# SONG RECOGNITION USING AUDIO FINGERPRINTING

*Varun Khatri, Lukas Dillingham, and Ziqi Chen*

Dept. of Electrical and Computer Engineering
University of Rochester
Rochester, NY 14627, USA

## ABSTRACT

An audio fingerprint is a compact and unique representation of an audio file or stream, which makes it possible to recognize a song in any format without the help of any embedded watermarks or meta-data. In this paper, we implement the Shazam's song recognition algorithm which uses audio fingerprinting and hash search method. Experimental results exhibit good robustness against noise and, compared to the traditional simple search strategy, the hash-based searching strategy shows great improvement in searching speed.

***Index Terms***— Audio fingerprinting, audio signal processing, acoustic identification, song recognition

## 1. INTRODUCTION

Since the 21st century, with the development of digital audio transportation and storage techniques, the need for recognizing a song or a clip has been increasing, which makes audio content recognition a hot topic in the audio signal processing field. As the name suggests, audio content recognition deals with recognizing the content of a stream of audio, either from a file on the computer or the internet or something which is recorded by a microphone or smart phone from the real world. Among all of those content-based audio recognition technologies, audio fingerprinting is famous for its ability to connect unknown audio or an audio clip in any kind of format to corresponding metadata [1].

An audio fingerprint is a condensed digital summary, deterministically generated from an audio signal, that can be used to identify an audio sample or quickly localize similar items in an audio database [2]. Most audio fingerprinting techniques involve time-frequency analysis of an audio signal to retrieve a pattern which is unique to the signal. These patterns are created by isolating certain data from the time-frequency representation of the signal. This data is selected assessing a certain property of the signal, which is robust against deterioration due to noise and other forms of compression. This process is followed for both the database audio and the query audio. Once the query fingerprint is ready, it is then compared with the existing fingerprints in the database and the software then returns the song name and artist, if it gets a match.

In the past few years, a lot of new audio fingerprint algorithms have been proposed. Li-Chun Wang [3] developed an efficient and robust audio search system based on audio fingerprint. Kim and his group [4] suggested a new audio fingerprint extraction method for identifying classical music. Seetharaman and Rafii [5] proposed a new cover song identification with 2DFT which is robust to noise and distortion. Lee et al. [6], and Xiong et al. [7], also obtained good experimental results in recognizing songs with their new systems. Because of the commercial application potential of the audio fingerprint, many industrial companies, like Philips [8] and Google [9-13], also make a lot of contribution.

In this paper, we choose to implement one of the most popular audio fingerprinting algorithms – Shazam's algorithm [3]. The remainder of this paper is organized as follows. The algorithm structure is described in Section 2. Then, in Section 3, several significant details about the implementation of Shazam's algorithm are shown. Lastly, Section 4 presents experimental results of comparing song clips with different noise levels to the song in the database.

## 2. ALGORITHM

The whole audio fingerprinting system include 2 essential sections: fingerprint extraction and matching method.

### 2.1. Fingerprint extraction

Because most digital audio files are stored in different formats and recorded with different sampling rates, pre-processing is required before generating audio fingerprint from the audio. Firstly, the sampled sound needs to pass from stereo to mono, since nowadays many sound files are stored in multi-channel. The frequency range of human hearing is 20 Hz to 20 kHz, so the sampling rate of digital audio files is usually 44.1 kHz. However, such a high sampling rate is unnecessary for the audio fingerprint, since most instruments are not capable of producing sound reaching such a high frequency, and it will also result in spectrograms generated with STFT too large, slowing the whole matching procedure.

In light of this deficiency, downsampling the signal with a cost of losing some information is acceptable and reasonable. Then, applying a window function to the signal and applying FFT for each window, a spectrogram is generated. In consideration of noise tolerance, only the time-frequency points with the strongest magnitude need to be kept. However, one point per frame will absolutely lead to critical loss of effective information, so dividing each frame into small bands are necessary. And then, a Max filter is applied to the spectrogram so that only peaks are retained. The chosen peaks form a constellation as seen in Fig.1, which corresponds to a fingerprint. The whole process for fingerprinting is shown in Fig. 2.

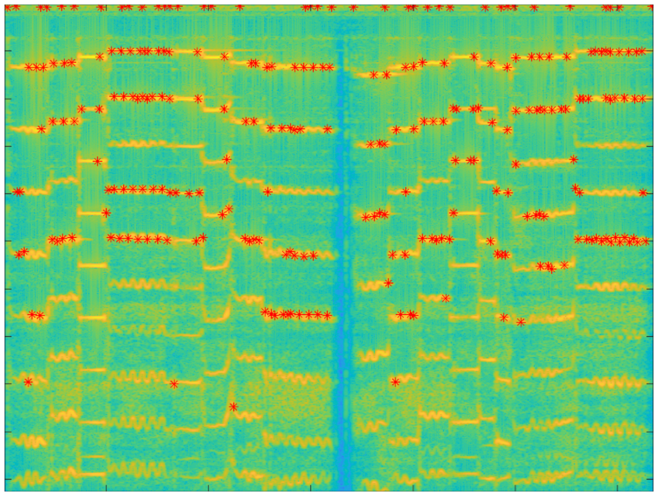


Fig.1. Constellation plotting.

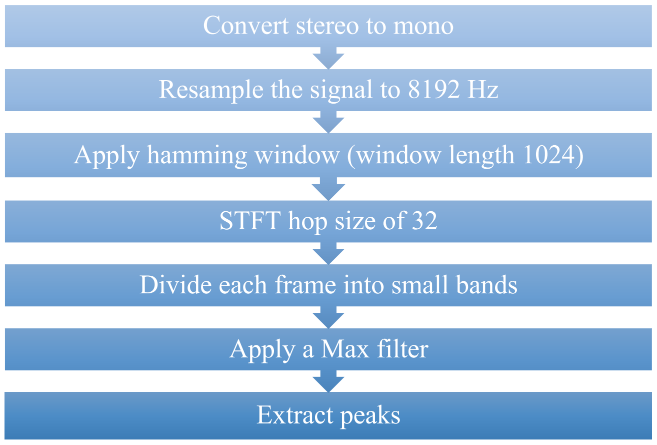

Convert stereo to mono
Resample the signal to 8192 Hz
Apply hamming window (window length 1024)
STFT hop size of 32
Divide each frame into small bands
Apply a Max filter
Extract peaks

Fig.2. The flow diagram of audio fingerprint extraction process.

### 2.2. Matching with hashes

The constellation points shown in Fig.1 are combined to form hashes, which improves the speed of matching when comparing a song clip with larger chunks of data (e.g. a database including 4 million songs).

This combinational hashing involves selecting a peak (anchor) and a corresponding zone containing other peaks (target zone) and forming pairs between them. The matching would then between done between pairs. This hashing search method increases the matching speed greatly. For example, if one anchor point is paired with 10 target points, this increase in elements being compared increases the search speed by a factor of 10000. A trade off with the combinational hashing is that it reduces the probability of matching as if one of the peaks is affected by distortion, then the pair will not be a match. Fig.3 shows the general matching procedure.

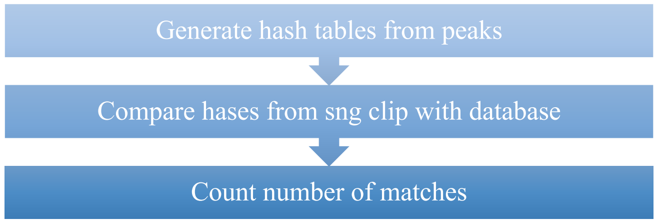

Generate hash tables from peaks
Compare hases from sng clip with database
Count number of matches

Fig.3. The flow diagram of matching process.

## 3. IMPLEMENTATION

### 3.1. Fingerprinting

The process of creating a fingerprint from a song or audio clip, as shown in Fig.2, is explained in more detail here. First it is necessary to convert a multichannel audio file to mono. From here, the audio is downsampled to 8192 Hz. The idea behind this is that the majority of the information in the spectrogram that defines a song is under the Nyquist frequency of 4096 Hz. The majority of most instruments, vocals, and percussion fall within this range. So, sacrificing the higher frequencies for increased efficiency is acceptable.

Once downsampled, the STFT is applied to create a spectrogram. A window of 1024 gives a sufficient amount of frequency resolution. A window length larger than this might make the peak locations more sensitive to noise, leading to a decrease in matching points. Conversely, a very small window size will lead to more false matches. One must also take into consideration that an audio clip could be started at any point in the song. Thus, it is required that the hop size be smaller to compensate for different start times of an audio clip. A larger hop size might not preserve the time location of peaks well, so a hop size of 32 was chosen.

After the spectrogram is created, the next task is to find the local magnitude peaks. To accomplish this, the frequency bins are divided into logarithmic bands. 6 bands were chosen, and the maximum of each band was taken. The result is shown in Fig.4.

From here, the spectrogram is treated as an image and a max filter is applied. This makes a neighborhood of pixels all have the maximum value in the neighborhood. The filtered image is then compared pixel by pixel to the original image. Only the pixels in the original image with the max value in the local neighborhood will be the same as the max filtered image, leaving only the local maximums. The value of the magnitude is not important, as it will vary depending on the recording of the audio clip. Only time and frequency locations are important, so the image is converted into a binary image with ones at the local maximums and zeros everywhere else. The resulting time frequency locations are the fingerprint of the audio as shown in Fig.5.

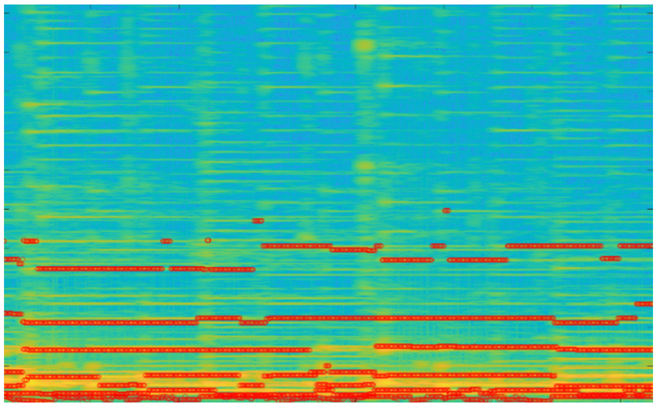


Fig.4. Magnitude maximums of each band in each frame

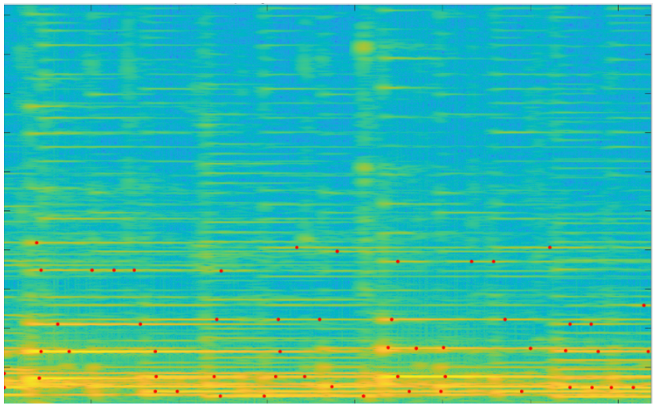


Fig.5. Time and frequency locations (red dots) of local maximums

### 3.2. Hash tables for quick searching

Once the time and frequency locations of peaks are stored for all songs in the database, a clip of audio can be compared to each song. A simple way of doing this is to slide and compare frame by frame the 2D array of points of the audio clip to the larger 2D array of each song in the database. This process can be seen as in Fig.6.

This method of comparison is extremely slow, however. A much more efficient approach is to use a hash table to store the data points. Creating pairs of points retains their relative locations and this can be used as elements in the table. A pair of points, the anchor point and the target point, forms a pair and a hash is created using the frequency of each point and the change in time between them. This is done for each point within a target zone for each anchor point. This is illustrated in Fig.7.

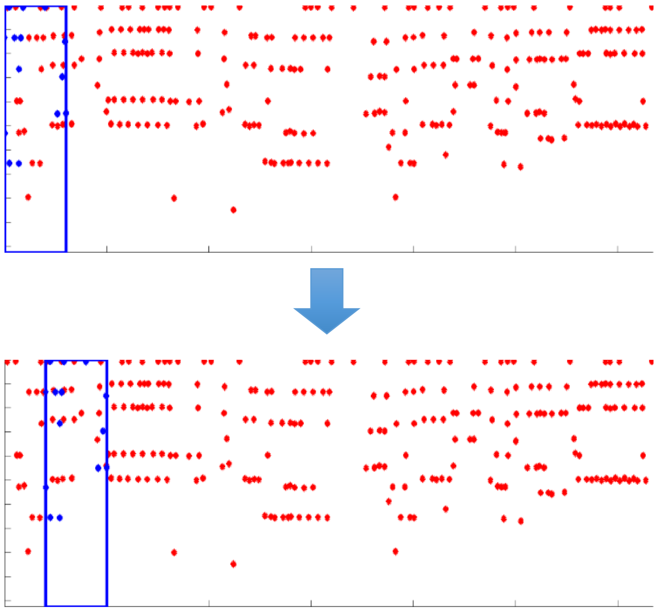


Fig.6. Comparing peak locations of an audio clip (blue) to the peak locations of a song (red).

$$\text{Table}[f_{anchor1}] = f_{target\ zone\ point} + \frac{\Delta t}{100}$$

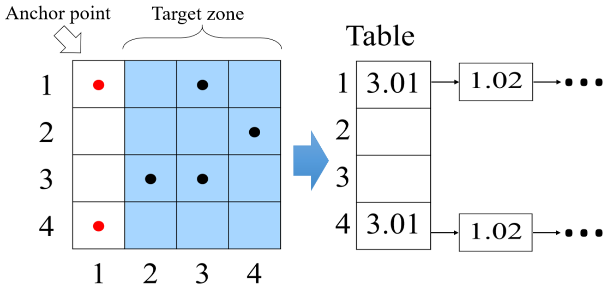
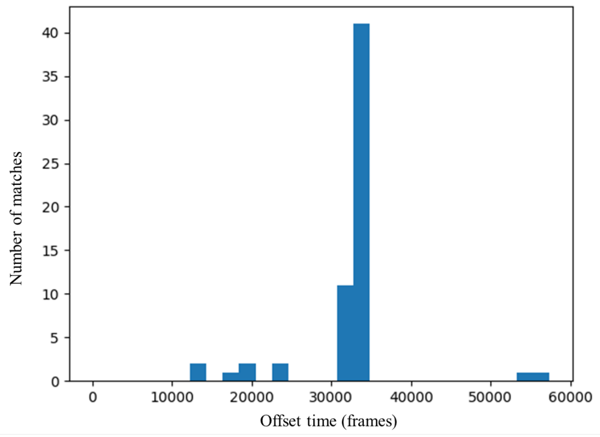


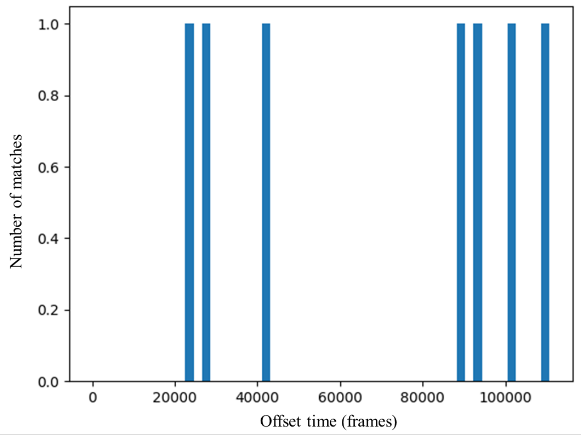
Fig.7. Creating a hash table from pairs of points.

For our purposes, we defined 50 frames to the right of the anchor point to be the target zone. For each point then, pairs are made for every point that is within this zone. The bin number of the anchor point is the index of the table and the element for this index is a float value defined by the bin number of the target point plus the change in time divided by 100. This type of table is a separate chaining hash table and the advantage of it is that for searching for a single entry the search time is at best O(1) and at worst O(n) if there are very large amounts of elements.

When a hash table is created for an audio clip (client side), it is then compared to all the hash tables in the database (server side). If one element is in the client-side table is also in the table of a song in the server side, then the count of number of matched elements is increased by one. Pure number of matching table elements is a decent approach to finding a matching song, however a more consistent approach is to view the statistics of offset times in frames for each mutual element. To do this, a separate table was created that stores the number of frames an anchor point is from the beginning of a song. From this table of offset times, it is possible to create a histogram of offset times for all mutual elements in the client and server tables. A matching song will have a spike in the histogram where the client-side audio clip overlaps with the song being compared. Even if two songs have a large number of mutual elements the histogram of the matching song will have a spike in offset times where the incorrect song will likely have a histogram of scattered and inconsistent offset times. This can be seen in Fig.8.



(a) Correct song



(b) Incorrect song

Fig.8. (a) Histogram of a matching song with a clearly defined spike. (b) Histogram of an song that is not a match.

## 4. RESULTS

In order to verify the noise tolerance of hash search method, a clip chosen from "Bistro Fada" was compared with "Bistro Fada" and 3 other songs. Searches were conducted adding different levels of noise (-3, 1, 7, and 15 dB SNR) to the clip, and, in addition, searching without noise adding was also evaluated. Two statistical values are employed as comparison indexes. The first index is Histogram Max, denoting the max value of histogram for each comparison, and the second index is Ratio, denoting number of matches divided by total number of hashes in clip.

Table I Hash search results

| | **-3 dB SNR** | |
|---|---|---|
| **Name** | **Ratio** | **Histogram Max** |
| Bistro Fada | 0.0651 | 40 |
| Kamaloka | 0.0091 | 6 |
| Goodbye Pork Pie Hat | 0.0115 | 6 |
| The Star Boys | 0.0429 | 22 |

| | **1 dB SNR** | |
|---|---|---|
| **Name** | **Ratio** | **Histogram Max** |
| Bistro Fada | 0.0857 | 65 |
| Kamaloka | 0.0111 | 8 |
| Goodbye Pork Pie Hat | 0.0121 | 7 |
| The Star Boys | 0.0656 | 26 |

| | **7 dB SNR** | |
|---|---|---|
| **Name** | **Ratio** | **Histogram Max** |
| Bistro Fada | 0.1797 | 99 |
| Kamaloka | 0.0124 | 5 |
| Goodbye Pork Pie Hat | 0.0183 | 5 |
| The Star Boys | 0.1217 | 37 |

| | **15 dB SNR** | |
|---|---|---|
| **Name** | **Ratio** | **Histogram Max** |
| Bistro Fada | 0.4645 | 136 |
| Kamaloka | 0.0561 | 6 |
| Goodbye Pork Pie Hat | 0.0508 | 11 |
| The Star Boys | 0.2787 | 40 |

| | **Original Song** | |
|---|---|---|
| **Name** | **Ratio** | **Histogram Max** |
| Bistro Fada | 0.6285 | 187 |
| Kamaloka | 0.0714 | 7 |
| Goodbye Pork Pie Hat | 0.0766 | 13 |
| The Star Boys | 0.3075 | 36 |

As revealed in Table I, although two indexes decrease when the amplitude of noise increases, the ratio and histogram max of the correct song are still salient, compared with indexes of the incorrect song, which verifies the robustness of the algorithm.

The software showed consistent performances across all genres of music, however the performance was slightly better for newly recorded songs, which have a fuller spectrum. This is understandable as the fingerprint for a song with a fuller spectrum would have more peaks and would consequently have a better probability of matching due to a greater number of points.

## 5. CONCLUSIONS

Overall, the performance of the algorithm was very good. The software was able to recognize songs even when they were exposed to high levels of noise. Using hashing data structures to store the peak pairs increased the search speed by large amounts.

## 6. REFERENCES

[1] Cano, Pedro, Eloi Batle, Ton Kalker, and Jaap Haitsma. "A review of algorithms for audio fingerprinting." In *2002 IEEE Workshop on Multimedia Signal Processing.*, IEEE, St.Thomas, VI, USA, 2002, pp. 169-173.

[2] ACRCloud Docs [Online] Available: https://www.acrcloud.com/docs/acrcloud/introduction/audio-fingerprinting/

[3] Avery Wang. "An Industrial Strength Audio Search Algorithm." In *ISMIR*, vol. 2003, pp. 7-13, 2003.

[4] Samuel Kim, Erdem Unal, and Shrikanth Narayanan. "Music fingerprint extraction for classical music cover song identification." In *2008 IEEE International Conference on Multimedia and Expo*, IEEE, Hannover, Germany, pp. 1261-1264, 2008.

[5] Prem Seetharaman, and Zafar Rafii. "Cover song identification with 2d fourier transform sequences." In *2017 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, IEEE, New Orleans, LA, USA, pp. 616-620, 2017.

[6] Keansub Lee, and Daniel PW Ellis. "Detecting music in ambient audio by long-window autocorrelation." In *2008 IEEE International Conference on Acoustics, Speech and Signal Processing*, IEEE, Las Vegas, NV, USA, pp. 9-12, 2008.

[7] Wei Xiong, Xiaoqing Yu, and Jianhua Shi. "An improved audio fingerprinting algorithm with robust and efficient." *IET International Conference on Smart and Sustainable City 2013*, Shanghai, China, pp. 264-267, 2013.

[8] Jaap Haitsma, and Ton Kalker. "A highly robust audio fingerprinting system." In *ISMIR* vol. 2002, pp. 107-115, 2002.

[9] Shumeet Baluja, and Michele Covell. "Audio fingerprinting: Combining computer vision & data stream processing." In *2007 IEEE International Conference on Acoustics, Speech and Signal Processing-ICASSP'07*, IEEE, Honolulu, HI, USA, 2007.

[10] Mehryar Mohri, Pedro Moreno, and Eugene Weinstein. "Robust music identification, detection, and analysis." In *8th International Conference on Music Information Retrieval, ISMIR 2007*, pp. 135-138, 2007.

[11] Shumeet Baluja, Michele Covell, "Waveprint: Efficient wavelet-based audio fingerprinting", *Journal of Pattern Recognition*, vol. 41, no. 11, pp. 3467-3480, November 2008.

[12] Thomas C. Walters, David A. Ross, and Richard F. Lyon. "The intervalgram: An audio feature for large-scale melody recognition." In *Proc. of the 9th International Symposium on Computer Music Modeling and Retrieval (CMMR)*, pp. 295-310, 2012.

[13] Beat Gfeller, Ruiqi Guo, Kevin Kilgour et al. "Now Playing: Continuous low-power music recognition." In *31st Conference on Neural Information Processing Systems (NIPS 2017)*, Long Beach, CA, USA, 2017.