Project report: Sacha-Zam

Sacha RUCHLEJMER
Master's Program
sqru@kth.se

1. INTRODUCTION

This project aims to reproduce the Shazam algorithm by comparing fingerprints extracted from the spectrum of the audio extract and the database. This problem is important because it enables us to determine how to extract unique features from an audio file. It has many applications; it can allow people to discover unknown artists and songs, but it also allows musicians to become famous.

2. METHOD

The method we implemented originates from lab4, which is inspired by [1]. The objective is to determine from which file in the Kikibouba dataset the queries used in the lab originate. The fingerprinting system operates by extracting specific points of interest, referred to as anchors. These anchors represent points with the highest energy in a section of the signal's spectrum. For each anchor, we consider all surrounding points within an arbitrary radius to create a hash. All these hashes collectively form the fingerprint of the audio file.

2.1 Anchors

To compute the anchors, we require the spectrum of our audio file. To obtain this, we will use the Short-Time Fourier Transform (STFT). This will provide us with a three dimensional graph displaying the energy for each frequency over time.

2.2 Fingerprint

To generate a unique fingerprint for an audio file, we must compute pairs of anchors. For this, we'll calculate the hash for each anchor pair. A hash takes the form: $(f_1, f_2, \Delta t_{12})$, where f_1 and f_2 represent the frequencies of the two anchors, and Δt_{12} denotes the time difference between these points.

Eventually, the fingerprint will consist of a list of hashes along with the times at which they occur, represented as: $F = ((\tau_1, h_1), (\tau_2, h_2), \ldots)$.

2.3 Matching detection

To identify a match between a sample file and an audio file, we will compare their fingerprints. For each hash in the sample's fingerprint, we will search for a matching hash in the audio file. This process is repeated for each audio file until we identify the correct match. To compare the hashes, we'll check if the sum of the magnitude differences across the hash dimensions is zero.

2.4 sorting the match

It's possible to have a match between a sample and an audio file, even if the sample wasn't extracted from that particular audio. This can happen because the hashes aren't unique and might appear in any audio, though not necessarily at the same time. Hence, assuming that the playback speed hasn't been altered in either audio, the matches should form a line defined by:

$$y = ax + b \tag{1}$$

where
a is the slope of the line,
b is the y-intercept.

So, if the song's speed remains unmodified, we should observe a slope close to 1.

From here, two methods were implemented:

2.4.1 Method: Time-Difference Fingerprint Matching

To pinpoint a genuine match, we look for a consistent time difference between occurrences in the sample and the audio file among the matches. Thus, we'll compute this offset for each match and examine a histogram for any significant peaks, defined by:

$$\Delta t = x - y \tag{2}$$

where
x Time of the match in the database audio file,
y Time of the match in the sample.

If we observe a Δt with a very high value over a short period of time between matches, we assume that it is a match.

2.4.2 Method: Line Rotational Histogram Analysis

In this approach, we take the output from the matching algorithm and rotate it using a rotational matrix to produce a vertical straight line. Since we know the slope of the matching line is 1, rotating by $\pi/4$ should yield a vertical line when the correct audio file is identified. Subsequently, we examine a histogram to see if there is a peak with a high occurrence at a specific abscissa. If such a peak is observed, we can infer a match, as this peak represents the line of the match.

3. RESULTS

First, we must extract our features and compute the anchors for all queries and the database. As illustrated in figure 1, we can extract the anchors from an audio signal.

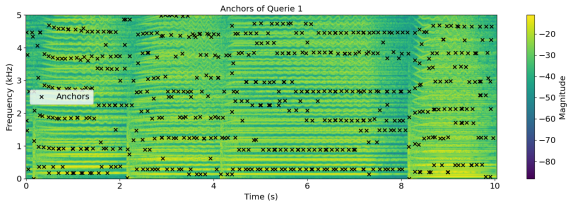

Figure 1. Anchor map of the query 1.

For the following results, we used the first method.

Now, we apply our matching algorithm to query 1 to search for a match within the Kikibouba dataset. We believe we have found a match; as evident in figure 2, there is a distinctive straight line with a slope close to 1 around the 30-second mark.

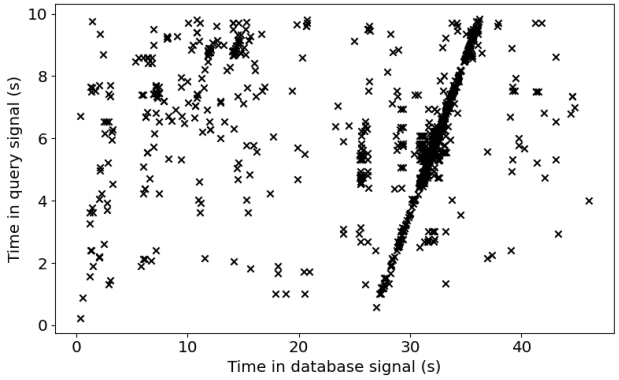

Figure 2. Match map with query 1.

Now, let's examine our histogram to check for any significant peaks.

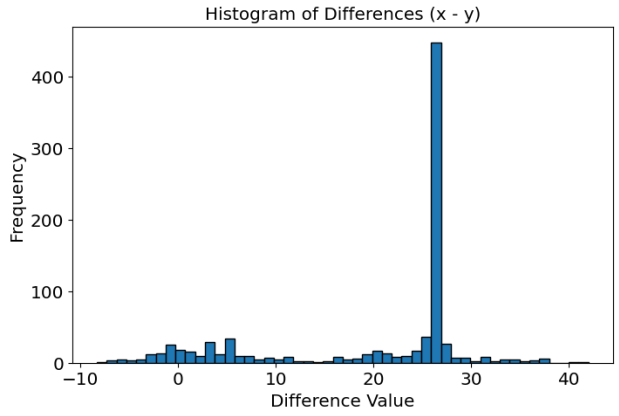

Figure 3. Histogram of the Δt value in the match.

As illustrated in figure 3, there is a highly distinctive peak with a frequency of occurrence of 400, while the others hover around 30.

Furthermore, when we examine matches on other files, as depicted in figure 4, it appears as a random cloud of crosses.

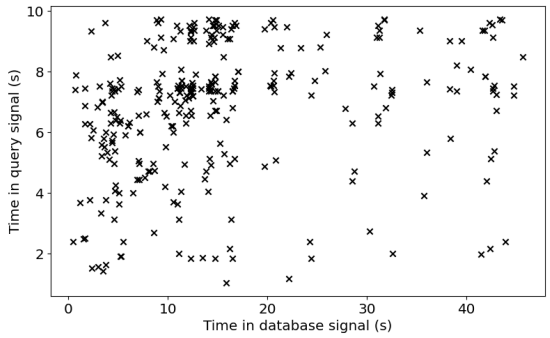

Figure 4. Match map with a no match with query 1.

And if we take a look at the histogram on the figure 5 we can see that there is no very high peak, and the higher go only to 30 when with a match it was raising to 400.

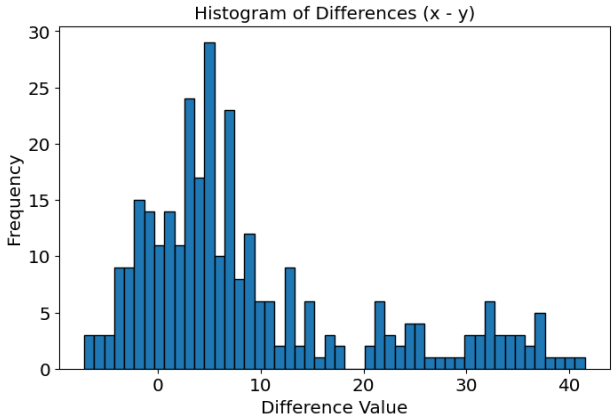

Figure 5. Histogram of the Δt value in the no match.

3.1 Using the Second method

With the Line Rotational Histogram Analysis method, we obtain the same results. This corroborates our assumptions about the correct files. However, it could not identify matches for those we were uncertain about or lacked evidence for.

Still, the method performs quite well. For the first query's detection, we successfully rotated the line to achieve a vertical orientation, as shown in figure 6.

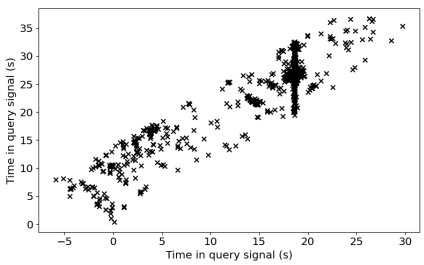

Figure 6. Match map rotated to produce a vertical line.

Moreover, when we aggregate all these matches into a histogram, a very distinctive peak is evident in figure 7, indicating a match.

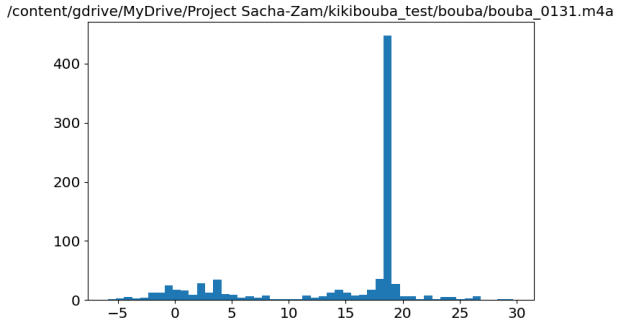

Figure 7. Histogram of matches based on the rotated match map.

We conclude that the first query was extracted from the file: `bouba_0131.m4a`**.** This file from the database provided us with the significant results for the first query. To summarize, we successfully determined the origins of 3 out of the 6 queries, as shown in table 1. For the others, we only have suspicions or clues, as evident in table 2.

	Query 1	Query 3	Query 4
sure	`bouba_0131`	`kiki_0199`	`kiki_0043`

Table 1. All the file from where comes the queries, for sure.

	Query 2	Query 5	Query 6
not sure	`bouba_0178`	not known	not a Bouba

Table 2. All the file from where comes the queries or leads.

4. DISCUSSION

My implementation successfully determines the origin of certain queries. However, its efficacy could be enhanced by incorporating a confidence level for predictions. Introducing this feature was challenging due to my reliance on a threshold for detecting matches, making the results almost binary. This thresholding might be why I struggle to pinpoint the exact origins of some queries; for the ambiguous ones, the histogram values remain low and never exceed this threshold.

The two methods used here are similar, but the second one has a reduced risk of producing false positives. With the first method, we might encounter a series of deltas with identical values, which, when aggregated, create a peak in the histogram. In contrast, even if this occurs in the second method, it is not an issue since they will not happen simultaneously and thus will not accumulate at the same peak.

We suspect that the files we couldn't identify originated from files not provided in our database. According to the article [2], there are 500 files in the Kikibouba database, but we only have 400. Despite our efforts, we couldn't find the missing files, so this assertion remains a hypothesis.

Before adopting the histogram method, we determined a match between two files by computing the slope of the line formed by all matching points. We initially assumed that for a genuine match, the line would have a consistent slope, with all matches aligning on the diagonal. However, this was not the case. Specifically, for the match with q1, the line was nearly horizontal, as depicted in figure 8.

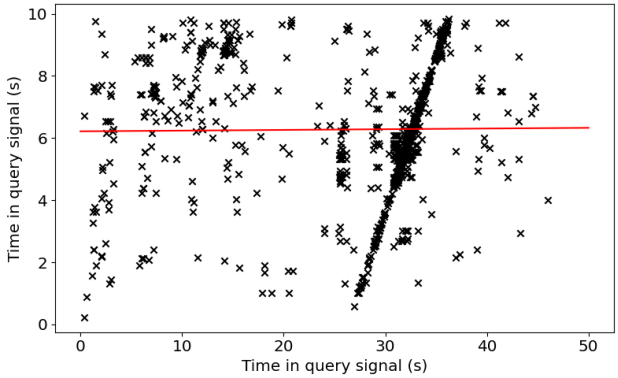

Figure 8. Straight line determine with the matches.

5. CONCLUSION

In this project, we attempted to reproduce the Shazam algorithm, an essential tool for audio fingerprinting and matching. Our approach focused on the extraction of unique fingerprints from audio samples and matching these against a database.

To achieve this goal, we implemented two methods with a considerable degree of success: three out of six queries were conclusively matched to their source files. However, for the other queries, we suspect they originate from a different database since even on the match map, we couldn't identify a specific line characteristic of an authentic match. Alternatively, they might have been subjected to noise to such an extent that we couldn't trace back to the original file.

The key learnings from this study include:

- The effectiveness of audio fingerprinting techniques based on anchor points in the spectrum.
- The usefulness of histogram-based techniques in distinguishing genuine matches from false positives.
- The significance of code optimization in scenarios involving extensive calculations.

For future work, we aim to improve the robustness of the matching algorithm and introduce a confidence level to better gauge the accuracy of our predictions, allowing us to also suggest other matches with slightly reduced confidence levels.

In conclusion, audio fingerprinting and matching have a lot of applications in music discovery and rights management. With continued research and refinement, such techniques can be made even more powerful and versatile.

6. REFERENCES

[1] A. L.-C. Wang, "An industrial-strength audio search algorithm," 2003.

[2] B. L. Sturm and N. Collins, "The kiki-bouba challenge," in *Proceedings of the 15th International Society for Music Information Retrieval Conference, ISMIR 2014*, 2014, pp. 21–26.