Audio Hashprints: Theory & Application

Timothy Tsai

Electrical Engineering and Computer Sciences
University of California at Berkeley

Technical Report No. UCB/EECS-2016-185
http://www2.eecs.berkeley.edu/Pubs/TechRpts/2016/EECS-2016-185.html

December 1, 2016

Audio Hashprints: Theory & Application

by

Timothy Jwoyen Tsai

A dissertation submitted in partial satisfaction of the

requirements for the degree of

Doctor of Philosophy

in

Engineering — Electrical Engineering and Computer Sciences

in the

Graduate Division

of the

University of California, Berkeley

Committee in charge:

Professor Nelson Morgan, Chair
Professor Avideh Zakhor
Professor David Brillinger

Spring 2016

Audio Hashprints: Theory & Application

Abstract

Audio Hashprints: Theory & Application

by

Timothy Jwoyen Tsai

Doctor of Philosophy in Engineering — Electrical Engineering and Computer Sciences

University of California, Berkeley

Professor Nelson Morgan, Chair

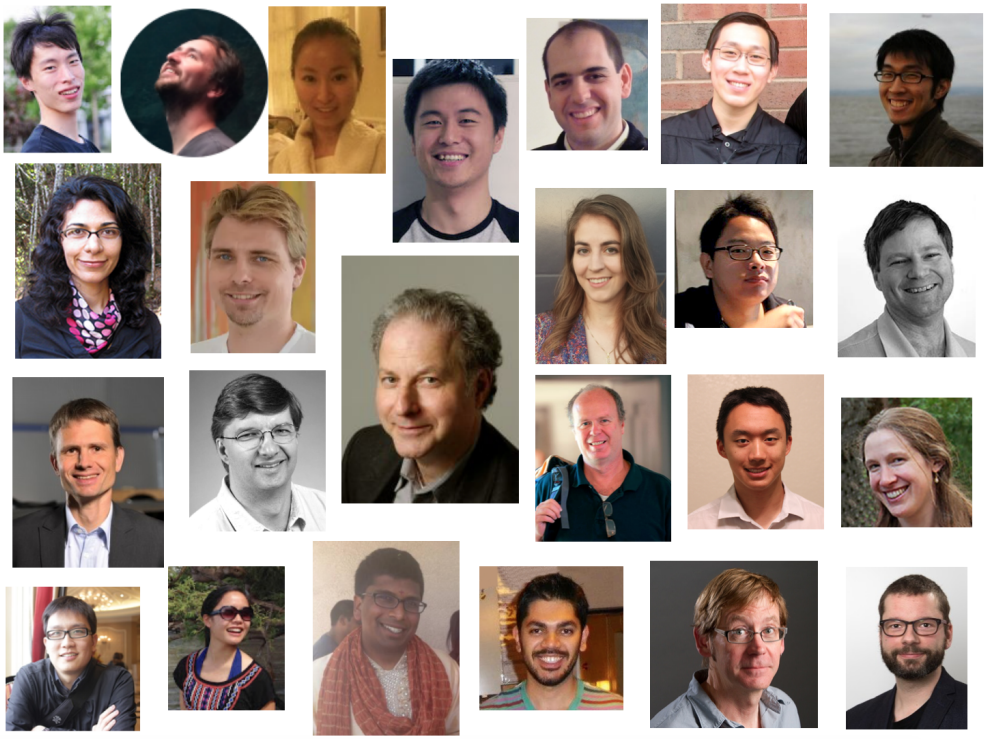
We rely heavily on search engines like Google to navigate millions of webpages, but a lot of content of interest is multimedia, not text data. One important class of multimedia data is audio. How can we search a database of audio data? One of the main challenges in audio search and retrieval is to determine a mapping from a continuous time-series signal to a sequence of discrete symbols that are suitable for reverse-indexing and efficient pairwise comparison. This talk introduces a method for learning this mapping in an unsupervised, highly adaptive way, resulting in a representation which we call audio hashprints. We will discuss the theoretical underpinnings that determine how useful a particular representation is in a retrieval context, and we show how hashprints are a suitable representation for tasks requiring high adaptivity. We investigate the performance of the proposed hashprints on two different audio search tasks: synchronizing consumer recordings of the same live event using audio correspondences, and identifying a song at a live concert. Using audio hashprints, we demonstrate state-of-the-art performance on both tasks.

To my parents, for all of their love and support

Contents

List of Figures

List of Tables



Acknowledgments

I am very grateful for the wonderful friends and colleagues that have made my time at Berkeley so wonderful. A few special thanks are in order. Thanks to Adam Janin for answering so many millions of my questions and immediately shooting down the 99% of ideas that were doomed to fail. You shortened my Ph.D. by about 5 years. Thanks to Andreas Stolcke for taking an interest in my work and providing lots of mentorship beyond my internship at Microsoft. I have grown so much from our collaborations. Thanks to Meinard Müller for being so incredibly generous with his time and expertise and for helping me to get involved in the music information retrieval community. I hope to pay your kindness forward to students in the next generation. And, finally, thanks to Nelson Morgan for being such a wonderful advisor, colleague, and supporter throughout my graduate studies. I had heard many scary stories of all the ways Ph.D. advisors make their students' lives miserable. Not only were they not true, my experience was just the opposite.

Chapter 1

Introduction & Background Work

This dissertation investigates the audio search and retrieval problem. In this introductory chapter, we will first provide the context and motivation behind the problem (section 1.1), describe the general problem statement (section 1.2), give a high-level overview of two relevant areas of previous work (sections 1.3 and 1.4), and lay out a roadmap for the remainder of the dissertation (section 1.5).

1.1 Context & Motivation

As a way of providing context and motivation for the problem of interest, let us begin by pointing out three trends in our society.

- **Trend #1: Accessing information efficiently.** We live in a world that places great value on accessing information efficiently. People don't want to go to Blockbuster to get a movie; they want to watch the movie on Netflix *right now*. People don't want to do research comparing ticket prices on different airline websites; they want that resesarch done on Expedia *right now*. People don't want to wait until they get back to their desktop computer to check their Facebook account; they want to see an update on their phones *right now*. If you can help people access information or content more efficiently, you can probably build a business on it. Our society highly values efficient information access.

- **Trend #2: Big data.** Every time we search for a piece of information on the web, we are stepping into a stadium full of books. The only way that we can find what we are looking for is to use search engines like Google to search through billions of webpages. The amount of data so far exceeds our capacity to wade through it ourselves that we are completely dependent on the *tools* we use to find what we are looking for.

- **Trend #3: Growth of multimedia.** Even though we rely heavily on search engines like Google to find relevant content, a lot of the data nowadays is not text data – it's

multimedia data. We live in an age of Youtube, Netflix, iTunes, Spotify, and Pandora. Cisco estimates that consumer internet video traffic made up 64% of all consumer internet traffic in 2014, and this percentage is expected to grow to 80% by 2019.[1]

The problem we will investigate grows out of the intersection of these three trends. There is a lot of multimedia data out there, and we are dependent on our tools to find what we are looking for in an efficient and reliable way.

1.2 Problem Statement

The question we will ask in this thesis is, "How can I search audio?" The general problem statement is illustrated in figure 1.1. Given a short, noisy query, we would like to find a match in a database of audio recordings. For example, let's say that a person goes to a Taylor Swift concert and records 6 seconds of the live performance on his or her cell phone. This short, noisy audio recording is the query. The database might be a collection of songs from various artists' studio albums. Given the 6 second recording of the live performance, we would like to identify the name of the song, get instant access to purchasing the original studio recording, or perhaps get the lyrics so that we can sing along. This live song identification scenario is one specific example of the general audio search problem shown in figure 1.1, and we will investigate this specific scenario later in this thesis.

One way we could approach this problem is to do a brute-force, exhaustive search. For example, we could simply compute the cross correlation between the query and all of the audio recordings in the database, and then identify the location in the database that has the highest cross correlation. This might work, but it would be very inefficient. And accessing information *efficiently* is one of our goals. Can we do any better?

To gain some inspiration, let's consider the following analogy. Imagine that a person comes to you with a book in their hand and they say, "There are one or more sentences in this book that contain the phrase 'Armstrong waited for Ullrich.' I would like you to identify where these sentences are located." One way that you could approach this problem is to read the book from cover to cover in order to find the sentences containing the target phrase. This approach is analagous to the cross correlation appraoch described above. But a much more efficient way to solve this problem would be to turn to the word index at the end of the book, look up all of the pages in which the word 'Armstrong' occurs, and then scan only those selected pages of interest. What would have taken many hours to do using a reading approach only takes a few moments to do using a word index. Indeed, the concept of reverse indexing is a central idea behind search engines like Google. Clearly, using a word index helps us solve this text search problem much more efficiently. The question is, "What does this word index correspond to in the audio domain?"

[1] `http://www.cisco.com/c/en/us/solutions/collateral/service-provider/ip-ngn-ip-next-generation-network/white_paper_c11-481360.html`

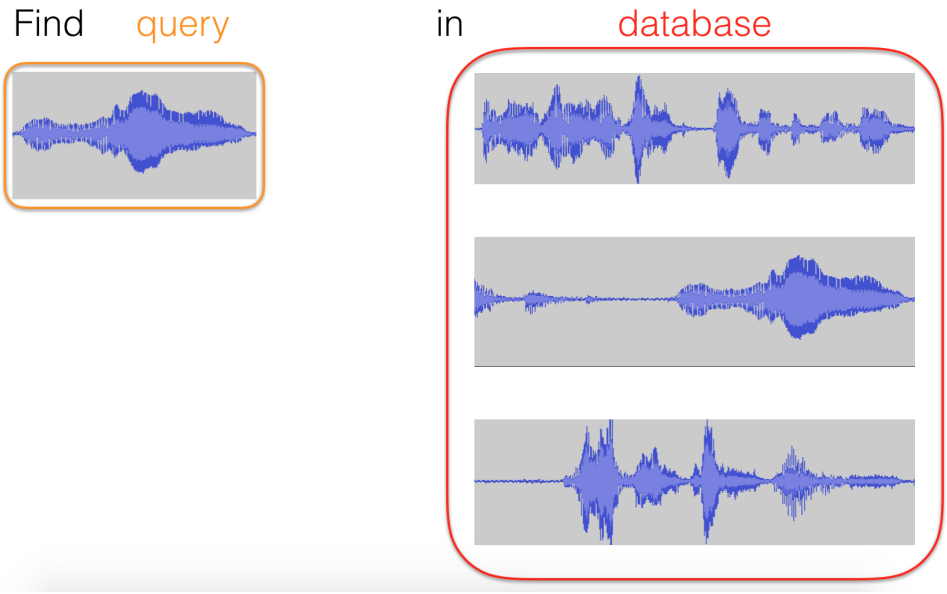


Figure 1.1: The general problem statement. Given a short, noisy audio query, we would like to find a match in a database of audio recordings.

One key factor that allows us to build a word index is the fact that words are discrete quantities. Discrete quantities like integers can be indexed; continuously-varying quantities like floating point numbers cannot. There must be a discretization process if we are to use indexing techniques. In other words, we would like to map a continuously-varying time-domain representation of audio to a sequence of discrete values. Once we have represented audio as a sequence of discrete values, then we *can* build an index on these discrete values. Figure 1.2 illustrates one such mapping, where the discrete values are represented as different colored boxes. These discrete values are often called "audio fingerprints" because they are a compact descriptor that can be used to uniquely identify segments of audio. For reasons that we will explain in chapter 2, however, we will refer to these discrete values as hash keys.

We have refined the problem statement to this: How can we map an audio signal to a sequence of hash keys in a way that facilitates efficient, robust search and retrieval? This is the specific question that we will address in this thesis. This dissertation proposes one such mapping that leads to a representation of audio which we call audio hashprints. Starting from first principles, we will motivate the design of this mapping, explain how hashprints are computed, and demonstrate how this representation can help achieve state-of-the-art performance in two different audio search and retrieval applications.

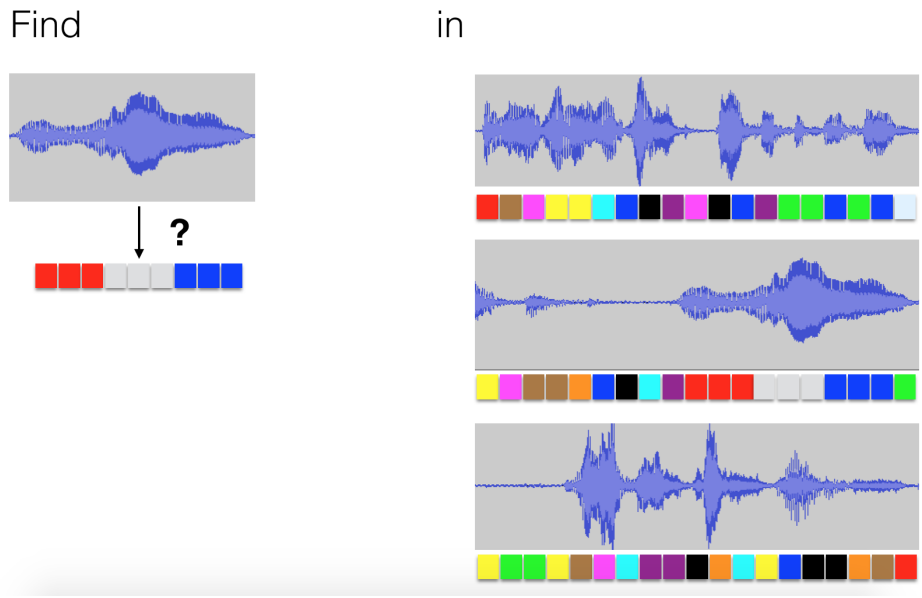


Figure 1.2: The refined problem statement. How can we map an audio signal to a sequence of hash keys in a way that facilitates audio search and retrieval?

1.3 Background Work: Audio Fingerprinting

There are two areas of background work that fit the general problem statement shown in figure 1.1: audio fingerprinting and audio matching. We will discuss audio fingerprinting in this section and audio matching in the next section. In addition to fragment-level retrieval tasks like audio fingerprinting and audio matching, there are also document-level retrieval tasks such as version identification and cover song retrieval. These tasks, however, will not be discussed in this chapter.

Problem definition. Audio fingerprinting (also commonly referred to as audio identification or exact-match audio search) is a special case of the general audio search problem where the goal is to find an exact match. Even though the query may have various types of noise and distortion, the underlying source signal is identical to the matching segment of the database. Two well-known applications of audio fingerprinting are music identification and copyright detection. The goal of music identification is to recognize a song playing on the radio by recording a short segment of it on a cell phone. The goal of copyright detection is to determine when a user uploads any video or audio content that is covered by a copyright. Note that in both of these applications, the type of noise and distortion is different. In the music identification scenario, there is additive noise from other sound sources in the environment, distortion from the acoustics of the room, and perhaps pitch-shifting on the audio signal in order to fit the song within an allotted broadcast timeframe. In the copyright detection scenario, the audio may be compressed or encoded in different formats, it may be

mixed with other audio tracks or sounds, or it may have been affected by frame dropping or audio equalization. Despite the differences in the types of noise and distortion, these are both examples of audio fingerprinting because we are looking for an exact match: the song playing on the radio is exactly the same as the original studio recording, and the uploaded video content is exactly the same as the original copyrighted movie. Audio fingerprinting means exact match.

Interest in industry & academia. Audio fingerprinting has received a lot of interest in both industry and academia. There are several commercially successful apps like Shazam and SoundHound that do music identification, and other similar apps like Intonow, Viggle, and MusicID. A lot of academic work in the audio fingerprinting literature has come out of companies like Google [3][4], Philips [32][33], Telefonica [2][101], Microsoft [35], and Gracenote [12]. Research on this topic has also benefited from organized evaluations like the TRECVID content based copy detection task [69], which provided an evaluation benchmark to compare the performance of different approaches [2][64][34][79][31][101][97][59][55][68][44].

Other applications. While most of the development in audio fingerprinting came from music identification and copyright detection, many other applications have been explored. Some of these applications include detecting repeating objects in audio streams [48][27][72][35][66], recognizing a TV channel in real-time [7], synchronizing two different versions of a movie [19] or two TV audio streams [18] or a music video and a studio album track [56], and performing self-localization of multiple recording devices [38]. Several other relevant applications of fingerprinting will be discussed in the application chapters.

Approaches. There are a wide range of approaches that have been explored in the audio fingerprinting literature. Rather than trying to explain the minutiae of individual approaches, we will instead describe the broad landscape of previous work by discussing three major factors or design decisions along which these approaches vary. In keeping with the focus of this dissertation, we will limit our discussion to the actual fingerprint representation itself, though of course a complete audio fingerprinting system will have other system components like the search mechanism.

- **Factor #1: Continuous vs discrete.** The first major factor is whether the representation is continuous-valued or discrete-valued. A vector of floating point numbers is an example of a continuous-valued representation, whereas a binary code would be a discrete-valued representation. The advantages of a continuous-valued representation are greater precision and ease of mathematical manipulation (when considering, say, an optimization problem). The advantage of a discrete-valued representation, as discussed earlier in this chapter, is that it can be indexed. There are also other advantages to using a binary code representation, which will be discussed further in chapter 4.

- **Factor #2: Threshold-based vs value-based.** Among the approaches that adopt a discrete representation, the approaches can be divided into what we will call threshold-based and value-based methods. A threshold-based method is one in which each bit of the representation is derived by computing some feature of interest and then applying

a hard threshold. This feature of interest could be, for example, the change in subband energies [32], spectral subband moments [49], or chroma [25]. A value-based method is one in which some features of interest are computed, and the values of the features themselves are directly encoded in the binary representation. One very commonly adopted approach is to encode the location of maxima, such as the absolute or relative location of spectral peaks [99][28][26][92], the location of maxima in wavelet coefficients [3][4], or the location of local spectral luminance maxima [87]. Note that these two approaches are not necessarily incompatible. The fingerprints proposed by Anguera et al. [2], for example, encode the location of a spectral peak for part of the representation, and adopt a threshold-based approach for the other bits.

- **Factor #3: Design method.** Another way to cluster fingerprint representations is to separate them by their design method. A majority of fingerprint representations are the result of manual design. These approaches often use features that have proven to be useful in other contexts, have useful mathematical properties, or carry some intuitive advantage. For example, spectral peaks are frequently used (e.g. [99][28][26][92]) because they are the most robust part of the signal to additive noise – if we were to increase the noise floor, the spectral peak would be the last part of the signal to be submerged. Other examples include methods based on modulation frequency features [94], chroma [25][57], spectral flatness [1][36], and spectral subband centroids and moments [82][83]. Note that value-based methods that encode the location of a maxima are almost all manually designed representations, since such quantities are generally not very amenable to mathematical optimization procedures. A second category of fingerprint representations are through a supervised learning process. Several works, for example, define a family of features and use boosting techniques to select the features that yield a maximally robust fingerprint [45][42][49]. A third category of fingerprint representations are through an unsupervised learning process. Many of these works combine a manually designed feature with an unsupervised learning method. For example, Ngo et al. [64] propose a bag-of-audio words representation by applying k-means clustering to standard MFCC features.

Table 1.1 shows a wide range of approaches in the audio fingerprinting literature. The leftmost column identifies the work by author and year, along with the bibliographic reference. Columns 2, 3, and 4 describe the approach along the three factors discussed above. Column 5 indicates whether the approach uses indexing techniques. The rightmost column includes a short description of the feature representation. This table is not meant to be an exhaustive list of all approaches, but rather a large sample that is representative of the literature.

Table 1.1: An overview of the audio fingerprinting literature. See text for an explanation of the columns.

Identifier	**cont/ discr**	**thresh/ value**	**design method**	**index**	**feature**
Allamanche01 [1]	cont	-	manual	no	MPEG-7 low level descriptors
Haitsma02 [32]	discr	thresh	manual	yes	changes in subband energy
Sukittanon02 [94]	cont	-	manual	no	modulation frequency features
Burges03 [9]	cont	-	super- vised	no	convolutional neural network
Wang03 [99]	discr	value	manual	yes	spectral peak pairs
Ke05 [45]	discr	thresh	super- vised	yes	boosted Viola-Jones filters
Ramalingam05 [75]	cont	-	manual	no	GMM on spectral centroid, crest factor, entropy, MFCC
Seo05 [82]	cont	-	manual	no	spectral subband centroids
Seo05 [83]	cont	-	manual	no	spectral subband moments
Herley06 [35]	cont	-	manual	no	Bark band correlation
Ibarrola06 [41]	discr	thresh	manual	no	entropy delta
Park06 [70]	discr	thresh	manual	yes	subband energy differences
Baluja07 [3]	discr	value	manual	yes	Haar wavelet coefficient maxima
Kim07 [49]	discr	thresh	super- vised	no	boosted spectral subband moments
Lebosse07 [51]	discr	thresh	manual	no	subband energy around energy peaks
Baluja08 [4]	discr	value	manual	yes	Haar wavelet coefficient maxima
Jang09 [42]	discr	thresh	super- vised	no	boosted filters

Continued on next page

Table 1.1 – continued from previous page

Identifier	**cont/ discr**	**thresh/ value**	**design method**	**index**	**feature**
Liu09 [54]	discr	thresh	manual	yes	changes in modulation frequency features
Ngo09 [64]	discr	value	manual/ unsuper-vised	yes	bag of MFCC words
Saracoglu09 [79]	discr	thresh	manual	yes	subband energy differences
Cotton10 [13]	discr	value	manual/ unsuper-vised	yes	pairs of sparse Gabor dictionary elements
Dupraz10 [20]	discr	value	manual	yes	spectral peaks
Liu10 [55]	discr	value	manual	yes	bag of words, MFCC & RASTA-PLP
Mukai10 [59]	discr	value	manual	yes	VQ subband energy
Son10 [91]	discr	thresh	manual	no	applying mask to Haitsma fingerprint
Uchida10 [97]	discr	thresh	manual	yes	filterbank energy differences
Younessian10 [101]	discr	thresh	manual	yes	subband energy changes
Fenet11 [26]	discr	value	manual	yes	spectral peak pairs
Liu11 [53]	discr	thresh	manual/ unsuper-vised	yes	PCA on MDCT, MFCC, MPEG-7, chroma; QUC-tree
Ramona11 [76]	cont	-	manual	no	modulation frequency features at onsets
Shi11 [87]	discr	value	manual	yes	spectral luminance maxima
Anguera12 [2]	discr	hybrid	manual	yes	spectral peak, local energy differences
Fenet12 [27]	discr	value	manual/ unsuper-vised	yes	sparse decomposition into MDCT dictionary elements

Continued on next page

Table 1.1 – continued from previous page

Identifier	cont/ discr	thresh/ value	design method	index	feature
Jegou12 [44]	discr	value	manual	yes	quantized mel spectrogram energy values
Radhakrishnan12 [72]	discr	thresh	manual	yes	random projections on coarse spectrogram
Coover14 [12]	discr	thresh	manual	yes	applying mask to Haitsma fingerprint
Khemiri14 [48]	discr	value	manual/ unsuper- vised	no	HMM, MFCCs
Malekesmaeili14 [57]	cont	-	manual	no	DCT coefficients of patches in time-chroma plane
Ouali14 [68]	discr	thresh	manual	no	binarized spectrogram
Seo14 [81]	discr	thresh	manual	yes	applying mask to Haitsma fingerprint
Six14 [89]	discr	value	manual	yes	spectral peak triples
Sonnleitner14 [92]	discr	value	manual	yes	spectral peak quads
George15 [28]	discr	value	manual	yes	spectral peaks
Nagano15 [63]	discr	value	manual	yes	VQ normalized spectrum

1.4 Background Work: Audio Matching

Problem definition. Audio matching (also commonly referred to as cross-version retrieval) is another special case of the general audio search problem where the goal is to find an item in the database that is *similar* to the query. Unlike the audio fingerprinting scenario, the underlying source signal in the query is *not* identical to the matching portion of the database. For example, a person might record a short segment of a live symphony performance and would like to retrieve other recordings of the same symphony. In this case, the query is similar to the database items insofar as they come from the same symphony, but the underlying signals will be different due to differences in tempo, articulation, phrasing, balance, dynamics, and other aspects of musical interpretation. Another example is if a concertgoer records a short segment of a live performance from a band and would like to purchase the original

Identifier	cont/ discr	thresh/ value	design method	index	feature
Dannenberg03 [14]	cont	-	manual	no	chroma
Hu03 [39]	cont	-	manual	no	chroma
Muller05 [62]	cont	-	manual	no	short-time chroma stats
Muller05 [61]	cont	-	manual	no	short-time chroma stats
Ibarrola06 [41]	discr	thresh	manual	no	entropy delta
Casey08 [11]	discr	thresh	manual	yes	LSH on chroma shingles
Kurth08 [50]	?	?	?	yes	?
Grosche12 [29]	discr	thresh	manual	yes	LSH on chroma-like shingles
Grosche12 [30]	discr	value	manual	yes	pairs of spectral peaks
Fenet13 [25]	discr	thresh	manual	yes	thresholded chroma at energy onsets
Rafii14 [74]	discr	thresh	manual	no	binarized CQT

Table 1.2: An overview of the audio matching literature. The columns match those in table 1.1.

studio recording. In this case, the query is similar to the matching database item insofar as they are both performances of the same song, but the underlying signals will be different due to differences in tempo, arrangement, instrumentation, and (lack of) digital audio effects or editing. This latter example is one of the two applications we will investigate later in this work. Audio matching means similarity match.

Definition of similarity. Note that the definition of what is 'similar' is task-specific. In the case of classical music, similarity is pretty clearly defined: two recordings are similar if they are performances of the same piece. We expact these audio signals to still have strong resemblance to one another, since they will have the exact same sequence of notes. In other genres of music, however, similarity may be much more subjective. Two different performances of the same jazz standard, for example, may be drastically different in time signature, instrumentation, style, and mood. Because queries are very short and because greater differences make the problem harder, most works in audio matching have primarily focused on classical music, where the differences are more predictable and controlled.

Approaches. Many audio matching approaches adopt techniques and ideas from the audio fingerprinting literature. However, because the underlying signals in the query and database are no longer identical, indexing techniques tend to be less effective due to the lack of specificity in the match. As a result, many audio matching approaches do not use indexing techniques, but instead use some form of exhaustive matching such as cross correlation or dynamic time warping. Other approaches [11] adopt probabilistic methods like locality

sensitive hashing to find nearest neighbors in high dimensional spaces. The combination of the above factors has limited most audio matching applications to small- and medium-sized databases. Scaling such approaches to large databases has proved to be a major challenge.

Table 1.2 shows a range of approaches in the audio matching literature. The structure of the table matches that in table 1.1 in order to facilitate comparison. As we can see, fewer of the approaches use indexing techniques for the reasons described above. We also note that the features tend to be focused on pitch-related information like chroma features, since we are trying to find matches that are similar in a musical sense.

1.5 Dissertation Structure

Figure 1.3 shows the structure of this thesis. The main body of this dissertation (chapters 2 through 6) has three parts. In the first part, we will lay a foundation of theory. In chapters 2 and 3, we will address to relevant theoretical questions of interest to us: "What makes a good hash key?" and "What makes a good hash bit?" Since a hash key is simply a collection of hash bits, these two questions essentially boil down to the questions "What makes a good team?" and "What makes a good individual?" In the second part (chapter 4), we will introduce audio hashprints. Audio hashprints are a representation of audio that are designed to facilitate audio search and retrieval. The design of the hashprints builds upon the key insights from our foundation of theory. The third part represents application. In chapters 5 and 6, we will see how hashprints can be used in two different application scenarios: aligning meeting recordings and live song identification.

The metaphor of building blocks shown in figure 1.3 was chosen for two reasons. First, it illustrates the relationship *between* the different parts of the dissertation. The audio hashprint representation is built upon the foundation of theory, and the applications in turn depend on the hashprint representation of audio. Second, it illustrates the fact that hashprints straddle the line between theory and application. They must be grounded in theory; they must also be useful in practice.

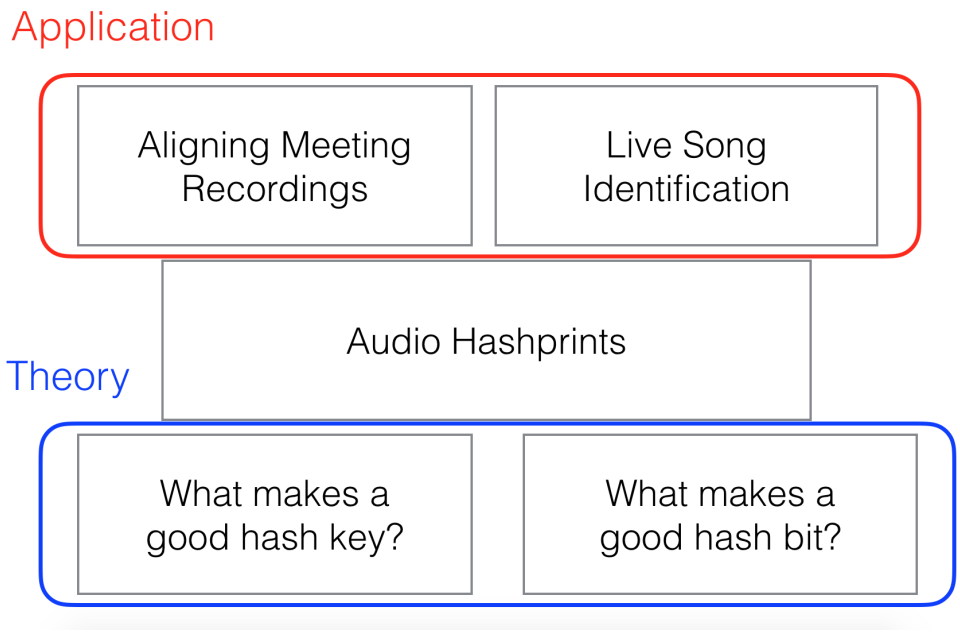


Figure 1.3: Dissertation structure. In the first part of the thesis, we will lay down a foundation of theory by considering two questions of interest. In the second part, we will propose a representation of audio called audio hashprints that build upon our foundation of theory. In the third part, we will see how audio hashprints can be used to solve two different tasks in a robust and efficient way.

Chapter 2

Theory: What Makes A Good Hash Key?

In this chapter, we will investigate the first theoretical question of interest: "What makes a good hash key?" This question will provide the first of two building blocks that will provide a foundation for all that follows. We will first define the problem statement (section 2.1), point out two incorrect answers to the question (sections 2.2 and 2.3), present one reasonable, intuitive answer to the question (section 2.4), and then offer experimental evidence to validate this proposed metric (section 2.5).[1]

2.1 Problem Statement

The question we would like to answer is, "What makes a good hash key?" In order to make this question precise, we must first define two terms: "hash key" and "good".

Here, "hash key" refers to a representation of audio that can be used to look something up in a hash table. In the illustration of the problem shown in figure 2.1, the query audio recording and the database audio recordings have all been mapped to sequences of discrete symbols, which are represented as different colored boxes. The hash keys are the colored boxes. They are the result of applying a mapping which takes a time-domain representation of audio as input, and outputs a sequence of discrete values.

It is useful to point out the relationship between the terms hash key and audio fingerprint. In the context of an audio fingerprinting application, the hash keys are often referred to as audio fingerprints. There is a reason why we adopt a different terminology: the discrete representation that we will propose in this thesis will be applied not only in a fingerprinting application, but also in a nonexact-match audio retrieval problem. Since the term "audio fingerprint" implicitly suggests that the problem is an exact-match fingerprinting application,

[1] Much of the content in this chapter was published in a conference paper entitled "An Information-Theoretic Metric of Fingerprint Effectiveness" [95].

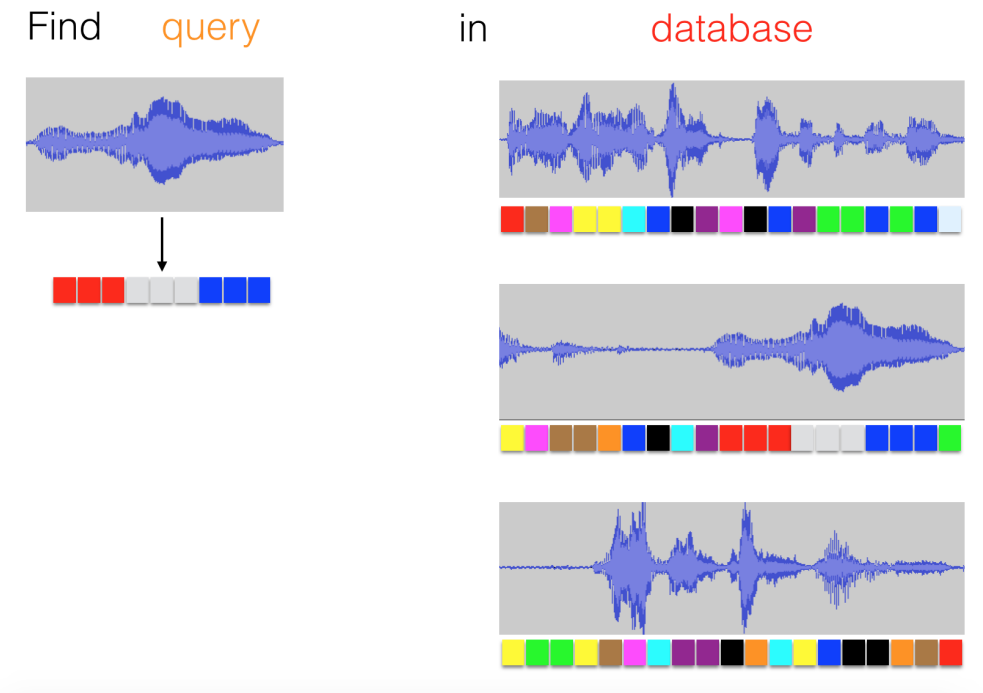


Figure 2.1: Problem statement. Given a short audio query, we would like to find a match in a database of audio recordings. Here, audio has been mapped into sequences of discrete symbols represented as colored boxes.

we have introduced a more generic term "hash key" that makes fewer assumptions about the nature of the retrieval problem. Audio fingerprints are hash keys in a fingerprinting context.

The term "good" refers to the purpose of the hash key: to facilitate efficient retrieval. So, a good hash key is a discrete representation of audio that allows us to perform audio search and retrieval in a robust and efficient manner. Of course, the definition of "good" may vary depending on the specific task at hand. Nonetheless, we can still try to reason about the general properties or characteristics that would make a hash key useful or not useful.

Another way to rephrase the question is, "What makes a good mapping?" In other words, how can we map a time-domain representation of audio to a sequence of discrete symbols in such a way that it allows us to efficiently find matches in a database?

2.2 Wrong Answer: High Accuracy

In order to gain some intuition about this question, let's first point out what the answer is *not*. A good hash key is *not* the one with the highest accuracy. Here, accuracy refers to the percentage of hash keys that match when comparing the query with the true match in the database.

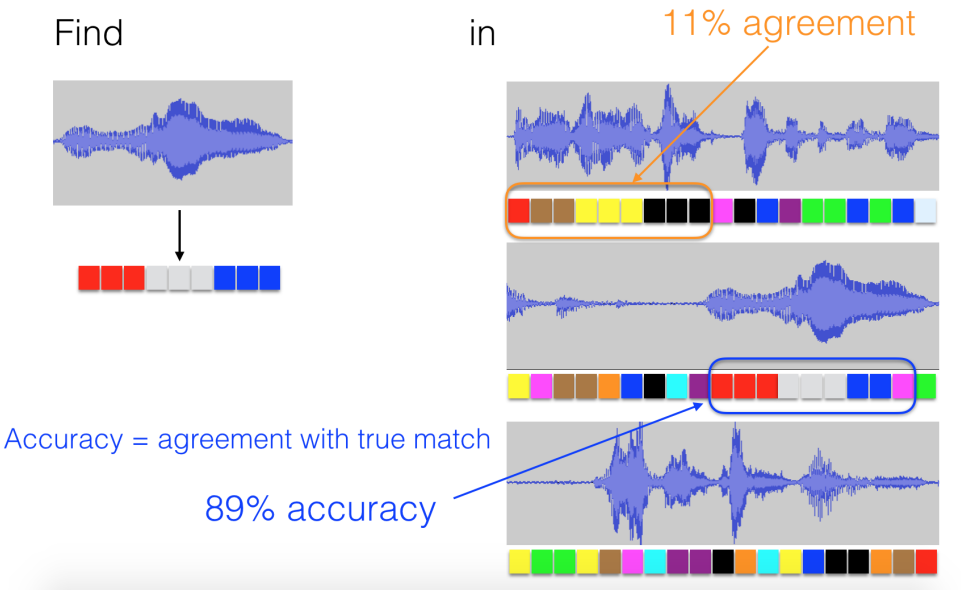


Figure 2.2: Accuracy is defined as the agreement between the query sequence and part of the database where the true match occurs. In the example shown above, the true match occurs at the end of the second database recording, and the accuracy is $\frac{8}{9} = 89\%$.

Consider the example shown in figure 2.2. The query has been mapped to a sequence of nine hash keys, shown at the left side of figure 2.2. The audio recordings in the database have also been mapped to sequences of hash keys. In this case, we can see that the true match occurs at the end of the second audio file in the database. When we compare the two corresponding sequences of hash keys, we see that 8 of the 9 boxes match in color. There is $\frac{8}{9} = 89\%$ agreement between the query and the true match and thus 89% accuracy. We might be tempted to think that a good hash key is the one that has highest accuracy.

But let's consider a second example, shown in figure 2.3. Here, the mapping is very simple: it simply maps everything to a red box. Of course, we would never use such a mapping, since we are simply throwing away all of the information in the signal. Yet it is useful to point out that in this case, there is 100% agreement between the query and the true match in the database, and thus 100% accuracy.

Clearly, there is something wrong with just focusing on accuracy. A good hash key is not simply the one with highest accuracy.

2.3 Wrong Answer: High Entropy

To gain more intuition, let's point out another example of what a good hash key is *not*. A good hash key is *not* the one with the highest entropy. Entropy is a measure of randomness and can be thought of as the number of bits required to encode the color of the box. If we

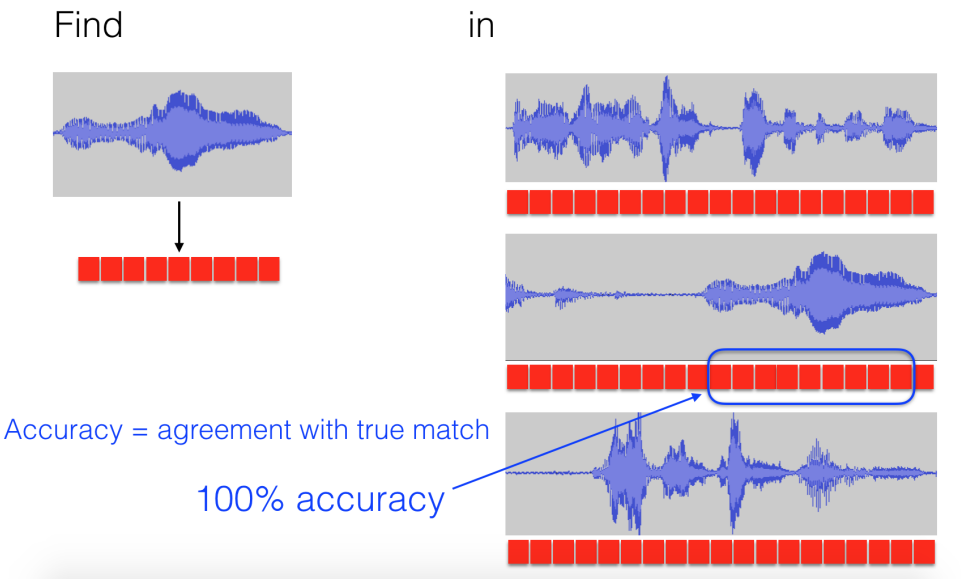


Figure 2.3: An example demonstrating that maximizing accuracy is not the correct goal to aim for. In this case, the accuracy is 100%, but clearly this mapping is not helpful in facilitating search and retrieval.

have 3 bits of entropy, we can encode 8 different colors. If we have 4 bits, we can encode 16 different colors. If we have 5 bits, we can encode 32 different colors. Figure 2.4 shows 3 examples of colors encoded by a 6 bit representation.

Consider the simplistic scenario where each bit is independent and correct 90% of the time. This means that if we encode the box color with 3 bits, the color of the box will be correct $.9^3 = 73\%$ of the time. If we encode the box color with 5 bits, the color of the box will be correct $.9^5 = 59\%$ of the time. If we encode the box color with 32 bits, the color of the box will be correct $.9^{32} = 3\%$ of the time. Clearly, using more bits does not necessarily result in a better hash key. A good hash key is not simply the one with the highest entropy.

Our intuition tells us that using too few bits is a bad thing. If we have too few colors, the hash keys will not be useful. But we can also see that using too many bits is also a bad thing. If we have too many colors, the hash keys will also not be useful. So what is it that makes a good hash key?

2.4 Useful Information Rate

Armed with more intuition, we now propose a metric to measure how good a hash key is. The metric we propose is *accuracy times entropy*. Let's consider the behavior of this quantity.

Consider first the relationship between entropy and the number of bits in the hash key. Assuming that the bits are all balanced and independent, these two two quantities will be

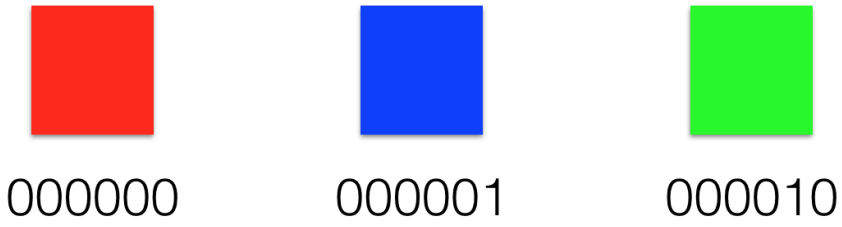


Figure 2.4: In our illustrations, the value of the hash key is represented by different colors. These are examples of 3 different colors being encoded by a 6 bit representation.

identical. The relationship between them will be a line with a slope of 1. Consider now the relationship between accuracy and the number of bits in the hash key. In our simplistic form, we saw in the previous section that the relationship is exponential: as the number of bits increases, the accuracy falls off exponentially. If we were to multiply these two functions, as shown in figure 2.5, we would get a hill-shaped function like the one shown in blue. The metric that we propose suggests that the best hash key is the one that maximizes the blue-colored hill. We call this hill-shaped function the useful information rate.

We can interpret the multiplication of these two functions in the following way. Each hash key communicates a certain number of bits of information. The problem is that sometimes this information is wrong – the hash key is incorrect. So our goal is not simply to maximize the total amount of information that the hash keys are communicating, but to maximize the amount of *correct* information that the hash keys are communicating. This is the quantity that we are most interested in, since it is the correct information that is useful to us. Accordingly, the term we use to describe this quantity is useful information rate.

Note that there are many ways that we could combine the accuracy and entropy functions. Useful information rate is one particularly simple way of combining them (i.e. multiplication) that has an intuitive interpretation from an information theory perspective. But does it correspond to anything meaningful in an actual system? We will investigate this in the next section.

2.5 Experimental Validation

In this section, we will present a simple experiment to verify and validate the intuition behind useful information rate. Note that this experiment is *not* intended to be a rigorous empirical proof – the rigorous empirical validation of these ideas will come in chapters 5 and 6, when we thoroughly investigate two different application scenarios. The simple experiment in this

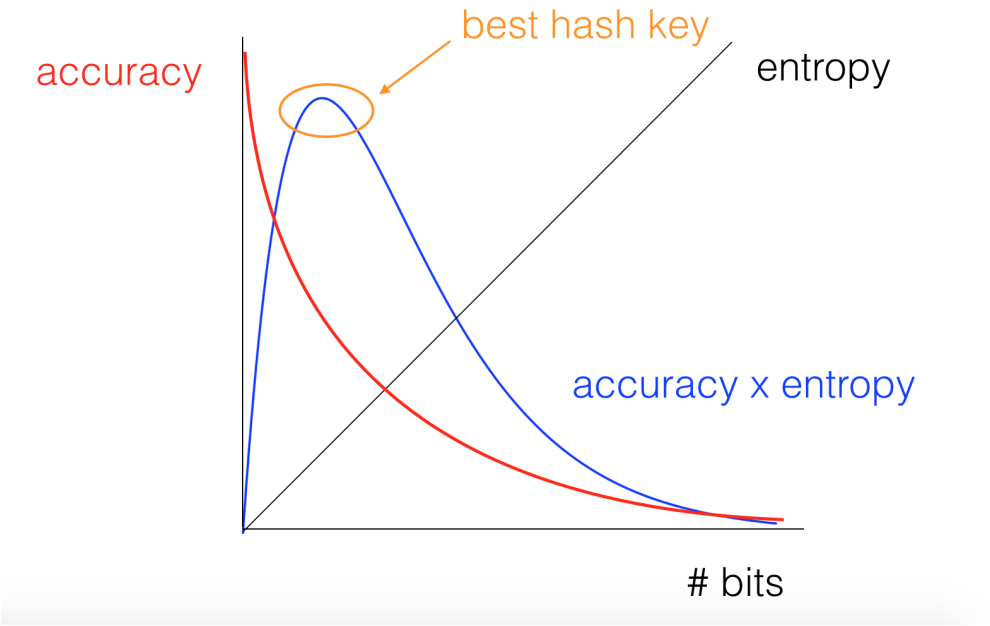


Figure 2.5: Useful information rate is the multiplication of accuracy and entropy. It represents the average amount of *correct* information that each hash key communicates. The blue curve shows the general shape of useful information rate as a function of the number of bits in the hash key.

section is intended more as a sanity check to validate our intuition.

Our intuition tells us that the useful information rate of a fingerprint design should correlate with its system-level performance. To test this hypothesis, we can measure the useful information rate and the system-level performance of a number of fingerprint designs, and then see if the two metrics correlate with one another. If the hypothesis is true, a better fingerprint design should have both a higher useful information rate and a higher system-level performance. We will describe our experimental validation in four parts: the fingerprint designs, how we measure the useful information rate, how we measure system-level performance, and the results we observe.

Fingerprint designs. We consider a family of fingerprint designs that are variants of the well-known Philips fingerprint. The Philips fingerprint is one of the most highly cited works in the literature, and is computed in the following manner:

- Compute a log mel spectrogram. Like the original paper [32], we consider 33 mel bands up to 2kHz and use 370 ms windows with 11.6 ms hop size.
- Compute fingerprint bits. The m^{th} fingerprint bit at time index n is given by

$$F(n,m) = sign(E(n,m) - E(n,m+1) + E(n+1,m+1) - E(n+1,m))$$

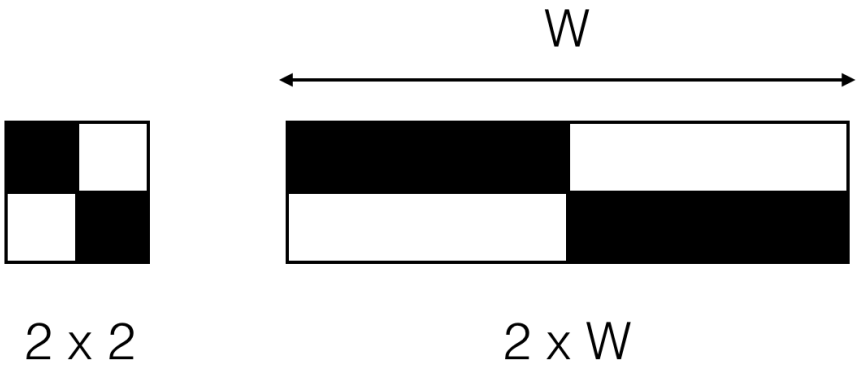


Figure 2.6: The 2×2 spectro-temporal filters used by the Philips fingerprint design are shown at left. Black corresponds to $+1$ value and white corresponds to a -1 value. For our experiments, we used $2 \times W$ spectro-temporal filters that are a generalization of the Philips design.

where $E(n, m)$ is the log mel spectrogram energy value at time index n for mel band m. We compute the $M = 32$ fingerprint bits using the above expression for $m = 1, \ldots, M$.

Note that the original Philips fingerprint design corresponds to applying a 2×2 checkerboard spectro-temporal filter at different frequency bands and thresholding the result at 0. The family of fingerprint designs is a simple variant of the Philips design in which we use a $2 \times W$ checkerboard spectro-temporal filter (see figure ?) instead of a 2×2 filter. For a given fixed filter width W, the fingerprints are computed in the following manner:

- Compute a log mel spectrogram. We consider 33 mel bands up to 2kHz and use 370 ms windows and 11.6 ms hop size.
- Compute fingerprint bits. The m^{th} fingerprint bit at time index n is given by:

$$F(n,m) = sign(\sum_{l=0}^{L-1} E(n-l,m) - E(n-l,m+1) + E(n+1+l,m+1) - E(n+1+l,m))$$

By defining the family of fingerprint designs in this way, the Philips fingerprint simply becomes a special case with filter width $L = 2$. We can then compare the performance of this fingerprint design as a function of the filter width L.

Measuring useful information rate. In order to measure useful information rate, we use the procedure shown in figure 2.7. We first take a set of clean audio recordings and "noisify" them by adding white gaussian noise at a specified signal-to-noise ratio (SNR).

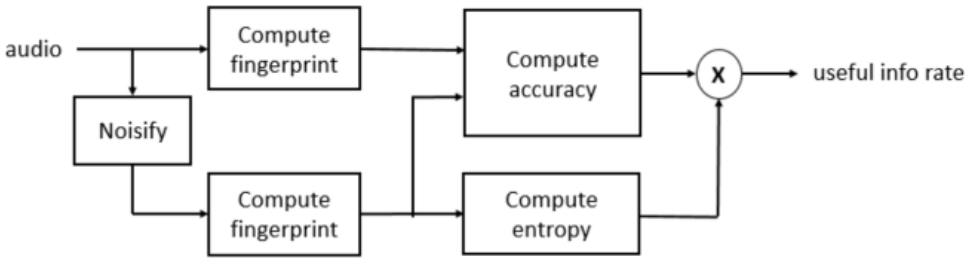


Figure 2.7: Block diagram of the process for computing useful information rate.

We then compute fingerprints on the clean and noisy recordings and measure the empirical useful information rate between the corresponding sequences of fingerprints. Here, accuracy simply referes to the fraction of corresponding fingerprints that match. We compute the useful information rate on 10000 randomly sampled frames (and their surrounding context) from the TRECVID content based copy detection data set [69], which contains internet archive videos.

Measuring system-level performance. To measure system-level performance, we created a database of fingerprints for 977 audio files from the TRECVID data, which amounts to 40 hours of audio data. We generated 500 noisy queries by randomly selecting 10 second segments and adding white gaussian noise at a desired SNR. To compute a score for each item in the database, we use the search method described by Wang [99], which accumulates a histogram of offsets from matching fingerprints. We then measured the accuracy as the percentage of queries that were identified correctly (i.e. the true match had the highest score). This is a common metric used to measure system-level performance on a fixed database search (e.g. [3][4][99][32]).

Results. Figures 2.8 and 2.9 show the useful information rate and system-level performance for these two experiments, respectively. Each group of bars shows the useful information rate or accuracy of the fingerprint model with filter width W at a given SNR. There are two things to notice when we compare figures 2.8 and 2.9.

First, we notice that useful information rate and system-level accuracy correlate directly when assessing the *relative* performance of various fingerprint designs. As filter width increases, both useful information rate and system-level accuracy increase. This suggests that a higher filter width corresponds to a more robust fingerprint design, and this increased robustness is reflected in both metrics. One benefit of the useful information rate metric is that we can compare fingerprint designs more efficiently than with a system-level performance metric. Note that figure 2.9 required more than 1000 times more data and significantly more setup and implementation than figure 2.8. For example, figure 2.8 did not require creating a database or implementing a search mechanism.

Second, we point out that the useful information rate metric is not indicative of *absolute* performance. For example, notice that a fingerprint filter width of 32 at 10dB SNR has

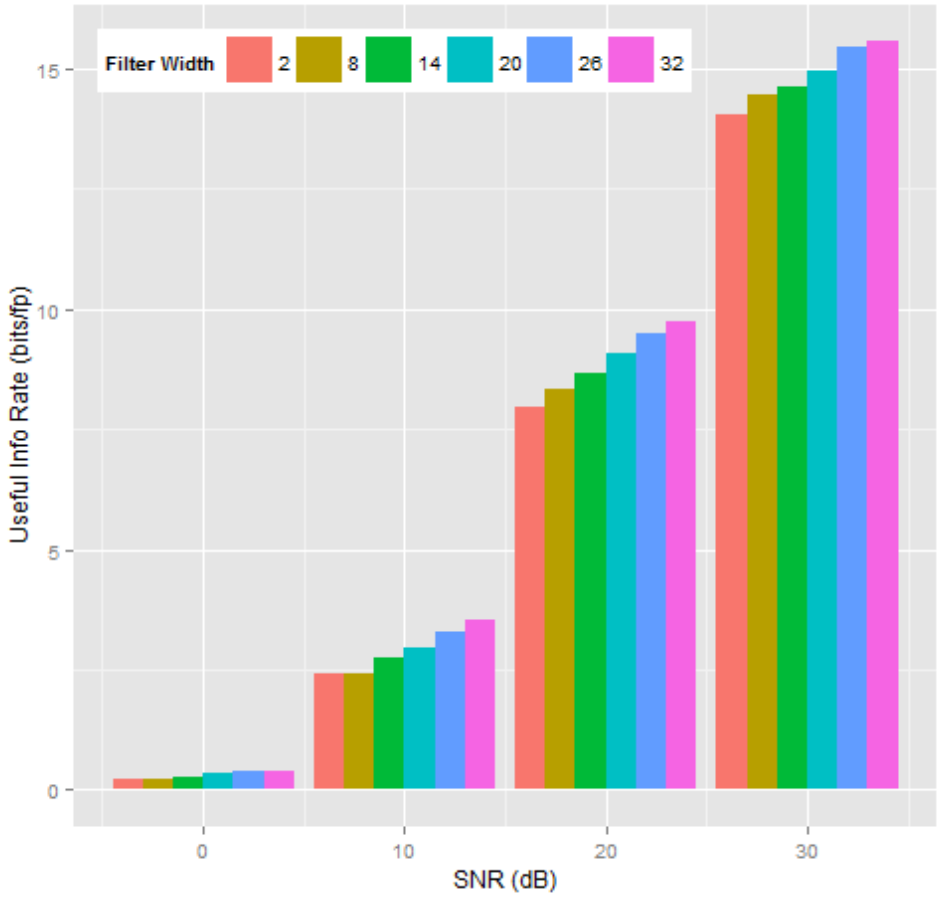


Figure 2.8: Useful information rate of various fingerprint designs. Each group of bars shows how useful information rate varies as a function of the filter width W at a fixed SNR.

higher system accuracy than a fingerprint filter width of 2 at 20dB SNR. In contrast, a fingerprint filter width of 32 at 10dB SNR has a much *lower* useful information rate than a fingerprint filter width of 2 at 20dB SNR. This is to be expected – the absolute performance numbers depend upon many factors besides the fingerprint design, such as the database size, how similar or different database items are to one another, and how much redundancy is built into the search mechanism. These experiments indicate that useful information rate is useful in assessing the *relative* performance of different fingerprint designs, but of course cannot predict the absolute system performance.

Figures 2.8 and 2.9 are consistent with our intuition that useful information rate and system-level accuracy correlate directly in evaluating fingerprint designs. A more robust fingerprint design will have both higher useful information rate and higher system-level performance. The useful information rate metric is preferable because it decouples the effect of other system factors and focuses only on the robustness of the fingerprint design. As a

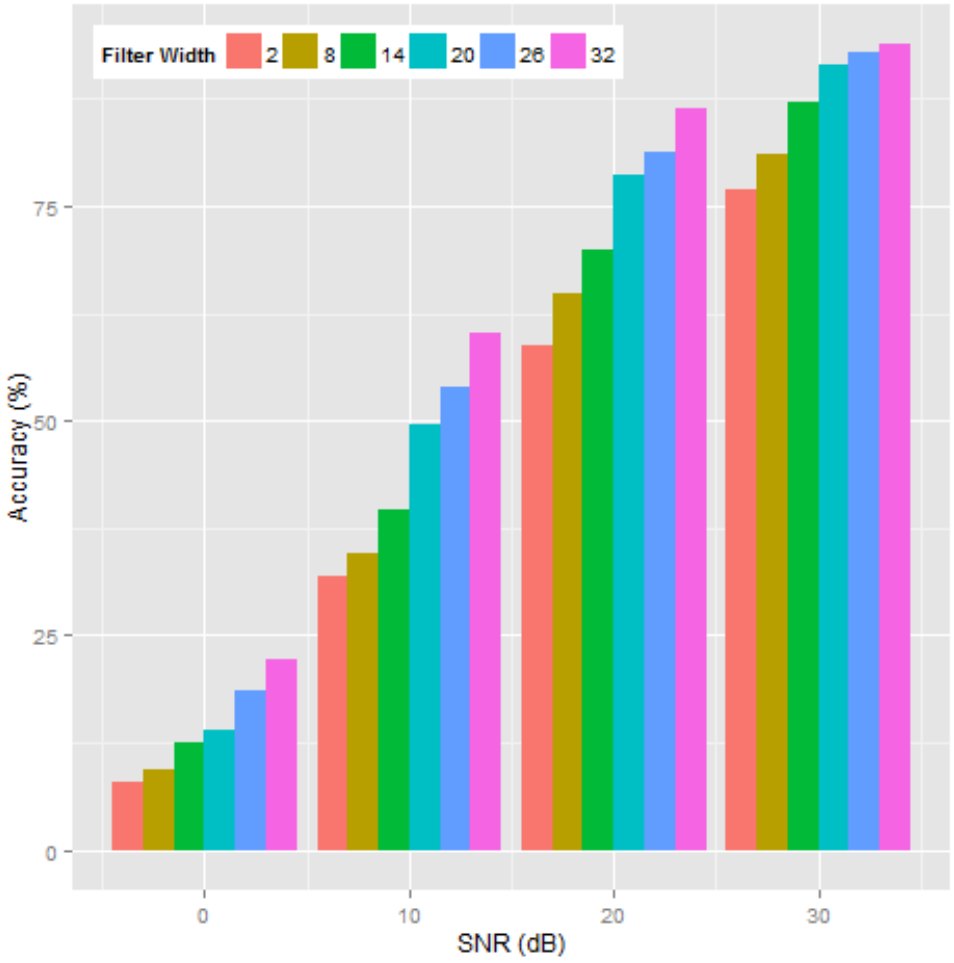


Figure 2.9: Accuracy of various fingerprint designs on an exact-match fingerprinting task. Each group of bars shows how accuracy varies as a function of the filter width W at a fixed SNR.

result, it requires much less data and very little setup to compute.

2.6 Recap

We have discussed the first theoretical question of interest: What makes a good hash key? After building some intuition by pointing out several wrong answers to the question, we propose an intuitive, information-theoretic metric called useful information rate that measures how good a hash key is. We then presented some simple experiments using empirical data to validate our intuition. The insights in this chapter will provide a foundation for the design of audio hashprints in chapter 4.

Chapter 3

Theory: What Makes A Good Hash Bit?

In the previous chapter, we asked the question, "What makes a good hash key?" In this chapter, we will ask a second fundamental question: "What makes a good hash *bit*?" We will first introduce the problem statement (section 3.1), and then discuss the three key characteristics that define a good hash bit (sections 3.2, 3.3, 3.4).

3.1 Problem Statement

A hash key is simply a collection of hash bits. In the previous chapter, we discussed what it is that makes a good hash key. In this chapter, we will discuss what it is that makes a good hash *bit*. Whereas in the previous chapter we focused on having the right colored box, in this chapter we will investigate the individual bits that *determine* the color of the box (see figure 3.1).

We can make an analogy between a hash key and a team of people. We have already asked the question, "What makes a good team?" In chapter 2, we proposed a way to measure how good a team is. That metric is called useful information rate. It consists of two components: high entropy and high accuracy. But now we ask the question, "What makes a good individual?"

A good individual is one who contributes to the team's mission. A good hash bit is one that helps the hash key have high entropy and high accuracy. Each of these two characteristics has certain consequences on what it means for a hash bit to be good. These consequences will be discussed in the next 3 sections.

3.2 Balanced

One consequence of *high entropy* is that each bit should be balanced. In other words, it should take on the value 0 half the time and 1 half the time. When this is true, the bit is

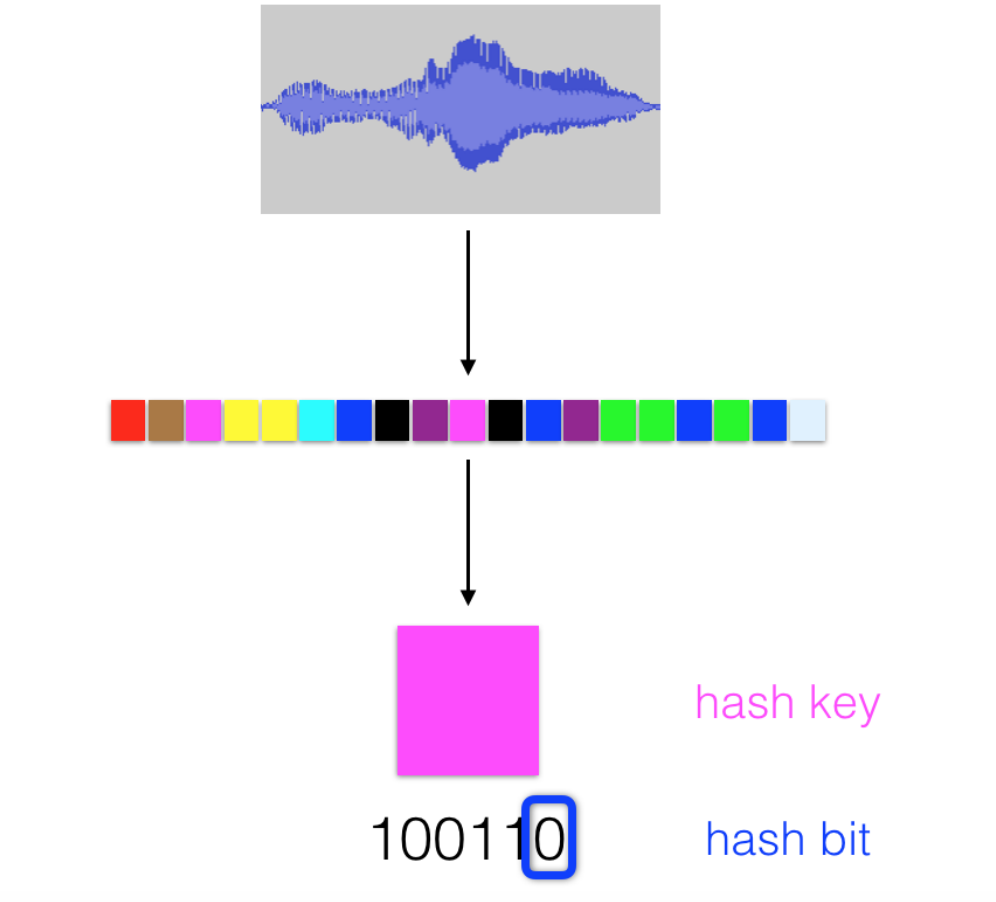


Figure 3.1: Hash key vs hash bit. A hash key is a collection of hash bits. In our graphical depiction of fingerprints, the color of the box represents the value of the hash key. The color is encoded by the values of the individual hash bits.

equivalent to a fair coin toss – it has maximum randomness and maximum entropy of 1 bit. Note that any imbalance in a bit will result in a decrease in entropy. In the extreme case where a bit always takes on the same value, the entropy is 0.

One way to map a random variable to a binary 0-1 value is to apply a hard threshold. If the random variable takes on a value that is less than the threshold, we assign a 0 bit value. If the random variable takes on a value that is greater than the threshold, we assign a 1 bit value. The question is, "Where do we set the threshold?" High entropy tells us that the threshold should be set to the median of the random variable's probability distribution. This ensures that the bit will have maximum entropy and thus communicate the most information. If the threshold is located in the tail of the distribution, the bit will have low entropy and communicate very little information (see figure 3.2).

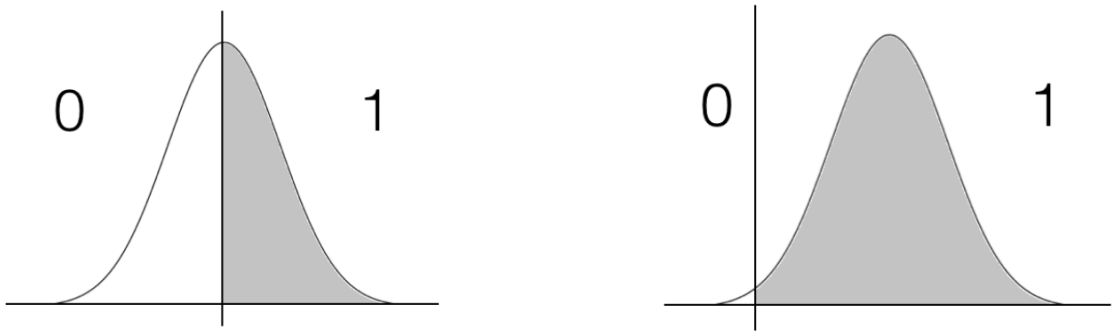


Figure 3.2: In order for the bits to be balanced, the threshold should be set to the median of the underlying probability distribution (left side). If the threshold is located in the tail of the distribution, the bit will have very little entropy (right side).

3.3 Uncorrelated

Another consequence of *high entropy* is that each bit should be uncorrelated with the other bits. Any correlations between hash bits represents inefficiency. For example, a single hash bit that is simply replicated N times will result in a hash key which still only contains a maximum of 1 bit of entropy. On the other hand, a set of N independent bits will have the maximum N bits of entropy. It is not enough that each bit in isolation contribute 1 bit of information, but that it contribute 1 bit of information that is *new*.

3.4 High Variance

Now let's consider the consequence of *high accuracy*. In the context of applying a threshold to a random variable, high accuracy corresponds to maximizing the variance of the random variable's probability distribution. To see this, consider the case when the random variable takes on a value that is very close to the threshold, which will be located at the median of the distribution (as discussed above). In the context of the hashprint framework, this random variable will be a linear combination of many spectrogram energy values, so its distribution will be roughly bell-shaped as a result of the central limit theorem. Because the random variable takes on a value close to the threshold, a small perturbation from noise may cause the feature to fall on the wrong side of the threshold, resulting in an incorrect bit. This situation can be minimized by maximizing the variance of the feature distribution. Figure 3.3 shows a comparison between two distributions with different variances. As we can see, the area of the red 'danger zone' around the threshold decreases as the variance of the distribution increases. Thus, when noise can be approximated as a symmetric perturbation with zero mean, robustness corresponds to high variance in the underlying feature distribution. While

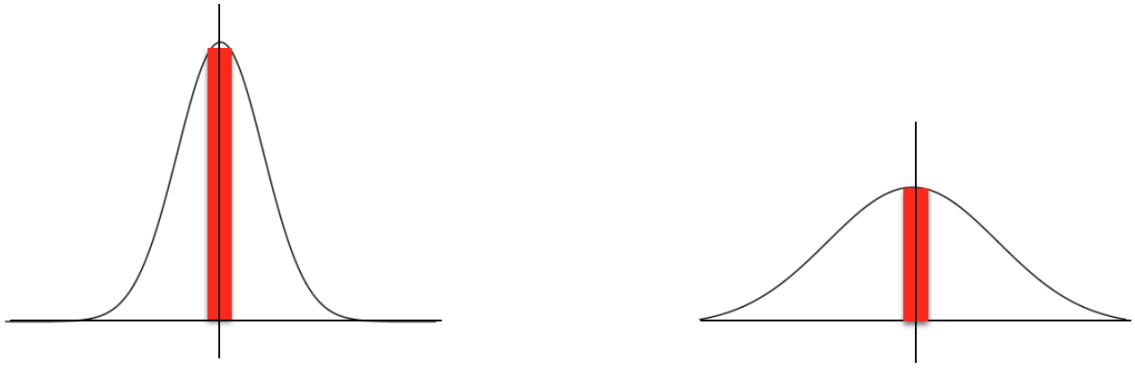


Figure 3.3: When thresholding a random variable, robustness corresponds to higher variance in the underlying probability distribution. When the random variable takes on a value very close to the median threshold, a small perturbation from noise may cause the variable to fall on the wrong side of the threshold, resulting in an incorrect bit.

this assumption about the noise is rarely true in a strict sense, it is nonetheless a reasonable approximation for many different types of noise.

3.5 Recap

In this chapter we asked the question, "What makes a good hash bit?" Since a hash key is simply a collection of hash bits, a good hash bit is one that helps the hash key to have high entropy and high accuracy. The two consequences of high entropy are that each bit should be balanced and uncorrelated with the other bits. The consequence of high accuracy is that each bit should be the result of thresholding a probability distribution that has high variance. So, what makes a good hash bit? Balanced, uncorrelated, and high variance.

Chapter 4

Audio Hashprints

Having laid down our foundation of theory, we are now ready to introduce audio hashprints. Audio hashprints are a representation of audio that facilitates audio search and retrieval. As our building block metaphor suggests, this representation is built upon the key insights established in the previous two chapters. This representation will in turn be a fundamental building block in tackling the two different application scenarios of interest. In this chapter, we introduce and explain what audio hashprints are. We will do this in five parts: the motivation behind developing such a representation, the mechanics of computing hashprints, the formulation of the filter learning problem, the rationale of the hashprint design, and its relation to previous work.

4.1 Motivation

We will explain the motivation behind the proposed fingerprint design in three parts: the design choice of using a binary representation, general principles of good fingerprint design, and other desirable application-specific characteristics.

Binary Representation

Hashprints are, first of all, a binary representation of audio. Using a binary feature representation has three important benefits.

1. **Compactness in memory.** Because hashprints are simply a collection of bits, we can represent each hashprint as a single 64-bit integer containing up to 64 bits of information. Since we may have to store a large amount of audio data in a searchable database, compactness in memory is an important consideration.

2. **Indexing.** Because hashprints are a discrete representation (rather than a continuous representation), we can directly use hashprints in a table lookup or reverse index.

These techniques are especially useful in retrieving matches from a large database in an efficient manner.

3. **Efficient distance computations.** We can compute the Hamming distance between two hashprints very efficiently by peforming a single logical xor operation between two 64-bit integers, and then counting the number of 1 bits in the result. This requires fewer operations compared to computing (say) the Euclidean distance between two vectors of floating point numbers. *When* some type of exhaustive search is necessary, these computational savings can be important in reducing the latency of the search.

For these reasons, we will focus on the problem of designing a binary representation that facilitates audio search and retrieval. We now turn our attention to general principles for how to design such a binary fingerprint representation.

Principles of Fingerprint Design

The formulation of audio hashprints grows out of two key principles of good fingerprint design.

1. **Design principle 1: Compactness.** A good fingerprint should represent a maximum amount of information in as little space as possible. There are two direct consequences of this principle in the context of our representation: each bit should be balanced, and each bit should be uncorrelated with all the other bits. Note that any imbalance in a bit or any correlation between bits will result in an inefficient representation. These two characteristics were discussed in sections 3.2 and 3.3.

2. **Design principle 2: Robustness.** A good fingerprint should be robust to noise. In the context of our hashprint design where each bit results from thresholding a feature, achieving robustness corresponds to maximizing the variance of the feature distribution. This was discussed in section 3.4.

The above two principles of fingerprint design are very general – we will always want the fingerprint to be compact and robust. In addition to these general characteristics, though, there are several other application-specific characteristics that we would like the representation to have. These are discussed in the next subsection.

Other Desirable Characteristics

There are three other characteristics that we would like hashprints to have. These characteristics are not general principles of good design, as in the previous subsection, but are instead useful in certain application-specific scenarios such as those explored in chapters 5 and 6.

1. **Highly adaptive.** The first desirable characteristic is that the representation be highly adaptive. The 'adaptive' part means that the representation is not a fixed representation like MFCCs, chroma features, or a hardcoded fingerprint extraction module. Rather, the representation should be tailored to a set of data in order to adapt to its unique characteristics. The 'highly' part means that we would ideally be able to learn a suitable representation given a small quantity of data, rather than requiring an extremely large amount of data. In the context of the scenarios explored in chapters 5 and 6, for example, we would like to learn a representation based on a few audio recordings of a meeting, or a collection of studio tracks from a musical artist.

2. **Unsupervised.** The second desirable characteristic is that the representation can be learned in an unsupervised manner. Often, collecting suitable training data, annotating ground truth, and setting up a supervised learning task requires a significant amount of time and effort. Other times, it is not possible to get suitable training data, or the available training data may not be representative of the actual test data. The two scenarios explored in chapter 5 and 6 are examples of these situations. In these scenarios, we would like to be able to learn the representation without any ground truth labels, so that the representation can be learned automatically on-the-fly without any manual effort.

3. **Volume-invariant.** The third desirable characteristic is that the representation should be volume-invariant. This means that an audio signal will yield the same representation even if it is multiplied by a constant factor. This is an important characteristic in many search and retrieval scenarios. Consider the aligning meeting recordings scenario – when a person speaks, the same signal will be picked up by multiple recording nodes but with varying attenuation levels (along with distortions, of course) depending on the distance to the speaker. Likewise, it is an important characteristic in the live song identification scenario – the noisy cell phone recording may not be at the same volume level as the original studio track. In audio search and retrieval applications, volume-invariance is usually a desired characteristic.

Identifying these three additional desired characteristics will help inform the design of the hashprint represenation. Having discussed the characteristics and properties that we would like to have in an ideal representation, we will now introduce and explain how hashprints are computed.

4.2 Mechanics

Figure 4.1 shows how to compute audio hashprints from a time-domain representation of an audio signal. The computation consists of 6 steps, which are described below.

1. **Compute spectrogram.** The first step is to compute a time-frequency representation of audio. This time-frequency representation can be selected to suit the characteristics

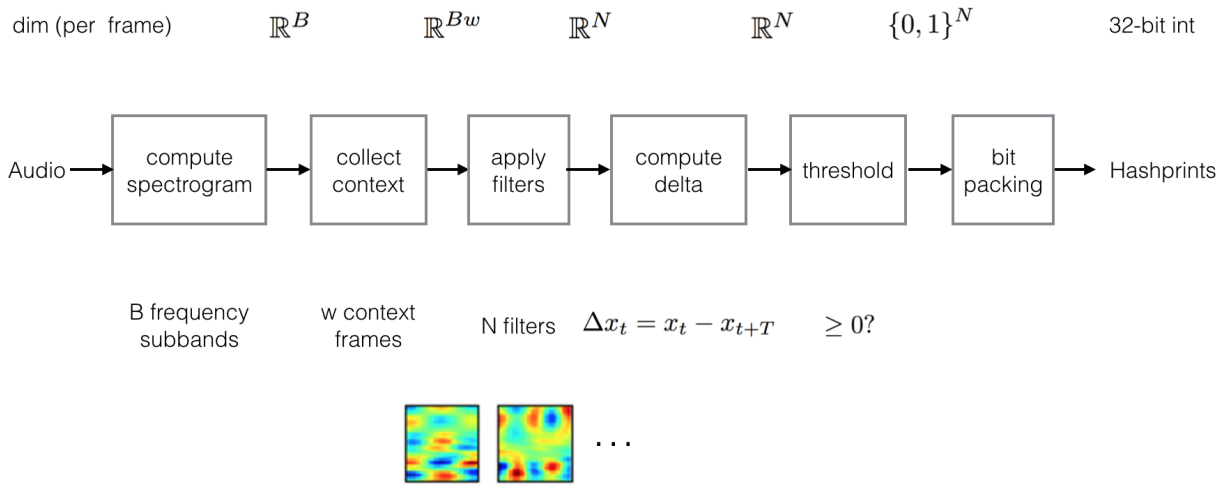


Figure 4.1: Block diagram of the fingerprint computation.

of the problem at hand. In the meeting recordings scenario that we will investigate in chapter 5, for example, we used a log mel spectrogram, since the underlying source signals are primarily speech. In the live song identification scenario that we will investigate in chapter 6, we instead use a constant Q transform that is designed to match the pitches of the western musical scale, since the underlying source signals are primarily music. At the end of this first step, the audio is represented at each frame by a vector of dimension B, where B is the number of frequency subbands.

2. **Collect context frames.** In addition to looking at the audio frame of interest, we also look at the neighboring frames to its left and right. When computing the fingerprint at a particular frame, we consider w frames of context. At the end of this second step, we represent each frame with a vector of dimension Bw.

3. **Apply spectro-temporal filters.** At each frame we apply N different spectrotemporal filters in order to compute N different specto-temporal features. Each spectro-temporal feature is a linear combination of the spectrogram energy values for the current frame and surrounding context frames. The weights of this linear combination are specified by the coefficients in the spectro-temporal filters. A graphical illustration of this process is shown in figure 4.2. These filters are learned in an unsupervised manner by solving a sequence of optimization problems, which we will discuss in detail in the next section. At the end of this third step, we have N spectro-temporal features per frame.

4. **Compute deltas.** For each of our N features, we compute the change in the feature

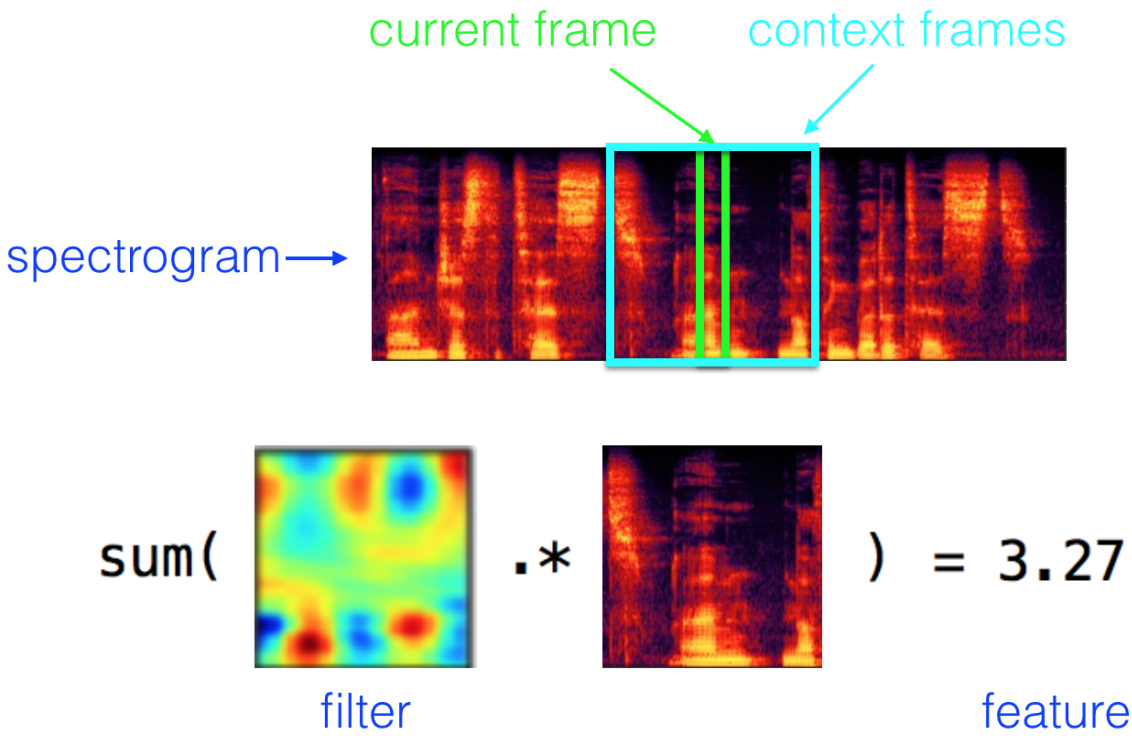


Figure 4.2: A graphical illustration of applying a spectro-temporal filter. To compute a spectro-temporal feature, we consider a linear combination of the spectrogram energy values for the current audio frame and several context frames. The weights of this linear combination are specified by the spectro-temporal filter. This process is shown in matlab notation.

value over a time lag T. If the feature value at frame n is given by x_n, then the corresponding delta feature will be $\Delta_n = x_n - x_{n+T}$. At the end of this fourth step, we have N spectro-temporal delta features per frame.

5. **Apply threshold.** Each of the N delta features is compared to a threshold value of 0, which results in a binary value. Each of these binary values thus represents whether the delta feature is increasing or decreasing over time (across the time lag T). At the end of the fifth step, we have N binary values per frame.

6. **Bit packing.** The N binary values are packed into a single 32-bit or 64-bit integer which represents the hashprint value for a single frame. This compact binary representation will allow us to store fingerprints in memory efficiently, do reverse indexing, or compute Hamming distance between hashprints very quickly.

Figure 4.1 shows a summary of the hashprint computation process. The text above the

boxes shows the dimension (per frame) at each stage of the process. The text below the boxes provides additional clarifying information. The only part of this figure that still needs explanation is how we learn the spectro-temporal filters that are applied in step 3. This will be addressed in the next section.

4.3 Filter Learning Problem

The filters are selected to maximize the variance of the resulting spectro-temporal features, while ensuring that these features are uncorrelated. The filter learning problem can be formulated as a series of optimization problems, which are described below. Consider a vector $\in \mathbb{R}^{Bw}$ containing the spectrogram energy values for an audio frame and its neighboring context frames, where w specifies the number of context frames and B specifies the number of frequency bands per frame. We can stack a bunch of these vectors into a large matrix $A \in \mathbb{R}^{M \times Bw}$, where M corresponds (approximately) to the total number of audio frames in a set of audio recordings. Let $S \in \mathbb{R}^{Bw \times Bw}$ be the covariance matrix of A, and let $x_i \in \mathbb{R}^{Bw}$ specify the coefficients of the i^{th} spectro-temporal filter. Finally, let N denote the number of bits in the fingerprint. Then, for $i = 1, \ldots, N$, we solve the following:

$$\begin{aligned} \text{maximize} \quad & x_i^T S x_i \\ \text{subject to} \quad & \|x_i\|_2^2 = 1 \\ & x_i^T x_j = 0, \ j = 1, \ldots, i-1. \end{aligned} \tag{4.1}$$

Each resulting x_i specifies the spectro-temporal filter weights corresponding to the i^{th} fingerprint bit.

Let's unpack the above formulation. The first line can be summarized as "maximize the variance." To see this, note that the variance of the features Ax_i can be expressed as $\frac{1}{M}\|\tilde{A}x_i\|_2^2$, where the columns of $\tilde{A}$ are zero mean. This objective is motivated by our design principle of robustness. The first constraint simply says, "finite energy." We could use a number of different ways to constrain the energy, and we choose the L2 norm for reasons that we will see shortly. The last constraint says, "uncorrelated filters." This constraint ensures that the filters are mutually orthogonal. This constraint is motivated by our design principle of compactness (uncorrelated bits).

The optimization problem 4.1 is exactly the eigenvalue/eigenvector problem, where x_i, $i = 1, ..., N$ are the N eigenvectors of S with highest eigenvalue. The benefit of this formulation is that it can be solved very efficiently using standard implementations. The eigenvectors, once "reassembled" as eigenfilters spanning w context frames, are the spectro-temporal filters that are applied in Step 3 of the fingerprint computation. These spectrotemporal filters yield the spectrotemporal features with maximum variance, ensuring that the fingerprint bits will be robust. The eigenvectors will also be orthogonal to one another, ensuring that the fingerprint bits will be uncorrelated. One big advantage of this formulation is that it can be done in an unsupervised fashion. We can thus design a robust fingerprint without labeled data.

To summarize the filter learning problem, we first construct the data matrix A containing spectrogram energy values from a set of audio data. We then compute the covariance matrix of A, and then compute the eigenvectors of the covariance matrix. These eigenvectors specify the spectro-temporal filters that we apply in the hashprint computation.

We will now turn our attention to an explanation and justification of *why* we use the computation process in Figure 4.1.

4.4 Rationale of Design

In this section, we will revisit each of the six steps of the hashprint computation. This time, however, we will focus on the rationale of each step, rather than the mechanics of the computation.

1. **Compute spectrogram.** The first step is to compute a suitable time-frequency representation of the audio signal. As mentioned previously, this time-frequency representation can be chosen to suit the task at hand. The rationale for this first step is to simply convert the time-domain signal into a representation where we can access the relevant spectral information.

2. **Collect context frames.** Instead of only considering the spectrogram energy values for the current audio frame of interest, we also consider the neighboring context frames to its left and right. The purpose of this is to represent the audio in a higher dimensional space, which allows for greater discrimination between data points than can be achieved when only looking at a single audio frame. The purpose of this step is thus to make our representation more discriminative.

3. **Apply spectro-temporal filters.** In the third step, we apply a set of spectro-temporal filters at each frame in order to generate N spectro-temporal features. Note that many standard feature representations are special cases of spectro-temporal features. For example, MFCCs can be interpreted as spectro-temporal features that are generated by applying a specific set of spectro-temporal filters to a log mel spectrogram. In this case, the spectro-temporal filters only span a single audio frame, and the filter coefficients are specified by the coefficients of the discrete cosine transform. Likewise, chroma features can be interpreted as spectro-temporal features that are generated by applying a specific set of spectro-temporal filters to a constant Q transform. Here, the spectro-temporal filters again only span a single audio frame, and the filter coefficients are selected to aggregate energy across musical octaves.

   Rather than using a fixed set of spectro-temporal filters, as in the case with MFCC or chroma features, we instead use filters that are learned and adapted to the data. As we saw in the previous section, these filters are selected to capture directions of maximum variance across the data. Also, rather than only focusing on a single audio frame, the learned spectro-temporal filters span w audio frames.

The rationale of this step is that it is generally better to use a feature representation that is adapted to the data, rather than a fixed, hardcoded representation. At the same time, we would like to avoid the hassle of annotating training data, so that the representation can be learned on-the-fly in an unsupervised manner. These considerations motivate the proposed filter learning problem discussed in the previous section. On a practical level, then, this feature representation should not require any more effort on the part of a researcher than using a standard feature representation like MFCC or chroma, since the learning is done on-the-fly.

4. **Compute deltas.** We cannot simply threshold the spectro-temporal features themselves, because the resulting fingerprint would not satisfy one other important characteristic: invariance to volume level. In order to work effectively in our use-case scenarios, the fingerprints must be invariant to acoustic energy level. So, the same audio signal attenuated or amplified should yield the same fingerprint values.

   Computing delta features is one way to achieve volume-invariance. We threshold the delta features at 0, so each bit encodes whether the spectro-temporal features are increasing or decreasing in time (across a time lag T). Note that this information is invariant to volume level when we use a log-energy representation such as a log mel spectrogram.[1] In contrast, applying a threshold to the features themselves (rather than the delta features) would not be invariant to volume level.

   There are, of course, other ways to achieve volume-invariance. Perhaps the most obvious way is to simply normalize features at every frame. Computing delta features is a more effective way to achieve volume-invariance than (say) L2 normalization for two reasons. First, it is computationally cheaper. Computing delta features simply requires one additional subtraction per feature per frame, whereas normalizing the spectral values in a set of context windows at each frame is much more expensive. Second, the delta features are far more robust. Each delta feature can thought of as the sum of two different variables, which effectively doubles the variance of the feature and thus increases its robustness. We have verified this empirically in both of the two scenarios explored in chapters 5 and 6.

   There is a tradeoff in the selection of the time lag T. For very small T, the delta features will have lower variance, since we are taking the difference between features that are immediately adjacent in time (and thus highly correlated). A larger T will thus yield a more robust fingerprint up until the point where the signal becomes decorrelated with itself. On the other hand, a very large T results in a fingerprint that is not very localized in time. So, the ideal T is the minimum time lag that ensures that the underlying audio signal has become decorrelated with itself. The selection of T could thus be determined empirically on-the-fly by measuring autocorrelation, or it could be set to a fixed value based on a priori assumptions. For example, in the aligning meeting

[1] e.g. $log(x[n]) - log(x[n+T]) = log(5x[n]) - log(5x[n+T])$

Step	Rationale
Computing spectrogram	access spectral info
Collect context frames	discriminative
Apply spectro-temporal filters	robust
Compute delta	volume-invariant
Apply threshold	balanced
Bit packing	compact

Table 4.1: The rationale behind each step of the hashprint computation process.

recordings scenario in chapter 5, we select a value of T based on typical speech rates in American English.

5. **Apply threshold.** The threshold is applied to the delta spectro-temporal features. Unless the audio is continuously increasing or decreasing in volume, these delta features will have a roughly symmetric distribution centered around 0. Since we would like to ensure that each bit is balanced, we set all of the thresholds to the median value of 0. This also has the added advantage that the threshold value does not need to be learned.

6. **Bit packing.** The last stage packages the N binary values into a 32-bit or 64-bit integer, depending on the computer architecture of the server or machine running the application. This representation has the benefit of being compact in memory, and it can also be manipulated easily as a built-in representation. For example, we can directly use the integer representation as a key in a hash table, or compute the Hamming distance between two hashprints using logical bitwise operators.

Before moving on, it is useful to summarize what each of the six steps is doing. The first step transforms the audio signal into a time-frequency representation that allows us to access relevant spectral information. The second step incorporates context frames in order to make the representation more discriminative. The third step uses the spectrogram energy information and generates a set of uncorrelated random variables whose distributions have maximum variance (which correlates with robustness). The fourth step computes delta features in order to ensure a volume-invariant representation. The fifth step applies a threshold at the median of the distribution in order to generate balanced bits. The sixth step then packages the binary values in memory in a compact way. Table 4.1 shows a summary of these six steps, along with a brief description of what each step accomplishes.

4.5 Relation to Previous Work

The design of audio hashprints uses ideas and techniques that are drawn from a rich literature in audio fingerprinting, cover song detection, machine learning, and other related fields. In this section, we will explain how various aspects of the design draw inspiration from and fit into the context of previous work. We will approach this from five different angles.

Continuous vs Discrete

One key design decision is using a binary (discrete) representation of audio. Binary representations of audio have been explored extensively in the audio fingerprinting literature, as we saw in table 1.1. They are an excellent design decision for fingerprinting applications because they are compact in memory and thus allow for efficient storage of large databases, and they are also suitable for use with indexing techniques.

Threshold-Based vs Value-Based

In section 1.3 we discussed the difference between threshold-based and value-based methods. Recall that a threshold-based method is one in which each bit of the representation is derived by computing some feature of interest and then applying a hard threshold. A value-based method is one in which some features of interest are computed, and the values of the features themselves are directly encoded in the binary representation. One very commonly adopted value-based approach is to encode the location of maxima such as spectral peaks. Given the above schema, it is useful to point out that hashprints are an example of a threshold-based approach where the features of interest are delta spectro-temporal features.

Design Method

Another way that we clustered fingerprint representations in section 1.3 was by their design method: manual design, supervised learning, or unsupervised learning. As we saw from table 1.1, most representations are manually designed, a few incorporate supervised learning, and a few combine manually designed features with unsupervised techniques.

The main contribution of audio hashprints to the fingerprinting literature is that they provide a highly adaptive method for learning a binary fingerprint representation in an entirely *unsupervised* manner. This characteristic can be very desirable in situations where labeled training data may not be available or is difficult to obtain. In chapters 5 and 6, we will investigate two such scenarios. In a live song identification scenario (chapter 6), for example, learning the representation in an unsupervised manner means that the binary representation can be tailored to each artist's music without having to collect and annotate training data for each artist.

There are other works (not in the audio fingerprinting literature) that explore learning binary representations in an unsupervised manner using deep neural networks. Because this is an important class of approaches, these will be discussed in a separate subsection below.

Shingling & Hashing

Another key design decision is to have each hashprint describe a relatively long temporal context containing multiple audio frames. Using multiple context frames in the manner described above is often referred to as shingling [11] or time delay embedding [86]. By considering a much higher dimensional space, this technique allows for greater discrimination on a single feature vector than could be achieved with only a single audio frame. Not surprisingly, it has proven to be very useful in handling cover song variations [11][86][85][50].

The hashprint uses hashing techniques to convert each audio shingle into a set of binary values. Given an audio shingle represented as a point in some high-dimensional space, it projects the point onto a direction of maximum variance, and then thresholds the projection to generate a bit.[2] In the hashing literature, this technique is called spectral hashing [100]. It can be thought of as a variant of locality sensitive hashing [15], where the projections are done in a data-dependent way instead of projecting onto random directions.

Hashprints can thus be thought of as applying spectral hashing to a shingle representation, along with a slight modification (computing delta features) to ensure volume-invariance.

Relation to DNNs

Given the recent surge of interest in deep neural networks (DNNs), it is instructive to consider how hashprints relate to such work. In the machine learning community, many recent works have explored binary encodings learned through DNN architectures [65][78][52]. In the music information retrieval community, Raffel and Ellis [73] propose such an approach for matching MIDI and audio files. Compared to these approaches, hashprints offer two potential advantages in a scenario like (say) live song identification. One advantage is that it learns the binary representation in an unsupervised manner. This is particularly helpful for a scenario like this, where collecting training data and annotating ground truth would be very time-consuming and laborious. The second advantage is that it requires relatively little data to learn a reasonable representation. This allows hashprints to "divide up" the search space in a way that is specific to the scenario at hand, even if only a small amount of data is available. In cases like these, a deep autoencoder [37][16] may not have sufficient data to converge to a good representation. So, our method straddles two extremes: it is adaptive to the data (unlike hardcoded, fixed representations), but it works well with small amounts of data (unlike representations based on DNNs).

Hashprints are similar to DNNs in that both are distributed representations. In fact, hashprints can be considered a single layer neural network, where each output node corre-

[2]More precisely, the hashprint thresholds the *delta* of the projection. The main text is left as is to make the connection to spectral hashing more clear.

sponds to a spectro-temporal filter, the weights for each output node correspond to the filter coefficients, and the output nonlinearity is a hard threshold.[3] So, in a sense, hashprints can be interpreted as a neural network that is tuned to work well with small amounts of data and forced to obey certain constraints (e.g. uncorrelated and volume-invariant).

4.6 Recap

In this chapter, we have introduced audio hashprints. We describe the motivation behind the representation, the mechanics of computing hashprints, the formulation of the unsupervised filter learning problem, the rationale of the hashprint design, and the relationship of various aspects of the hashprint design to previous work. Having defined what audio hashprints are, we will now see how hashprints are actually used in practice.

[3]More precisely, the weights for each output node will reflect the filter coefficients for the current frame and the filter coefficients for the corresponding delta frame. The main text is left as is for clarity of illustration.

Chapter 5

Application: Aligning Meeting Recordings

Now that we have introduced and explained the audio hashprint representation, we are ready to see how it can be used in practice. Our first application scenario is aligning meeting recordings. We will first introduce the problem statement (section 5.1), discuss relevant prior work (section 5.2), describe the proposed system (section 5.3), show experimental results (section 5.4), perform various analyses of interest (section 5.5), and then summarize several practical takeaway lessons (section 5.6).[1]

5.1 Problem Statement

More and more, mobile computing devices are carried by their users at all times, including when they engage in meetings with others. In this chapter we explore an application scenario in which multiple mobile devices could be used to generate a reasonably high-quality recording of a meeting, offering a low-cost alternative to potentially expensive recording equipment or software. In this scenario, meeting participants would use their mobile phones, tablets or laptop computers as audio recording devices in an unsynchronized manner. No matter when they arrive at the meeting, participants place their mobile devices on the table in front of them and begin recording. Assume person A arrives at time $t = 0$ minutes and begins recording. Person B arrives at time $t = 2$ minutes. Person C joins remotely via skype at $t = 5$ minutes, and he too simply places his mobile phone in front of him at his remote location. Person D arrives late at time $t = 25$ minutes and begins recording. Some people leave the meeting early; others stay late. At the end of the meeting, everyone has an audio recording. We would like to take these partial, unsynchronized, overlapping audio recordings and generate a single high-quality "summary" recording of the entire meeting. This chapter describes a method for accomplishing this in an efficient and robust manner.

[1] Much of the work in this chapter was published in a journal article entitled "Robust and Efficient Multiple Alignment of Unsynchronized Meeting Recordings" [96].

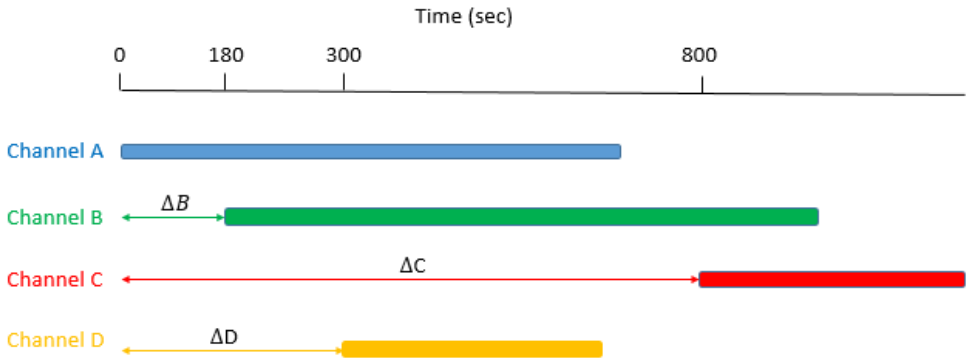


Figure 5.1: Graphical depiction of the temporal alignment problem.

The main problem that needs to be addressed in this application scenario is to align the audio files with each other in time, as shown in figure 5.1. Once the audio files are aligned in time, we can generate a summary recording by simply averaging the audio channels or using a blind beamforming approach. Note that the term "alignment" often refers to aligning text and audio (e.g. forced alignment), whereas here we are aligning audio to audio. No transcriptions are necessary for this type of alignment.

Note that explicit timestamping of recordings would not be a reliable way to align files. Clocks on mobile devices are not typically synchronized, and might not even record time at the required precision (on the order of milliseconds).

The most straightforward content-based alignment method is to use a simple cross-correlation method. While cross-correlation might work for aligning audio files with small time offsets, it may be prohibitively expensive for situations where the recordings are an hour long and the time offset might be 25 minutes, as in the example above. Additionally, simple cross-correlation alone will not handle transitive relationships: if A and B overlap, and B and C overlap, but A and C do not overlap, we should still be able to align all three using the information given. One can immediately come up with several ideas to improve the efficiency of a cross-correlation approach: performing the cross-correlation on FFT data rather than time samples, using only a segment of audio to compute the correlation estimates, etc. This paper represents the development of one such line of thought taken to its logical conclusion.

This chapter describes an approach to the multiple alignment problem that utilizes the hashprint representation described in chapter 3. The fact that hashprints are learned in an unsupervised fashion makes it possible to learn the fingerprint design on-the-fly, so that the fingerprint representation is adapted to each alignment scenario (i.e. group of audio files to be aligned) rather than being fixed based on a global training set. One significant benefit of this approach is that, because the fingerprint design is learned on-the-fly, our system requires little or no tuning. The robustness of the fingerprints can be adjusted very easily by tuning two hyper-parameters, according to the amount of computation we are willing to

perform. Once the audio is represented in a binary fingerprint representation, the alignment is accomplished using an iterative, greedy two-stage alignment algorithm which performs a rough alignment using efficient indexing techniques, and then refines the alignment estimate based on Hamming distance.

5.2 Related Work

Our approach for the alignment of multiple overlapping meeting recordings grows out of the development of audio fingerprinting techniques. These audio fingerprinting techniques largely developed in the context of music identification and online copy detection, which we discussed in chapter 1. In this section, we point out other applications of audio fingerprinting that are relevant to the task at hand.

Audio fingerprinting techniques have been applied to a variety of other applications besides music identification and copy detection. These include detecting repeating objects in audio streams [48][27][72][35][66], recognizing a TV channel in real-time [7], synchronizing two different versions of a movie [19], of two TV audio streams [18], or of a music video and a studio album track [56], and performing self-localization of multiple recording devices [38]. Of particular interest to the aligning meetings scenario, several works have explored the synchronization of consumer videos of the same live event. Shrestha et al. [88] apply the Philips fingerprint to match pairs of video in order to synchronize several video recordings of the same live event. Kennedy and Naaman [46] likewise apply the Shazam fingerprint in a pairwise manner to synchronize videos of live concert recordings. Su et al. [93], Bryan et al. [8], and Six and Leman [90] extend the work of Kennedy and Naaman by applying additional post-processing steps such as clustering or a more refined alignment.

The work described in this chapter explores a specific application which has hitherto not been studied: aligning unsychronized audio recordings of meetings, such as might be collected from participants' personal devices. This application scenario presents some unique challenges and requirements which led to the development of the hashprint design. While this dissertation presented hashprints in a previous chapter for pedagogical reasons, the hashprint design was actually developed in the context of solving the meeting alignment problem.

The primary contribution of our work on aligning meeting recordings is developing a method to learn an audio fingerprint design in an *unsupervised* manner. This allows the fingerprint representation to be adaptive to each alignment scenario (i.e. a set of meeting recordings that need to be aligned). Rather than fixing a fingerprint representation based on a separate training set, the fingerprint design is instead learned on-the-fly in a completely unsupervised fashion and adapted to perform well on the data at hand. Many fingerprint approaches are manually designed or learned in a supervised manner, but, to the best of our knowledge, this is the first entirely unsupervised adaptive fingerprinting method. This method requires very little data to train (on the order of seconds of speech), is efficient to compute, and works without any labeled data, which allows us to tailor the fingerprint design

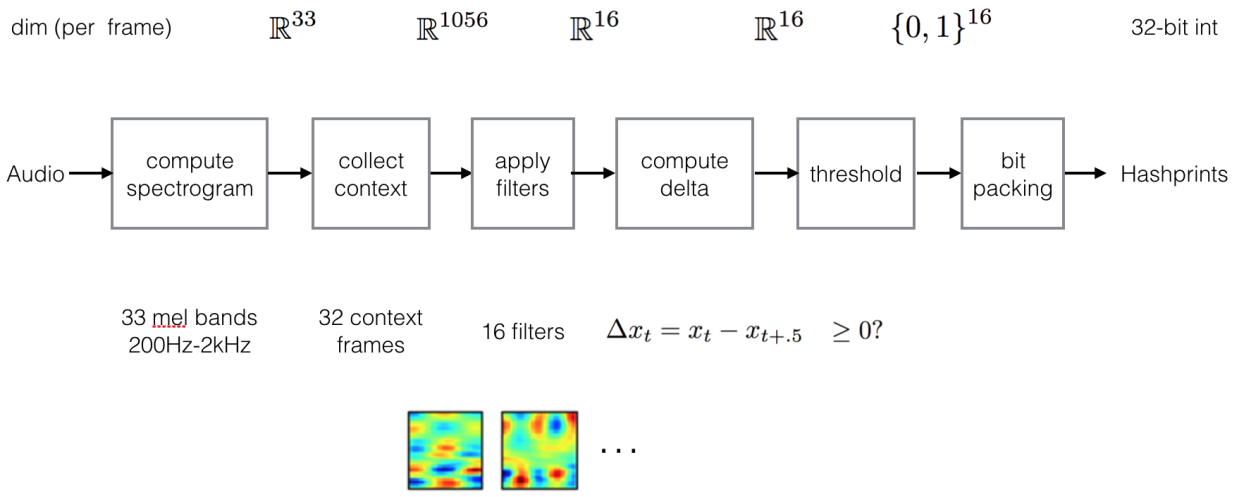


Figure 5.2: Block diagram of the hashprint computation for the aligning meeting recordings scenario.

to the particular characteristics of each alignment scenario. This method is, of course, the hashprints that we have described in chapter 3.

It is important to note that our current problem should not be confused with research work on microphone arrays. Research on microphone arrays focuses on small time lags between microphone elements in order to infer source location or inform beamforming weights. These works typically assume that the microphone elements record audio in a coordinated manner or are approximately synchronized, so that simple cross-correlation over small time lags can be used to align the recordings. Our focus here is on aligning audio files which might be offset by an arbitrary amount of time (e.g. 30 minutes), or may not be directly overlapping at all (i.e. A overlaps with B, B overlaps with C, but A and C do not overlap).

5.3 System Description

The proposed system will be described in two parts: the hashprint representation and the alignment algorithm.

Hashprint Representation

We explained how hashprints are computed in chapter 4. Here, we revisit that explanation, filling in each step with details that are specific to this application. The six steps are described below.

1. **Compute spectrogram.** Because the underlying source signal in a meeting scenario is speech, we compute a log mel spectrogram for the time-frequency representation.

We used 100 ms windows in time with 10 ms hop size to generate a linear spectrogram. We then integrated over 33 Mel frequency bands between 200Hz and 2000Hz and took the logarithm of band energies. These settings are similar to those used in several previous audio fingerprinting works [32][45][3][2].

2. **Collect context frames.** When computing the fingerprint at a particular frame, we consider $w = 32$ frames of context. So, at each frame we are working with a vector of dimension $33w = 1056$. In section 5.5, we will explore the effect of w on system performance.

3. **Apply spectro-temporal filters.** We compute a set of $N = 16$ features at each frame by applying N different spectrotemporal filters. So, each feature is a linear combination of the log Mel spectrogram values for the current frame and surrounding context frames. As pointed out in chapter 4, MFCCs are a special case of spectrotemporal filters in which the filter coefficients match the coefficients of the discrete cosine transform. Rather than using MFCCs, however, we use filters that capture the directions of maximum variance, as explained in section 4.3. In section 5.5, we will explore the effect of N on system performance.

4. **Compute deltas.** For each of our N features, we compute the change in the feature value over a time lag T. If the feature value at frame n is given by x_n, then the corresponding delta feature will be $\Delta_n = x_n - x_{n+T}$. As explained in section 4.4, we would like the choice of T to ensure that the underlying audio signal has become decorrelated with itself. Given that the underlying source signal will be speech and that typical speech rates in American English range between 110 - 150 words per minute, we select $T = 50$ frames (.5 seconds) as a conservative value that ensures decorrelation. It should be noted, though, that the value of T could be determined empirically on-the-fly by measuring autocorrelation, if necessary.

5. **Apply threshold.** Each of the N delta features is compared to a threshold value of 0, which results in a binary value. These bits represent whether the N features are increasing or decreasing across the time lag T.

6. **Bit packing.** The N binary values are packed into a single 32-bit integer which represents the fingerprint value for a single frame. This compact binary representation will allow us to store fingerprints in memory efficiently and to do reverse indexing in order to quickly look up fingerprint matches.

Figure 5.2 shows a block diagram of the hashprint computation process. The text above the boxes shows the dimension (per frame) at each stage of the process. The text below the boxes provides additional clarifying information. Note that the values shown in figure 5.2 refer to the "default" parameter settings for the hashprint system shown in the results section. We will explore the effect of several of these system parameters on system performance in section 5.5.

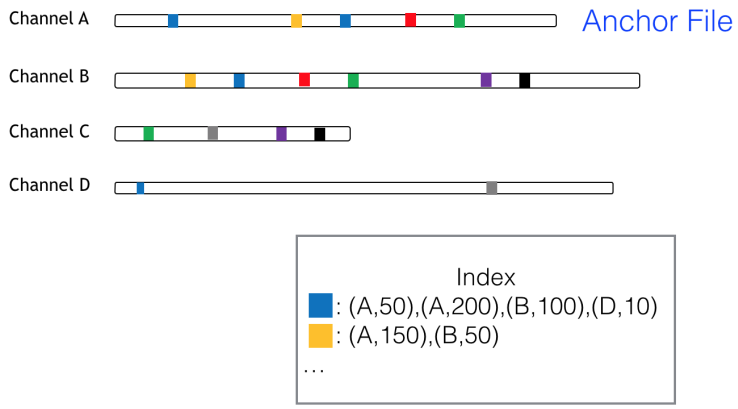


Figure 5.3: Illustration of example problem after step 1. Hashprint values are indicated as different colored boxes. For clarity, only some selected hashprint are shown. A reverse index is created on the hashprint values, and one of the recordings is selected as an anchor file to provide a time frame of reference.

Alignment Algorithm

In this subsection, we describe the algorithm used to align the multiple audio recordings in time. Consider the graphical depiction of the problem in Figure 5.1. Here, we see four different audio recordings denoted by A, B, C, and D. Person A begins recording first ($t = 0$), person B begins recording 180 seconds into the meeting, and so forth. Note the nontransitive relationships among the files: A overlaps with B, and B overlaps with C, but A and C do not overlap. It will be important for our algorithm to be able to handle these nontransitive relations, rather than simply comparing files in a pairwise manner.

The alignment algorithm has four steps, each described below. In order to make the explanation more clear, we also include illustrations on a sample problem in figures 5.3, 5.4, 5.5, and 5.6.

- **Step 1: Initialization.** The initialization step has four components. First, we learn the spectro-temporal filters for the hashprint design using the formulation described in section 4.3. This determines the N spectro-temporal filters that are adapted to the data at hand. Second, we use the learned filters to compute hashprints on all the audio recordings in the current alignment scenario. In the example shown in Figure 5.1, this means extracting hashprints from recordings A, B, C, and D. Third, we create a database which contains triples of the form $(hp, fileid, offset)$, where hp specifies the 32-bit hashprint value, $fileid$ specifies the audio recording, and $offset$ specifies the frame offset relative to the beginning of the audio recording. To make the hashprint lookups more efficient, we also create a reverse index which maps hashprint values to the list of triples with that hashprint value. Fourth, we select one of the audio

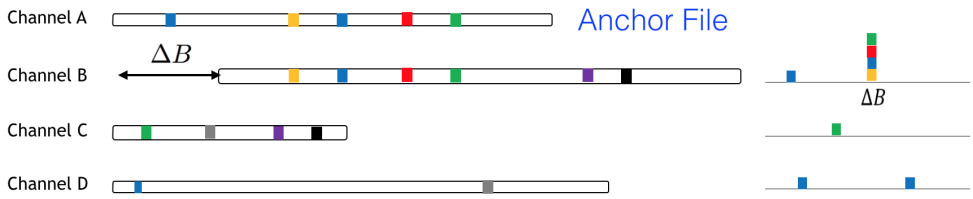


Figure 5.4: Illustration of example problem after step 2. Using the anchor file as the first query, we accumulate a histogram of the time offsets between matching hashprints. The histogram bin with the highest count indicates which recording has the strongest match, and it also indicates the rough relative alignment. In step 3, this rough alignment is adjusted using a more fine-grained alignment.

recordings to serve as our "anchor" file. In our experiments, we selected the anchor file to be the audio recording with highest average energy. All other time indices will be computed relative to the beginning of this anchor file. We denote this time index as the universal time index. The time scale in Figure 5.1 shows the universal time index when recording A is selected as the anchor file. Figure 5.3 shows the state of the alignment system after this step has been completed. Note that hashprint values are represented graphically as different colored boxes, and that only selected hashprints are shown for clarity of illustration.

- **Step 2: Find the best match.** Using the anchor file as a query, we find the audio recording that has the strongest match. We use the method proposed in the Shazam algorithm [99], which is based on histograms of time differences. A brief explanation is given here, and the reader is referred to the Shazam paper for more details. For every hashprint h at time offset $offset_{query}$ in the query file, we look up the list of matching hashprint triples $(h, U_i, offset_{U_i})$ in the database, where U_i, $i = 1, 2, \ldots, k$ is the $fileid$ for one of the k currently unaligned audio recordings. If the query file and an unaligned file U_i overlap in time, we expect there to be a lot of matching hashprint triples with a fixed time difference $\Delta t = offset_{query} - offset_{U_i}$. So, we can estimate the true alignment between the query and U_i by accumulating a histogram of the matching hashprint time differences Δt, and then scanning the histogram counts for a peak. If there are a lot of matching fingerprints at a particular time offset Δt, then Δt indicates the relative alignment between the query file and the unaligned audio recording. Note that there may be many spurious hashprint matches throughout the database, but it is very unlikely to have a lot of hashprint matches aligned in time. In this way, we accumulate a histogram of time offsets for each unaligned audio recording U_i, and we take the maximum bin count of each histogram as the match score for U_i. The unaligned audio recording with the highest match score is identified as the best match. Figure 5.4 shows the state of the alignment system after step 2 has been

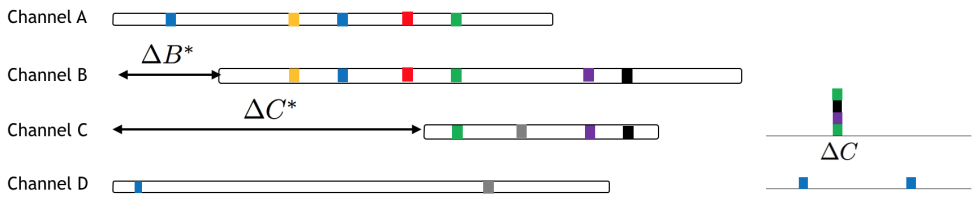


Figure 5.5: Illustration of example problem after repeating steps 2 and 3. After each file is aligned, we repeat the process using the most recently aligned file as the new query. We can again accumulate a histogram of matching hashprints to determine the strongest match among the remaining unaligned files, and then refine our estimate using a fine-grained alignment.

completed. Note that in this toy example, channel B has the strongest match with 4 matching hashprints at a relative offset of ΔB.

- **Step 3: Fine alignment.** We know the approximate offset Δt between the query file and the best match. However, this offset is not very precise, since its precision is limited by the width of the histogram bins. Also, since a fingerprint match requires all N fingerprint bits to be correct, the match score ignores a lot of more fine-grained information about fingerprint agreement. For these reasons, we do a fine-grained alignment between the query file Q and U^*, the unaligned audio recording with the best (rough) match score. We consider a range of possible offsets $[\Delta t - B, \Delta t + B]$, where B represents a search window size in frames. For each possible offset, we compare the corresponding hashprints in Q and U^* and determine what percentage of the hashprint bits agree. Note that here we are comparing individual hashprint bits, which allows us to detect partial matches, unlike in the best match step that compares only the full 32-bit hashprint values. These bit comparisons can be computed very efficiently using bit arithmetic, and they allow us a much more precise estimate of hashprint agreement. The offset Δt^* which yields the highest hashprint agreement is selected, and it specifies the starting time of U^* on the universal time index. U^* is added to the list of aligned files.[2] After this third step, the state of the alignment system will look exactly like figure 5.4, except that the relative alignment ΔB will be replaced with a more fine-grained alignment estimate $\Delta B*$.

- **Step 4: Repeat steps 2 and 3.** We repeat step 2 using the most recently aligned file as the query file. For all aligned files, frame offsets are adjusted to represent the universal time index. In other words, hashprint tuples $(hp, fileid, offset)$ will effectively become $(hp, fileid, offset + \Delta t^*)$. When accumulating histogram counts

[2]The method described above is useful for finding an approximate alignment on the order of the frame size (10 ms) very efficiently. If even finer precision is needed, one could do an additional cross-correlation in the time-domain around the approximate alignment.

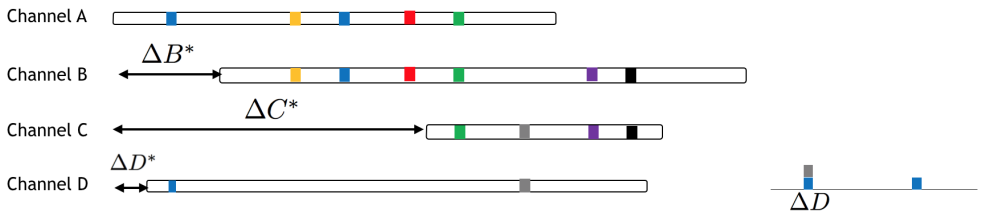


Figure 5.6: Illustration of example problem after all files have been aligned. The final relative alignment estimates are given by ΔB^*, ΔC^*, and ΔD^*.

for the current query file, we retain the histograms from previous steps and simply add additional counts. In this way, we accumulate evidence from all of the previously aligned files to help match unaligned files. This means that when we align the last recording (which will be the recording that had the lowest match scores), we will have the most data and evidence to help determine the optimal alignment. Steps 2 and 3 are thus repeated in like fashion until all files have been aligned. Figure 5.5 shows the state of the alignment system after steps 2 and 3 have been repeated to align channel C. Figure 5.6 shows the final state of the alignment system after all recordings have been aligned. Note that channel D has two hashprint matches in its histogram of offsets. Even though channel D only has one matching hashprint with channel A and one matching hashprint with channel C, the histogram aggregates the *cumulative* evidence from all previously aligned files. The final relative alignment estimates are given by $\Delta B*$, $\Delta C*$, and $\Delta D*$.

At the end of this process, we have estimates of the relative alignment among all the audio recordings. Figure 5.1 shows a possible representation of the alignment estimates after the entire alignment process has been completed.

5.4 Evaluation

The evaluation of the proposed system will be explained in three parts: the data, the evaluation metric, and the experimental results.

Data

To evaluate our system, we ran experiments on data extracted from the ICSI meeting corpus [43]. The original data set consists of multi-channel audio recordings of 75 real research group meetings, totaling approximately 72 hours of meetings. For each meeting, participants wore headsets which captured audio through close-talking microphones. Several tabletop microphones spread across the conference room table also collected audio data. These tabletop microphones included four high-quality omnidirectional microphones and two low-quality

microphones, each mounted on a small block of wood. The meetings ranged in length from 17 to 103 minutes, and the number of simultaneous audio channels ranged from 9 to 15 channels. The data set also contains manual annotations of what people said, who spoke, and when they spoke.

The ICSI meeting corpus provides useful source material that we can use to generate realistic query scenarios for the task at hand. The corpus has three important characteristics that make it suitable for our experiments. First, the data contains multiple synchronized recordings. The ICSI data set contains simultaneous audio recordings that are synchronized, which gives us ground truth alignments.[3] Second, the data has microphones placed throughout the conference room. In a realistic scenario where a group uses their portable devices to record a meeting, there will be diversity in microphone location, so this is an important characteristic to maintain. Third, the data contains a variety of microphone characteristics. In our scenario of interest, users would have different types of portable devices which would have different microphone characteristics, so diversity in microphone characteristics is an important aspect. The meeting data contains close-talking and far-field microphones, as well as both high-quality and low-quality microphones. It is useful to point out that data collected from a microphone array generally does not satisfy characteristics 2 and 3 above. While using actual data with mobile phones would be ideal, coordinating the collection of such a data set in sufficient quantity is outside the scope of this work. For the reasons described above, the ICSI meeting corpus provides good source material for our experiments.[4]

We generate each alignment scenario as follows. Given the audio data for a single meeting, we randomized the ordering of audio channels and performed the following steps on each channel in the random order.

1. Select an audio segment length from a uniform distribution $[0, T]$. In our experiments, we selected T to be 10 minutes.

2. Randomly select a time interval of this length from the full audio recording.

3. Verify that the selected audio segment has 30 seconds or more of temporal overlap with at least one other previously selected audio segment. If it does not, repeat Steps 1 and 2 until this condition is met.

In this way, each audio channel generates a randomly chosen audio segment, and every audio segment is guaranteed to have at least 30 seconds of overlap with at least one other audio segment. Since the above process is probabilistic, we can generate multiple query scenarios from a single meeting. We generated 10 query scenarios from each of the 75 meetings,

[3]Due to issues in the recording software, there were slight variations in the alignment of the individual audio channels. These misalignments are on the order of milliseconds, which is tolerable for the range of precision that we will study in this chapter.

[4]The AMI Meeting Corpus [10] would be another data set that is suitable for our study. We chose to use the ICSI meeting corpus because the meetings are not scripted, and there is greater diversity in the number of meeting participants and microphone types.

resulting in a total of 750 query scenarios and approximately 8500 alignments. We used 37 of the meetings for "training" the system, and the other 38 meetings for testing. Since our fingerprint design is entirely learned on-the-fly for each alignment scenario, there is no supervised training to do. The "training" set was primarily used for system debugging and for learning appropriate values for a few system hyper-parameters such as the histogram bin width.

Note that the above process of generating queries is probably more difficult and challenging than a typical use case scenario, since users would probably all record a very substantial chunk of the meeting, with an occasional user leaving the meeting early or entering very late. However, generating more difficult alignment scenarios with shorter audio segments and shorter amounts of temporal overlap will enable us to better characterize and test the robustness of our system.

Evaluation metric

We evaluate our proposed system by measuring the robustness and accuracy of the alignments in the following manner. Consider a single alignment scenario, such as the one depicted in Figure 5.1. If we use channel A as an anchor file, we can compute the time offset of all other files relative to A. These time offsets are denoted in Figure 5.1 as ΔB, ΔC, etc. Our alignment system will produce a set of hypotheses $\Delta B_{hyp}, \Delta C_{hyp}, \cdots$ for each alignment scenario. Since we generated the alignment scenario ourselves, we also know the true offsets $\Delta B_{ref}, \Delta C_{ref}, \cdots$. We then compare the estimated offset for each audio recording to the true offset. Let e denote the difference between the estimated offset (e.g. ΔB_{hyp}) and the true offset (e.g. ΔB_{ref}). If $|e| > \gamma$, where γ specifies an error tolerance, we consider that particular alignment to be incorrect. We can compute the fraction of alignments that are correct at a fixed error tolerance. By sweeping across a range of γ values, we can characterize the tradeoff between accuracy and error tolerance.

Note that an alignment scenario with K audio recordings will generate $K-1$ predictions that are either correct or incorrect. Our accuracy versus error tolerance curves aggregate the results of these predictions over all alignment scenarios. In addition to the accuracy versus error tolerance tradeoff, we can also succinctly characterize the performance of a system by looking at the accuracy at a fixed error tolerance.

It is useful to point out that the anchor file for scoring and the anchor file in our alignment system (as described previously) are totally independent concepts. Our evaluation metric should not depend on our selection of scoring anchor file, since this selection is arbitrary. Accordingly, for each alignment scenario, we consider all possible channels as the scoring anchor file, and choose the one which yields the highest accuracy. This step is necessary to prevent an unlucky selection from unfairly penalizing the results. For example, if the scoring anchor file is aligned incorrectly, the $N-1$ predicted alignments will all be incorrect, even if the other $N-1$ files are aligned correctly amongst themselves. By considering all possible scoring anchor files, this situation would (correctly) yield $N-2$ correct alignments and 1 incorrect alignment.

Results

In this section we present experimental results for our proposed system. As a baseline comparison to our adaptive fingerprint design, we also ran experiments with five other fingerprint designs: the Philips fingerprint [32], the Shazam fingerprint[5] [99], the boosted fingerprints proposed by Ke et al. [45], the MASK fingerprint [2], and the Panako fingerprint [89]. The Philips and Shazam fingerprints are the most well-known approaches in the literature, the work by Ke and colleagues is one of the most highly cited works among those that incorporate boosting into the fingerprint design process, and the MASK and Panako fingerprints are relatively recent works that extend previous approaches to provide greater robustness to changes in pitch and/or tempo. Thus, these five fingerprint designs span a range of different approaches and include both well-known and recent works.

Figure 5.7 shows the tradeoff between alignment accuracy and error tolerance for the six different fingerprints. To make the comparison as fair as possible, all six fingerprints were evaluated using the same cumulative alignment algorithm. However, there is one important difference to mention. The fine alignment step described in section 5.3 assumes that fingerprints are computed at every time frame and that the Hamming distance between fingerprints corresponds to a measure of dissimilarity. For the three approaches that do not satisfy these assumptions – Shazam, MASK, and Panako – the fine alignment step was omitted. For the experiments, we used the default parameter settings in the provided implementation or reference paper. The proposed adaptive fingerprint in Figure 5.7 uses 16 bits and 32 frames of context.[6] The ordering of the legend corresponds to the ordering of performance at 100ms error tolerance.

There are three things to notice about Figure 5.7. First, there are two separate dimensions along which we can measure the performance of a fingerprint design: precision and robustness. Note that precision has several meanings in different contexts (e.g. precision and recall, bit precision, etc.), and here we use precision to describe how precisely in time a fingerprint can localize a segment of audio. In figure 5.7, precision refers to the error tolerance at which the performance curve levels off. Robustness, on the other hand, refers to the accuracy at which the curve levels off. It is important to point out that these two dimensions are not necessarily correlated. Some fingerprints have high precision but low robustness, such as the Philips fingerprint. Other fingerprints have high robustness but low precision, such as the Shazam and Panako fingerprints. Both dimensions are important to consider in evaluating the effectiveness of a fingerprint design.

Second, the *relative* performance of the proposed system is very good. Among the six fingerprint designs that were evaluated, the adaptive fingerprint has the best performance both in terms of precision and robustness. The adaptive fingerprint is approximately tied with the Philips fingerprint for highest precision – they both level off at an error tolerance of around 50 ms. It is also slightly more robust than the three other highly robust fingerprints: Shazam, MASK, and Panako. It is interesting that all three of these other approaches level

[5]For the Shazam fingerprint, we used the implementation provided by Dan Ellis [21].

[6]The choice of 16 bits is discussed in Section 5.5.

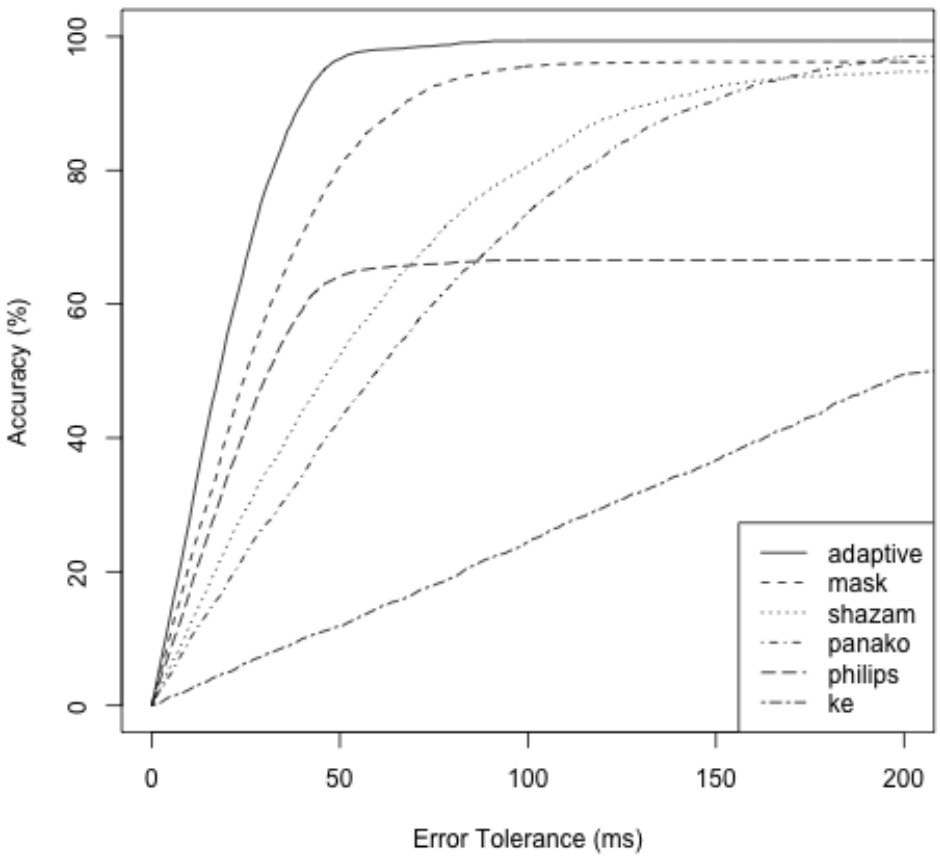


Figure 5.7: The tradeoff between accuracy and error tolerance for six different fingerprints. The ordering of the legend corresponds to the performance ordering at 100 ms error threshold.

off at approximately the same accuracy, perhaps because all three approaches focus on identifying the location of spectral peaks. While different fingerprints may offer different tradeoffs between precision and robustness, the decision here is clear: the adaptive fingerprints are best along both dimensions.

Third, the *absolute* performance of the proposed system is very good. Beyond simply performing well relative to other fingerprints, the hashprint has very good absolute performance numbers. As seen in Figure 5.7, the adaptive fingerprint achieves an accuracy of 99.4% for 100 ms error tolerance. The errors from this system generally came from close-talking microphone channels that contained only silence (e.g. the channel was not used or the person was silent) or local noise (e.g. the person wore the headset too closely and the microphone picked up their breathing patterns). Furthermore, we can improve the robust-

ness of the alignment even more (if needed) by providing more context information to the hashprint and by reducing the number of fingerprint bits in the lookup. We will explore the effect of these two factors on fingerprint robustness in the analysis section. We simply note here, however, that the proposed system works very reliably and robustly.

5.5 Analysis

In this section we will investigate and answer six questions of interest. These investigations will develop intuition and understanding of the inner workings, capabilities, and limitations of the proposed alignment system.

Effect of Number of Lookup Bits

The first question we will answer is, "How does the number of bits in the hashprint representation affect its robustness?" In many other works, fingerprints are often characterized by 32 bits so that each fingerprint can be represented compactly as a single 32-bit integer. Here, we investigate how the number of bits affects robustness.

Before presenting any experimental results, we can approach the question from a theoretical standpoint. Note that using a higher number of lookup bits results in higher specificity but lower accuracy. To see this, consider a 32-bit fingerprint whose bits are uncorrelated and balanced (each bit is 1 half the time and 0 half the time). If each bit independently has a $\alpha = 90\%$ probability of being correct given a noisy true match, then the fingerprint would be correct $(.9)^{32} \approx 3.4\%$ of the time. When compared to a randomly selected fingerprint, we would expect a random match approximately $\frac{1}{2^{32}}$ of the time. This corresponds roughly to one spurious match for every $10,000$ hours of audio. Clearly, this is far more specificity than we need for our application of interest, which involves a few tens of hours of audio at most. Now consider a 16-bit fingerprint in the same hypothetical scenario. This fingerprint would be correct $(.9)^{16} \approx 18.5\%$ of the time, and it would have roughly one spurious match for every 10 minutes of audio. This is a much more reasonable choice for our application of interest. The tradeoff essentially comes down to this: reducing the number of lookup bits by 1 increases the fingerprint true match accuracy by a factor of α (which was .9 in the example above) but also increases the number of spurious matches by a factor of 2.

Now that we know what the results should look like, we present our experimental results investigating the effect of the number of lookup bits. Figure 5.8 shows the accuracy (at a fixed 100 ms error tolerance) of the hashprint for three different lookup key sizes: 16, 24, and 32 bits. We did not run experiments with an 8-bit lookup since processing the large number of spurious matches would result in very long run times. Each group of bars compares the effect of lookup key size on a hashprint with a fixed amount of context, where we consider 2, 8, and 32 frames of context information.

As we expect, reducing the number of fingerprint lookup bits improves system accuracy. This improvement is dramatic for the Philips fingerprint, since the original 32-bit fingerprint

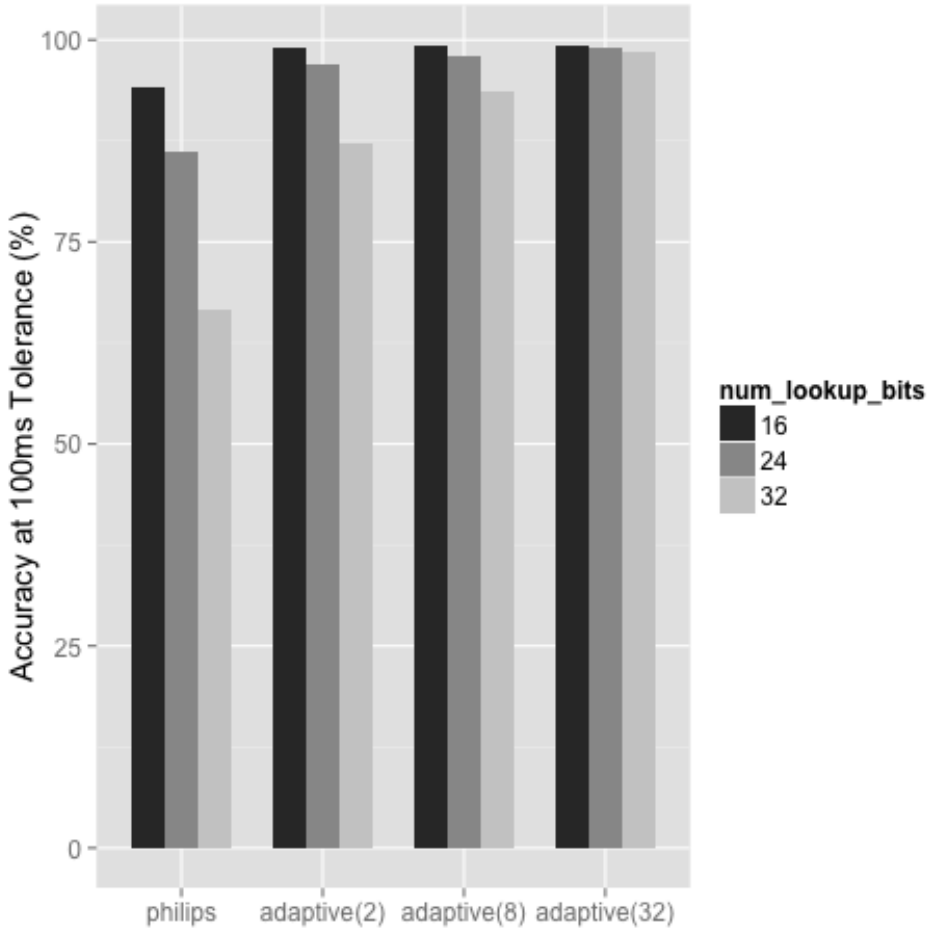


Figure 5.8: Determining the effect of the number of fingerprint lookup bits. The leftmost group shows the performance of the Philips fingerprint with a 16-, 24-, and 32-bit lookup. The three rightmost groups show the same comparison for the adaptive fingerprints with 2, 8, and 32 frames of context.

leaves a lot of room for improvement. For the more robust hashprints, there is still a clear but less dramatic improvement, since the results are nearly saturated already. We can also see the effect of context from this figure, but we will defer discussion of this until a later analysis subsection.

Effect of Overlap

The second question we will answer is, "How much temporal overlap is required to correctly align files?" This question will help us understand the conditions necessary to align recordings

correctly in meeting scenarios.

To answer this question, we created a slightly different experimental setup in order to isolate the effect of temporal overlap. This modified setup makes two changes to the main experimental setup described previously. First, only two randomly selected channels (rather than all channels) are used to generate each query scenario. This simplifies the setup and allows us to focus on the effect of the amount of overlap between two recordings. Since close-talking microphones often contain extended amounts of silence and we will be considering short amounts of overlap (in the range of a few seconds), we only considered the six tabletop microphone channels for these experiments. Second, we deterministically control the lengths of the two audio segments rather than selecting the lengths randomly as in the previous experiments. Specifically, one channel is selected to be the reference channel and is kept in its entirety (i.e. the entire original audio recording is used). The other channel is selected to be the query, and an N second segment is randomly selected from that channel. Thus, we are given an N second query from one tabletop microphone and we are trying to identify where in the meeting the query occurs in a different tabletop microphone. We can then measure how our accuracy of alignment depends on the length of query. Note that the meetings in the ICSI meeting corpus are typically around an hour long, so with an error tolerance of 100 ms the accuracy of random guessing would be about .006%.

Figure 5.9 shows the results of our overlap experiments. Each curve shows the effect of query length on the alignment accuracy at a fixed 100 ms error tolerance. Each point on the curve represents the accuracy averaged over approximately 1100 queries. The top three curves correspond to the hashprints with 2, 8, and 32 context frames, and the lowest curve corresponds to the Philips fingerprint.

There are two things to notice about the results in Figure 5.9. First, there is a dramatic improvement in using hashprints rather than the Philips fingerprint. For 15 second queries, for example, the hashprints have alignment accuracies of 94% and higher, while the Philips fingerprint has an accuracy of 14%. Second, using more context frames improves alignment robustness and shortens the minimum required temporal overlap. Note that the amount of temporal overlap needed to achieve saturated performance decreases as we include more and more context. With 2 context frames, we need about 30 seconds of overlap. With 8 context frames, this number drops to about 15 seconds. With 32 frames of context, 10 seconds of overlap is sufficient to reliably ensure correct alignment. These numbers assume typical meeting dynamics (i.e. the overlap will contain natural pauses but probably not long, extended silence).

Effect of Amount of Training Data

The third question we will answer is "How much data is necessary to learn a robust fingerprint design?" When meetings are very long, we would like to know how much data is actually necessary to learn a robust fingerprint design. Alternatively, when meetings are short, we would like to know if the amount of data is sufficient to learn a reasonable fingerprint design.

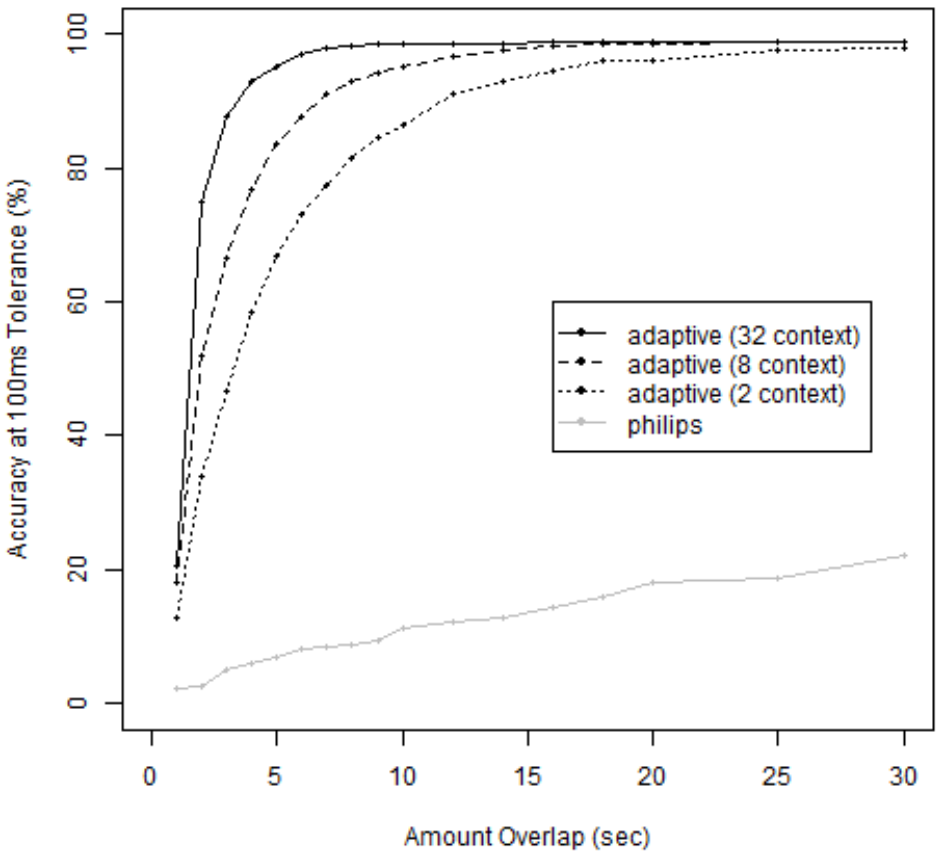


Figure 5.9: Determining how much temporal overlap between two recordings is necessary to find a correct alignment. The top three curves show the performance of the hashprint with 2, 8, and 32 frames of context. The bottom curve shows the performance of the Philips fingerprint.

To answer this question, we ran the original set of experiments with one change: instead of learning the eigenfilters on all of the available data, we selected one T-second random segment from each available channel and learned the eigenfilters only on the selected data. By varying the amount of available training data, we can determine how much data is necessary to learn a robust fingerprint design.

Figure 5.10 compares the alignment accuracy for adaptive fingerprints with T =1, 5, 30, and ∞ (using all available data for training). Each group of bars corresponds to a hashprint with a fixed amount of context frames, where we again consider 2, 8, and 32 frames of context.

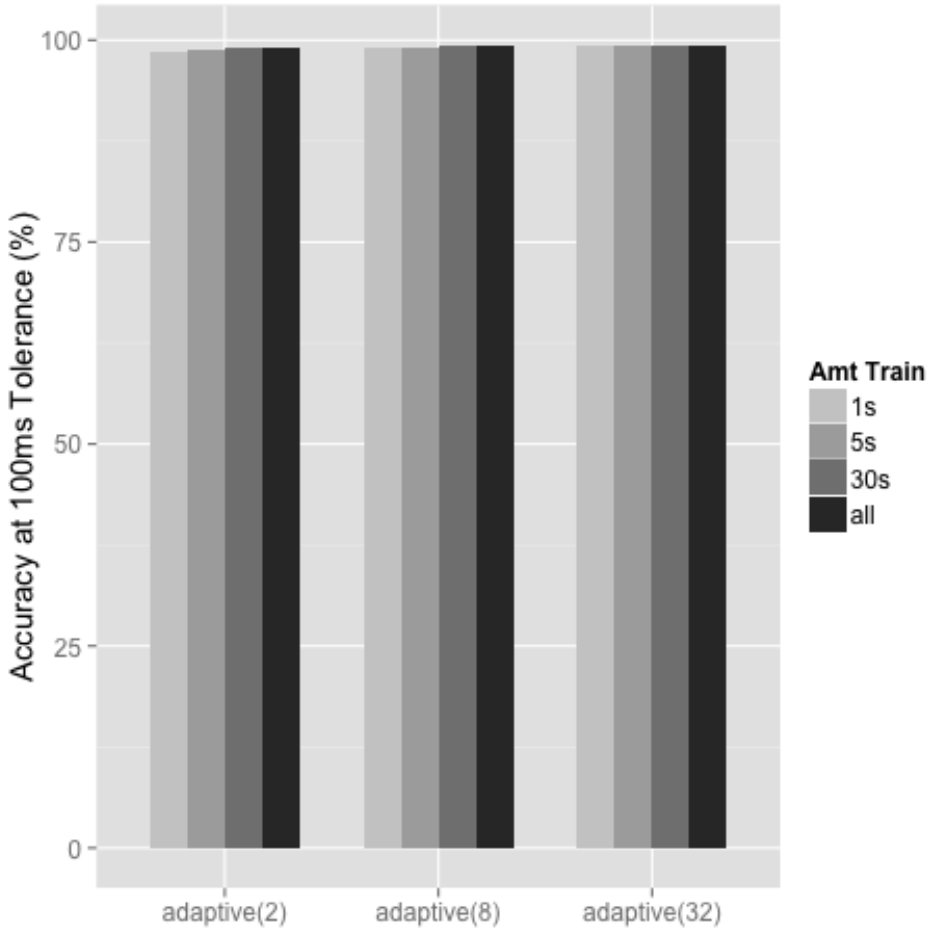


Figure 5.10: Determining the effect of the amount of training data. A T-second segment is randomly selected from each channel, and the fingerprint design is learned only on the selected data. Performance is shown for $T = 1$, 5, 30, and ∞ (use all available data). The three groups of bars show the performance of hashprints with 2, 8, and 32 frames of context.

Surprisingly, even just a 1 second segment from each channel provides enough information to learn a reasonable fingerprint design with good performance. The performance of the hashprints is approximately saturated for $T = 5$, so there is only very marginal benefit in using more than 5 seconds of training data from each channel. These results have two very encouraging implications: (1) the hashprints will work well for any length of meeting, even very short meetings, and (2) for very long meetings, we can achieve roughly the same level of performance with less computation by only training the filters on a very small subset of data.

Table 5.1: Comparing several variants of the alignment algorithm. The indicated numbers are the accuracy at a specified error tolerance. All experiments use 16-bit hashprints with 2 context frames.

Alignment Algorithm	Error Tolerance			
	25ms	50ms	75ms	100ms
Pairwise, no refinement	52.1%	82.9%	95.5%	98.7%
Cumulative, no refinement	51.2%	83.3%	95.1%	98.3%
Cumulative, with refinement	66.5%	96.5%	98.5%	99.2%

Effect of Context

The fourth question we will answer is, "How does the amount of context affect fingerprint robustness?" Since we explored a range of context values in all of our previous analysis experiments, we will simply revisit our previous results but now with a focus on the effect of context.

Earlier we saw that using fewer lookup bits improves accuracy at the expense of computation. This can be seen in Figure 5.8 by comparing the results within each group of bars. But we can see the effect of context in the same figure by comparing results *across* the groups of bars. For example, if we look at the rightmost bar in each group, we can see that the accuracy increases from 87.2% to 93.5% to 98.4% as we increase the context from 2 to 8 to 32 frames. For fewer lookup bits, we see a similar but less dramatic improvement, since the results are closer to saturation. Clearly, using more context makes the system more robust, though the amount of improvement in system-level accuracy depends on how saturated the results are.

We can similarly revisit the results in Figure 5.10 with a focus on context. Because so little data is needed to train robust filters, however, we see little differentiation between different amounts of context. For all practical application scenarios, there is more than enough data to learn a robust fingerprint design for up to 32 context frames.

Assessing the Alignment Algorithm

The fifth question we will answer is, "How much actual benefit is gained by aggregating cumulative evidence and doing a refined alignment?" We can tease apart the effect of these two components by starting with a pairwise out-of-the-box alignment approach, and then adding in these components one at a time.

Table 5.1 compares the performance of three different alignment algorithms. The top line shows the performance of a pairwise alignment approach, similar to the works by Kennedy and Naaman [46] and Su et al. [93]. This approach does not aggregate cumulative evidence and does not do a refined alignment step. The second line shows the performance when we aggregate cumulative evidence, but without a refined alignment. The third line shows the

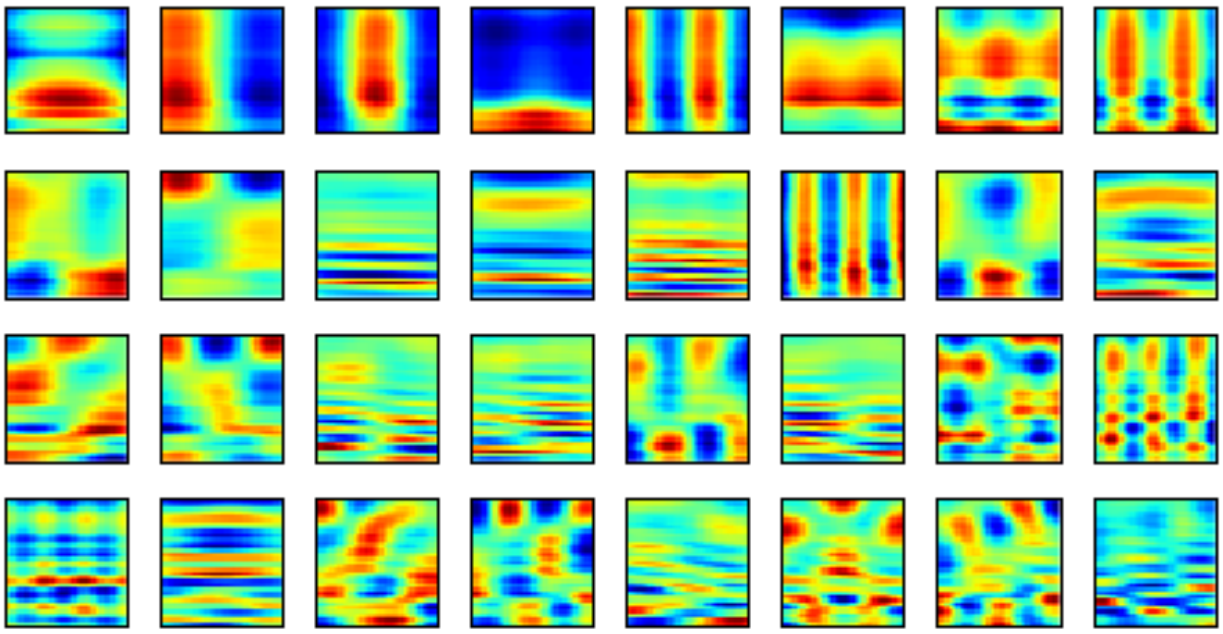


Figure 5.11: The top 32 learned spectro-temporal filters from one alignment scenario. The filters are arranged from left to right, and then from top to bottom.

performance when we aggregate cumulative evidence and perform a refined alignment. All experiments use a 16-bit adaptive fingerprint with 2 frames of context.

The results in Table 5.1 are somewhat surprising. There is a significant benefit in using our proposed approach over a simple pairwise out-of-the-box approach, but *all* of the benefit is coming from the refined alignment step. Comparing the top two rows of the table, we see that the accuracy is roughly the same. Sometimes the accuracy is slightly higher and sometimes it is slightly lower, depending on the error threshold. But there seems to be no measurable benefit (or detriment) to aggregating cumulative evidence in this scenario. On the other hand, doing a refined alignment yields drastic improvements at all error thresholds. For example, at a 25 ms error threshold, the refined alignment improves the accuracy from about 52% to 66%. As we might expect, the refined alignment improves accuracy the most for very small error thresholds.

Learned Filters

The sixth question we will answer is, "What do the learned filters look like?" The purpose of this question is not so much to improve system performance as it is to gain more intuition and understanding into what constitutes a robust fingerprint design.

Figure 5.11 shows the top 32 learned filters for one particular query scenario when the fingerprint is given 32 context frames. Recall that the filters are learned from scratch on

each query, so each query will have its own set of filters. Figure 5.11 shows one example set of filters. The filters progress from left to right, and then from top to bottom. So the upper leftmost is the first filter, and the lower rightmost is the 32^{nd} filter.

There are three observations to make about the filters shown in Figure 5.11. First, modulations in both time and frequency are useful. Some of the filters primarily capture modulations in time, such as filters 2, 3, 5, and 8 in Figure 5.11. Some of the filters primarily capture modulations in frequency, such as filters 1, 4, 6, and 13. Other filters capture modulations in both time and frequency, such as filters 7 and 15. The key thing to point out is that both are important, so we should not emphasize one to the exclusion of the other. For example, giving the adaptive filters only two frames of context forces the filters to focus almost entirely on frequency modulations rather than temporal modulations, since two frames is insufficient to capture much variation in the temporal dimension. Second, low modulations seem to be most important and useful. We can see a progression from low modulations to higher frequency modulations as we get to later and later filters. For example, the second, third, fifth, eighth, and fourteenth filters capture higher and higher frequency modulations in time. In general, we see a progression from slower modulations in the top few filters to faster modulations in later filters. Third, the filters are remarkably consistent from query to query. When we look at the learned filters for many different queries, we observe that the first 8-10 filters are usually very similar and in approximately the same order. As we progress to later filters, there is more diversity and difference from query to query. This suggests that these top few filters are capturing information that is characteristic of the genre of audio data (i.e. meeting speech), rather than something specific to what is being said in a particular meeting.[7]

5.6 Takeaway Lessons

In this section we discuss three practical takeaway lessons from our experimental findings.

Takeaway lesson #1: Hashprints are a *robust* way to align meeting recordings. Our system is able to achieve more than 99% alignment accuracy at a reasonable error tolerance (100ms) on alignment scenarios that are much more challenging and aggressive than would typically be found in practice (i.e. shorter audio recordings and less temporal overlap). The files that our system could not align correctly were close-talking microphone channels consisting almost entirely of silence or local noise such as the speaker breathing into the microphone. Only about 10 seconds of typical meeting speech is necessary to identify a correct alignment. Furthermore, since only a few seconds of data from each recording is needed to reliably learn a robust fingerprint design, the hashprints can be learned quickly and efficiently on meetings of any length, even short ones. In general, the proposed system should be able to align meeting recordings very reliably.

[7]It is also interesting to note that spectro–temporal filters favoring particular ranges of temporal and spectral modulations are commonly associated with cortical processing in the human auditory system (e.g. [71][58]).

L\K	2	4	6	8
5 min	17sec	25sec	38sec	52sec
10 min	24sec	42sec	67sec	97sec
20 min	42sec	80sec	139sec	211sec
30 min	60sec	120sec	214sec	340sec
60 min	125sec	272sec	506sec	795sec

Table 5.2: Average run time required to align K recordings each of length L using the proposed approach. Experiments were run on a 2.2 GHz Intel Xeon processor.

Takeaway lesson #2: The efficiency of the proposed hashprint approach is acceptable. Table 5.2 shows the average run time required to align K recordings that are each exactly L minutes long using the proposed approach. So, for example, aligning 8 files that are each 1 hour long takes 795 seconds on average. Here, we have chosen values of K and L that span a range of realistic scenarios. These measurements were taken on a single thread of a 2.2 GHz Intel Xeon processor. Note that the running time for the proposed approach is acceptable for any realistic scenario. The running times are on the order of minutes, which is acceptable for an offline task. It would also be trivial to parallelize this task for a further reduction in computation time.

Takeaway lesson #3: If more robustness is needed, it can be achieved very simply. There are two parameters that we can modify to improve the robustness of the fingerprint, both of which come at the cost of more computation. The first parameter is the amount of context. We saw in section 5.5 that increasing the amount of context improves the alignment accuracy and reduces the amount of temporal overlap needed for correct alignment. Varying this first parameter increases computation linearly. Note that increasing the amount of context by a factor of N means doing N times as many dense multiplications at each frame when extracting fingerprints. The second parameter is the number of fingerprint lookup bits. We saw earlier that decreasing the number of lookup bits increases the alignment accuracy. Varying this second parameter increases computation exponentially: for every 1 bit reduction in the lookup key size, we will have to process twice as many spurious fingerprint matches.

Given these two parameters, we can adopt the following simple strategy to increase the robustness if needed: First, we increase the context to gain robustness at the expense of a linear increase in computation. If the fingerprint is still not robust enough, we can then begin decreasing the number of fingerprint lookup bits and paying the heavier exponential cost. The number of lookup bits adjusts robustness at a coarse granularity, and the context adjusts robustness at a fine granularity. Because the proposed fingerprints are learned on the fly in an unsupervised manner, there is very little system tuning to do given a new set of data to align. We can start with a reasonable setting (e.g. 16-bit lookup with 32 frames of context) and, if more robustness is needed, we can follow the simple strategy above.

5.7 Recap

In this chapter, we have proposed a method for aligning a set of overlapping meeting recordings. Our method represents audio as a sequence of hashprints, where the hashprint transformation is learned on-the-fly for the recordings on hand. Using this representation, the recordings are then aligned with an iterative, greedy two-stage alignment algorithm which performs a rough alignment using indexing techniques, followed by a fine-grained alignment based on Hamming distance. Using the ICSI meeting corpus as source material to generate challenging alignment scenarios, our proposed method is able to achieve greater than 99% alignment accuracy at a reasonable error tolerance of 0.1 seconds. The method only requires a few seconds of audio from each channel to learn a robust fingerprint design, and can robustly identify an alignment with 10 seconds of temporal overlap in a typical meeting scenario. One of the greatest benefits of our approach is that it requires little to no system tuning given a new set of recordings to align, since the fingerprint representation is learned on-the-fly. We demonstrated how the robustness of the fingerprint can be improved by increasing the amount of context information or by decreasing the number of fingerprint lookup bits. We have discussed the tradeoffs of changing these two factors and proposed a simple strategy to adjust them in practice.

Chapter 6

Application: Live Song Identification

In the previous chapter we applied hashprints to the problem of aligning unsynchronized meeting recordings. In this chapter we will explore a second application of hashprints: live song identification. We will first introduce the problem statement (section 6.1), discuss relevant prior work (section 6.2), describe the proposed system (section 6.3), show experimental results (section 6.4), perform various analyses of interest (section 6.5), and then summarize several practical takeaway lessons (section 6.6).

6.1 Problem Statement

This chapter investigates the problem of live song identification. A person goes to a live concert, hears a song that he or she likes, and wants to know, "What song is this?" Ideally, the person can simply open an app on his or her cell phone, record a few seconds of the performance, and get an answer. Even if the song is already known, such an app could provide a convenient way for concertgoers to purchase music instantly. While there are several commercially successful apps like Shazam and SoundHound that can identify pre-recorded music playing on the radio, the technology for identifying live music is lagging behind. This chapter offers a step towards bridging that gap.

In order for such an app to be useful, it must return reliable results with minimal latency. Users will not use an app that has unreliable results or an app that makes them wait for more than a few seconds. In our study, we must therefore consider both the reliability of the results and the runtime latency. Accordingly, our goal is the following: given a 6 second cell phone recording of a band's live performance, identify the song with less than 1 second of latency and with results that are significantly better than the previous state-of-the-art.

	Fingerprinting	Cover Song	Live Song ID
Match Similarity	exact match	very fuzzy	somewhat fuzzy
Query Length	short	long	short
Noise Level	noisy	clean	noisy
Previous Work	lots	lots	[Rafii14]

Figure 6.1: Summary of related work. Live song identification is a hybrid of audio fingerprinting and cover song detection in the sense that it inherits the challenges and difficulties of both. Factors that make a problem easier are shown in green, and factors that make a problem more challenging are shown in red.

6.2 Related Work

Live song identification is a hybrid of two well-studied problems: audio fingerprinting and cover song detection.[1] Recall that audio fingerprinting attempts to uniquely identify a segment of audio in a database of clean recordings. This is what Shazam does. Cover song detection attempts to identify cover versions of the same song. Each of these well-studied problems has certain factors that make the problem challenging and certain factors that make the problem easier. These factors are summarized in figure 6.1.

For audio fingerprinting, one major factor that makes the problem challenging is that the queries are short and noisy. Also, audio fingerprinting applications like Shazam often have to run in real-time, so runtime latency can be a big challenge. The factor that makes audio fingerprinting much easier is that such systems typically assume an *exact* match in the underlying source signals, possibly with some simple systematic distortions (like tempo change or room acoustics) or other additive sounds in the environment.

For cover song detection, the factor that makes the problem challenging is that the match is very fuzzy — cover versions can differ in key, tempo, arrangement, and instrumentation, and these differences can make the matching problem much more subjective. One factor that makes cover song detection easier is that the problem is offline, so runtime latency

[1] Audio fingerprinting is also often referred to as audio identification or exact-match audio search. Cover song detection is also often referred to as version identification. See chapter 7 in [60] for a broad overview of both problems.

is less of a factor. Also, cover song systems typically assume that the "query" is a clean studio recording of an entire song, so length of query and other additive sound sources and distortion are typically not an issue.

Live song identification is a hybrid of these two problems in the sense that it inherits the challenges from both. Like audio fingerprinting, the queries are short and noisy, and the system must run in real-time. Like cover song detection, the match is somewhat fuzzy. While the differences between an artist's live performance and studio recording may be less drastic than the difference between two different cover versions, the live song identification problem must nonetheless cope with differences in key, tempo, arrangement, and instrumentation that are typical of live performances. To indicate the degree of difficulty in the match similarity, the entry for live song identification in figure 6.1 is shown in orange (not red).

There has been a lot of previous work in audio fingerprinting and cover song detection. We have already discussed previous work in audio fingerprinting extensively in chapter 1. Like audio fingerprinting, cover song detection has benefited greatly from organized evaluations in the academic community. The MIREX evaluation [17] allowed researchers to compare different approaches on a standardized benchmark, and it spurred a lot of progress on cover song detection [77][23][86][85]. The release of the million song dataset [6] has also allowed researchers to explore cover song retrieval at a large scale [22][40][47][67][5].

There have also been a number of works in audio matching [11][29][50][61]. Audio matching is another hybrid of audio fingerprinting and cover song detection, where the goal is to find segments of audio that are musically similar to a short audio query. There are a few major differences between these approaches and the live song identification scenario considered in this chapter. One difference is that many of these works are offline tasks, where the fragment length is too long for a real-time application (10 to 30 seconds) or runtime latency is not considered. Another difference is that the query fragments are often extracted from clean studio recordings, rather than from noisy cell phone recordings. Yet another significant difference is that these approaches mostly focus on classical music, where the audience is generally very quiet (unlike at a rock concert).

In contrast, live song identification based on short cell phone queries is relatively new and unexplored. One major reason for this is the difficulty of collecting and annotating a suitable data set. Rafii et al. [74] collect a set of cell phone recordings of live concerts for ten popular bands, and they propose a method for song identification based on a binarized representation of the constant Q transform. To the best of our knowledge, this is the only previous work that directly addresses our problem of interest. This previous work will serve as a baseline comparison to our proposed system.

In this chapter, we propose a solution to the live song identification problem that has two main components: (1) the hashprint representation of audio, which was introduced in chapter 3, and (2) a simple, flexible hashprint cross correlation matching algorithm that allows one to trade off accuracy for efficiency in order to accommodate the size of each artist's searchable database. Since we have already discussed the relationship of hashprints to previous work in chapter 3, we will focus here on discussing the relation of the matching algorithm to other approaches.

The proposed search mechanism is not particularly novel or complex, but it is nonetheless useful to compare it to other approaches. Most search mechanisms generally fall into one of two groups: index-based approaches and exhaustive approaches.

Index-based approaches use binary representations to look up matches in a table or index. This lookup can be improved in a variety of ways, including doing lookups on binary codes that are close in Hamming distance [32][45], using probabilistic methods like locality-sensitive hashing (LSH) and min-hash to improve retrieval performance [11][4], or accumulating lots of local fingerprint matches in an efficient data structure such as a histogram of offsets [99][2]. (The histogram of offsets was used in our approach to aligning meeting recordings.) The advantage of index-based approaches is that they offer the ability to scale to large databases. The disadvantage is that performance drops dramatically when moving from an exact-match problem to a nonexact-match problem like live song identification.

Exhaustive approaches simply do an exhaustive search through the database. The most common example of this is to perform a full dynamic time warping (DTW) between the query and all references sequences (e.g. [73][39]). DTW is a way to align two feature sequences with local tempo mismatches using dynamic programming techniques. Many works explore ways to speed up, optimize, or improve exhaustive searches. Some examples include assuming a constant global tempo difference and considering a number of tempo-adjusted queries [61], using a two-pass approach that first does a rough scoring in order to identify a set of promising candidates for a more fine-grained rescoring [25], or using techniques to deduce and skip computations that are unnecessary [63]. The advantage of exhaustive approaches is that they yield better results for nonexact-match problems. The disadvantage is the computational cost of exhaustively searching through the entire database.

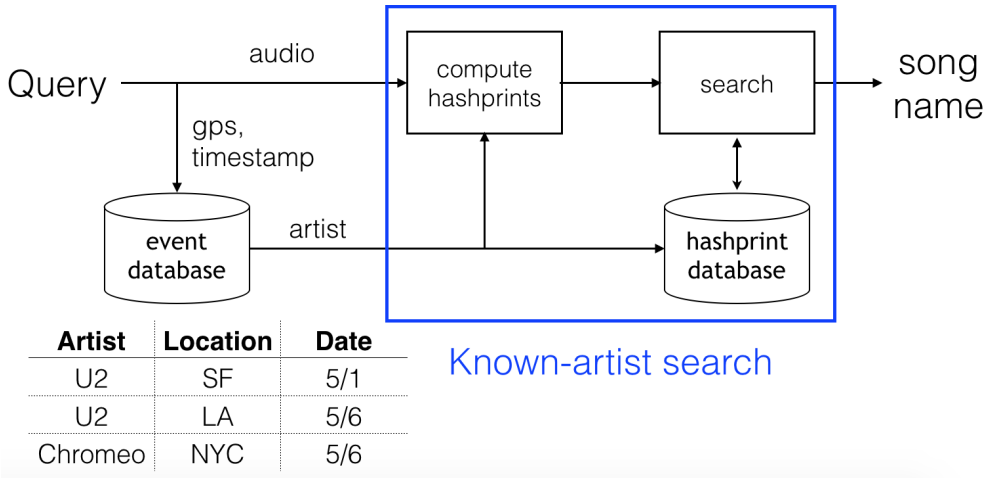
The proposed search algorithm is a very simple exhaustive approach that has one very useful characteristic: we can control the tradeoff between accuracy and computational cost. Provided that the database is not too big, this approach offers the good results that come with an exhaustive search while ensuring that the runtime latency will be kept to an acceptable level. The details of the matching algorithm will be described in the next section.

6.3 System Description

The proposed live song identification system will be described in three parts: the system architecture, the hashprint representation, and the search algorithm.

System Architecture

Figure 6.2 shows the system architecture for the proposed system. When a query is first submitted, the GPS coordinates of the cell phone and the timestamp information are used to associate the query with a concert in order to infer who the artist is. Once the artist has been inferred, the problem is reduced to a known-artist search: we assume we know who the artist is, and we are trying to determine which song is being played. The known-artist



Artist	Location	Date
U2	SF	5/1
U2	LA	5/6
Chromeo	NYC	5/6

Figure 6.2: System architecture of the live song identification system. Using GPS coordinates of the phone, the query is associated with a concert in order to infer who the artist is.

assumption is critical to reduce the search space down to a manageable size. Whereas the space of all possible songs may contain millions of possibilities, the set of songs released by a single musical artist is at most a few hundred.[2] Inferring the artist's identity requires assembling and maintaining a database of up-to-date concert event information. Since our evaluation data comes from concerts that have occurred in the past, we will simply assume that such a database is maintained and that we have this information available. In this chapter we will focus on addressing the known-artist search problem.

The system in Figure 6.2 makes two assumptions. The first assumption is that the artist's concert schedule is available online and can be stored in the database of concert event information. This is necessary to correctly associate the query with a concert. The second assumption is that the artist's studio albums determine the space of possible songs that are performed live. These two assumptions generally hold true for popular bands giving live performances. This system would *not* work, however, with an amateur musician performing at a local restaurant.

Hashprint Representation

We explained how hashprints are computed in chapter 4. Here, we revisit that explanation, filling in each step with details that are specific to this application. The six steps are described below.

1. **Compute spectrogram.** The first step is to compute a constant Q transform (CQT).

[2]This statement generally holds true for the genres of music considered in this work. For other genres such as jazz or classical music, the number of performed works may be larger.

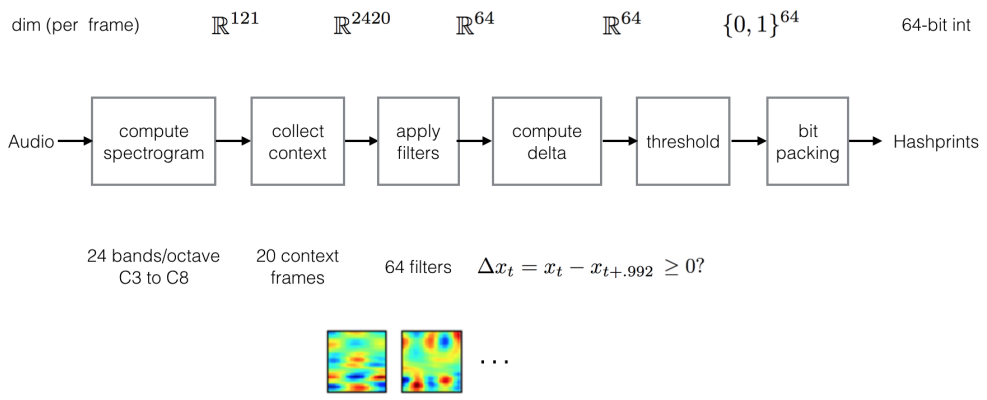


Figure 6.3: Block diagram of hashprint computation for the live song identification scenario.

The CQT is a time-frequency representation of audio that uses a set of logarithmically spaced filters with constant Q factor.[3] One benefit of the CQT is that the filters can be selected to match the pitches of the Western musical scale. Using the CQT thus allows our representation to capture musically meaningful pitch-related information, and it also allows us to efficiently compute pitch-shifted versions of a signal. In our experiments, we used the CQT implementation by Schörkhuber and Klapuri [80]. Similar to the work by Rafii et al. [74], we consider 24 subbands per octave between C3 (130.81 Hz) and C8 (4186.01 Hz). Using a quarter-tone resolution allows us to handle slight tuning differences as well as key changes. To mimic the nonlinear processing of the human auditory system, we take the logarithm of the subband energy values. At the end of this step, we have 121 subband log-energy values every 12.4 ms.

2. **Collect context frames.** When computing the fingerprint at a particular frame, we consider $w = 20$ frames of context. So, at each frame we are working with a vector of dimension $121w = 2420$. In section 6.5, we will explore the effect of w on system performance.

3. **Apply spectro-temporal filters.** At each frame, we apply a set of $N = 64$ spectro-temporal filters in order to generate N real-valued spectro-temporal features. Each spectro-temporal feature is a linear combination of the CQT log-energy values from the current audio frame and its context frames. As pointed out in chapter 4, chroma features are a special case of spectro-temporal features in which the spectro-temporal filter coefficients sum energy across musical octaves. Rather than using these hardcoded filters, however, we use filters that capture the directions of maximum variance, as explained in section 4.3. These filters are learned in an unsupervised manner based on each artist's studio recordings (i.e. the filters are learned per artist). In section 6.5,

[3]The Q factor of a filter is the ratio between the filter's center frequency and its bandwidth.

we will explore the effect of N on system performance. At the end of this third step, we have N features per frame.

4. **Compute deltas.** For each of our N features, we compute the change in the feature value over a time lag T. If the feature value at frame n is given by x_n, then the corresponding delta feature will be $\Delta_n = x_n - x_{n+T}$. In our experiments, we used a value of $T = .992$ seconds, which was determined empirically (see section 6.5).

5. **Apply threshold.** Each of the N delta features is compared to a threshold value of 0, which results in a binary value. These bits represent whether the N features are increasing or decreasing across the time lag T.

6. **Bit packing.** The sixth step is to package the N binary values per frame into a single 64-bit integer. This representation allows us to store hashprints very compactly in memory, and also to compute Hamming distances very efficiently using logical bit-wise operators. The output of this final step is a sequence of hashprints, where each hashprint is represented by a single 64-bit integer.

Figure 6.3 shows a block diagram of the hashprint computation process. The text above the boxes shows the dimension (per frame) at each stage of the process. The text below the boxes provides additional clarifying information. Note that the values shown in figure 6.3 refer to the "default" parameter settings for the hashprint system shown in the results section. We will explore the effect of several of these system parameters on system performance in section 6.5.

Search Algorithm

Figure 6.4 shows the offline and online portions of the known-artist search. During the offline portion of the search, we collect audio from the artist's studio albums, extract hashprints, and store the hashprint sequences into a database. In order to handle the possibility that the live performance may be performed in a different key (or a slightly different tuning) than the original studio track, we also consider pitch-shifted versions of the original studio tracks. We can compute pitch-shifted versions very efficiently by simply shifting the CQT coefficients of the original studio recording. Once the CQT coefficients have been shifted, the hashprints are computed as before by applying steps 2 through 5 (as described in the previous subsection). These steps 2 through 5 are collectively indicated in figure 6.4 as "Hamming embedding," since they take the CQT coefficients and embed them in a Hamming space. In our experiments we consider up to four quartertones above and below the original key. So, for each song in the database, there will be nine hashprint sequences corresponding to the nine different pitch-shifted versions of the song.

During the online portion of the search, the query hashprint sequence is compared to the database to find a match. This online search is done by performing hashprint cross-

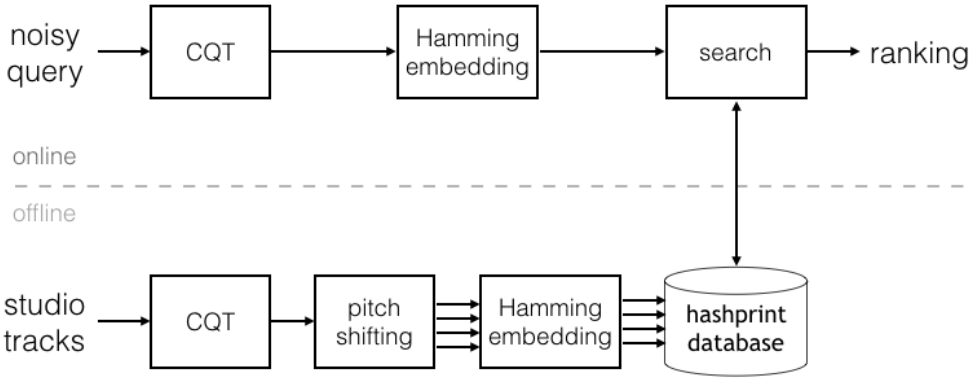


Figure 6.4: Offline and online portions of the known-artist search. During the offline portion, we populate the database with hashprint sequences from the artist's studio tracks and corresponding pitch-shifted versions. During the online portion, we match the query hashprint sequence against the database.

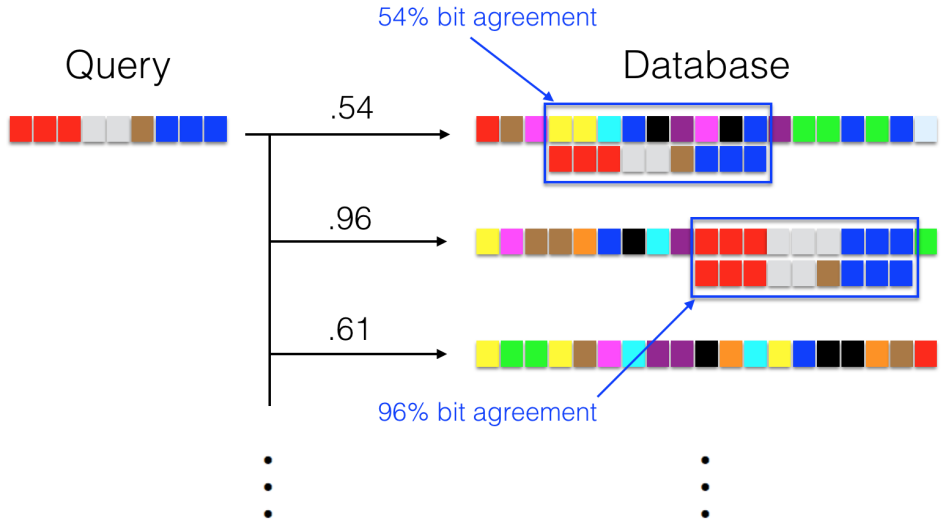


Figure 6.5: An illustration of hashprint cross-correlation matching. For each reference sequence in the database, we determine the offset which maximizes the bit agreement between the query and reference hashprint sequences. This bit agreement is then interpreted as the match score with the entire sequence. Hashprints are shown in the figure as different colored boxes.

correlation matching with downsampling and rescoring. Each of these three components is described below.

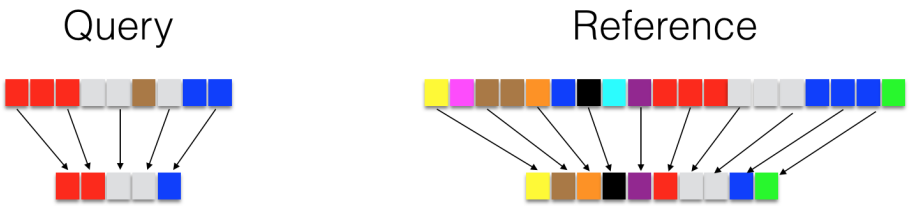


Figure 6.6: An illustration of downsampling a hashprint sequence by a factor of 2. Downsampling by a factor of B yields a factor of B^2 savings in search computation.

Hashprint cross correlation. For each hashprint sequence in the database, we determine the offset that maximizes the bit agreement rate between the query hashprint sequence and the corresponding portion of the reference hashprint sequence. This maximum bit agreement rate is used as the match score with the whole sequence. The maximum match score among the various pitch-shifted versions of a song is taken as the aggregate match score for the song. The songs in the database are then ranked by their aggregate match scores. Figure 6.5 shows an illustration of this matching process. Note that this approach is identical to a time-domain cross correlation approach except that the correlation is computed on sequences of hashprints rather than on sequences of time-domain samples. Note also that this approach assumes an approximately 1-to-1 tempo correspondence between the query and the studio album. We will investigate the validity of this assumption and also compare our results with a dynamic time warping approach in section 6.5.

Downsampling. In order to speed up the online search, we downsample both the query and reference hashprint sequences by a factor B. So, for example, when $B = 2$ we only consider every other hashprint when computing bit agreement scores. Downsampling by a factor B thus reduces the amount of search computation by a factor of B^2. Our hope is that we can reduce the search time without affecting the accuracy too much. The effect of this downsampling will be investigated in section 6.5. Figure 6.6 shows an illustration of downsampling by a factor of 2.

Rescoring. After sorting the reference sequences by their rough (downsampled) match scores, we can rescore the top L sequences using the full hashprint sequences without downsampling. We can then resort these top L sequences by their fine-grained match scores. Figure 6.7 shows an illustration of the rescoring process. The idea of rescoring is to keep the computation savings of a downsampling approach while retaining the reliability and performance of an exhaustive match without downsampling. The effect of this rescoring will be investigated in section 6.5.

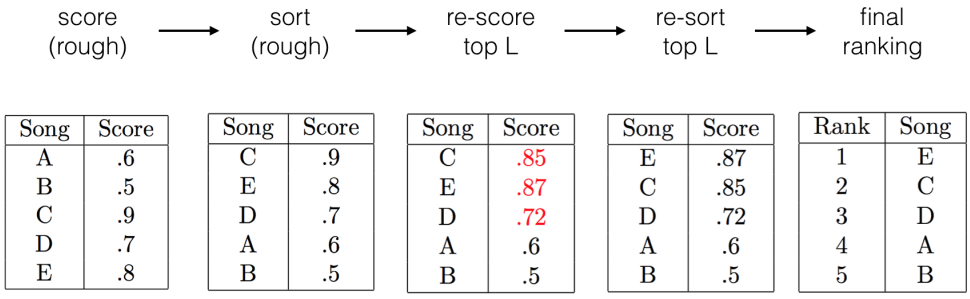


Figure 6.7: An illustration of rescoring. The downsampled sequences in the database are scored and sorted. The top L sequences are then re-scored without downsampling and re-sorted. This toy example shows this process when the database has 5 sequences and $L = 3$.

Artist Name	Artist ID	Genre	# Tracks
Big K.R.I.T.	Big	hip hop	71
Chromeo	Chr	electro-funk	44
Death Cab for Cutie	Dea	indie rock	87
Foo Fighters	Foo	hard rock	86
Kanye West	Kan	hip hop	92
Maroon 5	Mar	pop rock	66
One Direction	One	pop boy band	60
Taylor Swift	Tay	country, pop	71
T.I.	TI	hip hop	154
Tom Petty	Tom	rock, blues rock	193

Table 6.1: Overview of the Gracenote live song identification data. The database contains full tracks taken from artists' studio albums. The queries consist of 1000 6-second cell phone recordings of live performances (100 queries per artist).

6.4 Evaluation

We will describe the evaluation of the proposed system in three parts: the data, the evaluation metric, and the experimental results.

Data

We evaluated the proposed system on the Gracenote live song identification benchmark. This is a proprietary data set used for internal benchmarking purposes at Gracenote.[4] The data comes from 10 different musical artists spanning a range of genres including pop, rock,

[4]Thanks to Zafar Rafii and Markus Cremer at Gracenote for generously providing the data set.

country, and rap. Table 6.1 shows the 10 musical artists, along with a brief description and the number of studio recordings per artist. For each artist, there is a clean database and a set of noisy queries. The database consists of full audio tracks taken from the artist's released studio albums. The noisy queries are short cell phone recordings of live performances and were prepared in the following manner. Ten Youtube videos of live performances were downloaded for each artist. These ten videos came from 10 different songs and were all recorded on consumer cell phones. The videos were manually pre-processed to cut out non-music material at the beginning and end of the video (e.g. applause, introducing the song, etc). Finally, ten 6-second segments evenly spaced throughout the recording were extracted. There are thus 100 queries per artist, and 1000 queries in total.

Evaluation Metric

The evaluation metric we use is the mean reciprocal rank (MRR) [98], which is given by the following formula:

$$MRR = \frac{1}{N}\sum_{i=1}^{N}\frac{1}{R_i}$$

Here, N refers to the total number of queries in the benchmark, and R_i refers to the rank of the correct item for the i^{th} query. In cases where there is more than one correct item (i.e. the artist has more than one studio recording of the same song), the best rank is used. Note that MRR ranges between 0 and 1, where a low value close to zero corresponds to poor performance and 1 corresponds to perfect performance. Higher MRR is better.

Results

Figure 6.8 shows the performance of the proposed system along with five other baseline systems. Figure 6.9 shows the same results broken down by artist. The first baseline system ('ellis07' [23]) is an open-source cover song detection system based on beat-level chroma features. The second baseline ('hydraSVM' [77]) is another open-source cover song detection system that performs a system-level combination of three different cover song detection approaches [23][24][84]. The third baseline ('shazam' [99]) is an open-source implementation [21] of the well-known Shazam algorithm. The fourth baseline ('panako' [89]) is an open-source audio fingerprinting system that is designed to handle pitch-shifting and tempo changes. The fifth baseline ('rafii' [74]) is the previously proposed live song identification system at Gracenote. The two rightmost bars ('hprint(1)', 'hprint(3)') show the performance of the proposed system at downsampling rates of 1 and 3, respectively. There are four things to notice about the results in figures 6.8 and 6.9.

First, the audio fingerprinting and cover song detection baselines perform poorly. The first four baselines suggest that existing cover song detection and audio fingerprinting approaches may not be suitable solutions to the live song identification problem. Audio fingerprinting approaches typically assume that the underlying source signal is identical, and

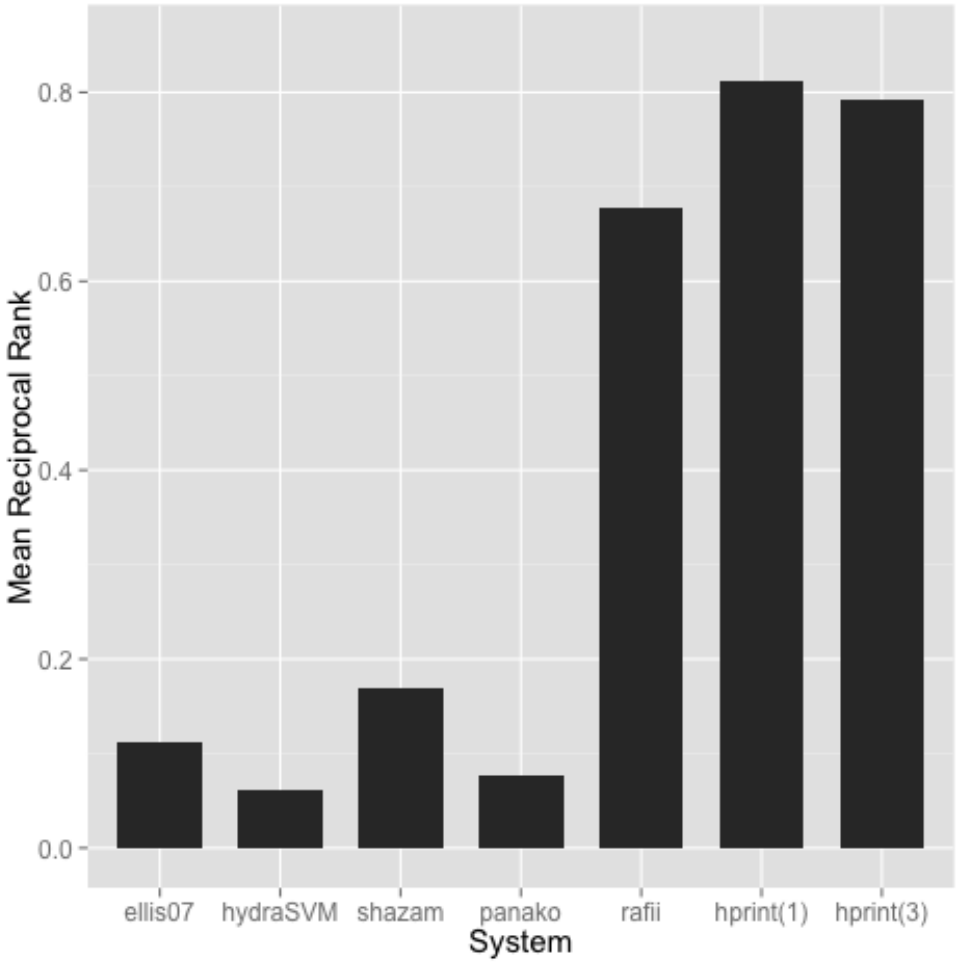


Figure 6.8: Performance of proposed hashprint system compared to five other baseline systems. The number in parenthesis indicates the downsampling factor of the hashprint cross-correlation search.

may not be able to cope with the variations in live performances. (Note that the Shazam algorithm performs okay for some artists like Chromeo that use digital sounds in live performances that may be identical to the original studio recordings.) On the other hand, cover song detection systems typically assume that an entire clean studio recording is available, and may not be able to cope with short, noisy queries.

Second, the proposed system significantly improves upon the previous state-of-the-art. Comparing the three rightmost systems, we see that the two versions of the proposed system improve the MRR from .68 ('rafii') to .81 ('hprint(1)') and .79 ('hprint(3)'). Given the reciprocal nature of the evaluation metric, this amounts to a significant improvement in performance.

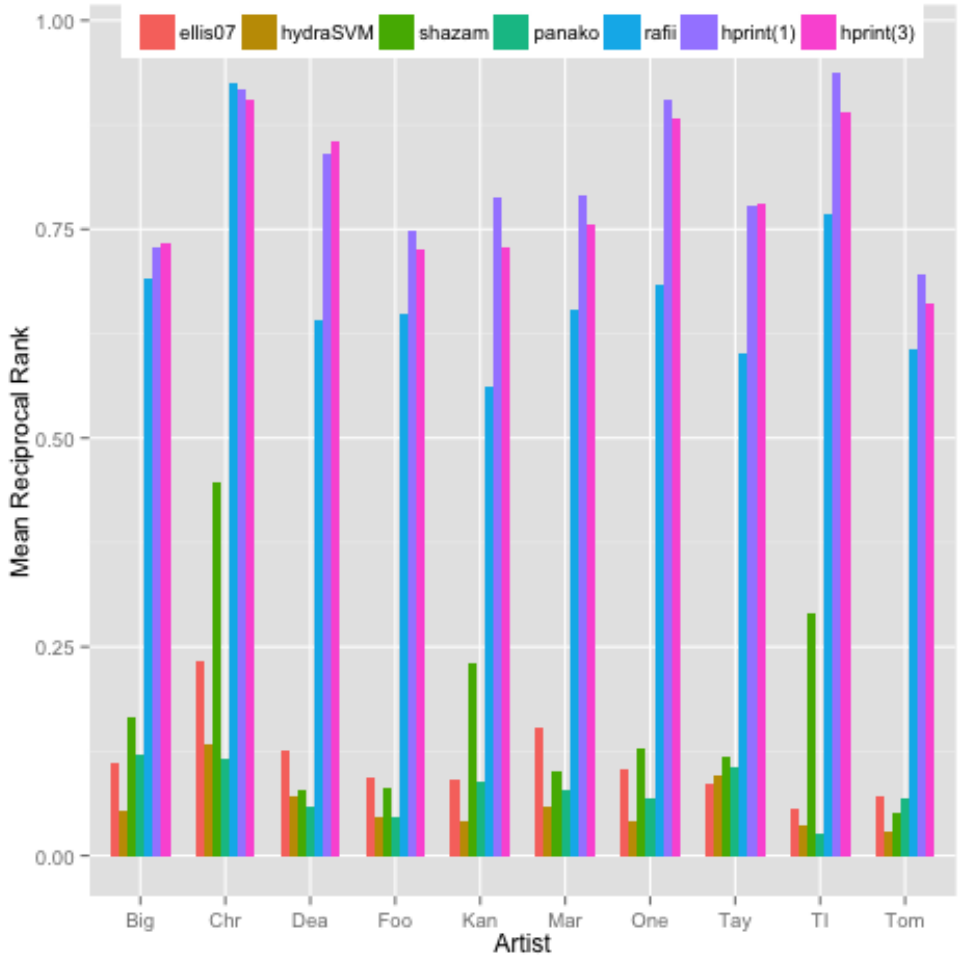


Figure 6.9: Breakdown of results by artist. The first three letters of the artist's name is shown at bottom.

Third, large computational savings can be achieved for a modest sacrifice in accuracy. Comparing the two proposed systems, we see that it is possible to reduce the search computation by a factor of $B^2 = 9$ while only reducing the MRR from .81 to .79. We will investigate the tradeoff between accuracy and search time in section **??**.

Fourth, performance varies by artist. Looking at figure 6.9, we can see that the performance of the various systems varies a lot from artist to artist. The systems generally agree on which artists are "hard" and which are "easy." One big factor that affects these results is how similar or different an artist's live performance is compared to the original studio recording. Another big factor is how many studio tracks the artist has. Note that the artists with highest and lowest scores (Chromeo and Tom Petty, respectively) correspond to the artists with the smallest and largest databases.

6.5 Analysis

In this section, we will investigate seven different questions of interest. These analyses will give us a deeper intuition and understanding of the various factors that affect system performance.

Effect of Tempo Mismatch

The first question of interest is, "How much do tempo mismatches affect our results?" As mentioned earlier, the hashprint cross correlation matching approach assumes that the live performance is at approximately the same tempo as the original studio recording. Clearly, this assumption is violated if the artist performs a song faster or slower than the studio version. We would like to measure how reasonable or unreasonable this assumption is.

In order to answer this question, we compared the hashprint cross correlation search algorithm with subsequence DTW. DTW is a common way to align two feature sequences with local tempo differences, and subsequence DTW is a variant of DTW that allows one sequence to begin at any offset in the other sequence. A brief overview of the subsequence DTW algorithm is given below, and the reader is referred to [60] (chapter 3) for a more detailed explanation. For each reference sequence in the database, we compute a Hamming distance cost matrix between the query and reference hashprint sequences, and then we determine the alignment path with lowest score using dynamic programming techniques. We use the set $\{(1,1),(1,2),(2,1)\}$ as possible step sizes, which allows for tempo differences up to a factor of 2. We then use the normalized alignment path score as a match score for the reference sequence. Comparing the hashprint cross-correlation and subsequence DTW approaches will indicate how important it is to handle local tempo differences.

Figure 6.10 shows the comparison between the hashprint cross correlation and subsequence DTW approaches. Since the amount of temporal context described by each hashprint can affect its ability to absorb timing mismatches, we also show this comparison across a range of context values w. There are three things to notice about Figure 6.10. First, hashprint cross correlation performs better than subsequence DTW. Across a wide range of context values, the hashprint cross correlation approach shows consistently better results than subsequence DTW. This indicates that the DTW algorithm is providing too many degrees of freedom in the matching process, which ends up hurting system performance. Second, more context helps up to a certain point. For both matching algorithms, using more context frames (up to 20) improves results significantly. Third, context helps hashprint cross-correlation more than it helps subsequence DTW. Note that the MRR scores using subsequence DTW range from .75 to .78, while the MRR scores using hashprint cross correlation range from .76 to .81. With the subsequence DTW approach, tempo mismatches can be handled by the matching algorithm even if the context of each hashprint is small. But with the hashprint cross correlation approach, tempo mismatches *cannot* be handled by the matching algorithm. In this case, the only factor that can mitigate the effect of tempo

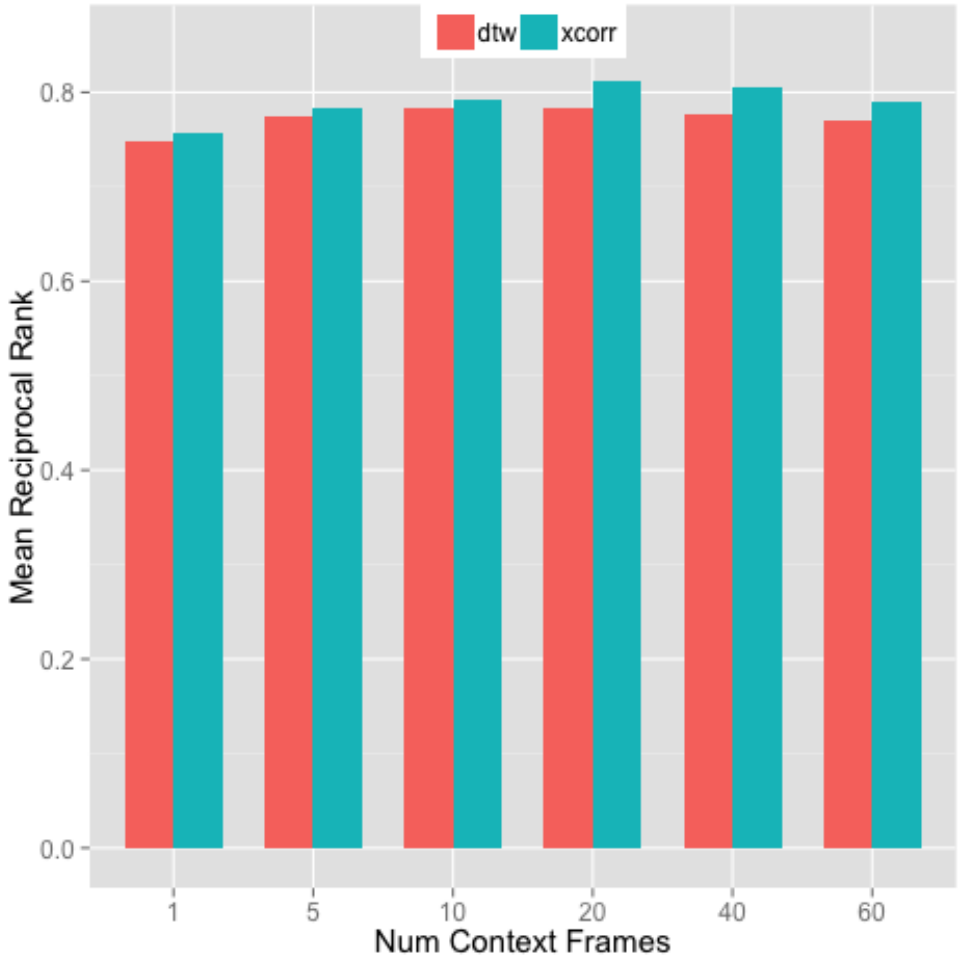


Figure 6.10: Comparison of DTW and hashprint cross correlation approaches. The horizontal axis refers to the number of context frames spanned by each hashprint.

mismatch is the amount of temporal context described by each hashprint, so context has a proportionally larger impact on system performance.

In summary, the tempo mismatches are small enough that across a short 6-second segment, we can approximately assume that the tempos are equal. Furthermore, increasing the amount of context in the hashprint representation allows individual hashprints to absorb slight misalignments that result from tempo mismatch. Of course, these conclusions can only generalize to the extent that these 10 musical artists are representative of other live song identification scenarios. For this data set, however, the conclusion is clear: hashprint cross correlation wins.

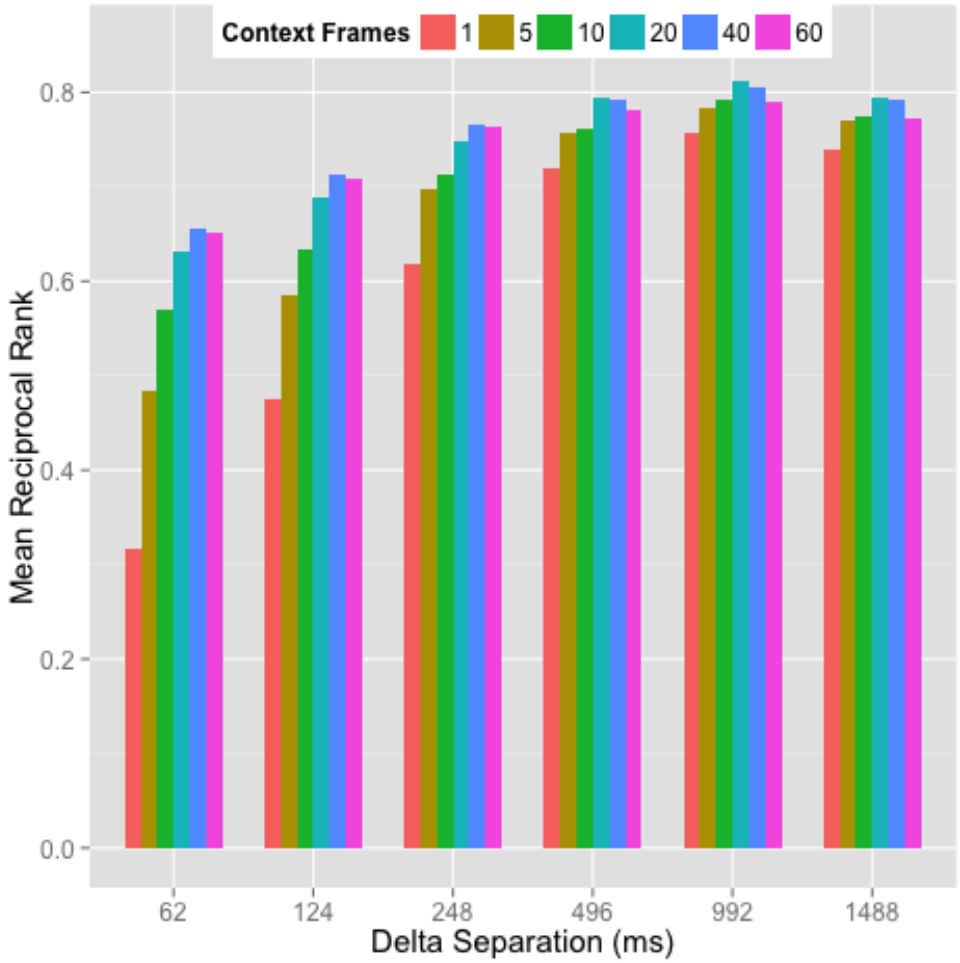


Figure 6.11: Effect of hashprint context window length and delta separation value on system performance.

Effect of Context & Delta

The second question of interest to us is, "How do context and delta separation affect system performance?" To answer this question, we ran experiments across a range of context values w and delta separation values T. For these experiments, we used hashprint cross correlation matching with a downsampling rate of 1 (i.e. no downsampling).

Figure 6.11 shows the effect of context and delta separation on system performance. We see that using more context helps up to a certain point, as discussed previously. We also see that using greater delta separation (up to 992 ms) also improves results significantly. For very small values of T, the delta spectro-temporal feature is the difference between two spectro-temporal features that are located close together in time and thus highly correlated. Because the difference between two highly correlated features is small, the variance of the resulting

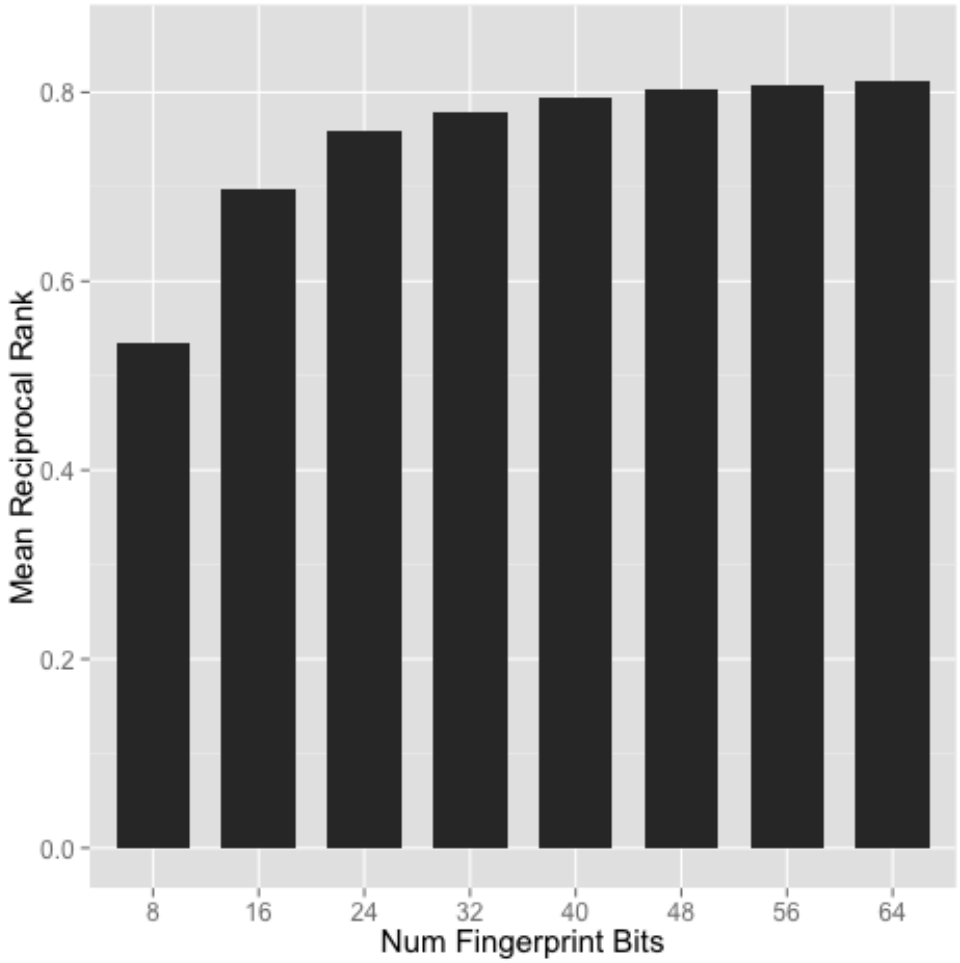


Figure 6.12: Effect of the number of hashprint bits on system performance.

delta features will be small, resulting in bits that are not robust. For large values of T, the delta spectro-temporal feature will be the difference between two independent features, which effectively yields double the variance. As T increases, however, we also effectively lose T seconds of data from our 6 second query. For example, when $T = 2$ seconds, we effectively have 4 seconds of time-varying data. Thus, there is a tradeoff in selecting the optimal delta separation value T. We can see from Figure 6.11 that $T = 992$ ms and $w = 20$ context frames yield the best results. These settings are used in all of the reported results in this chapter, unless otherwise noted.

Effect of Number of Bits

The third question of interest to us is, "How does the number of hashprint bits affect the results?" Figure 6.12 shows the effect of varying the number of bits in the hashprint representation. For these experiments, we used a hashprint cross correlation matching with a downsampling factor of 1. As we increase the number of bits in the hashprint representation, the system performance improves in an asymptotic fashion. By the time we reach 64 bits, the system performance has largely leveled off. We would expect only marginal improvement in performance if we went beyond 64 bits. This is quite convenient for our system design, since going beyond 64 bits would require twice the storage and computation on a 64-bit machine. We use a 64-bit hashprint representation for all other results reported in this paper.

Effect of Learning

The fourth question of interest to us is, "How much does the unsupervised learning actually help?" It could be the case, for example, that the benefit in our system performance simply comes from using more context frames, rather than from learning the filter coefficients. In order to answer this question, we repeated the 'hprint(1)' experiment in Figure 6.8 with one major change: instead of using the learned filters, we use filters with random coefficients. Using random coefficients corresponds to a locality sensitive hashing (LSH) approach, which was described in section 4.5. When we ran this experiment, the MRR of the system falls from .81 ('hprint(1)') to .65 (LSH-based approach).

There are two things to note about this result. First, using random coefficients still performs quite well. This LSH-based approach (MRR .65) performs almost as well as the previous state-of-the-art system proposed by Rafii et al. [74] (MRR .68). So, even if the unsupervised learning was omitted, the resulting system would still perform relatively well. The effectiveness of the LSH technique has been validated by its widespread adoption in practice. Second, the learning helps a lot. The structural elements of the hashprint approach (e.g. spectro-temporal filters, amount of context, delta separation, etc) yields a performance that is on par with the previous state-of-the-art, but the unsupervised learning of filter coefficients is what provides the significant improvement beyond previous results. The unsupervised learning does indeed help a lot.

Effect of Downsampling & Rescoring

The fifth question of interest to us is, "How do downsampling and rescoring affect system performance?" The purpose of introducing downsampling and rescoring is to reduce the search latency, so we must consider both the accuracy and the efficiency of the system. Figure 6.13 shows the effect of downsampling and rescoring on system accuracy. The five groups of bars show the MRR of the system as the downsampling rate increases from 1 to 5. Each pair of bars compares the performance with and without rescoring. Table 6.2 shows the effect of downsampling and rescoring on system efficiency. To quantify the efficiency of

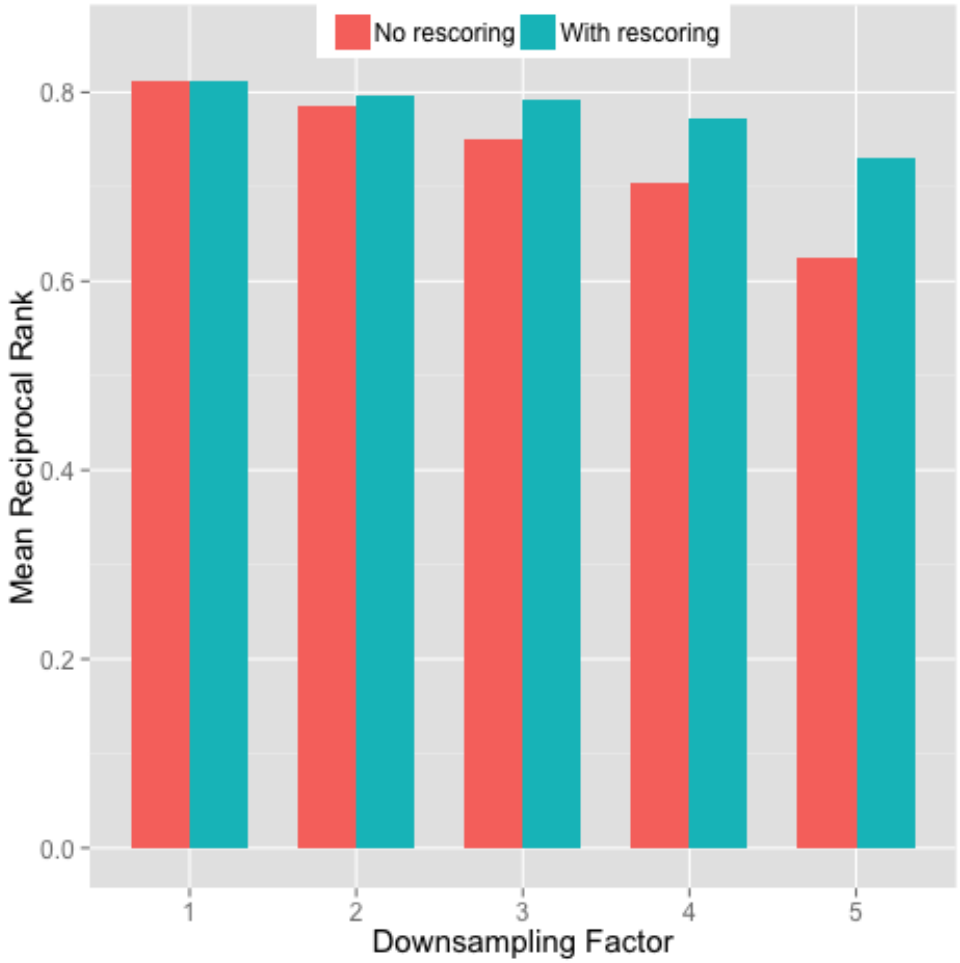


Figure 6.13: Effect of downsampling and rescoring on system performance.

the system, we measure the average amount of time it takes to process each 6 second query. We ran the experiments on a single core of a 2.2 GHz Intel Xeon processor. The average amount of time required to compute the CQT and search the database is shown in columns 3 and 4, and the total average runtime is shown in column 5.

There are three things to notice about the results in Figure 6.13 and Table 6.2. First, downsampling provides a tradeoff between accuracy and efficiency. As we increase the downsampling rate, the MRR gets worse (Figure 6.13) and the average runtime gets better (Table 6.2). We can choose a downsampling rate to achieve the desired tradeoff. Second, rescoring significantly improves MRR with low impact on runtime. Comparing the two bars in each group in Figure 6.13, we can see that rescoring helps to recoup a lot of the MRR that is lost as a result of downsampling. At the same time, this rescoring has very little impact on average runtime. Comparing the runtime of systems with and without rescoring in Table

Downsample	Rescoring	CQT	Search	Total
1	no	.51	2.87	3.43
2	no	.50	.68	1.23
3	no	.49	.31	.85
4	no	.52	.18	.74
5	no	.49	.12	.66
1	yes	.53	2.90	3.48
2	yes	.50	.72	1.26
3	yes	.51	.35	.90
4	yes	.50	.21	.76
5	yes	.49	.15	.69

Table 6.2: Effect of downsampling and rescoring on average processing time per query. Columns 3 and 4 show average time in seconds required to compute the CQT and perform the search, respectively. Column 5 shows the average total processing time per query. Experiments were run on a single core of a 2.2 GHz Intel Xeon processor.

6.2, for example, we see that rescoring only adds 2 to 5 ms to the total runtime. Third, we can substantially reduce the runtime with only a modest sacrifice to MRR. With a downsampling rate of 3 with rescoring, for example, we can reduce the runtime from 3.48 seconds to 0.90 seconds while only decreasing the MRR from .81 to .79. This is the 'hprint(3)' system shown at the far right of Figure 6.8. Note that the system proposed by Rafii et al. [74] had a (self-reported) average runtime of 10 seconds per query, so our proposed system may offer substantial savings in search computation.

One other important factor to mention is the cost of computing the CQT. We can see from Table 6.2 that the CQT is a fixed cost to the system, regardless of the downsampling rate. Downsampling can reduce the runtime of the search algorithm, but it can only asymptotically reduce the total runtime down to this fixed cost. In a real commercial system, the CQT could be computed on the query in a streaming manner, so that the latency experienced by the user effectively depends only on the search algorithm. Such an implementation, however, is beyond the scope of this work.

Effect of Database Size

The sixth question of interest to us is, "How does the database size affect system performance?" In order to answer this question, we ran a set of controlled experiments in which we artificially fixed the size of the database (measured by the number of songs). We considered a range of database sizes that would be realistic for an artist of the given genres. (Note that a very established musician like Tom Petty has about 200 songs.) When the fixed database size is less than the artist's actual database size, we removed songs from the database to

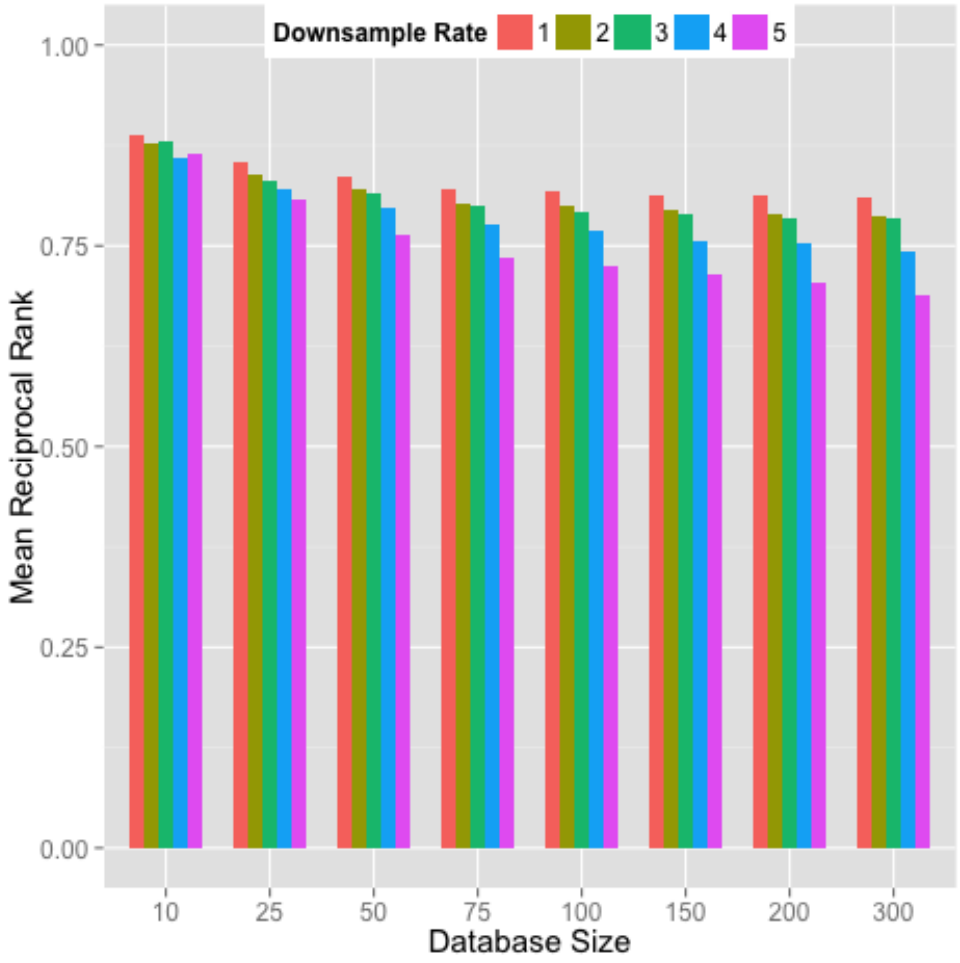


Figure 6.14: Effect of database size on system performance.

achieve the desired size. When the fixed database size is greater than the artist's actual database size, we padded the database with randomly selected songs from the other musical artists in order to achieve the desired size. Since the songs from other artists may result in a database containing more variation in style and genre than is typical within the artist's own repertoire, these results should be interpreted with caution. Nonetheless, such an analysis can provide a rough indicator of how database size would affect results.

Figure 6.14 shows the MRR of the system across different database sizes and different downsampling rates. Figure 6.15 shows the corresponding average runtimes. Note that the database size here refers to the number of original studio recordings, so the actual number of pitch-shifted reference sequences in the database will be nine times greater. There are three things we can observe about the results in Figures 6.14 and 6.15.

First, MRR decreases asymptotically in the database size. As database size increases

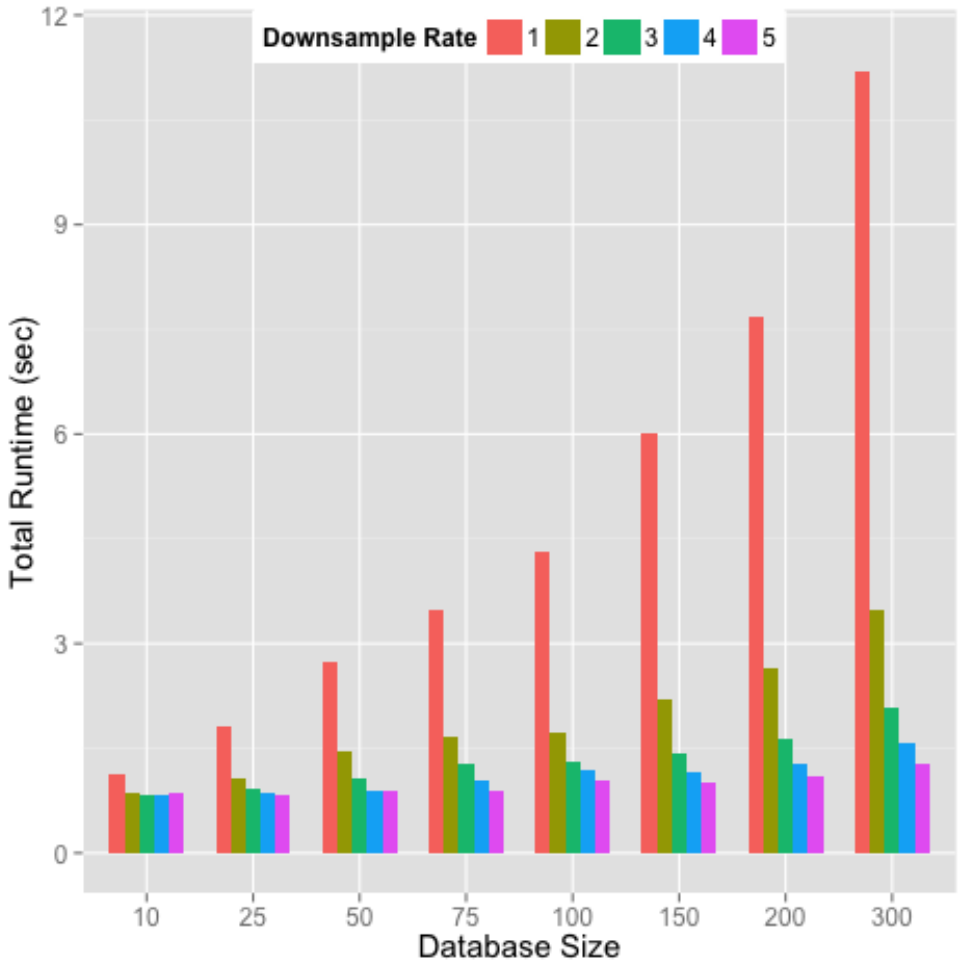


Figure 6.15: Effect of database size on average processing time per query.

beyond 75, there is only a marginal decline in MRR. For example, with a downsampling factor of 1, the MRR falls from .89 to .82 as the database size increases from 10 to 75, but it only falls from .82 to .81 as the database size increases from 75 to 300. Similarly, with a downsampling factor of 3, the MRR falls from .88 to .80 as the database size increases from 10 to 75, but it only falls from .80 to .78 as the database size increases from 75 to 300.

Second, the runtime increases linearly in the database size. (Note that the database sizes shown in the figures are not spaced linearly.) Since the hashprint cross correlation search is doing a linear scan across the database, the search time will be proportional to the size of the database. The total runtime thus consists of fixed costs (such as computing the CQT) and a variable cost that is proportional to the database size. Though actual runtime will of course depend on memory usage and CPU load, a rough guideline is that there are about .6 seconds of fixed costs and each hashprint sequence takes about 4 ms to score (without

downsampling).

Third, the downsampling rate can be selected to stay below a maximum acceptable runtime latency. One advantage of downsampling is that we can control the tradeoff between accuracy and efficiency in an artist-specific way. For artists with a small database — where latency is not an issue — we can use a downsampling rate of 1 to maximize the reliability of the results. For artists with a large database — where latency will be an issue — we can use a higher downsampling rate to guarantee that the average latency stays below an acceptable threshold. In our experiments, for example, we can guarantee an average MRR of .75 and average runtime latency of 1.3 seconds for databases up to size 200, and we can guarantee an average MRR of .74 and average runtime latency of 1.6 seconds for databases up to size 300. The downsampling rate can thus be tailored to each artist to achieve the desired tradeoff between accuracy and efficiency.

Filters

The seventh question of interest to us is, "What do the learned filters look like?" This analysis can help us gain a deeper intuition about what type of information the hashprints are capturing. Figures 6.16 and 6.17 show the top 64 learned filters for two different musical artists, Big K.R.I.T. and Taylor Swift. The filters are arranged first from left to right, and then from top to bottom. The filters cover the frequency range C3 (130.81 Hz) to C8 (4186.01 Hz) and span .372 seconds.[5] There are four things to notice about the filters in Figures 6.16 and 6.17.

First, the filters contain both temporal and spectral modulations. Some filters primarily capture modulations along the temporal dimension, such as filters 3, 4, and 5 on the first row of figure 6.16 that contain vertical bands. Some filters primarily capture modulations along the spectral dimension, such as the filters on row 2 of figure 6.17 that contain many horizontal bands. Other filters capture both temporal and spectral modulations, such as filters 15 and 16 in figure 6.16, which seem to capture primarily temporal modulations in the higher frequencies and primarily spectral modulations in the lower frequencies. The important thing to point out is that both types of modulations are important: if the hashprint representation were constrained to describing only a single audio frame, for example, it would be unable to characterize an important aspect of the signal.

Second, the filters capture both broad and fine spectral detail. Some filters describe the broad shape of the spectrum, such as filters 2 and 6 in figure 6.16. Other filters capture fine spectral details such as filters 22, 23, and 24 in the same figure. Often feature representations like chroma or MFCCs tend to focus on either fine spectral structure or broad spectral shape, but here we see that the hashprints are able to capture both types of information in a simple, unified framework.

[5]Note that here we use a slightly larger amount of context ($w = 30$) to make the filters easier to visualize. We can see from figure **??** that using anywhere between 20 and 40 context frames yields roughly the same performance.

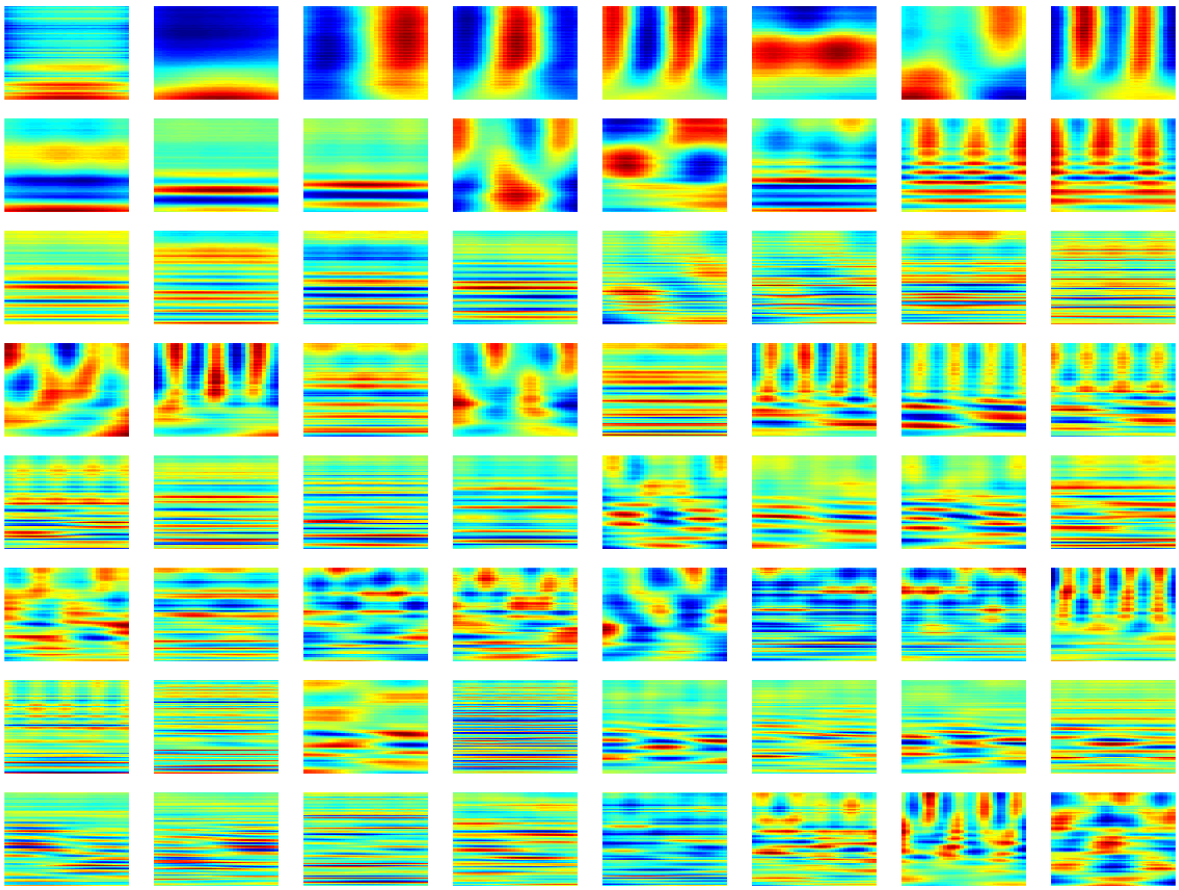


Figure 6.16: Top 64 learned filters for Big K.R.I.T. The filters are arranged first from left to right, and then from top to bottom. Each filter spans .372 seconds and covers a frequency range from C3 to C8.

Third, there is a progression from lower modulation frequencies to higher modulation frequencies. When we look at filters 3, 4, 5, and 8 on the first row of figure 6.16, we can see a progression from lower temporal modulation frequencies to higher and higher modulation frequencies. We see a similar progression in filters 2, 6, 9, 10, and 11 in figure 6.16 for spectral modulation frequencies. It appears that lower modulation frequencies are more useful in the sense that they yield features with greater variance.

Fourth, the filters are artist-specific. When we compare the learned filters for Big K.R.I.T. with the learned filters for Taylor Swift, we see that the first four filters are quite similar. After these first four, however, the filters begin diverging and reflecting the unique characteristics of each artist's music. We notice, for example, that many more of the filters for Big K.R.I.T. emphasize temporal modulations, perhaps a reflection of the fact that rap music tends to be more percussion and rhythm-based. In contrast, the filters for Taylor Swift seem

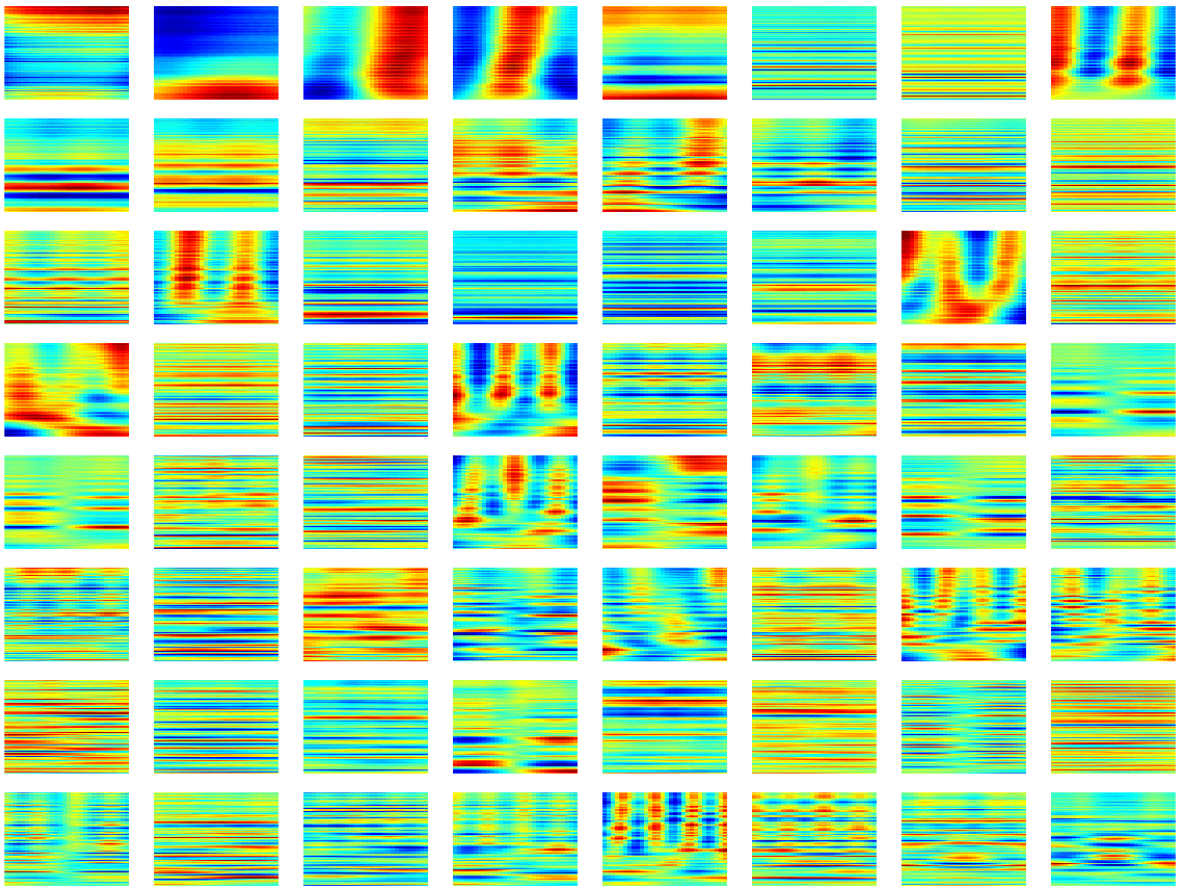


Figure 6.17: Top 64 learned filters for Taylor Swift.

to primarily capture pitch-related information, which perhaps reflects the fact that country and pop music tends to be more harmony-based. The fact that the learned filters diverge so quickly is also an argument for using a representation that is adaptive to each artist. For a known-artist search, it may be more advantageous to use a highly adaptive representation that is specific to each artist, rather than using a single unified representation based on a huge training set.

6.6 Takeaway Lessons

In this section we discuss three practical takeaway lessons from our experimental findings. These take into account what we have observed and learned in the context of both live song identification (chapter 6) and aligning meeting recordings (chapter 5).

Takeaway Lesson #1: Hashprints can be useful for both exact match and similarity

match. We have seen how hashprints have led to state-of-the-art performance on two different tasks. In chapter 5, we showed how hashprints outperform five other fingerprint representations in aligning unsynchronized meeting recordings. This is an exact match scenario in which we use hashprint values to look up matching occurrences in a reverse index. In this chapter, we showed how hashprints significantly improve upon the previous state-of-the-art in live song identification. This is a nonexact match scenario in which we use the Hamming distance between hashprints as a measure of similarity. Though they are used in very different ways in chapters 5 and 6, the hashprint framework has proven to be useful in both exact and nonexact audio search and retrieval.

Takeaway Lesson #2: The hashprint framework can adapt to a wide variety of audio signals. In the aligning meeting recordings scenario, we observe that hashprints are able to effectively characterize speech signals for the purpose of exact matching. In the live song identification scenario, we observe that hashprints are able to effectively characterize music signals for the purpose of similarity matching. When we compare the learned spectro-temporal filters in the meeting recordings scenario (figure 5.11) with the filters in the live song identification scenario (figures 6.16 and 6.17), we can see how the filters reflect the unique characteristics of the underlying signals. For example, many of the learned filters in the live song identification scenario have lots of very thin horizontal bands, suggesting that the filters are capturing pitch-related information in the musical signals. In contrast, we do not see such structures in the learned filters from the meeting recordings scenario. Furthermore, the filters are able to reflect the unique characteristics of each musical artist, as discussed in section 6.5. The hashprints are able to capture broad spectral shape as well as fine spectral structure, temporal modulations as well as spectral modulations, and combinations of all such varieties and flavors. The effectiveness of the hashprint framework on these two very different application scenarios suggests that it can adapt to a wide variety of audio signals.

Takeaway Lesson #3: The hashprint framework can be very easily tuned to new applications. Comparing the hashprints in the meeting recordings scenario (figure 5.2) and the live song identification scenario (figure 6.3), we can see that there are only four differences: the time-frequency representation (mel spectrogram vs CQT), the number of context frames (32 vs 20), the number of filters (16 vs 64), and the delta separation value (.5s vs .992s). In general, this is a very small number of "knobs" that need to be tuned, even as we make a fairly drastic switch from speech signals to music signals. These knobs can also be set to reasonable default settings based on a few simple guidelines (see next paragraph). In contrast, a researcher who wants to learn a representation using a DNN architecture faces a bewildering array of choices including the number of hidden layers, the size of each layer, the type of output nonlinearity, the initialization and training schemes, convolutional vs recurrent, etc. If a DNN-based binary representation is a nuclear bomb, hashprints are a Swiss army knife. These two approaches occupy very different niches. For tasks in which we are willing to invest a significant amount of time, energy, computation, setup, and infrastructure, DNN-based representations are the much better tool of choice. But for many practical tasks in which we have little or no data and would like a no-hassle, easily tuned, easy-to-use

representation that just works reasonably well, hashprints may be our friend.

Here are several simple guidelines for setting the "knobs" on a hashprint representation. Of course, the optimal settings will depend heavily on the application at hand. Nevertheless, it is useful to future researchers to present a few rough guidelines that can act as a starting point.

- Spectrogram. If the signal is speech, use a mel spectrogram. If the signal is music, use a CQT. If in doubt, use a CQT – it is more computationally expensive, but it allows the hashprints to consider both broad and fine spectral structure.
- Context frames. Context frames help a lot up to a certain point (sections 6.5 and 5.5), and increasing beyond that either helps or hurts only marginally. A reasonable default setting to start with is 32 context frames.
- Number of filters. If the task is an exact match using indexing techniques (like aligning meeting recordings), more bits will not correspond to better performance. In this case, the main tradeoff is computation for accuracy – fewer bits will require sifting through more false matches, but will yield better performance. 16 bits is a reasonable starting point for such tasks. If the task is a nonexact match (like live song identification), more bits will be better. In this case, the best selection is to use the most number of bits that can fit into a single integer on the machine. This will be 64 bits or 32 bits, depending on the computer architecture.
- Delta separation. The ideal delta separation value is the minimum lag T that ensures that the signal has become decorrelated with itself. This can be set based on a priori assumptions (e.g. the average speaking rate when dealing with speech signals) or computed empirically by measuring autocorrelation. For typical music and speech signals, $T = 1$ second is a reasonable setting.

6.7 Recap

In this chapter we have introduced a method for identifying live performances from popular bands. The system uses a hashprint representation of audio based on spectro-temporal filters that are tailored to each artist's music. The matching is done using a hashprint cross correlation matching algorithm that can be tuned to achieve a desired runtime latency. We evaluated the performance of the proposed system on the Gracenote live song identification benchmark, and we show that the proposed system improves the mean reciprocal rank of the previous state-of-the-art from .68 to .79, while simultaneously reducing the runtime latency from 10 seconds down to 0.9 seconds. We also conduct extensive analyses to understand the capabilities and limitations of the hashprint representation as well as the search mechanism.

Chapter 7

Conclusion & Future Work

We have laid a foundation of theory, introduced audio hashprints, and seen how hashprints can be used in two different application scenarios. In this final chapter, we will summarize the main concepts we have learned (section 7.1) and then introduce the three avenues of future work (sections 7.2, 7.3, and 7.4).

7.1 Summary

Figure 7.1 is copied from chapter 1 and encapsulates the structure of this thesis. We first lay down a foundation of theory by considering two theoretical questions of interest. We then introduce a binary representation of audio called audio hashprints, and then apply this hashprint representation to two different applications. Below, we give a concise summary of what we have learned in each chapter.

- **What makes a good hash key?** *Maximizing useful information rate.* We have proposed an information-theoretic metric that measures how good or bad a hash key is. The metric is called useful information rate because it represents the average amount of correct information that each hash key communicates. We discuss the intuition behind this metric and also present some empirical evidence to show that it correlates with system-level performance.

- **What makes a good hash bit?** *Balanced, uncorrelated, high variance.* A good hash bit is one that helps the hash key to have high entropy and high accuracy (and thus high useful information rate). Each of these two qualities have consequences on the individual hash bits. High entropy means that each bit should be balanced and uncorrelated with the other bits. high accuracy means that the underlying probability distribution for each bit should have high variance.

- **Audio hashprints:** *highly adaptive, unsupervised, facilitates search.* Based on the insights from these two theoretical questions, we proposed a representation of audio

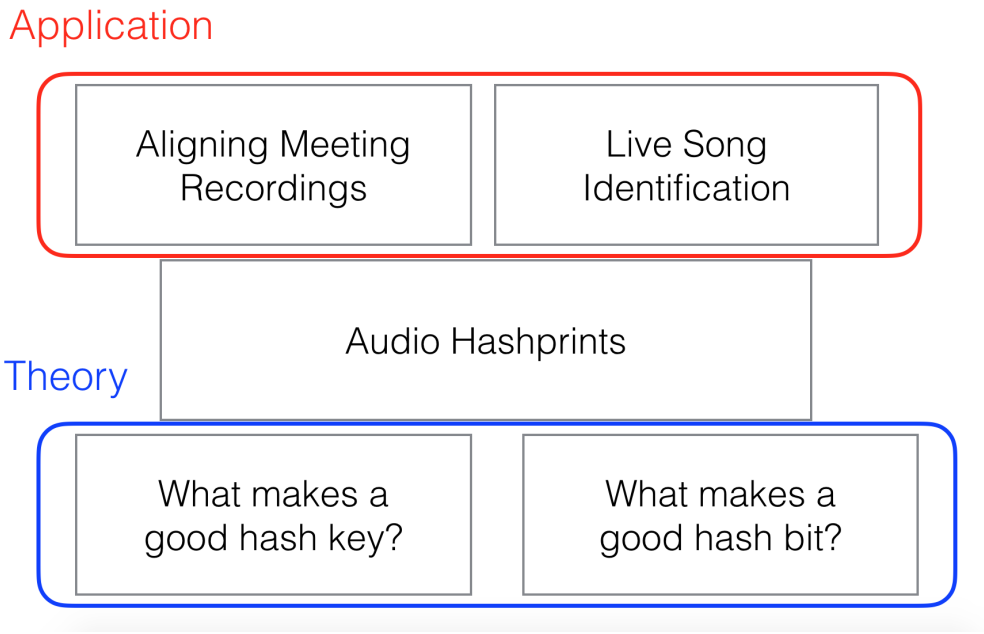


Figure 7.1: A diagram showing the high-level structure of this thesis. This figure is copied from chapter 1.

called audio hashprints. The hashprint representation is based on a set of spectro-temporal filters that can be learned in an unsupervised manner on a small amount of data (and thus highly adaptvie). The binary nature of the representation makes it useful for facilitating audio search and retrieval through indexing techniques or computing Hamming distances very efficiently.

- **Aligning meeting recordings.** *Better results, less hassle.* The first application scenario is aligning a set of unsynchronized meeting recordings. We present a method to accomplish this in a robust, efficient manner using audio hashprints and a two-stage alignment method. The proposed system has better performance than five other state-of-the-art fingerprint methods, achieving a higher accuracy at a lower error tolerance. It also has the benefit that the representation is learned on-the-fly in an unsupervised manner, so that no supervised training is necessary.

- **Live song identification.** *Better results, faster search.* The second application scenario is identifying a song given a short, noisy cell phone recording of a live performance. We present a system to identify performances by popular bands. This sytem uses an audio hashprint representation combined with a two-stage hashprint cross-correlation matching approach. Compared to the previous state-of-the-art approach, the proposed system significantly improves the reliability of the results, while simultaneously reducing the search latency by an order of magnitude.

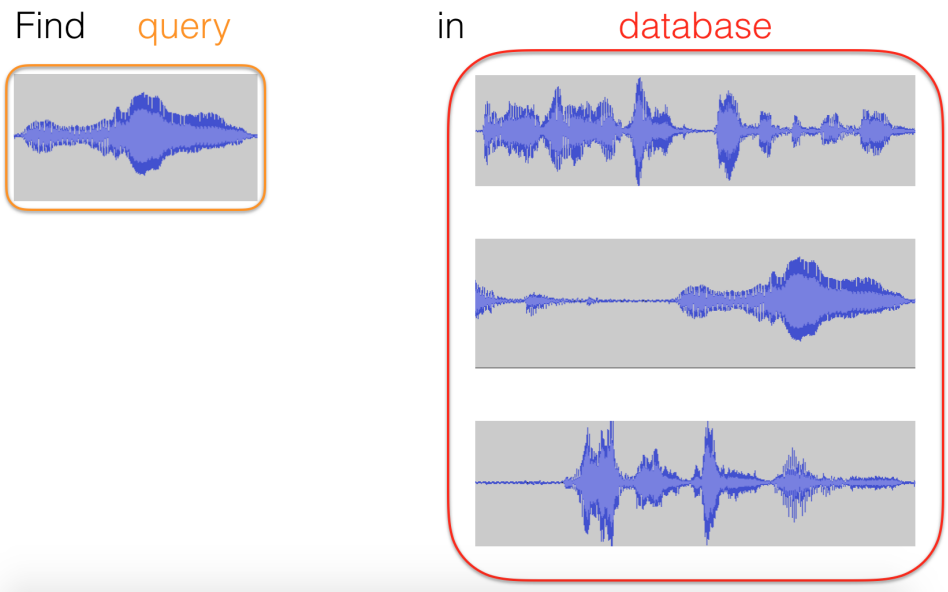


Figure 7.2: Original problem statement: given an audio query, how can we find a match in a database of audio recordings?

Having summarized the main lessons learned, we will now devote the last several sections to avenues of future work.

7.2 Future Work: Medical Diagnosis

This dissertation has focused entirely on audio search and retrieval. But audio search and retrieval is one specific example of the more general problem of *time-series* search and retrieval. We can see that the original problem statement shown in figure 7.2 has nothing in it that is particular to audio. There are many other applications and domains besides audio where the hashprint representation may be useful.

One such domain is medicine. Instead of the query being an audio recording of a live performance, it could instead be an ECG measurement of a person's heartbeat (see figure 7.3). Given a short, noisy measurement, we would like to compare it to a database of known signals for the purpose of classification and diagnosis. Some of the techniques in this dissertation will transfer into this new domain; some of the techniques will not. It will be worthwhile to investigate *how much* of this work will transfer.

One of the benefits of the hashprint representation is that there are few "knobs" to turn. Because the representation is learned in an unsupervised manner and adapts itself to the data, there are fewer aspects of the design that are task-specific. To the extent that fewer aspects of the representation are manually designed and tailored to audio, the hashprint framework may be more flexible in adapting to other types of time-series signals.

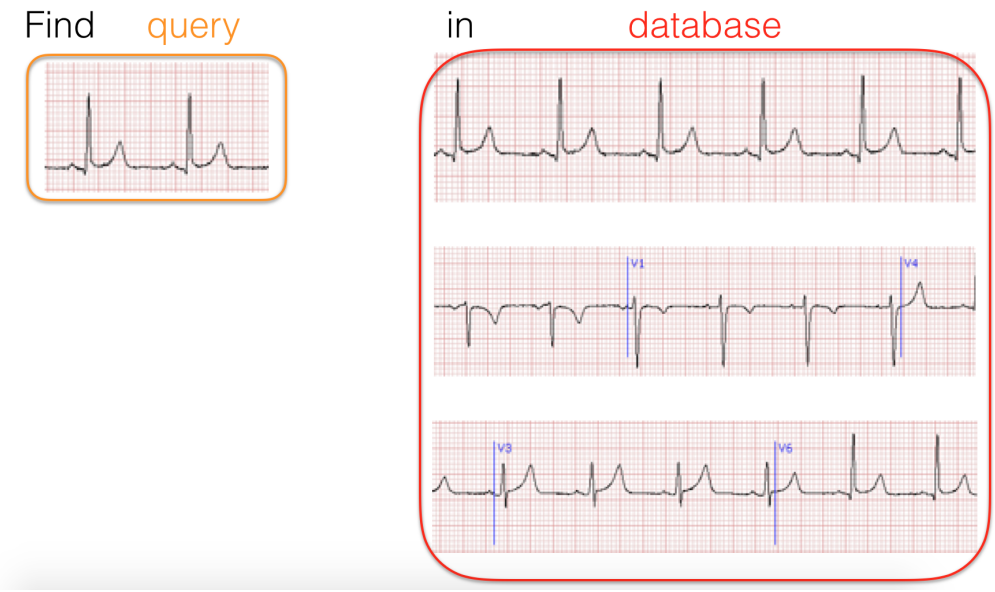


Figure 7.3: Modified problem statement: given an ECG query, how can we find a match in a database of known ECG recordings?

Personalized medicine is a promising and relevant domain to explore. In an age of skyrocketing healthcare costs, cloud-based medical diagnosis offers the possibility of drastically reducing the cost of *some* medical diagnoses. It also offers the possibilitiy of extending healthcare to places that might otherwise not have it. In developing countries, people may not have access to medical facilities or expertise. But if they have a cell phone, an internet connection, and a suitable measurement sensor, they may still be able to get access to some medical diagnoses.

7.3 Future Work: Coupling with DNNs

In section 4.5, we explained how the hashprint representation can be interpreted as a single layer neural network. We also discussed in section 6.6 how hashprints and DNN-based approaches occupy different niches: DNNs are well suited to scenarios where a lot of labeled training data is available, while hashprints are well suited to scenarios where we would like the representation to be highly adaptive to a small set of unlabeled data. DNNs are very powerful; hashprints are very flexible. Can we have a representation that has both qualities?

We can. (Maybe.) Since hashprints can be interpreted as a single layer neural network, these two approaches can be combined easily. We could first learn a DNN-based representation based on a very large training set. Using this representation as a starting point, we could then learn one additional layer using the hashprint framework. In the live song identification scenario, for example, we could learn a deep autoencoder on a very large database of music,

and then learn the (last) hashprint layer on a specific musical artist. This would offer the benefits of a DNN-based representation, while retaining the adaptivity and useful properties of the hashprint framework. Testing this idea on the live song identification scenario would be a natural starting point to explore this direction.

7.4 Future Work: Characterizing Music

When we examined the learned filters in the live song identification scenario (figures 6.16 and 6.17), we observed that the hashprint representation is quite flexible along several dimensions. For one, it is able to capture modulations in both dimensions of the time-frequency plane, which in music corresponds to rhythmic and spectral information. For another, it is able to describe both broad spectral shape and fine spectral structure, which in music corresponds to timbre and harmony. Whereas many standard feature representations like chroma and MFCC tend to focus on only one dimension of music – either harmonic content or broad spectral shape – hashprints are able to describe many important dimensions of music in one simple, unified framework. For this reason, it may be useful in characterizing music in a richer, more complex way than relying on representations like chroma and MFCC.

Hashprints can be easily tuned to focus on different aspects of music. If temporal, rhythmic information is desired, we can use a large number of context frames. If temporal, rhythmic information is *not* desired, we can use a single context frame to force the representation to focus only on spectral information. If broad spectral shape is the information of interest, we can use a log mel spectrogram as the time-frequency representation. If fine spectral structure is also of interest, we can use a CQT as the time-frequency representation instead. If *only* pitch-like information is desired, we can use a CQT, learn the filters, and select only those filters that have thin horizontal lines by applying a line detection algorithm to the learned filters. The benefit of the hashprint framework is that it can characterize many different aspects of an audio signal without the headache of setting up a supervised training or manually designing a feature.

One possible starting point is to explore rhythm-based features. If we choose a time-frequency representation that only has a few very broad frequency subbands and use a large number of context frames, we can force the hashprint representation to focus its representational power on characterizing temporal, rhythmic information. This can serve as a useful feature in analyzing drum beats for the purpose of (say) transcription of discrete events like onsets of high hat, bass drum, and cymbal. This task would also be interesting to explore since it differs from most other transcription tasks that focus primarily on harmonic content.

Bibliography

[1] E. Allamanche et al. "Content-based Identification of Audio Material Using MPEG-7 Low Level Description". In: *Proc. International Society for Music Information Retrieval (ISMIR)*. 2001.

[2] X. Anguera, A. Garzon, and T. Adamek. "MASK: Robust Local Features for Audio Fingerprinting". In: *Proc. IEEE International Conference on Multimedia and Expo (ICME)*. 2012, pp. 455–460.

[3] S. Baluja and M. Covell. "Audio Fingerprinting: Combining Computer Vision & Data Stream Processing". In: *Proc. IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*. 2007, pp. 213–216.

[4] S. Baluja and M. Covell. "Waveprint: Efficient wavelet-based audio fingerprinting". In: *Pattern Recognition* 41.11 (May 2008), pp. 3467–3480.

[5] T. Bertin-Mahieux and D.P. Ellis. "Large-scale cover song recognition using hashed chroma landmarks". In: *IEEE Workshop on Applications of Signal Processing to Audio and Acoustics (WASPAA)*. 2011, pp. 117–120.

[6] T. Bertin-Mahieux et al. "The Million Song Dataset". In: *Proc. International Society for Music Information Retrieval (ISMIR)*. 2011, pp. 591–596.

[7] I. Bisio et al. "A television channel real-time detector using smartphones". In: *IEEE Transactions on Mobile Computing* 14.1 (2015), pp. 14–27.

[8] N.J. Bryan, P. Smaragdis, and G.J. Mysore. "Clustering and synchronizing multi-camera video via landmark cross-correlation". In: *Proc. IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP'12)*. 2012, pp. 2389–2392.

[9] C.J. Burges, J.C. Platt, and S. Jana. "Distortion Discriminant Analysis for Audio Fingerprinting". In: *IEEE Transactions on Speech and Audio Processing* 11.3 (2003), pp. 165–174.

[10] J. Carletta. "Unleashing the killer corpus: experiences in creating the multi-everything AMI Meeting Corpus". In: *Language Resources and Evaluation* 41.2 (2007), pp. 181–190.

[11] M. Casey, C. Rhodes, and M. Slaney. "Analysis of minimum distances in high-dimensional musical spaces". In: *IEEE Transactions on Audio, Speech, and Language Processing* 16.5 (2008), pp. 1015–1028.

[12] B. Coover and J. Han. "A Power Mask based audio fingerprint". In: *Proc. IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*. 2014, pp. 1394–1398.

[13] C. Cotton and D. Ellis. "Audio fingerprinting to identify multiple videos of an event". In: *Proc. IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP'10)*. 2010, pp. 2386–2389.

[14] R. Dannenberg and N. Hu. "Polyphonic audio matching for score following and intelligent audio editors". In: *Proceedings of the International Computer Music Conference*. 2003, pp. 27–34.

[15] M. Datar et al. "Locality-sensitive hashing scheme based on p-stable distributions". In: *Proc. Annual Symposium on Computational geometry*. 2004, pp. 253–262.

[16] L. Deng et al. "Binary coding of speech spectrograms using a deep auto-encoder". In: *Interspeech*. 2010, pp. 1692–1695.

[17] J. Downie et al. "Audio Cover Song Identification: MIREX 2006-2007 Results and Analyses". In: *Proc. International Society for Music Information Retrieval (ISMIR)*. 2008, pp. 468–474.

[18] N.Q. Duong, C. Howson, and Y. Legallais. "Fast second screen TV synchronization combining audio fingerprint technique and generalized cross correlation". In: *IEEE International Conference on Consumer Electronics-Berlin (ICCE-Berlin)*. 2012, pp. 241–244.

[19] N.Q. Duong and F. Thudor. "Movie synchronization by audio landmark matching". In: *Proc. IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP'13)*. 2013, pp. 3632–3636.

[20] E. Dupraz and G. Richard. "Robust frequency-based audio fingerprinting". In: *IEEE International Conference on Acoustics, Speech, and Signal Processing (ICASSP)*. 2010, pp. 281–284.

[21] D. Ellis. *Robust Landmark-Based Audio Fingerprinting*. 2015. URL: `http://labrosa.ee.columbia.edu/matlab/fingerprint/`.

[22] D. Ellis and T. Bertin-Mahieux. "Large-scale cover song recognition using the 2D fourier transform magnitude". In: *Proc. International Society for Music Information Retrieval (ISMIR)*. 2012, pp. 241–246.

[23] D. Ellis and G. Poliner. "Identifying 'Cover songs' with chroma features and dynamic programming beat tracking". In: *IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*. 2007, pp. 1429–1432.

[24] D.P. Ellis and C. Cotton. "The 2007 Labrosa Cover Song Detection System". In: *Music Information Retrieval Evaluation Exchange (MIREX)*. 2007.

[25] S. Fenet, Y. Grenier, and G. Richard. "An Extended Audio Fingerprint Method with Capabilities for Similar Music Detection". In: *Proc. International Society for Music Information Retrieval (ISMIR)*. 2013, pp. 569–574.

[26] S. Fenet, G. Richard, and Y. Grenier. "A Scalable Audio Fingerprint Method with Robustness to Pitch-Shifting". In: *Proc. International Society for Music Information Retrieval (ISMIR)*. 2011, pp. 121–126.

[27] S. Fenet et al. "A framework for fingerprint-based detection of repeating objects in multimedia streams". In: *IEEE Proceedings of the 20th European Signal Processing Conference (EUSIPCO)*. 2012, pp. 1464–1468.

[28] J. George and A. Jhunjhunwala. "Scalable and robust audio fingerprinting method tolerable to time-stretching". In: *IEEE International Conference on Digital Signal Processing (DSP)*. 2015, pp. 436–440.

[29] P. Grosche and M. Müller. "Toward characteristic audio shingles for efficient cross-version music retrieval". In: *IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*. 2012, pp. 473–476.

[30] P. Grosche and M. Müller. "Toward Musically-Motivated Audio Fingerprints". In: *Proceedings of the IEEE International Conference on Acoustics, Speech, and Signal Processing (ICASSP)*. Kyoto, Japan, 2012, pp. 93–96.

[31] V.N. Gupta, G. Boulianne, and P. Cardinal. "CRIM's Content-Based Audio Copy Detection System for TRECVID 2009". In: *Multimedia Tools and Applications* 60.2 (2012), pp. 371–387.

[32] J. Haitsma and T. Kalker. "A Highly Robust Audio Fingerprinting System". In: *Proc. International Society for Music Information Retrieval (ISMIR)*. 2002, pp. 107–115.

[33] J. Haitsma, T. Kalker, and J. Oostveen. "Robust audio hashing for content identification". In: *International Workshop on Content-Based Multimedia Indexing*. Vol. 4. 2001, pp. 117–124.

[34] M. Héritier et al. "CRIM's Content-Based Copy Detection System for TRECVID". In: *TRECVID 2009 Workshop*. Gaithersburg, Maryland, USA, 2009.

[35] C. Herley. "ARGOS: Automatically extracting repeating objects from multimedia streams". In: *IEEE Transactions on Multimedia* 8.1 (2006), pp. 115–129.

[36] J. Herre, E. Allamanche, and O. Hellmuth. "Robust Matching of Audio Signals Using Spectral Flatness Features". In: *IEEE Workshop on the Applications of Signal Processing to Audio and Acoustics*. New Platz, New York, USA, Oct. 2001, pp. 127–130.

[37] G. Hinton and R. Salakhutdinov. "Reducing the dimensionality of data with neural networks". In: *Science* 313.5786 (2006), pp. 504–507.

[38] T.K. Hon et al. "Audio fingerprinting for multi-device self-localization". In: *IEEE/ACM Transactions on Audio, Speech and Language Processing (TASLP)* 23.10 (2015), pp. 1623–1636.

[39] N. Hu, R. Dannenberg, and G. Tzanetakis. "Polyphonic audio matching and alignment for music retrieval". In: *IEEE Workshop on Applications of Signal Processing to Audio and Acoustics (WASPAA)*. 2003.

[40] E.J. Humphrey, O. Nieto, and J.P. Bello. "Data Driven and Discriminative Projections for Large-Scale Cover Song Identification". In: *Proc. International Society for Music Information Retrieval (ISMIR)*. 2013, pp. 149–154.

[41] A.C. Ibarrola and E. Chávez. "A robust entropy-based audio-fingerprint". In: *IEEE International Conference on Multimedia and Expo*. 2006, pp. 1729–1732.

[42] D. Jang et al. "Pairwise Boosted Audio Fingerprint". In: *IEEE Transactions on Information Forensics and Security* 4.4 (2009), pp. 995–1004.

[43] A. Janin et al. "The ICSI Meeting Corpus". In: *Proc. IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP'03)*. 2003, pp. 364–367.

[44] H. Jégou et al. "Babaz: a large scale audio search system for video copy detection". In: *Proc. IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*. 2012, pp. 2369–2372.

[45] Y. Ke, D. Hoiem, and R. Sukthankar. "Computer Vision for Music Identification". In: *Proc. IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR)*. 2005, pp. 597–604.

[46] L. Kennedy and M. Naaman. "Less Talk, More Rock: Automated Organization of Community-Contributed Collections of Concert Videos". In: *Proc. ACM International Conference on World Wide Web*. 2009, pp. 311–320.

[47] M. Khadkevich and M. Omologo. "Large-Scale Cover Song Identification Using Chord Profiles". In: *Proc. International Society for Music Information Retrieval (ISMIR)*. 2013, pp. 233–238.

[48] H. Khemiri, D. Petrovska-Delacretaz, and G. Chollet. "Detection of repeating items in audio streams using data-driven ALISP sequencing". In: *IEEE International Conference on Advanced Technologies for Signal and Image Processing (ATSIP)*. 2014, pp. 446–451.

[49] S. Kim and C.D. Yoo. "Boosted Binary Audio Fingerprint Based on Spectral Subband Moments". In: *Proc. IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*. 2007, pp. 241–244.

[50] F. Kurth and M. Müller. "Efficient index-based audio matching". In: *IEEE Transactions on Audio, Speech, and Language Processing* 16.2 (2008), pp. 382–395.

[51] J. Lebossé, L. Brun, and J.C. Pailles. "A robust audio fingerprint extraction algorithm". In: *Proceedings of IASTED International Conference: Signal Processing, Pattern Recognition, and Applications*. 2007, pp. 269–274.

[52] V. Liong et al. "Deep Hashing for Compact Binary Codes Learning". In: *Proc. IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. 2015, pp. 2475–2483.

[53] C.C. Liu and P.F. Chang. "An Efficient Audio Fingerprint Design for MP3 Music". In: *Proc. ACM International Conference on Advances in Mobile Computing and Multimedia (MoMM'11)*. 2011, pp. 190–193.

[54] Y. Liu, H.S. Yun, and N.S. Kim. "Audio fingerprinting based on multiple hashing in DCT domain". In: *IEEE Signal Processing Letters* 16.6 (2009), pp. 525–528.

[55] Y. Liu et al. "Coherent Bag-Of Audio Words Model for Efficient Large-Scale Video Copy Detection". In: *Proc. ACM International Conference on Image and Video Retrieval*. 2010, pp. 89–96.

[56] R. Macrae, X. Anguera, and N. Oliver. "MuViSync: Realtime music video alignment". In: *IEEE International Conference on Multimedia and Expo (ICME)*. 2010, pp. 534–539.

[57] M. Malekesmaeili and R.K. Ward. "A local fingerprinting approach for audio copy detection". In: *Signal Processing* 98 (2014), pp. 308–321.

[58] N. Mesgarani, M. Slaney, and S.A. Shamma. "Discrimination of speech from non-speech based on multiscale spectro-temporal modulations". In: *IEEE Transactions on Audio, Speech, and Language Processing* 14.3 (2006), pp. 920–930.

[59] R. Mukai et al. "NTT Communication Science Laboratories at TRECVID 2010 Content Based Copy Detection". In: *Proc. of TRECVID*. 2010.

[60] M. Müller. *Fundamentals of Music Processing*. Springer, 2015.

[61] M. Müller, F. Kurth, and M. Clausen. "Audio Matching via Chroma-Based Statistical Features". In: *Proceedings of the International Society for Music Information Retrieval (ISMIR)*. 2005, pp. 288–295.

[62] M. Müller, F. Kurth, and M. Clausen. "Chroma-Based Statistical Audio Features for Audio Matching". In: *Proceedings of the Workshop on Applications of Signal Processing (WASPAA)*. New Paltz, New York, USA, Oct. 2005, pp. 275–278.

[63] H. Nagano et al. "A fast audio search method based on skipping irrelevant signals by similarity upper-bound calculation". In: *Proc. IEEE Acoustics, Speech and Signal Processing (ICASSP)*. 2015, pp. 2324–2328.

[64] C.W. Ngo et al. "VIREO/DVMM at TRECVID 2009: High-Level Feature Extraction, Automatic Video Search, and Content-Based Copy Detection". In: *Proc. of TRECVID*. 2009, pp. 415–432.

[65] M. Norouzi, D. Blei, and R. Salakhutdinov. "Hamming distance metric learning". In: *Advances in neural information processing systems (NIPS)*. 2012, pp. 1061–1069.

[66] J. Ogle and D. Ellis. "Fingerprinting to Identify Repeated Sound Events in Long-Duration Personal Audio Recordings". In: *Proc. IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP'07)*. Honolulu, Hawaii, USA, Apr. 2007, pp. 233–236.

[67] J. Osmalsky et al. "Combining Features for Cover Song Identification". In: *Proc. International Society for Music Information Retrieval (ISMIR)*. 2015, pp. 462–468.

[68] C. Ouali, P. Dumouchel, and V. Gupta. "A robust audio fingerprinting method for content-based copy detection". In: *IEEE International Workshop on Content-Based Multimedia Indexing (CBMI)*. 2014, pp. 1–6.

[69] P. Over et al. "TRECVID 2011 — An Overview of the Goals, Tasks, Data, Evaluation Mechanisms and Metrics". In: *Proc. of TRECVID*. 2011.

[70] M. Park, H. Kim, and S.H. Yang. "Frequency-temporal filtering for a robust audio fingerprinting scheme in real-noise environments". In: *Electronics and Telecommunications Research Institute Journal* 28.4 (2006), pp. 509–512.

[71] B.N. Pasley et al. "Reconstructing speech from human auditory cortex". In: *PLoS Biol* 10.1 (2012), e1001251.

[72] R. Radhakrishnan and W. Jiang. "Repeating segment detection in songs using audio fingerprint matching". In: *IEEE Asia-Pacific Signal & Information Processing Association Annual Summit and Conference (APSIPA ASC)*. 2012, pp. 1–5.

[73] C. Raffel and D. Ellis. "Large-Scale Content-Based Matching of MIDI and Audio Files". In: *Proc. International Society for Music Information Retrieval (ISMIR)*. 2015, pp. 234–240.

[74] Z. Rafii, B. Coover, and J. Han. "An audio fingerprinting system for live version identification using image processing techniques". In: *IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*. 2014, pp. 644–648.

[75] A. Ramalingam and S. Krishnan. "Gaussian mixture modeling using short time fourier transform features for audio fingerprinting". In: *IEEE International Conference on Multimedia and Expo*. 2005, pp. 1146–1149.

[76] M. Ramona and G. Peeters. "Audio Identification Based on Spectral Modeling of Bark-Bands Energy and Synchronization Through Onset Detection". In: *Proc. IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP'11)*. 2011, pp. 477–480.

[77] S. Ravuri and D. Ellis. "Cover song detection: from high scores to general classification". In: *IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*. 2010, pp. 65–68.

[78] R. Salakhutdinov and G. Hinton. "Semantic hashing". In: *International Journal of Approximate Reasoning* 50.7 (2009), pp. 969–978.

[79] A. Saracoglu et al. "Content Based Copy Detection with Coarse Audio-Visual Fingerprints". In: *IEEE International Workshop on Content-Based Multimedia Indexing (CBMI)*. 2009, pp. 213–218.

[80] C. Schörkhuber and A. Klapuri. "Constant-Q transform toolbox for music processing". In: *Sound and Music Computing Conference*. 2010, pp. 3–64.

[81] J.S. Seo. "An asymmetric matching method for a robust binary audio fingerprinting". In: *IEEE Signal Processing Letters* 21.7 (2014), pp. 844–847.

[82] J.S. Seo et al. "Audio Fingerprinting Based on Normalized Spectral Subband Centroids". In: *Proc. IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*. 2005, pp. 213–216.

[83] J.S. Seo et al. "Audio Fingerprinting Based on Normalized Spectral Subband Moments". In: *IEEE Signal Processing Letters* 13.4 (2006), pp. 209–212.

[84] J. Serra and E. Gomez. "A Cover Song Identification System Based on Sequences of Tonal Descriptors". In: *Music Information Retrieval Evaluation Exchange (MIREX)*. 2007.

[85] J. Serra, E. Gómez, and P. Herrera. "Audio cover song identification and similarity: background, approaches, evaluation, and beyond". In: *Advances in Music Information Retrieval*. 2010, pp. 307–332.

[86] J. Serra, X. Serra, and R. Andrzejak. "Cross recurrence quantification for cover song identification". In: *New Journal of Physics* 11.9 (2009), p. 093017.

[87] Y. Shi, W. Zhang, and J. Liu. "Robust Audio Fingerprinting Based on Local Spectral Luminance Maxima Scheme". In: *Proc. Interspeech*. 2011, pp. 2485–2488.

[88] P. Shrestha, M. Barbieri, and H. Weda. "Synchronization of Multi-Camera Video Recordings Based on Audio". In: *Proc. ACM International Conference on Multimedia*. 2007, pp. 545–548.

[89] J. Six and M. Leman. "Panako: a scalable acoustic fingerprinting system handling time-scale and pitch modification". In: *Proc. International Society for Music Information Retrieval (ISMIR)*. 2014.

[90] J. Six and M. Leman. "Synchronizing multimodal recordings using audio-to-audio alignment". In: *Journal on Multimodal User Interfaces* 9.3 (2015), pp. 223–229.

[91] W. Son et al. "Sub-fingerprint masking for a robust audio fingerprinting system in a real-noise environment for portable consumer devices". In: *IEEE Transactions on Consumer Electronics* 56.1 (2010), pp. 156–160.

[92] R. Sonnleitner and G. Widmer. "Quad-based audio fingerprinting robust to time and frequency scaling". In: *Proc. International Conference on Digital Audio Effects*. 2014, pp. 173–180.

[93] K. Su et al. "Making a Scene: Alignment of Complete Sets of Clips Based on Pairwise Audio Match". In: *Proc. ACM International Conference on Multimedia Retrieval (ICMR'12)*. 2012.

[94] S. Sukittanon and L.E. Atlas. "Modulation Frequency Features for Audio Fingerprinting". In: *Proc. IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*. 2002, pp. 1773–1776.

[95] T. Tsai, G. Friedland, and X. Anguera. "An information-theoretic metric of fingerprint effectiveness". In: *IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*. IEEE. 2015, pp. 340–344.

[96] T. Tsai and A. Stolcke. "Robust and Efficient Multiple Alignment of Unsynchronized Meeting Recordings". In: *IEEE/ACM Transactions on Audio, Speech, and Language Processing* 24.5 (2016), pp. 833–845.

[97] Y. Uchida et al. "KDDI Labs and SRI International at TRECVID 2010: Content-Based Copy Detection". In: *Proc. of TRECVID*. 2010.

[98] E.M. Voorhees. "The TREC-8 Question Answering Track Report". In: *Proceedings of the 8th Text Retrieval Conference*. 1999, pp. 77–82.

[99] A. Wang. "An Industrial-Strength Audio Search Algorithm". In: *Proc. International Society for Music Information Retrieval (ISMIR)*. 2003, pp. 7–13.

[100] Y. Weiss, A. Torralba, and R. Fergus. "Spectral Hashing". In: *Advances in Neural Information Processing Systems (NIPS)*. 2009, pp. 1753–1760.

[101] E. Younessian et al. "Telefonica Research at TRECVID 2010 Content-Based Copy Detection". In: *Proc. of TRECVID*. 2010.