Московский авиационный институт

# Реферат

на тему: «Метод идентификации музыкальных произведений по аудио фрагментам концертных исполнений»

Выполнил: студент гр. М8О-406Б
Давид Гринберг

Москва 2020

# Содержание

# 1 Введение

В современном мире наблюдаются следующие тенденции:

— Люди нетерпеливы и привыкли к легкому и быстрому доступу к информации

— Количество доступной информации неукротимо растет и человек не в состоянии справиться с ее потоком без использования поисковиков

— Существенная часть информации – аудиофайлы

Конкретный пример: люди, посещающие различные музыкальные мероприятия, часто сталкиваются с ситуацией, когда на сцене выступает музыкант, а название песни или даже имя исполнителя неизвестно (например, на фестивале). Конечно, можно спросить ближайшего человека, но в таких местах обычно очень шумно. Кроме того нет гарантий, что у этого человека найдется ответ на вопрос. Существует множество методов и сервисов для нахождения музыкальных произведений по отрывку, однако у них есть ряд ограничений:

— Сервисы вроде Shazam способны распознавать только оригиналы

— Некоторые сервисы умеют искать произведения по мелодии, но у них довольно низкая точность.

— Сервисы, которые ищут каверы или ремиксы (для защиты авторских прав) не приспособлены к нахождению зашумленных отрывков, поскольку предполагают, что кавер записывался в студийных условиях

В этой работе рассмотрен метод, лишенный всех вышеобозначенных недостатков. Поскольку музыкальных произведений в мире очень много (например, в социальной сети VK 400 миллионов треков), то очень важно уметь быстро и эффективно по памяти обрабатывать эти данные. Кроме того этот метод применим не только к песням, так как аудиофайл представляет собой временной ряд.
Цели данной работы:

1. Разработать библиотеку, которая предоставляла бы гибкий и удобный интерфейс для эффективной обработки и поиска аудиофайлов

2. Реализовать клиент для идентификации концертных записей, используя разработанную библиотеку

# 2 Теоретическая часть

## 2.1 Звук

Звук - это вибрация, которая распространяется через воздух (или воду). Например, при прослушивании музыки с компьютера колонки производят вибрации, которые распространяются по воздуху, пока не достигнут уха человека.

Вибрации можно смоделировать с помощью синусоидальных волн.

### 2.1.1 Чистый тон

Чистый тон - это тон синусоидальной формы волны. Характеристики синусоиды:

— Частота: количество циклов в секунду. Единица измерения - Герц (Гц), например, 100 Гц = 100 циклов в секунду.
— Амплитуда (связана с громкостью звука): размер каждого цикла.

Эти характеристики расшифровываются человеческим ухом для формирования звука. Человек может слышать чистые тоны от 20 Гц до 20000 Гц, и этот диапазон уменьшается с возрастом. Для сравнения, свет, который видит человек, состоит из синусоид от $4 * 10^{14}$ Гц до $7.9 * 10^{14}$ Гц.

Человеческое восприятие громкости зависит от частоты чистого тона. Например, чистый тон с амплитудой равной 10 и частотой 30 Гц будет тише, чем чистый тон с амплитудой 10 и частотой 1000 Гц. Человеческие уши воспринимают звук в соответствии с психоакустической моделью.

Чистых тонов в природе не существует, однако каждый звук в мире - это сумма нескольких чистых тонов с разными амплитудами.

### 2.1.2 Музыкальные ноты

Ноты разделены на октавы. В большинстве западных стран октава представляет собой набор из 8 нот (A, B, C, D, E, F, G в большинстве англоязычных стран) со следующим свойством:

— Частота ноты в октаве удваивается в следующей октаве. Например, частота A4 (A в 4-й октаве) на частоте 440 Гц в 2 раза превышает частоту A3 (A в 3-й октаве) на 220 Гц и в 4 раза больше частоты A2 (A во 2-й октаве) на 110 Гц.

Частотная чувствительность ушей логарифмическая. Это означает, что:

— между 32.70 Гц и 61.74 Гц (1-я октава)
— или между 261.63 Гц и 466.16 Гц (4-я октава)
— или между 2 093 Гц и 3 951.07 Гц (7-я октава)

Человеческие уши распознают одинаковое количество нот.

### 2.1.3 Тембр

Одна и та же нота может звучать по-разному, если ее играют гитара, пианино или скрипка. Причина в том, что у каждого инструмента свой тембр для данной ноты.

Для каждого инструмента воспроизводимый звук представляет собой множество частот, которые звучат как данная нота (научный термин для музыкальной ноты - высота звука). Этот звук имеет основную частоту (самая низкая частота) и несколько обертонов (любая частота выше основной).

Большинство инструментов производят гармоничные звуки. Для этих инструментов обертоны являются кратными основной частоты и называются гармониками. Например, композиция чистых тонов A2 (основной), A4 и A6 является гармонической, тогда как композиция чистых тонов A2, B3, F5 является негармоничной.

Многие ударные инструменты (например, тарелки или барабаны) создают негармоничные звуки.

Примечание: высота звука (воспринимаемая музыкальная нота) может отсутствовать в звуке, воспроизводимом инструментом. Например, если инструмент воспроизводит звук с чистыми тонами A4, A6 и A8, человеческий мозг интерпретирует полученный звук как ноту A2. Эта нота / высота звука будет A2, тогда как самая низкая частота звука - A4 (этот факт называется отсутствующим основным).

### 2.1.4 Цифровое представление звука

Чтобы хранить и проигрывать звук на электронных устройствах, его нужно оцифровать.

#### 2.1.4.1 Семплирование

Аналоговые сигналы - это непрерывные сигналы, что означает, что если взять одну секунду аналогового сигнала, то ее можно разделить на части, которые длятся доли секунды. В цифровом мире нельзя позволить себе хранить бесконечное количество информации. Нужно иметь минимальную единицу времени, например, 1 миллисекунду. В течение этого промежутка времени звук не сможет измениться, поэтому этот промежуток должен быть достаточно коротким, чтобы цифровой сигнал звучал как аналоговый, и достаточно большой, чтобы ограничить пространство, необходимое для хранения.

Эта задача называется семплированием.

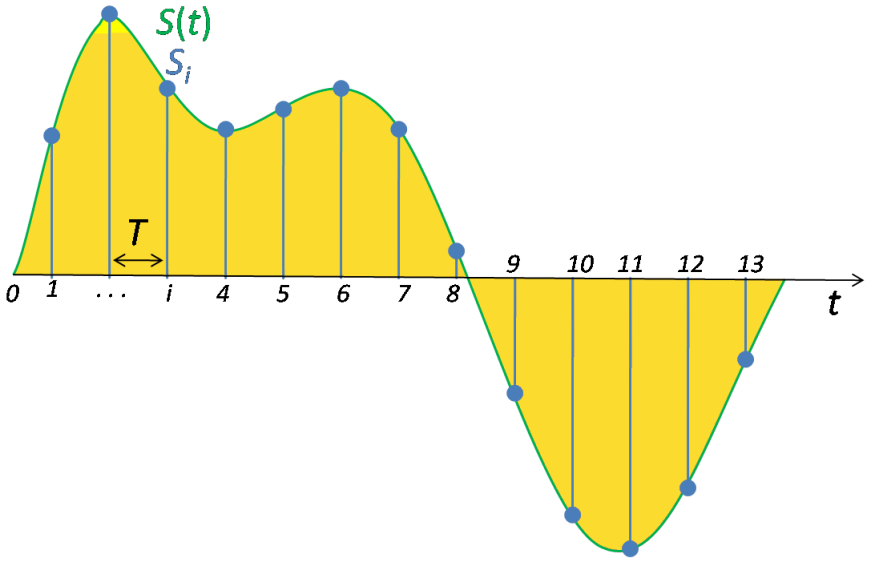


Рисунок 2.1 — Пример семплирования

Стандартная единица времени в цифровой музыке составляет 44100 единиц (или сэмплов) в секунду. Эта величина выбрана в связи с теоремой Котельникова, из которой следует, что для оцифровки синусоиды частоты $F$ требуется, по меньшей мере, 2 точки на цикл. Так как человек слышит в пределах 20 кГц, то соответственно для оцифровки сигналов нужно использовать вдвое больше точек.

#### 2.1.4.2 Квантизация

Громкость измеряет разницу между самым низким и самым высоким уровнем звука в песне. Как и в случае с семплированием, для оцифровки сигнала требуется иметь ограниченное количество уровней громкости.

Эта задача называется квантизацией.

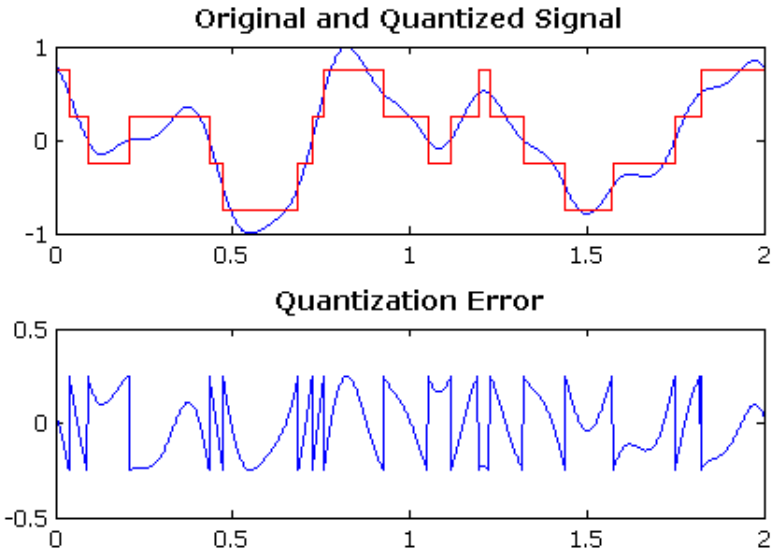


Рисунок 2.2 — Пример квантизации

### 2.1.5 Спектрограмма

Музыкальное произведение исполняется несколькими инструментами и певцами. Все эти инструменты производят комбинацию синусоидальных волн на нескольких частотах, и в целом комбинация синусоидальных волн еще больше.

Можно визуализировать музыку с помощью спектрограммы. В большинстве случаев спектрограмма представляет собой трехмерный график, где:

— по оси X представлено время (точнее его промежуток),

— по оси Y представлена частота чистого тона

— третье измерение описывается цветом и соответствует амплитуде частоты в определенное время.

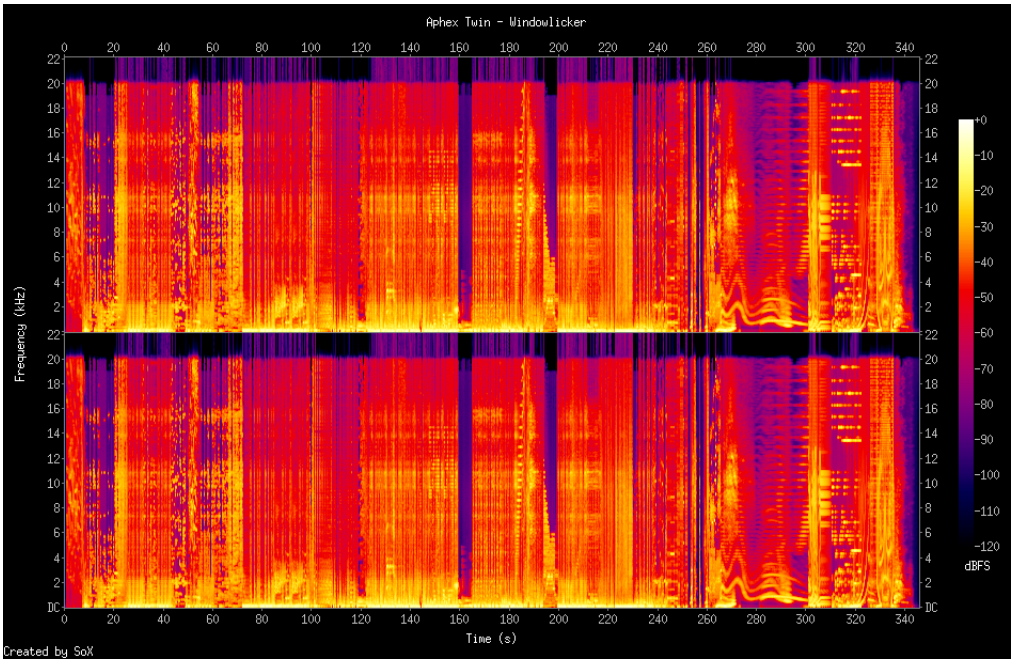


Рисунок 2.3 — Спектрограмма Aphex Twin – Windowlicker

### 2.1.6 Дискретное преобразование Фурье

Для того, чтобы вычислить спектрограмму дискретного сигнала, нужно найти его частоты. Это можно сделать с помощью дискретного преобразования Фурье (ДПФ). ДПФ применяется к дискретным сигналам и его результатом является дискретный спектр (частоты внутри сигнала).

Формула ДПФ:

$$X(n) = \sum_{k=0}^{N-1} x(k) \times e^{-i\frac{2\pi nk}{N}}$$

, где $N$ – размер окна (количество семплов), $X(n)$ – $n$-ый диапазон частот, $x(k)$ – $k$-ый семпл сигнала

## 2.2 Техника акустического отпечатка

Для того, чтобы эффективно хранить и искать аудиофайлы, нужно найти какое-нибудь компактное представление, которое при этом будет максимально правдоподобно их описывать. Это представление называется акустическим отпечатком (фингерпринтом) аудиофайла. Существует множество видов таких отпечатков, но большинство методов находят представление аудиофайлов в виде вектора хешей.

Факторы эффективности:

1. Хеши максимизируют произведение функций энтропии и точности:

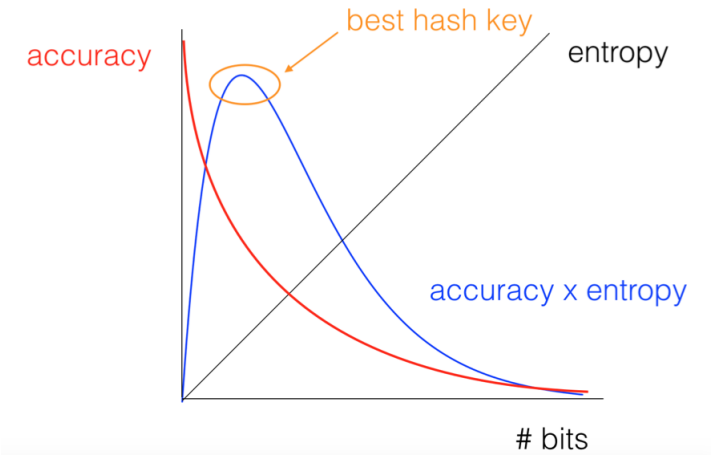


Рисунок 2.4 — Зависимость эффективности от точности и энтропии

2. Биты хешей сбалансированы, декоррелированы и имеют высокую дисперсию

### 2.2.1 Общая идея

Многие алгоритмы фингерпринтинга выглядят так:

1. Посчитать спектрограмму аудиофайла
2. Применить на ней какую-либо оконную функцию (спектрально-временные фильтры)
3. Конвертировать результат в вектор хешей

## 2.3 Метод хешпринтов

Этот метод предложен в [1]. Он, как и многие другие, находит представление аудиофайла в виде вектора хешей.

Метод отличается следующими характеристиками:

1. Обучение без учителя
2. Высокая адаптивность к данным
3. Независимость от силы сигнала (громкости звука)

Самой важной отличительной чертой метода является обучение без учителя. Такие методы, как, например, Chromaprint, описанный в [2], используют заранее подготовленные спектрально-временные фильтры. Метод хешпринтов находит эти фильтры непосредственно при индексации, что позволяет ему учитывать специфику данных.

Рисунок 2.5 — Фильтры, используемые Chromaprint

### 2.3.1 Алгоритм вычисления хешпринта

Для вычисления хешпринта, содержащего $N$ бит, нужно проделать следующее:

1. Посчитать спектрограмму.

Результат этапа: матрица $Spectrogram \in \mathbb{R}^{B \times n}$, где $B$ – количество частотных диапазонов,

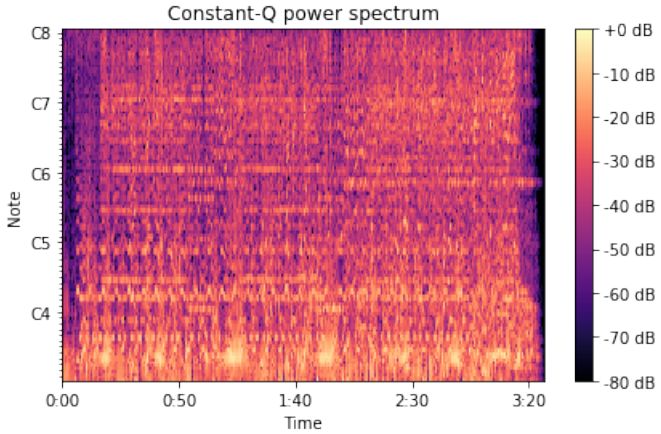


Рисунок 2.6 — Спектрограмма

$n$ – количество временных диапазонов.

2. Собрать контекстные фреймы полученной спектрограммы. Фреймы рассчитываются следующим образом:

$$frame_i = V_{i-w}...V_{i+w}$$

, где $V_i$ – столбец спектрограммы, $w$ – количество столбцов контекста.

Результат этапа: матрица $Frames \in \mathbb{R}^{Bw \times n}$

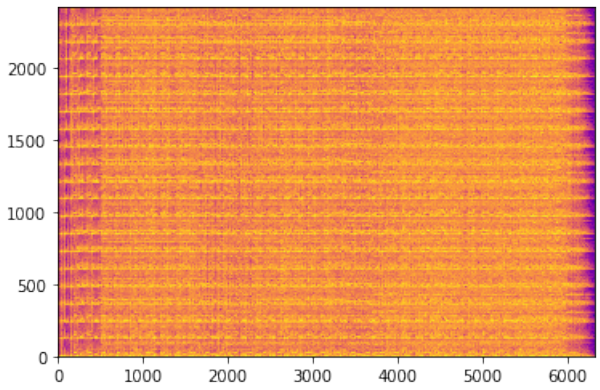


Рисунок 2.7 — Матрица фреймов

3. Применить к фреймам спектрально-временные фильтры. Фильтры представляют собой $N \times Bw$ матрицу и расситываются с помощью алгоритма обучения без учителя путем решения задачи оптимизации.
Результат этапа: матрица признаков $Features \in \mathbb{R}^{N \times n}$.

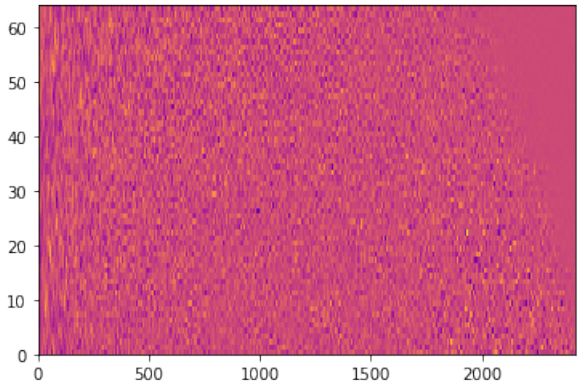


Рисунок 2.8 — Фильтры

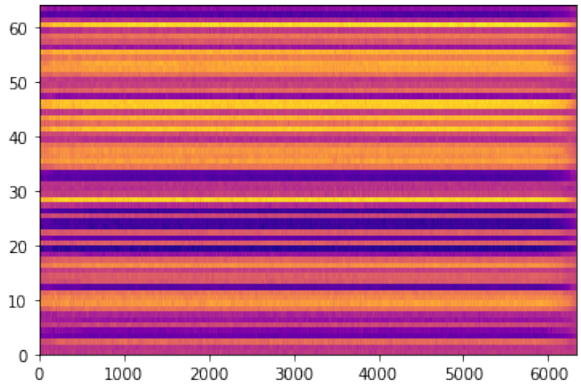


Рисунок 2.9 — Матрица признаков

4. Посчитать дельту – изменение признаков в течение промежутка $T$. Дельта рассчитывается по формуле:

$$\Delta_i = feature_i - feature_{i+T}$$

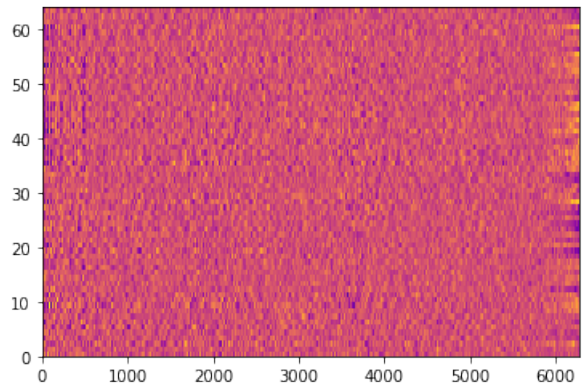


Рисунок 2.10 — Дельта

5. Наложить функцию порога и упаковать результат в хешпринты:

$$hashprint_i = intN(\Delta_i > 0)$$

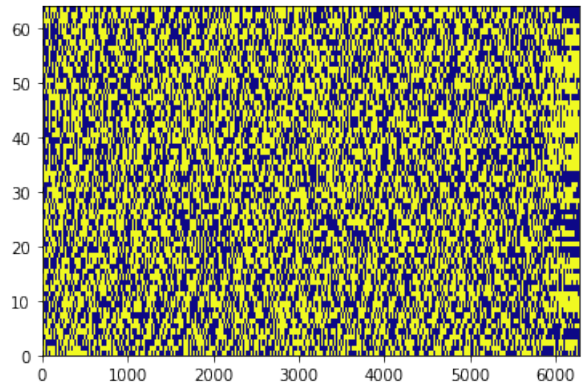


Рисунок 2.11 — Результат наложения функции порога

### 2.3.2 Вычисление спектрально-временных фильтров

Фильтры подбираются таким образом, чтобы признаки, полученные при их наложении, имели максимальную дисперсию и в то же время были декоррелированы. Для этого можно применить метод главных компонент (PCA):

1. Посчитать ковариационные матрицы для всех матриц фреймов в базе и просуммировать их.

Результат этапа: матрица $CovarianceMatrix \in \mathbb{R}^{Bw \times Bw}$

2. Найти $N$ собственных векторов с максимальными собственными значениями.

Результат этапа: матрица $Filters \in \mathbb{R}^{N \times Bw}$

## 2.4 Задача идентификации живых отрывков

Поскольку речь идет о нечетком поиске, то мы хотим учесть как можно больше нюансов (признаков) сигнала. Поэтому будем представлять аудиофайлы в виде

64-битных хешпринтов. Также в качестве спектрограммы возьмем CQT спектрограмму - она хороша тем, что ее частотные диапазоны подобраны таким образом, что они соответствуют конкретным нотам.

### 2.4.1 Хранение и поиск

У живого исполнения с большой долей вероятности будет много общих признаков в определенные моменты времени со студийным оригиналом. Также мы знаем, что если какой-то аудиофайл является отрывком другого, то это значит что существует такой отступ $offset$, что $fragment \approx original[offset..]$.

Таким образом, задача обретает вид:

$$ans = \underset{original,offset}{\arg\min} \; d(fragment, original[offset..])$$

, где $d$ – некоторая метрика. Проще говоря, мы хотим минимизировать расстояние между отрывком и студийным оригиналом. Также стоит отметить, что такой подход подразумевает, что музыкант не сильно изменил темп исполнения по сравнению с оригиналом.

### 2.4.2 Почему обратный индекс не подходит

Хранение и поиск аудиофайлов можно было бы организовать следующим образом:

1. Построим обратный индекс вида:

$$hashprint \rightarrow [...\{song\_id, offset\}...]$$

2. Заводим счетчик. Для каждого хешпринта отрывка найдем все пары $\{song\_id, offset\}$, в которых содержатся соответствующие хешпринты и увеличим для них счетчик.

3. Возвращаем пару с максимальным значением счетчика.

У такого подхода есть одна проблема. Мы имеем дело с пространством довольно большой размерности и вероятность того, что в отрывке и оригинале в один момент времени встретятся полностью одинаковые хешпринты очень мала.

### 2.4.3 Вариант авторов метода

В качестве метрики берется сумма расстояний Хемминга между соответствующими хешпринтами при прикладывании отрывка к оригиналу. Также подразумевается, что есть некий сервис, который по GPS определит ближайший концерт и соответственно исполнителя, что позволит проверить лишь малую часть базы.
Сам же поиск выглядит так:

1. Для каждого оригинала из базы: прикладываем к нему отрывок и ищем такой отступ, чтобы сумма расстояний Хемминга между соответствующими хешпринтами была минимальной.

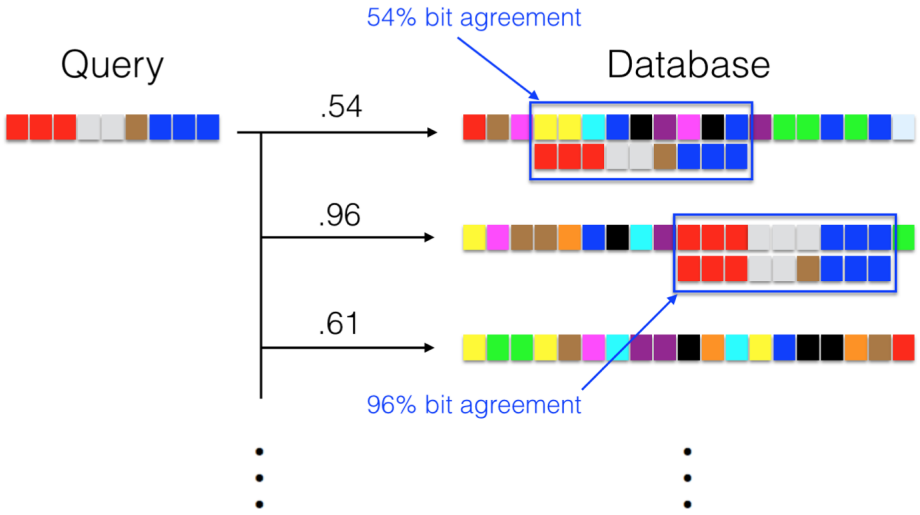


Рисунок 2.12 — Пример обработки запроса

2. Собираем результаты, сортируем и возвращаем top-N.

Минус такого подхода – GPS сервис:

1. Точка отказа. Упадет сервис – упадет вся система
2. Затраты на разработку и поддержку сервиса
3. Если это внешний сервис, то первый пункт становится еще большей проблемой

Конечно, чаще всего человек знает на чей концерт он пришел и нужды в GPS сервисе не будет. С другой стороны в наше время наблюдается рост числа музыкантов (по крайней мере в России), поскольку делать музыку и делиться ей стало легче чем когда-либо. Многие из этих музыкантов неизвестны широкому кругу слушателей, и они не готовы тратить деньги на рекламу своего творчества, но периодически выступают на различных концертах и фестивалях, где публика их возможно не узнает.

Но главная проблема в том, что этот метод имеет большой потенциал для применения в других областях (например, классификация ЭКГ), в которых GPS сервис никак не поможет. Чтобы сократить время поиска, нужно научиться каким-то образом отсекать большую часть данных.

### 2.4.4 Метод k-ближайших соседей

Можно взять вариант поиска с обратным индексом и заменить обратный индекс на поиск k-ближайших соседей. Нам не критично, чтобы находились абсолютно все ближайшие соседи, поэтому можно использовать методы приближенного поиска ближайших соседей (approximate nearest neighbor), у которых выше производительность.

Мною было рассмотрено два алгоритма:

— Метод случайных проекций.
— Иерархический маленький мир (HNSW)

Лучше всего себя показал HNSW.

#### 2.4.4.1 Метод случайных проекций

Разбивает пространство гиперплоскостями и строит несколько бинарных деревьев. Реализация: annoy

– Производительность хуже, чем у другого алгоритма
– Требует много памяти

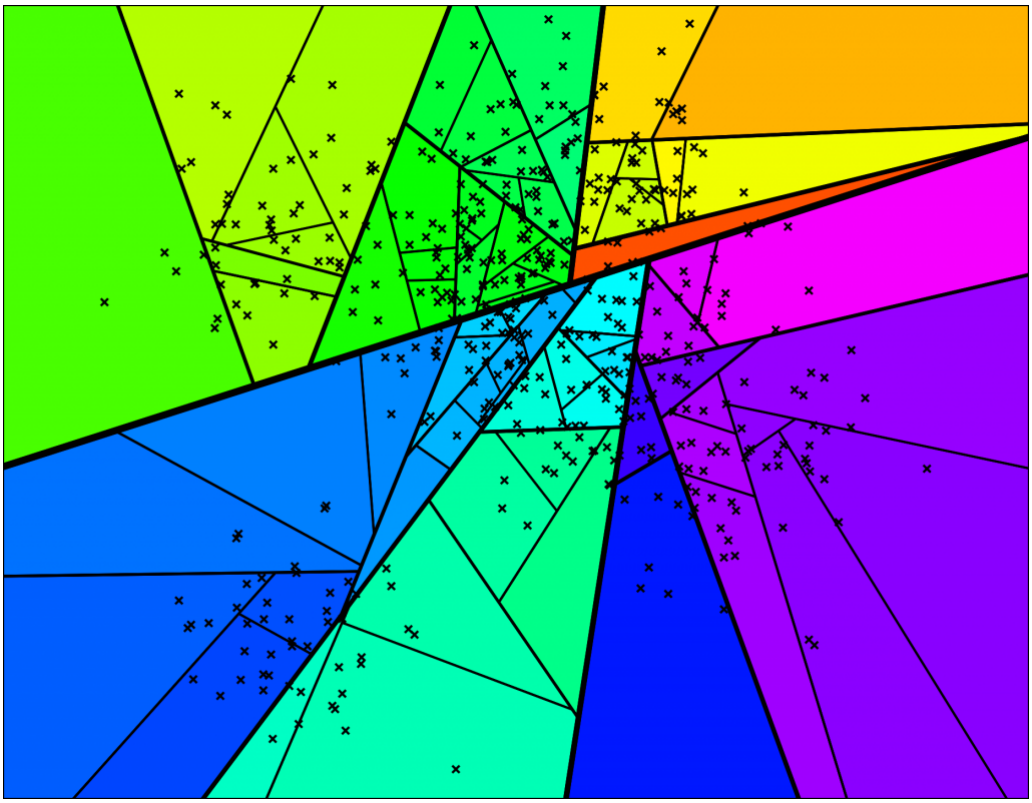

Рисунок 2.13 — Пространство, разбитое гиперплоскостями

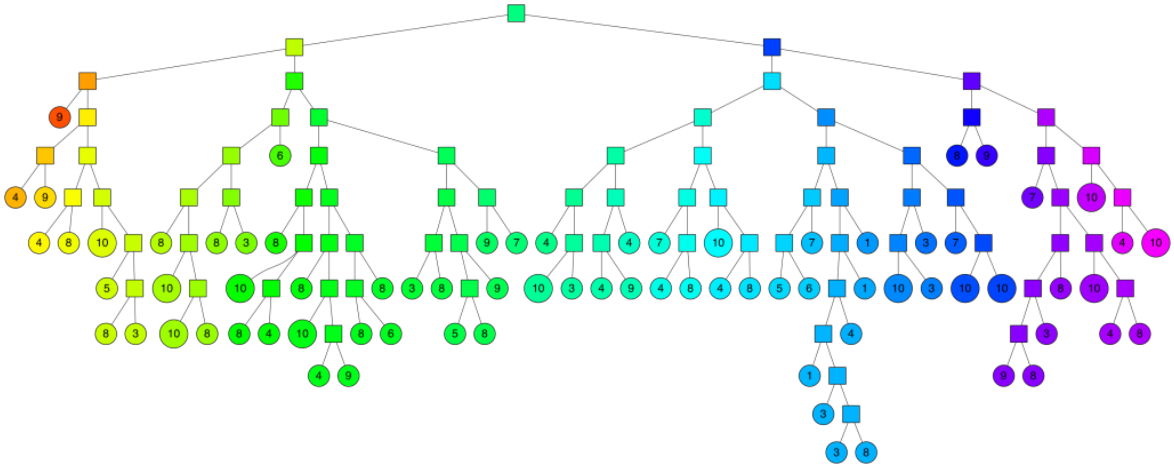

Рисунок 2.14 — Соответствующее разбиению бинарное дерево

#### 2.4.4.2 Иерархический маленький мир

Граф «мир тесен» – это такой граф, в котором типичное расстояние $L$ между двумя произвольно выбранными вершинами растёт пропорционально логарифму от числа вершин: $L \propto \log N$.

В библиотеке nmslib реализован алгоритм HNSW, описанный в [3]. HNSW совмещает в себе свойства графа «мир тесен» и списка с пропусками. По сути это иерархия графов, которая состоит из $n$ слоев: на нулевом слое представлены все объекты, а по мере увеличения слоя – все меньшая и меньшая их подвыборка. При этом все объекты на слое $n + 1$ есть и на слое $n$.

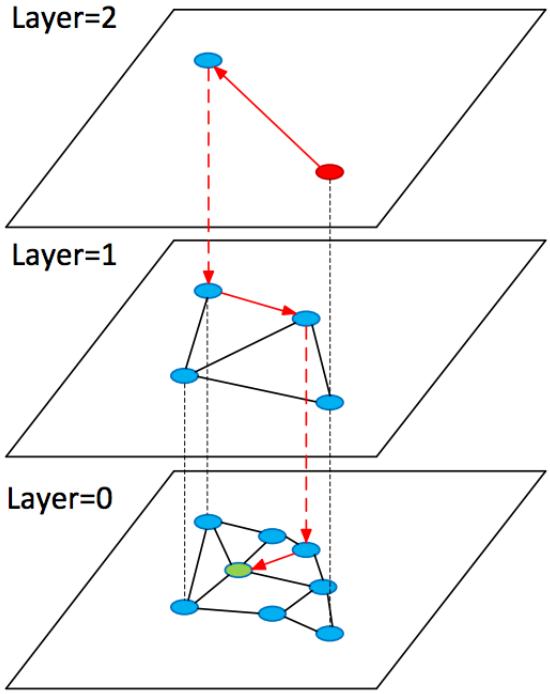


Рисунок 2.15 — Пример графа

При поиске старт происходит со случайной вершины в графе верхнего слоя, там мы быстро находим близкие к запросу вершины и возобновляем поиск с них на предыдущем слое.

Алгоритм очень легко масштабируется – можно сделать ребро между вершинами, находящимися на разных физических машинах. Также у HNSW хорошая производительность и небольшие затраты на память.

# 3 Практическая часть

В рамках ВКР была реализована библиотека hpfw (ссылка на Github). С использованием инструментов библиотеки была решена задача идентификации музыкальных произведений по фрагментам концертных исполнений. Для этой же задачи написан Telegram-бот (ссылка на бота).

## 3.1 Технологии

В библиотеке hpfw используются:

— C++17
— essentia – для вычисления спектрограмм
— Eigen3 – для линейной алгебры
— cpp-taskflow – для распараллеливания индексации

Для библиотеки также написан Python-клиент.

## 3.2 Архитектура библиотеки

В центре библиотеки два класса – *Collector* (коллектор) и *HashprintHandle* (хеншпринт-хендл). Эти классы связаны паттерном «Стратегия». Хендлы предоставляют инструменты (функции) для вычисления хешпринтов. Коллекторы используют эти инструменты по своему усмотрению и занимаются непосредственно вычислением хешпринтов. Благодаря такой структуре, можно будет легко тестировать различные способы распараллеливания индексации.

## 3.3 Клиент

Пока что не очень ясно, где стоит проводить черту между библиотекой и клиентом. На данный момент клиент использует только класс *Collector*.

## 3.4 Telegram-бот

Пример работы:

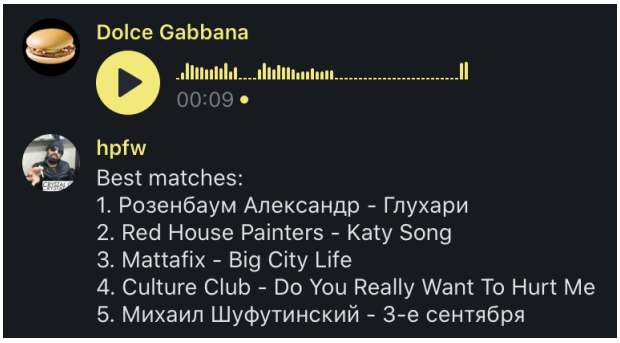


Рисунок 3.1

### 3.5 Результаты

Замеры проводились на Intel i5-6360U (4) @ 2.00GHz, 8GB RAM

— На индексацию одного трека уходит в среднем 5.7 секунд

— На полный поиск отрывка по базе из 167 треков (без индекса) уходит 4 миллисекунды

— Точность поиска 216 9-секундных отрывков по базе из 167 треков – 0.78.

— Одна спектрограмма занимает около 10 МБ (правда, их не всегда нужно хранить)

Стоит отметить, что поиск осуществлялся без знания исполнителя, то есть это оценка в худшем случае.

# Заключение

Текст заключения

## Список использованных источников

1. Tsai, T. (2016). Audio Hashprints: Theory & Application. (Doctoral dissertation, EECS Department, University of California, Berkeley).

2. Yan Ke, Derek Hoiem, Rahul Sukthankar. (2005). Computer Vision for Music Identification, Proceedings of Computer Vision and Pattern Recognition.

3. Leonid Boytsov and Bilegsaikhan Naidan (2013). Engineering Efficient and Effective Non-metric Space Library. In Similarity Search and Applications - 6th International Conference, SISAP 2013, A Coruña, Spain, October 2-4, 2013, Proceedings (pp. 280–293). Springer.

4. Bogdanov, D., Wack N., Gómez E., Gulati S., Herrera P., Mayor O., et al. (2013). ESSENTIA: an Audio Analysis Library for Music Information Retrieval. International Society for Music Information Retrieval Conference (ISMIR'13). 493-498.

5. Schörkhuber, C., Klapuri, A., Holighaus, N., & Dörfler, M. (n.d.). A Matlab Toolbox for Efficient Perfect Reconstruction Time-Frequency Transforms with Log-Frequency Resolution

6. Gael Guennebaud, Benoit Jacob, & others. (2010). Eigen v3.

7. T. Huang, C. Lin, G. Guo and M. Wong, "Cpp-Taskflow: Fast Task-Based Parallel Programming Using Modern C++,"2019 IEEE International Parallel and Distributed Processing Symposium (IPDPS), Rio de Janeiro, Brazil, 2019, pp. 974-983, doi: 10.1109/IPDPS.2019.00105.

# Приложение А  Первое Приложение