Реализованы алгоритмы проверки МФЗ на языке C++ для евклидовой метрики;
- Добавлена возможность использовать строковые метрики для соответствующих алгоритмов;
- Реализована система подсветки для получения полной информации, если МФЗ не удерживается.

Исходный код доступен на Github[1] (имя пользователя Aviu00).

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

Список литературы

[1] Abedjan Ziawasch, Golab Lukasz, Naumann Felix. Data Profiling: A Tutorial // Proceedings of the 2017 ACM International Conference on Management of Data. — SIGMOD '17. — New York, NY, USA : Association for Computing Machinery, 2017. — P. 1747–1751.

[2] Andrew A.M. Another efficient algorithm for convex hulls in two dimensions // Information Processing Letters. — 1979. — Vol. 9, no. 5. — P. 216–219.

[3] Desbordante: a Framework for Exploring Limits of Dependency Discovery Algorithms / Maxim Strutovskiy, Nikita Bobrov, Kirill Smirnov, George Chernishev // 2021 29th Conference of Open Innovations Association (FRUCT). — 2021. — P. 344–354.

[4] GoogleTest - Google Testing and Mocking Framework. — `https://github.com/google/googletest`.

[5] Indyk Piotr. Sublinear Time Algorithms for Metric Space Problems // Proceedings of the Thirty-First Annual ACM Symposium on Theory of Computing. — STOC '99. — New York, NY, USA : Association for Computing Machinery, 1999. — P. 428–434.

[6] Metric Functional Dependencies / Nick Koudas, Avishek Saha, Divesh Srivastava, Suresh Venkatasubramanian // 2009 IEEE 25th International Conference on Data Engineering. — 2009. — P. 1275–1278.

[7] Preparata Franco P., Shamos Michael Ian. Convex Hulls: Extensions and Applications // Computational Geometry: An Introduction. — New York, NY : Springer New York, 1985. — P. 150–184. — ISBN: 9781461210986.

[8] Toussaint Godfried. Solving Geometric Problems with the Rotating Calipers // In Proc. IEEE MELECON '83. — 1983. — P. 10–02.

[9] Левенштеин В. И. Двоичные коды с исправлением выпадений, вставок и замещений символов // Докл. АН СССР. — 1965. — Т. 163. — С. 845–848.