Evaluating musical fingerprinting systems

Alastair Porter

Schulich School of Music
McGill University
Montreal, Canada

December 2012

A thesis submitted to McGill University in partial fulfillment of the requirements for the degree of Master of Arts.

Abstract

Audio fingerprinting is a process that uses computers to analyse small clips of music recordings to answer a common question that people who listen to music often ask: "What is the name of that song I hear?" Audio fingerprinting systems identify musical content in audio and search a reference database for recordings that contain the same musical features. These systems can find matching recordings even when the query has been recorded in a public space and contains added noise. Different audio fingerprinting algorithms are better at identifying different types of queries, for example, queries that are short, or have a large amount of noise present in the signal. There are few comprehensive comparisons of fingerprinting systems available in the literature that compare the retrieval accuracy of fingerprinting systems with a wide range of querys.

This thesis presents an overview of the historical developments in audio fingerprinting, including an analysis of three state-of-the-art audio fingerprinting algorithms. The thesis introduces factors that must be considered when performing a comparative evaluation of many fingerprinting algorithms, and presents a new evaluation framework that has been developed to address these factors. The thesis contributes the results of a large-scale comparison between three audio fingerprinting algorithms, with an analysis recommending which algorithms should be used to identify music queries recorded in different situations.

Résumé

Le système d'empreinte audio est un procédé qui analyse de courts extraits de musique avec un ordinateur pour répondre à une question courante : « Quelle est le nom de cette chanson que j'écoute ? ». Les systèmes d'empreintes audio identifient le contenu musical d'un enregistrement et cherchent des documents sonores possédant les même traits musicaux au sein d'une base de données de référence. Ces systèmes sont capables de fonctionner même si les requêtes qui leur sont transmises sont enregistrées dans un espace public, avec de nombreuses sources de bruit extérieur. Les différents algorithmes d'empreinte audio se distinguent par le type de requête qu'ils peuvent traiter : certains se concentrent sur des requêtes de courte durée, d'autres sont optimisés pour pouvoir être performant même dans des conditions de bruit très défavorables. Dans la littérature, il existe peu d'études comparatives poussées traitant spécifiquement des performances des systèmes de reconnaissance par empreinte audio dans un large éventail de cas.

Cette thèse présente une vue d'ensemble de l'histoire du développement des systèmes d'empreinte audio. Cette thèse introduit en suite des facteurs qui doivent être pris en compte lors de l'évaluation comparative de plusieurs algorithmes pour la reconnaissance par empreinte audio. De plus, ce travail présente un nouveau cadre d'évaluation développé afin d'incorporer ces facteurs. Cette thèse combine les résultats d'une comparaison à grande échelle de trois algorithmes d'identification d'empreinte audio avec une analyse recommandant lequel de ces algorithmes est le plus efficace pour identifier la plus grande variété d'extraits audio.

Acknowledgements

Many people helped me during the process of producing this thesis and I would like to take this oppourtunity to thank all of them.

I would first like to thank my supervisor, Ichiro Fujinaga, for providing simulating work and research oppourtunities in the DDMAL research lab, and for providing valuable feedback during the time that I was working on my thesis.

Thank you to my labmates in the DDMAL lab, and to everybody in the Music Technology department at McGill who helped me to grow my interest in music research but were just as happy to discuss any aspect of music or technology.

To everyone who welcomed me to Canada and Montréal—flatmates, fellow students, and friends alike: You are all too numerous to list here, but you gave me a great few years that I'll never forget. Thank you.

Thanks to the authors of the software that I used in the evaluation for making the code available—Brian Whitman for Echoprint, Lukáš Lalinský for Chromaprint, and Dan Ellis for the landmark code. Thanks again to Dan and Lukáš, who were more than willing to answer questions that I had about implementation details in their respective software. I would also like to thank Ashley Burgoyne for his assistance in helping me to understand and present the statistics in my results, and Cécile Charvet and Bertrand Scherrer for helping me to translate the abstract for this thesis.

Finally, I would like to thank my parents who always support me no matter what country I live in.

Contents

1 Introduction **1**
1.1 Audio fingerprinting 2
1.1.1 Applications of fingerprinting 4
1.2 Requirements of audio fingerprinting algorithms 5
1.3 The audio fingerprinting process 6
1.4 Contributions of this thesis 8
1.4.1 Fingerprinting algorithms used in the evaluation 9
1.5 Organisation of this thesis 9

2 Audio Fingerprinting: An overview **11**
2.1 Related audio retrieval techniques 11
2.1.1 Query by humming 12
2.1.2 Query by description 12
2.1.3 Query by example 13
2.1.4 Similarity retrieval 13
2.1.5 Watermarking 14
2.1.6 Graphical Audio Summaries 15
2.2 Audio fingerprinting techniques 16
2.2.1 Computing codes from features 17
2.2.2 Computing codes with machine learning 19
2.2.3 Increasing fingerprinting speed 20
2.2.4 Commercial Fingerprinting systems 21
2.2.5 Fingerprinting services 22
2.3 Measuring statistics 23

2.4 Evaluating fingerprinting accuracy 24

3 Analysis **27**
3.1 Echoprint 28
3.1.1 Preprocesing and transform 28
3.1.2 Hashing 29
3.1.3 Storage 30
3.1.4 Lookups 31
3.2 Chromaprint 31
3.2.1 Preprocessing and transform 31
3.2.2 Hashing and storage 32
3.2.3 Lookups 33
3.3 Landmark 33
3.3.1 Preprocessing and transform 33
3.3.2 Feature selection 33
3.3.3 Hashing and storage 34
3.3.4 Lookups 35

4 Evaluating fingerprinting algorithms **37**
4.1 Choosing evaluation criteria 37
4.2 Evaluation framework 38
4.2.1 Distributed computation 39
4.2.2 Modularity 39
4.2.3 Repeatability 40
4.2.4 Collecting statistics 41
4.2.5 Query modifications 41
4.2.6 Evaluation process 44
4.3 Document retrieval statistics 45
4.3.1 Precision, recall, and specificity 46
4.3.2 Confidence intervals 47
4.3.3 Sensitivity 48
4.4 A comparison of audio fingerprinting algorithms 49
4.4.1 Experiment 49

4.4.2 Algorithm 1 setup: Echoprint 49
4.4.3 Algorithm 2 setup: Chromaprint 51
4.4.4 Algorithm 3 setup: Landmark 51

5 Results **53**
5.1 Query length 53
5.2 Modified queries 56
5.3 Noise 61
5.4 Discussion 66

6 Conclusion and further work **67**
6.1 Contributions 68
6.2 Further work 68

Bibliography **71**

List of Tables

4.1 Query modifications used in the experiment 50

5.1 Accuracy results for unmodified queries, three fingerprinting algorithms, and six query lengths with lower limits (LL) and upper limits (UL). 54
(a) Precision (%) 54
(b) Recall (%) 54
(c) Specificity (%) 54
(d) Sensitivity (d') 54
5.2 The expected retrieval numbers for a 30-second query and actual numbers from the three fingerprinting systems. 55
5.3 Precision for modified queries 57
5.4 Recall for modified queries 58
5.5 Specificity for modified queries 59
5.6 Sensitivity (d') for modified queries 60
5.7 Precision for queries modified with added noise 62
5.8 Recall for queries modified with added noise 63
5.9 Specificity for queries modified with added noise 64
5.10 Sensitivity (d') for queries modified with added noise 65

List of Figures

1.1 Common steps perform in audio fingerprinting algorithms to convert audio to a fingerprint. 7
1.2 The main processes of a fingerprinting system incorporating a fingerprinting algorithm, and a service to look up unknown queries and return a matching recording. 8
1.3 An example of a portion of a fingerprint from the Echoprint algorithm, consisting of a series of hash value and timestamp pairs. 8

3.1 Onsets detected by Echoprint 30
3.2 Calculating the time delta between pairs of onsets to create a hash. 30
3.3 The match filters used in Chromaprint. 32
3.4 A graphical representation of the Chromaprint for "Ziggy Stardust". 32
3.5 A "constellation map" of hash pairs generated by the Landmark algorithm for a 15 second query of "Ashes to Ashes", by David Bowie. 34
3.6 Encoding features of the hash for a peak pair into a single 20 bit integer. . 35

4.1 Contract for fingerprinting modules. 40
4.2 Chaining filters together. 42
4.3 Flowchart of the evaluation framework process. 44

Chapter 1

Introduction

Recorded music is pervasive in our society. It is broadcast over radio waves and on the Internet. It is used as a background to videos, television, and movies. Shops play it in the background while people are shopping. People carry around thousands of songs every day on portable audio players and smartphones, with access to millions more by streaming from the Internet. Personal music players can show the name of the recording that is currently playing, but when listening to music in public environments it can be a challenge to recognise every song that you listen to in a day.

It is an impressive skill to be able to listen to a small clip of music and recognise almost immediately the title, composer, or performer of the work. Sometimes though, you may not recognise the song, or it might be familiar and on the tip of your tongue, but you just can't remember the name. You may be in a cafe or shop as a song that you want to know more information about comes on, but it is not introduced or you miss the DJ's introduction. Given the ever increasing amount of music written, recorded, and performed it is virtually impossible for any one person to recognise every song. It seems suitable to delegate such a task to a computer.

Computers have been used to develop music-related applications since the time that they were first available in research environments (Downie 2003). Music information retrieval (MIR), a music-specific branch of information retrieval, is concerned with using computers to store and analyse digital collections of music in all forms (e.g., sheet music, recorded music, metadata about music) and allow queries to obtain analyses and results from them. *Audio fingerprinting* is a facet of MIR that uses computers to listen to and recognise

recordings of songs, much like people can listen to a piece of music and say what the name of the song is. In the audio fingerprinting process, a computer algorithm is used to analyse a corpus of music, identifying features that can be used to uniquely identify musical works in a corpus (an audio "fingerprint", much like fingerprints uniquely identify people). Once a corpus of audio fingerprints has been created, the same algorithm can be used to generate a fingerprint of an unknown clip of audio. The corpus can be searched for fingerprints that are the same as the fingerprint generated by the unknown query in order to retrieve information about the recording.

1.1 Audio fingerprinting

Audio fingerprinting is used to take a short sample of an unknown audio recording and retrieve metadata about the recording. It does this by converting the data-rich audio signal into a series of short numerical values (or *hashes*) that aim to uniquely identify a musical recording. Audio fingerprinting systems keep large databases of fingerprints for millions of known audio recordings. To identify an unknown audio recording query, the query's fingerprint is generated and compared to the reference database to find recordings that have identical or similar fingerprint hashes. Unlike symbolic music notation query methods, such as query-by-contour and query-by-humming (Ghias et al. 1995), which use symbolic musical information (i.e., knowledge of the instruments and specific notes played in a segment of audio), audio fingerprinting uses lower-level spectral information in a signal to generate a unique identifier of the audio.

An audio fingerprinting system should be able to recognise recordings of songs in the same way that a person can. If a person can recognise a song from a short clip of audio comes then an ideal computer system should also be able to recognise the song. This means that a hashing algorithm should utilise the perceptual aspects of the audio contained in the file and not just the way that the file itself is encoded digitally. These so-called content-based identification systems (CBIDs) are named as such because they identify matches based on the musical content of the recordings in the corpus rather than the way that the file is stored by the computer. Furthermore, because humans are able to recognise a recording as being the same even in the presence of small changes to the query, such as tempo variations or noise, effective fingerprinting algorithms should also be able to correctly identify queries that have been modified in similar ways. Haitsma et al. (2001) call fingerprinting "robust

hashing", because it indicates that a fingerprinting algorithm should generate a hash based on the input that is robust to modifications to the audio that do not dramatically alter the sound. They suggest that one approach to perform this robust hashing is to design an algorithm that approximates the human auditory system:

> A robust audio hash is a function that associates to every basic time-unit of audio content a short semi- unique bit-sequence that is continuous with respect to content similarity as perceived by the [human auditory system]. (Haitsma et al. 2001, p. 2)

In order to match a query to a recording in a reference database, a fingerprinting algorithm must generate identical hashes for the reference recording and query. The hashes must be identical even when the query is very short, or when given a recording that sounds similar to a human, but has a different audio signal. If a the recording has been made, for example, with a microphone in a noisy room then the fingerprinting algorithm should be able to separate the music in the recording from any additional noise recorded by the microphone.

Because fingerprinting algorithms should generate similar hashes on audio that sounds the same to an ear but may not be stored the same on disk, hash functions such as cryptographic hashes are unsuitable for this task. Cryptographic hashes generate a significantly different output when given a similar but not identical input. Two clips of audio that sound the same to a human could have a significantly different form when stored in a computer. Different digitisation techniques (e.g., copying to a computer from CD or from Vinyl) can result in a different representation on disk for the same song. Perceptual coding techniques such as MP3, Ogg Vorbis, and Apple AAC result in files that sound nearly identical but have completely different on-disk representations. Also, because of the desired requirement of being able to identify a segment of a recording from any point in the recording, it would be unfeasible to create a set of hashes for all song durations starting from all points in the song.

Reference databases of audio should contain fingerprint codes for the entire duration of audio. This is because it is also useful to identify a song when only a portion of the song is given as a query. This segment could be recorded from any point in the song, especially if the recording is made in a public place from music that is being played over a PA system.

On the other hand, fingerprint lookup systems do not need to store a copy of the audio that is used to create the fingerprint. Fingerprints can be generated and submitted to a database by any person with a copy of the audio. While commercial fingerprinting systems can get fingerprints directly from the music distributors, it is also possible to obtain fingerprints for rare music, out of print music, or music released through independent labels directly from people who have copies of the audio.

1.1.1 Applications of fingerprinting

Audio fingerprinting is useful to a variety of people who create and consume music. In addition to be able to identify unknown songs, an audio fingerprinting system can be used to determine proof of ownership, to track music as it is distributed to consumers over the radio or other distribution systems, or to add value for consumers (Gomes et al. 2003).

Consumer playback devices can check the fingerprint of an audio signal to determine if it is authorised to play that signal. Music distributors, for example, bulk CD copying companies, can ensure they are not unknowingly duplicating audio for which a customer does not have a license to copy.

Broadcast monitoring can be performed to create an accurate list of the audio that was broadcast by a radio station, for the purposes of creating a list of songs that are broadcast. This list can be used to ensure that royalties are correctly payed to artists whose music is played. Batlle, Masip, and Guaus (2002) call this process "song spotting", a process that involves first separating songs from other non-musical content in an audio stream (e.g., separating songs from DJs talking or advertisments) and then performing audio fingerprinting on the music segments to recognise the artist and song title. A similar technique to song spotting is used in online services such as the YouTube video sharing site[1]. The automated rights-verification system finds music that is used in the background of videos and checks to see if the owner of the music has registered it in a reference database. YouTube, for example, allows rights-holders to choose an action if unlicensed audio is used: remove the audio from the video, allow the video to remain, or allow it to remain with the addition of advertisements, the revenue of which is shared between YouTube and the rights-holder[2].

[1] `http://youtube.com`
[2] `http://www.youtube.com/t/contentid`

Audio consumers can use fingerprinting to access value-added services related to the songs being fingerprinted. A common use of fingerprinting allows consumers to quickly and accurately copy music onto their computer from physical media and name it correctly for storing and searching. Software on a computer can generate a fingerprint of each track of music as it is copied off a CD and quickly look up artist and title metadata from a central database. Mobile phones and personal devices have let people be closer to information about things happening around them, including details of songs that are playing in public or private venues. Smartphone apps, such as Shazam[3] and Soundhound[4] let people use their phone to record a small segment of a song and almost immediately return information about it, including title, artist, song lyrics, related audio, upcoming concert appearances or where to buy a copy of the song online. These services can identify songs that are playing over speakers in venues such as cafes or bars, even if there is other background noise such as people talking.

A 2001 press release by the Recording Industry Association of America and the International Federation of the Phonographic Industry (R.I.A.A. 2001) asks for proposals for fingerprinting systems to be used in rights management, copyright enforcement, and consumer audio tasks. Suggested examples for uses of fingerprinting systems included tracking music played by terrestrial and Internet radio stations for distributing royalties, checking if an audio file has the right to be transmitted over a network, and providing value-added data and other promotions to people listening to certain recordings.

1.2 Requirements of audio fingerprinting algorithms

The primary purpose of an audio fingerprinting system is to identify metadata about a song based on a short segment of the song's audio signal. Haitsma et al. (2001) and Cano et al. (2005) each describe a list of desirable retrieval criteria that a fingerprinting system should fulfil:

- A system must be able to generate compact fingerprints that can be quickly located in a reference database, with a low error rate. The reference database must able to be updated easily with fingerprints of newly released or digitised audio.

[3] `http://shazam.com`
[4] `http://soundhound`

- Fingerprinting systems must be able to identify audio even if it has been altered in different ways. Some common alterations that a system should be able to recover from include ambient noise (for example, if a recording was made in a public place or while driving in a car), and transmission interference (a reduction in quality or increase in noise due to the medium which the audio is transmitted through, for example, FM radio or GSM cellphone networks).
- Recordings should be able to be identified even if the audio has been modified before it was broadcast. For example, frequency equalisation, or audio compression may be applied by radio stations or a home stereo system (Cano, Batlle, Mayer, and Neuschmied 2002). Resampling of audio results in a speed-up or slow down of the audio which results in an associated increase or decrease in the pitch of the signal.
- A fingerprinting algorithm should be reliable. It should minimise the number of false positives returned to any query (ideally false positives should be zero). It should be robust enough to detect audio correctly even if the query has been distorted during recording.
- A fingerprinting algorithm should be granular, that is, able to correctly identify a song from just a small segment of audio. This segment could come from any part of the song, not just the beginning.
- A fingerprinting algorithm must be versatile enough to compute a fingerprint from any format of audio signal stored on a computer.
- The fingerprint should be computationally inexpensive to generate, and adding new songs to the database should not result in any perceptible decrease in the speed at which lookups can be performed.

1.3 The audio fingerprinting process

Most audio fingerprinting algorithms follow a common sequence of steps when transforming an audio signal to a fingerprint (Figure 1.1): preprocessing, framing and overlap, transform and analysis, feature extraction, and fingerprint generation (Cano 2007).

The first step, preprocessing, converts all input signals into a common format for analysis by the algorithm. Often, this step involves converting the input to a mono signal and lowering the sampling rate from the standard CD rate of 44,100 Hz. The exact process differs for different algorithms. Next, an algorithm takes the time-series audio signal

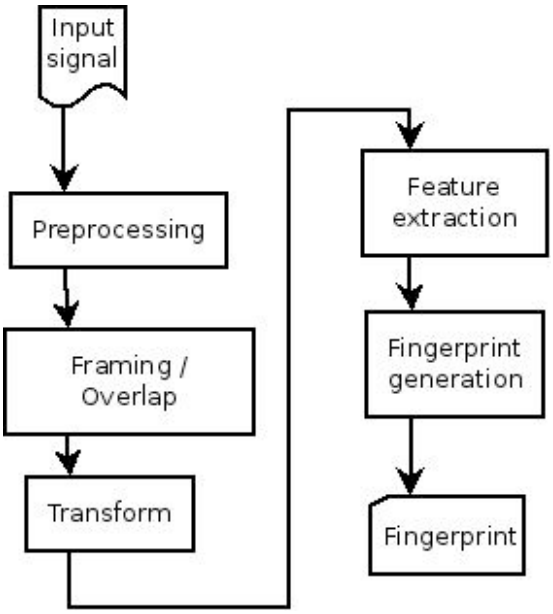

Figure 1.1 Common steps perform in audio fingerprinting algorithms to convert audio to a fingerprint.

and converts it into a frequency-domain signal from which more information can be extracted. Framing and overlap determines how many samples to consider when calculating a transform of a time-domain signal. Each frame has a window applied to it to assist in calculations, and the frames are processed in overlapping chunks from the time-series signal. The feature extraction process takes the signal that has been converted into the frequency domain and selects salient features that are used to characterise the audio. Finally, once the features have been chosen and extracted from the signal they need to be converted into a fingerprint representation that can be stored in a database and compared to unknown query signals.

For a full fingerprinting software suite, the procedure does not end at the fingerprinting algorithm (Figure 1.2). Once the fingerprint has been generated, it must be stored in a reference database. The numerical representation of the fingerprint is usually too unwieldy to be used as an identifier (Figure 1.3), so a smaller unique identifier is used. This could be as simple as the artist and song name, or a short unique string. A fingerprinting system will provide a lookup service. This lookup operation should be able to take an unknown input query and match the query's fingerprint with a fingerprint that is in the reference database, returning the identifier of the song that the query best matches, and optionally,

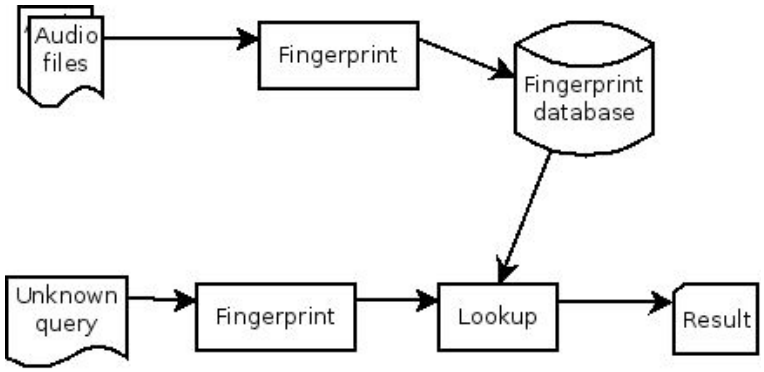

Figure 1.2 The main processes of a fingerprinting system incorporating a fingerprinting algorithm, and a service to look up unknown queries and return a matching recording.

the location in the song that the query comes from. If the song being looked up is not in the reference database, the fingerprinting system should report that the song is not in the database, rather than giving a wrong answer.

```
168069 13 465942 13 52579 13 558476 13 739869 13 741460 13 380305 14
415399 14 661073 14 709215 14 74563 14 82703 14 1002532 15 1030366 15
106211 15 187156 15 348044 15 351350 15 35265 15 395259 15 403763 15
45438 15 474191 15 557925 15 793952 15 860815 15 883227 15 887181 15 90861
15 971810 15 1000650 16 405802 16 664990 16 771321 16 80513 16 949484
16 414620 38 47256 38 620886 38 63360 38 806318 38 971075 38 1007816 39
1014266 39 1022230 39 1036491 39 224005 39 340504 39 342618 39 394503 39
```

Figure 1.3 An example of a portion of a fingerprint from the Echoprint algorithm, consisting of a series of hash value and timestamp pairs. The full fingerprint for this 3 minute, 35 second long song is 58 kilobytes in size. The Echoprint server internally refers to this song with the identifier TRMQSLA132F3989B92

1.4 Contributions of this thesis

This thesis provides a survey of the current state of the art in audio fingerprinting algorithms. It gives an overview of many current algorithms and also provides an in-depth study of three algorithms that are currently in widespread use commercially and in research projects.

We perform an evaluation of the three algorithms, and present statistics on the accuracy of the three algorithms for performing fingerprint lookups on a large collection of music. The audio queries are made to the fingerprinting systems are modified to simulate different scenarios that may be encountered when performing a fingerprinting lookup, such as short queries, damaged audio, degraded audio, and audio mixed with a noisy environment. This broad evaluation directly compares different fingerprinting algorithms in identical situations. As part of the evaluation, we have developed an extensible evaluation suite. This suite allows fingerprinting algorithms to be tested repeatedly under the same circumstances and collects results of each evaluation and helps in generating statistics. This evaluation platform can be used by other researchers to test new fingerprinting algorithms in a controlled environment.

1.4.1 Fingerprinting algorithms used in the evaluation

There are a large number of fingerprinting algorithms that have been developed. Different algorithms perform the fingerprinting and identification steps differently, and have different strengths in recognising types of music. For the evaluation in this thesis we chose three algorithms that use different techniques to generate a fingerprint. We chose these specific algorithms for two reasons. The first reason is because they all use significantly different techniques for generating a fingerprint. The second reason is because each algorithm is freely available to download and run on a server. By running our own version of the server we are able to carefully control the audio that is added to the database and so can tell if the result returned by the algorithm is correct or not. The algorithms are: Echoprint (Ellis et al. 2011), Chromaprint (Lalinský 2012), based on the algorithm presented by Ke, Hoiem, and Sukthankar (2005), and a landmark hashing algorithm (Ellis 2009), based on (Wang 2003). The algorithms are all used actively in commercial (Echoprint), community (Chromaprint), and research (Landmark) environments. Each fingerprinting algorithm has freely available source code for both the fingerprinting component and the lookup system, which we make use of in the evaluations.

1.5 Organisation of this thesis

The rest of this thesis is organised as follows: Chapter 2 performs a review of the history of audio fingerprinting and other closely related technologies. Current commercial and

research uses of fingerprinting are also discussed. Chapter 3 provides an in-depth review of the three algorithms chosen for this experiment. The review covers the signal processing specifics about how the fingerprinting algorithm generates its fingerprints and how lookups are performed. Chapter 4 describes the experiment designed to compare the accuracy of retrieval of these three algorithms. The results for the experiment are described in Chapter 5. An overview of the thesis, experiment, and concluding remarks are given in Chapter 6.

Chapter 2

Audio Fingerprinting: An overview

Computers have been used for music-related retrieval tasks for almost as long as they have been available for research (Foote 1998). Audio fingerprinting, however, is a fairly recent application of signal processing techniques to music information retrieval. Up to the end of the 1990s, there was little research on audio fingerprinting. Instead most music information retrieval research centered around symbolic retrieval of music, and automatic classification of music based on style or instrument.

This chapter begins by giving an overview of music information retrieval technologies which contain aspects that were later developed into audio fingerprinting systems. It continues with an outline of the audio fingerprinting process, with examples of different techniques used by a number of different researchers. It concludes with an overview of current commercial applications of audio fingerprinting.

2.1 Related audio retrieval techniques

Music information retrieval techniques have increased in scope and complexity since computers were first used in connection with music-related applications. Often the complexity of techniques increased as the power of contemporary computers allowed more interesting musical features to be calculated. This section describes a number of information retrieval techniques that were developed either as precursors to audio fingerprinting, or use similar signal processing techniques to perform analysis of audio.

2.1.1 Query by humming

Query by humming (QBH) (Ghias et al. 1995) is a technique that lets a person get information about an unknown song by humming or singing the main melody of the song. A QBH system transcribes the query into symbolic form and then searches for the melody in a database of symbolic song melodies. By representing the melody as a string of characters, existing fuzzy string matching techniques can be used to find partial matches of melodies in a reference database. The effect of mistakes in the sung query can be reduced by representing melodies and search queries as contours–a string of characters indicating only if each note is higher, lower, or the same as its preceding note. Even with these fuzzy matching techniques, QBH systems can encounter problems if the melody as sung by a person is sufficiently different from the stored melody, or if the searcher sings a melody that is not stored in the database (Byrd and Crawford 2002). Automatically transcribing an audio signal into a score, and finding a salient melody in a polyphonic score are still active research problems (Poliner et al. 2007). Song, Bae, and Yoon (2002) reduce some of the requirements to accurately transcribe a sung query by using what they call a "mid-level melody representation". This representation of a melodic phrase describes a query by its spectral content than transcribing the audio signal into symbolic notes which allows for a less-exact matching system to be used, reducing the effect of incorrectly sung queries.

2.1.2 Query by description

An early method of retrieving audio data from a computer database was query by description. In this kind of system, a database holds a list of audio files along with a textual description of each clip. These systems are able to retrieve music both by description of the sound itself (e.g., thunder or applause) and by a description of the the music (e.g., music with a saxophone and piano playing). These type of systems are implemented as a text search database. For example, to find all pieces of audio with a saxophone, such a system would simply find all occurrences of a tag "saxophone". Early versions of these databases were manually annotated, that is, the description of each audio clip was entered by a person (Foote 1997). As databases of music grew larger, it began to take longer and longer for expert listeners to label sounds. At the time the development of social tagging on the Internet had yet to be developed, and so what now might have been solved by crowdsourcing was not an option. A problem with manual annotation of audio is that descriptions can

be subjective, and use different words to what was used in a query. For example, an audio clip tagged with the word "sax" may not be returned for the query "saxophone".

2.1.3 Query by example

As databases grew too large to annotate manually, computer systems were developed to automatically identify the sounds in audio clips and identify either the instrument or the type of noise (e.g., door slam, scream). Wold et al. (1996) and Pye (2000) use Hidden Markov Models (Rabiner and Juang 1986) to create models of each sound. When presented with an unknown sound, the same analysis is performed and the model is used to predict a label for the sound. Wold et al. (1996) take clips of audio and measure the loudness, pitch, brightness, bandwidth, and harmonicity of the audio signal. This information is first used to determine if there is more than one sound in the clip, with a large sudden variation in these measurements can indicate a new sound. Once individual sounds have been isolated, the system is trained by taking these features and a manually provided description of the sound. The system learns what values of each feature correspond to each description. Subramanya, Simha, Narahari, and Youssef (1997) identify features after transforming the audio with the Harr transform, discrete Fourier transform (DFT), and discrete cosine transform (DCT). In experiments, the DCT was found to create features that most accurately identified audio clips.

2.1.4 Similarity retrieval

Cover song detection algorithms find different recordings of songs. These different recordings could be performed by the same performer (e.g., studio and live recordings), by different performers, or could be "radio edits"—a modified version of a recording made for playback on commercial radio.

Cover song detection systems share some signal processing techniques with fingerprinting systems. They often use similar methods to reduce the dimensionality of input audio and then characterise the spectral content as a numerical hash. A similarity detection system then performs a similarity measure on the generated hashes. These matching algorithms allow more variation in these hashes than fingerprinting algorithms (Miotto and Orio 2008; Foote 2000). Similarity measure systems often take into account the entire struc-

ture of a song, rather than using a short 10–20 second query as a fingerprinting systems do.

Detecting different recordings of Western classical music can be considered a special case of cover song detection. Because each recording of a Western classical work(e.g., a movement of a symphony) is performed from the same score, the overall structure of the audio is the same regardless of what orchestra records the music. While two different recordings of the same score may be played at different speeds, points in the score are time-invariant, occurring at the same relative location in each recording. distance between points in the score remain the same. Yang (2001) matches Western classical music by identifying spectral peaks in a recording. Recordings by different orchestras should have peaks that are positioned with the same ratios of distances between them. Other classical music matching systems treat this detection as a specialised subset of cover song detection (Crawford, Mauch, and Rhodes 2010; Müller, Kurth, and Clausen 2005).

2.1.5 Watermarking

Audio watermarking is the act of inserting hidden information into an audio stream, that cannot be detected by the human ear. It can be used for many of the same uses as fingerprinting, for example, discovering the copyright status of a song, or providing metadata for a given recording. Additional metadata can be included in a watermark, including both information about the work or artist, or additional information (e.g., news broadcast from a radio station along with a song). Audio watermarking techniques are able to store data in an audio stream at rates of up to 150kbps, providing enough space to include metadata about the currently playing song, or even artwork.

Because watermarking physically affects a signal, it is important to choose a technique that is inaudible to a person. This suggests adding the watermark to a part of the auditory spectrum that is too high for most people to hear. A disadvantage of using this part of the spectrum is that many perceptual coding techniques such as the MP3 format remove parts of the spectrum that are inaudible to people. Therefore a good watermarking algorithm must be robust against encoding formats that modify the audio signal.

Unlike audio fingerprinting, watermarking does not require a centralised lookup database. This means that devices that check for the validity of streams do not need to have either a local database of fingerprints (that could quickly get out of date), or an Internet connection

to connect to a central server that compares fingerprints. The advantage of watermarking is that there is no central server that must remain operational in order for other systems to run.

Fingerprinting has an advantage to watermarking in that it can be applied retroactively to a library of audio. An effective watermarking system must be developed before the first piece of audio is released with embedded watermarks, and improvements cannot be applied to audio that has already been released.

Watermarking and fingerprinting can be combined for verification purposes. Gomez et al. (2002) use an existing audio fingerprinting technique (Neuschmied, Mayer, and Batlle 2001) to generate a fingerprint of an audio file and then embed that fingerprint in the audio file itself as a watermark. This technique allows a playback system to verify the integrity of an audio file before it is played and ensure that it has not been tampered with.

Watermarks that guarantee the authenticity of an audio file, need to be digitally signed in order to prevent tampering. By signing a large number of signals with the same key, a malicious attacker may be able to derive components of the key by statistical analysis of many signals. Mıhçak and Venkatesan (2001) present a solution that uses a component of the audio's fingerprint as part of the key to make deriving the key more difficult.

Gomes et al. (2003) compare fingerprinting and watermarking and discuss the potential applications for both techniques. They suggest that while both fingerprinting and watermarking can both be used for copyright protection purposes, watermarking can be more versatile because it can store any kind of information in the watermark, whereas fingerprinting can only identify a recording. They state that watermarks need to have a low energy so that they are not audible in a recording. This makes them more prone to distortion and unable to be read if the audio file is damaged or encoded using low bitrate audio formats. Fingerprinting is a more robust method for identifying a recording. Because the reference database is separate from the audio, unlike watermarks, it can be updated at any time with new information.

2.1.6 Graphical Audio Summaries

Graphical audio summary algorithms generate a graphical image that uniquely represents the structure of a recording. Like audio fingerprints, these summaries can be used visually

identify recordings without listening to the audio of the recordings. Images are more easily comparable by people than the numerical representation that an audio fingerprinting algorithm generates. Audio summary algorithms generally use the same computation steps as a fingerprinting algorithm, but produce a graphic as their final output representation (Yoshii and Goto 2008). Bartsch and Wakefield (2005) generates auditory summaries of music by finding repeating phrases such as a chorus and renders it to audio.

2.2 Audio fingerprinting techniques

This section presents an overview of existing fingerprinting algorithms. An analysis of the three fingerprinting algorithms evaluated in this thesis (Ke et al. 2005; Wang 2003; Ellis et al. 2011) is presented in Chapter 3.

Section 1.1 introduced the idea that audio fingerprinting systems should recognise music in a manner similar to that which people use to recognise music. In order to perform this recognition, fingerprinting algorithms must extract meaning from the audio signal. One way to extract meaning is to compute *features* of the audio, where a feature could be some element with musical meaning (e.g., the pitch of musical content, or the rhythm of the music), or could be based upon computed qualities of small segments of audio. Different fingerprinting algorithms extract different types of features, and the specific algorithms will therefore have different characteristics based upon the nature of the features computed from an audio signal.

Fingerprinting algorithms convert features in to numerical codes, or hashes, that represent the value of a feature. If similar sounding features generate the same hashes then the fingerprinting system can find matching audio by identifying identical codes.

Most audio fingerprinting algorithms calculate features from the audio domain, usually using the Fourier transform. The short-time Fourier transform (STFT) calculates the frequency content at fixed time points along the audio signal, generating analysis frames. Features are often calculated on a per-frame basis. It is common for algorithms use features that reflect music or the way people hear music, and so usually analyse the spectrum in logarithmically spaced bands.

This section describes the ways that different algorithms convert audio features in to numerical fingerprints. The first part describes algorithms that compute codes directly

from the values of the features. The second part introduces algorithms that use machine learning methods to convert features to hashes.

2.2.1 Computing codes from features

Amplitude

Papaodysseus et al. (2001) present a system designed to work on radio broadcast audio that does not have any additional noise present in the signal. This system works by splitting the audio into frames and taking the discrete Fourier transform. The spectrum of each frame is split into 48 exponential bins. From these bins, a "band representative vector" is created, with each element containing a 1 if there is a spectral peak in that frequency bin and a 0 if not. To determine if a query exists in the database, a band representative vector for a query is calculated and the database is searched. Matches are identified if band representative vectors only differ in 2–3 bits (when a spectral peak may not exist in a query where it does in the database. Band vectors compared by performing a bitwise *and* operation between two vectors. The algorithm can match queries to a reference database even if there is a small increase in speed (up to 4%) or frequency boosting in the audio query.

Wang (2003) describes the algorithm that is used in the Shazam music recognition service. The algorithm takes a short-time Fourier transform of the audio and selects a "constellation" of spectral peaks that have an amplitude that is larger than the peaks in a surrounding area. The time and frequency distances between pairs of these peaks is encoded into a hash that represents the audio. This algorithm is discussed in more detail in Section 3.3.

Baluja and Covell (2008) use a wavelet transform (Graps 1995) to convert an audio signal into the time domain. They then use a similar method to Wang (2003) to generate fingerprint codes from the distance in time and frequency between nearby audio peaks.

The Philips algorithm and improvements

Haitsma, Kalker, and Oostveen (2001) develop an accurate fingerprinting algorithm that has become well known for its accuracy, gaining the name of the "Philips algorithm" after the research group that they worked at. First, each audio frame is converted into the frequency spectrum using the Fourier transform. The spectrum between 300Hz and 2000Hz

is then split into 33 bands according to the bark scale, where each band has the bandwidth of a musical tone. The difference in energy between each band is encoded as a 0 or a 1 depending on if the energy between the bands increases or decreases. This results in a 32-bit value per sub-fingerprint. A fingerprint is made up of 256 sub-fingerprints, corresponding to about 3 seconds of audio. A custom index system is used to provide fast lookups of queries into a reference database. Each sub-fingerprint hash in the query is looked up in the reference database, and all tracks that contain this hash are added to a candidate list. The candidate match that has the lowest bit error rate to the query fingerprint is given as the matching track. This lookup system requires there to be at least one sub-fingerprint between the query and reference fingerprint, otherwise it will not return a match.

Aspects of this general algorithm have been improved and analysed by numerous publications. Doets and Lagendijk (2004) and Balado, Hurley, McCarthy, and Silvestre (2007) both perform theoretical analyses of the algorithm to generate models of it, the first to characterise how the output changes with the audio is compressed, and the second to determine the upper probability that no sub-fingerprints of differing audio share an identical hash. Kashino, Smith, and Murase (1999) and Kimura, Kashino, Kurozumi, and Murase (2001) present improved lookup times for searching for candidate matches. Miller, Rodriguez, and Cox (2005) increases lookup speeds by building a 256-ary tree for fast lookups—with one branch for each candidate sub-fingerprint. Liu, Cho, Yun, Shin, and Kim (2009) shows how with heavily distorted query signals there may not be at least one identical hash between the reference recording and the query. In this situation, a match would not be found. To increase the chance of a match in this situation, the new proposed algorithm generates two separate hashes using different techniques. The second hash is created by taking the discrete cosine transform (DCT) of the initial sub-fingerprints. These "metahashes" are stored in another hash table and both hashes are used to select candidate tracks. Results show that with especially noisy query signals, the dual-hash system correctly identifies a larger number of queries.

Onsets

The Echo Nest Musical Fingerprint (ENMFP) (Ellis et al. 2010) and Echoprint (Ellis et al. 2011) use the difference in time between musical onsets (generally equivalent to the beginning of notes) Echoprint calculates onset features in eight evenly spaced frequency

bands from 0–5512Hz. ENMFP calculates onsets by using a comb filter to perform beat detection Jehan (2005). The time difference between pairs of onsets is combined with the frequency band that the event occurs in to create a hash. More details about the fingerprint and lookup process of Echoprint are presented in Section 3.1.

2.2.2 Computing codes with machine learning

Some audio fingerprinting systems use machine learning methods to convert audio into fingerprint codes. These systems have a "training" process where a machine learning model is used to map musical sounds to identifiers. When performing a lookup process the query audio is processed in the same way and the generated model is used to predict the identifiers that create the audio signal. The identifiers can be compared to the identifiers created using the import process.

Kastner et al. (2002) converts audio to the frequency domain and then calculates the spectral flatness of each frame. Spectral flatness characterises how 'tone-like' a sound is, compared to noise. The algorithm uses the Vector Quantization (VQ) pattern recognition method to cluster similar flatness values together. Queries are matched by finding the nearest neighbours of the flatness values generated by the query audio and are matched the reference track that gives the most number of closest matches. Allamanche, Herre, Hellmuth, Fröba, Kastner, and Cremer (2001) creates a VQ model in the same manner as Kastner et al. (2002) but use a vector of psychoaccoustic features to represent each frame, including loudness, flatness, and sharpness. The spectral flatness method described by Kastner et al. (2002) is formally specified in the MPEG7 standard (Chang, Sikora, and Purl 2001), a content description standard published by the Motion Picture Experts Group (MPEG).

Batlle, Masip, and Guaus (2002) use MFCCs (Mel-frequency Cepstral Coefficients) as a method of extracting features and create models of stored audio using hidden Markov models (Rabiner and Juang 1986). In order to reduce the effect of noise on the audio query, they model noise using a linear filter and apply its inverse to the audio query before generating a fingerprint. The HMMs are used to create a sequence of states that represent frames of the recording. A full recording is represented by an ordered set of states. To find a match, a query is split into frames and the most likely state to create each segment is

calculated using the Viterbi algorithm. The model can also be used to find if recordings have been edited (sections removed) or if two different recordings have been mixed together.

Cano et al. (2002) present a system called AudioDNA which uses sequence matching methods from biology and text searching. Audio is converted into the frequency spectrum with a Fourier transform the spectrum is split into MFCCs. An HMM is trained to represent similar MFCCs with one of 32 'genes'. A clip of audio is represented by a sequence of these genes. Queries are matched by creating a gene sequence by using the HMM to estimate genes for each MFCC and are then compared to the gene sequences in the reference database.

Ke, Hoiem, and Sukthankar (2005) build on the matching technique developed by Haitsma, Kalker, and Oostveen (2001). The method uses a machine learning algorithm called Adaboost, with an iterative thresholding algorithm to find the best way to create black and white image masks that can be applied to a spectral representation of the audio. These image masks are created outside of the fingerprinting process, and are part of the algorithm. Once black and white masks are applied to the audio file then matches are performed by finding subfingerprints with low bit-error rates. A variation of this algorithm, Chromaprint, is discussed further in Section 3.2.

2.2.3 Increasing fingerprinting speed

Speeding up fingerprint calculation

Transforming audio content into the frequency domain is a computationally expensive process. Some algorithms use methods to reduce the computation time needed to generate the frequency bands. Pye (2000) skip the transform step by performing fingerprinting on MP3 files. MP3 encodes information about each frequency separately, so this data can be directly extracted out from the file. Seo, Haitsma, and Kalker (2002) use a Fourier-Mellin transform, which is faster to compute than a Fourier transform. The Fourier-Mellin transform also results in fingerprint that is robust to speed increases by up to 10%. The fingerprint hash is calculated in a similar manner to Haitsma and Kalker (2002). Papaodysseus et al. (2001) uses an adaptive FFT, which uses the results from the previous frame's FFT calculation as a starting point for the current frame's transform.

Speeding up lookups

Audio fingerprinting systems need to perform lookups of queries quickly. As more files are added to the reference database they should remain fast. An approach to maintaining a fast lookup speed while performing query lookups is to split up the reference database over many machines and send the query to all database machines simultaneously. Each lookup machine will return the best match given the contents of that machine's database, and then the best match of all results is chosen and returned to the client. Mahedero et al. (2004) use a cluster of machines communicating using CORBA to perform a distributed search and collate results. They use their previously published fingerprinting algorithm (Batlle, Masip, and Guaus 2002) to perform the fingerprinting process. Shrestha and Kalker (2004) integrates a distributed fingerprinting system using the algorithm developed by Haitsma and Kalker (2002). This system integrates a distributed peer-to-peer filesharing application that has knowledge of the content being shared between peers. The system can request an identification of a music file and if another node on the network reports it as being copy protected the file is not transferred.

2.2.4 Commercial Fingerprinting systems

Systems that accurately perform audio fingerprinting are useful in a commercial context. As has already been discussed, there is lots of interest in using fingerprinting in systems to monitor the distribution of audio, and for verifying that audio is authorised to be played.

An early application to enter into the digital music scene, Napster, was ordered as part of a court settlement to integrate fingerprinting into its application. Napster was a peer-to-peer file sharing application that allowed computer users to download MP3s of songs from other users on the network. Napster implemented the Relatable[1] algorithm in their software (Cremer et al. 2001).

In a similar copyright infringement detection system, Google uses both audio and video fingerprinting on its video sharing site, YouTube. YouTube lets any person with an account upload videos to be watched by anyone else who visits the site. In response to complaints by the music industry about the use of copyrighted material, Google released a content identification system that recognises when a copyrighted song is being played as the audio track of a video. The YouTube system lets rights holders determine the action to be carried

[1] `http://www.relatable.com`

out if a copyrighted song is put on a video: silence the audio; place advertisements on the video and take part in the revenue stream generated; remove the video from YouTube completely; or take no action[2].

In consumer-facing applications, audio fingerprinting has found success in recording identification in public areas. The Shazam[3] and Soundhound[4] smartphone applications record a segment of a song playing in a public area, such as a cafe or on a radio and return information about the song being played. Early versions of Shazam recorded audio over the cellphone network, and returned metadata in a text message to the calling phone. This required a fingerprinting algorithm that was robust to not just ambient noise, but also to the degradation present in GSM phone codecs (Wang 2006). Shazam was developed before the use of smartphones, and was initially designed to record audio over a phone call. Because the transmission of voice calls is designed to efficiently carry voice and not music the algorithm needed to be robust to the significant reduction in signal bandwidth that the phone network imposed. With the release of these applications on smartphones, the fingerprint can be calculated on the device, though the recording quality is still low because of the device microphones. Shazam and Soundhound also provide additional services over song identification, for example related information about the artist and song, song lyrics, and links to purchase the song from an online store.

Other companies that provide commercial fingerprinting services include Gracenote[5], Relatable[6], and Amplifind Music Services[7].

2.2.5 Fingerprinting services

In non-commercial systems, audio fingerprinting has found a use among music fans and consumers. The online music encyclopedia MusicBrainz[8] aims to create a comprehensive list of all recorded music and musicians. It contains mappings from two fingerprinting systems, PUID and Chromaprint, to recordings that are listed in its database. Fingerprints and mappings are contributed by volunteers who generate fingerprints from recordings copied

[2] `http://www.youtube.com/t/contentid`
[3] `http://shazam.com`
[4] `http://soundhound.com`
[5] `http://www.gracenote.com`
[6] `http://www.relatable.com`
[7] `http://www.amplifindmusicservices.com`
[8] `http://musicbrainz.org`

from CD, or otherwise recorded into a digital format. The result of this mapping can be used to accurately update metadata in audio files in an automated manner. Track metadata can be looked up by the fingerprint of a file, reliably identifying an unknown file. Musicbrainz originally used Relatable's TRM fingerprinting service[9], before switching to PUID (Holm and Hicken 2006) in 2006. Recently (2012) they have changed to AcoustId[10], an open-source implementation of Ke, Hoiem, and Sukthankar (2005).

Last.fm[11] provide a service that lets music listeners report in real time what song they are listening to ("scrobbling"). The site provides music listening recommendations based on collaborative filtering using the large number of listener contributions. Last.fm developed a fingerprinting service in order to more accurately identify songs as members were listening to them, rather than needing to rely on potentially error-prone metadata[12].

More recently, The Echo Nest[13] released two fingerprinting systems, ENMFP (Ellis, Whitman, Jehan, and Lamere 2010), and the open source Echoprint (Ellis, Whitman, and Porter 2011), which is discussed in further detail in Section 3.1, for use by music developers to accurately link audio and music metadata in the Echo Nest developer ecosystem. The fingerprinting services allow developers to upload a recording's fingerprint and gain access to metadata, analysis information, and cross-referenced metadata to a number of other music services on the Internet. The fingerprinting algorithm and software for the lookup server are released under open-source licenses, allowing developers to set up separate fingerprinting servers for private use.

2.3 Measuring statistics

A fingerprinting answer can give the correct answer to a query or it can give the incorrect answer. The accuracy of fingerprinting systems can be evaluated by calculating the proportion of correct answers that they provide. The retrieval of a fingerprinting system is measured as a fraction of how many queries it successfully identifies out of all queries that were given to it. This is one of the most common rates reported in the literature (Kastner et al. 2002; Wang 2003; Baluja and Covell 2007; Jang et al. 2009; Fenet et al. 2011)

[9] `http://www.relatable.com`
[10] `http://acoustid.org`
[11] `http://last.fm`
[12] `http://blog.last.fm/2007/08/29/audio-fingerprinting-for-clean-metadata`
[13] `http://the.echonest.com`

Cano, Batlle, Mayer, and Neuschmied (2002) and Jang, Yoo, Lee, Kim, and Kalker (2009) report only the number of errors made, including errors where a query was not retrieved when it should have been (false negative) and where the wrong recording was retrieved from a query (false positive). The rate at which false negatives and false positives occur are also called type I and type II error rates, respectively (Lutz 2009). As well as a single accuracy rate, Fenet et al. (2011) report the false alarm rate—how often the system gives a result when the query was known to not be in the database.

Some fingerprinting systems can be configured to trade off the number of correct results to the number of times that it does not return an answer. To visualise this tradeoff between recall and precision as a parameter of the system changes, the Receiver Operator Characteristic (ROC) curve can be used. Covell and Baluja (2007) and Chandrasekhar, Sharifi, and Ross (2011) compare the accuracy of different algorithms as the accuracy rate is changed.

Ellis, Whitman, and Porter (2011) Present a weighted metric, P_{err} which measures the probability that the retrieval algorithm will make a mistake when given a query. It gives more weight to false positives than to false negatives, with the rationale that giving an incorrect answer is worse than reporting that a query is not in the database.

2.4 Evaluating fingerprinting accuracy

For fingerprinting algorithms that are resilient to noise in the query signal, there is a common set of alterations that simulate the effect of noise on queries in order to test the robustness of an algorithm. The alterations should be representative of real-life degradations that could be applied to a query signal in fingerprinting situations. Haitsma, Kalker, and Oostveen (2001) show a comprehensive list of signal degradations, which include

- Downsample to a lossless codec (e.g., MP3) at 128kbps and 32kbps bitrates
- Apply an allpass filter
- Apply audio compression
- Apply equalization
- Add echo to the query
- Pass the query through a bandpass filter
- Resample the query to a different bitrate (and therefore change the pitch)

- Introduce noise by converting to analog (recording to tape) then convert back to digital

Herre, Allamanche, and Hellmuth (2001) perform a similar list of alterations, including changing the amplitude of the query signal (both by a constant value, and with compression), resampling the query with a speed change, performing EQ, encoding as MP3, and adding background noise.

Chandrasekhar, Sharifi, and Ross (2011) measure the suitability of fingerprinting algorithms for mobile devices, for example, the effect of encoding a query using the GSM audio codec, which is often used by cellphones. Their comparison of three different fingerprinting algorithms only measures a fingerprinting algorithm's suitability to being used in a mobile device, and thus concentrates on statistics like computational power required to calculate the fingerprint, and the size in bytes of the fingerprint for transmission.

Chapter 3

Analysis

This chapter presents an overview and analysis of the three audio fingerprinting systems that are evaluated in Chapter 5 of this thesis.

These algorithms were chosen because of their availability and reputation in the academic and audio communities. Each algorithm has a freely available implementation which can be run as a standalone server. This ability to run a server independently from any hosted service was an important requirement as it allowed us to host our own servers containing a known dataset of audio that we could use to verify the results of a fingerprint lookup from.

We describe the fingerprinting, storage, and lookup processes for the following fingerprinting algorithms:

- **Echoprint**: A freely available audio fingerprint algorithm[1] and server[2] created and released by music technology company The Echo Nest, described by Ellis et al. (2011). The algorithm is designed to be robust against noise in queries and the server is scalable to support over 50 concurrent queries per second. Both the fingerprinting algorithm and the server are made available under the open source licenses (MIT and Apache 2.0, respectively). The Echo Nest hosts a fingerprint server that performs over 5 million lookups per day[3].

[1] `https://github.com/echonest/echoprint-client`
[2] `https://github.com/echonest/echoprint-server`
[3] `http://notes.variogr.am/post/27796385927`

- **Chromaprint**: An implementation of the algorithm described by Ke et al. (2005), with some features from Jang et al. (2009) and Müller, Kurth, and Clausen (2005). The Chromaprint system is used by the Musicbrainz project, described in Section 2.2.5. An accompanying server application, Acoustid, provides a database of fingerprints and a server to look up queries on. At the time of writing the hosted Acoustid service[4] contains 14 million fingerprints representing over 5 million distinct songs and performs about 2 million lookups per day[5]. The Chromaprint algorithm is released under the Lesser GPL license and the server under the MIT license.

- **Landmark**: A Matlab implementation of Wang (2003), also referred to commonly as the "Shazam algorithm", due to it being the basis of the commercial Shazam audio fingerprinting service[6]. Shazam performs over ten million lookups from around the world every day[7]. It can identify an unknown audio segment recorded by a mobile phone in a noisy room in under 10 seconds. This implementation of the algorithm provides the fingerprinting and server/lookup process. It is available online for download[8]. This implementation is often in other literature when comparing the Shazam algorithm to other fingerprinting algorithms (Chandrasekhar, Sharifi, and Ross 2011; Fenet, Richard, and Grenier 2011).

3.1 Echoprint

The Echoprint algorithm works by finding *onsets*—points in time where musical notes occur. Features are created by calculating the difference in time between subsequent onsets and creating a hash of these time values. Matching recordings are found by looking for identical hashes in the reference database.

3.1.1 Preprocesing and transform

Audio signals coming in to the Echoprint algorithm are converted to mono and their sample rate is reduced to 11025 Hz. In order to prevent sudden noisy events such as pops and bangs

[4] `http://acoustid.org`
[5] `http://acoustid.org/stats`
[6] `http://shazam.com`
[7] `http://www.shazam.com/music/web/about.html`
[8] `http://labrosa.ee.columbia.edu/matlab/fingerprint`

from being mistaken for musical onsets, the input signal is "whitened". To perform the whitening, a 40-pole linear predictor filter is generated from the input signal. This filter changes each sample to be estimated by a smoothed value of the previous 40 samples. This process reduces the amplitude of sudden peaks in the signal.

Once the audio has been downsampled and whitened it is transformed into the frequency domain. Echoprint uses a 128 band cosine filter bank to perform this transform (Ramstad and Tanem 1991). The filterbank is moved over the signal with a hop size of 32 samples. The resulting frequency bands are grouped in to eight equally spaced bins by summing the absolute difference of adjacent bands. The eight bins a equally spread out from 0 Hz to 5512.5 Hz.

3.1.2 Hashing

Echoprint hashes are calculated based on the time difference between musical onsets in each band. The first step of the hashing process is to detect the onsets in the audio signal. Onsets are detected in each frequency band independently. In each band, an envelope follower is used to measure the amplitude of the band. When the amplitude reaches a threshold, an onset is registered. After an onset has been detected, 128 samples must pass before the next onset. The amplitude of the onset is multiplied by an exponentially decaying curve to calculate a new threshold value. Subsequent detected onsets must exceed this threshold in order to be counted. The multiplier decay is adaptive to the number of onsets that are being detected. Echoprint has a target of generating one onset per second in each frequency band. If onsets are being generated at more than this rate, the multiplier decay is increased, resulting in a larger threshold to exceed. If the rate of onsets decreases too much, the multiplier is decreased to compensate.

Figure 3.1 shows a graphical view of the spectrogram of an audio track generated by the filter bank, and split in to eight bands. Onsets in each band are superimposed over the top. Onsets can occur at different times in each frequency band.

To encode the onsets to numerical values, the algorithm considers the time of each onset (o) and the time of its four successors ($s1$–$s4$). A hash value is created by taking the time delta between pairs of the five onsets (Figure 3.2), and the band that they occur in.

The two hash values and band index are stored in a 40 bit (5 byte) number (two bytes for each delta and 1 byte for the band index). The number is reduced to a 32 bit integer

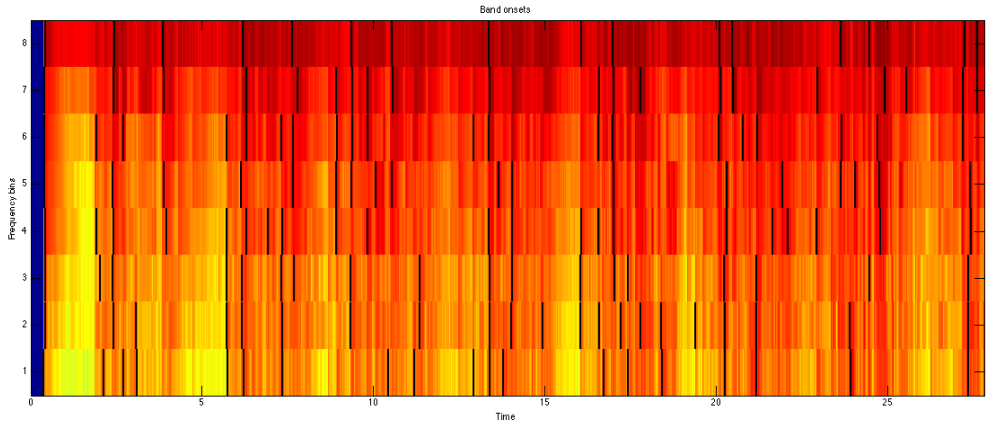

Figure 3.1 Onsets detected by Echoprint on eight frequency bands for the first 30 seconds of "Life on Mars", by David Bowie.

bytes 1–2	$s1 - o$	$s1 - o$	$s2 - o$	$s1 - o$	$s2 - o$	$s3 - o$
bytes 3–4	$s2 - s1$	$s3 - s1$	$s3 - s2$	$s4 - s1$	$s4 - s2$	$s4 - s3$

Figure 3.2 Calculating the time delta between pairs of onsets to create a hash.

with the MurmurHash algorithm (Appleby 2009).

Each onset and set of successors generates six hashes. By pairing onsets and successors, the algorithm adds robustness against the failure of onsets to be detected. If one onset is missed then there will still be some matching hashes at that point in time. This hashing method results in approximately 48 hashes per second of audio (8 bands, 1 onset per second, 6 hashes per onset).

3.1.3 Storage

Hashes are paired with the time that the onset occurs in the audio query. The `offset hash` pairs for a single recording are split into a number of sub-recordings, each 60 seconds long, overlapping with the previous sub-recording by 30 seconds. Hashes are split in this manner because the matching component of the system scores recordings by the number of times a hash in the recording matches hashes in the query. If the hashes were not split then long recordings would receive an unfair advantage at the lookup stage because recordings with repeated content could generate the same hash value at many points in the recording. The hash values are stored in an inverted index, where each value contains reference the

sub-recording and time in the full recording that the hash occurs at. The Echoprint server application uses Apache Solr[9], a fast text search engine, to store the hash index. The full set of `offset hash` values for each recording are stored in a separate database for use in the lookup process.

3.1.4 Lookups

Lookups are performed in two steps. The same fingerprinting process is performed on the query signal, resulting in a set of `offset hash` pairs. For the first step the time values are discarded. The inverted index is searched to find all 60 second sub-recordings that contain a hash value that is present in the query. These sub-recordings are ordered by the number of times a recording hash matches a query hash. The 15 sub-recordings with the highest number of matching hashes are returned. If more than one sub-recording for the same recording is returned from this stage, all but one of them are discarded.

The final score of each candidate recording is calculated by trying to fit the query hashes to the recordings. This is done by calculating the time difference between the onset time in the query and the onset time in the recording for each hash in the recording, and keeping a sum of the number of times each time difference occurs. If a query fingerprint is similar to a recording fingerprint then there will be a large number the same time difference. This sum is used as the score of each candidate recording. If the recording with the highest number of matching hashes has more than twice as many matching hashes as the next recording then it is returned as the match to the query, otherwise no match is returned.

3.2 Chromaprint

3.2.1 Preprocessing and transform

Input audio to Chromaprint is converted to mono and downsampled to 11025 Hz. The audio signal is converted in to the frequency domain by performing a short-time Fourier transform (STFT) with a frame size of 4096 samples (190 ms) and a 2/3 overlap (2731 samples). The resulting spectrum is converted to 12 bins representing the *chroma* of the signal. Each bin in the chromagram represents the energy that is present in a musical note. The 12 bins represent the 12 notes of the diatonic scale (Kurth and Muller 2008).

[9] `http://lucene.apache.org/solr`

3.2.2 Hashing and storage

Figure 3.3 The match filters used in Chromaprint.

To calculate hash values for Chromaprint, the six filter shapes in Figure 3.3 are used. Using the AdaBoost technique described by Jang et al. (2009), the algorithm generates 16 filters that are composed of different sizes of the six filters shown above. These 16 filters are pre-calculated as part of the Chromaprint algorithm and do not change.

A 12-by-16 sliding window is moved over the chromagram one sample at a time. For each frame, the 16 generated filters are applied to the window. To apply a filter, the filter sums the amount of energy in the white area and subtracts the amount of energy in the black area, resulting in a single value. Each of the filters quantizes the energy value to a 2-bit number (from 0–3). The 2-bit value is encoded using Gray coding, resulting in a binary sequence where each value differs from the previous and next value by only one bit.

The 2-bit hash values from each of the 16 filters are converted to a single 32-bit integer representing the subfingerprint of the 12-by-16 window. The window is advanced one sample to calculate the next subfingerprint (Figure 3.4). The subfingerprints are stored in an inverted index pointing to the recording in which they occur, and the full fingerprint is stored in a database.

Figure 3.4 A graphical representation of the Chromaprint for the first 10 seconds of "Ziggy Stardust", by David Bowie. Each vertical block represents a 32 bit integer, with 1 encoded as black an 0 as white.

3.2.3 Lookups

The generated hashes are considered to be robust enough that at least one 32-bit hash value in the reference recording also exists in the query. This assumption is used to select the candidate recordings. Given a query, all tracks in the reference database that contain one of the hashes in the query are retrieved from the inverted index.

Noise in the query can result in bits being flipped in subfingerprints. Due to the nature of the Gray coding used in the hash generation step, slight changes to the chromagram are manifested as a change in the 2 bit number returned from each filter. The real score of each recording is calculated by counting how many subfingerprints in the query match a subfingerprint in the candidate recording within a Hamming distance of 2 (that is, there are 0, 1, or 2 changed bits between the two subfingerprints). The recording with the highest score is returned as a match to the query.

3.3 Landmark

3.3.1 Preprocessing and transform

Before feature extraction is performed, the signal is converted to mono and the sampling rate is reduced to 11025 Hz.

In order to transform the signal into the frequency domain the algorithm performs a short-time Fourier transform (STFT, Portnoff 1981) with a window size of 46.4ms (512 samples) and a hop size of 32ms (352 samples). The Fourier transform results in frequency bins 21.5 Hz wide. A Hamming window is used on each frame before performing the transform.

3.3.2 Feature selection

After the signal has been converted to the frequency domain the next step is to select features from the spectrum. The Landmark algorithm uses peaks in the amplitude of the spectrum in each frame to find features to encode as the fingerprint.

To find the target maximum amplitude, the highest amplitude in the first 10 frames of the STFT is found. Each time a peak is found, a threshold is updated to be 0.998 of the amplitude of that peak. The next peak that is selected must exceed this threshold, after which the threshold value is updated again. This amplitude decay is chosen to achieve a target hash distribution of approximately 7 hashes per second. The threshold can be

decreased to find more peaks in the audio, which can increase accuracy but at the expense of computation time to create hashes from all of the peaks and search for them in the database. One the peaks have been found in a frame the amplitude of the peaks is discarded. The time (in 512 sample steps) and frequency (in 21.5 Hz bins) of each peak is stored for the next step.

Once the amplitude peaks have been selected, pairs of peaks are gathered together. The pairing algorithm only pairs peaks within 31 frequency bins (∼484 Hz) and 63 time steps (2016 ms) of each other. Only the closest 3 peaks in time to each other are selected. This "fanout factor" can be increased to generate more hashes to make matching more reliable, at the expense of computation time. Plotting peak pairs on the STFT graph results in what is called a "constellation map", as the lines between peaks somewhat resemble constellations in the night sky (Figure 3.5).

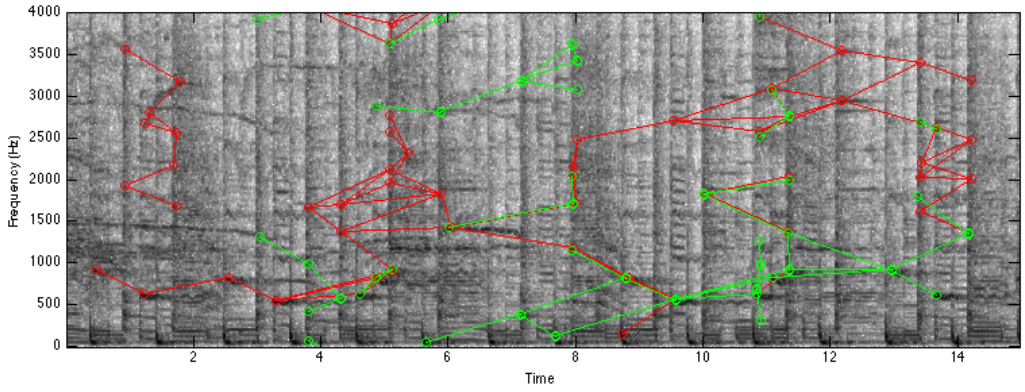

Figure 3.5 A "constellation map" of hash pairs generated by the Landmark algorithm for a 15 second query of "Ashes to Ashes", by David Bowie.

3.3.3 Hashing and storage

Hash pairs are represented as the two frequency bins that the points lie in, and the time between them. These values are encoded in a simple hash. The reference algorithm uses a 20 bit hash, consisting of 8 bits for the starting frequency (F1), 6 bits for the delta between F1 and F2 (FD), and 6 bits for the difference in time between the two peaks (TD). Time differences are stored in units of 32 ms (the STFT hop size).

Figure 3.6 shows how the details of a pair of peaks are encoded in to a single hash value. This hash value is stored in an inverted index, with the hash number referencing the recording ID and the time in the recording that the hash is from.

```
(start time, Freq 1, Freq 2, Time delta) = (917, 59, 44, 14)

= 00111010110001001110
  \__F1__/\_FD_/\_TD_/

= 240718
```

Figure 3.6 Encoding features of the hash for a peak pair into a single 20 bit integer containing the start frequency (F1), the delta between F1 and F2 (FD) and the time delta between the peaks (TD).

3.3.4 Lookups

To perform a lookup of a query the same hashing process is performed to create a set of hashes for the query signal. To increase the chance of generating matching hashes, three more sets of hashes are created by advancing the signal by 1/4, 1/2, and 3/4 of the window size and the STFT and repeating the hashing process. When finding peaks in the signal the target density for landmarks is increased from 7 per second to 20 per second by reducing the threshold multiplier to generate more possible matching hashes.

Once a set of hashes has been generated the lookup table is searched to return all reference recordings that contain a hash present in the query signal. The candidate recordings are ordered by the number of matching hashes between the query and the candidate recordings. The candidate recording with the most matching hashes is returned as the match to the query. Because each hash points to a recording as well as the offset that the hash occurs in the recording the point in time that the query is from can be recreated.

Chapter 4

Evaluating fingerprinting algorithms

A contribution of this thesis is a large-scale evaluation of a selection of publicly available audio fingerprinting algorithms. This chapter discusses criteria of an evaluation suite designed to compare different audio fingerprinting algorithms. The evaluation suite is designed to allow many fingerprinting algorithms to be simultaneously evaluated, each using the same input query parameters. The chapter begins by introducing the aspects of fingerprinting systems that should be tested in an evaluation, before introducing a framework that fulfils these requirements. We present an experiment that uses this evaluation suite to compare the three fingerprinting algorithms that were discussed in Chapter 3. The results of this experiment are presented in Chapter 5.

4.1 Choosing evaluation criteria

Many publications introducing new fingerprinting algorithms or improvements only publish retrieval results for experiments performed with a small number of recordings in the reference database (e.g., Batlle, Masip, and Guaus (2002), 2000; Fragoulis et al. (2001), 450; Papaodysseus et al. (2001), 1000). While this small number of reference files is fine for producing preliminary results for a retrieval system, a larger dataset is useful to be able to identify issues that might only become apparent when the size of the dataset increases. Small datasets increase the risk of not including styles of music that may cause problems for a fingerprinting algorithm (Catalán 2009). To be representative of a wide range of music, an evaluation should be performed using a large reference database.

Other reports of algorithm success rates provide a larger database of test songs: Kastner et al. (2002) use a database of 85,000 recordings, but only 30-second excerpts. Haitsma and Kalker (2002) test with 10,000 tracks, and Wang (2003) with 20,000. Recent evaluations test even more: Fenet, Richard, and Grenier (2011) use 30,000 files and Ellis, Whitman, and Porter (2011) 100,000. This literature often only presents the retrieval results of the single algorithm described in the paper and do not directly compare the algorithm with other algorithms. Due to differences in the corpus used in different experiments, and different ways of processing input signals, it can be difficult to compare the success rate of different algorithms based on results from different publications. Many papers that present results for modified queries refer to Haitsma et al. (2001) for a list of possible modifications, but sometimes do not give complete descriptions of the modifications in order to accurately reproduce the results. Some publications present comparative results of more than one algorithm (Chandrasekhar, Sharifi, and Ross 2011), but only showing results for a specific set of criteria. A comprehensive evaluation should compare many fingerprinting algorithms under the same conditions.

A key feature of many audio fingerprinting systems is to be able to record queries in a noisy environment and still obtain a correct match. Some algorithms are designed to accurately identify music from a particular environment, for instance, music that has been broadcast via FM radio and has been changed from its original signal prior to transmission. In the absence of queries recorded from these situations, it can be useful to artificially manipulate the query signal in order to simulate the behaviour of these sorts of treatments.

The result of an evaluation should be an indication of how good a fingerprinting system is at identifying an unknown audio query given a corpus of known audio. Such a metric should indicate how often the algorithm fails to identify a recording given a query, or gives the wrong answer.

4.2 Evaluation framework

We have developed an audio fingerprinting evaluation framework to compare multiple fingerprinting algorithms against each other. The framework is written in Python[1], and uses a MySQL[2] database server to store intermediate data. The evaluation system was specifi-

[1] `http://python.org`
[2] `http://mysql.com`

cally designed so that it can be run in parallel on many computers at the same time, with a queue server used to distribute work between these separate computers. Any fingerprinting algorithm can be tested using the framework through a *module*, a software interface that provides a mapping between that fingerprinting algorithm and the evaluation suite. The module interface presents a consistent API (Application Programming Interface) for the evaluation system, allowing for different fingerprinting algorithms to be easily added to an evaluation. The framework also allows for evaluations on one or more algorithms to be run using queries created from the same collection of recordings with the same signal modifications.

4.2.1 Distributed computation

The evaluation framework makes use of multiple computers in order to speed up the evaluation process. The evaluation framework, which uses the RabbitMQ [3] queue broker system, can be installed on many computers and run simultaneously. The queue contains a list of actions to perform in the running of a single evaluation, for example, to take a single file, turn it into a query, and then look up the query in a fingerprinting system. Each of the algorithms being tested can run independently on different computers, or many computers can concurrently run an evaluation for a single algorithm, to test simultaneous access to the fingerprint lookup server. The queue system provides a locking mechanism to ensure that no two workers attempt to perform a fingerprint action on the same file.

4.2.2 Modularity

Each algorithm to be tested is represented by a module in the evaluation framework. Modules are required to fulfil a simple contract in order to give a consistent interface for the framework to fingerprint and test different algorithms. The contract defines the input and output of each step that the module performs. Figure 4.1 shows the contract that modules in the evaluation framework must adhere to.

An audio fingerprinting system consists of two main steps: importing audio into a reference database and querying the database for the metadata on an unknown piece of audio. Both of these steps require a process that generates a fingerprint from a segment of audio. The **Fingerprint** step takes an audio signal and computes the numerical fingerprint

[3] `http://www.rabbitmq.com`

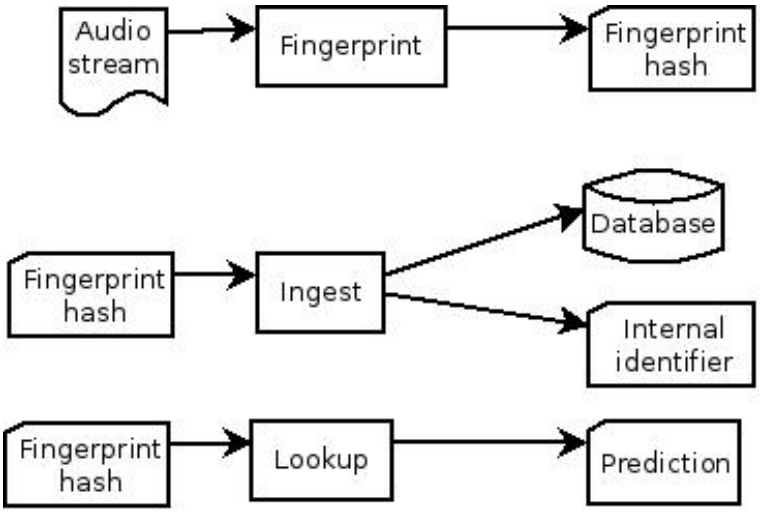

Figure 4.1 Contract for fingerprinting modules. Each module must be able to generate a fingerprint, add a fingerprint to a database, and look up a query.

that represents the audio passed to it. The **Ingest** step takes the fingerprint of a whole audio file and adds it to the algorithm's reference database. In order to succinctly reference the algorithm's idea of a recording, the ingest method returns a short unique identifier that maps to the fingerprint codes.

The evaluation framework maintains a mapping between source files and these fingerprint IDs. Comparison of these IDs provides the main metric for evaluating the accuracy of the algorithms. The **Lookup** step takes a fingerprint that has been generated from a short query of music and searches for the fingerprint in the algorithm's database. It returns the internal ID that it believes belongs to the query that was passed in. The evaluation framework stores this identifier and can match it to the actual ID of the file used to perform the query.

4.2.3 Repeatability

Experimental repeatability is important when performing evaluations. Having a record of the steps performed during the evaluation can help to reproduce results and determine what changes to the process result in an increase or decrease in accuracy. Being able to repeat an experiment with the same inputs is also useful during the development of fingerprinting algorithms. By performing the same evaluation on different variations of an algorithm it is possible to determine what modifications to an algorithm increase its accuracy. In order

to ensure experiments can be repeated exactly, the evaluation framework keeps a record of the choices of files it chooses while running as well as the query modifications made to query audio.

4.2.4 Collecting statistics

The accuracy of fingerprinting systems can be measured by the successful retrieval rate. To calculate retrieval rate statistics for each fingerprinting algorithm with a particular query, the framework stores the result of each query to the fingerprinting system. The framework maintains a list of all of the audio files being used in the evaluation. The framework can compare the result to the expected fingerprint to determine if the fingerprinting system was correct or incorrect. As a starting point, the evaluation suite provides tools to calculate the true positive, false positive, true negative, and false negative rates. From these statistics precision, recall, specificity, and sensitivity can be calculated. These metrics are discussed in further detail in Section 4.3. By storing these results directly rather than computing statistics as the evaluation is running, new statistics can be calculated if needed, without needing to run the evaluation again.

In addition to retrieval statistics the framework also captures runtime information of the algorithm, including the speed taken to fingerprint the audio file, and the speed taken to perform a lookup. These metrics are important because some algorithms may trade fingerprinting and lookup speed for accuracy.

4.2.5 Query modifications

Effective fingerprinting algorithms are able to identify audio queries that differ from the reference recording in the database. These modifications can be due to noise introduced if the query is recorded in a noisy environment (e.g., in a car or public venue) or because the audio was intentionally modified (e.g., for radio transmission, or to reduce file size to make downloads faster). In order to simulate these kinds of modifications to a query signal the evaluation framework contains a number of filters that can alter the audio before it is used as a query. The filters can be used individually or joined together in a sequence, using the audio output of one filter as the input to another, to adjust many aspects of the audio query at the same time (see Figure 4.2 for an example). The following list of modifications was created based on other publications that discuss the robustness requirements of

fingerprinting algorithms (Haitsma et al. 2001; Cano et al. 2002). The name of each filter as used in the evaluation suite is given in parentheses. The modifications were performed with the Sound eXchange (sox) sound processing program[4].

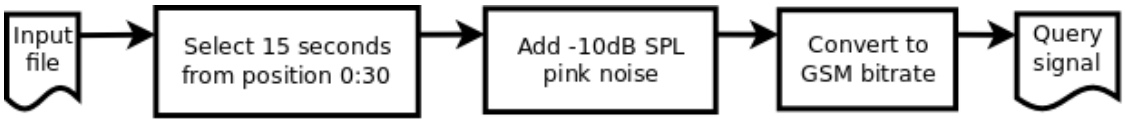

Figure 4.2 Chaining filters together. The output of the first filter (excerpt section) is used as input to the second filter (adding noise), and so on, resulting in a signal that can be used for a query.

- **Query position & length** (`chop`): A fingerprinting algorithm should be able to identify an unknown recording from a short audio query. In many occasions the entire song may not be able to be recorded, in which case the fingerprinting algorithm must still be able to identify the query. Not only does this reduce the time needed to generate a fingerprint and compare the fingerprint to the reference database, but it also means that a person trying to identify a part of a recording in realtime does not need to record the song for the full duration.

- **Audio bitrate** (`bitrate`): Music encoded in the MP3 format has its quality expressed as an audio bitrate, which characterises how much audio content remains in the signal stream after parts of the frequency spectrum are removed from the signal when it is encoded. MP3s encoded for transmission on the Internet often have low bitrate in order to reduce the file size and therefore the time needed for download. A notable example of this is the low-quality video setting on the video sharing site YouTube. Many music videos are uploaded to YouTube at the lowest video quality setting, which has an associated audio bitrate of 64 kbps. Audio encoded at this rate has a noticeably lower quality than the same recording played from a CD but should still be correctly identified by a fingerprinting algorithm.

- **GSM audio:** (`gsm`): Many commercial audio fingerprinting services provide the ability to record audio using a smartphone and perform a lookup. Because such devices were original designed for speech, the codec used to store recorded audio is

[4] `http://sox.sourceforge.net`

designed to efficiently encode speech rather than music. This means that fingerprint lookup systems that transmit queries over a phone call rather than the Internet need to correctly identify audio that has been compressed for telephone transmission. The GSM standard is an industry standard speech transmission format. GSM audio is mono with a sample rate of 8000kHz.

- **Playback speed:** (`samplerate`): Audio broadcast by radio stations is often manipulated by the station before transmission to make the audio sound more energetic. One technique used is to adjust the sample rate of the audio, resulting in an increase in the speed of the audio when played back. A side-effect of this sample rate modification is that the pitch of the audio signal is also increased.

- **Compression & Equalization:** (`fm`): In order to make broadcast music sound more appealing after being transmitted over FM radio, some radio stations increase the amplitude of some frequency bands before transmitting the signal. These alterations are used to make the audio sound better in certain listening situations, such as on car stereos.

- **Noise:** (`noise`): To simulate the recording of an audio query from an environment where there is ambient noise, the *noise* filter can mix in any audio with the query. The evaluation framework provides some sample noise that can be used to create queries. These noise samples are pink noise, noise from a car driving on the road, and spoken conversation. Pink noise was chosen because it has the same perceptual loudness at all frequencies, as opposed to other noise signal such as white noise. The noise samples are available at three different volume levels, 0dB SPL, −10dB SPL, and −20dB SPL. Audio at 0dB SPL is measured to have an normalised amplitude in the range 1−−1. The −10dB steps are perceived as a halving of the volume of the audio. The sample noise queries were downloaded from the freesound audio archive[5]. The two noise sounds used are "Bar Crowd - Logans Pub - Feb 2007.wav"[6] and "Driving in Streamwood IL with the windows down (05-04-2009)"[7].

[5] `http://freesound.org`
[6] `http://www.freesound.org/people/lonemonk/sounds/31487/`
[7] `http://www.freesound.org/people/audible-edge/sounds/72830/`

4.2.6 Evaluation process

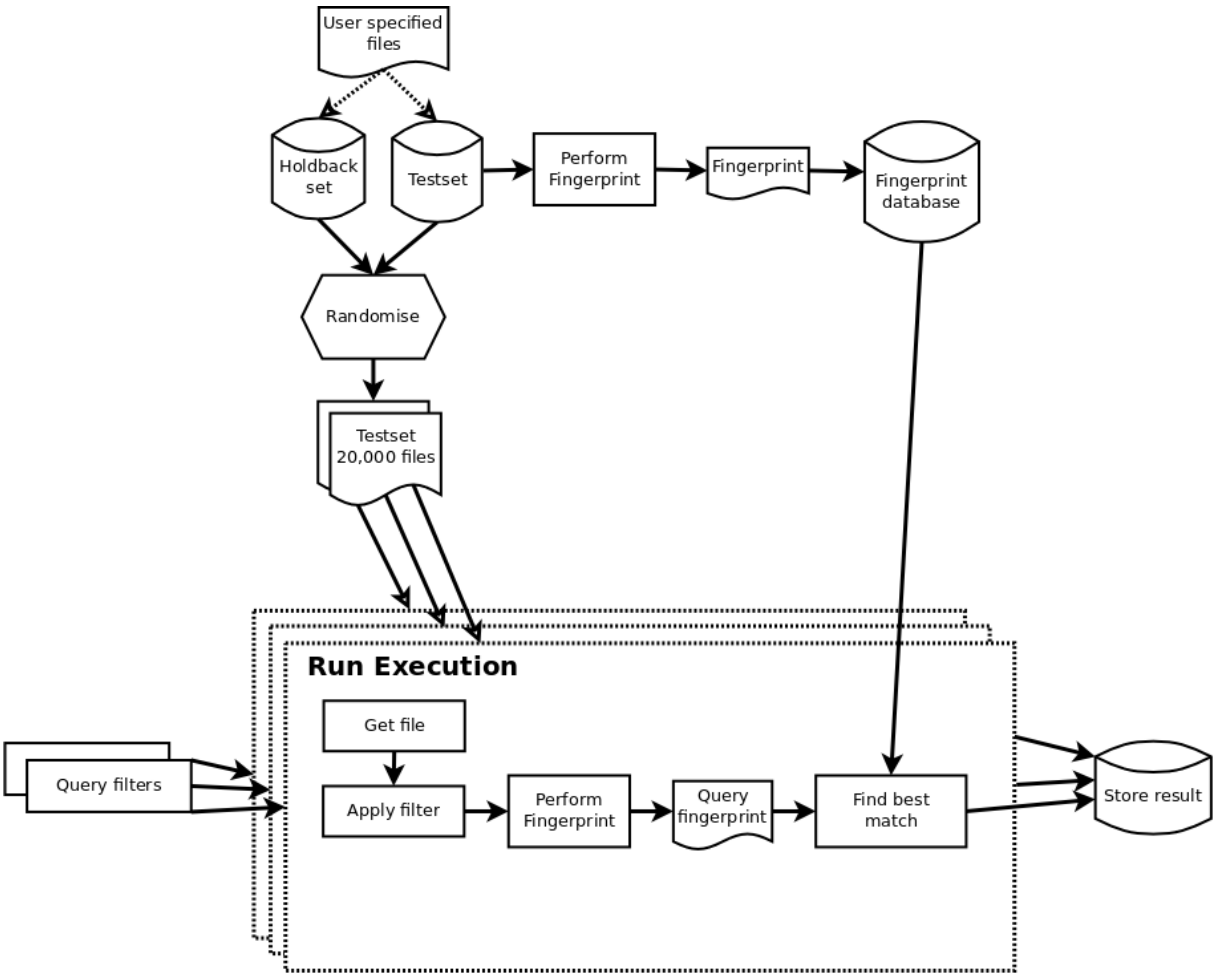

Figure 4.3 A flowchart of how the evaluation framework imports audio to fingerprinting algorithms and then performs lookup queries

The main database and queue software for the evaluation system are installed on a single computer. Each worker machine is able to connect to these servers to write results and retrieve work. All workers have access to the same test files, accessible at the same location on each worker. Figure 4.3 shows the steps that are performed when adding files to a reference database and performing the evaluation.

To import the audio, the user specifies a source of files to be used for the evaluation. The evaluation framework reads the list of files and performs a fingerprint action on each of them with each fingerprint algorithm. This becomes the reference fingerprint for the recording.

The fingerprint is sent to the fingerprinting system server, which stores it and returns a unique identifier. The evaluation suite records the unique combination of input file, fingerprint algorithm, and unique identifier. A small percentage of files are not fingerprinted (by default, 20%). These files are used as a holdback set, and used to test the behaviour of the fingerprinting lookup process when given a query for a recording that is not in the reference database.

A *run* is an execution of a random set of files that are all modified with the same query filter and then looked up in a fingerprint server. To directly compare two or more fingerprinting algorithms, a set of runs with the same testset, same query filters, and different fingerprint algorithms can be created.

To execute a run, a worker reads test files from a queue. It performs each of the query modifications in order and uses the result as an input to the *lookup* method of the module for the fingerprinting algorithm under test. The lookup method returns the unique identifier that is the best match of the fingerprinting system for the given query. The framework stores this result in the database. The result returned for this specific run can be compared to the actual unique identifier created when the file was added to the fingerprint database.

4.3 Document retrieval statistics

During an evaluation of a fingerprinting system, the evaluation framework enumerates through all of the files in the test set. Each file is modified in the same way to generate a query signal. Some of the queries are known to be in the database of the fingerprinting system, and others are known to not be in the database (the holdback set). When given a query signal, a fingerprinting system can return one of two responses: It can respond that there is no known recording in the database that matches the query or it can return the identifier of a recording that it thinks matches the query.

Based on the known fact that a query is in the system database or not, and the response from the system, the evaluation framework can classify the response in one of five categories:

- **True positive** (*tp*): The system correctly identifies the recording from the query
- **True negative** (*tn*): The system correctly identifies that no recording matches the query
- **False negative** (*fn*): The system incorrectly reports that the query does not exist in the database when it does

- **False positive** (*fp*): The system incorrectly reports the wrong recording when the correct recording does exist in the database
- **False accept** (*fa*): The system reports that a recording was found when it does not exist in the database

Often the last two categories (false positive, false accept) are both reported as false positives—the system reported a result ('positive') but it was incorrect. We report the false accept and false positive values separately in order to discover if fingerprinting systems have different error rates for both of these categories.

By counting the number of times each of these cases occur in an evaluation we can calculate statistics that provide a general indication of the accuracy of a fingerprinting system.

4.3.1 Precision, recall, and specificity

The *precision* (Equation 4.1a) of an algorithm measures how many of the positive results reported by the the system were correct. A positive result occurs when a system reports that a query exists in the reference database. This value could be correct (tp), incorrect (fp), or the query is not in the database and the system should not have reported it (fa).

The *recall* (Equation 4.1b, also called sensitivity in some fields) measures how often an algorithm returns the correct recording when the recording is known to exist in the database. It weighs the number of correct responses (tp) against the number of times the system says the query is not in the database (fn), or incorrectly chooses a different recording (fp). It is important to consider both the precision and recall together when evaluating the retrieval accuracy of a system. As a contrived example, a system could successfully identify one known recording ($tp = 1$), and report that no recording exists for 99 other queries, when they in fact do exist in the database ($fn = 99$). In this case precision would be 100%, however the recall would only be 1%.

Finally, we consider *specificity* (Equation 4.1c), the measurement of how good the system is at identifying negative results, where the query is known to not be in the reference database. This metric compares the correct responses (tn) against responses where the system incorrectly reports that the query matches a recording (fa).

In audio fingerprinting systems it is desirable to have a high precision and a high specificity. Low values for either of these metrics means that the system is reporting the

incorrect recording as the result to a query. We propose that fingerprinting systems should favour precision and specificity over recall—that is, if at all uncertain about a match, the system should report that the query is unknown, rather than give an incorrect answer.

$$precision = \frac{tp}{tp + fp + fa} \tag{4.1a}$$

$$recall = \frac{tp}{tp + fn} \tag{4.1b}$$

$$specificity = \frac{tn}{tn + fa} \tag{4.1c}$$

From these three evaluation metrics it is possible to derive the size of the five classification categories (tp, tn, fp, fn, fa).

We do not calculate the f-measure, a single measurement that weighs both the precision and recall values. We favour instead the sensitivity metric presented in Section 4.3.3, as it also considers the number of false accepts made by the retrieval system.

4.3.2 Confidence intervals

In addition to calculating retrieval metrics for a system, we can provide a confidence interval surrounding the estimate. When we calculate the confidence interval of a value for the fingerprinting algorithm, we are indicating a level of uncertainty where we expect the distribution of that value to fall. With smaller data sets the confidence interval may be wider, indicating that there are not enough reported values in order to make an accurate estimate of a particular metric.

We use the "add two success and two failures" adjusted Wald interval as described by Agresti and Coull (1998). In this equation, n represents the numerator of the fraction for which the interval is being calculated, and d the denominator. For example, when calculating the confidence interval of precision, $n = tp, d = tp + fp$. The lower limit (LL) and upper limit (UL) compute an approximate 95% confidence that the metric falls within the bounds of the limits.

$$p_0 = \frac{n+2}{d+4} \tag{4.2a}$$

$$LL, UL = p_0 \pm 2 \cdot \sqrt{p_0 \cdot \frac{1-p_0}{d+4}} \tag{4.2b}$$

The wider the confidence interval, the more uncertainty there is that the value reported is representative of the sample.

4.3.3 Sensitivity

The final metric that we will show for the evaluation is the sensitivity of the retrieval system (Macmillan and Creelman 1991). Sensitivity measures if a how much better a system is than randomly choosing an answer.

We introduce one more metric, the false accept rate (far).

$$far = \frac{fa}{fa + tn} \tag{4.3}$$

The false accept rate is a reformulation of specificity (Equation 4.1c). It measures the probability of error rather than the probability of success, $far = 1 - specificity$.

The sensitivity measure calculates the difference between the mean values of $recall$ and far. The further apart these means are, the more the system is able to correctly identify queries. Overlaps in the distributions of these values causes uncertainty that leads to false positives, false negatives, and false accepts.

The sensitivity of the classifier, d', is given by

$$d' = Z(recall) - Z(far), \tag{4.4}$$

where $Z(n)$ is the inverse Gaussian distribution. The Z transform converts the mean values into standard deviation units.

The d' value indicates if a signal can be differentiated from a randomly selected answer. The measure has a generally accepted upper limit of $d' = 4.65$, when $recall = 0.99$ and $far = 0.01$. It is possible, however, to compute values as high as 6.93 with some $recall$

and far. A value of $d' = 0$ indicates that $recall = far$ and the selection of an answer is effectively random. Higher values of d' indicates that the system is able to successfully discriminate between true and false answers.

If there are adjustable parameters in a fingerprinting system to trade off the *recall* and far values then this trade off can visualised by plotting the d' value for different settings on an ROC (receiver operating characteristic) curve. This graph plots the trade off between true positives and false positives. We do not report a metric based on the ROC curve in this evaluation because we performed no parameter tuning for any of the tested algorithms.

4.4 A comparison of audio fingerprinting algorithms

4.4.1 Experiment

We performed an evaluation using the presented evaluation framework and the three fingerprinting algorithms introduced in Chapter 3. We wrote a module for each of the algorithms in the experiment, fulfilling the *ingest* & *lookup* contract. The experiment was run using the Codaich dataset (McKay, McEnnis, and Fujinaga 2006). This collection contains 30,283 unique audio files in MP3 format covering a wide range of genres such as pop, western classical, jazz, and a selection of world music. To test that the false accept rate (Section 4.3.3) of the algorithms is low, we withheld 20% of the recordings in the dataset. As these files were not added to the reference databases, when a query is made with one of the recordings the fingerprinting system should return no match. The remainder of the recordings were imported into the reference databases. We selected 20,000 files over 60 seconds in length to perform the evaluation with. Each query file was modified using each of the modifications shown in Table 4.1. These generated queries were used to perform a lookup using each fingerprinting system. Results for this experiment are presented in Chapter 5. Each of the fingerprinting algorithms were set up as follows.

4.4.2 Algorithm 1 setup: Echoprint

The Echoprint server software requires the Solr search server[8] and Tokyo Tyrant[9] key-value store to be installed. The Echoprint server software[10] package comes with a library written

[8] `http://lucene.apache.org/solr`
[9] `http://fallabs.com/tokyotyrant`
[10] `http://echoprint.me/server`

Query start time	Query length	Query modifications
0	8, 15, 30 seconds	none
30	8, 15, 30 seconds	none
0	15, 30 seconds	96k MP3 bitrate
0	15, 30 seconds	64k MP3 bitrate
0	15, 30 seconds	Convert to mono
0	15, 30 seconds	Change sample rate to 22k
0	15, 30 seconds	Change sample rate to 8k (GSM)
0	15, 30 seconds	FM Radio modifications
0	15, 30 seconds	Increase volume by 20%
0	15, 30 seconds	Decrease volume by 50%
0	15, 30 seconds	Decrease volume by 20%
0	15, 30 seconds	Increase speed 1.0%
0	15, 30 seconds	Increase speed 2.5%
0	15, 30 seconds	Increase speed 5.0%
0	15, 30 seconds	Decrease speed 1.0%
0	15, 30 seconds	Decrease speed 2.5%
0	15, 30 seconds	Decrease speed 5.0%
0	8, 15, 30 seconds	Mix 0dB SPL pink noise
0	8, 15, 30 seconds	Mix −10dB SPL pink noise
0	8, 15, 30 seconds	Mix −20dB SPL pink noise
0	8, 15, 30 seconds	Mix 0dB SPL road noise
0	8, 15, 30 seconds	Mix −10dB SPL road noise
0	8, 15, 30 seconds	Mix −20dB SPL road noise
0	8, 15, 30 seconds	Mix 0dB SPL bar noise
0	8, 15, 30 seconds	Mix −10dB SPL bar noise
0	8, 15, 30 seconds	Mix −20dB SPL bar noise

Table 4.1 Query modifications used in the experiment

in Python to interact with this software to store fingerprints in the databases and perform lookups. The library also calculates the final distance measure between an input query and results from the search server.

4.4.3 Algorithm 2 setup: Chromaprint

We installed the Acoustid server[11] application. The server uses the PostgreSQL database[12] and a custom in-memory inverted index, `acoustid-index`, in order to speed up the lookup process. An existing Python library, pyacoustid[13] was used to interact with the acoustid webservice from the evaluation software.

4.4.4 Algorithm 3 setup: Landmark

The author of the Landmark fingerprinting code makes available a compiled binary, audfprint[14], for the purposes of evaluating the algorithm. By using a compiled binary it was unnecessary to interact with Matlab, which the system was written in. The only additional dependency is the freely available Matlab runtime. Because the audfprint system did not have a separate server component, and the startup process took some time (a matter of seconds) the lookup method in the Landmark module was adapted to process queries in blocks of 100 files. A list of queries was sent to the audfprint lookup program, which returned a list of predicted recordings.

[11] `http://acoustid.org/server`
[12] `http://postgres.org`
[13] `https://github.com/sampsyo/pyacoustid`
[14] `http://labrosa.ee.columbia.edu/matlab/audfprint`

Chapter 5

Results

This chapter presents the results of the experiment described in Section 4.4. We present precision, recall, specificity, and d' values for all types of modified query and compare the results returned by each fingerprinting system.

5.1 Query length

Table 5.1 shows the accuracy statistics for fingerprinting algorithms with differing query lengths[1]. For the queries starting at time $t = 0$ we see that the precision for Echoprint and Chromaprint are almost 100%. The specificity for all algorithms is also high, indicating that there are few false alarms. The high recall and specificity values for Echoprint and Chromaprint show that they are able to generate fingerprints that uniquely identify almost all recordings in the reference database. The Landmark algorithm has a lower recall due to many recordings having identical hashes generated, which we discuss in further detail below.

The Chromaprint algorithm does not work well on short queries (8 s), but as the query length is increased to 30 seconds it provides the highest accuracy of all the algorithms. Nevertheless, it only achieves high precision and recall on queries that are taken from the beginning of the recording, not from queries starting from part way through the recording. This means that Chromaprint is only useful in cases where the query can be chosen from the

[1]In this and all other tables presented in this section, accuracy percentages are rounded to the nearest whole number. Error limits are bounded within the range 0%–100%

Query length	8 s			15 s			30 s			0:30–0:38			0:30–0:45			0:30–0:60		
Algorithm	P	LL	UL	P	LL	UL	P	LL	UL	P	LL	UL	P	LL	UL	P	LL	UL
Echoprint	100	100–100		100	100–100		100	99–100		91	88–94		99	99–100		100	99–100	
Chromaprint	100	98–100		99	99–100		99	99–100		0	0–100		100	16–100		100	44–100	
Landmark	94	93–94		94	94–94		94	94–94		94	94–95		94	94–94		94	93–94	

(a) Precision (%)

Query length	8 s			15 s			30 s			0:30–0:38			0:30–0:45			0:30–0:60		
Algorithm	R	LL	UL	R	LL	UL	R	LL	UL	R	LL	UL	R	LL	UL	R	LL	UL
Echoprint	63	63–64		90	90–91		96	96–96		2	2–2		55	54–56		87	86–87	
Chromaprint	3	3–3		96	96–97		99	99–99		0	0–0		0	0–0		0	0–0	
Landmark	63	63–64		88	87–88		94	94–94		63	63–64		88	88–89		92	92–93	

(b) Recall (%)

Query length	8 s			15 s			30 s			0:30–0:38			0:30–0:45			0:30–0:60		
Algorithm	S	LL	UL	S	LL	UL	S	LL	UL	S	LL	UL	S	LL	UL	S	LL	UL
Echoprint	99	99–100		99	98–99		98	98–99		100	99–100		99	99–99		99	98–99	
Chromaprint	100	100–100		99	99–100		99	99–100		100	100–100		100	100–100		100	100–100	
Landmark	99	99–100		99	98–99		98	98–99		99	99–99		99	98–99		98	98–99	

(c) Specificity (%)

Query length	8 s	15 s	30 s	0:30–0:38	0:30–0:45	0:30–0:60
Algorithm	d'	d'	d'	d'	d'	d'
Echoprint	2.80	3.55	3.90	0.74	2.49	3.34
Chromaprint	0.46	4.35	4.81	0.00	0.00	0.00
Landmark	2.82	3.41	3.65	2.77	3.42	3.54

(d) Sensitivity (d')

Table 5.1 Accuracy results for unmodified queries, three fingerprinting algorithms, and six query lengths with lower limits (LL) and upper limits (UL).

recording (e.g., when identifying files on a computer). In which case, short query lengths are less important, as a query of any length can be taken from the file.

For short queries (8 s) both Echoprint and Landmark give similar results, however Landmark gives more consistent results when the query is from any point in the file compared to both Echoprint and Chromaprint, even though it has a lower precision. The Landmark algorithm performs poorly with 8-second queries, compared to 15- and 30-second queries. This result was unexpected, as the Shazam smartphone application which is at least partially based on this algorithm has been observed to identify music recorded in a room with queries about this length.

The precision values for the Landmark algorithm are significantly lower than the other two algorithms, indicating that the Landmark system often returns the incorrect recording in response to a query. To show this further, we give the breakdown of results for the case of 30-second queries to each of the three algorithms (Table 5.2). This evaluation was out of 20,000 recordings.

FP system	True positive	True negative	False negative	False positive	False accept
Expected	15959	4041	0	0	0
Echoprint	15300	3980	655	4	61
Chromaprint	15813	4012	88	58	29
Landmark	15017	3966	61	881	75

Table 5.2 The expected retrieval numbers for a 30-second query and actual numbers from the three fingerprinting systems.

Landmark has a significantly higher number of false positive results than the other two systems. This means that it gave the wrong answer to a query often. We observed through the results that the fingerprint algorithm would sometimes generate the same hashes for different recordings. When a query was made for one of these recordings, more than one recording would be returned with matching hashes. If the number of matching hashes was the same in more than one recording then it was not possible for the algorithm to make a decision on what recording was correct.

We can see that Echoprint has a very low false positive rate, though this is at the expense of its false negative rate, which is significantly higher than both Chromaprint and Landmark.

5.2 Modified queries

The next series of tests involve modifying the queries in ways that represent real-world query modifications. These tests changed the bit rate, sample rate, speed, volume, frequency equalization, and number of channels of the queries (as described in Section 4.2.5) before searching the fingerprint database. The results for this series of tests are presented in Tables 5.3–5.6.

The query modification that had the largest impact on the accuracy rate was the altering of the speed of the query. Even small increases or decreases in the speed (1%) caused a complete failure in recall rates for Echoprint and close to 0% recall for Landmark. Only Chromaprint achieved some success, with 27% recall for 30-second queries that had been sped up by 1% and 18% recall for queries that had been slowed down by the same amount. Increasing and reducing the speed by more than 1% resulted in effectively no accuracy. Lowering the sample rate to 8kHz also had an effect on recall rates. Audio sampled at 8 kHz has a Nyquist frequency of 4 kHz, meaning that there are no frequencies above this value present in the signal. As all of the algorithms use frequencies above 4 kHz to generate the hashes that form the fingerprint of a query, the hashes will be different on audio that originally contained audio above this frequency.

Changing the bitrate of queries encoded in MP3 format, adjusting the volume, and changing frequency equalisation had a small effect on the retrieval rates. Converting the queries to mono and reducing the sample rate to 22 kHz had no effect on the accuracy. All of the algorithms perform a preprocessing step on the signal before generating a fingerprint. The preprocessing step for all algorithms reduces the sample rate to 11 kHz and converts the signal to mono, and so making these modifications before sending the query to the fingerprinter has no effect.

For all of the query modifications that did not significantly affect the retrieval rate (excluding the speed increase and the sample rate of 8 kHz), Chromaprint performed the best in almost all situations, except those that did not start at the beginning of the recording. From Section 5.1 it was seen that Chromaprint performed poorly on any query not taken from the beginning of the recording. For modified queries that were to be taken from any point in a recording both Echoprint and Chromaprint give similar results, with Echoprint winning slightly due to its increased precision.

	Echoprint				Chromaprint				Landmark			
	15 s		30 s		15 s		30 s		15 s		30 s	
	P	LL–UL	P	LL–UL	P	LL–UL	P	LL–UL	P	LL–UL	P	LL–UL
Original query	100	100–100	100	99–100	99	99–100	99	99–100	94	94–94	94	94–94
Reduce bitrate												
96 Kbps	100	100–100	100	100–100	99	99–100	99	99–100	94	94–94	94	93–94
64 Kbps	100	99–100	100	100–100	99	99–100	99	99–100	94	94–94	94	93–94
Speed up												
1%	34	26–43	7	3–15	99	98–100	99	99–99	91	86–93	92	90–93
2.5%	0	0–7	1	0–8	98	93–100	98	95–99	20	2–65	11	0–46
5%	0	0–7	0	0–5	100	16–100	100	16–100	0	0–72	0	0–41
Slow down												
1%	23	16–33	3	1–9	100	99–100	99	99–100	89	85–93	90	88–92
2.5%	0	0–6	0	0–5	100	92–100	99	96–100	0	0–56	17	3–47
5%	0	0–8	0	0–6	0	0–100	0	0–100	0	0–100	0	0–41
Adjust volume												
50%	100	100–100	100	99–100	99	99–100	99	99–100	94	94–94	94	94–94
80%	100	100–100	100	100–100	99	99–100	99	99–100	94	94–94	94	94–94
120%	100	100–100	100	99–100	99	99–100	99	99–100	94	94–94	94	94–94
Convert to mono	100	99–100	100	99–100	99	99–100	99	99–100	94	93–94	94	93–94
Downsample												
22 kHz	100	99–100	100	99–100	99	99–100	99	99–100	94	93–94	94	94–94
8 kHz	100	99–100	100	99–100	99	99–100	99	99–100	94	93–95	94	94–95
Radio EQ	100	99–100	100	99–100	99	99–100	99	99–100	94	94–94	94	94–94

Table 5.3 Precision for modified queries

	Echoprint				Chromaprint				Landmark			
	15 s		30 s		15 s		30 s		15 s		30 s	
	R	LL–UL	R	LL–UL	R	LL–UL	R	LL–UL	R	LL–UL	R	LL–UL
Original query	90	90–91	96	96–96	96	96–97	99	99–99	88	87–88	94	94–94
Reduce bitrate												
96 Kbps	86	85–86	93	93–94	89	89–90	99	99–99	84	83–84	94	93–94
64 Kbps	86	85–86	94	93–94	89	89–90	99	99–99	84	83–84	94	93–94
Speed up												
1%	0	0–0	0	0–0	9	9–10	27	26–28	2	1–2	7	6–7
2.5%	0	0–0	0	0–0	1	1–1	2	2–3	0	0–0	0	0–0
5%	0	0–0	0	0–0	0	0–0	0	0–0	0	0–0	0	0–0
Slow down												
1%	0	0–0	0	0–0	5	5–6	18	18–19	1	1–2	6	6–6
2.5%	0	0–0	0	0–0	0	0–0	1	1–1	0	0–0	0	0–0
5%	0	0–0	0	0–0	0	0–0	0	0–0	0	0–0	0	0–0
Adjust volume												
50%	86	85–86	93	93–94	90	90–91	99	99–99	85	84–85	94	93–94
80%	86	85–86	94	93–94	90	90–91	99	99–99	85	84–85	94	93–94
120%	86	85–86	94	93–94	90	90–91	99	99–99	85	84–85	94	94–94
Convert to mono	90	89–90	96	95–96	90	89–90	99	99–99	82	81–82	93	93–94
Downsample												
22 kHz	90	89–90	96	95–96	90	89–90	99	99–99	80	79–81	93	93–94
8 kHz	28	27–29	34	33–35	58	57–59	96	95–96	10	10–11	50	49–50
Radio EQ	89	88–89	95	95–95	89	89–90	99	99–99	87	87–88	94	94–94

Table 5.4 Recall for modified queries

	Echoprint				Chromaprint				Landmark			
	15 s		30 s		15 s		30 s		15 s		30 s	
	S	LL–UL	S	LL–UL	S	LL–UL	S	LL–UL	S	LL–UL	S	LL–UL
Original query	99	98–99	98	98–99	99	99–100	99	99–100	99	98–99	98	98–99
Reduce bitrate												
96 Kbps	99	99–99	99	98–99	100	99–100	99	99–100	99	99–99	98	98–99
64 Kbps	99	98–99	99	98–99	100	99–100	99	99–100	99	99–99	98	98–99
Speed up												
1%	100	99–100	100	99–100	100	100–100	100	100–100	100	100–100	100	100–100
2.5%	100	99–100	100	100–100	100	100–100	100	100–100	100	100–100	100	100–100
5%	100	99–100	100	99–100	100	100–100	100	100–100	100	100–100	100	100–100
Slow down												
1%	99	99–100	99	99–100	100	100–100	100	100–100	100	100–100	100	100–100
2.5%	100	99–100	100	99–100	100	100–100	100	100–100	100	100–100	100	100–100
5%	100	100–100	100	99–100	100	100–100	100	100–100	100	100–100	100	100–100
Adjust volume												
50%	99	99–99	99	98–99	99	99–100	99	99–100	99	98–99	98	98–99
80%	99	98–99	99	98–99	99	99–100	99	99–100	99	98–99	98	98–99
120%	99	98–99	99	98–99	99	99–100	99	99–99	99	98–99	98	98–99
Convert to mono	99	98–99	99	98–99	100	99–100	99	99–100	99	98–99	98	98–99
Downsample												
22 kHz	99	98–99	99	98–99	99	99–100	99	99–100	99	99–99	98	98–99
8 kHz	100	99–100	99	99–100	100	99–100	99	99–100	100	100–100	99	99–99
Radio EQ	99	98–99	98	98–99	100	99–100	99	99–100	99	98–99	98	98–99

Table 5.5 Specificity for modified queries

	Echoprint		Chromaprint		Landmark	
	15 s	30 s	15 s	30 s	15 s	30 s
	d'	d'	d'	d'	d'	d'
Original query	3.55	3.90	4.35	4.81	3.41	3.65
Reduce bitrate						
96 Kbps	3.38	3.72	3.86	4.70	3.27	3.64
64 Kbps	3.32	3.76	3.84	4.68	3.27	3.64
Speed up						
1%	0.00	0.00	1.97	2.27	0.97	1.41
2.5%	0.33	0.00	1.10	1.50	0.00	0.00
5%	0.33	0.29	0.00	0.00	1.16	0.97
Slow down						
1%	0.00	0.00	0.72	2.58	0.87	1.23
2.5%	0.29	0.31	0.00	0.00	0.97	0.00
5%	0.52	0.40	0.00	0.00	0.00	0.00
Adjust volume						
50%	3.37	3.72	3.84	4.69	3.30	3.66
80%	3.34	3.75	3.83	4.70	3.28	3.65
120%	3.34	3.74	3.85	4.67	3.30	3.63
Convert to mono	3.51	3.89	3.91	4.71	3.16	3.62
Downsample						
22 kHz	3.51	3.90	3.82	4.70	3.16	3.63
8 kHz	2.15	2.15	2.88	4.21	1.66	2.39
Radio EQ	3.43	3.81	3.89	4.70	3.40	3.65

Table 5.6 Sensitivity (d') for modified queries

5.3 Noise

The final experiment took three different types of noise and mixed it in to the audio query at three different volume levels before performing the query. The retrieval results are presented in Tables 5.7–5.10.

Adding noise to queries had a significant effect on the accuracy rates of all of the fingerprinting algorithms that were tested. The recall statistics visibly show that the accuracy of all fingerprinting systems increase both as the query length increases, and as the level of added noise is reduced.

For all algorithms the precision and specificity remain similar to the original unmodified queries, indicating that the systems are not making any more false positives or false alarms. The majority of the errors are instead false negatives, resulting in a drop in recall, where the system was unable to identify the recording when given a query. Chromaprint introduces some uncertainty in precision values for shorter queries and louder noise, e.g., Pink noise and Car noise 8 s.

An interesting observation of the precision statistics for the Landmark algorithm is that it actually increases for almost all query modifications, including very noisy queries. Recalling from Section 5.1 that the Landmark algorithm had lower precision than expected because it tended to provide a match with very few matching hashes, it seems that with the noise added to the query the number of matching hashes decreased enough for the landmark system to no longer consider recordings to be the same as the query.

With queries mixed with noise at 0 dB all algorithms perform poorly, but Echoprint manages to edge out the other two. It performs worse on uniform pink noise than on the other two types of noise, which are less uniform. Specificity is high for all algorithms at all types of noise and noise levels, indicating that the added noise is not creating any hashes that match recordings in the reference database.

For shorter length queries, Echoprint has the best recall. It also has a very high precision, close to 100% most of the time. Along with the high specificity, this shows that the algorithm frequently does not find a match, but when it does there is a good chance that the result is correct.

Again, for quiet noise and long queries, Chromaprint comes out on top. For louder and shorter queries, Echoprint is better, followed by Landmark.

	8 s		15 s		30 s	
	P	LL–UL	P	LL–UL	P	LL–UL
Echoprint						
Original query	100	100–100	100	100–100	100	99–100
Pink noise						
0 dB	99	96–100	99	98–99	99	99–100
-10 dB	100	99–100	100	99–100	100	99–100
-20 dB	100	99–100	100	99–100	100	99–100
Car noise						
0 dB	100	99–100	100	99–100	100	99–100
-10 dB	100	100–100	100	100–100	100	100–100
-20 dB	100	100–100	100	100–100	100	100–100
Babble noise						
0 dB	99	99–100	99	99–100	99	99–100
-10 dB	100	100–100	100	100–100	100	99–100
-20 dB	100	100–100	100	100–100	100	99–100
Chromaprint						
Original query	100	98–100	99	99–100	99	99–100
Pink noise						
0 dB	0	0–100	95	73–100	100	97–100
-10 dB	98	88–100	99	99–100	99	99–100
-20 dB	100	98–100	99	99–100	99	99–100
Car noise						
0 dB	100	83–100	99	99–99	99	99–100
-10 dB	100	73–100	99	98–100	99	99–100
-20 dB	99	97–100	99	99–100	99	99–100
Babble noise						
0 dB	100	50–100	100	96–100	99	99–100
-10 dB	100	96–100	99	99–100	99	99–100
-20 dB	100	98–100	99	99–100	99	99–100
Landmark						
Original query	94	93–94	94	94–94	94	94–94
Pink noise						
0 dB	95	90–98	96	95–98	96	96–97
-10 dB	96	95–97	96	95–97	96	95–96
-20 dB	96	95–96	96	95–96	95	95–95
Car noise						
0 dB	96	95–97	96	95–96	95	95–96
-10 dB	95	95–96	95	94–95	94	94–95
-20 dB	95	94–95	94	94–95	94	94–94
Babble noise						
0 dB	96	95–97	96	95–97	96	95–96
-10 dB	96	95–96	96	95–96	95	95–96
-20 dB	95	95–96	95	95–95	95	94–95

Table 5.7 Precision for queries modified with added noise

	8 s		15 s		30 s	
	R	LL–UL	R	LL–UL	R	LL–UL
Echoprint						
Original query	63	63–64	90	90–91	96	96–96
Pink noise						
0 dB	1	1–2	5	5–6	8	8–9
-10 dB	14	13–14	30	29–30	39	39–40
-20 dB	39	38–39	62	62–63	73	72–73
Car noise						
0 dB	25	24–25	48	47–48	58	58–59
-10 dB	64	63–65	84	84–85	91	90–91
-20 dB	77	76–77	92	92–93	96	96–96
Babble noise						
0 dB	11	10–11	20	19–20	18	18–19
-10 dB	35	34–36	52	51–53	55	55–56
-20 dB	60	59–61	77	76–77	82	81–82
Chromaprint						
Original query	3	3–3	96	96–97	99	99–99
Pink noise						
0 dB	0	0–0	0	0–0	2	1–2
-10 dB	0	0–0	19	19–20	48	47–49
-20 dB	2	2–2	72	71–73	90	90–90
Car noise						
0 dB	0	0–0	12	12–13	35	34–36
-10 dB	0	0–0	7	7–8	24	23–24
-20 dB	1	1–1	58	57–59	81	80–81
Babble noise						
0 dB	0	0–0	1	1–1	6	5–6
-10 dB	1	1–1	33	32–34	63	63–64
-20 dB	3	2–3	78	77–78	93	92–93
Landmark						
Original query	63	63–64	88	87–88	94	94–94
Pink noise						
0 dB	1	1–1	4	4–4	16	16–17
-10 dB	7	7–7	21	21–22	47	46–48
-20 dB	22	21–22	46	45–46	72	71–72
Car noise						
0 dB	14	13–14	34	34–35	63	62–64
-10 dB	42	41–42	69	68–70	87	87–88
-20 dB	54	53–54	79	79–80	92	91–92
Babble noise						
0 dB	9	8–9	16	16–17	33	32–34
-10 dB	26	26–27	44	43–44	65	64–66
-20 dB	43	42–44	65	64–66	82	82–83

Table 5.8 Recall for queries modified with added noise

	8 s		15 s		30 s	
	S	LL–UL	S	LL–UL	S	LL–UL
Echoprint						
Original query	99	99–100	99	98–99	98	98–99
Pink noise						
0 dB	100	100–100	100	100–100	100	100–100
-10 dB	100	100–100	100	99–100	99	99–100
-20 dB	99	99–100	99	99–99	99	99–99
Car noise						
0 dB	100	99–100	99	99–100	99	99–99
-10 dB	100	99–100	99	99–99	99	98–99
-20 dB	99	99–100	99	98–99	99	98–99
Babble noise						
0 dB	100	100–100	100	99–100	100	99–100
-10 dB	100	99–100	99	99–100	99	99–99
-20 dB	99	99–100	99	99–99	99	98–99
Chromaprint						
Original query	100	100–100	99	99–100	99	99–100
Pink noise						
0 dB	100	100–100	100	100–100	100	100–100
-10 dB	100	100–100	100	100–100	100	100–100
-20 dB	100	100–100	100	99–100	99	99–100
Car noise						
0 dB	100	100–100	100	100–100	100	100–100
-10 dB	100	100–100	100	100–100	100	100–100
-20 dB	100	100–100	100	100–100	99	99–100
Babble noise						
0 dB	100	100–100	100	100–100	100	100–100
-10 dB	100	100–100	100	100–100	100	99–100
-20 dB	100	100–100	100	99–100	99	99–100
Landmark						
Original query	99	99–100	99	98–99	98	98–99
Pink noise						
0 dB	100	100–100	100	100–100	100	100–100
-10 dB	100	100–100	100	99–100	99	99–99
-20 dB	100	100–100	99	99–100	99	98–99
Car noise						
0 dB	100	100–100	99	99–100	99	99–99
-10 dB	100	99–100	99	99–99	98	98–99
-20 dB	100	99–100	99	98–99	98	98–99
Babble noise						
0 dB	100	100–100	100	100–100	100	99–100
-10 dB	100	100–100	99	99–100	99	99–99
-20 dB	100	99–100	99	99–99	98	98–99

Table 5.9 Specificity for queries modified with added noise

	8 s	15 s	30 s
	d'	d'	d'
Echoprint			
Original query	2.80	3.55	3.90
Pink noise			
0 dB	0.99	1.22	1.55
-10 dB	1.79	2.08	2.19
-20 dB	2.26	2.69	2.91
Car noise			
0 dB	1.97	2.40	2.61
-10 dB	2.95	3.34	3.59
-20 dB	3.26	3.68	3.95
Babble noise			
0 dB	1.60	1.78	1.80
-10 dB	2.28	2.56	2.53
-20 dB	2.79	3.08	3.12
Chromaprint			
Original query	0.46	4.35	4.81
Pink noise			
0 dB	0.00	0.00	0.17
-10 dB	0.00	2.43	2.73
-20 dB	0.25	3.24	3.77
Car noise			
0 dB	0.00	2.33	2.59
-10 dB	0.00	0.87	2.46
-20 dB	0.05	2.98	3.41
Babble noise			
0 dB	0.00	0.00	0.74
-10 dB	0.00	2.59	3.00
-20 dB	0.40	3.36	3.93
Landmark			
Original query	2.82	3.41	3.65
Pink noise			
0 dB	0.00	1.43	1.83
-10 dB	1.62	1.90	2.34
-20 dB	2.03	2.36	2.79
Car noise			
0 dB	1.94	2.16	2.64
-10 dB	2.43	2.85	3.26
-20 dB	2.70	3.07	3.50
Babble noise			
0 dB	1.94	1.89	2.25
-10 dB	2.21	2.35	2.69
-20 dB	2.48	2.75	3.09

Table 5.10 Sensitivity (d') for queries modified with added noise

5.4 Discussion

This evaluation tested three fingerprinting algorithms on a test set of 20,000 recordings. Each test file was modified in a different way to represent the kind of query that a fingerprinting service might receive, including audio that had spectral content removed, altered, or was mixed with varying levels of noise.

The Chromaprint algorithm performed well with almost all modified queries, especially when the query was long. One major failing of the Chromaprint algorithm is that it requires queries to be taken from the beginning of the recording. It is possible that some changes to the matching algorithm for this fingerprinting system would result in high retrieval rates for queries taken from different points in the recording as well.

For short signals, both Echoprint and the Landmark algorithm perform well. The Landmark algorithm has a lower recall, resulting in a significant number of false positive matches. It is likely that the recall of the Landmark algorithm can be improved by adjusting some of the parameters used to generate the hashes from an audio signal.

For query identification on audio that has light to moderate modifications or noise, Echoprint is the recommended fingerprinting system to use. For queries using exact copies of the recordings, Chromaprint is the ideal choice.

Chapter 6

Conclusion and further work

This thesis presented a review of fingerprinting algorithms and developed a new evaluation framework which was used to compare the accuracy of three different fingerprinting algorithms when presented with different audio queries.

Audio fingerprinting algorithms convert audio signals into a sequence of numerical codes that not only uniquely identify individual recordings, but are the same for perceptually similar sounding music. An audio fingerprinting algorithm performs five main steps in generating a fingerprint: preprocessing, framing, transform, feature extraction, and fingerprint generation. The first chapter gave a general background of audio fingerprinting and described the fingerprinting process. Specific audio fingerprinting algorithms were described in Chapter 2 along with a history of audio fingerprinting and technologies that are related to audio analysis and fingerprinting.

Chapter 3 presented an analysis of three audio fingerprinting algorithms that are widely used in industry and academia. The three algorithms, Echoprint, Chromaprint, and the Landmark algorithm perform steps for preprocessing, framing, and transform, but differ in the feature extraction and fingerprint generation stages. The algorithms also use different techniques when identifying queries in a reference database.

We introduced a list of criteria that should be considered when evaluating audio fingerprinting algorithms in order to accurately compare the results of two or more algorithms. Chapter 4 introduced an experimental framework that was designed to perform this evaluation. The framework was developed to allow many different fingerprinting algorithms to be tested simultaneously. The framework allows the same queries to be made to multiple

fingerprinting algorithms to see how they respond to identical stimuli. The framework has a library of alterations that can be made to queries to simulate the way that real-world signals differ from reference audio.

We performed an evaluation of the Echoprint, Chromaprint, and Landmark fingerprinting systems using the evaluation framework. Each of the fingerprinting systems was tested with 63 different types of query. The results of this evaluation were discussed in Chapter 5. The results show that for a fingerprinting environment using audio that is very close to the original signal, the Chromaprint algorithm is an excellent choice, however, it does not work well when presented with audio taken from any point in a recording. For queries that contain a moderate amount of noise, both Echoprint and the Landmark algorithm are good choices, with the Landmark algorithm performing slightly better on shorter queries.

6.1 Contributions

The major contribution of this thesis was the development of a framework for performing repeatable evaluations of audio fingerprinting algorithms. This framework can be used by other researchers while developing new algorithms and for testing their algorithms against other systems. Because the framework is able to repeat the conditions of an experiment it is suitable for evaluating different versions of the same algorithm as well as comparing differing algorithms.

The execution of an evaluation of the three major audio fingerprinting algorithms is also a contribution of this thesis. The success of the evaluation show that a comprehensive experiment can be performed in a controlled manner with known audio and known query modifications. More audio fingerprinting algorithms could be added to this evaluation to obtain more results. The results of the evaluation are useful to people wanting to choose an audio fingerprinting algorithm for their own use.

6.2 Further work

The evaluation framework presented here is expandable and can be used to evaluate other fingerprinting systems. We envisage a MIREX-like evaluation competition (Downie et al. 2005) in which researchers can submit fingerprinting algorithms that are evaluated against other entries.

In order for this kind of evaluation to be performed, an infrastructure around these evaluations would need to be established. One requirement for real-world evaluations is access to a large reference database of audio. We started to move in this direction with our 30,000 recording evaluation, but commercial fingerprinting systems still contain orders of magnitude more songs (millions) in their database. Collecting a large corpus of audio while remaining within distribution rights may be difficult. Testing with such a large database also takes time. Even for our modest test of 20,000 files, a full evaluation for a single type of query modification took over 4 hours. This kind of evaluation can be split over many machines in order to reduce the total time required for the evaluation, but it still represents a large amount of computation time needed to test a wide range of query types.

We would like to see queries that are more representative of real-world environments. For example, the queries that were created by mixing noise with the original query recording were generated synthetically. A more real-world evaluation situation would be to record audio in the environment that we would like to test, for example, with a sound recorder in a car or cafe. The amount of noise in these recordings would differ based on when the recording was made. The recordings would need to be manually classified to ensure that they were evaluated correctly. For queries simulating radio playback we could either record real playback from a radio broadcast, or perform an analysis on a broadcast signal to see how it differs from the signal distributed on CD.

The framework that has been developed is a powerful tool for testing a large number of fingerprinting algorithms and evaluating their accuracy. We hope that by providing direct comparisons of algorithm accuracy in a large-scale public evaluation contest, we can generate competition to advance the speed and accuracy of fingerprinting algorithms further in the future.

References

Agresti, A., and B. A. Coull. 1998. Approximate is better than "exact" for interval estimation of binomial proportions. *The American Statistician* 52 (2): 119–26.

Allamanche, E., J. Herre, O. Hellmuth, B. Fröba, T. Kastner, and M. Cremer. 2001. Content-based identification of audio material using MPEG-7 low level description. In *Proceedings of the International Symposium on Music Information Retrieval*.

Appleby, A. 2009. MurmurHash. Last accessed 15 December 2012, `https://sites.google.com/site/murmurhash/`.

Balado, F., N. Hurley, E. McCarthy, and G. Silvestre. 2007. Performance analysis of robust audio hashing. *IEEE Transactions on Information Forensics and Security* 2 (2): 254–66.

Baluja, S., and M. Covell. 2007. Audio fingerprinting: Combining computer vision data stream processing. In *Proceedings of the IEEE International Conference on Acoustics, Speech, and Signal Processing*, Volume 2, 213–6.

Baluja, S., and M. Covell. 2008. Waveprint: Efficient wavelet-based audio fingerprinting. *Pattern Recognition* 41 (11): 3467–80.

Bartsch, M., and G. Wakefield. 2005. Audio thumbnailing of popular music using chroma-based representations. *Proceedings of the IEEE Transactions on Multimedia* 7 (1): 96–104.

Batlle, E., J. Masip, and E. Guaus. 2002. Automatic song identification in noisy broadcast audio. In *Proceedings of the International Conference on Signal and Image Processing*.

Byrd, D., and T. Crawford. 2002. Problems of music information retrieval in the real world. *Information Processing & Management* 38 (2): 249–72.

Cano, P. 2007. Content-based audio search from fingerprinting to semantic audio retrieval. Ph. D. thesis, Universitat Pompeu Fabra.

Cano, P., E. Batlle, T. Kalker, and J. Haitsma. 2005. A review of audio fingerprinting. *The Journal of VLSI Signal Processing* 41 (3): 271–84.

Cano, P., E. Batlle, H. Mayer, and H. Neuschmied. 2002. Robust sound modeling for song detection in broadcast audio. In *Audio Engineering Society Convention 112*, 1–7.

Catalán, C. 2009. Quality assessment and enhancement of an industrial-strength audio fingerprinting system. Master's thesis, Universitat Pompeu Fabra.

Chandrasekhar, V., M. Sharifi, and D. A. Ross. 2011. Survey and evaluation of audio fingerprinting schemes for mobile query-by-example applications. In *Proceedings of the International Society for Music Information Retrieval Conference*, 801–6.

Chang, S., T. Sikora, and A. Purl. 2001. Overview of the MPEG-7 standard. *IEEE Transactions on Circuits and Systems for Video Technology* 11 (6): 688–95.

Covell, M., and S. Baluja. 2007. Known-audio detection using Waveprint: Spectrogram fingerprinting by wavelet hashing. In *Proceedings of the IEEE International Conference on Acoustics, Speech, and Signal Processing*, Volume 1, 237–40.

Crawford, T., M. Mauch, and C. Rhodes. 2010. Recognizing classical works in historical recordings. In *Proceedings of the International Society for Music Information Retrieval Conference*.

Cremer, M., B. Froba, O. Hellmuth, J. Herre, and E. Allamanche. 2001. Audioid: Towards content-based identification of audio material. In *Audio Engineering Society Convention 110*.

Doets, P., and R. Lagendijk. 2004. Stochastic model of a robust audio fingerprinting system. In *Proceedings of the International Conference on Music Information Retrieval*, 349–52.

Downie, J. 2003. Music information retrieval. *Annual Review of Information Science and Technology* 37 (1): 295–340.

Downie, J., K. West, A. Ehmann, E. Vincent et al. 2005. The 2005 Music Information Retrieval Evaluation eXchange (MIREX 2005): Preliminary overview. In *Proceedings of the International Conference on Music Information Retrieval*, 320–3.

Ellis, D. 2009. Robust landmark-based audio fingerprinting. Last accessed 15 December 2012, `http://labrosa.ee.columbia.edu/matlab/fingerprint`.

Ellis, D. P. W., B. Whitman, T. Jehan, and P. Lamere. 2010. The Echo Nest musical fingerprint. In *Proceedings of the International Society for Music Information Retrieval Conference*.

Ellis, D. P. W., B. Whitman, and A. Porter. 2011. Echoprint—an open music identification service. In *Proceedings of the International Society for Music Information Retrieval Conference*.

Fenet, S., G. Richard, and Y. Grenier. 2011. A scalable audio fingerprint method with robustness to pitch-shifting. In *Proceedings of the International Society for Music Information Retrieval Conference*.

Foote, J. 1997. Content-based retrieval of music and audio. In *Proceedings of SPIE Multimedia Storage and Archiving Systems II*, Volume 3229, 138–47.

Foote, J. 1998. An overview of audio information retrieval. *ACM Multimedia Systems* 7 (1): 2–10.

Foote, J. 2000. Arthur: Retrieving orchestral music by long-term structure. In *Proceedings of the International Symposium on Music Information Retrieval*.

Fragoulis, D., G. Rousopoulos, T. Panagopoulos, C. Alexiou, and C. Papaodysseus. 2001. On the automated recognition of seriously distorted musical recordings. *IEEE Transactions on Signal Processing* 49 (4): 898–908.

Ghias, A., J. Logan, D. Chamberlin, and B. Smith. 1995. Query by humming: Musical information retrieval in an audio database. In *Proceedings of the Third ACM International Conference on Multimedia*, 231–6.

Gomes, L. d. C., P. Cano, E. Gómez, M. Bonnet, and E. Batlle. 2003. Audio watermarking and fingerprinting: For which applications? *Journal of New Music Research* 32 (1): 65–81.

Gomez, E., P. Cano, L. Gomes, E. Batlle, and M. Bonnet. 2002. Mixed watermarking-fingerprinting approach for integrity verification of audio recordings. In *Proceedings of the International Telecommunications Symposium*.

Graps, A. 1995. An introduction to wavelets. *IEEE Conference on Computational Science & Engineering* 2 (2): 50–61.

Haitsma, J., and T. Kalker. 2002. A highly robust audio fingerprinting system. In *Proceedings of the International Conference on Music Information Retrieval*, 144–8.

Haitsma, J., T. Kalker, and J. Oostveen. 2001. Robust audio hashing for content identification. In *Proceedings of the International Workshop on Content-Based Multimedia Indexing*, Volume 4, 117–24.

Herre, J., E. Allamanche, and O. Hellmuth. 2001. Robust matching of audio signals using spectral flatness features. In *IEEE Workshop on the Applications of Signal Processing to Audio and Acoustics*, 127–30.

Holm, F., and W. T. Hicken. 2003. Audio fingerprinting system and method. US Patent, 7,013,301, filed, Mar. 14, 2006, and issued, Mar. 14, 2006.

Jang, D., C. Yoo, S. Lee, S. Kim, and T. Kalker. 2009. Pairwise boosted audio fingerprint. *IEEE Transactions on Information Forensics and Security* 4 (4): 995–1004.

Jehan, T. 2005. Creating music by listening. Ph. D. thesis, Massachusetts Institute of Technology.

Kashino, K., G. Smith, and H. Murase. 1999. Time-series active search for quick retrieval of audio and video. In *Proceedings of the IEEE International Conference on Acoustics, Speech, and Signal Processing*, Volume 6, 2993–6.

Kastner, T., E. Allamanche, J. Herre, O. Hellmuth, M. Cremer, and H. Grossmann. 2002. MPEG-7 scalable robust audio fingerprinting. In *Audio Engineering Society Convention 112*.

Ke, Y., D. Hoiem, and R. Sukthankar. 2005. Computer vision for music identification. In *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, 597–604.

Kimura, A., K. Kashino, T. Kurozumi, and H. Murase. 2001. Very quick audio searching: introducing global pruning to the time-series active search. In *Proceedings of the IEEE International Conference on Acoustics, Speech, and Signal Processing*, Volume 3, 1429–32.

Kurth, F., and M. Muller. 2008. Efficient index-based audio matching. *IEEE Transactions on Audio, Speech, and Language Processing* 16 (2): 382–95.

Lalinský, L. 2012. Chromaprint. Last accessed 15 December 2012, `http://acoustid.org/chromaprint`.

Liu, Y., K. Cho, H. Yun, J. Shin, and N. Kim. 2009. DCT based multiple hashing technique for robust audio fingerprinting. In *Proceedings of the IEEE International Conference on Acoustics, Speech and Signal Processing*, 61–4.

Lutz, S. 2009. Hokua–a wavelet method for audio fingerprinting. Master's thesis, Brigham Young University.

Macmillan, N., and C. Creelman. 1991. *Detection theory: A user's guide*. Cambridge: Cambridge University Press.

Mahedero, J., V. Tarasov, E. Batlle, E. Guaus, and J. Masip. 2004. Industrial audio fingerprinting distributed system with CORBA and web services. In *Proceedings of the International Conference on Music Information Retrieval*.

McKay, C., D. McEnnis, and I. Fujinaga. 2006. A large publicly accessible prototype audio database for music research. In *Proceedings of the International Conference on Music Information Retrieval*, 160–3.

Mıhçak, M., and R. Venkatesan. 2001. A perceptual audio hashing algorithm: A tool for robust audio identification and information hiding. In *Proceedings of the International workshop on Information Hiding*, 51–65.

Miller, M., M. Rodriguez, and I. Cox. 2005. Audio fingerprinting: Nearest neighbor search in high dimensional binary spaces. *The Journal of VLSI Signal Processing* 41 (3): 285–91.

Miotto, R., and N. Orio. 2008. A music identification system based on chroma indexing and statistical modeling. In *Proceedings of the International Conference on Music Information Retrieval*, 301–6.

Müller, M., F. Kurth, and M. Clausen. 2005. Audio matching via chroma-based statistical features. In *Proceedings of the International Conference on Music Information Retrieval*, 288–95.

Neuschmied, H., H. Mayer, and E. Batlle. 2001. Content-based identification of audio titles on the internet. In *Proceedings of the First International Conference on Web Delivering of Music*, 96–100.

Papaodysseus, C., G. Roussopoulos, D. Fragoulis, T. Panagopoulos, and C. Alexiou. 2001. A new approach to the automatic recognition of musical recordings. *Journal of the Audio Engineering Society* 49 (1): 23–35.

Poliner, G., D. Ellis, A. Ehmann, E. Gomez, S. Streich, and B. Ong. 2007. Melody transcription from music audio: Approaches and evaluation. *IEEE Transactions on Audio, Speech, and Language Processing* 15 (4): 1247–56.

Portnoff, M. 1981, jun. Time-scale modification of speech based on short-time Fourier analysis. *IEEE Transactions on Acoustics, Speech and Signal Processing* (3): 374–90.

Pye, D. 2000. Content-based methods for the management of digital music. In *Proceedings of the IEEE International Conference on Acoustics, Speech, and Signal Processing*, Volume 6, 2437–40.

Rabiner, L., and B. Juang. 1986. An introduction to hidden markov models. *IEEE ASSP Magazine* 3 (1): 4–16.

Ramstad, T., and J. Tanem. 1991. Cosine-modulated analysis-synthesis filterbank with critical sampling and perfect reconstruction. In *Proceedings of the International Conference on Acoustics, Speech, and Signal Processing*, 1789–92. IEEE.

R.I.A.A. 2001. Request for information on audio fingerprinting technologies. Last accessed 15 December 2012, `http://www.ifpi.org/content/section_news/20010615.html`.

Seo, J., J. Haitsma, and T. Kalker. 2002. Linear speed-change resilient audio fingerprinting. In *Proceedings of the IEEE Workshop on Model based Processing and Coding of Audio*.

Shrestha, P., and T. Kalker. 2004. Audio fingerprinting in peer-to-peer networks. In *Proceedings of the International Conference on Music Information Retrieval*.

Song, J., S. Bae, and K. Yoon. 2002. Mid-level music melody representation of polyphonic audio for query-by-humming system. In *Proceedings of the International Conference on Music Information Retrieval*.

Subramanya, S., R. Simha, B. Narahari, and A. Youssef. 1997. Transform-based indexing of audio data for multimedia databases. In *Proceedings of the IEEE International Conference on Multimedia Computing and Systems*, 211–18.

Wang, A. 2003. An industrial strength audio search algorithm. In *Proceedings of the International Conference on Music Information Retrieval*.

Wang, A. 2006. The Shazam music recognition service. *Communications of the ACM* 49 (8): 44–8.

Wold, E., T. Blum, D. Keislar, and J. Wheaten. 1996. Content-based classification, search, and retrieval of audio. *IEEE Multimedia* 3 (3): 27–36.

Yang, C. 2001. Music database retrieval based on spectral similarity. Technical Report 2001-14, Stanford InfoLab.

Yoshii, K., and M. Goto. 2008. Music thumbnailer: Visualizing musical pieces in thumbnail images based on acoustic features. In *Proceedings of the International Conference on Music Information Retrieval*.