Evaluating the Potential of Similarity Estimation Using Hyperloglog for Query by Humming Problem

Arthur Lubambo

A Thesis Submitted in Partial Fulfilment

of the requirements for the

Degree of

Master of Science in Data Analytics

Diagram

Description automatically generated with medium confidence

February 2023

Supervisor: Kislay Raj

# Abstract

The ability to search songs by its melody, singing or humming is quite valuable when the user cannot remember the title, artist, or lyrics, to execute a traditional text-based search. This area is called Querying by Humming (QBH), it poses a significant challenge in Music Information Retrieval (MIR), requiring the identification of a musical piece based on a user's vocal rendition. It is not an easy task due multiple variations in the way the people sing, and the complex types of data such as audio signals. Given the multiple layers involved in the process, there are heavy algorithms to be executed while the person does a search. This project analysed the usage of Hyperloglog++ (HLL) algorithm into the QBH problem, aiming to find a quicker way to compute similarities between queries and songs. The framework studied extracts sets from songs in wave form combining music source separation model with a midi transcriptions model, with natural language techniques, creating sets representing the queries and songs. These sets are used to calculate the similarity between songs and queries. This project compared the potential of HLL algorithm in the similarity calculation comparing it with a baseline that uses sets operations. A detailed analysis was conducted with the trade-offs between memory usage and performance. The paper concludes by discussing limitations and avenues for future research, highlighting intrinsic issues in estimating intersections and similarities between sets using HLL of highly distinct cardinalities sets.

# Acknowledgments

To Do

# Introduction

## Query by Humming

The Query by Humming (QBH) problem is a significant challenge in the area of Music Information Retrieval (MIR), which consists of retrieving or identifying a piece of music based on a user's vocal singing or humming of a melody. This problem encapsulates the difficulty of transforming a non-verbal, subjective representation of a musical piece into a format suitable for comparison and retrieval within a computational framework.

QBH has an inherent complexity due to the variability of vocalizations reproduced by humans. Different from text-based queries, where keywords usually have good matching with the corpus of documents, the humming or vocals do not have by default a standardized notation or representation that is a compatible directly with the songs in the database, to identify and retrieve the musical piece searched in the database. The natural variations of pitch, rhythm, tempo, and timbre reproduced by users make it particularly a hard problem.

This problem was originally studied by (Asif Ghias et al., 1995) who introduced a pioneering approach that utilised the UP, DOWN, and SAME notation (for example, SUDUDDSUDUD) for representing the pitch movements of hummed melodies. This encoding was associated with an algorithm for string matching that tolerates mismatches to develop the core of an engine that retrieves a melody MIDI stored in the database that matches best with the humming.

The literature review will cover methods studied by other researchers in the area using a wide range of strategies such as dynamic time warping (DTW), Locality Sensitive Hashing (LSH), and others. There are different contexts and variants of this problem where each method has its own strength or weakness. A common bottleneck between these techniques is the computational resources involved in executing the information retrieval, in time and memory complexity.

Under these circumstances this project studied the potential of an alternative framework, using Hyperloglog++ Data Structure in a method to calculate the similarity between query and songs for QBH problem. The algorithm proposed is based in the representation of each query and melody as a set, then calculation of the overlapping coefficient (equivalent to inclusion coefficient of the songs in the query set), using it as a similarity measurement.

Eq. Overlapping Coefficient between two Sets

Applying the [Eq. Overlapping Coefficient between two Sets] to on each set corresponding to a query and a song, the songs can be ranked based on the overlap score for each query.

The underlying concept of similarity calculation involves utilizing the inclusion-exclusion principle [Eq inclusion-exclusion], to establish a framework for calculating the similarity between the queries and songs. Instead of using the tradition sets operations to calculate the overlapping coefficient, it is to use Hyperloglog++ operations (merge and cardinality estimation) to approximate the result, benefiting from the good characteristics of this data structure such as low memory usage. Associating each Query to a Hyperloglog++ instance (HLL) and each Song in the Database to another HLL, then computing the similarity by an overlap coefficient, using the estimation of the intersection size, using HLL operations.

Eq. Inclusion–exclusion principle for two sets A and B [Eq inclusion-exclusion],

It was proposed a discrete musical features extraction framework to create the original sets to be used as baseline for comparison with the Hyperloglog++. This features extraction has multiples steps, from music source separation to extract the vocals for songs, then midi transcription, sequences encodings, combined with n-grams to create a discrete set with musical information representing the queries and songs in a comparable format.

## Research Objectives

This project assessed the potential of using Hyperloglog++ in the QBH problem. Analysing if it would effectively drive improvements in memory usage without compromising the performance.

### Develop a Hyperloglog-based Similarity Estimation Framework for Query by Humming

It consists of creating sets of encodings for queries and songs then using the overlapping index estimation based on Hyperloglog++ (HLL) data structure, developing a framework for HLL parameters tuning for query by humming problem.

### Investigate the relationship between memory usage vs performance in Hyperloglog++ based Similarity Estimation

it includes running experiments to discover the impact of memory reduction and performance, establishing a modelling for the relationship between the hyper-parameters and accuracy. Analysing the hyper-parameters impact on memory usage. Finetuning it will bring the key value for the proposed method and its benefits.

### Compare and contrast Hyperloglog++ based Similarity Estimation Framework with a baseline algorithms performance by metrics.

it includes experimentation and comparison about the weakness and strongness of each type of algorithm used in the experiments under multiple aspects beyond memory usage, and performance, but also analyses in what musical context they tend to work better or not.

## Hypotheses

**Hypothesis 1:** The memory usage of HLL-based algorithms is smaller than the baseline set-based solution.

**Hypothesis 2:** The Mean Reciprocal Ranking (MRR) of HLL-based algorithms is equal to the MRR from the equivalent exact set-based solution.

## Scope and Limitations

The project focused on the QBH problem, so the algorithm developed is not appropriate for plagiarism detection. The dataset and techniques developed and studied are suitable for the Western music context (12 notes system), so it is not adequate to use it for microtonal music pitch less music, or other types of music. The proposed algorithm requires a discrete musical notation to fit in a set modelling.

The project applied different variants of algorithms with these characteristics to analyse the HLL estimation version versus the exact Set version cardinality computation. As the focus of this project is assess the impact of HLL, the scope was delimited to only vocals songs.

## Report Overview

The project is structured in the following sections: Introduction; Literature Review, where the key concepts and studies in the area is reviewed; Data Collection, analysing the data source and its characteristics; Methodology, which the framework, and data processing pipeline are explained in more details including the design of experiments and evaluation are discussed; In the Evaluation section a deep dive in the of explanations, of results and data characteristics are covered, supporting the outcomes of this project; The Conclusion contains a summary about the results and the scope which it is involved, finishing with recommendation for future works section, indicating possible areas and gaps to be to be explored based on the key findings of this project.

# Literature Review

## Query By Humming

Query by Humming (QBH) is an area of Music Information Retrieval that studies the content-based search in the song database. Humming serves as a natural and efficient method for searching a musical audio database by vocalizing the melody of a song. It was originally studied by (Asif Ghias et al., 1995), using the pitch movements “Up”,” Down”, and “Same”, to encode the query, and substring searching in a database of 183 songs was conducted using parameters to accept a controlled level of mismatch. The big challenge at that time was encoding the audio signal in a string (‘U’, ‘D’, ‘S’), due to computational power and algorithms available, taking up to several minutes to process 10 seconds of audio. In the time this study was conducted the computer power were very reduced compared to what can be found in a simple laptop in 2024. This Research opens the area of QBH it was quite interesting how a simple method using a UDS encoding could produce good results. It is important to highlight this initial study were based in a song database in MIDI format.

Later, other approaches were introduced to this problem such as dynamic time warping (DTW) a technique used in Times Series, but it could be modelled specifically for the audio processing context. For example (Fu et al., 2007) used the pitch level over time to model it as a times series using the dynamic programming technique DTW to calculate the matching level. Even with the optimizations, it has a quadratic cost to calculate the match between two series where n is the length of the series.

(Tripathy et al., 2009) they followed a very similar approach to the original (Asif Ghias et al., 1995) it created a layer for Wav to MIDI from the query and then used an algorithm for string matching, but they used a different method using dynamic programming to calculate the edition distance between strings.

In general, all the methods try different approaches to deal with the mismatching between what the song is, with the humming sung by a human. It can be observed that it is not only a time alignment between the query and the song. It is also a pitch alignment problem as was highlighted by (Stasiak, 2012), who proposed a method inspired by auto-adaptive human behaviour for ignoring errors in sung melodies.

All the algorithms were focused on the performance of the method so far, but there is another fundamental aspect to enable the implementation of it in practical applications, the scalability. It is a natural question, to understand how to apply it in a large database. (Guo et al., 2013) Introduced the application of Locality Sensitive Hashing (LSH), a technique that creates an index based on hashes operations considering the similarity between the content. They had applied this method developing other layers to tackle the problem of key transposing, it is when the music and the query are not in the same tone. So it was more scalable and improved the performance with better mean reciprocal rank (MRR).

(Alexios Kotsifakos et al., 2011) proposed a subsequence matching framework capable of dealing with gaps either in the query or the target song, performing better than the other Dynamic Programming methods, and maintaining the same time complexity.

Considering the diversity of queries, (Wang and Jyh-Shing Roger Jang, 2015) started to extract other type information from the queries that were not being used before, the lyrics. It is quite common besides humming the people to sing part of the song, or even whistle. So this study explored how the usage of lyrics in addition to the melody might impact the performance of QBH. They have combined speech recognition techniques to extract the lyrics of the humming combining it with the melody distance, reducing the error rates significantly. For obvious reasons, it would just impact music and queries with lyrics. The other challenge of this method is the language context, to make it generalizable it is necessary to have multiple languages trained, and lyrics structured for all the songs. So it implies extra effort in its implementation.

A common problem with the songs is the complex extraction of the melody in an automated way. The original studies were based on existing MIDI databases with the songs already in the symbolic format, annotated by humans. Producing it algorithmically through raw audio format is more complex, the song usually the song has multiple instruments, with harmony and a combination of more than one melody at the same time. With this challenge in mind, (Alfaro-Paredes, Alfaro-Carrasco and Ugarte, 2021) used a voice separation to improve the melody extraction from the songs, it demonstrated better results for the encoded melody for the song, consequently improving the matching algorithms.

Most of the studies were trying to use improvements in the string match algorithm to consequently have better results with the QBH, but (Velankar and Parag, 2018) changed the paradigm, introducing the matching using n-grams and inverted index.

They combined it with the Mean Normalized Frequency (MNF) Algorithm and developed a method called “Unified Algorithm for Melodic Music Similarity”. The big advantage of this method is that it tackles the information retrieval reducing the search space for the songs where the n-Grams matches. So, the pre-computed indexes help the reduction of the query time. The N-Gram contains each segment of intervals for the song, so even with imperfections on pieces of the query encoding is unlikely to compromise the whole result.

As a counterpoint to (Velankar and Parag, 2018) the work from (Ulfi and Mandala, 2022) highlighted that the “Unified Algorithm” has issues with performance and works slow for big datasets. They also implement the algorithm “Query by Humming System using Frequency-Temporal Attention Network” but apply enhancements to the Partial Matching of queries on it. In the end, it concludes the method has problems with scaling for big datasets. It might indicate a gap to be explored by applying adjustments in the algorithms for scalability efficiency.

In General, It is possible to be seen this problem can be tackled with different materials: some researchers focused in the encoding and matching algorithm from the query and songs using symbolic MIDI database already prepared, while others managed to encapsulate as well the songs manipulation to extract the melodies from the raw signal format. It is a harder problem to cover both layers. Because working from an annotated dataset the musical information has been already reviewed and cleaned manually by humans, is much more trustable without noises, or nuances that the automated transcriptions might result. But it is not in all applications it is a viable option once it is costly task to describe the data manually, mainly for big datasets.

It is possible to classify the types of musical encoding used to calculate the similarity in two classes ‘Discrete’ and ‘Continuous’. The Discrete transform the waveform into a sequence of symbols/discrete numbers, for instance Up Down Same, or Rounded Pitch equivalents. While the Continuous is the sequence of continuous numbers such as wave, frequencies, spectrogram, chromagram, raw pitch. There are also researchers that combined the traditional formats with textual data such as lyrics, and textual transcription of the queries to enrich the query song matching.

A very important aspect of this problem is the scalability of the methods involved dealing with big datasets, as it can be seen in this literature review it was covered by a few of the researchers but it still requires further investigation. Since the beginning it is a practical problem, because initially even with considered ‘small’ datasets today, in the past the computing power and memory were quite restrictive. Today with better resources the problem is the volume of data in practical applications. So, the efficiency is a very relevant aspect from QBH.

### Melody Extraction

The melody extraction or Transcription is the name of a task that transforms the musical audio signal in a temporal pitch notation with the sequence of notes. The Melody extraction on its own is already a sophisticated problem to be covered, when combined to query by humming it is an essential step for the QBH algorithms that deals with the raw format songs databases.

Traditionally, the methods used for these tasks are algorithmically based such as (de Cheveigné and Kawahara, 2002) used to fundamental frequency (F0) transcription. This approach might be good enough to get the transcription of a single instrument or singer recording, but when there are records with mixed sources making harmony, or multiple melodical lines, it gets more complicated to be covered with this approach.

The most recent application of Machine Learning techniques on it shifts the usage from fully algorithmic solutions to data-driven models. For instance, (Yu et al., 2021) used a method inspired by the human perception of frequency, time and intensity for audio, applying Convolutional Neural Networks to extract the melody. Or (Donahue, Thickstun and Liang, 2022) use generative models to create the synthetic dataset and combine it with annotated datasets to improve the melody transcription. Or (Bittner et al., 2022) who proposed a lightweight neural network for musical transcription, it is compatible with polyphonic outputs and could be applied to a wide range of instruments and vocals.  
We could see some of the methods, explicitly or implicitly used source separation for better results in the QBH, It is especially important as a pre-processing layer for the melody extraction phase. (Défossez, 2022) used a combination of Frequency Domain with Time Domain, introducing the method “Hybrid Spectrogram and Waveform Source Separation”, it is based on deep neural networks architecture.

The advantage under-explored from Source Separation for QBH is the ability to encode more than one simultaneous melody. It might improve the multi-vocal, counterpoint songs allowing the matching with the query to be on any of the melodic lines.

## Music Theory

Music theory is the academic discipline that explores the principles behind the composition, harmony, rhythm, and structure of music. It provides a framework for analysing and understanding the intricate relationships between musical elements, aiding both composers and performers in their artistic development. It is not a rigid set of rules to be followed instead, it is a dynamic framework that fosters creativity and facilitates deeper comprehension and communication of musical expression.

### Rhythm

According to (Schmidt-Jones, 2013) rhythm is a foundational element of music, emphasizing its role in temporal organization and providing structure to musical compositions. Alongside melody, harmony, timbre, and texture, rhythm forms the cornerstone of musical expression, with its placement of sounds over time being essential for music's unfolding and coherence. While melody and harmony often dominate discussions in music theory, (Schmidt-Jones, 2013) suggests that rhythm is equally indispensable, if not more so, given its intrinsic relationship with time. The hierarchical organization of rhythm, including concepts such as beat, meter, duration, and time signature, elucidates how rhythm is structured and perceived, offering frameworks for analysis and composition.

### Timbre

(Schmidt-Jones, 2013) elucidates the concept of timbre, also known as colour in music, as a fundamental element distinct from pitch, dynamics, and duration. Timbre encompasses the unique qualities of a musical sound that differentiate it from others, even when sharing identical pitch, duration, and volume. This distinction come from the complex waveforms produced by musical instruments, containing multiple frequencies that contribute to the perceived colour of the sound. The brain is capable to identify the pitch but it is also able to differentiate the mixture of other frequencies in the signal. The balance and interaction of these frequencies, particularly evident in the initial attack of a note, determine the distinctive timbre of each instrument or voice. Moreover, the text emphasizes the discerning ability of the human ear and brain to perceive subtle variations in timbre, enabling differentiation not only between instrument types but also between specific instruments or performers.

### Melody

(Schmidt-Jones, 2013) highlight that the melody consist of a sequence of notes with distinct pitch and duration that collectively form a cohesive musical line. Unlike mere successions of notes, a melody consists of those notes that prominently engage the listener's attention, delineating the primary musical theme. Additionally, the author introduces terminology pertinent to discussions of melody, such as the melodic line representing the core sequence of notes and ornaments or embellishments, which enrich the melodic texture without altering its essential structure. the book highlights the nuanced aspects of melody, and its role as a central and captivating element within musical compositions. In the context of QBH in this project it is possible to see challenges around different ways of singing with less or more ornaments. It will be covered in next chapters more details about it.

### Melodic Motion

The concept of melodic motion is explained in (Schmidt-Jones, 2013), delineating two primary forms: conjunct and disjunct. Conjunct motion characterizes a melody that ascends and descends gradually, with small pitch changes between successive notes, akin to step-wise or scalar motion. In contrast, disjunct motion describes a melody marked by rapid rises and falls, featuring large intervals between consecutive notes, often referred to as "leaps." The text also acknowledges that many melodies exhibit a blend of conjunct and disjunct motion, highlighting the dynamic interplay between these contrasting forms within musical compositions. The melody motion is the core concept used in the algorithms experimented in this project, due the fact that same melody in different pitch has the same relative motion. The humans can recognize as same melody two melodic lines with same motions but starting in different notes. It is an important characteristic in the melodical encoding explored in the next chapters.

### Harmony

In music harmony happens when multiple notes are executed simultaneously. It is important highlight that harmony is not necessarily "harmonious", depending on the intention of the composer it might be dissonant. The definition itself only refers to simultaneous notes. In this project it will not be explored to much once the focus is the melody and its movements.

### Counterpoint

The counterpoint is when there are more than one melody played at the same time, making a particular type of harmony. The counterpoint might be executed by same or different instruments or voices. It made a particular strong impact in this project, it will be discussed in future chapters the challenges around songs with counterpoint. How to distinguish and compute what are each independent melody that is played simultaneously.

### Texture

(Schmidt-Jones, 2013) explains musical texture referring to the density and complexity of elements within a piece of music at any given moment. Texture can be characterized as thick or thin, reflecting the presence of many or few layers of musical material. Various configurations contribute to texture, such as rhythm alone, a melody with chordal accompaniment, or multiple interlaced melodies. Understanding these terms can helps the musicians or audience appreciate deeper the musical structure. For this project this concept is useful to understand and give vocabulary to analyse the types of songs and how they perform with each type of algorithm and why.

### Types of Textures

**Monophonic:** is characterized by the presence of a single melodic line without of any accompanying harmony or counterpoint. While rhythmic accompaniment may exist, it does not detract from the singular focus of the melodic line, which consists of distinct pitches. This simplicity of texture allows for a clear and unobstructed presentation of the melodic material, making monophonic music a historically significant and aesthetically distinct form of musical expression.

**Homophonic:** is characterized by a single clearly melodic line that naturally captures the listener's attention, while other parts provide accompaniment or harmonies. Informally, references to chords, accompaniment, or harmony are often associated with homophonic compositions. These accompanying parts, while possessing their own melodic qualities, are distinguishable from the main melody by either sharing the same rhythm or serving primarily to fill in the chords or harmonies.

**Polyphonic:** is the type of texture characterized by multiple melodic lines at the same time. It is also called polyphony, counterpoint, or contrapuntal music. It can be executed by multiple instruments or vocals mixing independent melodies, that together creates a musical experience.

**Heterophonic: “**A heterophonic texture is rare in Western music. In heterophony, there is only one melody, but different variations of it are being sung or played at the same time.”, (Schmidt-Jones, 2013), It will not be covered in this project as the introduction already highlighted, the scope of this project is the Western Music.

In the book (Schmidt-Jones, 2013) there are references for examples of song from each texture type. Note that it is common the songs has multiple types of textures in the same song in different parts, so the examples are just guidance.

## Music Source Separation

Music Source separation is an area in the MIR that tackles the problem of separate mixed musical audio into individual audio sources. For instance, in this project it was used the Demucs a cutting-edge music source separation model, leveraging advanced deep learning techniques, designed isolate vocals, drums, bass, and other instruments with remarkable precision. Its underlying architecture allows it to analyse and decompose complex audio signals, providing high-quality source separation results. By processing audio data through neural networks, Demucs effectively learns the intricate patterns and characteristics of different sound sources, enabling it to separate them accurately. Its architecture is based on Transformers with cross-domain attention mechanism, processing the sound signal with time and frequency domain at the simultaneously, using 5 layers of encoders and 5 layers of decoders, then the last layer applies a transformation between frequency domain part into waveform that is combined with the output from the temporal part and then the prediction is computed.

. [TODO Demucs <https://github.com/adefossez/demucs?tab=readme-ov-file>

]

## MIDI

The Musical Instrument Digital Interface (MIDI) format is a standard protocol for the transmission and manipulation of musical data. Originally conceived in the early 1980s, MIDI revolutionized the landscape of music technology by standardizing communication between electronic musical instruments and computing devices. Its hierarchical structure encompasses a lot of parameters, including note-on/off messages, pitch, velocity, and control changes, facilitating precise control and expression across a diverse array of MIDI-compatible devices. Furthermore, MIDI's non-audio data format renders it inherently lightweight and versatile, enabling seamless integration into digital audio workstations (DAWs), synthesizers, samplers, and other musical peripherals. It is widely used as a format for musical composition, and edition using software dedicated to manipulating this digital transcription. The MIDI itself do not the any audio, instead it contains the events with musical meta-data, that can be executed by a player to output the audio. Specifically in this project it was use a python library to manipulate midi called Pretty\_MIDI [Add reference] introducing a different abstraction above the MIDI format, making a hierarchy between the notes and instruments.

[Add Reference for MIDI https://colinraffel.com/publications/ismir2014intuitive.pdf]

### MIDI automated transcription

Automatic Music Transcription (AMT) is the process of converting audio recordings of music into symbolic representations, such as musical notation or MIDI files, using computational algorithms.

In this project it was used the Basic Pitch [https://basicpitch.spotify.com/about], a lightweight neural network that can transcribe a wide range of musical instruments including vocals, supporting polyphonic outputs. It uses a deep learning architecture with convolutional layers. It has in the after the input layey the Harmonic CQT (HCQT) transformation, that is an efficient way to retrieve the harmonic information about the signal, suitable for musical frequencies. The architecture transforms the input into three posteriorgram, capturing the onset start of each note, the notes quantized, and the pitch dynamic (capturing bending, glissando or vibrato for instance).

## Counting Distinct Elements Cardinality of Sets

Cardinality Calculation or Counting distinct elements is a way to measure the size of a set. In the practical applications it is common to extract a set or the cardinality of a set from a stream data. It means that the streaming data might have repetitions. This a well know problem and the exact solution usually is tackled applying HashSet, that combines hashes with linked listed to tackle hash collisions, requiring linear memory in certain conditions. However, in practical applications with large datasets, it starts to be impractical to use this method.

### Probabilistic Cardinality Estimators

Given the impracticality of the exact count of distinct elements in a stream for cases that deals with a big dataset, it was studied alternative methods to estimate an approximation for the cardinality using less memory than the exact solution.

### Linear Count

The Linear Counting (LC) algorithm was one of the firsts methods to solve the distinct counting problem in linear runtime and constant memory space. In the same article [https://www.sciencedirect.com/science/article/abs/pii/0306437987900147] they proposed three methods for this problem, the first and simplest one was the LC. The main principle behind this algorithm is the application of a hash function to similar to a hash set, but instead of storing a linked list, it just saves a binary value indicating if the hash already appeared there. each bitmap position is an associated to a hash value. In case there was no collisions of hashes, the number of positions set as 1 in the bit map would match with the estimation, but collisions happen. So, they calculate an estimation of collisions based in the total of empty positions in the bitmap, and the size of the bitmap in order to adjust the distinct count approximation.

### Probabilistic Count

The probabilistic count (PC) proposed in [https://algo.inria.fr/flajolet/Publications/src/FlMa85.pdf] have done a different approach, from LC. It used the statistical properties of the hash distributions to estimate the count of distinct elements. It is based in a hash function that provides a good distribution of numbers (sufficiently uniform). For example, they explored the number of leading zeros as a property to be observed. The pattern has probability of happening. The estimation is based in the inverse of this probability. After applying it in practical, the authors realize the assumption about the uniform distribution is not perfect accurate, so they applied a correction factor to the estimation. However, this algorithm still susceptible to

### Loglog

Even with the correction factor, there are a high amount of variability. Depending on the case, even with small number of elements it might reach a sequence of zero. Adding a big error in the estimation. [https://link.springer.com/chapter/10.1007/978-3-540-39658-1\_55] Adjust the original idea of PC to use multiple estimators and calculate the mean between them to get the final number. So, they used the first k bits from the hashes to identify what estimator would be update, and the rest of the bits to estimate the leading zeros. It significantly reduces the variance of the estimations. In addition to it, it enables estimations for numbers different from powers of 2, allowing a more smooth and granular estimation. It might have issues, with small datasets, where not all buckets (estimators) have at least a value filled.

### Hyperloglog

The Hyperloglog (HLL) [https://algo.inria.fr/flajolet/Publications/FlFuGaMe07.pdf] is the evolution of loglog with further improvements. It replaces the arithmetic mean to harmonic mean, because experimentally it was evaluated it works better and it is less sensitive to outliers. Another improvement was to divide the problem in three ranges to tackle the calculation in a more accurate way. For smaller sets, it is applied a small set correction and big set correction. This type of correction supported significant improvements.

### Hyperloglog++

In [https://dl.acm.org/doi/10.1145/2452376.2452456] the authors applied further improvements in the HLL algorithm, reducing even more the memory usage and the accuracy. Including the usage of a 64-Bit Hash Function, bias correction adjustments and in the memory usage for sparse representation.

### Probabilistic Algorithms and Data Structures Applied

In the past 80-90s the power computing was very limited (low memory, low CPU power, disk latency), so the researchers started to create and develop new techniques to deal with datasets much bigger than the memory, as mentioned before. Nowadays, the power computing is much more advanced, but the amount of data exploded exponentially. So even with a much more powerful resources, the algorithms and techniques for distributed computing, are still and more relevant.

In QBH a big challenge is to enable more accurate results, with algorithms that scale well for big datasets. For instance, Spotify Dataset has more than 100 Million Tracks (Spotify, 2023). It is necessary to have more efficient methods for QBH. This Project will explore the potential of HLL++ in the QBH Task.

As mentioned previously one of the most impactful data structures created was the HyperLogLog. (Flajolet et al., 2007) and its evolutions HLL++ making it possible to estimate the size of sets beyond 10^9 with a standard error of 2% while using a memory of only 1.5 kilobytes. The application of this Data Structure in Big Data was disruptive, making it possible the count distinct elements estimation in a distributed system extremely quickly. (Heule, Nunkesser and Hall, 2013) made improvements to the method increasing the accuracy and reducing even more memory usage.

There are a few studies applying other types of probabilistic data structure for QBH As mentioned previously, there were some methods that used locality-sensitive hashing LSM (Guo et al., 2013) (Matti Ryynänen and Anssi Klapuri, 2008) and MinHash to index the songs for QBH. Although there are improvements, in the MinHash algorithm called HyperMinHash (Yun William Yu and Weber, 2020), there is a good opportunity to investigate if this new implementation would impact the performance of the QBH.

In this project, one of the applications of HLL could be inclusion coefficient estimation as (Nazi et al., 2018) demonstrate in an efficient method applied in database columns schema analysis. This coefficient could be defined as the fraction of the intersection of two sets relative to the smaller set. This concept can be used as an inspiration for another context. For example in the QBH, the matching between the query and the song can be modelled as the inclusion coefficient index between the melodic encodings of the query and song, using it as a set of keys. So the challenge is to transform into how to encode it with relevant information for matching it. In addition to it, the HyperLogLog methods could be used to optimize the execution, creating a scalable implementation for it.

### Cardinality Estimation

The cardinality is the measurement of the size of a set, in other words, it refers to the number of distinct elements in a set or a population. The computation of cardinality is relevant in multiple domains. Depending on the context it might be impractical to calculate the exact solution, due to the size of the dataset, or the nature of the distributed data across multiple nodes. Under this scenario, it was developed solutions that give up an exact result for controlled approximated estimation, using probabilistic data structures getting significant efficiency gains in memory and time complexity to execute the estimation. The literature review covers in more detail the different algorithms for cardinality estimation. It was chosen the Hyperloglog++ as an algorithm in this project due to its superior accuracy, efficiency, and versatility. Its advanced techniques and optimizations make it a preferred choice for many applications requiring fast and accurate estimation of set cardinalities, particularly in the context of large-scale datasets and memory-constrained environments.

## Similarity Measures

TO DO

## Mean Reciprocal Ranking

Reciprocal ranking (RR) is a method used in information retrieval and machine learning to evaluate the effectiveness of ranking algorithms in presenting relevant results to users. It assesses the quality of rankings by considering the position of relevant items within the ranked list. In reciprocal ranking, the relevance of items is typically represented as binary values (relevant or non-relevant), and the reciprocal of the rank of the first relevant item is computed. This reciprocal value provides a measure of the effectiveness of the ranking algorithm, with higher reciprocal ranks indicating more relevant items appearing higher in the ranked list. Reciprocal ranking is commonly used in evaluating search engines, recommendation systems, and other applications where the goal is to present the most relevant content to users. The reciprocal Ranking is simply the reverse of the ranking index, considering the ranking starting in 1 instead of 0. For instance if the first relevant document for a query is in the ranking 1 it would be equivalent of 1 in RR. In Case it was at ranking 2 it would reflect into 0.5 (1/2) as RR.

Note on this project there is only 1 relevant document in the database per query, as there is no multiple versions of same song in the database.

When aggregated over multiple queries, this metric is referred to as the Mean Reciprocal Rank (MRR). [ADD equation for MRR]

[Add reference https://link.springer.com/referenceworkentry/10.1007/978-0-387-39940-9\_488]

# Dataset

The dataset for QBH in general might be a challenging to be produce. It is necessary to be done with a carefully designed procedure to avoid insertion of bias. It has been found (Salamon, Serrà and Gómez., 2012) a database with 118 recordings of sung melodies, used in multiples other studies. It was built from 17 subjects, keeping a good gender balance and a wide level of music knowledge level, from zero to amateur musicians. They were presented with a written list of songs, and they were free to pick the ones they knew and were asked to sing/humming any part of the melody for recording. There was no restriction on time, or what part of the melody must be sung, they were free to sing with or without lyrics. They did not listen to the original song before recording it, what it quite important to avoid pitch bias, they could sing in any tone, so they just reproduce it by memory. All the records were made from a simple microphone from a laptop in order to simulate a realistic scenario for QBH. This dataset was chosen once it is robust and meets the good quality criteria and procedures on all the experimentation designs and it is widely used in other research. So, the sampling method and type to pick the dataset was respectively Non-probability sampling and judgment sampling.

It is important to highlight the QBH for copyrights reason, does not contain the songs database, however it is easy to build once it contains very popular songs. On the other hand, it might make the results less directly comparable, because each version of the same song might have nuances that makes the whole QBH tasks easier or harder. For instance, if a researcher uses an annotated MIDI database from songs, with very clean melodies as a starting point, it is much simpler, and likely to have matches than a dataset the melody extractions made by machine. The QBH task in the second case accumulates the challenges and imprecisions of each layer of the data processing.

The duration queries in the dataset used have recordings with a duration of 26.8 seconds in average, varying from 11 to 98 seconds. Due cost efficiency for this experiment, it was used the canonical songs database 481 songs, instead of the full collection with 2125 songs that contains different versions of the same song. Note it might negatively impact the performance, once we are relying in a single version of the song in the database. For instance, the version might have particularities noise or extra challenges that makes harder to extract the high-level features for the matching mechanism.

# Methodology

## Proposed primary research methodology

The primary research methodology employed in this study centred around experimental investigations. The goal as mentioned previously is to investigate if the usage of Hyperloglog++ in the similarity estimation using the inclusion-exclusion principle will bring enhancements in the algorithm's execution and will not affect negatively the query-by-humming performance in a significant way.

In many other areas applications this type of data structure is demonstrating significant improvements in cardinality estimations, bring leaps in the algorithm efficiency. The Literature Review demonstrated a lack of studies with the HyperLogLog++ applied to Query By Humming, so this research aims to explore this approach and evaluate the results. The methodological framework was structured across 3 phases: data preparation, experimentation, and evaluation. In the [RMD] it is possible to see the breakdown of what is involved on each stage.

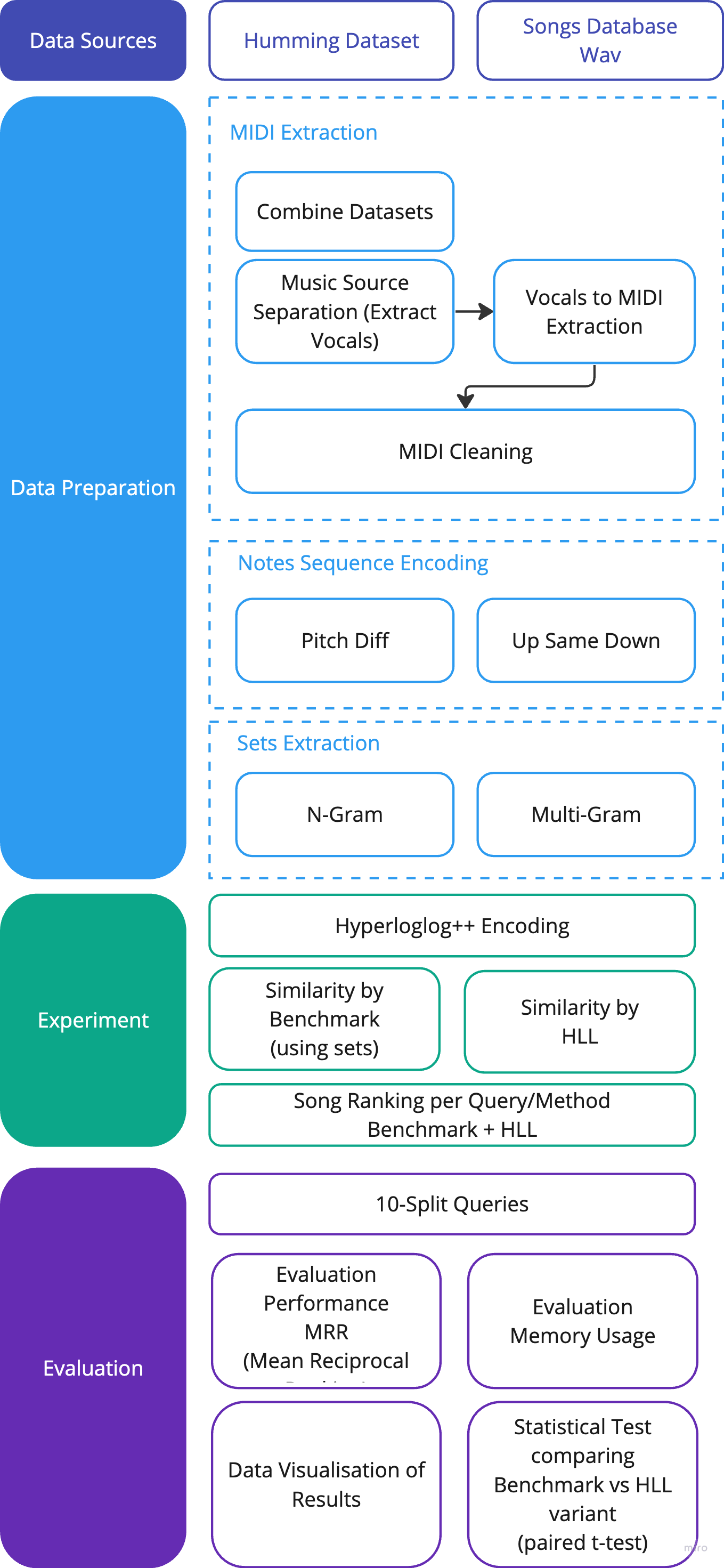


Fig .[RMD] Diagram with Research methodology

As Data Source It was used a suitable secondary datasets for the primary research of this capstone project. I was chosen reliable set data source [ref], with representativeness, and good references in other studies. It will be discussed in the next chapter more details about the data sources involved in this project.

The Data preparation phase, involved 3 main layers, MIDI extraction, Notes Sequence Encoding, then Sets Extraction. The data source was provided in WAV format, it is necessary process the audio signal to construct a symbolic representation of the melodies, for either queries and songs. In the next chapters covers details about each step on the MIDI extraction. The Notes Sequence Encoding is a Layer to convert the Notes into an format that is robust to variations in pitch, time, and duration between query and songs. Then the Sets Extraction transform the Sequences of Discrete Encoding into Sets where the similarity operations will be executed.

In the experimental phase Hyperloglog++ (HLL) data structures were created for each set variant query and song. Then then similarity is calculated for the baseline overlap coefficient using sets operations, followed by HLL based estimation using cardinality estimations and the inclusion exclusion principle, more details will be discussed about it in next sections. After calculating the similarity between each pair query-song, the ranking of the desired song is for each method and set combination.

In the evaluation phase, the queries were shuffled and split in partitions, with MRR (Mean Reciprocal Ranking) and memory calculated for each combination of method partition. Subsequently a comparison was made applying statistical test and data visualizations to assess the hypotheses mentioned previously in the introduction.

## Data Preparation

This layer contains the steps necessary to transform raw signal data from sound WAV format to Sets.  
In this project, all audios from query and songs were provided in raw format.

A diagram of a music flow

Description automatically generated

Fig. Data Flow

In the [Fig. Data Flow] it can be seen the overview of data processing flow layers, from wav format in query and songs, to music source separation for vocals extraction, then the midi transcription, with a layer to clean overlapping notes and other imprecisions such as outliers notes in the midi created automatically. Then each midi is converted into a sequence encoding between two options provided (PitchDiff and UpSameDown), for later a set be created applying n-grams or multi-grams in the sequences. Each layer in the data processing flow will be covered in the details in the following sections.

### Midi Extraction

#### Music Source Separation

From the raw Wav songs and queries files it was made different processing flows. The songs contains in the audio file the whole track, with mixed signal of multiple instruments (including harmonical, melodical, and percussion), voice(s), crowd. Extract relevant high-level features from this signal is not an easy task. In western popular music, in general the main melody is produced by vocals. So It is natural try to separate the mixed track into channels with different sources.

For that it was used the current state-of-art model Demucs based in Hybrid Transformers for Music Source Separation [https://arxiv.org/abs/2211.08553] as mentioned in Literature Review, it can separate the sources between: ‘vocals’, ‘drums’, ‘bass’, ’other’. The vocals includes any voice, including multi-vocal songs, the drums, include any percussive sound, bass contains the low frequency bass instruments, and other has any other background accompaniment. As the queries has only vocals, it was necessary to execute this layer in the queries recordings, but it was quite useful to run it in the songs database. A the focus of this project was to analyse the Hyperloglog++ potential on the QBH, the scope of this project was delimited to vocal songs. For future works it would be interesting to extract in addition melodies from the non-vocal parts of the song.

#### Vocals to MIDI Transcription

The Vocals to MIDI operation was executed for both types of recordings the query and the vocals of songs extracted in previous by the Demucs model. Then it was applied the BASIC [https://arxiv.org/abs/2203.09893] mentioned in the literature review to extract the MIDI transcription automatically using a deep learning model. It was chosen due the good quality results, lightweight to run, open source. See in [ADD REFERENCE for MIDI sample TO DO] links for sample of MIDI created from BASIC. The resultant MIDI file, contains a list of instruments, where each instrument has a list of notes. Each note has a start time, end time, and pitch. The pitch follows a numeration form MIDI note number see table in the Appendices. Note the BASIC is capable to extract polyphonic music, in case there is multi-voice songs the midi generated has notes that is played simultaneously.

### Notes Sequences Encoding

From the low level signal with the two layers below it was extracted the MIDI that is a symbolic high-level representation of the song. Although it is not enough to match directly the part of the song with the query with both MIDIs, because the subjects might sing or humming the song from a different start, beat or even in a different. In addition to it subject and singers might different interpretations of the song adding extra notes, ornaments. Given all these circumstances, based on the concept explored in the literature review melodic motion is an important feature to be extracted, it makes the encoding invariant between recordings in different keys, also transform the notes in sequences helps the notion of time and duration being relativized. What matters in this approach is the order and the melodic movement, instead the precise time/duration of each note.

#### Pitch Diff

For capturing the melodic movements, the sequence of notes was processed to get the difference between the pitch of the current and the previous note.

A diagram of a flowchart

Description automatically generated

Fig. Illustrative example of PitchDiff extraction

So, a sequence of N notes will be converted in a sequence of N-1 pitch differences. In the example illustrated in the [Fig. Illustrative example of PitchDiff extraction] produce a sequence (Δ1, Δ2, Δ3, Δ4, Δ5, Δ6, Δ7), where each is the difference of the pitch between the next note and the current note.

The reason why this feature is relevant resides in the fact of being invariant to pitch transformation, and time. Regardless of adding +x half-steps in all notes from the melody all the Pitch Diff will be preserved. It is also, suitable to extract any ‘cut’ of the song and being consistent. For instance, if the pitch diff were relative to the first note instead of previous, if the song variant starts from different all the consecutive pitch diff would have a different encoding.

#### Up Same Down

The pitch difference the exact melodic motion in a scale of half-steps precision. It might bring problems if the subjects recording has inaccurate relative pitches in the humming. One approach the original research (Asif Ghias et al., 1995) used the Up, Same, Down representation of the sequence.  
It would be equivalent to transform the pitch diff sequence into a transformation: ‘Up’ if diff greater than zero, ‘Same’ if diff is equal to zero, and if diff less than zero ‘Down’. I can be observed that it is equivalent of reducing the resolution of the information in the pitch encoding, capturing the same melodic movement but transforming granular integer information into a just three classes encoding.

### Sets Extraction

After having all the sequences from songs and queries based in the two methods (UpSameDown and PitchDiff), it is necessary to match them. The sequence themselves are not directly comparable. There are traditional methods such as dynamic time warping, that could be used for this type of problem, but it usually requires quadratic complexity cost to compare two strings. To overcome it, this framework proposed a set extraction to compute parts of the melodic movements, it removes completely the time dependence of each part of the song. Each element of the set corresponds to an encoding of the melodic movement. Using this approach each query and song will be associated to a set of melodic movements.

In this project it was tested two types of functions for M, the N-Gram and Multi-Gram. Transforming then in a set, the song set M(S) and query set M(Q) are now comparable, using the overlapping coefficient.

#### N-Gram

A natural way to extract the elements for the set creation is define a size for the melodic movements parts. In other words, a size of a substrings to be extracted from the original sentence. Inheriting the concepts from Natural Language Processing, it is equivalent to produce N-Grams from a sequence of tokens. Where each token is a PitchDiff or a UpSameDown from the Sequence Encoded in the previous step. Each N-gram Produced is a elements representing a part of the melody with its movements. In general, it is similar to the Bag of words approach, where each element is equivalent to a term in the search mechanism. See in the [Fig. Illustration of Sequence to Set Function Transformation.] a illustration explaining the transformation.

A diagram of a number

Description automatically generated

Fig. Illustration of Sequence to Set Function Transformation.

This method introduces a hyper parameter: N to be defined. How to fine tune it properly? If N is too small, the corpus of different terms would be very small, making it uncapable to retrieve the elements properly. For instance: N = 2 in the UpSameDown Encoding would create only 3^2 = 9 elements in the whole corpus ‘vocabulary’, making it harder to separate what right song in the database matching the query. On the other hand if it is too big, it makes less robust for subject mistakes, or ornaments, once it needed to get a perfect melodic encoding for longer, in addition the query set would have a limited set. For example: a query sequence with size k creates (k+1-N) elements based on N-Grams. There is a clear limit of N, it needs to be smaller or equal to k. In this extreme scenario, the query would have only 1 element in the set. This elements probably will not be enough to discriminate the songs that matches with it.

Analysing the query size, it was found that mean size of sequence is 33, minimum is 9, maximum 106. Considering this case, it was tested N=5, 8, 10, 12, even with N=10;12 that it is above the minimum k, it was tested because there is a chance of having a positive impact in the performance of the other bigger queries compromising the few ones that are too small.

#### Multi-Gram

An interesting strategy to overcome the finetuning of the N hyperparameter is to create a set that is a union of multiple N-Grams with different N values. This approach was named as Multi-Gram. In this case the hyperparameter is a pair (k,j) where j>=k, representing the union the N-Gram from N=k until N=j.

The Multi-Gram proposal helps increasing the cardinality of query set, towards a better discrimination power with the songs in the database.

## Experiment

The experiment consists calculate the similarity between queries and songs using the baseline method and the alternatives methods using HLL encoding, then comparing the results of the ranking. Each section will cover more details of each layer.

### Similarity

In the previous stages it was created an equivalent set for each song and query with melodical movements information encoded. In the similarity part, it was created a data frame with all possible pairs between queries and songs, from that the similarity measurement is calculated based on the similarity of the respective queries and songs sets involved. The rationale behind it is if the melodic information between sets are similar the query and song has a good match, if they are not similar, probably it is a non-match.

There are several methods to calculate similarity between sets such as Sørensen–Dice coefficient; Jaccard index, overlapping coefficient and others. In the proposed framework it was chosen the overlapping coefficient, also called Inclusion coefficient (\*considering the query as a target set), because the other metrics has problems with the difference in size. For instance, the Jaccard index, because the queries sets are much smaller than the song set. In the table [Table. Average Cardinality of Sets per Method] it can be seen the songs set sizes are in average more than 8 times the query set sizes.

Consequently, the Jaccard index would penalize bigger songs even if the whole query set is included in the song set. For this reason, it was chosen the overlapping coefficient, that would only consider the size of the query (\*the smaller set) in the denominator. Note the Sørensen–Dice coefficient has a similar problem.

### Similarity by overlapping coefficient

The overlapping coefficient, calculate the cardinality of the intersection between the query set and song set divided by the cardinality of the smaller set between query and song set (\*in practice the query is the smaller always). Analysing the Algorithmic complexity to make this calculation involves compute the intersection between sets and count the cardinality of the intersection, and the query set. For intersection calculation between A and B is the O(min(size(a),size(b))), while computing the size of a set is O(1). So the computation of overlapping coefficient between two given sets, requires O(size(query Set)) to be computed.

### Hyperloglog++ Encoding

As mentioned in the literature review the Hyperloglog++ is a data structure capable to estimate the count of distinct elements in list. It is equivalent to compute the estimation of a cardinality for the set. The idea behind this data structure is compromise the exact solution using a controlled error estimation but using less memory in the computation. As mentioned before, it has a hyper parameter: **p** it defines how registers will be used in the estimation (2^p). In this project it was tried multiple combinations of **p={4,8,12,16}** , as higher is p more accurate is the cardinality estimation, but it uses more memory.

### Similarity Estimation using Hyperloglog++

The HLL++ has the following functionalities: add an element to the sketch, merge two HLL, and estimate the cardinality. Note it is not possible to compute a HLL for the intersection given two HLL.

But in this project, it will be used principle of inclusion-exclusion equation to compute the intersection cardinality.[Eq Ref]

In this case the estimation of the cardinality of A intersection B the sum of the estimation of cardinality of A and B, subtracted by estimation of the union A,B (merged HLL).

So the ratio of cardinality of intersection and min(card(A), card(B)) provides the estimation for overlapping coefficient.

For estimating the cardinality of a given HLL is O(R) constant based in the number of registers, then to compute the merge between two HLL is O(R) where R is the number of registers. Consequently, the algorithm to compute the estimation of overlapping coefficient is O(R) in time.

In terms of memory, the HLL++ uses O(R) but there are some improvements for sparse sets. [Add reference]

### Song Ranking

After have the similarity calculated for all combination of pairs Query Songs for all methods including the combination between sequences types (PitchDiff, UpSameDown), the Sets Extraction Types (N-grams, Multigram), and similarity estimation method (Baseline, HLLs). For each query , method the respective ranking is computed, to evaluated.

## Evaluation Plan

The evaluation of the methods are based in the Mean Reciprocal Ranking (MRR), and Memory Usage. The queries were divided into shuffled 10 splits, so a statistical test were applied to compare the MRR between the samples. Additionally the memory size if the sets and HLL were measured and compared. All comparisons were paired between the baseline set based operation versus multiple variants of the HLL hyper parameters.

### Statistical Test

The statistical test applied was Paired Samples T-Test [Add Reference <https://link.springer.com/chapter/10.1007/978-94-6351-086-8_4>] , because for each query split we have equivalent pairs between baseline and a target hyperlolog++ variant. It can be assessed the performance of QBH of each split of query under the same circumstances using different methods.

## Tools and Technologies

It was used in the experiments a simple laptop MacBook Air, Apple M2, 8 GB of RAM. It was not required the usage of bigger cluster for heavier computational power to execute experiments. As it can be seen in the attachments in the git repository, it was used Jupyter Notebooks with the experiments scripts. In addition, There were open source libraries used in the experiments such as:

**Librosa**, for audio processing music feature extraction and manipulation; **Pandas**, for general data analytics tasks; **pretty midi** to manipulate midi data; **nltk** to compute NGrams; **seaborn** and **matplotlib** for data visualization; **Hyperloglog** it a python library with the Hyperloglog++ implementation; **HTDemucs** is a machine learning model for music source separation; **BASIC** a model for automated music transcription, it was used to create the MIDI from the wav.

For production usage of these algorithms, it would be necessary for bigger databases a distributed computing architecture such as Hadoop, Spark, or equivalent. The nature of the algorithms applied on the experiments are suitable for distributed computing, once it is implemented using the tools mentioned previously.

## Ethical and Risk Considerations

The ethical considerations for this project are based on Ethics and data protection (HAYES and KUYUMDZHIEVA, 2021). It is clear that the data involved in this research is NOT sensitive, as does NOT deal with data concerning children, or vulnerable people. The data used has a Creative Commons Attribution 4.0 International license. It is NOT invasive neither put at risk any kind of freedom of the participants. The source organization that collected the data, had the consent of the people to use the data for this type of research.

There is no sensitive Personal Identifiable Information (PII) such as Sex Orientation, Race, Religion, or similar. The data involved is already anonymised. Given the nature of the data involved, it is not necessary to implement of the DPIA process. The Data was collected in Europe and does not have any PII, so There is no Transfer of personal data to non-European countries at all.

It is important to highlight a possible risk of miss-usage of the algorithm developed in this research, such as plagiarism detection. It would be an unfair and unethical usage if it were implemented as an automated decision-making. Any decision made by this algorithm must be reviewed by a human. For example: In case someone uses the proposed similarity algorithm in an automated system for automated plagiarism detection, the responsible for operating it must ensure to have a human review for each positive detection, to consider this application ethical. The risk will mitigated by informing in the public repository of good practices of usage of this method.

Any software, or library used in this research will be following a proper license such as MIT, Apache License, BSD license or equivalent. In the case of proprietary software or cloud platform usage, the proper permission will be in place. Regarding the Sampling Method, The intrinsic bias involved will be managed by explicitly representativeness and relevance of data selection with clear criteria and justification. For each stage of the research will be reviewed the proper usage and reduction of bias using a proper technique from the data preparation, until data visualization and Results presentation.

After taking all these measures and in case any other ethical concern was not properly detected, any external person can report it through GitHub issue, and adequate measures will be put in place.

The context this project will be based on is Western music theory (12 pitches), so for example it would NOT be adequate to use it different types of music such ad microtonal music, pitch less or equivalent.

Applying the method developed on this project in real projects must be considered carefully once depending on the context it might exclude songs from other cultures to have accurate results. Due to the limitations of datasets and time constraints, it will be tackled with clear advice about the usage.

# Evaluation and Analysis

For the evaluation of the proposed Hyperloglog potential in the query by humming, the experiments were executed as mentioned in the Methodology section. It was used the MTG-QBH dataset and the songs database was gathered. Originally the database contained 481 songs and 118 queries. But after analysing the dataset, it was excluded from the collection the songs with non-vocal songs to respect the scope of the project. The resultant dataset contained 473 songs and 112 queries. After applying the split mentioned before, it was produced 10 splits with 2 of them with 12 queries, and 8 with 11 queries. Note that the song database was not split, so the whole 473 songs are searchable for all splits.

A diagram of a method

Description automatically generated

Fig. X Diagram Combination of Attributes for Methods

## Performance

Applying all the 50 methods to the 10 splits queries, result in 500 results. Analysing the MRR breakdown per Method and Similarity Type, applying the average, it is possible to the results in [TABLE RESULTS]. It highlights that even the baseline method just had acceptable results for the encoding for **pitchDiff-NGram-5** and **pitchDiff-MultiGram-5-10,** with respectively 0.142 and 0.136**.** For all the other types in baseline, the results got poor results (MRR < 0.04). Mainly the UpSameDown Sequence encoding, it is probably due to the excessive low granularity of information, for instance the UpSameDown-NGram-5 can allow in the maximum different ‘terms’ in the corpus, making it harder to discriminate the songs. It can be observed it has a slight improvement for higher number of N-Grams and Multi-Gram, but it still low. As the N number in N-Gram increase, the number of elements for each query is reduced as well. So in general the **UpSameDown** did not perform well.

A screenshot of a table

Description automatically generated

Table. [TABLE RESULTS] Results By SIMILARITY TYPE/SET TYPE [ADD PROPER TABLE]

A graph with different colored bars

Description automatically generated

Fig. X Mean of 10 Splits MRR per Method Type

For **PitchDiff** method the lower number of n-gram could make it work properly, because each item in the sequency has a wide range of possibilities as it can be seen in the histogram distribution [Hist PitchDiff Query & Song] these queries have the majority of the 94.5% of the pitches diff between -5 and 5, with -0.008 as average and 3.027 as standard deviation. So it provides bigger range of possibilities compared to the UpSameDown approach. But on the other hand, in case is has longer elements (bigger N parameter in N-Gram), it requires a good level of pitch precision for the subject’s humming or singing. Another interesting observation is the number of half-steps in the songs pitch diff, much wider, with a standard deviation of 7.649753 compared to the queries ones. Given the histogram shape, it probably indicates the presence of noise the melody transcription from the method. It indicates a point of investigation to be tackled for future works.

A graph of a column

Description automatically generated A graph of a line graph

Description automatically generated

Fig x [Hist PitchDiff Query & Song]

After applying the paired t-Test comparing each method using the Baseline Similarity versus HLL Similarity measurement. It was made a Hypotheses test for each of the 40 HLL based Methods compared to the respective 10 baseline method.

Let , and the baseline method of the method x

The Null Hypotheses is

And the Alternative Hypotheses is

Applying the significance level of 0.05; it could be obtained the following results in the Table. [t-test-table].

Analysing first the **pitch\_diff-ngram\_5** method it can be seen the only HLL variant that can have null hypotheses accepted is the HLL\_16. So, the performance is significantly equal to the baseline. On the other hand, for all the other lowers HLL with p= {4,8,12} has the rejected. It is expected lower values has worse performance due bigger estimation errors.

While the **pitch\_diff-multi\_gram\_5\_10** had all rejected. So, the usage of Hyperloglog impact significantly the performance for this type of sets.

Given the upfront low performance of the baseline similarity for the other methods, the analysis of the complements models needs to be made carefully. In this case the fact of accepting the means that the baseline similarity and the HLL similarity performed equally bad for these cases. It happened for all the **UpSameDown** experiments, while the only two experiments from **PitchDiff** that demonstrate rejecting the in results were **pitch\_diff-ngram\_8** and **pitch\_diff-ngram\_10**.

p\_valueset\_typesimilarity\_methodcol\_indexmean\_mrrmean\_baseline\_mrrreject\_nh30.567506pitch\_diff-ngram\_5hll\_16pitch\_diff-ngram\_5\_set\_hll\_16\_index0.1441330.141676False20.013417pitch\_diff-ngram\_5hll\_12pitch\_diff-ngram\_5\_set\_hll\_12\_index0.0657660.141676True00.002757pitch\_diff-ngram\_5hll\_4pitch\_diff-ngram\_5\_set\_hll\_4\_index0.0224310.141676True10.000936pitch\_diff-ngram\_5hll\_8pitch\_diff-ngram\_5\_set\_hll\_8\_index0.0113620.141676True350.002279pitch\_diff-multi\_gram\_5\_10hll\_16pitch\_diff-multi\_gram\_5\_10\_set\_hll\_16\_index0.0742470.134536True340.004979pitch\_diff-multi\_gram\_5\_10hll\_12pitch\_diff-multi\_gram\_5\_10\_set\_hll\_12\_index0.0312960.134536True320.000942pitch\_diff-multi\_gram\_5\_10hll\_4pitch\_diff-multi\_gram\_5\_10\_set\_hll\_4\_index0.0181560.134536True330.001272pitch\_diff-multi\_gram\_5\_10hll\_8pitch\_diff-multi\_gram\_5\_10\_set\_hll\_8\_index0.0162800.134536True100.399318pitch\_diff-ngram\_8hll\_12pitch\_diff-ngram\_8\_set\_hll\_12\_index0.0328980.036361False110.005777pitch\_diff-ngram\_8hll\_16pitch\_diff-ngram\_8\_set\_hll\_16\_index0.0241720.036361True80.154073pitch\_diff-ngram\_8hll\_4pitch\_diff-ngram\_8\_set\_hll\_4\_index0.0148290.036361False90.139559pitch\_diff-ngram\_8hll\_8pitch\_diff-ngram\_8\_set\_hll\_8\_index0.0137280.036361False260.602410pitch\_diff-ngram\_12hll\_12pitch\_diff-ngram\_12\_set\_hll\_12\_index0.0376350.035790False190.030041pitch\_diff-ngram\_10hll\_16pitch\_diff-ngram\_10\_set\_hll\_16\_index0.0293350.035790True270.345960pitch\_diff-ngram\_12hll\_16pitch\_diff-ngram\_12\_set\_hll\_16\_index0.0291460.035790False240.378210pitch\_diff-ngram\_12hll\_4pitch\_diff-ngram\_12\_set\_hll\_4\_index0.0279590.035790False160.345020pitch\_diff-ngram\_10hll\_4pitch\_diff-ngram\_10\_set\_hll\_4\_index0.0261300.035790False170.304414pitch\_diff-ngram\_10hll\_8pitch\_diff-ngram\_10\_set\_hll\_8\_index0.0231540.035790False180.064773pitch\_diff-ngram\_10hll\_12pitch\_diff-ngram\_10\_set\_hll\_12\_index0.0181790.035790False250.162416pitch\_diff-ngram\_12hll\_8pitch\_diff-ngram\_12\_set\_hll\_8\_index0.0146780.035790False390.273617UpSameDown-multi\_gram\_5\_10hll\_16UpSameDown-multi\_gram\_5\_10\_set\_hll\_16\_index0.0276230.028031False380.123509UpSameDown-multi\_gram\_5\_10hll\_12UpSameDown-multi\_gram\_5\_10\_set\_hll\_12\_index0.0256220.028031False360.150615UpSameDown-multi\_gram\_5\_10hll\_4UpSameDown-multi\_gram\_5\_10\_set\_hll\_4\_index0.0153160.028031False370.076738UpSameDown-multi\_gram\_5\_10hll\_8UpSameDown-multi\_gram\_5\_10\_set\_hll\_8\_index0.0147950.028031False230.337096UpSameDown-ngram\_10hll\_16UpSameDown-ngram\_10\_set\_hll\_16\_index0.0257030.026030False220.109873UpSameDown-ngram\_10hll\_12UpSameDown-ngram\_10\_set\_hll\_12\_index0.0192390.026030False210.094303UpSameDown-ngram\_10hll\_8UpSameDown-ngram\_10\_set\_hll\_8\_index0.0125020.026030False200.069541UpSameDown-ngram\_10hll\_4UpSameDown-ngram\_10\_set\_hll\_4\_index0.0107000.026030False140.801614UpSameDown-ngram\_8hll\_12UpSameDown-ngram\_8\_set\_hll\_12\_index0.0315350.025066False150.464767UpSameDown-ngram\_8hll\_16UpSameDown-ngram\_8\_set\_hll\_16\_index0.0249370.025066False120.310286UpSameDown-ngram\_8hll\_4UpSameDown-ngram\_8\_set\_hll\_4\_index0.0179320.025066False130.149486UpSameDown-ngram\_8hll\_8UpSameDown-ngram\_8\_set\_hll\_8\_index0.0141680.025066False300.886845UpSameDown-ngram\_12hll\_12UpSameDown-ngram\_12\_set\_hll\_12\_index0.0283250.024538False310.569100UpSameDown-ngram\_12hll\_16UpSameDown-ngram\_12\_set\_hll\_16\_index0.0249940.024538False290.210286UpSameDown-ngram\_12hll\_8UpSameDown-ngram\_12\_set\_hll\_8\_index0.0165550.024538False280.056052UpSameDown-ngram\_12hll\_4UpSameDown-ngram\_12\_set\_hll\_4\_index0.0097320.024538False60.945654UpSameDown-ngram\_5hll\_12UpSameDown-ngram\_5\_set\_hll\_12\_index0.0202020.013553False50.866142UpSameDown-ngram\_5hll\_8UpSameDown-ngram\_5\_set\_hll\_8\_index0.0188270.013553False70.953417UpSameDown-ngram\_5hll\_16UpSameDown-ngram\_5\_set\_hll\_16\_index0.0178810.013553False40.317311UpSameDown-ngram\_5hll\_4UpSameDown-ngram\_5\_set\_hll\_4\_index0.0124170.013553False

Table. [t-test-table].

The natural question after analysing it is: why only the HLL\_16 had equivalent performance in the best method? The reason behind it is the error rate from the Hyperloglog data structure.

The mean absolute error expected in the cardinality estimation in the HLL for p= 4, 8, 12, 16 are respectively 26%, 6,5%, 1,6% e 0,41%, based in the equation [error\_eq].

Add HLL error equation [error\_eq].

So, combining it with the principle of inclusion-exclusion equation, the intersection cardinality estimation [intersection eq]. has a natural cumulative error proportional to the size of the size of the bigger set the union estimation. In the cases the intersection is small, the relative error will be big compared to the intersection cardinality.

A graph of blue dots

Description automatically generated A graph with numbers and lines

Description automatically generated

A graph with a line

Description automatically generated A graph with a line

Description automatically generated

Fig. Scatter Plot Comparisson Union Size vs HLL estimation for union in pitch\_diff-ngram\_5\_set.

It can be seen in the union of the sets and the union estimation has a clear linear relationship, increasing the correlation as the HLL has more registers, what it is expected.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| HLL p | Union Mean Abs. Error | Union Abs. Error Std Deviation | Union Mean Relative Error % | Intersection Mean Abs. Error |
| 4 | 57.12 | 61.38 | 20.05 | 23.36 |
| 8 | 11.20 | 13.24 | 3.94 | 4.92 |
| 12 | 2.65 | 2.91 | 0.91 | 1.00 |
| 16 | 0.65 | 0.73 | 0.22 | 0.18 |

Table. Error metrics for union and intersection estimation using HLL for pitch\_diff-ngram\_5\_set

The scatterplot graphics [Fig] demonstrate a good correlation between the estimation of the union and the actual union size. The experimental relative error, demonstrate acceptable values mainly from the HLLs different from HLL4, all the values are under the expected error rate, estimated theoretically.

A graph with blue dots

Description automatically generatedA graph of a diagram

Description automatically generated with medium confidence

A graph with blue dots

Description automatically generatedA graph with blue dots

Description automatically generated

Fig. Intersection Size vs Estimation with HLL for pitch\_diff-ngram\_5\_set

Then analysing the intersection scatterplot, it is possible to note that the performance starts to be poor. The reason behind it is the order of magnitude of the union abs error and the intersection size. It can be visualised that as the intersection size is bigger the estimation error is smaller. Comparing the mean absolute error of the union and the intersection error they are under the same magnitude, given the intersection is much smaller than union the relative error of the intersection is huge compared to the union. It happens due the method of estimating the intersection size.

Eq. Estimation Error

As consequence from [Eq. Estimation Error], the magnitude of the error is similar to the combined error of other estimations. Also, when the intersection is bigger, the relative error tends to be reduced.

Note that the accuracy of the similarity estimator is a consequence of the intersection, so if the intersection error is high the similarity error is also expected to be high.

A graph with blue dots

Description automatically generated A graph with blue dots

Description automatically generated

A graph with blue dots

Description automatically generated A graph with blue dots

Description automatically generated

Fig. Similarity vs Estimation with HLL for pitch\_diff-ngram\_5\_set

I can be observed that in all of the graphs comparing with the intersection the similarity got highly related to the Intersection. Also, it can be seen that even with the points scattered for lower similarities/intersections, the QBH problem might be still have a good performance, once the target song tends to be in the high similarity range, that would bring more accuracy to the estimation. Analysing the HLL16 plot in the [fig.] it can be seen a linear relationship for higher similarities, and as the points got closer to the 0 the estimation gets more noise. So, despite wide error for low similarities in the HLL16, the query can match properly the relevant songs with good accuracy for higher similarities, making the performance of QBH using HLL equivalent to the baseline.

## Memory

The memory required to store the Hyperloglog are completely related to the number of registers, given the equation [Add equation number of registers], the number of registers grows exponentially in relation to the parameter p. The Hyperloglog++ has improvements in the memory consumption to store it in sparse mode depending on the cardinality of the set added to the HLL sketch.

A graph of data usage

Description automatically generated with medium confidence

Fig. A Memory usage comparison in songs encoding by method and Data Structure

Note for all of the methods the HLL4 used significantly less memory, followed by HLL8. As mentioned previously, the amount of memory to store the HLL relates to the number of registers, so regardless the method all HLL had the same memory usage for each respective p. While the set data structure depends on how the elements in the set are encoded and the cardinality of the set.

For statistical significance, it was applied the paired t-Test in the memory usage comparison, between each HLL data structure and the respective set.

Let , and

The Null Hypotheses is

And the Alternative Hypotheses is

it will provide 40 results for each dataset, with significance level of 0.05 See in tables below:

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Data Structure | **Method** | **Memory Usage HLL** | **Memory Usage Set** | **P-value** | **Reject Null Hypotesis** |
| **hll\_4** | pitch\_diff-ngram\_5 | 507 | 31837 | 2.29E-113 | TRUE |
| hll\_4 | UpSameDown-ngram\_5 | 551 | 14369 | 5.61E-250 | TRUE |
| hll\_4 | pitch\_diff-ngram\_8 | 509 | 34505 | 1.12E-113 | TRUE |
| hll\_4 | UpSameDown-ngram\_8 | 509 | 30455 | 1.51E-160 | TRUE |
| hll\_4 | pitch\_diff-ngram\_10 | 504 | 36497 | 7.07E-115 | TRUE |
| hll\_4 | UpSameDown-ngram\_10 | 512 | 37326 | 5.39E-132 | TRUE |
| hll\_4 | pitch\_diff-ngram\_12 | 508 | 38219 | 2.72E-115 | TRUE |
| hll\_4 | UpSameDown-ngram\_12 | 507 | 41004 | 2.20E-120 | TRUE |
| hll\_4 | pitch\_diff-multi\_gram\_5\_10 | 533 | 218893 | 1.77E-137 | TRUE |
| hll\_4 | UpSameDown-multi\_gram\_5\_10 | 525 | 157724 | 3.52E-161 | TRUE |

Table B. Memory Comparison for HLL4

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Data Structure | **Method** | **Memory Usage HLL** | **Memory Usage Set** | **P-value** | **Reject Null Hypotesis** |
| hll\_8 | pitch\_diff-ngram\_5 | 2530 | 31837 | 1.44E-104 | TRUE |
| hll\_8 | UpSameDown-ngram\_5 | 2532 | 14369 | 7.36E-222 | TRUE |
| hll\_8 | pitch\_diff-ngram\_8 | 2530 | 34505 | 1.47E-105 | TRUE |
| hll\_8 | UpSameDown-ngram\_8 | 2525 | 30455 | 1.33E-149 | TRUE |
| hll\_8 | pitch\_diff-ngram\_10 | 2527 | 36497 | 3.62E-107 | TRUE |
| hll\_8 | UpSameDown-ngram\_10 | 2531 | 37326 | 6.83E-124 | TRUE |
| hll\_8 | pitch\_diff-ngram\_12 | 2531 | 38219 | 6.45E-108 | TRUE |
| hll\_8 | UpSameDown-ngram\_12 | 2527 | 41004 | 2.26E-113 | TRUE |
| hll\_8 | pitch\_diff-multi\_gram\_5\_10 | 2586 | 218893 | 4.60E-136 | TRUE |
| hll\_8 | UpSameDown-multi\_gram\_5\_10 | 2582 | 157724 | 4.61E-159 | TRUE |

Table C. Memory Comparison for HLL8

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Data Structure | **Method** | **Memory Usage HLL** | **Memory Usage Set** | **P-value** | **Reject Null Hypotesis** |
| hll\_12 | pitch\_diff-ngram\_5 | 37395 | 31837 | 1.05E-07 | TRUE |
| hll\_12 | UpSameDown-ngram\_5 | 37406 | 14369 | 0 | TRUE |
| hll\_12 | pitch\_diff-ngram\_8 | 37395 | 34505 | 0.00976738 | TRUE |
| hll\_12 | UpSameDown-ngram\_8 | 37392 | 30455 | 2.44E-20 | TRUE |
| hll\_12 | **pitch\_diff-ngram\_10** | **37393** | **36497** | **0.44353805** | **FALSE** |
| hll\_12 | **UpSameDown-ngram\_10** | **37397** | **37326** | **0.94698486** | **FALSE** |
| hll\_12 | **pitch\_diff-ngram\_12** | **37394** | **38219** | **0.4995853** | **FALSE** |
| hll\_12 | UpSameDown-ngram\_12 | 37392 | 41004 | 0.00446281 | TRUE |
| hll\_12 | pitch\_diff-multi\_gram\_5\_10 | 37470 | 218893 | 4.50E-111 | TRUE |
| hll\_12 | UpSameDown-multi\_gram\_5\_10 | 37461 | 157724 | 1.35E-120 | TRUE |

Table D. Memory Comparison for HLL12

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Data Structure | **Method** | **Memory Usage HLL** | **Memory Usage Set** | **P-value** | **Reject Null Hypotesis** |
| hll\_16 | pitch\_diff-ngram\_5 | 576533 | 31837 | 0 | TRUE |
| hll\_16 | UpSameDown-ngram\_5 | 576542 | 14369 | 0 | TRUE |
| hll\_16 | pitch\_diff-ngram\_8 | 576534 | 34505 | 0 | TRUE |
| hll\_16 | UpSameDown-ngram\_8 | 576529 | 30455 | 0 | TRUE |
| hll\_16 | pitch\_diff-ngram\_10 | 576530 | 36497 | 0 | TRUE |
| hll\_16 | UpSameDown-ngram\_10 | 576535 | 37326 | 0 | TRUE |
| hll\_16 | pitch\_diff-ngram\_12 | 576533 | 38219 | 0 | TRUE |
| hll\_16 | UpSameDown-ngram\_12 | 576531 | 41004 | 0 | TRUE |
| hll\_16 | pitch\_diff-multi\_gram\_5\_10 | 576615 | 218893 | 6.09E-220 | TRUE |
| hll\_16 | UpSameDown-multi\_gram\_5\_10 | 576603 | 157724 | 0 | TRUE |

Table E. Memory Comparison for HLL16

For all the cases except pitch\_diff-ngram\_10, UpSameDown-ngram\_10, pitch\_diff-ngram\_12 in the HLL\_12 data structure, the memory usage was statistically significant different. So It can be seen the memory savings would be achieved in both options HLL\_4 and HLL\_8 for all methods. While the HLL\_16 used more memory for all the methods. While the HLL\_12 had three methods that the null hypotheses cannot be rejected pitch\_diff-ngram\_10, UpSameDown-ngram\_10, pitch\_diff-ngram\_12, and other cases where the HLL spend more memory, others with less memory, see the table.

Analysing both equations [error\_rate] and [Add equation number of registers], it is possible to see the clear trade-off between memory and accuracy of cardinality estimation. As mentioned in the previous section, this framework is using the cardinality estimation to estimate overlapping coefficient, so the error rate for the similarity is higher than the original cardinality error. So, to improve the accuracy and reach a MRR statistically equal to the baseline in the QBH problem, it was necessary to have HLL with p=16. The problem is that this HLL memory usage is much bigger compared to the original baseline sets memory storage. While the HLL\_4 and HLL\_8 saved memory but with a poor performance. So, the benefits of using the HLL were not met, once the only method that preserved the quality used too much memory, and the one with massive savings in memory had significantly lower performance in the QBH problem.

## Analysis

Overall, after analysing the whole results of the experiment it is possible to see the usage of the HLL in the QBH problem using the inclusion-exclusion principle to estimate the similarity of queries songs is not efficient. The case it demonstrated similar performance compared to the set equivalent version used more resources to be computed. For obvious reasons the result is associated to the original sets baseline performance, so it is a natural questioning: “Would another pipeline of encoding queries and songs or another set creation method bring better results for HLL in the QBH problem?” According to the evidence displayed in this project possible no. The core problem of this approach was the cumulative error in the intersection set estimation, impacting the overlapping coefficient estimation directly. The reason for such a big problem is the different sizes of the sets between queries, songs, and intersections. The songs sets are in average 8.9 times bigger than queries sets [See Table]. It is a property intrinsic to the QBH problem. If they have the same order of cardinality, the intersection error would not be that different compared to the current format. Unless some study tackles this problem in a completely different way, that would not have this proportion issue between the query and song sets cardinality.

|  |  |  |  |
| --- | --- | --- | --- |
| Method | **Cardinality  Query Set Avg.** | **Cardinality  Song Set Avg.** | **Song/Query Cardinality Ratio** |
| UpSameDown-multi\_gram\_5\_10\_set | 147 | 1064.7 | 7.2 |
| UpSameDown-ngram\_10\_set | 24.1 | 234.6 | 9.7 |
| UpSameDown-ngram\_12\_set | 22.3 | 244.2 | 11 |
| UpSameDown-ngram\_5\_set | 22.5 | 93.2 | 4.1 |
| UpSameDown-ngram\_8\_set | 25.5 | 203.5 | 8 |
| pitch\_diff-multi\_gram\_5\_10\_set | 160.1 | 1506.9 | 9.4 |
| pitch\_diff-ngram\_10\_set | 24.3 | 250.8 | 10.3 |
| pitch\_diff-ngram\_12\_set | 22.4 | 249.2 | 11.1 |
| pitch\_diff-ngram\_5\_set | 28.9 | 248.8 | 8.6 |
| pitch\_diff-ngram\_8\_set | 26.3 | 252.1 | 9.6 |

Table. Average Cardinality of Sets per Method

Another important aspect is that all the cardinalities were very small compared to typical usage of HLL, so the memory reduction were not achieved as the original dataset is not too big.

In addition to it, one probable issue in scaling up this framework for QBH for bigger databases, is the limited number of different terms in the whole query set corpus. Making it harder to discriminate the relevant songs between a huge range of options in the database. It is equivalent of having too many documents with the same labels, so the number of similar matches would be too high. To avoid it, it might be necessary longer queries, what is impractical at some point, or it can be created new ways to encode efficiently the information and increase the dimensionality of the queries and songs sets, with more details. So, there will be always trade-offs, between database size, subject execution robustness, query size.

# Conclusion and Discussion

The capstone project proposed to assess the potential usage of Hyperloglog++ data structure in the QBH problem. For that it was implemented successfully a framework to compare the impact of QBH using the similarity method based on traditional overlapping coefficient and the proposed similarity algorithm using the overlapping based in HLL cardinality estimation.

In the Introduction it was covered the problem explanation, with an overview about what was covered in the whole project report. Then the literature review, it was presented the main relevant research, key topics, and concepts in the QBH area and Distinct Counting estimators. In the Methodology chapter the framework design and experiment description were defined and explained deeply, detailing the tools, and presenting the ethical aspects considerations of this project. Then in the Evaluation and Analysis, the results were examined and a deeper investigation about the interpretation and justification for the results were covered. The performance and memory comparison were made and analysed against the baseline method explained in the methodology.

The evaluation and statistical results were made successfully, reaching a negative conclusion about the usage of the HLL in this problem. It was found that was necessary to have a big HLL with a high number of registers to reach similar levels of performance compared to the baseline. Consequently, it would not bring the desired common benefits of HLL in terms of significant memory reduction, as it was consuming more memory to process the data. It was discussed and investigated, pointing the reasons behind this issue. It was found the problem between the intersection estimation error and similarity estimation, are intrinsically connected, the principle of inclusion and exclusion have issues dealing with sets with a such different cardinalities.

## Recommendations for future research

Future research can study the impact of the Similarity HLL Based in another context where the cardinality of sets involved are not so different and containing a discrete and high dimensional dataset. It could possibly take benefits from the framework presented in this project. For instance, collaborative filtering or item-based recommendation systems, they might have good characteristics compatible to this framework, being a potential positive case.

There are also room to find different ways to encode the sets to increase quality of the baseline method as well. Mainly for polyphonic songs. It can be found other methods to use the HLL in a different way to calculate the inclusion coefficient, that it is promising better accuracy, it would be an interesting area of research to adjust the framework studied in this project applying other strategies such as [https://www.vldb.org/pvldb/vol11/p1097-nazi.pdf]. in the QBH problem.

# Appendix

## MIDI Pitch Table

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| *Pitch* | *Frequency* | *MIDI* | *Pitch* | *Frequency* | *MIDI* | *Pitch* | *Frequency* | *MIDI* |
| C0 | 16.352 | 12 | C4 | 261.626 | 60 \*MC | C8 | 4186.009 | 108 \*PH |
| C#0 | 17.324 | 13 | C#4 | 277.183 | 61 | C#8 | 4434.922 | 109 |
| D0 | 18.354 | 14 | D4 | 293.665 | 62 | D8 | 4698.637 | 110 |
| D#0 | 19.445 | 15 | D#4 | 311.127 | 63 | D#8 | 4978.032 | 111 |
| E0 | 20.602 | 16 | E4 | 329.628 | 64 | E8 | 5274.042 | 112 |
| F0 | 21.827 | 17 | F4 | 349.228 | 65 | F8 | 5587.652 | 113 |
| F#0 | 23.125 | 18 | F#4 | 369.994 | 66 | F#8 | 5919.912 | 114 |
| G0 | 24.500 | 19 | G4 | 391.995 | 67 | G8 | 6271.928 | 115 |
| G#0 | 25.957 | 20 | G#4 | 415.305 | 68 | G#8 | 6644.876 | 116 |
| A0 | 27.500 | 21 \*PL | A4 | 440.000 | 69 | A8 | 7040.000 | 117 |
| A#0 | 29.135 | 22 | A#4 | 466.164 | 70 | A#8 | 7458.620 | 118 |
| B0 | 30.868 | 23 | B4 | 493.883 | 71 | B8 | 7902.133 | 119 |
|  |  |  |  |  |  |  |  |  |
| C1 | 32.703 | 24 | C5 | 523.251 | 72 | C9 | 8372.019 | 120 |
| C#1 | 34.648 | 25 | C#5 | 554.365 | 73 | C#9 | 8869.845 | 121 |
| D1 | 36.708 | 26 | D5 | 587.330 | 74 | D9 | 9397.273 | 122 |
| D#1 | 38.891 | 27 | D#5 | 622.254 | 75 | D#9 | 9956.064 | 123 |
| E1 | 41.203 | 28 | E5 | 659.255 | 76 | E9 | 10548.083 | 124 |
| F1 | 43.654 | 29 | F5 | 698.457 | 77 | F9 | 11175.305 | 125 |
| F#1 | 46.249 | 30 | F#5 | 739.989 | 78 | F#9 | 11839.823 | 126 |
| G1 | 48.999 | 31 | G5 | 783.991 | 79 | G9 | 12543.855 | 127 |
| G#1 | 51.913 | 32 | G#5 | 830.609 | 80 | G#9 | 13289.752 | - |
| A1 | 55.000 | 33 | A5 | 880.000 | 81 | A9 |  | - |
| A#1 | 58.270 | 34 | A#5 | 932.328 | 82 | A#9 |  | - |
| B1 | 61.735 | 35 | B5 | 987.767 | 83 | B9 |  | - |
|  |  |  |  |  |  |  |  |  |
| C2 | 65.406 | 36 | C6 | 1046.502 | 84 |  |  |  |
| C#2 | 69.296 | 37 | C#6 | 1108.731 | 85 |  |  |  |
| D2 | 73.416 | 38 | D6 | 1174.659 | 86 |  |  |  |
| D#2 | 77.782 | 39 | D#6 | 1244.508 | 87 |  |  |  |
| E2 | 82.407 | 40 | E6 | 1318.510 | 88 |  |  |  |
| F2 | 87.307 | 41 | F6 | 1396.913 | 89 |  |  |  |
| F#2 | 92.499 | 42 | F#6 | 1479.978 | 90 |  |  |  |
| G2 | 97.999 | 43 | G6 | 1567.982 | 91 |  |  |  |
| G#2 | 103.826 | 44 | G#6 | 1661.219 | 92 |  |  |  |
| A2 | 110.000 | 45 | A6 | 1760.000 | 93 |  |  |  |
| A#2 | 116.541 | 46 | A#6 | 1864.655 | 94 |  |  |  |
| B2 | 123.471 | 47 | B6 | 1975.533 | 95 |  |  |  |
|  |  |  |  |  |  |  |  |  |
| C3 | 130.813 | 48 | C7 | 2093.005 | 96 |  |  |  |
| C#3 | 138.591 | 49 | C#7 | 2217.461 | 97 |  |  |  |
| D3 | 146.832 | 50 | D7 | 2349.318 | 98 |  |  |  |
| D#3 | 155.564 | 51 | D#7 | 2489.016 | 99 |  |  |  |
| E3 | 164.814 | 52 | E7 | 2637.021 | 100 |  |  |  |
| F3 | 174.614 | 53 | F7 | 2793.826 | 101 |  |  |  |
| F#3 | 184.997 | 54 | F#7 | 2959.956 | 102 |  |  |  |
| G3 | 195.998 | 55 | G7 | 3135.964 | 103 |  |  |  |
| G#3 | 207.652 | 56 | G#7 | 3322.438 | 104 |  |  |  |
| A3 | 220.000 | 57 | A7 | 3520.000 | 105 |  |  |  |
| A#3 | 233.082 | 58 | A#7 | 3729.310 | 106 |  |  |  |
| B3 | 246.942 | 59 | B7 | 3951.066 | 107 |  |  |  |

 [https://homes.luddy.indiana.edu/donbyrd/Teach/MusicalPitchesTable.htm]

## MIDI sample from Basic Sample

To DO

## Sample Demucs

To DO

# References

To do