NTNU
Norwegian University of
Science and Technology

Combining Audio Fingerprints

Vegard Andreas Larsen

Master of Science in Computer Science
Submission date: June 2008
Supervisor: Trond Aalberg, IDI

Norwegian University of Science and Technology
Department of Computer and Information Science

Problem Description

The thesis will investigate the possibility of combining two or more existing acoustic fingerprinting solutions into a common solution, to generate a more universal fingerprint. By combining the probabilities from each fingerprinting solution in various ways into a combined score, the solution will be more adept at finding equivalent recordings than a single solution will be. The thesis will test the fingerprinting systems accuracy separately, and then combine them in different ways to see if a better set of matches can be found.

Assignment given: 15. January 2008
Supervisor: Trond Aalberg, IDI

Abstract

Large music collections are now more common than ever before. Yet, search technology for music is still in its infancy. Audio fingerprinting is one method that allows searching for music.

In this thesis several audio fingerprinting solutions are combined into a single solution to determine if such a combination can yield better results than any of the solutions can separately. The solution is used to find duplicate music files in a personal collection.

The results show that applying the weighted root-mean square (WRMS) to the problem most effectively ranked the results in a satisfying manner. It was notably better than the other approaches tried. The WRMS produced 61% more correct matches than the original FDMF solution, and 49% more correct matches than libFooID.

Contents

List of Figures

List of Tables

Acknowledgments

Thanks to Trond Aalberg for supervising me throughout the semester with this thesis.

Thanks to Gian-Carlo Pascutto for providing me with helpful hints regarding libFooID, even though he no longer actively maintains the project. Thanks to Kurt Rosenfeld for giving me insight into the inner workings of fdmf. Thanks to Steinar Gunderson for providing helpful pointers on how to modify his system.

Thanks to (in alphabetical order) Trond Aalberg, Anne Cecilie Burhol, Hilde Halland, Francis Rath and Frode Sandholtbråten for proof-reading drafts of this thesis.

Vegard Andreas Larsen

Trondheim, June 3rd 2008

Chapter 1

Introduction

1.1 Motivation

Music is everywhere. In the 150 years since its invention recorded music has come into everyday use. Music players are now so compact that mobile phone producers include them in almost every phone they sell.

It is now normal to have large digital music collections that were unfeasible even ten years ago. When computers became a commodity, it was mostly used for textual documents. It took many years before it was also used for music. Music is different in fundamental ways from text in that it is not easily searchable without associated metadata. In the extreme case of a music collection without any associated metadata it can be very difficult to locate a song in that collection.

With a large music collection it is not uncommon to have multiple copies of songs or even entire albums. Without even knowing it, you may have copies of a certain album in two different formats, meaning that the size of the files, or even the quality, may be different.

Audio fingerprinting systems are in use today for a variety of different uses, such as automatically finding metadata for a given recording, intellectual property rights management or clustering of similar music to find music suggestions. When you listen to a song on your car stereo, a modern fingerprinting system could tell you the performing artist and the song title if it has the fingerprint of the song in its database. Broadcast monitoring could be made easier if there were fingerprinting systems in place to automatically transcribe a list of music being played on the radio or on TV. This thesis confines the area of study to finding such duplicates in a music collection.

1.2 Objectives

This thesis focuses on finding audibly similar music in a collection. The main objective is better performing fingerprint solutions that correctly identifies audibly similar music. Statistical and probabilistic combination methods are applied to audio fingerprints from different systems to increase recognition rates.

This thesis seeks to answer the following questions:

- Can combining multiple descriptors from separate authors produce a better fingerprint?
- Which methods of combining descriptors produce consistently better results?
- How do the descriptors tested rank in performance?

1.3 Approach

This thesis uses a music collection with 9536 music files. The music collection is a real-world example of what might be in someone's personal music collection. The music collection consists mainly of MP3 and WMA files in a ratio of approximately 8:1.

This thesis uses the fdmf and libFooID software, and in addition includes fingerprinting software from a masters thesis. fdmf and libFooID (in the form of Foosic) are both projects that can quickly be tested by downloading them of the Internet. An individual audio fingerprint, known as a descriptor, for a file can be compared to the same descriptor for another file. The result is a percentage that indicates the similarity between the two files. Outputs from multiple systems are combined using several approaches, e.g. the average and a naïve Bayes classifier, to give a combined score for how similar the two files being combined are.

The first 1000 matches are compared to a reference list of verified equivalent files, and the failure rate is used as a measure to compare performance characteristics of a given combination.

1.4 Results

This thesis has proven that combining descriptors from multiple sources is a viable way to create a better fingerprint. It finds that a weighted root-mean square (WRMS) of a given set of 8 descriptors gives the best results with the given music collection. When picking descriptors carefully, other combination methods can be used with results nearly as good as the WRMS.

Among the tested descriptors it is found that libFooID is the most reliable, followed closely by fdmf_0, two of the MFCC descriptors, F1CC average, fdmf_2 and fdmf_1. This thesis also found that several of the descriptors cannot be used to reliably identify recordings, amongst them are song length, centroid, steepness, mean/square ratio and the rate of zero crossings.

1.5 Structure

The report is divided into the following chapters.

- Chapter 2 presents basic sound theory, examines previous work in the field, and presents the software solutions to be tested.
- Chapter 3 presents the methods used to test and combine the various solutions into a working system. It shows how fingerprints are compared, how the results are combined, and how the accuracy of a system is measured.

- Chapter 4 presents the results of this thesis, and discusses any discrepancies seen.
- Chapter 5 presents the conclusion to this thesis, and outlines improvements that could be made to the system.

Chapter 2

Pre-study

This chapter examines the basis for audio fingerprints, and looks at previous work in the area of music information retrieval (MIR). It then looks at the individual descriptors that will be combined into a single solution.

2.1 Sound theory

Sound is physical waves moving through a medium such as air with a frequency between approximately 20 Hz and 20 kHz. The physical waves can be described as sequential compression and decompression of adjacent molecules in the medium. Sound waves are mostly started by vibration in a physical object, setting the medium into motion.

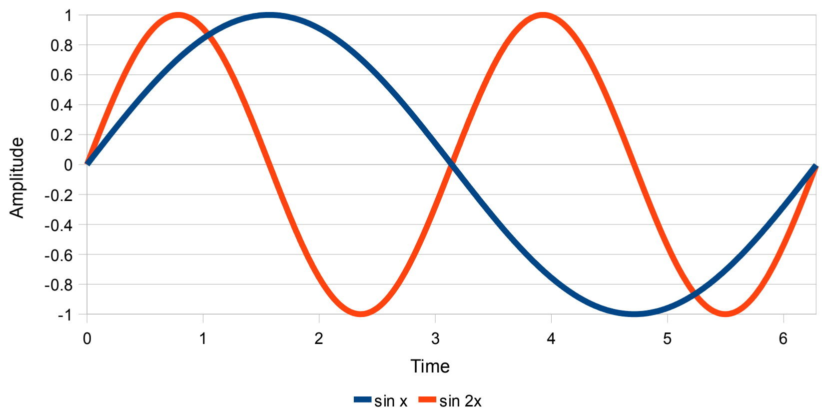
Interactions between waves grows complex very quickly. As can be seen in figure 2.2, the combination of three simple waves result in a wave that is very unpredictable. In the real world several hundred single waves combine to make the sound of a violin.

2.1.1 Waves and interactions

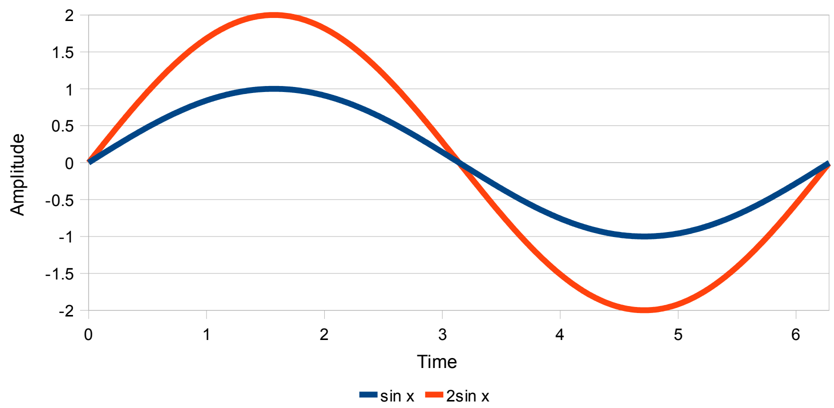
Pure sound waves are characterized by two main properties; the *frequency* and the *amplitude*. The frequency measures how often the wave repeats itself, and is measured in Hertz (Hz). 1 Hz indicates an event occurring once every second. Figure 2.1(a) shows two sine waves, one with a frequency of $\frac{1}{2\Pi}$, the other with a frequency of $\frac{1}{\Pi}$. Waves with a higher frequency has a shorter distance between two local maxima, a distance known as the *cycle* or *period* of a wave.

The amplitude of the wave is a measurement of how much of the medium the wave can displace during one period. When talking about amplitude, one usually means the peak amplitude, which is the distance from neutral to the maxima. The waves presented in figure 2.1(b) has peak amplitudes of 1 ($\sin x$) and 2 ($2 \sin x$). A single sine wave will sound like a clear, steady tone.

Waves interact in very complex ways, which produce different sounds to the human ear. Waves can collide, and provide positive and negative feedback that either neutralizes the sound wave, or increases its amplitude. In extreme cases where two waves with the same frequency collide and maxima line up, the amplitude is added together for the two waves. This is known as the waves being



(a) Sine waves of different frequency



(b) Sine waves of different amplitude.

Figure 2.1: Features of waves illustrated.

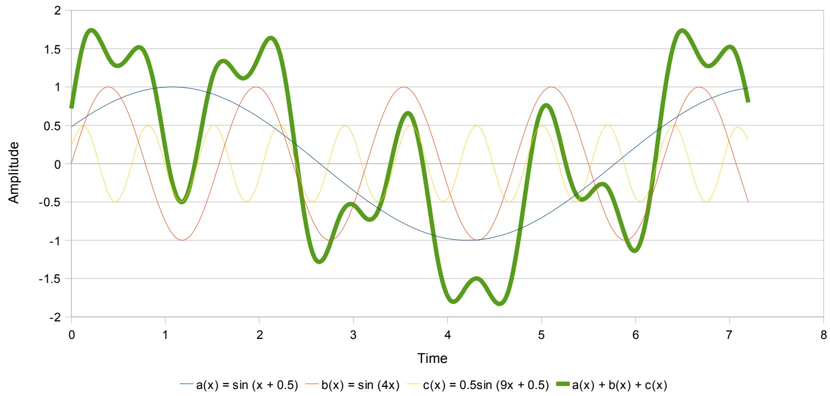


Figure 2.2: Combination of three sine waves results in complex wave.

in phase. When the waves are out of phase they cancel each other out. The combined wave is always the sum of the displacement of the individual waves at any given point.

Waves are defined by the wave equation, a second-order linear partial differential equation. It gives the propagation of waves with a given speed v, with ∇^2 being the Laplacian. For a detailed explanation of the wave equation, the reader is referred to an article in MathWorld [38].

$$\nabla^2 = \frac{1}{v^2}\frac{\partial^2 \psi}{\partial t^2} \tag{2.1}$$

2.1.2 Humans and sound

Sound is perceived by humans through the ears, where the sound waves hit the tympanic membrane[1]. The tympanic membrane transfers the sound waves' kinetic energy to the ossicles in the middle ear, which again transfers the energy to the cochlea. Inside the cochlea fluids are set in motion, and tiny hairs in the Organ of Corti register the movements and signals the brain. The brain then interprets these signals as sound. For more details, see [29].

The human auditory system does not perceive sound linearly, and the sound pressure level, the volume, is often quoted in decibel (dB). The sound pressure level is logarithmic, and measures the root-mean square change from the ambient pressure caused by a sound wave (equation 2.2). The reference sound pressure p_{ref} is $20\mu Pa$. Human perception of sound loudness roughly follows the sound pressure level on the decibel scale, meaning that a doubling of the sound pressure will be perceived as a constant increase, no matter what the previous sound pressure.

$$L_p = 10\log_{10}\left(\frac{p_{rms}^2}{p_{ref}^2}\right) = 20\log_{10}\left(\frac{p_{rms}}{p_{ref}}\right) \tag{2.2}$$

[1] Colloquially known as the *eardrum*.

Note	Frequency	n
G	392Hz	-2
G♯ / A4♭	415Hz	-1
A4	440Hz	0
A4♯ / B♭	466Hz	1
B	494Hz	2
C	523Hz	3
C♯ / D♭	554Hz	4
D	587Hz	5
D♯ / E♭	622Hz	6
E	659Hz	7
F	698Hz	8
F♯ / G♭	740Hz	9
G	784Hz	10
G♯ / A5♭	831Hz	11
A5	880Hz	12

Table 2.1: The chromatic scale around A4, with frequencies and n-values (see equation 2.3).

Pitch is another facet of sound that the human auditory system perceives roughly logarithmically [10]. It is the perceived frequency of sound, and the perception can be changed by playing other frequencies simultaneously. Pitch is close to the fundamental frequency of a sound, yet the perceived pitch can change subtly when harmonics are introduced.

Harmonics and pitch are closely related. Harmonics are the effects produced when multiple sound waves, all with frequencies that are multiples of a common fundamental frequency, occur concurrently. The canonical example is the musical note A4, which in current Western music is defined to have a frequency of 440 Hz. The 440 Hz A4 note is considered the first harmonic for itself. The 880 Hz overtone is called the second harmonic, and corresponds to the musical note A5. 1320 Hz is the third harmonic (A6). Those familiar with music notation will notice that moving from A4 to A5 is the same as moving up an octave.

For Western music the range between two harmonics is usually divided into 12 half-steps, of which seven are assigned one-letter names [20]. When only using these seven denotations (A, B, C, D, E, F, G), you are using the diatonic scale. When also looking at the half-steps between some of these notes, you are using the chromatic scale. It is worth noting that notes in the diatonic scale is not evenly spaced. The frequency of a note is calculated as in equation 2.3, where n is the number of half-steps away from A4. The equation has been used to generate table 2.1.

$$f = 2^{n/12} \times 440 \tag{2.3}$$

2.1.3 Digital sound

For sound to be stored in a computer it must be digitized. Since computers operate using discrete numbers — contrasted to the continuous spectrum of sound — digitizing sound carries an inherent degradation. Sound is *sampled* at

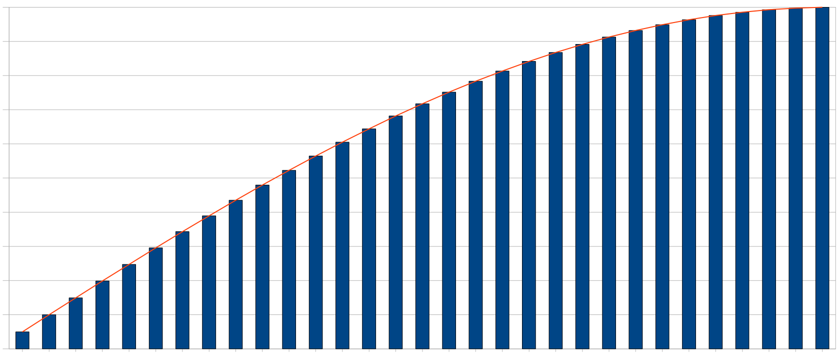


Figure 2.3: The line shows a continuous line, representing the continuous audio signal in the medium, and the vertical bars representing the sampled values.

regular intervals, and the computer stores the sound pressure. The sampling frequency is also measured in Hz, or samples per second, and specifies how often the samples are taken. Due to the Nyquist-Shannon sampling theorem[2] , the sampling frequency has to be at least twice the highest frequency of the signal you are trying to capture. Specifically;

> If a function $f(t)$ contains no frequencies higher than W cps[3], it is completely determined by giving its ordinates at a series of points spaced $\frac{1}{2W}$ seconds apart.[23]

Since human hearing is limited upwards at around 20kHz, most sound today is sampled at 44.1kHz or 48kHz using 16 bits to represent the sound pressure in each sample. Because of the high number of samples required, digitally stored sound has high storage requirements. For a single CD, which can contain up to 74 minutes of sound, a staggering 650 megabytes is required. This results in a bit rate of approximately 1411kbps; or around $172\frac{kB}{sec}$.

There is an alternative way to store music using a discrete computer encoding. The best known is MIDI (Musical Instrument Digital Interface), where the individual notes and pauses are stored using a compact digital encoding. It can be thought of as a computer equivalent to sheet music, with instructions as to which instrument should be played at what time, using a given volume and pitch. A three minute song may therefore be encoded in as little as a few kilobytes of data, but cannot contain lyrics and can only use the instruments specified in the MIDI standard. Music encoded as MIDI is therefore perceptually very different to a human listener. Headway is being made in this kind of compact representation, using three-dimensional models of instruments and implementing physical laws to synthesize the sound [4]. This thesis is only concerned with the representation of sound as sampled audio.

[2]sometimes referred to as the Shannon sampling theorem, or simply the sampling theorem
[3]cycles per second

2.1.4 Compressing digital sound

Because of the high storage requirements for raw digital sound, several methods for compressing the sound has emerged. Traditional compression techniques, used for e.g. text documents, can also be used for compressing sound. The compression ratio gained from traditional compression techniques as applied to sound is inadequate for enabling storage of large amounts of digital sound.

Other techniques evolved as a response, with the two main branches being lossless and lossy compression techniques. With lossless compression, it is guaranteed that a file that has been compressed and then decompressed is bit for bit identical to the original. With lossy encoding, the compression algorithm can throw away non-significant data to decrease size. If the algorithm is programmed to discard data that is hard for humans to perceive, it the loss of data can be acceptable to human listeners. A lossy algorithm works very badly with text, as letters, words or sentences may disappear completely from the text. Traditional text compression techniques are therefore lossless.

The most famous audio codec is MP3 (MPEG-1 Audio Layer 3), a lossy compression that was designed for the MPEG video encoding format to store audio [5]. Before it was chosen as an audio layer in the MPEG-1 standard, it was known as ASPEC (Adaptive Spectral Perceptual Entropy Coding) [6]. The most common MP3 file format has a bit rate of 128kbps, which is approximately 10 percent of the original file, assuming the original was from a CD. The MP3 format discards data or reduces precision in data that cannot be easily perceived by humans, based on psycho-acoustic models. Several other lossy formats, such as Ogg Vorbis, Windows Media Audio and AAC exist.

Lossless formats only look at efficient ways of storing the sound data, such as finding recurring patterns, and do not reduce precision in or discard data. These formats typically reduce the files size to 40 to 50 percent of the original file's size. Formats here include FLAC, Windows Media Audio Lossless and ATRAC.

2.2 Music Information Retrieval

The field of music information retrieval (MIR) is very broad, and encompasses a wide range of possible uses. The field is mainly concerned with extracting information from music contents, without relying on human-supplied metadata. Various research areas include:

Speech recognition Systems that automatically extract lyrics from a song could help index large amounts of audio recordings for search using standard information retrieval techniques. Problems with this approach include large amounts of background noise[4], variations in speech patterns and the large amount of languages used in music.

Automated transcription Producing sheet music from any piece of music, suitable for an orchestra or soloist to play. The current models for understanding sound waves — the models that are used for extracting the individual notes for an individual instrument — are not adequate to understand the complex interactions between sound waves produced by multiple

[4] Noise in the context of extracting the spoken word.

instruments playing in unison. It can be hard for untrained humans to differentiate between different types of instruments in a sound recording.

Query by humming (QBH) Allows searching through a database of music by humming or whistling a song to the computer. The computer then identifies the fundamental frequencies of the humming, and looks them up in the indexed music. The music indexed by current QBH systems has to be in structured form and not as sampled audio.

Audio fingerprinting Automatically finds the identifying characteristics in a piece of music. Some audio fingerprint systems allow nearest neighbor searches for music, others concentrate solely on identifying a specific recording of a song. Audio fingerprint systems can e.g. be used to identify the currently playing song on the radio or to find duplicates in a personal music collection.

Feature extraction Used for identifying the features of a musical track, such as e.g. beats per minute. The Echo Nest has a free API that can be used to extract various features from audio, and has an example that allows a user to put together a seamless mix of several songs automatically [7].

Genre classification A wide variety of music genres is represented in present-day music. A lot of work has gone into producing a system that can automatically classify music into genres, and that can compare similarities between different genres of music.

2.2.1 Properties of music

Describing music is a very hard task, and attempts to reduce the complexity of this task is usually done by splitting the music into different aspects. Downie presents seven facets that can be used to describe music, listed here for convenience [9]:

Pitch facet The perceived frequency of a sound. Hard to determine from a recording.

Temporal facet Duration of a musical event, i.e. the length of holding a given note.

Harmonic facet Several instruments playing at once with pitch frequencies that are multiples of a given fundamental frequency for that instrument.

Timbral facet The property that differentiates the sound from two different instruments, i.e. how a clarinet and a trumpet is different.

Editorial facet Instructions given to the performers that may change between recordings.

Textual facet Lyrics of a song.

Bibliographic facet Any metadata, such as title, artist and composer. Usually textual.

Most MIR software only concern themselves with a subset of these facets. The techniques used in this thesis do not at all rely on the editorial, textual or bibliographic facets.

String	MD5
A MD5 hash value changes quickly.	`eb0583fed3abca7103419cfce517046e`
A MD5 hash changes quickly.	`0e04f19f5c76bce72cf68e0c007346a0`
A MD5 hash value changes drastically.	`6cabee6499ab39254ae765dde785c7a9`

Table 2.2: MD5 hashes of three similar text strings.

2.3 Audio fingerprinting

Audio fingerprinting can be loosely defined as a process that produces a small data sequence that uniquely can identify a specific piece of sound. Contrasted to a human fingerprint, an audio fingerprint can be thought of as a severely one-way compressed copy of the sound. Some features are desirable in fingerprints, and these features are highly interconnected. Some features are:

Compactness The fingerprints should be small, so they can be easily stored and searched through.

Spatial placement Fingerprints of songs that are related should be less different than fingerprints of songs that are not related using a given distance measure, such as Euclidean distance. This allows for the fingerprints to be used to find similar music, not only identical music, or clustering music into genres.

Robustness A small or medium amount of noise should not affect the generated fingerprint significantly. This also applies to transformations that might occur when using various different compression algorithms. If the fingerprint is not robust, this results in false negatives.

Reliability Songs should preferably never be mislabeled as other songs, an event known as a false positive.

Granularity How much audio is needed to construct a fingerprint. For applications where you have a recording of an entire song, the entire song can be used. In other cases you might only have a few seconds of audio.

Destructiveness An audio fingerprint does not have to be reconstructable into the original, or perceptually similar, audio.

Audio fingerprints are sometimes mistakenly compared to hashes, because hashes by definition produce a small fingerprint for a large amount of data. For example, a very common hash function is MD5, which can be used to verify the integrity of a downloaded file. Hash functions meant for use in cryptography — such as MD5 — produce completely different outputs if even a single bit is changed in the input value. As can be seen in table 2.2, a small change in the text string, changes the MD5 hash completely. This is a property that is undesirable in audio fingerprinting, as it has to be resistant to small changes in the output. Ideally, a small change in the audio will only result in a correspondingly small change in the audio fingerprint.

		Actual condition	
		Same song	Different song
Fingerprint indicates	Same song	True positive	False positive
	Different song	False negative	True negative

Table 2.3: Failure modes.

2.3.1 Use cases

Audio fingerprints can be used in a wide variety of situations, both by consumers and corporations. Consumers may want to figure out if they have a song in their music library, or if they have to purchase it. They may want to search through their music library for songs that are similar to the song they are listening to, or eliminate duplicates in their library. Consumers will also be interested in identifying music that is played on the radio, and there exists several solutions for cell phones. New SonyEricsson cell phones are delivered with a software called TrackID, which allows you to record a few seconds of a song using your cell phone, and send the sample fingerprint to an online service. SonyEricsson has partnered with Gracenote for the fingerprint identification. According to their website, Gracenote has a database of over 80 million tracks [12].

Corporations have different uses. The music industry will be interested in verifying that music shared on file-sharing networks are not copyrighted, and can use audio fingerprints to test against their entire music library. Google wants to choose advertisements based on the music it can identify from background noise in the room you are sitting in [1].

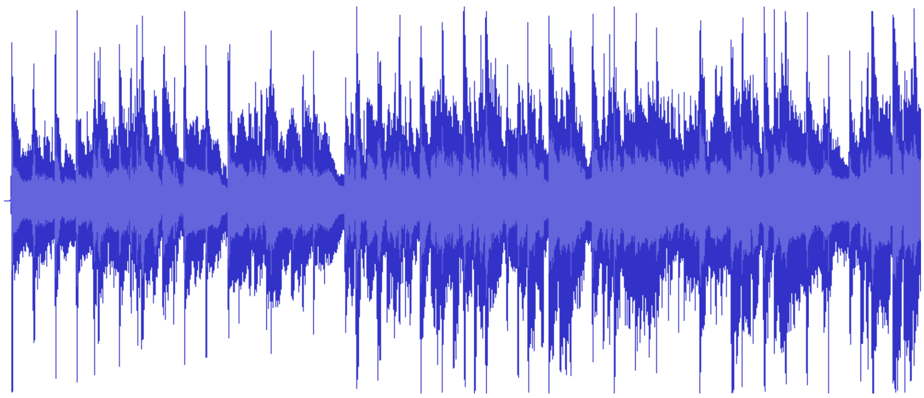
2.3.2 Misclassifications

When comparing two fingerprints from two tracks, there are four possible outcomes. Note that equivalence is context-sensitive.

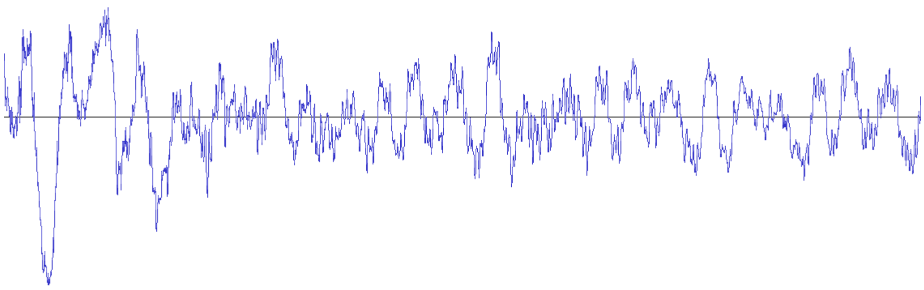
1. The two fingerprints are from the same track, and the comparison indicates that the tracks are equivalent. This is known as a *true positive*.
2. The two fingerprints are from different tracks, and the comparison indicates that the tracks are not equivalent. This is known as a *true negative*.
3. The two fingerprints are from different tracks, but the comparison indicates that the tracks are equivalent. This is known as a *false positive*[5].
4. The two fingerprints are from the tracks song, but the comparison indicates that the tracks are not equivalent. This is known as a *false negative*[6].

Table 2.3 shows these failure modes in relation to each other. False negatives and false positives are the failure modes of a classification. False positives and false negatives can be said to be errors of equal magnitude when looking at audio fingerprints. In the domain of spam filtering a false positive (classifying a non-spam mail as spam) is much worse than a false negative (classifying a spam mail as non-spam) [13].

[5] False positives are called *type I* or α errors by statisticians.
[6] False negatives are called *type II* or β errors by statisticians.



(a) First 30 seconds



(b) 120ms extract

Figure 2.4: Time-based representation of the first 30 seconds and a 120ms extract of "deLillos - Fullstendig oppslukt av frykt". Y-axis is power scale, X-axis is time.

The error analysis in this thesis concerns itself mainly with false positives for two main reasons:

1. Since the output is ranked according to probability, false positives will be very easy to spot in the top results.
2. Because this thesis operates on a real life music collection, the false negatives will be very hard to find in such a large collection.

2.3.3 Analyzing audio

There are multiple ways in which an audio fingerprint can be constructed. Three methods are outlined here: time-based representation, frequency spectrum-based representation and wavelets.

When looking at audio using a normal editor, one usually looks at the *time-based representation* in the shape of a waveform. In such a representation the power is represented on the Y-axis, while the time runs along the X-axis. This representation allows you to see features such as rhythm and beats per minute, BPM, by counting peaks. An example can be seen in figure 2.4.

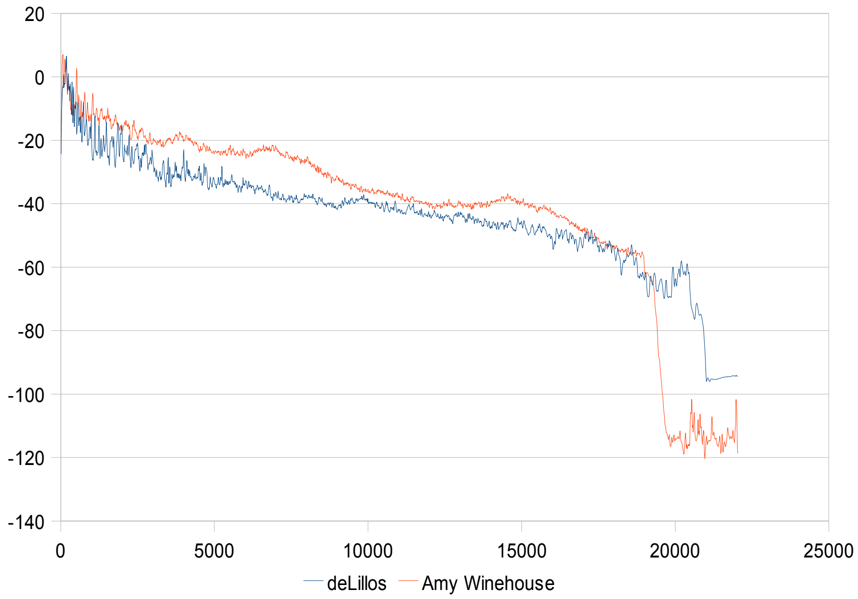


Figure 2.5: The frequency spectrum of the first 23.8 seconds of "deLillos - Fullstendig oppslukt av frykt" vs. "Amy Winehouse - Rehab" using a Hann window function. X-axis is frequency in Hz, Y-axis is power in dB.

The *frequency spectrum-based representation* is usually found using a short-time Fourier transform (STFT), and shows which frequencies are used at what power levels. The STFT is applied to short sequences of sound, depending on the size of the features one attempts to extract. The frequency spectrum can then be reduced using various algorithms to be used as a descriptor. One common algorithm is to establish frequency ranges called bins, and sum the power levels for frequencies in that bin.

The *wavelet representation* uses a wavelet transform to create a range of wavelets that allows examination of frequency components on a suitable scale [14]. Wavelets are also easier to compute than the Fourier transform, with an asymptotic run time of $O(n)$, compared to the fast Fourier transform's $O(n \log n)$.

> How? Construct a function, shift it by some amount, and change its scale. Apply that structure in approximating a signal. Now repeat the procedure. Take that basic structure, shift it, and scale it again. Apply it to the same signal to get a new approximation. And so on. It turns out that this sort of scale analysis is less sensitive to noise because it measures the average fluctuations of the signal at different scales.[14]

2.3.4 Intellectual property

The ownership of music is considered under the broad term *intellectual property* (IP). Buying a CD does not mean that you own the music, but that you own a license to play that music. IP law covers the legal aspects and varies from country to country. Most countries' copyright laws include a term known as *fair use*, which specifies under which conditions one can use a copyrighted work without paying royalty fees.

Audio fingerprints are usually destructive, and is therefore not covered by copyright law, as the result can not be used to infringe on the original work. Non-destructive fingerprints may be covered, and could probably not be shared freely. However, this thesis focuses solely on destructive audio fingerprints.

Several of the audio codecs mentioned and used in this thesis are patented. Norway does not recognize software patents, and such issues are therefore not considered [24].

2.4 State of the art

The field of music information retrieval progresses quickly, and there exists several solutions that are use different descriptors than the ones used in this thesis.

Several solutions have been based directly on the work of Ke, Hoiem and Sukthankar [18], which concentrates on applying computer vision techniques on the spectrogram of audio files. By using a STFT, the spectrogram contains the power of logarithmically spaced frequency bands, much like the process used in Gunderson [15]. Using machine learning they identify a set of filters that perform well. The implementation is freely available under the GPL, and this work has formed the basis for at least two systems. libFingerprint from last.fm is an adaptation of the work presented by Ke *et. al.*, with improvements mainly in the speed of lookups and featuring a cleaner API [17].

Google released a set of papers that details Waveprint, which do not rely on machine learning but rather on wavelets [8, 2]. Wavelets are introduced easily in [14], and allows examination of both the small and large features of a signal, and is considered an alternative to Fourier analysis. None of the software solutions examined in this thesis use wavelets.

Another type of audio fingerprinting models the audio signal by using a sinusoid generated by parameters such as amplitude, phase and frequency, and takes into account the residual noise. The sinusoidal models are extracted using Fourier analysis. Some sinusoidal peaks are selected, mainly those that have high amplitudes and conforms to the sinusoidal model. Betser *et. al.* details the selection process, and finds an increased recall compared to Haitsma [16]. The system can recognize segments as short as 1 second [28].

Some papers state the goal of finding audio recordings that descend from the same musical work, such as Miotto and Larsen [22, 19]. Larsen demonstrates that systems such as fdmf and libFooID cannot be used to identify a more abstract recording reliably. Miotto builds a statistical model of a performance that predicts possible alternative performances. The original performance is segmented and various audio features are extracted from the segments. Hidden Markov models are applied to enable identification.

MIRtoolbox — a package built for Matlab — can be used as an introduction to MIR [35]. It allows easy processing of many common MIR methods, such as finding the mean/square ratio, spectral measures such as MFCC and cepstrum, skewness and centroid. It also has an extensive filter bank, and can quickly be used to test the effects of filters. It can also be used to analyze the pitch of a song, resulting in a range of key candidates, or to find the rhythm in a song.

2.5 Software

For testing our hypothesis that several descriptors can be used to produce a more accurate fingerprint, a set of existing software is used. Throughout this thesis these are referred to as the individual descriptors. The choice of descriptors were made based on the availability of the software.

2.5.1 fdmf

The fdmf (find duplicate music files) package is a small software package designed explicitly to find duplicate music files in a collection. It is written mostly in GNU C, with parts written in Perl. fdmf consists of a program that fingerprints all the audio files, and stores it in an internal database. This database is then used by a second program that matches all fingerprints against all other fingerprints in the database, and prints out the results.

2.5.1.1 Fingerprints

fdmf generates a combined fingerprint of 768 bits, that is in fact three separate descriptors of 256 bits. The first 256 fingerprint-bits are a summary of the energy spectrum of the audio file. The following 256 bits are a summary of the ratio spectrum, a mathematical equivalent to the power spectrum [32]. The final 256 bits are a summary of the twist spectrum.

Rosenfeld details the generation of the descriptors [31, 30]. The descriptors are generated from 250 ms segments of a mono-channel version of the original audio. After applying the STFT, four band energies are calculated for each segment. Equations 2.4 - 2.6 show how these band energies are converted into chunk metrics [31]:

$$cm_1 = be_1 + be_2 + be_3 + be_4 \tag{2.4}$$

$$cm_2 = {}^{(be_3+be_4)}\!/_{(be_1+be_2)} \tag{2.5}$$

$$cm_3 = {}^{(be_1+be_3)}\!/_{(be_2+be_4)} \tag{2.6}$$

A second STFT is applied to the chunk metrics, to limit the time misalignment that might occur in different recordings of files. 256 frequencies are chosen uniformly across the resulting spectra. These values are used to quantize the three chunk metrics into a 768 bit fingerprint.

2.5.1.2 Comparing fingerprints

When comparing two audio fingerprints, fdmf looks at the three descriptors separately. For each of the three descriptors it calculates the Hamming distance between the parts. If the Hamming distance is less than the threshold for that descriptor, the score for those two audio files matching is increased. If all three descriptors has a difference smaller than their threshold, the files are considered to be equivalent. The thresholds are specified at startup as a vector, and if not given, a default threshold vector is used.

2.5.2 libFooID

libFooID has not been documented in any research papers, but its website details the fingerprinting process [27]. The audio is normalized, and combined to a mono recording. Starting silence is skipped, and the first 100 seconds of the audio is processed. The audio is resampled to 8000Hz, and only 90 seconds of the audio is used.

A Hann-windowed discrete Fourier transform is applied to 8192 sample blocks, resulting in 87 frequency spectra frames. The frequency spectrum is partitioned into the Bark scale using a modified version of Traunmüller's formula [34]:

$$z = \left[\frac{26.81f}{1960 + f}\right] - 0.53 \tag{2.7}$$

The first Bark band is ignored, and the final band is enlarged, which leaves 16 Bark bands.

$$z_2 = \begin{cases} z + 0.15 \times (2 - z) & z < 2 \\ z & 2 \leq z \leq 20.1 \\ z + 0.22 \times (z - 20.1) & z > 20.1 \end{cases} \tag{2.8}$$

A least square regression line fit is used on the spectral data, and the correlation coefficient and the dominant spectral line per frame is stored using 2 and 6 bits, respectively. The fingerprint is stored using 424 bytes, and includes some metadata, such as the length of the song.

2.5.2.1 Comparing fingerprints

Comparing fingerprints is done by using a simple Hamming distance for the dominant spectral lines, and uses a quadratically weighted Hamming distance for the correlation coefficient.

2.5.3 Gunderson's descriptors

A set of descriptors as presented in Gunderson's master thesis was also used in this thesis [15]. The thesis implements a variety of basic descriptors, such as the spectral centroid, play time, rate of zero crossings (with and without Schmitt triggering), steepness, and the mean/square ratio. Gunderson also implements mel frequency cepstral coefficients (MFCC) and floor-1 cepstral coefficients (F1CC), that are included in this thesis.

The descriptors implemented by Gunderson are briefly explained below, and are explained in depth in this thesis itself. Common to all of the descriptors implemented, with the exception of song length, is that they only look at the first 30 seconds of a song.

2.5.3.1 Centroid

A spectral measure that finds the "center of mass" for the audio signal. The audio signal is transformed to the frequency domain by a discrete Fourier transform.

2.5.3.2 Song length

The song length was implemented as the length of the track in seconds, but could also be the number of samples if the sample rate is equal in all songs.

When using lossy compression methods with variable bit rates, the entire track may have to be decoded to be able to determine the song length, if this is not given in any of the metadata. The very common MP3 file format, when encoded using a variable bit rate, does not store the song length automatically as metadata, and as such the song length can only be estimated until the entire file has been decoded.

2.5.3.3 Mean/square ratio

The mean/square ratio measures the average distance from zero, calculated after the audio signal has been normalized. Equation 2.9 is from [15].

$$r = \frac{avg\left(|x|\right)}{RMS} = \frac{\frac{\sum |x_i|}{N}}{\sqrt{\frac{\sum x_i^2}{N}}} = \frac{\sum |x_i|}{\sqrt{N \sum x_i^2}} \tag{2.9}$$

2.5.3.4 Steepness

Steepness is a frequency measure that is less affected by noise than the zero crossing rate. It is computed by taking the mean of the numerical derivative of the audio signal, as in equation 2.10, again from [15].

$$s = \frac{1}{N-1} \sum_{i=1}^{N-1} |x_i - x_{i-1}| \tag{2.10}$$

2.5.3.5 Rate of zero crossings (with and without Schmitt triggering)

The rate of zero crossings counts the number of times the signal changes sign divided by the number of samples. Schmitt triggering defines a guard band around the x-axis that needs to be crossed for a zero crossing to be counted. Schmitt triggering increases the resilience to noise.

2.5.3.6 Mel frequency cepstral coefficients

Mel frequency cepstral coefficients (MFCC) is based on the mel scale that attempts to mimic the human auditory system's perception of varying pitches at constant loudness. The frequency is transformed to the mel scale using the following equation:

$$B\left(x\right) = 1127 \ln\left(1 + \frac{x}{700}\right) \tag{2.11}$$

Related frequencies are linearly grouped into spectral bands on the mel scale. MFCC outputs several sets of individual descriptors, of which the important ones are mfcc_avg and mfcc_delta_avg. mfcc_avg is titled the *first zero moment*, and is calculated separately for the 32 coefficients:

$$\hat{\mu}_m = \frac{1}{N} \sum_{i=0}^{N-1} c_{m,i} \tag{2.12}$$

mfcc_delta_avg is calculated as the average (as in equation 2.12) of the first derivative of the coefficients.

$$\triangle c_{m,i} = |c_{m,i+1} - c_{m,i}| \tag{2.13}$$

2.5.3.7 Floor-1 cepstral coefficients

Floor-1 cepstral coefficients (F1CC) is the descriptor tested by Gunderson to determine if audio masking techniques used in lossy compression could be reliably used when creating fingerprints. Floor-1 is based on the Ogg Vorbis psychoacoustical model, which models how certain weak tones are hidden by louder nearby tones [11, 36].

The calculation of fingerprints is otherwise equivalent to MFCC, and the descriptors named f1cc_avg and f1cc_delta_avg are considered by Gunderson to be the most accurate.

2.5.3.8 Comparing fingerprints

Gunderson's descriptors outputs a vector of floating point values. To compare two fingerprints to each other Gunderson experiments with using both the Euclidean distance and Mahalanobis distance, the latter of which is scale-invariant. For this thesis, only the Euclidean distance has been used. For details about how the distance measure is converted to a probability of match, see 3.3.3.1.

Chapter 3

Method

This chapter describes the work that was done to combine the three solutions chosen for this thesis. It outlines the structure of the value-added software that was written for this thesis.

3.1 Music collection

The music collection used consists of two merged personal collections, with contents in WMA, MP3 and Ogg Vorbis formats, using high bit rates. The collection contains in total 9536 music files, of which an estimated 2200 combinations of two files (see 3.1.3) are considered duplicates. In total, the files consume 57GB of disk space. It would take 26 days and nights to play at normal speed. The collection is strongly biased to contain MP3 files over WMA files, with 8429 MP3 files, 1027 WMA files and 80 Ogg Vorbis files.

The sample collection is meant to mimic an average personal collection, and no attempts were made to remove duplicate songs or albums when combining the two separate music collections. A listing of the albums in the collection can be found in appendix C. The collection is stored in a folder hierarchy with the naming scheme <artist>/<album>/<song>, but this naming scheme is not strictly enforced throughout the collection.

An alternative music collection could be created by using perfectly preserved originals, encoded in a lossless format, and re-encoding these into several popular formats. This approach was not used for two main reasons:

1. No access to a large lossless music collection.

2. In the real world, a collection does not usually contain several copies of the same song in different formats. The results would be skewed towards how resistant an algorithm is at ignoring encoding artifacts.

3.1.1 Equivalent music

The question of what defines two songs as being equivalent has never been settled, and is usually considered to be a matter of personal preference, context and usage scenario. To examine this more closely, a terminology is needed to

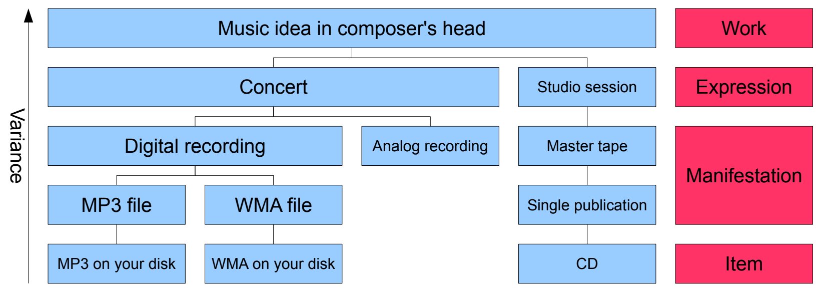


Figure 3.1: Simplistic illustration of the FRBR terminology as applied to the lifespan of a fictional song.

describe music. IFLA presents the FRBR[1] model, which can be used as such a terminology [25, 19]. It defines four entities that can be used to describe music. Originally it was intended to be used for more general bibliographic records to describe relationships between any items that can be stored in a library.

Behind every piece of music is the intellectual idea of the music, as imagined by the composer. This is the essence of that song, and is referred to as the *work*. The work can be expressed in a variety of ways, i.e. the composer might write sheet music for the song, or performing the song using an instrument. These are both considered *expressions* of that *work*. If someone uses a microphone to record the composer performing the song, the recording constitutes a *manifestation* of the expression.

> The entity defined as *manifestation* encompasses a wide range of materials, including manuscripts, books, periodicals, maps, posters, sound recordings, films, video recordings, CD-ROMs, multimedia kits, etc. As an entity, *manifestation* represents all the physical objects that bear the same characteristics, in respect to both intellectual content and physical form. [25]

This means that our composer might write down the sheet music for the song, but if the sheet music is printed into many copies, those copies as a collection is the manifestation. A single copy is known as an *item*. The model also allows for works that are derivatives of each other, or when the changes are smaller, as different expressions of the same *work*.

In music a remix of a song can be said to be a derived *expression* of the original expression. In much the same way, a live performance of a song is a different *expression* than the original studio recording of a song. For the purposes of this thesis, live performances and remixes are considered as different enough to be nonequivalent with the original recording.

In several genres of music it is common to issue remastered versions of a song. When remastering, a sound technician has gone back to the original recording and removed noise or otherwise enhanced the audio. A remastering of a song is

[1]Functional Requirements for Bibliographic Records

considered a different *manifestation* of a song, and the audible difference from the original is sometimes hard for a human to detect. In this thesis such a remastered version is considered equivalent with the original. When digitally storing music, different encodings can be used. These encodings usually introduce various artifacts into the audio. Different encodings are also considered different *manifestations* of a song, and are considered to be equivalent in this thesis.

In figure 3.1 the FRBR hierarchy is illustrated with an example. The song is thought of by the composer, who plays it at a concert. At the concert it is recorded in both digital and analog format. For some reason, the analog recording gets ignored, while the digital recording is encoded into two formats (MP3 and WMA) and is distributed. The composer also records the song in a recording studio, resulting in a master tape, which subsequently is produced into a CD, which you can purchase copies of in your local record store.

When determining equivalence in this thesis, where the songs are not related in any of the ways mentioned above, the deciding factor for determining equivalence has been whether or not a human would say that the songs sound the same after having listened to the song once.

3.1.2 What are duplicate pairs?

For this thesis we define *a duplicate pair* as two tracks that cannot be easily distinguished from each other by a human listener. This definition might vary from study to study, and our definition is dependent on that this thesis chose a real-life music collection. In FRBR terminology, this will usually mean that the tracks are the same or derived manifestations. Yet in some cases, they might not be related manifestations, but still the same expression.

Because some of the descriptors used do not analyze more than 30 seconds of a track, a decision was made that two tracks will also be considered equivalent if they are audibly equivalent to a human for the first 30 seconds, as long as their total length is approximately equal.

This definition of a duplicate pair does not consider the fact that three or four tracks may be indistinguishable from each other, and that such clusters will in fact significantly impact the numbers of duplicate pairs. Assuming a cluster of four tracks that are related to each other, a total of 6 duplicate pairs can be found, as illustrated in figure 3.2. The number of duplicate pairs found for a cluster of n equivalent tracks is given by $\binom{n}{2}$, a number that increases quickly for large clusters.

3.1.3 Estimating duplicate pair count

A very simple process was used to estimate the number of duplicate pairs in the collection. The files in the collection have names that roughly corresponds to the title of the song. The file name is split into individual words, ignoring everything but letters and punctuation that naturally occurs within a word. Words shorter than three letters are ignored.

The tokenized version of two file names are then compared to each other, and the number of tokens found in both file names is counted. This value is then divided by the number of tokens in the longest file name, and if the result is

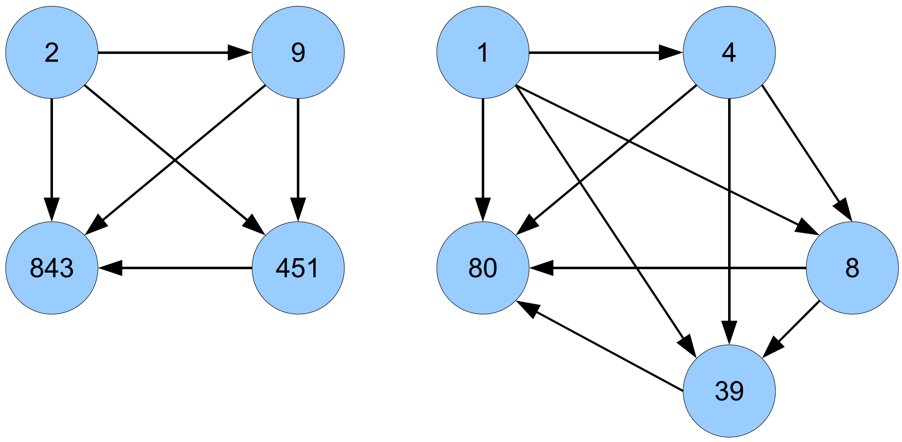


Figure 3.2: The edges represent a single duplicate pair, while the circles represent a track. The numbers indicate the identifier of the song, and note that the arrows always direct to a larger identifier. Left: the 6 combinations found when 4 identical songs are found. Right: the 10 combinations found when 5 identical songs are found.

Algorithm 1 Pseudo-code for checking if two file names are close enough to be considered a potential duplicate pair.

```
if artistname1 <> artistname2:
    return false;
tokens1 = tokenize(filename1)
tokens2 = tokenize(filename2)
all_tokens = merge(tokens1, tokens2)
common_tokens_count = 0
for token in all_tokens:
    if token in tokens1 and token in tokens2:
        increase(common_tokens_count)
tokens_count = max(count(tokens1), count(tokens2))
if tokens_count < 1:
    return false;
ratio = common_tokens_count / tokens_count
return ratio > 0.6
```

greater than 0.6, and the files are from the same artist (indicated by the folder they are located in), the two file names are considered a match.

The number gained from this method is likely to be grossly over-dimensioned because songs might have the same file name even though they are not audibly similar. For example, a significant number of music files in the collection contains live versions of songs which are not counted as duplicates of a studio recording. This error is compounding, as it will increase the count greatly when there are clusters of matching songs. This method of estimation resulted in 6991 combinations of two files listed as duplicate pairs.

It is also very likely that the comparison of just file names will produce a number of false positives. To estimate the amount of false positive, 500000 random combinations of file names were sampled. Using the method above produced 129 combinations that it believed to be the same song. When verifying these 129 combinations by hand, only 47 were likely to be equivalent based on the file name. We can therefore conclude that the actual number of hits is less than 35 percent of the number of hits, ergo around 2400 combinations.

When sampling all the combinations were the file names match and checking for the word "live" in the two file paths in the combination, around 10 percent of the combinations turned out to contain one song from a live recording and one from a studio recording. After this very simple check, we are left with an estimate of 2200 duplicate pairs.

3.1.4 Verifying duplicate pairs

Since the test is based around a non-controlled, real life collection of music, knowing in advance how many and which combinations should be considered duplicate pairs is not trivial. Verifying the identified duplicate pairs therefore becomes a highly labor-intensive manual process. However, since we know that we have approximately 2200 duplicate pairs (see 3.1.3), the amount of work is somewhat reduced.

When the various averages had been computed, an administration interface was set up to verify each duplicate pair. To aid in the process, the same code that was used in 3.1.3 to find duplicate pairs was used to highlight combinations that based on their file names could be actual duplicates. The first[2] few thousand potential duplicate pairs were then read through, and a human indicated to the system which were actual duplicate pairs.

This process might be error prone, and surely some of the duplicate pairs verified by a human will be wrong. The administration interface showed potential duplicate pairs with high rankings from both separate descriptors and the combined solutions, so the duplicate pairs that were verified is concentrated heavily in the top ranked results for each combined solution.

An assumption was made that perceived equality when talking about music is transitive. That is, given that song A is a duplicate of song B, and song B is a duplicate of song C, we can say that A is a duplicate of song C. This assumption might not hold in all cases, where subtle changes may make song C more different from A than B was. For a more concrete, yet fictional example, take three different recordings of the Billie Holiday song "Georgia On My Mind", titled "Take 1", "Take 2" and "Take 3". The "Take 1" and "Take 2" recordings

[2] as ranked by the individual descriptors

might be similar enough that they can be listed as duplicate pairs, and likewise for "Take 2" and "Take 3", while the "Take 1" and "Take 3" might be different enough for them not to be considered a duplicate pair. Please note that the example is fictional, and that no examples of non-transitive duplicates are known to exists in our collection.

In total 2057 duplicate pairs were verified.

3.2 Descriptor architecture

The combined solution should be able to quickly find equivalent audio files in a large collection. Several aspects have to be taken into consideration when designing the solution.

Decoding The most time-consuming process when creating the fingerprints is decoding the audio, and as such, the solution is designed to only decode the audio once and keep the audio data in memory. Then all the fingerprinting solutions can fetch the data from main memory. Once the fingerprints has been generated, the decoded audio can be discarded.

Output The systems should all be able to compare two fingerprints and produce a probability that the files are equivalent. This makes it possible to average the probabilities from multiple engines.

3.3 Software

Three different software packages where combined to form a single software system, that in one pass generates fingerprints for a set of audio files. The combined solution needs every system to output a probability of two songs being equivalent, and some of the systems had to be modified to output such a probability.

3.3.1 fdmf

By examination of the source code, fdmf in its original version returns only a list of the audio files that has a Hamming distance less than the threshold vector. The Hamming distance counts the number of positions in which two signals differ. We used the Hamming distance d_i and the threshold value t_i for each fingerprint part i to generate a probability P according to equation 3.1.

$$P_i(d_i, t_i) = 1 - \frac{d_i}{t_i} \tag{3.1}$$

The probability measure thus has a resolution of t_i, and ensures that the entire range will be used. The threshold values are used since a threshold of 255 would mean that the fingerprints are complete inverses of each other. fdmf's default threshold values were used.

Descriptor	t_i
Centroid	50
Song length	5
Mean/square ratio	0.005
Steepness	25.0
Rate of zero crossings	2000
Rate of zero crossings (w/ Schmitt triggering)	1000
MFCC average	40
MFCC delta average	3.0
F1CC average	20
F1CC delta average	5

Table 3.1: Thresholds for various descriptors by Gunderson.

3.3.2 libFooID

Software was written around libFooID to create fingerprints from PCM audio data, since such software was not available for Linux with source code. Source code for our software can be found in appendix B. The software stores the fingerprints in the database table tblFingerprint, as described in section 3.4.1.

The software supplied with libFooID to compare fingerprints was adapted to automatically compare all the fingerprints in the database against each other, in much the same way as with fdmf.

3.3.3 Gunderson's descriptors

The fingerprinting software was used without modifications to generate fingerprints for all of the audio files. A separate parser was then written to read the files produced into our fingerprint table (see 3.4.1). A script compares all of these fingerprints to each other, generating a probability that the two files match. The probability is calculated as in equation 3.2, where $d(f_1, f_2)$ is the Euclidean distance between the fingerprints f_1 and f_2, and t_i is the minimum Euclidean distance required for that descriptor according to table 3.1.

$$P_i(f_1, f_2) = 1 - \frac{d(f_1, f_2)}{t_i} \tag{3.2}$$

3.3.3.1 Determining thresholds

The thresholds presented in table 3.1 was determined by pulling a random sample of 10 000 Euclidean distances for a given descriptor, and then sort the distances ascendingly. Due to the enormous amount of combinations we are only interested in Euclidean distances that are significantly smaller than the rest. With 9536 songs in our database, there are $\binom{9536}{2} = \frac{9536 \times 9535}{2} = 45462880$ possible ways to combine two files, and we are only interested in identifying the duplicate pairs in our collection, which we have estimated to be around 2200. We are therefore only interested in 0.005 percent of the combinations presented.

Figure 3.3 shows a plot of the samples taken from the mfcc_avg descriptor. By looking at similar plots for the different descriptors, we determined appropri-

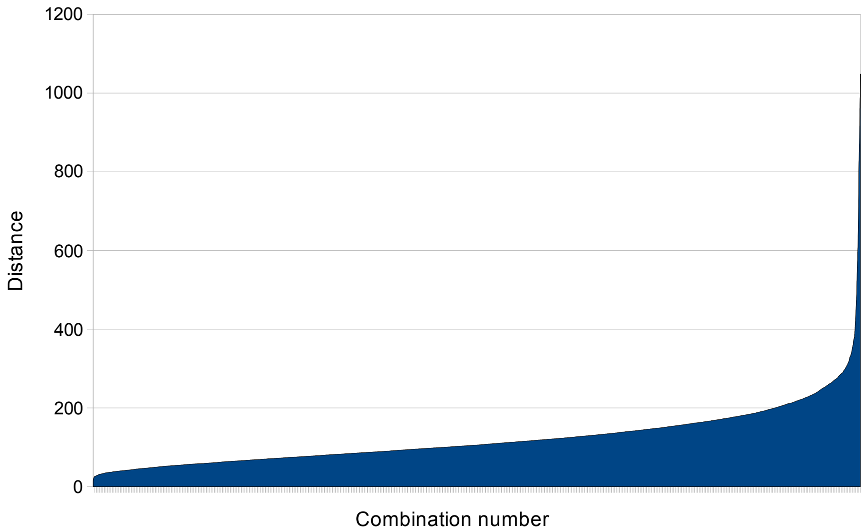


Figure 3.3: 10000 random Euclidean distances sampled from the mfcc_avg descriptor.

ate thresholds that at would result in around 50000 matches for each descriptor. 50000 was chosen to allow for a large safety margin.

3.4 System architecture

The system uses a MySQL database to store intermediate data, and several processing steps to combine the data in meaningful ways.

3.4.1 Database architecture

The fingerprints and scores from the various descriptors are fed into a common database to allow for easy querying. The database structure is presented in Figure 3.4.

It consists of four tables. Every file that is fingerprinted is present in tblFile, where the path to the file is stored, along with the size of the file. A unique identifier is assigned to each file. For quick lookups the MD5 sum of the file's path is used as the index. Since all of the paths have a common prefix and a MD5 causes a shorter prefix lookup in the index.

tblType contains a list of the different descriptors used. Since fdmf produces three separate fingerprints, it has three entries in this table. tblFingerprint contains the fingerprints for files for those descriptors that do not have their own internal fingerprint database. tblScore contains the likelihood that two files are equivalent when comparing with a given descriptor. If a descriptor finds no similarity between two files, it will not insert a row into tblScore to save calculation time and space. The subsequent calculation will account for missing 0-score rows. tblSummedScore is used for storing the final likelihood

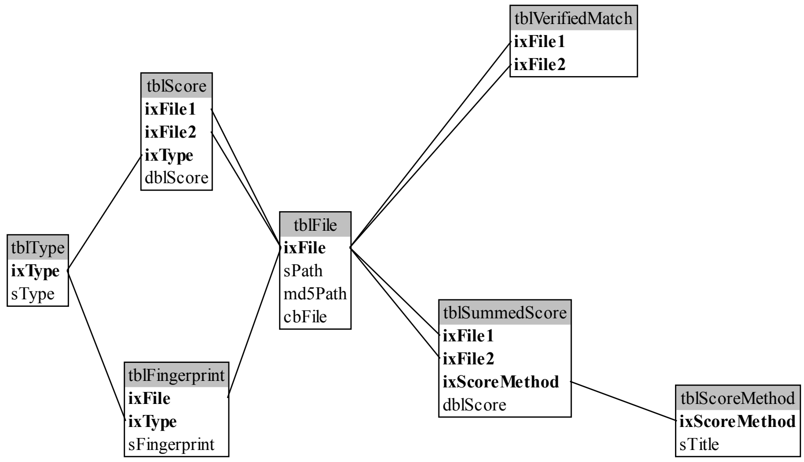


Figure 3.4: Database architecture of the combined solution. Primary keys are marked with a bold font.

that two files are equivalent, using different combination methods, as outlined in section 3.5.

It is worth noting that for all the tables that contain the two identifiers ixFile1 and ixFile2, it is always the case that ixFile1 is smaller than ixFile2. This assertion holds as long as all descriptor matching is guaranteed to be symmetric. All the descriptor matching is done using the Euclidean distance. The Euclidean distance is a metric, and metrics are symmetric by definition [37].

3.4.2 Data flow

The combined system modifies the software packages used to input data in several processing steps. There are three separate fingerprint processes that are run on the audio data; fdmf, libFooID and Gunderson's descriptors. fdmf uses it's own internal database to store the fingerprint data, while both libFooID and Gunderson's descriptors relies on our MySQL database.

There are three comparison processes, that compares fingerprints for all files against each other using their descriptors, and stores the results in tblScore. A single process combines these results (as presented in 3.5), and enters the results into tblSummedScore. The entire process is outlined in figure 3.5.

3.5 Combining results

Several methods exists for finding a central tendency among a range of values, and the goal of this thesis is to highlight the merits of each of these methods. In the following, the likelihood for two songs matching as given by descriptor i is designated p_i.

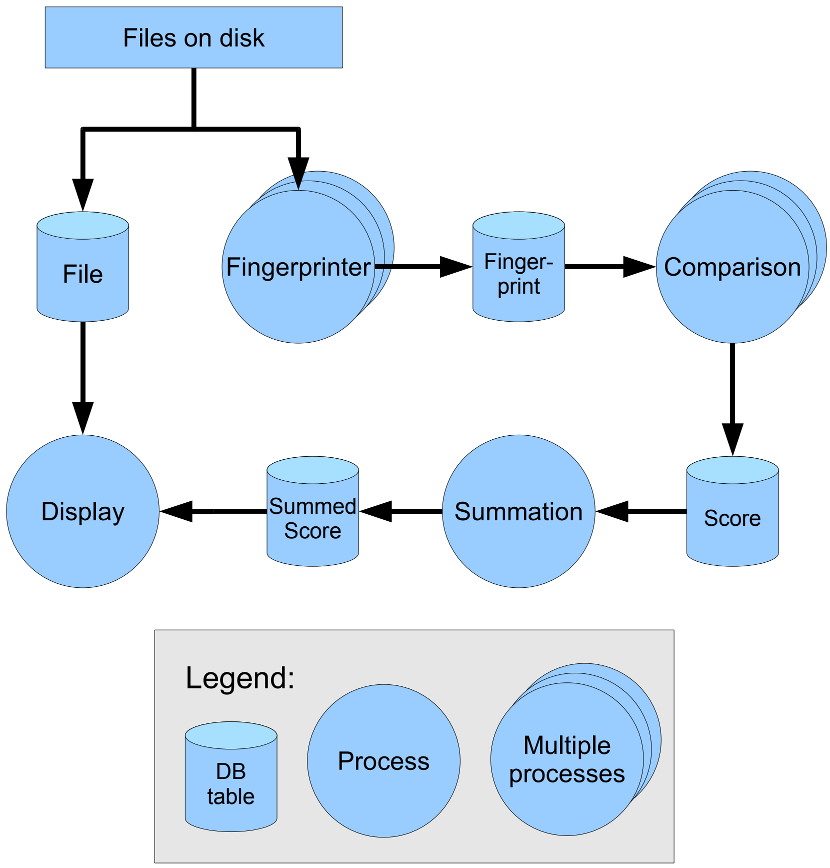


Figure 3.5: The flow of data through the system.

In addition to a likelihood measure given by various averages, the standard deviation is calculated when we are dealing with statistical measures. The standard deviation using average x is given in equation 3.3, where P_x is the probability given by e.g. the arithmetic mean or the truncated mean. It is worth noting that calculating the standard deviation is very similar to taking the root-mean square, except that you sum the squares of deviation. Also worth noting is that the standard deviation can not be compared directly unless the same averaging method is used.

$$\sigma_x = \sqrt{\frac{1}{I}\sum_{i=0}^{I}(p_i - P_x)^2} \tag{3.3}$$

Results are then sorted by descending values of probability, and then by ascending values of standard deviation. This ensures that high-probability duplicate pairs are sorted first, and when two duplicate pairs have the same probability, the standard deviation is the tie-breaker. If both the probability and standard deviation match, the files are sorted according to which duplicate pair's first file was first entered into the system.

Some of these methods are highly affected by outliers, and some descriptors produces lots of outliers. Therefore, in addition to running on the full set of data, some where also run on a set of the top performing descriptors, while ignoring the other descriptors. These were the arithmetic mean, the root-mean square, the truncated arithmetic mean and the Bayesian classifier. The descriptors that were chosen for this set was fdmf_0, fdmf_1, fdmf_2, libfooid, mfcc_avg, mfcc_delta_avg, mfcc_f1_avg and mfcc_f1_delta_avg. The median was used only for the top performing descriptors.

3.5.1 Arithmetic mean

The arithmetic mean (AM), commonly referred to as the *mean*, is the simplest and most common averaging method. It is one of several methods to indicate the central tendency of a range of numbers, and is calculated as in equation 3.4 .

$$P_{am} = \frac{1}{I}\sum_{i=0}^{I} p_i \tag{3.4}$$

The arithmetic mean is a good measure for distributions that have a small number of outliers. Outliers can significantly impact the result. A high standard deviation would usually indicate that the arithmetic mean is unsuitable.

3.5.2 Root-mean square

The root-mean square (RMS) of a set of values is the square root of the mean of values squared. It is most useful when values shift from positive to negative, and is well known in electrical engineering and with audio enthusiasts under the abbreviation RMS.

$$P_{rms} = \sqrt{\frac{1}{I}\sum_{i=0}^{I} p_i^2} \tag{3.5}$$

3.5.3 Weighted arithmetic mean

The weighted arithmetic mean (WAM) is an averaging method used when different values are assigned different weights. If we know that one fingerprinting method has a higher likelihood of producing false positives, we can reduce it's weight, and thus reduce it's final impact on the score. For the weighted arithmetic mean, each likelihood p_i has an associated weight w_i.

$$P_{wam} = \frac{\sum_{i=0}^{I} w_i p_i}{\sum_{i=0}^{I} w_i} \tag{3.6}$$

The arithmetic mean is a special case of the weighted mean, where all the weights are equal. The weights were adjusted manually for some runs, based on the reputation of individual descriptors.

3.5.4 Weighted root-mean square

A weighted version of the root-mean square measure was derived from the ordinary RMS measure after seeing the performance of the weighted arithmetic mean. Using the weights from weighing by performance and the second set of manual weights, equation 3.7 was used to computed a weighted root-mean square (WRMS).

$$P_{rms} = \sqrt{\frac{\sum_{i=0}^{I} w_i p_i^2}{\sum_{i=0}^{I} w_i}} \tag{3.7}$$

The formula was adapted from the root-mean square for this thesis, but has been used in a similar form in e.g. [3].

3.5.5 Truncated mean

A truncated mean discards some data values to get rid of outliers. The truncated arithmetic mean was used to get rid of the worst outliers, by removing an equal number of values on each end of the input data, and calculating the arithmetic mean of the remaining data. We only truncated two values, one from each side of the data set.

3.5.6 Median

The median is a simplistic measure that does not work well at all when all the descriptors are present. With a large number of descriptors returning perfect matches for non-related songs, the median will be more or less random in many situations.

When working with the set of known good descriptors, it could very well indicate a tendency among the values. The median is not affected as considerably by outliers as other averages.

3.5.7 Naïve Bayes

Bayesian analysis lends itself elegantly to the classification problem and is used with a high degree of success in spam filters. By starting with Bayes theorem, a formula for calculating the probability that two songs match given a set of clues will be derived. This approach is called naïve Bayes (NB) [21].

A prerequisite for using Bayesian analysis is that the scores returned by the varying engines is the actual probability of the two songs matching. It is a well known fact that this assumption does not hold for many of the descriptors, and as such, the Bayesian analysis has been limited to only encompass the 8 best-ranked descriptors from weighing by performance (see 3.5.9).

In the following, $P(M)$ is the probability of a correct match. $P(C_i)$ is the probability for the two songs matching given by descriptor i. The formulas assume only two clues, while the final formula is easily expandable.

$$P(M|C_1 \cap C_2) = \frac{P(C_1 \cap C_2|M)\,P(M)}{P(C_1 \cap C_2)} \tag{3.8}$$

$$= \frac{P(C_1 \cap C_2|M)\,P(MS)}{P(M)\,P(C_1 \cap C_2|M) + P(\neg M)\,P(C_1 \cap C_2|\neg M)} \tag{3.9}$$

Bayes' theorem was used to get to 3.8, and use the usual expanded form of Bayes' to get 3.9. To continue the assumption that our clues are independent is needed, which turns the algorithm into a naïve Bayes algorithm. We can now substitute $P(C_1 \cap C_2|M) = P(C_1|M)\,P(C_2|M)$:

$$= \frac{P(C_1|M)\,P(C_2|M)\,P(M)}{P(M)\,P(C_1|M)\,P(C_2|M) + P(\neg M)\,P(C_1|\neg M)\,P(C_2|\neg M)} \tag{3.10}$$

Applying Bayes' theorem three times gives, after some calculation:

$$= \frac{\frac{P(M|C_1)P(M|C_2)}{P(M)}}{\frac{P(M|C_1)P(M|C_2)}{P(M)} + \frac{P(\neg M|C_1)P(\neg M|C_2)}{P(\neg M)}} \tag{3.11}$$

Substituting $a = P(M|C_1)$, $b = P(M|C_2)$ and $S = P(M)$ leaves a simpler formula:

$$L(F_1, F_2) = \frac{\frac{ab}{S}}{\frac{ab}{S} + \frac{(1-a)(1-b)}{1-S}} \tag{3.12}$$

This formula expands as you might expect:

$$L(F_1, F_2) = \frac{\frac{abc}{S}}{\frac{abc}{S} + \frac{(1-a)(1-b)(1-c)}{1-S}} \tag{3.13}$$

Depending on the number of clues, a generalized formula is presented below:

$$L(F_1, F_2) = \frac{\frac{\prod C_i}{S}}{\frac{\prod C_i}{S} + \frac{\prod 1-C_i}{1-S}} \tag{3.14}$$

In our database, a total of 31 million potential duplicate pairs will be evaluated using this algorithm. An estimated 2200 correct duplicate pairs exist. This means $P(M) = S \approx \frac{2200}{31000000} = 8 \times 10^{-5}$, and that the probability for picking a correct duplicate pair from the potential duplicate pairs is 0.008 percent. However, when using this value, the formula proved to be to extremely sensitive to outliers. After repeated experimentation the value was therefore manually adjusted to 0.8. The value had to be adjusted because otherwise a tremendous amount of evidence would be required for the naïve Bayes algorithm to classify two songs as a match. A value of 0.8 results in the values being more spread out across the available spectrum.

3.5.7.1 Problems with naïve Bayes

There are two main problems with Bayesian analysis as applied to our system. The first is the reason for excluding descriptors. As an example, assume that we have three descriptor scores, length with a probability of 1.0, fdmf_0 with a score of 0.2 and mfcc_avg with a score of 0.1. Since the probability of the two songs being similar is 1 as given by the length descriptor, the combined probability is also 1. This means that any descriptor producing a perfect score will completely distort the result. Inversely, if any descriptor returns zero, the combined score will also be zero.

Two solutions can be used to prevent this problem. First, the scores are capped at both ends, and is always between 0.01 and 0.99. This ensures that a single descriptor cannot override the rest of the results. Second, one can pick scores from accurate descriptors liberally, which we are doing when we are selecting the top performing descriptors.

The second problem lies with the assumption that the scores from all the descriptors are statistically independent. This assumption is likely not to hold, as the algorithms used by the various descriptors are similar. This problem is frequently ignored when writing Bayesian classifiers and is why this is considered a *naïve* Bayes algorithm.

3.5.8 Discarded approaches

Several averaging methods were not selected for this approach, mostly due to the the large quantity of zero values in the data set. For example, the harmonic mean, defined as in equation 3.15 will result in division by zero, and distort the result. The harmonic mean is therefore not applicable in our tests.

$$P_{hm} = \frac{I}{\sum_{i=0}^{I} \frac{1}{p_i}} \tag{3.15}$$

The geometric mean is another average where the I-th root of the product of values is taken. Equation 3.16 shows the calculation, but due to the multiplication of probabilities, we will always get zero probability if only one of the results is zero.

$$P_{gm} = \sqrt[I]{\prod_{i=0}^{I} p_i} \tag{3.16}$$

3.5.9 Determining weights

Some of the descriptors tested produced a larger number of potential duplicate pairs than others. Some even two orders of magnitude apart. It is likely that a potential duplicate pair from a descriptor that produces few potential duplicate pairs, will be more likely to be correct than a potential duplicate pair from a descriptor that produces one hundred times as many.

By this assumption, four different ways of calculating the weights were devised. In one run, the weight w_i was calculated using the number of non-zero scores n_i, compared to the maximum number of non-zero scores produced by a single descriptor n_{max}. It is shown as equation 3.17. The multiplication by 2, is to ensure that the lowest assigned weight is 0.5, instead of 0.

$$w_i = 1 - \frac{n_i}{2 \times n_{max}} \tag{3.17}$$

This method, referred to as weighing by count, only looks at the number of potential duplicate pairs found to determine which descriptors are trustworthy. So, if a descriptor produces a lot of very low-ranked potential duplicate pairs (e.g. with a score of 0.01 for 99% of the comparisons), it will be ranked as unreliable, even if it is not. Another weighing scheme looks at the sum of the of scores instead, so that a descriptor that produces very many low-ranking results will not be penalized as harshly, while a descriptor that produces many high-ranking results compared to the amount of results will be penalized harder.

More specifically, the weight for a given descriptor i was calculated using the sum of scores s_i, as presented in equation 3.18, where s_{max} is the highest s_i.

$$w_i = 1 - \frac{s_i}{2 \times s_{max}} \tag{3.18}$$

The third method uses the arithmetic mean of the scores given by one descriptor. A high average is assumed to indicate that the descriptor is less trustworthy. Equation 3.19 shows the calculation, where $s_{j,i}$ is the j-th score for descriptor i, and J_i is the number of scores for descriptor i.

$$w_i = 1 - \frac{\sum_{j=0}^{J_i} s_{j,i}}{J_i} \tag{3.19}$$

The most accurate way to get a measure of the usefulness of a descriptor, is to look at its actual performance. After having finished the database of verified duplicate pairs, the performance was measured using the false positive rate in the highest-ranked one thousand matches. This method was titled weighing by performance.

Using these equations on the fingerprint database produced weights as can be seen in figure 3.6. It is apparent from weighing by performance that a set of the descriptors are nearly useless, an indication that weighing by average also seems to support. Only weighing by performance recognizes mfcc_delta_avg as a good indicator, with the exception of weighing by average. Almost all of these methods seem to agree that fdmf_1 is a bad estimator as well.

In addition to the automatically computed weights, several sets of weights were created manually, based on the performance of the different weighing schemes. These weights are shown in figure 3.7. The first set of manual weights was chosen before the relative merits of the different descriptors were known, and was based on their reputation. Manual weights method 2 and 3 are based on weighing by performance, where weights less than 0.05 and 0.1 respectively are ignored. This makes the manual weights comparable to the other means when only a chosen set of descriptors is used, as it is the same set of descriptors that is used.

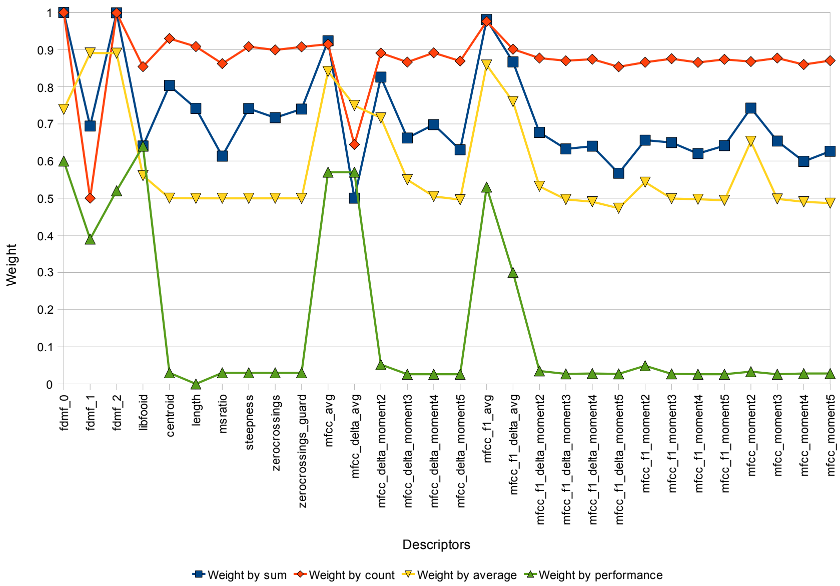


Figure 3.6: Weights as generated by equation 3.17, 3.18, 3.19 and weighing by performance. Note that the weights are relative to weights within its series, and as such, can not be compared directly to weights given by another series.

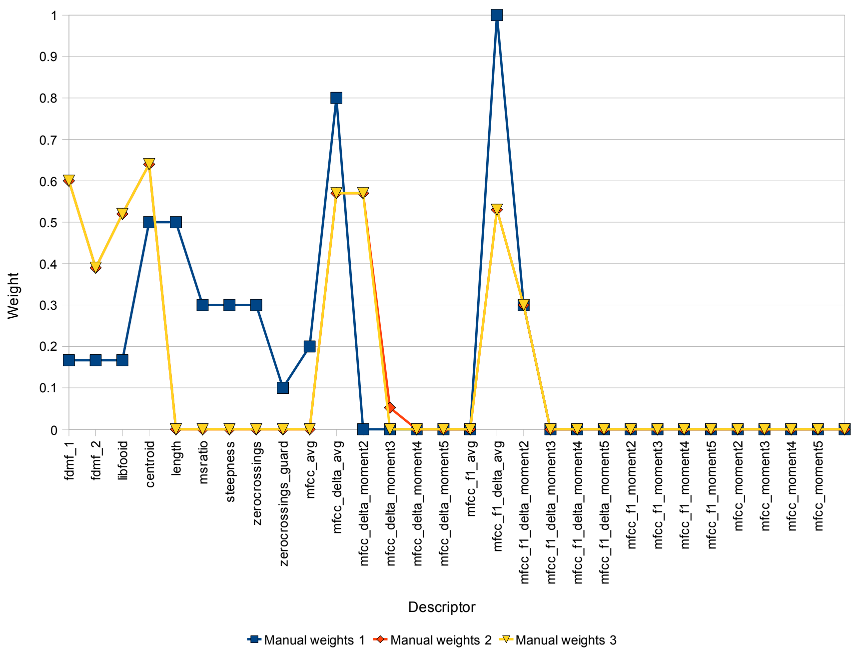


Figure 3.7: Three sets of manual weights.

Descriptor	Weights						
	Sum	Count	Avg.	Perf.	M. 1	M. 2	M. 3
fdmf_0	1	1	0.74	0.6	0.17	0.6	0.6
fdmf_1	0.69	0.5	0.89	0.39	0.17	0.39	0.39
fdmf_2	1	1	0.89	0.52	0.17	0.52	0.52
libfooid	0.64	0.85	0.56	0.64	0.5	0.64	0.64
centroid	0.8	0.93	0.5	0.03	0.5	0	0
length	0.74	0.91	0.5	0	0.3	0	0
msratio	0.61	0.86	0.5	0.03	0.3	0	0
steepness	0.74	0.91	0.5	0.03	0.3	0	0
zerocrossings	0.72	0.9	0.5	0.03	0.1	0	0
zerocrossings_guard	0.74	0.91	0.5	0.03	0.2	0	0
mfcc_avg	0.92	0.91	0.84	0.57	0.8	0.57	0.57
mfcc_delta_avg	0.5	0.64	0.75	0.57	0	0.57	0.57
mfcc_delta_moment2	0.83	0.89	0.72	0.05	0	0.05	0
mfcc_delta_moment3	0.66	0.87	0.55	0.03	0	0	0
mfcc_delta_moment4	0.7	0.89	0.5	0.03	0	0	0
mfcc_delta_moment5	0.63	0.87	0.5	0.03	0	0	0
mfcc_f1_avg	0.98	0.98	0.86	0.53	1	0.53	0.53
mfcc_f1_delta_avg	0.87	0.9	0.76	0.3	0.3	0.3	0.3
mfcc_f1_delta_moment2	0.68	0.88	0.53	0.04	0	0	0
mfcc_f1_delta_moment3	0.63	0.87	0.5	0.03	0	0	0
mfcc_f1_delta_moment4	0.64	0.87	0.49	0.03	0	0	0
mfcc_f1_delta_moment5	0.57	0.85	0.47	0.03	0	0	0
mfcc_f1_moment2	0.66	0.87	0.54	0.05	0	0	0
mfcc_f1_moment3	0.65	0.88	0.5	0.03	0	0	0
mfcc_f1_moment4	0.62	0.87	0.5	0.03	0	0	0
mfcc_f1_moment5	0.64	0.87	0.49	0.03	0	0	0
mfcc_moment2	0.74	0.87	0.65	0.03	0	0	0
mfcc_moment3	0.65	0.88	0.5	0.03	0	0	0
mfcc_moment4	0.6	0.86	0.49	0.03	0	0	0
mfcc_moment5	0.63	0.87	0.49	0.03	0	0	0

Table 3.2: The set of weights used by the weighted arithmetic mean. The weighted root-mean square only use the sets labeled manual 2 and performance.

Chapter 4

Results

This chapter presents the results from the experiments done in chapter 3.

4.1 Evaluating results

Several measurements are common when evaluating search results. A very common approach is to use a precision versus recall curve to outline how the system performs in retrieving relevant results. Since our system does not have a complete list of which results are relevant, the precision is impossible to compute. The F-measure, defined as $\frac{2pr}{p+r}$, where p is precision and r is recall, is excluded for the same reason.

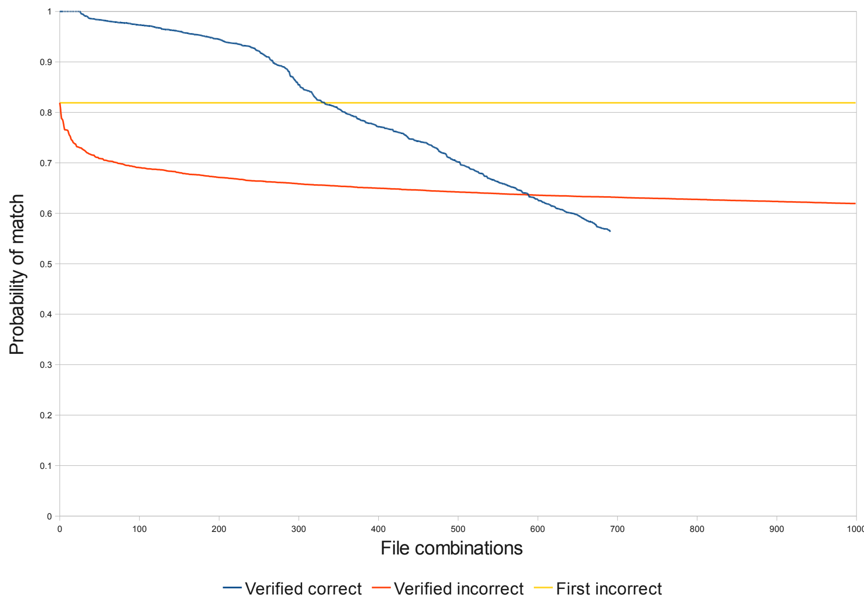
Instead an approach that measures the number of false positives in the first thousand results were chosen. For each descriptor, we extract the thousand highest-ranked results, and compare these results to our list of verified duplicate pairs. By grouping the results into groups of one hundred, we can see roughly how many trustworthy results a descriptor produces. The descriptors are then ranked by the percentage of false positives in the first thousand positive results. After the first thousand results none of the individual descriptors or combined solutions produced less than 50% error rates when divided into partitions of one hundred.

This method of evaluation takes into account the fact that results have limited use if not sorted correctly. Google became popular because they realized that the important thing about search was putting relevant results first [33, 26]. This also applies to finding duplicates in a music collection, the most relevant results should appear first.

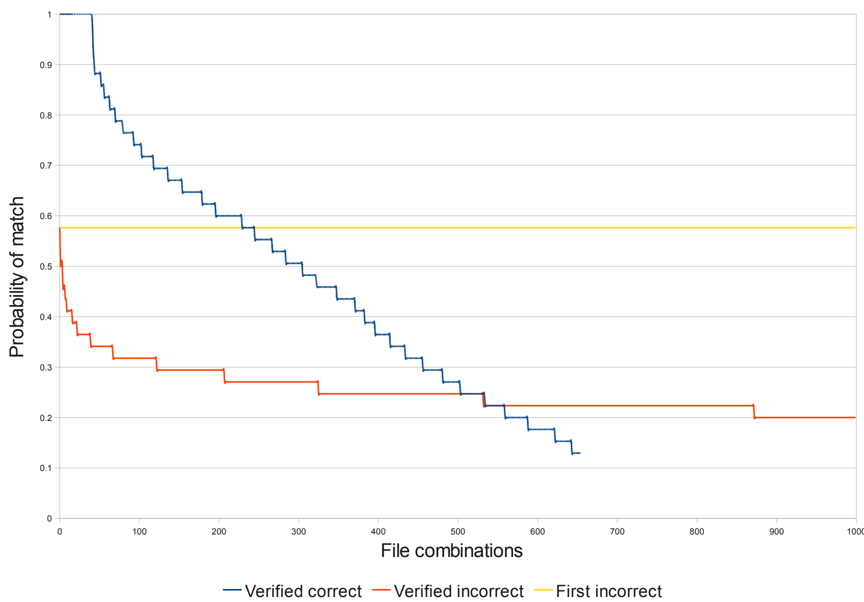
4.2 Individual descriptors

The results from the individual descriptors were not impressive. To examine the results closely, the first thousand results were ranked and then partitioned into 10 partitions of 100 results. Figures 4.2 and 4.3 show the results from the individual descriptors, with the x-axis being the partitions, and the y-axis being the number of false positives.

As can be seen in the graph, several of the descriptors do reasonably well. mfcc_avg and mfcc_f1_avg do not produce a single false positive in the first 300



(a) mfcc avg



(b) fdmf 2

Figure 4.1: Results from two individual descriptors graphed. Data generated from list of potential duplicate pairs ordered by the likelihood given by the individual descriptors. The top line shows verified duplicate pairs, while the lower line shows the incorrect matches. The middle, straight line highlights the first error.

ranked results. libfooid, fdmf_0, fdmf_2, mfcc_delta_avg and mfcc_f1_delta_avg also show promising results, and produced few false positives in the first 400 results.

Several of the descriptors, such as song length, perform badly. The first 3 599 results returned by that descriptor all indicate perfect matches, and ranking results is therefore reduced to which files were entered first into the database. It is therefore obvious that song length as a descriptor is completely inadequate for it's intended purpose, and it can at best be used as an optimization detail.

Figure 4.4 shows the total false positive rate for the first thousand for each individual descriptor. These numbers were used to generate the weights by performance as detailed in section 3.5.9.

The results from a subset of the descriptors were analyzed to determine how they fail, and if there are patterns in the results that can be exploited. The top 100 erroneous results from each descriptor was, and the three highest ranking erroneous matches was excerpted for the reader's reference.

4.2.1 fdmf_0

This descriptor represents the energy spectrum of the audio signal, and is one of the better-performing descriptors. Considering the first few erroneous matches, only two has a probability higher than 0.5.

File 1	File 2	Score
Billie Holiday - Spoken Introduction	Sound of electric drill	0.86667
Billie Holiday - A Sunbonnet Blue	Sound of electric drill	0.57333
Billie Holiday - Just One of Those Things	Taube - Sa länge skutan kan gå	0.44

Table 4.1: First three erroneous duplicate pairs from fdmf_0.

None of these audio recordings are in any way perceptually similar, which indicates that the energy spectrum does not look at the audio in the same way as a human would. The electric drill recording is a recording that was inadvertently included in the collection, but illustrates that music collections very often has short recordings that might not be music.

Worth noting with this descriptor, is that it has not given a full score to two files of different file formats. This could be because the music collection contains more MP3 files than WMA files, but the absence of WMA files in the first 50 results is noteworthy, as they appear regularly from that point onwards, always with a score lower than 0.9.

4.2.2 fdmf_1

Looking at the ratio spectrum fingerprint generated by fdmf, it is the poorest performing of the fdmf descriptors, however it does better than some of the other descriptors.

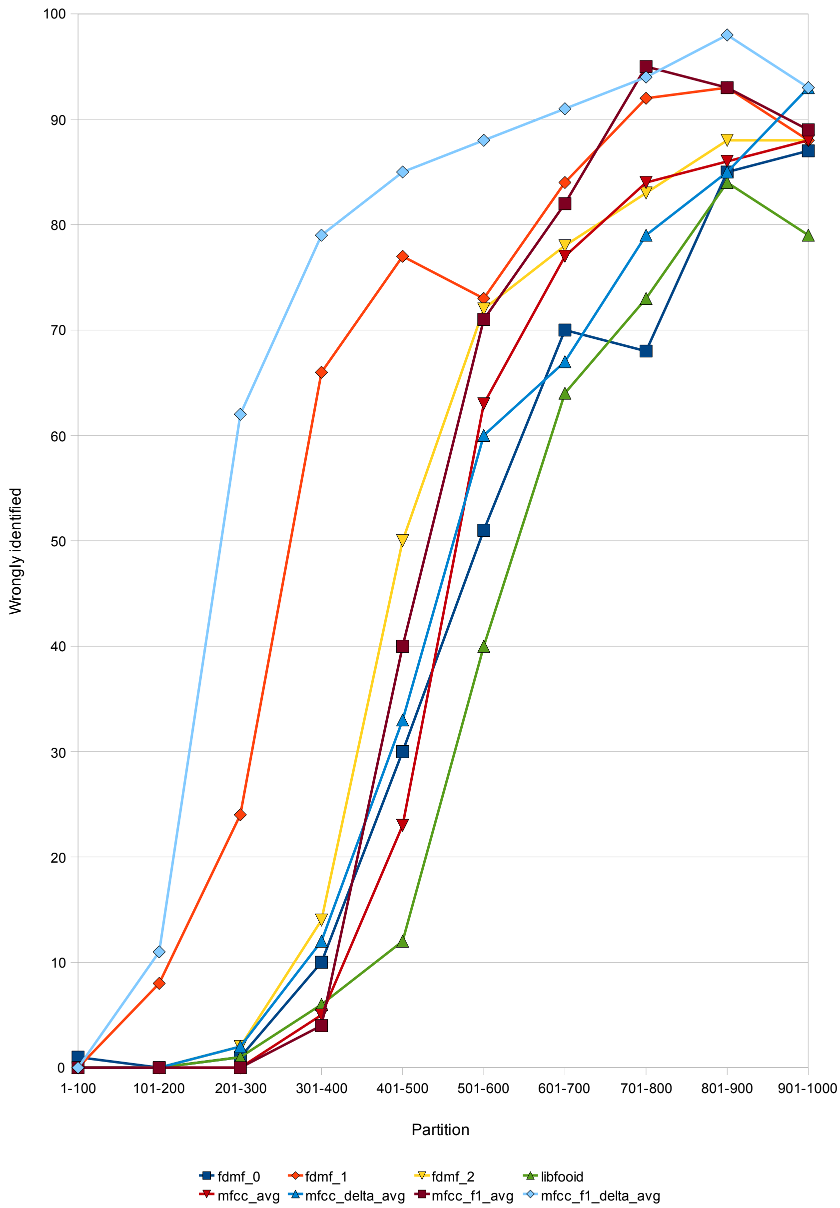


Figure 4.2: Number of false positives in the first 1000 results for the best performing descriptors, divided into partitions of 100 results. x-axis shows partition number, y-axis shows the number of false positives.

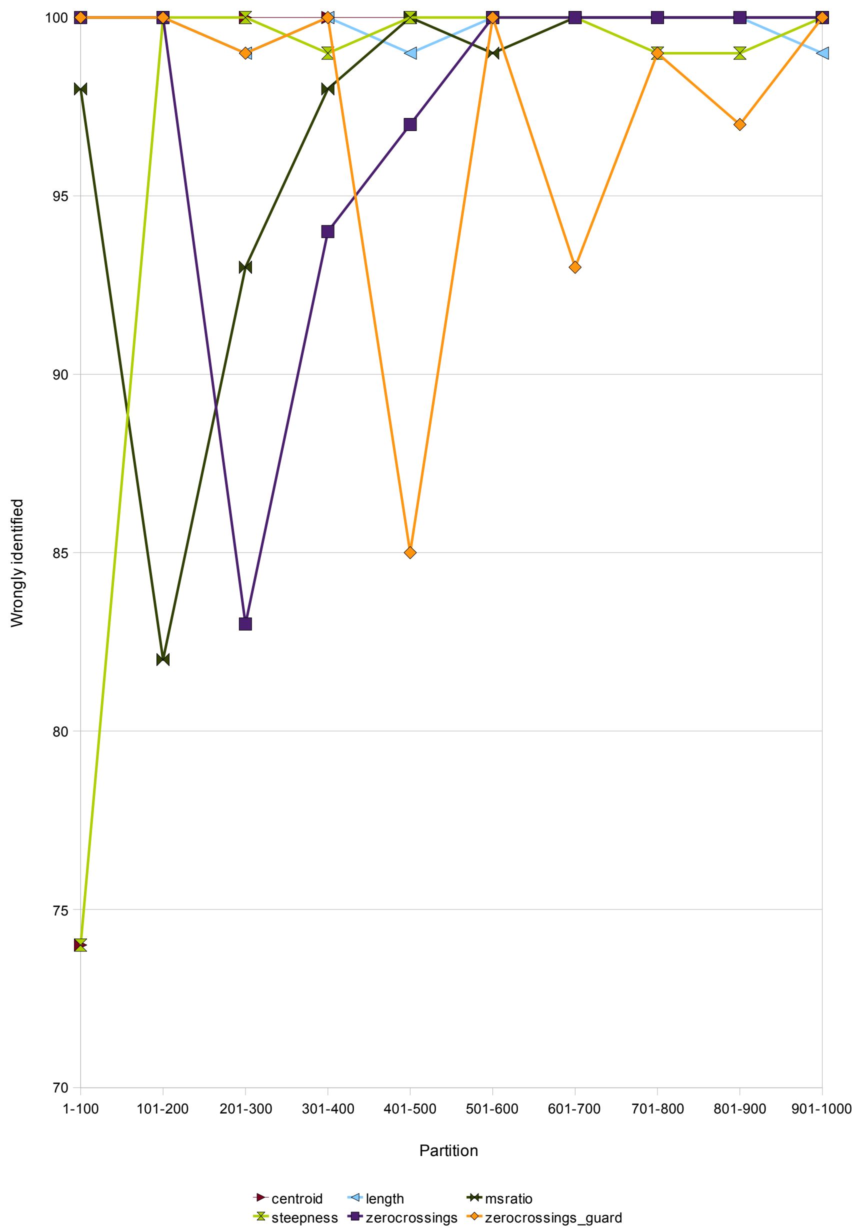


Figure 4.3: Number of false positives in the first 1000 results for the simple descriptors, divided into partitions of 100 results. x-axis shows partition number, y-axis shows the number of false positives. Note that the scale varies from figure 4.2.

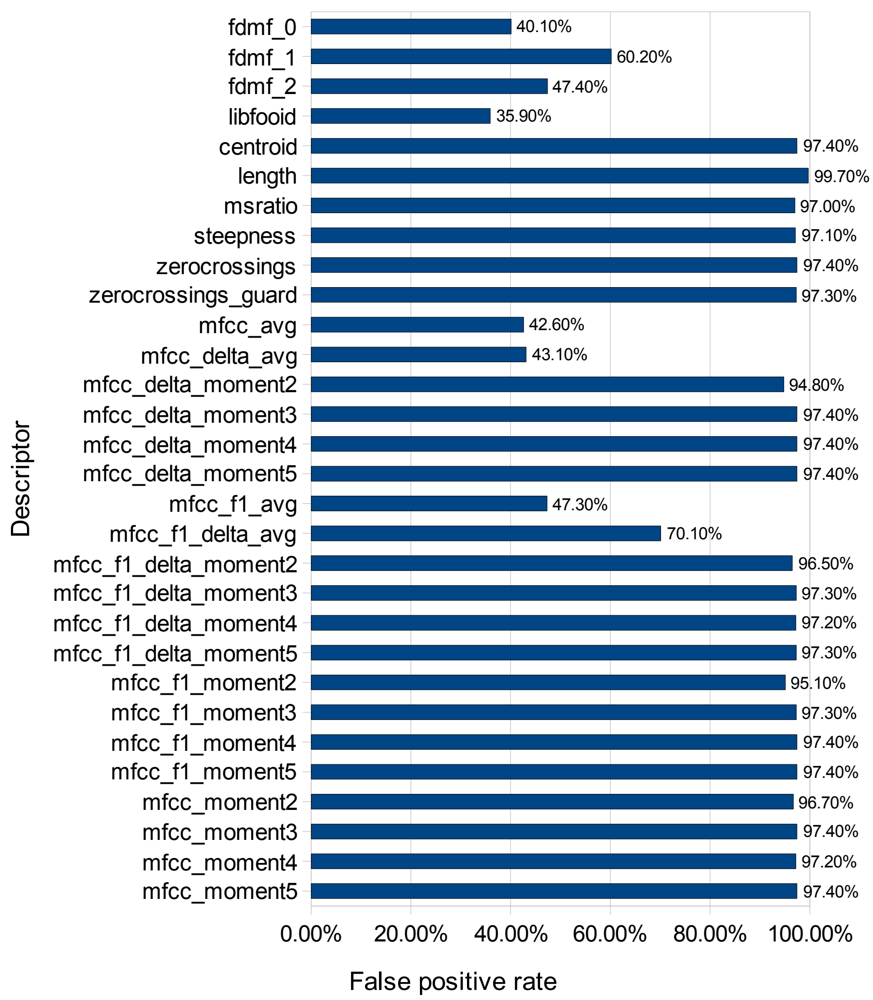


Figure 4.4: False positive rate from the individual descriptors for the first thousand results.

File 1	File 2	Score
The Doors - Hello, I Love You	Taube - Stockholmsmelodi	0.791304
The Doors - Hello, I Love You	Vamp - Ingeborg	0.773913
Hugh Cornwell - Snapper	Taube - Stockholmsmelodi	0.773913

Table 4.2: First three erroneous duplicate pairs from fdmf_1.

Unlike fdmf_0, fdmf_1 produces a large number of high-probability matches that are erroneous. Among the wrong results, some songs are repeatedly matched against a variety of songs, while the songs themselves sound very different to a human.

The same tendency as for fdmf_0 in detecting duplicate pairs with different file types is present in fdmf_1. The first cross-match occurs at position 50, with a score of less than 0.9.

4.2.3 fdmf_2

The twist spectrum fingerprint from fdmf produces few high-probability erroneous matches. Only the first 16 erroneous matches have a score higher than 0.4.

File 1	File 2	Score
Billie Holiday - Spoken Introduction	Sound of electric drill	0.576471
Billie Holiday - Yesterdays	Billie Holiday - Our Love Is Different	0.505882
Billie Holiday - Jeepers Creepers	Billie Holiday - Our Love Is Different	0.505882

Table 4.3: First three erroneous duplicate pairs from fdmf_2.

Again we see the same audio matching in number 1 for fdmf_0, and a predominance for Billie Holiday recordings. Common for those recordings are the poor quality and the noise level, which may confuse the descriptor.

Again, fdmf_2 seems to match worse on files with different file formats. This might indicate that fdmf's spectrum analysis is sensitive to encoding artifacts in various file formats.

4.2.4 libFooID

libFooID produces a range of high-probability erroneous duplicate pairs, while having the lowest false-positive rate of the tested descriptors. However, the first three erroneous duplicate pairs are all moody, string-heavy music with no drums, which could indicate that libFooID picks up the lack of percussion as a similarity.

This descriptor deserves a special mention, as it is the only complete fingerprinting solution that is used without decomposing it into its internal parts. It is also the best performing single descriptor tested.

File 1	File 2	Score
Keith Jarret - Hymn For Good Friday	Michael Andrews - Carpathian Ridge	0.852641
Brian Eno - Harmonic Studies	Michael Andrews - Cellar Door	0.851268
Michael Andrews - Carpathian Ridge	Michael Andrews - The Tangent Universe	0.505882

Table 4.4: First three erroneous duplicate pairs from libFooID.

Also libFooID seems to rank differently encoded files somewhat lower than files in the same file format.

4.2.5 Centroid

The centroid does not perform well, and a large amount of audio files produce very high probability matches. The erroneous duplicate pairs identified by the centroid do not appear to have anything in common when listened to by a human.

File 1	File 2	Score
Talking Heads - Listening Wind	The Triffids - Jerdacuttup Man	1
Neil Young - Trans Am	Rush - Fly By Night	1
Apoptygma Berzerk - Black Pawn	The Waterboys - Fisherman's Blues	0.999999

Table 4.5: First three erroneous duplicate pairs from the centroid descriptor.

4.2.6 Song length

The song length is such an obviously flawed descriptor to use for audio identification of a single audio file. It therefore comes as no surprise that it is the worst performing descriptor.

> The usefulness of the track length as a descriptor is highly dependent on the use case in question: In discerning different tracks from CD, it can be an excellent measure, in genre classification, it is nearly useless, and finally, in some cases (such as when identifying what music is being played on the radio at a given instant), it might not be available at all. [15]

It is hard to imagine the song length as a reliable indicator of the contents of the music, as a lot of the songs produced today is between 3 and 4 minutes long. It can be useful in detecting outliers, such as quickly skipping a comparison of two songs if the songs are 10 minutes different in length.

	Song name	Length	File size
File 1	Björk - Hunter	4:15	6 160 832 bytes
File 2	Hugh Cornwell - Always The Sun (live)	4:15	8 390 786 bytes
File 1	Björk - Venus As A Boy	4:41	6 796 152 bytes
File 2	Enya - Water Shows The Hidden Heart	4:41	6 796 956 bytes
File 1	Björk - Army of Me	3:55	5 695 546 bytes
File 2	The Rolling Stones - It Won't Take Long	3:55	8 330 019 bytes

Table 4.6: First three erroneous duplicate pairs from using track length as a descriptor.

If the track length is available, using it as an optimization to quickly remove tracks that vary greatly in length could work, and is done by e.g. libFooID. It cannot be trusted as an individual descriptor due to the very high likelihood that other songs fall within the same range.

4.2.6.1 Song length as an optimization

The song length is a descriptor that is very easy to understand, and so are the reasons for why it will not work as a descriptor for identifying single audio files. The average song length in our collection is 230 seconds (3 minutes and 50 seconds). In our collection, a total of 2826 songs are within 30 seconds of the average song length. 1465 songs are within 15 seconds from the average track length.

If using the song length as an optimization and automatically failing comparisons of two songs with a difference in length exceeding 30 seconds; you can skip approximately 85% of the actual fingerprint comparisons on average. In our system, this excluded 231 verified duplicate pairs, as our criteria specifies only looking at the first minute of audio.

4.2.7 Mean/square ratio

The mean/square ratio performs on an equal level with the centroid, but has a large number of erroneous duplicate pairs with a perfect score. The erroneous duplicate pairs found do not carry a perceptual similarity when listened to by a human.

File 1	File 2	Score
Björk - Joga	Led Zeppelin - Moby Dick	1
Billy Idol - Crank Call	David Bowie - Strangers When We Meet	1
Billie Holiday - Romance in the Dark	Tom Waits - My Baby Left Me on A Trash Day	1

Table 4.7: First three erroneous duplicate pairs from the mean/square ratio descriptor.

4.2.8 Steepness

Steepness is another badly performing descriptor, and reports many high-probability erroneous duplicate pairs. The erroneous duplicate pairs again show no discernible pattern in what triggers a match using that descriptor.

File 1	File 2	Score
Apoptygma Berzerk - More Serotonin Please	Billie Holiday - Interview 1a	0.999998
Annie Lennox - Loneliness	deLillos - Fange i ditt eget bur	0.999997
Kevin Bloody Wilson - Flowers	XTC - Knights In Shining Karma	0.999996

Table 4.8: First three erroneous duplicate pairs from the steepness descriptor.

4.2.9 Rate of zero crossings

Both with and without a guard band, the ratio of zero crossings produce a large amount of erroneous duplicate pairs with perfect scores. This means the results are as random as the song length, and no discernible pattern can be found in the duplicate pairs it finds.

4.2.10 Mel frequency cepstral coefficients

mfcc_avg is one of the best performing descriptors for our task, and has an apparent tendency in its erroneous duplicate pairs to link two songs by the same artist. In fact, of the top ten erroneous duplicate pairs produced, only two were different artists. In the eight cases, all the songs matched were found on the same album.

File 1	File 2	Score
Tom Waits - I'll Shoot The Moon	Tom Waits - Pony	0.818893
Dane Cook - Someone Shit On The Coats	Dane Cook - Driveway Intruder	0.806178
Tom Waits - Eyeball Kid	Tom Waits - Pony	0.79033

Table 4.9: First three erroneous duplicate pairs from the mfcc_avg descriptor.

This clearly indicates that mfcc_avg identifies songs using the same instruments and having the same remastering as more similar.

When looking at mfcc_delta_avg, it is apparent that it follows some of the same pattern. It performs on the same level as mfcc_avg, but produces different erroneous duplicate pairs.

File 1	File 2	Score
Johnny Cash - Where We'll Never Grow Old	Johnny Cash - I'm Bound For The Promised Land	0.857711
Kevin Bloody Wilson - The Browneye Medley	Kevin Bloody Wilson - Australian Anthems	0.851686
Perssons Pack - I all vår tid	deLillos - Nå lever den av seg selv	0.843354

Table 4.10: First three erroneous duplicate pairs from the mfcc_delta_avg descriptor.

The various mfcc_moment and mfcc_delta_moment descriptors are as inaccurate as the centroid and the steepness. Examples of how they fail have therefore not been included.

4.2.11 Floor-1 cepstral coefficients

f1cc_avg performs slightly worse for our tasks than mfcc_avg. Considering the improvements made to F1CC over MFCC, it was intended to increase the resilience to encoding artifacts. This might have changed the algorithm sufficiently to make it perform worse on other tasks.

File 1	File 2	Score
Two To Tango - Vision	deLillos - Den feite mannen	0.725673
Dane Cook - Struck By A Vehicle	Dane Cook - Abducted	0.71712
Kevin Bloody Wilson - Take It Like A Man	Tom Waits - Get Behind The Mule	0.716681

Table 4.11: First three erroneous duplicate pairs from the f1cc_avg descriptor.

The tendency to identify songs from the same artist as duplicate pairs is decreased in f1cc_avg compared to mfcc_avg, but still present. f1cc_avg produces a significantly different set of erroneous duplicate pairs than does mfcc_avg.

File 1	File 2	Score
Astor Piazolla - Nuevo Tango	Nick Cave & The Bad Seeds - Sleeping Annaleah	0.88834
Billie Holiday - Stars Fell on Alabama	Billie Holiday - 'Deed I Do	0.882058
Johnny Cash - Where We'll Never Grow Old	Johnny Cash - If We Never Meet Again This Side Of Heaven	0.881461

Table 4.12: First three erroneous duplicate pairs from the f1cc_delta_avg descriptor.

Considering f1cc_delta_avg, it too produces significantly different erroneous duplicate pairs from mfcc_delta_avg, yet persists in the tendency of identifying duplicate pairs where both songs are from the same artist.

4.3 Combined solutions

When looking at figure 4.5 it is immediately avident that the applied methods works as a method of producing more stable results.

The best performing combination was the weighted root-mean square, which only had a failure rate of 4.6% among the first thousand results. This means it correctly identified 11 more duplicate pairs than the runner-up.

4.3.1 Arithmetic mean

The arithmetic mean is highly affected by the large number of always high-probability descriptors. It therefore has a higher failure rate than many of the individual descriptors, and can therefore not be used without an algorithm for choosing reliable descriptors.

When using the arithmetic mean on the chosen descriptors only, the results markedly improve. The total error rate is now only 8%, and it has no erroneous results in the first five hundred. This suggests that it works well as a method of producing a stable fingerprint.

4.3.2 Root-mean square

As with the arithmetic mean, the root-mean square produces useless results when not limiting the set of descriptors. However, when choosing the descriptor set carefully, the root-mean square performs slightly better than the arithmetic mean on the same descriptors, with a failure rate of 6.6%.

4.3.3 Weighted arithmetic mean

The weighted arithmetic mean is among the most interesting combination methods in our set, and produces some of the lowest failure rates when certain sets of weights are used. First, weighing by the number of results produced by a descriptor did not provide a significant change in the results, and neither did weighing by the sum of probabilities. Weighing by the average score did not produce a large variation in the number of correct results found.

Weighing by performance did produce very interesting results, and is the sixth best performing method. Our manual weight adjustments, that removed descriptors with weights less than 0.05 and 0.10 produced a decrease in failure rates by 1 and 0.9 percentage points respectively.

It is noteworthy that the WAM with manual weights set 2 and 3 did produce its first erroneous duplicate pair later than any of the other methods used.

4.3.4 Weighted root-mean square

The weighted root-mean square (WRMS) measure was introduced after seeing the success rates of the weighted arithmetic mean.

WRMS in this thesis used the same weights as weighing by performance for the weighted arithmetic mean, and the second set of manual weights because it ranked better in the weighted arithmetic mean. When weighing by performance, WRMS achieved a failure rate of 6.4%, which is 0.5 percentage points lower than

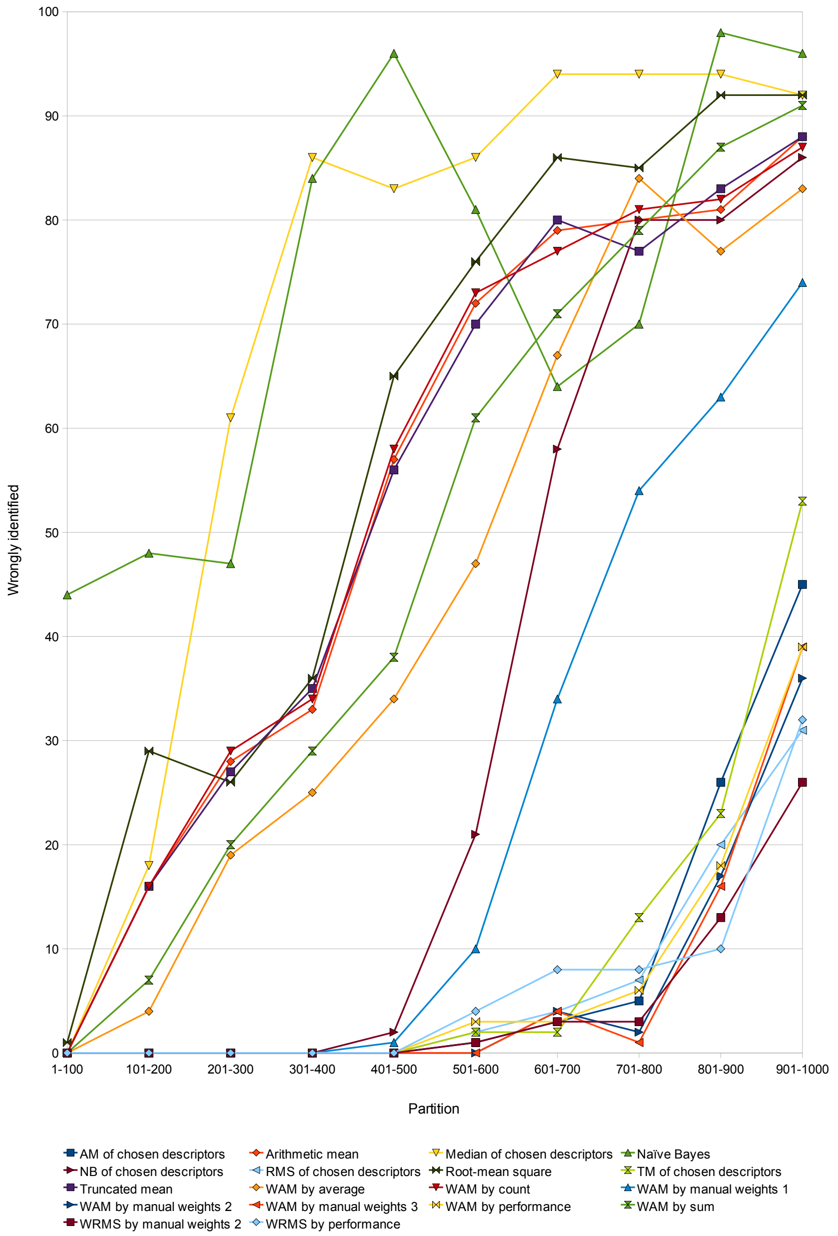


Figure 4.5: Number of false positives in the first 1000 results from the combined solutions, divided into partitions of 100 results. x-axis shows partitions, y-axis shows the number of false positives.

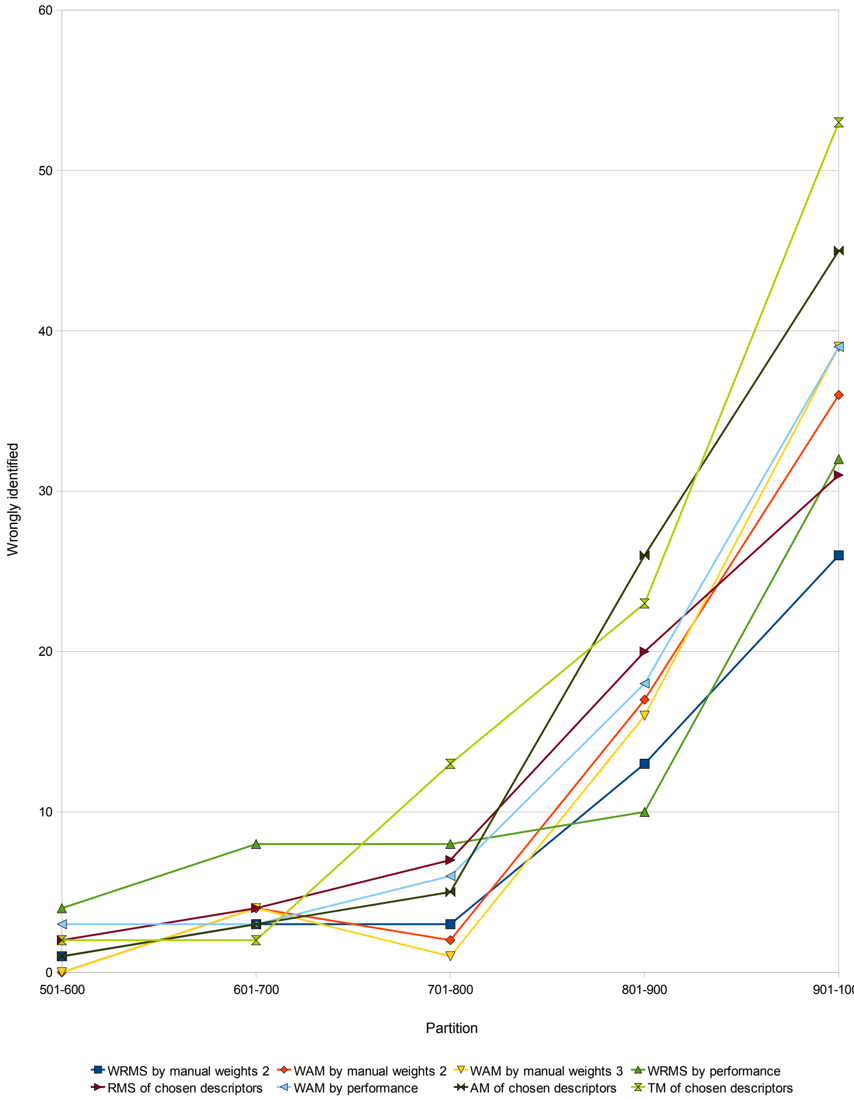


Figure 4.6: Zoomed in view of the lower right part of figure 4.5 for a subset of the combination methods.

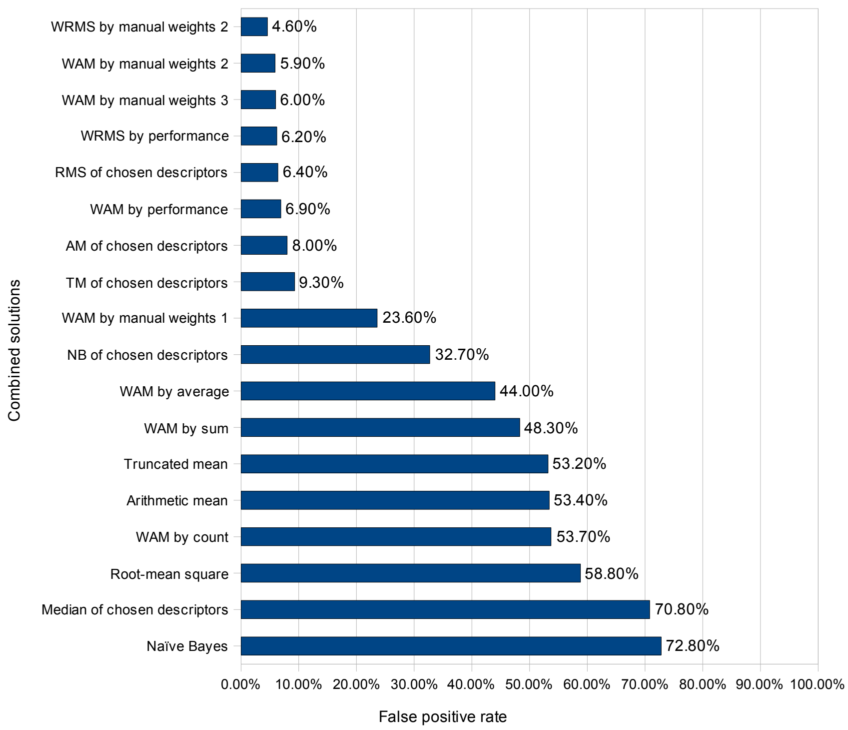


Figure 4.7: False positive rate from the combined solutions for the first thousand results.

Name	Partitions										False positive rate	First error at result #
	1	2	3	4	5	6	7	8	9	10		
WRMS using manual weights 2	0	0	0	0	0	1	3	3	13	26	4.60%	596
WAM using manual weights 2	0	0	0	0	0	0	4	2	17	36	5.90%	603
WAM using manual weights 3	0	0	0	0	0	0	4	1	16	39	6.00%	624
WRMS by performance	0	0	0	0	0	4	8	8	10	32	6.20%	513
RMS of chosen descriptors	0	0	0	0	0	2	4	7	20	31	6.40%	560
WAM by performance	0	0	0	0	0	3	3	6	18	39	6.90%	525
AM of chosen descriptors	0	0	0	0	0	1	3	5	26	45	8.00%	583
TM of chosen descriptors	0	0	0	0	0	2	2	13	23	53	9.30%	547
WAM using manual weights 1	0	0	0	0	1	10	34	54	63	74	23.60%	448
NB of chosen descriptors	0	0	0	0	2	21	58	80	80	86	32.70%	489
WAM by average	0	4	19	25	34	47	67	84	77	83	44.00%	168
WAM by sum	0	7	20	29	38	61	71	79	87	91	48.30%	161
TM	0	16	27	35	56	70	80	77	83	88	53.20%	136
AM	0	16	28	33	57	72	79	80	81	88	53.40%	128
WAM by count	0	16	29	34	58	73	77	81	82	87	53.70%	124
RMS	1	29	26	36	65	76	86	85	92	92	58.80%	96
Median of chosen descriptors	0	18	61	86	83	86	94	94	94	92	70.80%	113
NB	44	48	47	84	96	81	64	70	98	96	72.80%	1

Table 4.13: The false positive rate of combinations, sorted from lowest to highest. The position of the first erroneous duplicate pair was included for reference.

WAM. When weighing using the second manual set of weights, it only had a 4.8% error rate, which is clearly the best achieved in this thesis.

4.3.5 Truncated mean

The truncated mean is a small improvement in results over the arithmetic mean when run on all the descriptors, with a difference of only two more correctly identified results. This is not a statistically significant difference from the arithmetic mean.

When run on the chosen set of descriptors, it performs worse than the arithmetic mean, with a failure rate of 9.3%. Discarding values therefore seems to be a bad idea when working with a carefully chosen set of descriptors.

4.3.6 Median

Of the combination methods working on the reduced set of descriptors, the median performs worst of all. It is simply not a good indicator of central tendency for this use, with a failure rate of 70.8%.

4.3.7 Naïve Bayes

Results-wise, the naïve Bayes approach does not perform as well as its competitors. Even when considering only the top descriptors, it had a failure rate of 32.8%, which is worse than most other methods. When used on all descriptors, it has a failure rate of 72.8%.

It does have one redeeming feature, that can come in handy in certain cases. By changing the value of $P(M)$ manually, you can adjust how much a negative or positive probability affect the final score, and this allows you to produce a smaller number of high-probability matches that are almost guaranteed to be correct. This is a trade-off that will limit the number of false positives, but it will dramatically increase the number of false negatives as well.

Because of the results seen, it is likely that the assumption that the score given by a descriptor can be interpreted as a measure of probability is false.

4.3.8 Original FDMF

The results from the original FDMF package was easy to reproduce using the new architecture. FDMF does not provide any ranking of its results, and simply lists them in the order the files are encountered by the fingerprinter. The same conditions were applied here.

FDMF found 627 duplicate pairs, of which 593 were verified to be correct. This means that compared to the best combined solutions presented, FDMF results in 38% fewer correct results.

Because of the lack of a sorting algorithm in FDMF, a simple arithmetic mean was applied to the scores given by the three descriptors to give it a ranked score. When using this sorting, FDMF had only 1 erroneous match in the first 500 scores.

Descriptor	Weight
fdmf_0	0.6
fdmf_1	0.39
fdmf_2	0.52
libfooid	0.64
mfcc_avg	0.57
mfcc_delta_avg	0.57
mfcc_f1_avg	0.53
mfcc_f1_delta_avg	0.3

Table 4.14: Weights by performance for various descriptors.

4.4 Examples

Some examples of potential duplicate pairs were picked from our collection, so the properties of the various methods in different situations can be examined. Only a subset of the averaging methods are presented here for the sake of readability. Only the top 8 best performing descriptors are considered. For reference, the weights used are found in table 4.14.

(a) Scores

Descriptor	Score
fdmf_0	0.63
fdmf_1	0.81
fdmf_2	0.88
libfooid	0.85
mfcc_avg	0.98
mfcc_delta_avg	0.84
mfcc_f1_avg	0.98
mfcc_f1_delta_avg	0.48

(b) Combined scores

Method	Score	SD
Truncated mean	0.8307	0.1894
Weighted arithmetic mean	0.7790	0.5147
Arithmetic mean	0.8054	0.1620
Root-mean square	0.8216	0.1628
Bayesian	0.9999	-
Median	0.8436	0.1665
Weighted RMS	0.8313	0.2272

Table 4.15: Scores for "Dido - Do You Have A Little Time" in MP3 and WMA encodings. The system correctly scores this as a good match.

4.5 Improving search speed

The analyzed system is not in any way optimized for quick searches. In fact, the calculations required to perform a search that matches all files against each other takes several hours on a fairly up to date computer. An indexing scheme is needed.

A very simplistic approach is to use any one of the scalar descriptors. A small script (see B.4.5) was created to assess the viability of these descriptors automatically. For each descriptor the range of the distances between verified duplicate pairs was found. Only fingerprints within the range $[v - \delta, v + \delta]$ were examined as candidate matches, where v is the scalar descriptor value for the query file, and δ is two times the average distance between verified duplicate pair fingerprints for that descriptor.

(a) Scores

Descriptor	Score
fdmf_0	0
fdmf_1	0.10
fdmf_2	0
libfooid	0.65
mfcc_avg	0.63
mfcc_delta_avg	0.81
mfcc_f1_avg	0.41
mfcc_f1_delta_avg	0.81

(b) Combined scores

Method	Score	SD
Truncated mean	0.4158	0.2909
Weighted arithmetic mean	0.4201	0.7753
Arithmetic mean	0.4248	0.2480
Root-mean square	0.5358	0.2172
Bayesian	0.5035	-
Median	0.6379	0.2243
Weighted RMS	0.5260	0.3024

Table 4.16: Scores for two separate recordings of "Billie Holiday - Solitude". The songs are so different they were not considered a match when verified by a human listener. Most combinations score this as a low-probability match, yet still sees the similarity in the songs.

(a) Scores

Descriptor	Score
fdmf_0	0
fdmf_1	0
fdmf_2	0
libfooid	0
mfcc_avg	0.53
mfcc_delta_avg	0.73
mfcc_f1_avg	0.30
mfcc_f1_delta_avg	0.77

(b) Combined scores

Method	Score	SD
Truncated mean	0.21051	0.3429
Weighted arithmetic mean	0.2705	0.5862
Arithmetic mean	0.2932	0.2460
Root-mean square	0.4353	0.1701
Bayesian	0.5425	-
Median	0.6315	0.1362
Weighted RMS	0.4030	0.1618

Table 4.17: Scores for "David Bowie - Ricochet" and "Vamp - Svin på skog". Despite some similarities reported by the mfcc-series of descriptors, fdmf and libfooid correctly sees no match in these two songs. Yet, the both the median and the naïve Bayes classifier reports higher scores than wanted.

The length descriptor is the clear winner, as it results in both the lowest number of candidate files on average, and finds a higher percentage of the verified duplicate pairs than any other descriptor.

When trying multiple combinations for descriptors, one expects a sharp drop in the number of combinations that has to be tried, and an increase in the number of verified matches that was removed. An attempt was made were combinations of one or more scalar descriptors were used as an index, to see that no combination of descriptors produced any kind of "magic index".

Descriptor	Avg. # of comp.	% found	Avg. distance
Centroid	3538	84.71%	370
Track length	2487	87.71%	18.2
Mean/square ratio	3595	86.27%	0.017
Steepness	3465	84.72%	125
Zerocrossings	3463	85.7%	9100
Zerocrossings w/ guard band	3520	85.37%	5049

Table 4.18: Performance characteristics of the six scalar descriptors if they were to be used as an indexing scheme, when searching for all the verified duplicate pair files.

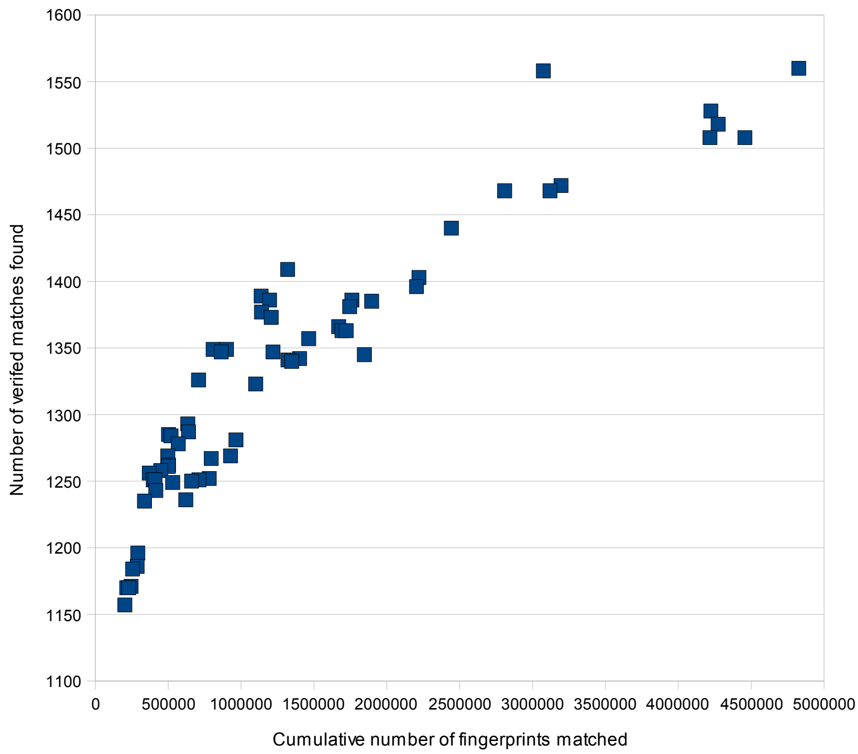


Figure 4.8: The number of verified duplicate pairs that were removed versus the number of files that would be included in the search when sequentially searching for all duplicate pairs. An optimum solution would be ranked in the upper left corner.

Chapter 5

Conclusion

Large music collections are now more common than ever before. Yet, search technology for music is still in its infancy. Audio fingerprinting is one method that allows searching for music.

In this thesis several audio fingerprinting solutions are combined into a single solution to determine if such a combination can yield better results than any of the solutions can separately. The solution is used to find duplicate music files in a personal collection.

A real-life music collection with 9536 music files was used. Audio fingerprints for all files were created using three software suites, namely fdmf, libFooID and Gunderson's master thesis. By combining the likelihood from multiple system that two files match, recognition rates were increased.

5.1 Results

This thesis finds that using different averaging methods to produce a more stable audio fingerprint works satisfactorily, and can be used to effectively combine different systems.

Among the methods of combination that was tested, the weighted root-mean square most effectively ranked the results in a satisfying manner. It was notably better than the other approaches tried. The WRMS produced 61% more correct matches than the original FDMF solution, and 49% more correct matches than libFooID.

The individual descriptors can be ranked according to performance in the following order, from best to worst: libfooid, fdmf_0, mfcc_avg, mfcc_delta_avg, mfcc_f1_avg, fdmf_2, fdmf_1 and mfcc_f1_delta_avg. The remaining descriptors all ranked considerably worse than the ones listed here, and were found to be too untrustworthy to be included in calculations.

5.2 Evaluation

The implementation of this thesis went smoothly. However, there are certain aspects that could be attempted for improving the proposed system.

The most important improvement is likely to be a strictly controlled music collection. Constructing a music collection from original CDs, and encoding the audio with lossless and lossy encodings, and introducing a certain amount of noise into the recordings would be helpful. With such a collection it would also be possible to create precision vs. recall curves for the system.

Writing the glue code using PHP and MySQL seems to have been a mistake, as it was chosen because of the author's familiarity with the language. A better choice would have been to use C and a binary database system, as putting the data into MySQL proved to be slow. Data processing in PHP is very slow compared to native languages, such as C.

An approach that tries to use averages on all possible combinations of descriptors could potentially improve the solution even further. This approach could be used to determine whether the results from individual descriptors have high degrees of overlap, and could potentially exclude some descriptors from the recommended set.

5.3 Further work

The combination of software used in this thesis has not been optimized for speed, but for being malleable enough for conducting the tests required to provide better results. As such, speed has not been a priority, and the system therefore is very slow, even on a relatively small collection such as the one we tested on.

Most of the fingerprinting engines are written in C or C++, and it will be relatively easy to combine these into a modular system that allows plugging in new fingerprinting engines easily. By writing the fingerprint comparer in C or C++, instead of an interpreted language such as PHP, the system could easily be made to search for a fingerprint in an already fingerprinted collection in $O(n)$ time.

Bibliography

[1] Michael Fink; Michele Covell;Shumeet Baluja. Social- and interactive-television applications based on real-time ambient-audio identification. 2007. Accessed on the 24th of March 2008.

[2] S. Baluja and M. Covell. Audio fingerprinting: Combining computer vision & data stream processing. *Acoustics, Speech and Signal Processing, 2007. ICASSP 2007. IEEE International Conference on*, 2:II–213–II–216, 15-20 April 2007.

[3] R. H. Blessing. Outlier treatment in data merging. *Journal of Applied Crystallography*, 30(4):421–426, 1997.

[4] Mark Bocko. Music file compressed 1,000 times smaller than mp3, 2008. Accessed on April 2nd 2008.

[5] K. Brandenburg. MP3 and AAC explained. *Proceedings of AES 17th International Conference, XP008004053*, pages 99–110.

[6] K. Brandenburg, J. Herre, J.D. Johnston, Y. Mahieux, and E. Schroeder. ASPEC: Adaptive spectral entropy coding of high quality music signals. *Proc. 90th Conv. Aud. Eng. Soc*, 1991.

[7] The Echo Nest Corporation. The echo nest. Accessed on the 21st of May 2008.

[8] M. Covell and S. Baluja. Known-audio detection using waveprint: Spectrogram fingerprinting by wavelet hashing. *Acoustics, Speech and Signal Processing, 2007. ICASSP 2007. IEEE International Conference on*, 1:I–237–I–240, 15-20 April 2007.

[9] J. Stephen Downie. Music information retrieval (chapter 7). *Annual Review of Information Science and Technology 37*, pages 295–340, 2003.

[10] T. Eriksson and H.G. Kang. Pitch quantization in low bit-rate speech coding. *Acoustics, Speech, and Signal Processing, 1999. ICASSP'99. Proceedings., 1999 IEEE International Conference on*, 1, 1999.

[11] Xiph.org foundation. Vorbis i specification, 2004. Accessed on the 21st of May 2008.

[12] Gracenote. Gracenote: Musicid. Accessed on the 21st of May 2008.

[13] Paul Graham. *Hackers & Painters*. O'Reilly, 2004. ISBN 059600662-4.

[14] Amara Graps. An introduction to wavelets. *IEEE Comput. Sci. Eng.*, 2 (2):50–61, 1995.

[15] Steinar H. Gunderson. Musical descriptors: An assessment of psychoacoustical models in the presence of lossy compression. Master's thesis, 2007.

[16] Jaap Haitsma, Ton Kalker, and Job Oostveen. Robust audio hashing for content identification. In *International Workshop on Content-Based Multimedia Indexing (CBMI'01)*, Brescia, Italy, September 2001. Accessed on the 1st of February 2008.

[17] Richard Jones. Audio fingerprinting for clean metadata, 2007. Accessed on the 21st of May 2008.

[18] Yan Ke, D. Hoiem, and R. Sukthankar. Computer vision for music identification. *Computer Vision and Pattern Recognition, 2005. CVPR 2005. IEEE Computer Society Conference on*, 1:597–604 vol. 1, 20-25 June 2005.

[19] Vegard Andreas Larsen. Determining audio fingerprint boundaries. Norwegian University of Science and Technology, 2007.

[20] Alison Latham. *The Oxford Companion to Music*. Oxford University Press, 2002. ISBN 0198662122.

[21] D.D. Lewis. Naive (Bayes) at forty: The independence assumption in information retrieval. *Proceedings of ECML-98, 10th European Conference on Machine Learning*, pages 4–15, 1998.

[22] Ricardo Miotto and Nicola Orio. A methodology for the segmentation and identification of music works. In *ISMIR 2007 - Proceedings of the 8th International Conference on Music Information Retrieval, Vienna, Austria, September 23-27*, pages 273–278. Österreichische Computer Gesellschaft, 2007.

[23] H. Nyquist. Certain topics in telegraph transmission theory. *Proceedings of the IEEE*, 90(2):280–305, 2002.

[24] Justis og politidepartementet. *Lov om patenter*. 2007. ISBN 82-504-1193-5. Accessed on the 4th of February 2008.

[25] IFLA Study Group on the Functional Requirements for Bibliographic Records. *Functional Requirements for Bibliographic Records: Final Report*. UBCIM Publications-New Series. K.G.Saur, München. Accessed on the 21st of May 2008.

[26] L. Page, S. Brin, R. Motwani, and T. Winograd. The pagerank citation ranking: Bringing order to the web, 1998.

[27] Gian-Carlo Pascutto. foosic - the living music database, 2006. Accessed on the 3rd of February 2008.

[28] Michaël Betser; Patrice Collen; Jean-Bernard Rault. Audio identification using sinusoidal modeling and application to jingle detection. In *ISMIR*

2007 - Proceedings of the 8th International Conference on Music Information Retrieval, Vienna, Austria, September 23-27, pages 139–142. Österreichische Computer Gesellschaft, 2007. Accessed on the 30th of March 2008.

[29] Trent D. Stephens Rod R. Seeley, Philip Tate. *Essentials of anatomy and physiology*. McGraw-Hill, 6th edition edition, 2006. ISBN 007110746-0.

[30] Kurt Rosenfeld. Merging music collections without redundancy, October 2005. Accessed on the 27th of April 2008.

[31] Kurt Rosenfeld. Learning to recognize duplicate music files, May 2007. Accessed on the 27th of April 2008.

[32] J.G. Shao-Jen Lim; Harris. Analog implementation of ratio spectrum computation. *Circuits and Systems, 1998. ISCAS '98. Proceedings of the 1998 IEEE International Symposium on*, 1:277–280 vol.1, 31 May-3 Jun 1998.

[33] Joel Spolsky. *Joel on Software: And on Diverse and Occasionally Related Matters That Will Prove of Interest to Software Developers, Designers, and Managers, and to Those Who, Whether by Good Fortune or Ill Luck, Work with Them in Some Capacity*. Apress, 1 edition, 2004. ISBN 1590593898. 125-130 pp.

[34] Hartmut Traunmuller. Analytical expressions for the tonotopic sensory scale. *The Journal of the Acoustical Society of America*, 88(1):97–100, 1990.

[35] University of Jyväskylä. *A Matlab toolbox for music feature extraction from audio*, September 2007.

[36] J.M. Valin and C. Montgomery. Improved noise weighting in CELP coding of speech-applying the Vorbis psychoacoustic model to Speex. *Audio Engineering Society Convention*, 2006.

[37] Eric W Weisstein. Metric, 2004. Accessed on the 21st of May 2008.

[38] Eric W Weisstein. Wave equation, 2004. Accessed on the 21st of May 2008.

Index

Appendix A

Software used

GNU GCC GNU's Compiler Collection (GCC) is one of the most widely used compilers for C code. Available from `http://www.gnu.org/`.

LYX A free software document processor that uses LaTeX. Available from `http://www.lyx.org/`.

Eclipse Programming environment, originally designed for Java development. Available from `http://www.eclipse.org/`.

PHP Scripting language originally designed for creating dynamic web pages. Available from `http://www.php.net/`.

PHPEclipse Plugin for Eclipse that simplifies development in PHP. Available from `http://www.phpeclipse.de/`.

phpMyAdmin MySQL administration interface written in PHP. Available from `http://www.phpmyadmin.net/`.

MySQL Open source DBMS. Available from `http://www.mysql.com/`.

Ubuntu Free GNU/Linux-based operating system . Available from `http://www.ubuntu.org/`.

Appendix B

Source code

The source code presented in this appendix is lisenced under the same license as the software it modifies.

B.1 fdmf

The original source code and the modifications made to fdmf are licensed under the GPL.

B.1.1 db.h

Database connectivity headers.

```
#ifndef DB_H_
#define DB_H_

#include <stdio.h>
#include <fcntl.h>
#include <unistd.h>
#include <sys/types.h>
#include <sys/stat.h>
#include <pwd.h>
#include <string.h>
#include <gdbm.h>
#include <assert.h>
#include </usr/include/mysql/mysql.h>

int db_setup(void);
void db_close(void);
int db_find_file(char *, int);
int db_insert_file(char *, int, int);
void db_insert_fingerprint(int, int, char *, int);
void db_add_score(int, int, int, int, int);

#endif /*DB_H_*/
```

B.1.2 db.c

Database connectivity and functions for inserting data.

```
#include "vector_pairs.h"

#define SQL_INSERT_SCORE "INSERT INTO tblScore (ixFile1, ixFile2, ixType
    , dblScore) VALUES (?, ?, ?, 1 - (? / ?))"
#define SQL_SELECT_FINGERPRINT "SELECT ixFile FROM tblFingerprint WHERE
    ixFile = ? AND ixType = ?"
```

```
#define SQL_INSERT_FINGERPRINT "INSERT INTO tblFingerprint (ixFile, ixType, sFingerprint) VALUES (?, ?, ?)"
#define SQL_SELECT_FILE "SELECT ixFile FROM tblFile WHERE md5Path = MD5(?)"
#define SQL_INSERT_FILE "INSERT INTO tblFile(sPath, md5Path, cbFile) VALUES (?, MD5(?), ?)"

MYSQL *db_conn;
MYSQL_STMT *sqlInsertFile;
MYSQL_STMT *sqlSelectFingerprint;
MYSQL_STMT *sqlInsertFingerprint;
MYSQL_STMT *sqlSelectFile;
MYSQL_STMT *sqlInsertScore;

/*
 * Lets create a local cache of every ixFile value we get from
 * MySQL, so we won't have to query for it every time.
 */
int *file_cache;
int file_cache_ready = 0;

int db_setup()
{
    char *server     = "localhost";
    char *user       = "master";
    char *password   = "7aBPK8huJQz3TNQS";
    char *database   = "master";

    db_conn = mysql_init(NULL);
    if (!mysql_real_connect(db_conn, server, user, password, database, 0, NULL, 0))
    {
        fprintf(stderr, "%s\n", mysql_error(db_conn));
        exit(0);
    }
    printf("database connected\n");

    sqlInsertScore = mysql_stmt_init(db_conn);
    mysql_stmt_prepare(sqlInsertScore, SQL_INSERT_SCORE, strlen(SQL_INSERT_SCORE));

    sqlSelectFingerprint = mysql_stmt_init(db_conn);
    mysql_stmt_prepare(sqlSelectFingerprint, SQL_SELECT_FINGERPRINT, strlen(SQL_SELECT_FINGERPRINT));

    sqlInsertFingerprint = mysql_stmt_init(db_conn);
    mysql_stmt_prepare(sqlInsertFingerprint, SQL_INSERT_FINGERPRINT, strlen(SQL_INSERT_FINGERPRINT));

    sqlSelectFile = mysql_stmt_init(db_conn);
    mysql_stmt_prepare(sqlSelectFile, SQL_SELECT_FILE, strlen(SQL_SELECT_FILE));

    sqlInsertFile = mysql_stmt_init(db_conn);
    mysql_stmt_prepare(sqlInsertFile, SQL_INSERT_FILE, strlen(SQL_INSERT_FILE));

    return(0);
}

void db_close()
{
    mysql_close(db_conn);

    mysql_stmt_close(sqlInsertScore);
    mysql_stmt_close(sqlSelectFingerprint);
    mysql_stmt_close(sqlInsertFingerprint);
    mysql_stmt_close(sqlSelectFile);
    mysql_stmt_close(sqlInsertFile);
}

void init_file_cache(int size)
{
    int i;
```

```
    if (!file_cache_ready)
    {
        file_cache = (int*) malloc(sizeof(int) * size);
        for (i = 0; i < size; i++)
            file_cache[i] = -1;
        file_cache_ready = 1;
    }
}

int db_find_file(char *file, int cFileNo)
{
    int ixFile = -1;
    MYSQL_BIND bind[1];
    unsigned long length;
    my_bool is_null;
    my_bool error;

    if (file_cache[cFileNo] != -1)
        return file_cache[cFileNo];

    memset(bind, 0, sizeof(bind));

    length = strlen(file);

    bind[0].buffer_type = MYSQL_TYPE_STRING;
    bind[0].buffer = file;
    bind[0].buffer_length = strlen(file);
    bind[0].is_null = 0;
    bind[0].length = &length;

    if (mysql_stmt_bind_param(sqlSelectFile, bind))
    {
        fprintf(stderr, "mysql_stmt_bind_param() failed in db_find_file()\n");
        fprintf(stderr, " %s\n", mysql_stmt_error(sqlSelectFile));
        exit(0);
    }
    if (mysql_stmt_execute(sqlSelectFile))
    {
        fprintf(stderr, "mysql_stmt_execute() failed in db_find_file()\n");
        fprintf(stderr, " %s\n", mysql_stmt_error(sqlSelectFile));
        exit(0);
    }

    memset(bind, 0, sizeof(bind));

    bind[0].buffer_type= MYSQL_TYPE_LONG;
    bind[0].buffer = (char *)&ixFile;
    bind[0].buffer_length = 4;
    bind[0].length = &length;
    bind[0].is_null = &is_null;
    bind[0].error = &error;

    if (mysql_stmt_bind_result(sqlSelectFile, bind))
    {
        fprintf(stderr, "mysql_stmt_bind_result() failed in db_find_file()\n");
        fprintf(stderr, " %s\n", mysql_stmt_error(sqlSelectFile));
        exit(0);
    }

    if (mysql_stmt_store_result(sqlSelectFile))
    {
        fprintf(stderr, "mysql_stmt_store_result() failed in db_find_file()\n");
        fprintf(stderr, " %s\n", mysql_stmt_error(sqlSelectFile));
        exit(0);
    }

    if (mysql_stmt_num_rows(sqlSelectFile) == 0)
        return -1;

    if (mysql_stmt_fetch(sqlSelectFile))
```

```
    {
        fprintf(stderr, "mysql_stmt_fetch() failed in db_find_file()\n")
            ;
        fprintf(stderr, " %s\n", mysql_stmt_error(sqlSelectFile));
        exit(0);
    }
    if (ixFile != -1)
        file_cache[cFileNo] = ixFile;

    /* fprintf(stderr, "%s %d %ld %d\n", file, ixFile, length, error);
        */
    return ixFile;
}

/* returns ixFile, -1 if error */
int db_insert_file(char *file, int cFileNo, int cFiles)
{
    struct stat fileBuf;
    int ixFile = -1;
    long cbFile = 0;
    MYSQL_BIND bind[3];
    unsigned long str_len;

    init_file_cache(cFiles);

    ixFile = db_find_file(file, cFileNo);
    if (ixFile != -1)
        return ixFile;

    memset(bind, 0, sizeof(bind));

    stat(file, &fileBuf);
    cbFile = fileBuf.st_size;

    str_len = strlen(file);

    bind[0].buffer_type = MYSQL_TYPE_STRING;
    bind[0].buffer = (char *)file;
    bind[0].buffer_length = strlen(file);
    bind[0].length = &str_len;
    bind[0].is_null = 0;

    // yes, we're duping info, let mysql handle MD5 hashing
    bind[1].buffer_type = MYSQL_TYPE_STRING;
    bind[1].buffer = (char *)file;
    bind[1].buffer_length = strlen(file);
    bind[1].length = &str_len;
    bind[1].is_null = 0;

    bind[2].buffer = (char *)&cbFile;
    bind[2].buffer_type = MYSQL_TYPE_LONG;
    bind[2].is_null = 0;
    bind[2].length = 0;

    if (mysql_stmt_bind_param(sqlInsertFile, bind))
    {
        fprintf(stderr, "mysql_stmt_bind_param() failed in
            db_insert_file()\n");
          fprintf(stderr, " %s\n", mysql_stmt_error(sqlInsertFile));
          exit(0);
    }

    if (mysql_stmt_execute(sqlInsertFile))
    {
        fprintf(stderr, "mysql_stmt_execute failed in db_insert_file()!\
            n");
          fprintf(stderr, " %s\n", mysql_stmt_error(sqlInsertFile));
        exit(0);
    }

    /* return db_find_file(file); */
    return mysql_insert_id(db_conn);
}
```

```
int db_has_fingerprint(int ixFile, int ixType)
{
    int ixReturn = -1;
    MYSQL_BIND bind[2];
    unsigned long length;
    my_bool is_null;
    my_bool error;

    memset(bind, 0, sizeof(bind));

    bind[0].buffer_type = MYSQL_TYPE_LONG;
    bind[0].buffer = (char *)&ixFile;
    bind[0].is_null = 0;
    bind[0].length = 0;

    bind[0].buffer_type = MYSQL_TYPE_LONG;
    bind[0].buffer = (char *)&ixType;
    bind[0].is_null = 0;
    bind[0].length = 0;

    if (mysql_stmt_bind_param(sqlSelectFingerprint, bind))
    {
        fprintf(stderr, "mysql_stmt_bind_param() failed in
            db_has_fingerprint()\n");
        fprintf(stderr, " %s\n", mysql_stmt_error(sqlSelectFingerprint))
            ;
        exit(0);
    }
    if (mysql_stmt_execute(sqlSelectFingerprint))
    {
        fprintf(stderr, "mysql_stmt_execute() failed in
            db_has_fingerprint()\n");
        fprintf(stderr, " %s\n", mysql_stmt_error(sqlSelectFingerprint))
            ;
        exit(0);
    }

    memset(bind, 0, sizeof(bind));

    bind[0].buffer_type= MYSQL_TYPE_LONG;
    bind[0].buffer = (char *)&ixReturn;
    bind[0].buffer_length = 4;
    bind[0].length = &length;
    bind[0].is_null = &is_null;
    bind[0].error = &error;

    if (mysql_stmt_bind_result(sqlSelectFingerprint, bind))
    {
        fprintf(stderr, "mysql_stmt_bind_result() failed in
            db_has_fingerprint()\n");
        fprintf(stderr, " %s\n", mysql_stmt_error(sqlSelectFingerprint))
            ;
        exit(0);
    }

    if (mysql_stmt_store_result(sqlSelectFingerprint))
    {
        fprintf(stderr, "mysql_stmt_store_result() failed in
            db_has_fingerprint()\n");
        fprintf(stderr, " %s\n", mysql_stmt_error(sqlSelectFingerprint))
            ;
        exit(0);
    }

    if (mysql_stmt_num_rows(sqlSelectFingerprint) == 0)
        return 0;
    else
        return 1;
}

void db_insert_fingerprint(int ixFile, int ixType, char *fingerprint,
    int fingerprint_length)
{
    char *sqlInsert;
```

```
    char *sFingerprint;

    if (db_has_fingerprint(ixFile, ixType))
        return;

    sqlInsert = (char *) malloc(2048);
    sFingerprint = (char *) malloc(2048);
    mysql_real_escape_string(db_conn, sFingerprint, fingerprint,
        fingerprint_length);

    sprintf(sqlInsert, "INSERT INTO tblFingerprint (ixFile, ixType,
        sFingerprint) VALUES (%d, %d, '%s')", ixFile, ixType,
        sFingerprint);
    mysql_query(db_conn, sqlInsert);
}

void db_add_score(int ixFile1, int ixFile2, int ixType, int bDistance,
    int bThreshold)
{
    MYSQL_BIND bind[5];
    int i;

    memset(bind, 0, sizeof(bind));

    for (i = 0; i < 5; i++)
    {
        bind[i].buffer_type = MYSQL_TYPE_LONG;
        bind[i].is_null = 0;
        bind[i].length = 0;
    }

    if (ixFile1 < ixFile2)
    {
        bind[0].buffer = (char *)&ixFile1;
        bind[1].buffer = (char *)&ixFile2;
    }
    else
    {
        bind[0].buffer = (char *)&ixFile2;
        bind[1].buffer = (char *)&ixFile1;
    }
    bind[2].buffer = (char *)&ixType;
    bind[3].buffer = (char *)&bDistance;
    bind[4].buffer = (char *)&bThreshold;

    if (mysql_stmt_bind_param(sqlInsertScore, bind))
    {
        fprintf(stderr, "mysql_stmt_bind_param failed in db_add_score()
            !\n");
        fprintf(stderr, " %s\n", mysql_stmt_error(sqlInsertScore));
        exit(0);
    }

    if (mysql_stmt_execute(sqlInsertScore))
    {
    /* failing this insert is perfectly legitimate, as this comparison
        may already
     * exist somewhere in the database, so we won't do any error
         reporting.
     */
    /*
        fprintf(stderr, "mysql_stmt_execute failed in db_add_score()!\n
            ");
        fprintf(stderr, " %s\n", mysql_stmt_error(sqlInsertScore));
        exit();
    */
    }
}
```

B.1.3 run_tests.c

Modifications for inserting music files into the database, along with potential duplicate pairs.

```
#include "vector_pairs.h"

int run_tests(char *dblk, int *hashes, char **names,
              int vecs, int *thres, char delim) {
    /* compare all pairs of vectors for possible matches */
    int i, j, k, bitcount[256], type1_err=0, type2_err=0;
#ifdef INSERT_INTO_DB
    int ixFile = 0;
    int ixFile2 = 0;
    char *temp;
#endif
    char terminator;
    struct stat stat_results;
    if (delim == '\0')
    {
        terminator = '\0';
    }
    else
    {
        terminator = '\n';
    }
    setup_bitcount_tbl(bitcount);
    for (i = 0; i < vecs; i++)
    {
        /* code for mysql database insertion */

#ifdef INSERT_INTO_DB
        ixFile = db_insert_file(names[i], i, vecs);
        temp = (char *) malloc(VECTOR_BYTES);
        strncpy(temp, dblk + i * MULTI_VEC_LEN + 0 * VECTOR_BYTES, VECTOR_BYTES);
        db_insert_fingerprint(ixFile, 1, temp, VECTOR_BYTES); /* fdmf_0 */
        strncpy(temp, dblk + i * MULTI_VEC_LEN + 1 * VECTOR_BYTES, VECTOR_BYTES);
        db_insert_fingerprint(ixFile, 2, temp, VECTOR_BYTES); /* fdmf_1 */
        strncpy(temp, dblk + i * MULTI_VEC_LEN + 2 * VECTOR_BYTES, VECTOR_BYTES);
        db_insert_fingerprint(ixFile, 3, temp, VECTOR_BYTES); /* fdmf_2 */
        free(temp);
#endif

        for (j = i + 1; j < vecs; j++) {
#ifdef INSERT_INTO_DB
            ixFile2 = db_insert_file(names[j], j, vecs);
#endif
            int distance[NUM_TESTS], score, test;
            char *ptr_a = dblk + i * MULTI_VEC_LEN;
            char *ptr_b = dblk + j * MULTI_VEC_LEN;
            int basenames_match, contents_similar;
            /* calculate distance between two vectors for each test*/
            for (score = test = 0; test < NUM_TESTS; test++) {
                for (distance[test] = k = 0; distance[test] <= thres[test] && k < VECTOR_BYTES; k++) {
                    unsigned char c = *(ptr_a + k) ^ *(ptr_b + k);
                    distance[test] += bitcount[c];
                }
                ptr_a += VECTOR_BYTES;
                ptr_b += VECTOR_BYTES;

                if (distance[test] < thres[test])
                {
                    score++;
#ifdef INSERT_INTO_DB
                    db_add_score(ixFile, ixFile2, test+1, distance[test], thres[test]);
#endif
```

```
                }
            } /* post: score contains the number of tests that passed */
            contents_similar = (score >= SCORE_THRESHOLD) ? 1 : 0;
            basenames_match = (hashes[i] == hashes[j]) ? 2 : 0;
#ifdef IGNORE_GHOST_FILES
            /* If either file of a pair does not exist, ignore the pair
                */
            if (contents_similar + basenames_match) {
                if (stat(names[i], &stat_results)) continue;
                if (stat(names[j], &stat_results)) continue;
            }
# endif
            switch (contents_similar + basenames_match) {
                case 0: /* different contents, different basenames */
                    /* do nothing */
                    break;
                case 1: /* similar contents, different basenames */
                      printf("%s%c%s%c", names[i], delim, names[j],
                          terminator);
                    type1_err++;
                    break;
                case 2: /* different contents, same basenames */
                  /*     printf("type 2: %s%c%s%c", names[i], delim,
                                names[j], terminator); */
                    type2_err++;
                    break;
                case 3: /* similar contents, same basenames */
                      printf("%s%c%s%c", names[i], delim, names[j],
                          terminator);
                    break;
            }
            if (contents_similar + basenames_match == 1 ||
                contents_similar + basenames_match == 3)
            {
                printf("%d%c%d%c%d%c", distance[0], delim, distance[1],
                    delim, distance[2], delim);
            }

        }
    }
    fprintf(stderr, "%d\t%d\t%d\t", thres[0], thres[1], thres[2]);
    fprintf(stderr, "%d\t%d\n", type1_err, type2_err);
    return(0);
}
```

B.1.4 fdmf

Software that runs fdmf and fooid on all the files in a specific folder. Modified to run fooid.

```
#!/usr/bin/perl -w
# Kurt Rosenfeld 2004, 2005, 2006
# GPL
use strict;
use GDBM_File;
use Digest::MD5 qw(md5 md5_hex);
use Encode qw(encode_utf8);
use File::Copy;
use File::Basename;
use Cwd;

my $db_file = glob '~/.fdmf';
my $musicdir = $ARGV[0] or die "RTFM";
my $NUM_BANDS = 4;

my $SONICREDUCER = find_sr() or die "Can't find good sonic_reducer";
# $SONICREDUCER = "cat /mnt/ramdisk/audio.raw | ".$SONICREDUCER;

die "Unable to run spline. Is it in your path?" if `spline /dev/null`;

# We cache the table of filenames and their envelope spectra.
```

```
if (stat $db_file) {  # keep the old db file just in case
    File::Copy::copy "$db_file", "$db_file.old" or die "copy: $!";
}

tie my %DB, 'GDBM_File', glob("$db_file"), &GDBM_WRCREAT|&GDBM_SYNC, 0640;
# If your perl/GDBM can't do SYNC mode, comment the line above and use this:
#tie my %DB, 'GDBM_File', glob("$db_file"), &GDBM_WRCREAT, 0640;

# Recurse through the directory, searching for mp3s
my @filelist = recurse_from_dir($musicdir);

# Compare that filelist to the already-cached files in the database.
my @uncached = grep((not $DB{"$_"}), @filelist);

my $filecount = 0;

print STDERR "Found ",$#filelist+1," files, ",$#uncached+1," uncached.\n";

# this loop will add any uncached files to the database
FILE: for my $file (@uncached) {
    open(FILE, $file) or die "Can't open $file: $!";
    my $file_md5 = Digest::MD5->new->addfile(*FILE)->digest;
    print STDERR $filecount++, "/", $#uncached, " waiting on $file\n";
    # If an identical file is already indexed, reuse its summary.
    foreach my $k (keys %DB) {
        my $db_file_md5 = substr $DB{$k}, 100, 16;
        if ($file_md5 eq $db_file_md5) {
            print STDERR "Identical files (using cached summary):\n";
            print STDERR "\t$file\n\t\tAND\n\t$k\n";
            my $cached_summary  = substr $DB{$k}, 0, 96;
            $DB{$file} = $cached_summary . base_hash($file) . $file_md5;
            next FILE;
        }
    }
    my $summary;
    next FILE if sonic_reduce($file, \$summary);
    $DB{$file} = $summary . base_hash($file) . $file_md5;
} # post: %DB is complete

untie %DB;

#########################################################################
########################### THE END #####################################
#########################################################################

sub base_hash {
    substr md5(encode_utf8(basename(shift()))), 0, 4;
}


sub recurse_from_dir {
    my ($dir) = @_;
    my @filelist;
    my $file_ptrn = '^[^.]'; # ignore dot files

    unless (opendir(PARSEDIR, "$dir")) {
        printf STDERR "skipping $dir because $!\n";
        return;
      }

    foreach my $file (sort(readdir(PARSEDIR))) {
        my $fullname = $dir . "/" . $file;
        $fullname =~ s/\/+/\//g;
        if (-d $fullname && $file !~ /^\..*/) {
            push @filelist, recurse_from_dir($fullname);
        }
        elsif ($file =~ /$file_ptrn/i && -r $fullname) {
            $filelist[++$#filelist] = $fullname;
        }
    }
    closedir(PARSEDIR);
```

```perl
    return @filelist;
}


sub sonic_reduce {
    # ARGUMENT: filename of music file
    # RETURN by REFERENCE: 768-bit string for fdmf database
    # RETURN VALUE: 0 for success, nonzero for nonsuccess.
    my $f =  shift;
    my $summary_ref = shift;
    my @SR;
    my @DECODE_CMD;
    return -1 if decode_cmd($f, \@DECODE_CMD);

    pipe PIPE1_OUT, PIPE1_IN;

    my $pid1 = fork;
    if ($pid1 == 0) { # we are child 1 (the decoder)
        close(PIPE1_OUT) or die "$!";
        open(STDOUT, ">&PIPE1_IN") or die "$!";
        exec @DECODE_CMD or die "$!";
    }
    else { # we are still the parent (having had one child of two)
        close(PIPE1_IN) or die; # child 1 will write on this
        pipe PIPE2_OUT, PIPE2_IN;
        # we will wait for the decode to be done
        # the original fdmf did not do this, as it got input from
        # stdin, that was produced in real time by the decoder
        # we now need all that input for later software
        waitpid $pid1, 0;
        my $pid2 = fork;
        if ($pid2 == 0) { # we are child 2 (the sonic_reducer process)
            close(PIPE2_OUT) or die "$!";
            open(STDIN,  "<&PIPE1_OUT") or die "$!";
            open(STDOUT, ">&PIPE2_IN") or die "$!";
            exec $SONICREDUCER." /mnt/ramdisk/audio.raw" or die "$!";
        }
        else { # we are still the parent (having had both children)
            close(PIPE2_IN) or die "$!";
            @SR = <PIPE2_OUT>;
        }
#        waitpid $pid1, 0;
        waitpid $pid2, 0;
    }

    if ($#SR != 767) {
        print STDERR "sonic_reduce had trouble with $f.  Corrupt audio file?\n";
        return -2;
    }
    my @e = @SR[0   .. 255]; # energy spectrum summary
    my @r = @SR[256 .. 511]; # ratio (high/low) spectrum summary
    my @t = @SR[512 .. 767]; # twist (odd/even ratio) spectrum summary
    my $j = join("", quantize(@e), quantize(@r), quantize(@t));
    $$summary_ref = pack("b*", $j);

    system ("/home/vegard/Masteroppgave/Kode/fdmf/fooid", "/mnt/ramdisk/audio.raw", $f);
    my $md5 = md5_hex($f);
    system ("/home/vegard/Masteroppgave/Kode/fdmf/descriptor /mnt/ramdisk/audio.raw > /home/vegard/Masteroppgave/Kode/results/descriptors/$md5");
    system ("echo \"$md5\" >> /home/vegard/Masteroppgave/Kode/results/descriptors/filelist.txt");

    return 0;
}

sub decode_cmd {
    # This routine looks at the filename extension of a music file.
    # It returns the shell command for decoding it.
    # These also work, if you don't want to use mplayer:
    #     @$cmd_ref = ("mpg123", "-s", "-q", "$f");
    #     @$cmd_ref = ("ogg123", "-d", "raw", "-f", "-", "$f");
```

```
    my $f = shift;
    my $cmd_ref = shift;
    my $filename_extension = lc substr $f, -3;
    if ($filename_extension =~ /mp3|ogg|m4a|wma|wav|\.ra/) {
#        @$cmd_ref = ("mplayer", "-nortc", "-ao", "pcm",
#                     "-aofile", "/dev/stdout", "$f");
#        @$cmd_ref = ("mplayer", "-ao", "pcm:file=/dev/stdout", "$f");
        @$cmd_ref = ("mplayer", "-really-quiet", "-msglevel", "all=1", "-nojoystick", "-nolirc", "-ao", "pcm:fast:file=/mnt/ramdisk/audio.raw", "$f");
        return 0;
    }
    else {
        print STDERR "Filename $f doesn't have an extension that we handle.\n";
        return -1;
    }
}


sub find_sr {
    my $sr_path;
    $sr_path = dirname($0) . "/sonic_reducer"; # look in same dir as fdmf
    return $sr_path if -x $sr_path;
    $sr_path = getcwd() . "/sonic_reducer"; # look in the current directory
    return $sr_path if -x $sr_path;
    return 0; # we failed to find sonic_reducer
}

sub quantize {
    # one bit quantize with median value as threshold
    my @s;
    my $median = median(@_);
    foreach my $i (0 .. $#_) {
        $s[$i] = ($_[$i] > $median) ? 1 : 0;
    }
    return @s;
}

sub median {
    my $size;
    my $median;
    my @sorted = sort {$a <=> $b} @_;
    $size = scalar @sorted;
    if ($size % 2){
        $median = $sorted[($size-1)/2];
    }
    else {
        $median = ($sorted[$size/2] + $sorted[($size/2)-1])/2;
    }
}
```

B.2 libFooID

libFooID also uses db.h and db.c from fdmf. The source code is available under the GPL.

B.2.1 foo.h

Headers.

```
#ifndef FOO_H
#define FOO_H

#include <assert.h>
#include <stdlib.h>
#include <string.h>
```

```c
#include <stdio.h>
#include <math.h>
#include <limits.h>
#include <fooid.h>
#include <sys/stat.h>
#include "db.h"

#endif /*FOO_H*/
```

B.2.2 foo.c

Console tool that reads WAVE files from disk, generates fingerprints (using libFooID), and writes them to the database.

```c
#include "foo.h"

#define SAMPLE_RATE 44100
#define CHANNELS 2
#define N_BLOCKS 1024

t_fooid *fooid;

int fsize(char *name)
{
    struct stat stbuf;

    if (stat(name, &stbuf) == -1)
    {
        fprintf(stderr, "fsize: can't access %s\n", name);
        return 0;
    }
    return stbuf.st_size;
}

int main(int argc, char *argv[])
{
    FILE *fp;
    signed short *data;
    int fp_more = 1;
    int fp_data_size;
    unsigned long fp_fingerprint_size;
    unsigned char *fp_fingerprint;
    int file_size;
    int audio_length;
    int whereami, ixFile;

    if (argc == 1 || argc > 3)
    {
        fprintf(stderr, "Usage:\n");
        fprintf(stderr, "\t%s <raw-audiofile> [<original-filename>]:\n",
            argv[0]);
        exit(1);
    }
    else
    {
        if ((fp = fopen(argv[1], "rb")) == NULL)
        {
            fprintf(stderr, "Can't open %s\n", argv[1]);
            exit(1);
        }
        else
        {
            data = (signed short *)malloc(sizeof(signed short) *
                N_BLOCKS);
            fooid = fp_init(SAMPLE_RATE, CHANNELS);
            if (ferror(fp))
            {
                fprintf(stderr, "Error %d!\n", ferror(fp));
            }

            /*
```

```
             * Lets skip the header (we know we get 44100 Hz, 16 bit, 2
                channel)
             */
            fseek(fp, 44, SEEK_SET);
            whereami = 44;

            while ((!feof(fp)) && (fp_more == 1) && !ferror(fp))
            {
                fp_data_size = fread(data, sizeof(signed short),
                    N_BLOCKS, fp);
                whereami += fp_data_size;
                fp_more = fp_feed_short(fooid, data, fp_data_size);
            }
            fclose(fp);

            fp_fingerprint_size = fp_getsize(fooid);

            file_size = fsize(argv[1]);

            /*
             * The * 2 is because it is 16 bits, while the file size is
                in 8 bits
             */
            audio_length = (file_size / (float) (SAMPLE_RATE * CHANNELS
                * 2)) * 100;

            fp_fingerprint = (unsigned char *)malloc(sizeof(unsigned
                char) * fp_fingerprint_size);
            if (fp_calculate(fooid, audio_length, fp_fingerprint) < 0)
            {
                fprintf(stderr, "fp_calculate failed!\n");
            }

            if (argc == 3)
            {
                db_setup();

                ixFile = db_insert_file(argv[2], 0, 1);

                db_insert_fingerprint(ixFile, 4, (char *) fp_fingerprint
                    , fp_fingerprint_size);

                db_close();
            }

            fp_free(fooid);
            free(data);
            free(fp_fingerprint);
        }
    }

    return 0;
}
```

B.2.3 Database.java

Database connectivity for Java.

```
import java.sql.*;

public class Database {
    public Connection conn;
    public Database()
    {
        try {
            Class.forName("com.mysql.jdbc.Driver").newInstance();
            conn = DriverManager.getConnection(
                "jdbc:mysql://localhost/master?" +
                "user=master&password=didyouthinkyouwouldfindithere");
        } catch (Exception ex) {
            // handle any errors
            System.out.println("SQLException: " + ex.getMessage());
```

```java
            ex.printStackTrace();
        }
    }
}
```

B.2.4 FingerPrint.java

Compares a number of fingerprints to each other, and stores the comparison data in the database.

```java
import java.io.ByteArrayInputStream;
import java.io.DataInputStream;
import java.io.FileInputStream;
import java.sql.*;
import java.util.*;

/**
 * User: Gian-Carlo Pascutto
 * Date: 28-apr-2006
 * Time: 18:14:20
 *
 * Modified to allow for database inserts by Vegard Andreas Larsen.
 */
public class FingerPrint {
    private int version;
    private int length;
    private int avg_dom;
    private int avg_fit;
    private int[][] r;
    private int[] dom;

    public static void main(String[] args)
    {
        Database db = new Database();
        try {
            PreparedStatement selectFilelist = db.conn.prepareStatement("SELECT ixFile, sFingerprint FROM tblFingerprint WHERE ixType = 4");
            ResultSet files = selectFilelist.executeQuery();
            ArrayList<File> rgFiles = new ArrayList<File>();
            files.first();
            while (files.next())
            {
                File f = new File();
                f.ixFile = files.getInt(1);
                f.sFingerprint = files.getBinaryStream(2);
                rgFiles.add(f);
            }

            PreparedStatement insertScore = db.conn.prepareStatement("INSERT IGNORE INTO tblScore (ixFile1, ixFile2, ixType, dblScore) VALUES (?, ?, 4, ?)");

            for (int i = 0; i < rgFiles.size(); i++)
            {
                FingerPrint ofp1 = new FingerPrint();
                rgFiles.get(i).sFingerprint.reset();
                ofp1.read(new DataInputStream(rgFiles.get(i).sFingerprint));
                for (int j = i + 1; j < rgFiles.size(); j++)
                {
                    FingerPrint ofp2 = new FingerPrint();
                    rgFiles.get(j).sFingerprint.reset();
                    ofp2.read(new DataInputStream(rgFiles.get(j).sFingerprint));
                    float value = ofp1.Match(ofp2);
                    if (value != 0.0)
                    {
                        insertScore.setInt(1, rgFiles.get(i).ixFile);
                        insertScore.setInt(2, rgFiles.get(j).ixFile);
                        insertScore.setFloat(3, value);
```

```
                        insertScore.execute();
                    }
                }
                System.out.println("Finished " + rgFiles.get(i).ixFile +
                    "!");
            }

        }
        catch (Exception e) {
            e.printStackTrace();
        }
    }

    public FingerPrint() {
        r = new int[87][16];
        dom = new int[87];
    }

    public void read(DataInputStream fds) throws java.io.IOException,
        Exception {
        byte buff[] = new byte[424];
        int read = fds.read(buff);

        ByteArrayInputStream ds = new ByteArrayInputStream(buff);

        version = ServerUtil.readLEShort(ds);

        if (version != 0 || read != 424) {
            throw new Exception("Fingerprint version not supported.\n");
        }

        length = ServerUtil.readLEInt(ds);
        avg_fit = ServerUtil.readLEShort(ds);
        avg_dom = ServerUtil.readLEShort(ds);

        for (int i = 0; i < 87; i++) {
            for (int j = 0; j < 16; j += 4) {
                int temp = ds.read();

                r[i][j + 0] = (temp >>> 6) & 0x3;
                r[i][j + 1] = (temp >>> 4) & 0x3;
                r[i][j + 2] = (temp >>> 2) & 0x3;
                r[i][j + 3] = temp & 0x3;
            }
        }

        int dompos = 0;

        for (int i = 0; i < (66 / 3); i++) {
            int temp = ds.read();
            int temp2 = ds.read();
            int temp3 = ds.read();

            // 3 * 8 bits = 24 bits = 4 * 6 bit doms
            dom[dompos++] = ((temp >>> 2) & 0x3F);
            dom[dompos++] = (((temp & 0x03) << 4) | ((temp2 >>> 4) & 0
                x0F));
            dom[dompos++] = (((temp2 & 0x0F) << 2) | ((temp3 >>> 6) & 0
                x3));
            if (dompos < 87) {
                dom[dompos++] = (temp3 & 0x3F);
            }
        }

        ds.close();
    }

    public void readFrom(String s) throws java.io.FileNotFoundException,
                                          java.io.IOException, Exception
                                          {
        read(new DataInputStream(new FileInputStream(s)));
    }

    public void displaySummary() {
```

```
        System.out.println("\tLength: " + length);
        System.out.println("\tAVG DOM: " + avg_dom);
        System.out.println("\tAVG FIT: " + avg_fit);
    }

    public void displayFull() {
        displaySummary();

        for (int i = 0; i < 87; i++) {
            StringBuffer line = new StringBuffer(2 * 16 + 1);

            for (int j = 0; j < 16; j++) {
                int tr = r[i][j];
                switch (tr) {
                    case 0:
                        line.append("0 ");
                        break;
                    case 1:
                        line.append("1 ");
                        break;
                    case 2:
                        line.append("2 ");
                        break;
                    case 3:
                        line.append("3 ");
                        break;
                    default:
                        line.append("X ");
                        break;
                }
            }

            System.out.println(line + " DOM: " + dom[i]);
        }
    }

    public float Match(FingerPrint fp) {
        if (quickMatch(fp)) {
            return fullMatch(fp);
        } else {
            return 0.0f;
        }
    }

    public boolean quickMatch(FingerPrint fp) {
        /*
            length within 30 seconds
        */
        if (length + 100 * 30 < fp.length) {
            return false;
        }
        if (length - 100 * 30 > fp.length) {
            return false;
        }

        /*
            average FIT within 0.4
        */
        if (avg_fit + 400 < fp.avg_fit) {
            return false;
        }
        if (avg_fit - 400 > fp.avg_fit) {
            return false;
        }
        /*
            average DOM within 6 units
        */
        if (avg_dom + 600 < fp.avg_dom) {
            return false;
        }
        if (avg_dom - 600 > fp.avg_dom) {
            return false;
        }

```

```
            return true;
        }


        // XXX: early exit here
        public float fullMatch(FingerPrint fp) {
            /*
                determine max sensible frame
            */
            int maxframe = Math.round((length * 0.9765625f) / 100.0f);
            maxframe = Math.min(maxframe, 87);

            /*
                set up 'flaw' counters
            */
            int[] rf = new int[4];
            int[] df = new int[64];
            int tdf, trf;

            rf[0] = 0;
            rf[1] = 0;
            rf[2] = 0;
            rf[3] = 0;
            tdf = 0;
            trf = 0;

            for (int i = 0; i < 64; i++) {
                df[i] = 0;
            }

            for (int f = 0; f < maxframe; f++) {
                for (int b = 0; b < 16; b++) {
                    int rdiff = Math.abs(r[f][b] - fp.r[f][b]);
                    rf[rdiff]++;
                    trf += rdiff;
                }
                int ddiff = Math.abs(dom[f] - fp.dom[f]);
                df[ddiff]++;
                tdf += ddiff;
            }

            /*
                DOM flaws are linear, penality points = how far we're off

                FIT flaws are nonlinear
                    1-off:  1 penalty point
                    2-off:  3 penalty points
                    3-off:  9 penalty points
            */

            final int maxrflaw = 9 * 16 * maxframe;
            final int maxdflaw = (63 * maxframe) / 4;

            int w_trf = rf[1] + rf[2] * 4 + rf[3] * 9;
            int w_tdf = tdf / 4;

            int totalflaws = w_trf + w_tdf;
            final int maxflaws = maxrflaw + maxdflaw;

            /*
                percentage is ratio of our
                flaws to max theorethical flaws
            */
            float perc = ((float) (totalflaws)) / ((float) (maxflaws));
            /*
                we expect a random track to get
                about halfway there, so confidence
                of 50% -> 0%, and a match to get
                nowhere, so confidence of 0% -> 100%
            */
            float conf = ((1.0f - perc) - 0.5f) * 2.0f;

            /*
                limit to sane values
```

```
        */
        conf = Math.min(conf, 1.0f);
        conf = Math.max(conf, 0.0f);

        return conf;
    }

    public int getVersion() {
        return version;
    }

    public int getAvg_dom() {
        return avg_dom;
    }

    public void setAvg_dom(int avg_dom) {
        this.avg_dom = avg_dom;
    }

    public int getAvg_fit() {
        return avg_fit;
    }

    public void setAvg_fit(int avg_fit) {
        this.avg_fit = avg_fit;
    }

    public int getLength() {
        return length;
    }

    public void setLength(int length) {
        this.length = length;
    }
}
```

B.3 Gunderson's descriptors

The software is licensed under the GPL.

B.3.1 wave.h

Headers for reading WAVE files from disk.

```
#ifndef _WAVE_H
#define _WAVE_H 1

#include "common.h"

void read_wave_file(Sample &s, char *filename);

#endif /* !defined(_WAVE_H) */
```

B.3.2 wave.cpp

Functions for reading WAVE files from disk.

```
#include <stdio.h>
#include <assert.h>

#include "wave.h"

#define READ_BLOCKS 1024
#define WAVE_HEADER_SIZE 44

void read_wave_file(Sample &s, char *filename)
{
```

```
    unsigned file_pos = 0;
    long file_len;
    long data_size;
    signed short *data;
    FILE *fp;

    fp = fopen(filename, "rb");
    assert(fp != NULL);
    fseek(fp, 0, SEEK_END);
    file_len = ftell(fp);
    assert(file_len > FEATURE_SAMPLES * 2 + WAVE_HEADER_SIZE);


    long samples = (file_len - WAVE_HEADER_SIZE) / (2 * 2);
    s.samples.reserve(FEATURE_SAMPLES);

    fseek(fp, 44, SEEK_SET);
    data = (signed short *)malloc(sizeof(signed short) * READ_BLOCKS);
    while ((!feof(fp)) && file_pos < FEATURE_SAMPLES * 2)
    {
        data_size = fread(data, sizeof(signed short), READ_BLOCKS, fp);
        // data_size - 1 because we need to be able to read at least 2 samples
        for (int i = 0; i < data_size - 1; i += 2)
        {
            signed short l, r;
            l = data[i    ];
            r = data[i + 1];

#if 0
                /*
                 * There's a lot of codec delay in MP3. This skips
                 * the first 1105 samples (LAME's estimate of the
                 * delay for our test MP3s) to make lining up plots
                 * easier.
                 */
                static unsigned delayed = 0;
                if (++delayed < 1105)
                    continue;
#endif

            s.samples.push_back(0.5 * (double(r) + double(l)));
        }
        file_pos += data_size;
    }
    s.length = (samples / 44100.0);
    free(data);
}
```

B.3.3 read_descriptors.php

Reads the output from Gunderson's descriptors into the database.

```
<?php
/**
 * This file reads a list of files produced by Gunderson's descriptors,
 * and inserts all the fingerprints into tblFingerprint.
 *
 * @author Vegard Andreas Larsen <vegarl@stud.ntnu.no>
 */
require_once('include/include.php');

// where do we find our files?
define('PATH_RESULTS', '/home/vegard/Masteroppgave/Kode/results/descriptors/');

$oDB = CDatabase::Get();

// Let's have a lookup list of the index of a fingerprint type.
// E.g. $rgixType['centroid'] == 6
$sql = "SELECT sType, ixType FROM tblType";
```

```
$rgixType = $oDB->extended->GetAssoc($sql, null, null, null,
    MDB2_FETCHMODE_ASSOC, false);

// Read the list of files and remove excessive fat
$rgFiles = file(PATH_RESULTS.'filelist.txt');
$rgFiles = array_map('trim', $rgFiles);

// Our queries
$sqlInsert = "INSERT IGNORE INTO tblFingerprint (ixFile, ixType,
    sFingerprint) VALUES (?, ?, ?)";
$sqlFind = "SELECT ixFile FROM tblFile WHERE md5Path LIKE ?";

$i = 0;
echo "Starting...\n";
// Loop through files
foreach ($rgFiles as $md5Path)
{
    $rgDescriptors = file(PATH_RESULTS.$md5Path);
    $rgDescriptors = array_map('trim', $rgDescriptors);
    $ixFile = $oDB->extended->GetOne($sqlFind, null, array($md5Path),
        array('text'));
    foreach ($rgDescriptors as $line)
    {
        // Line structure as .ini files
        list($sDescriptor, $sValues) = explode('=', $line);

        /*
         * Some of the fingerprints have very large values, and this
         * is something Gunderson does in his Voronoi program. It should
         * have been done when the fingerprints were generated.
         *
         * We are taking the Y-th root of any moments named *_momentY.
         */
        if (preg_match("/^(_moment)(\d)$/", $sDescriptor, $matches))
        {
            $rgValues = explode(',', $sValues);
            foreach ($rgValues as $k => $dbl)
            {
                if ($dbl < 0.0)
                    $rgValues[$k] = -pow(-$dbl, 1.0 / $matches[2]);
                else
                    $rgValues[$k] = pow($dbl, 1.0 / $matches[2]);
            }
            $sValues = implode(',', $rgValues);
        }
        // Do we know what kind of descriptor this is?
        // The fingerprint files also have a bunch of non-necesary data.
        if (isset($rgixType[$sDescriptor]))
        {
            $ixType = $rgixType[$sDescriptor];
            // Insert data into tblFingerprint
            $oDB->extended->execParam($sqlInsert, array($ixFile, $ixType
                , $sValues), array('integer', 'integer', 'text'));
        }
    }
    $i++;
    if ($i % 100 == 0)
        echo "\t[".date('H:i:s')."] Processed $i files\n";
}
echo "Done!\n";

?>
```

B.3.4 process_descriptor_scores.php

Compares the fingerprints and generates the scores for potential duplicate pairs.

```
<?php
/**
 * This file compares the fingerprints produced by Gunderson's
     descriptors using
```

```
 * Euclidean distance, and generates a score for each fingerprint comparison.
 *
 * Outputs  SQL insert statements to descriptor_inserts.sql that should
 * processed by MySQL at a later stage.
 *
 * @author Vegard Andreas Larsen <vegarl@stud.ntnu.no>
 */
require_once('include/include.php');
ob_end_flush();

$rgMax = array(
    6 => 50.0,              // centroid
    7 => 5.0,               // time
    8 => 0.005,             // msratio
    9 => 25.0,              // steepness
    10 => 2000.0,           // zerocrossings
    11 => 1000.0,           // zerocrossing_guard
    12 => 40.0,             // mfcc_avg
    13 => 3.0,              // mfcc_delta_avg
    18 => 20.0,             // mfcc_f1_avg
    19 => 5.0,              // mfcc_f1_delta_avg

    14 => 50,               // mfcc_delta_moment2
    15 => 12500,            // mfcc_delta_moment3
    16 => 6000000,          // mfcc_delta_moment4
    17 => 4350000000,       // mfcc_delta_moment5

    20 => 1500,             // mfcc_f1_delta_moment2
    21 => 1250000,          // mfcc_f1_delta_moment3
    22 => 1175000000,       // mfcc_f1_delta_moment4
    23 => 1410000000000,    // mfcc_f1_delta_moment5

    24 => 6000,             // mfcc_delta_moment2
    25 => 6100000,          // mfcc_delta_moment3
    26 => 10000000000,      // mfcc_delta_moment4
    27 => 12800000000000,   // mfcc_delta_moment5

    28 => 2000,             // mfcc_moment2
    29 => 1700000,          // mfcc_moment3
    30 => 2900000000,       // mfcc_moment4
    31 => 4100000000000,    // mfcc_moment5
    );


$oDB = CDatabase::Get();

// we can ignore the _delta_ moments of we need to
//$rgTypes = array(14, 15, 16, 17, 20, 21, 22, 23, 24, 25, 26, 27, 28, 29, 30, 31);
$rgTypes = array_keys($rgMax);

foreach ($rgTypes as $ixType)
{
    if (!isset($rgMax[$ixType]))
        continue;

    $fp = fopen("inserts3.sql", "w");

    fwrite($fp, "INSERT DELAYED IGNORE INTO tblScore (ixFile1, ixFile2, ixType, dblScore) VALUES ");
    $fCommaFirst = false;

    $sqlSelectFingerprints = "SELECT ixFile, sFingerprint FROM tblFingerprint WHERE ixType = ? ORDER BY ixFile";
    $cbBefore = memory_get_usage(true);
    foreach ($rgTypes as $ixType)
    {
        // So we don't have to do sqrt() a few million times, we cache
        // the maximum Euclidean distance squared
        $dblMaxDistance[$ixType] = $rgMax[$ixType] * $rgMax[$ixType];
        $rgoFingerprints[$ixType] = $oDB->extended->GetAll($sqlSelectFingerprints, null, array($ixType), array('integer'), MDB2_FETCHMODE_ORDERED);
```

```
        foreach ($rgoFingerprints[$ixType] as $k => $f)
        {
            $rgoFingerprints[$ixType][$k][1] = explode(',', $rgoFingerprints[$ixType][$k][1]);
        }
    }
    $cbAfter = memory_get_usage(true);

    echo (($cbAfter - $cbBefore)/1024)."kB memory in use before starting\n";

    $cFingerprints = count($rgoFingerprints[$rgTypes[0]]);
    $cInserts = 0;
    for ($i = 0; $i < $cFingerprints; $i++)
    {
        echo "\ti=$i\n";
        for ($j = $i + 1; $j < $cFingerprints; $j++)
        {
            foreach ($rgTypes as $ixType)
            {
                $dblSum = 0;
                /* Calculate the Euclidean squared.
                 * This could have been in a method
                 * call, but as we are inside our
                 * third loop already, we'll keep the
                 * overhead to a minimum.
                 */
                foreach ($rgoFingerprints[$ixType][$i][1] as $k => $dblValue1)
                {
                    $dblDiff = $dblValue1 - $rgoFingerprints[$ixType][$j][1][$k];
                    $dblSum += $dblDiff * $dblDiff;
                }

                // Is the Euclidean distance acceptably small?
                if ($dblSum < $dblMaxDistance[$ixType])
                {
                    // Only take the sqrt() if necessary
                    $dblEucl = sqrt($dblSum);
                    $ixFile1 = $rgoFingerprints[$ixType][$i][0];
                    $ixFile2 = $rgoFingerprints[$ixType][$j][0];
                    $dblScore = 1.0 - ($dblEucl / $rgMax[$ixType]);
                    if ($fCommaFirst)
                        $sOut .= ",\n";
                    else
                        $fCommaFirst = true;

                    $cInserts++;
                    $sOut .= "($ixFile1, $ixFile2, $ixType, $dblScore)";
                }
            }
        }
        // Let's have a maximum of 100 value groups in each INSERT.
        if ($cInserts >= 100)
        {
            fwrite($fp, $sOut);
            $sOut = '';
            fwrite($fp, ";\n\n");
            fwrite($fp, "INSERT DELAYED IGNORE INTO tblScore (ixFile1, ixFile2, ixType, dblScore) VALUES ");
            $fCommaFirst = false;
            echo "Wrote $cInserts inserts to file.\n";
            $cInserts = 0;
        }
    }

    // If we have something more to write, let's do it now.
    fwrite($fp, $sOut);
    $sOut = '';
    fwrite($fp, ";\n\n");
    fclose($fp);
    echo "Done with everything.\n";
}
```

```
?>
```

B.4 Combining descriptors

B.4.1 duplicate_functions.php

This script contains functions to check if two file names are likely to be the same song.

```
<?php
/**
 * This script contains a variety of functions for comparing two file
 * names against each other, and determining if they could be the same
 * song.
 *
 * @author Vegard Andreas Larsen <vegarl@stud.ntnu.no>
 */
function tokenize_filename($s)
{
    // remove anything before a space-hyphen-space sequence
    // (often artist or album name)
    $s = preg_replace("/.* - /i", '', $s);
    $s = preg_replace("/[-\s\d]+/", ' ', strtolower($s));
    $rgTokens = split(' ', $s);
    foreach ($rgTokens as $key => $value)
    {
        if (strlen($value) <= 2)
            unset($rgTokens[$key]);
    }
    return $rgTokens;
}

function filename($s)
{
    return pathinfo($s, PATHINFO_FILENAME);
}

function artistname($s)
{
    $sDir = pathinfo($s, PATHINFO_DIRNAME);
    $rgDirs = explode('/', $sDir);
    return strtolower($rgDirs[3]);
}

function compare_filename($s1, $s2, $rgTokens1 = NULL, $rgTokens2 = NULL)
{
    if ($rgTokens1 == NULL)
        $rgTokens1 = tokenize_filename(filename($s1));
    if ($rgTokens2 == NULL)
        $rgTokens2 = tokenize_filename(filename($s2));
    $cHits = 0;
    $rgAllTokens = array_unique(array_merge($rgTokens1, $rgTokens2));
    foreach ($rgAllTokens as $token)
    {
        if (in_array($token, $rgTokens1) && in_array($token, $rgTokens2))
            $cHits++;
    }
    $cTokens = max(count($rgTokens1), count($rgTokens2));
    if ($cTokens < 1) return false;

    return ((artistname($s1) == artistname($s2) &&
        ($cHits / (float) $cTokens) > 0.6));
```

```
}
?>
```

B.4.2 estimate_filename_accuracy.php

This script attempts to match 500 000 random combinations of file names in the database, and prints out the ones that match. See 3.1.3.

```
<?php
/**
 * This script picks 500 000 random files to compare, and checks
 * if they have file names that match. The file names that match
 * printed to stdout, and was verified by a human.
 *
 * @author Vegard Andreas Larsen <vegarl@stud.ntnu.no>
 */
require_once('include/include.php');
$oDB = CDatabase::Get();
$oSmarty = CSmarty::Get();

include('duplicate_finder.php');

$sqlSelectFile = "SELECT ixFile, sPath FROM tblFile";
$rgoFiles = $oDB->extended->GetAssoc($sqlSelectFile, null, null, null, MDB2_FETCHMODE_ASSOC, false);

$rgoTokens = array();
foreach ($rgoFiles as $ixFile => $sPath)
{
    $rgoTokens[$ixFile] = tokenize_filename(filename($sPath));
}

$cNumMatch = 0;
for ($i = 0; $i < 500000; $i++)
{
    $x = rand(0, count($rgoFiles));
    $y = rand(0, count($rgoFiles));
    while ($x == $y || !isset($rgoFiles[$x]) || !isset($rgoFiles[$y]))
    {
        $x = rand(0, count($rgoFiles));
        $y = rand(0, count($rgoFiles));
    }
    if (compare_filename($rgoFiles[$x], $rgoFiles[$y], $rgoTokens[$x], $rgoTokens[$y]))
    {
        $cNumMatch++;
        echo "<br/>\n".str_replace("/mnt/media/", '', $rgoFiles[$x])."<br/>\n";
        echo str_replace("/mnt/media/", '', $rgoFiles[$y])."<br/>\n";
    }
}

echo $cNumMatch."\n";
?>
```

B.4.3 estimate_duplicate_count.php

This script finds the number of potential duplicate pairs in the music collection by looking at their file names. See 3.1.3.

```
<?php
/**
 * This script finds the number of potential duplicates
 * when only looking at the file names.
 *
 * @author Vegard Andreas Larsen <vegarl@stud.ntnu.no>
 */
require_once('include/include.php');
$oDB = CDatabase::Get();
```

```

include('duplicate_functions.php');

$sqlSelectFile = "SELECT ixFile, sPath FROM tblFile ORDER BY ixFile ASC"
    ;
$rgoFiles = $oDB->extended->GetAssoc($sqlSelectFile, null, null, null,
    MDB2_FETCHMODE_ASSOC, false);

/*
 * Pre-process the file names into tokens, since we
 * will be doing about 45M checks.
 */
$rgoTokens = array();
$ixFileMax = 0;
foreach ($rgoFiles as $ixFile => $sPath)
{
    $rgoTokens[$ixFile] = tokenize_filename(filename($sPath));
    $ixFileMax = $ixFile;
}

/*
 * Start counting duplicates.
 */
$cDupCount = 0;
$rgoMatch = array();
for ($i = 0; $i <= $ixFileMax; $i++)
{
    // Progress indicator
    // echo "$i\t$cDupCount\n";

    if (!isset($rgoFiles[$i])) continue;

    for ($j = $i + 1; $j <= $ixFileMax; $j++)
    {
        if (!isset($rgoFiles[$j])) continue;

        if (compare_filename($rgoFiles[$i], $rgoFiles[$j],
            $rgoTokens[$i], $rgoTokens[$j]))
        {
            $cDupCount++;
        }
    }
}
echo $cDupCount;
?>
```

B.4.4 calculate_summed_score.php

This script combines the scores from individual descriptors into combined scores.

```
<?php
/**
 * This script combines match scores for several descriptors using the
 * methods outlined in the thesis. It needs a lot of memory, probably
 * around 512MB or so.
 *
 * Inserts the results directly into MySQL.
 *
 * @author Vegard Andreas Larsen <vegarl@stud.ntnu.no>
 */
require_once('include/include.php');
$oDB = CDatabase::Get();

define('C_AM', 1); // arithmetic mean
define('C_RMS', 2); // root-mean square
define('C_WAM_SUM', 3); // weighted arithmetic mean by sum
define('C_TM', 4); // truncated mean
define('C_WAM_COUNT', 5); // weighted arithmetic mean by count
define('C_WAM_MAN1', 6); // weighted arithmetic mean by manual weights 1
define('C_WAM_PERF', 7); // weighted arithmetic mean by performance
define('C_WAM_AVG', 8); // weighted arithmetic mean by average
define('C_WAM_MAN2', 9); // weighted arithmetic mean by manual weights 2
```

```
define('C_WAM_MAN3', 10);// weighted arithmetic mean by manual weights 2
define('C_BAYES', 11); // naÃ¯ve Bayes
define('C_WRMS', 18); // weighted root mean square
/*
 * Methods operating on a limited set of descriptors.
 */
define('C_BAYES_CHOSEN', 12); // naÃ¯ve Bayes
define('C_AM_CHOSEN', 14); // arithmetic mean
define('C_RMS_CHOSEN', 15); // root mean square
define('C_TM_CHOSEN', 16); // truncated mean
define('C_MEDIAN_CHOSEN', 17); // median
define('C_WRMS_CHOSEN', 19); // weighted RMS

define('BAYES_PROB', 0.8);

define('CUTOFF_SCORE', 0.2);

define('FILES_IN_PASS', 25);


/**
 * Root mean square
 */
function calculate_rms($rgValues, $cCount)
{
    $dblSum = 0;
    foreach ($rgValues as $dblValue)
        $dblSum += ($dblValue * $dblValue);
    return sqrt($dblSum / (float) $cCount);
}

/**
 * Arithmetic mean
 */
function calculate_am($rgValues, $cCount)
{
    return array_sum($rgValues) / (float) $cCount;
}

/**
 * Weighted arithmetic mean
 */
function calculate_wam($rgValues, $rgWeight, $dblSumWeights)
{
    if ($dblSumWeights == 0) return 0;
    $dblSum = 0;
    foreach ($rgValues as $ix => $dbValue)
    {
        $dblSum += ($dbValue * $rgWeight[$ix]);
    }
    return $dblSum / $dblSumWeights;
}

/**
 * Weighted root mean square
 */
function calculate_wrms($rgValues, $rgWeight, $dblSumWeights)
{
    $dblSum = 0;
    foreach ($rgValues as $ix => $dblValue)
        $dblSum += $rgWeight[$ix] * ($dblValue * $dblValue);
    return sqrt($dblSum / (float) $dblSumWeights);
}

/**
 * Median
 */
function calculate_median($rgValues, $cCount)
{
    array_pad($rgValues, $cCount, 0);
    sort($rgValues);
    while (count($rgValues) > 2)
    {
```

```
        array_pop($rgValues);
        array_shift($rgValues);
    }
    if (count($rgValues) == 1)
        return array_pop($rgValues);
    else
        return array_sum($rgValues) / count($rgValues);
}

/**
 * Truncated arithmetic mean.
 */
function calculate_truncated_mean($rgValues, $cCount)
{
    // make sure 0-values are counted
    array_pad($rgValues, $cCount, 0);
    // sort
    sort($rgValues);
    // remove last and first
    array_pop($rgValues);
    array_shift($rgValues);
    return calculate_am($rgValues, $cCount - 2);
}

/**
 * Naive Bayes
 */
function calculate_bayes($rgValues)
{
    $dblProd = 1; $dblInvProd = 1;
    foreach ($rgValues as $dblValue)
    {
        $dblValue = max(0.01, min(0.99, $dblValue));
        $dblProd *= $dblValue;
        $dblInvProd *= (1 - $dblValue);
    }
    $dblProd /= BAYES_PROB;
    $dblInvProd /= (1 - BAYES_PROB);
    $dblSum = $dblProd + $dblInvProd;
    if ($dblSum != 0)
        return $dblProd / $dblSum;
    else
        return 0;
}

/**
 * Standard deviation
 */
function calculate_sd($dblMean, $rgValues, $cCount)
{
    $dblSum = 0;
    foreach ($rgValues as $ix => $dblValue)
    {
        $dblSum += ($dblValue - $dblMean) * ($dblValue - $dblMean);
    }
    return sqrt($dblSum / (float) $cCount);
}

/*
 * Which weights do we need?
 */
$rgWAMs = array(C_WAM_SUM, C_WAM_COUNT, C_WAM_AVG, C_WAM_MAN1,
    C_WAM_PERF, C_WAM_MAN2, C_WAM_MAN3);

/*
 * Let's figure out the weights for the weighted arithmetic means.
 */
$sqlSelectWeight = "SELECT ixType, dblWeightSum, dblWeightCount,
    dblWeightAvg, ".
  "dblWeightMan1, dblWeightMan2, dblWeightMan3, dblWeightPerformance ".
  "FROM tblType";
$rW =& $oDB->query($sqlSelectWeight);
$rgWeight = array();
```

```
$cCount = 0;
while ($r = $rW->fetchRow())
{
    $rgWeight[C_WAM_SUM][$r->ixType] = $r->dblWeightSum;
    $rgWeight[C_WAM_COUNT][$r->ixType] = $r->dblWeightCount;
    $rgWeight[C_WAM_MAN1][$r->ixType] = $r->dblWeightMan1;
    $rgWeight[C_WAM_MAN2][$r->ixType] = $r->dblWeightMan2;
    $rgWeight[C_WAM_MAN3][$r->ixType] = $r->dblWeightMan3;
    $rgWeight[C_WAM_PERF][$r->ixType] = $r->dblWeightPerformance;
    $rgWeight[C_WAM_AVG][$r->ixType] = $r->dblWeightAvg;
    $cCount++;
}
/*
 * Build an array of weight sums, indexed by which set of weights we are using
 * This is used later on for the weighted arithmetic mean, we'll just cache
 * this for later use.
 */
foreach ($rgWAMs as $ixWAM)
{
    $rgSumWeights[$ixWAM] = array_sum($rgWeight[$ixWAM]);
}
$rW->free();

/*
 * Determine the number of files. (Or more accurately, the range of ixFile)
 */
$cNumFiles = $oDB->extended->GetOne("SELECT MAX(ixFile) FROM tblFile");

/*
 * This section allows us to automatically start multiple subprocesses
 * of this script, to use more than one processor, and automatically
 * dividing the work load.
 *
 * Examples:
 *   $ php calculate_summed_score.php 4
 *     Runs 4 subprocesses. Output in output.txt.
 *   $ php calculate_summed_score.php
 *     Runs in 1 subprocess. Output to stdout.
 *   $ php calculate_summed_score.php 40 50 "one"
 *     Runs one process, calculating for files 40-90, using the
 *     process identifier "one".
 */
$ixRow = 0;
$sProcess = '';
if ($argc == 4 && is_numeric($argv[1]) && is_numeric($argv[2]))
{
    // use parameters as range to use
    $ixRow = (int) $argv[1];
    $cNumFiles = (int) $argv[2];
    $sProcess = $argv[3];
}
else if ($argc == 2 && is_numeric($argv[1]))
{
    // we are being asked to use multiple processes
    $cNumProc = (int) $argv[1];

    $cFilesPerProc = (int) ($cNumFiles / $cNumProc);
    // make sure we are a multiple of FILES_IN_PASS
    $cFilesPerProc += FILES_IN_PASS - ($cFilesPerProc % FILES_IN_PASS);

    exec('rm output.txt');

    $cmd = "php calculate_summed_score.php %d %d '%s' >> output.txt &";
    $n = 1;
    for ($i = 0; $i < $cNumFiles; $i += $cFilesPerProc)
    {
        $max = min($i + ($cFilesPerProc - 1), $cNumFiles);
        echo "Launching subprocess ($i -> $max)...\n";
        exec(sprintf($cmd, $i, $max, $n));
        $n++;
    }
```

```

    die();
}

echo "[$sProcess] Starting to sum ($cNumFiles files in total)...\n";

/*
 * Start calculating everything.
 */

$rgSelectedTypes = array(1,2,3,4,12,13,18,19);
$cSelectedCount = count($rgSelectedTypes);
while ($ixRow <= $cNumFiles)
{
    $sqlSelectScores = "SELECT ixFile1, ixFile2, ixType, dblScore ".
        "FROM tblScore WHERE ixFile1 BETWEEN $ixRow AND ".
        ($ixRow + FILES_IN_PASS - 1)." ORDER BY ixFile1, ixFile2";
    $res =& $oDB->query($sqlSelectScores);

    while ($row = $res->fetchRow())
    {
        $sKey = $row->ixFile1.'-'.$row->ixFile2;
        if ($sPrevKey != $sKey && !empty($sPrevKey))
        {
            // remember that $row has changed, so we can't look at those
                values
            list($ixFile1, $ixFile2) = split('-', $sPrevKey);

            /*
             * First run everything on the set of all descriptors
             */
            $dblRMS = calculate_rms($rgValues, $cCount);
            if ($dblRMS > CUTOFF_SCORE)
            {
                $dblRMS_SD = calculate_sd($dblRMS, $rgValues, $cCount);
                $rgInsert[] = '('.$ixFile1.', '.$ixFile2.', '.C_RMS.', '
                    .$dblRMS.','.$dblRMS_SD.')';
            }

            $dblWRMS = calculate_wrms($rgValues, $rgWeight[C_WAM_MAN2],
                $rgSumWeights[C_WAM_MAN2]);
            if ($dblWRMS > CUTOFF_SCORE)
            {
                $dblWRMS_SD = calculate_sd($dblWRMS, $rgValues,
                    $rgSumWeights[C_WAM_MAN2]);
                $rgInsert[] = '('.$ixFile1.', '.$ixFile2.', '.C_WRMS.',
                    '.$dblWRMS.','.$dblWRMS_SD.')';
            }

            $dblAM = calculate_am($rgValues, $cCount);
            if ($dblAM > CUTOFF_SCORE)
            {
                $dblAM_SD = calculate_sd($dblAM, $rgValues, $cCount);
                $rgInsert[] = '('.$ixFile1.', '.$ixFile2.', '.C_AM.', '.
                    $dblAM.','.$dblAM_SD.')';
            }

            foreach ($rgWAMs as $ixWAM)
            {
                $dblWAM = calculate_wam($rgValues, $rgWeight[$ixWAM],
                    $rgSumWeights[$ixWAM]);
                if ($dblWAM > CUTOFF_SCORE)
                {
                    $dblWAM_SD = calculate_sd($dblWAM, $rgValues,
                        $rgSumWeights[$ixWAM]);
                    $rgInsert[] = '('.$ixFile1.', '.$ixFile2.', '.$ixWAM
                        .', '.$dblWAM.','.$dblWAM_SD.')';
                }
            }

            $dblTM = calculate_truncated_mean($rgValues, $cCount);
            if ($dblTM > CUTOFF_SCORE)
            {
```

```
            $dblTM_SD = calculate_sd($dblTM, $rgValues, $cCount - 2);
            $rgInsert[] = '('.$ixFile1.', '.$ixFile2.', '.C_TM.', '.$dblTM.','.$dblTM_SD.')';
        }

        $dblBayes = calculate_bayes($rgValues)
        if ($dblBayes > CUTOFF_SCORE)
        {
            $rgInsert[] = '('.$ixFile1.', '.$ixFile2.', '.C_BAYES.', '.$dblBayes.',0)';
        }

        /*
         * Lets build a separate value array for the set of chosen descriptors.
         * This allows using the same functions to calculate summed scores.
         */
        $rgSelectedValues = array();
        foreach ($rgSelectedTypes as $ixType)
        {
            $rgSelectedValues[$ixType] = $rgValues[$ixType];
        }

        /*
         * Calculations for the chosen descriptors.
         */
        $dblMed = calculate_median($rgSelectedValues, $cSelectedCount);
        if ($dblMed > CUTOFF_SCORE)
        {
            $dblMed_SD = calculate_sd($dblMed, $rgSelectedValues, $cSelectedCount);
            $rgInsert[] = '('.$ixFile1.', '.$ixFile2.', '.C_MEDIAN_CHOSEN.', '.$dblMed.','.$dblMed_SD.')';
        }

        $dblRMS = calculate_rms($rgSelectedValues, $cSelectedCount);
        if ($dblRMS > CUTOFF_SCORE)
        {
            $dblRMS_SD = calculate_sd($dblRMS, $rgSelectedValues, $cSelectedCount);
            $rgInsert[] = '('.$ixFile1.', '.$ixFile2.', '.C_RMS_CHOSEN.', '.$dblRMS.','.$dblRMS_SD.')';
        }

        /*
         * Why is the WRMS here, yet not the WAM? Because the WAM was calculated using
         * all sets of weights, but we only chose one set of weights for the WRMS.
         * This means that when limiting the set of descriptors, yet using the same weights
         * we effectively get the third set of manual weights. Weird, yet correct.
         */
        $dblWRMS = calculate_wrms($rgSelectedValues, $rgWeight[C_WAM_MAN2], $rgSumWeights[C_WAM_MAN2]);
        if ($dblWRMS > CUTOFF_SCORE)
        {
            $dblWRMS_SD = calculate_sd($dblWRMS, $rgSelectedValues, $rgSumWeights[C_WAM_MAN2]);
            $rgInsert[] = '('.$ixFile1.', '.$ixFile2.', '.C_WRMS_CHOSEN.', '.$dblWRMS.','.$dblWRMS_SD.')';
        }

        $dblAM = calculate_am($rgSelectedValues, $cSelectedCount);
        if ($dblAM > CUTOFF_SCORE)
        {
            $dblAM_SD = calculate_sd($dblAM, $rgSelectedValues, $cSelectedCount);
            $rgInsert[] = '('.$ixFile1.', '.$ixFile2.', '.C_AM_CHOSEN.', '.$dblAM.','.$dblAM_SD.')';
```

```
                }

                $dblTM = calculate_truncated_mean($rgSelectedValues,
                    $cSelectedCount);
                if ($dblTM > CUTOFF_SCORE)
                {
                    $dblTM_SD = calculate_sd($dblTM, $rgSelectedValues,
                        $cSelectedCount - 2);
                    $rgInsert[] = '('.$ixFile1.', '.$ixFile2.', '.
                        C_TM_CHOSEN.', '.$dblTM.','.$dblTM_SD.')';
                }

                $dblBayes = calculate_bayes($rgSelectedValues)
                if ($dblBayes > CUTOFF_SCORE)
                {
                    $rgInsert[] = '('.$ixFile1.', '.$ixFile2.', '.
                        C_BAYES_CHOSEN.', '.$dblBayes.',0)';
                }

                /*
                 * Reset the value array for the next run.
                 */
                unset($rgValues);
                $rgValues = array();
            }
            $rgValues[$row->ixType] = $row->dblScore;
            $sPrevKey = $sKey;
        }
        $res->free();

        /*
         * Insert all our cumulated results. Uses INSERT DELAYED since we
             don't need
         * to inspect this right away.
         */
        $sql = "INSERT DELAYED INTO tblSummedScore (ixFile1, ixFile2,
            ixScoreMethod, dblScore, dblSD) VALUES ";
        $sql.= implode(', ', $rgInsert);
        $oDB->query($sql);

        $i += count($rgInsert);
        $rgInsert = array();
        $ixRow += FILES_IN_PASS;
        echo "\t[".date('H:i:s')."] [$sProcess] Processed files starting
            from $ixRow ($i cumulative database inserts).\n";
}

echo "[$sProcess] Done!\n";

echo "[$sProcess] Entered $i scores into tblSummedScore!\n";
echo "[$sProcess] Finished!\n";
?>
```

B.4.5 find_index.php

This script finds the average distance within verified duplicate pairs for the simple descriptors song length, centroid, steepness, mean/square ratio and the rate of zero crossings. These are all scalar descriptors that can be indexed by. See 4.5 for motivation.

```
<?php
/**
 * This script finds the average -- and the minimum and maximum --
 * distance for verified duplicates with the simple descriptors
 * song length, centroid, msratio, steepness and zerocrossings.
 * It also lists the percentage of verified duplicates that will
 * be found when searching within 2 times the average distance
 * for a given descriptor.
 *
 * @author Vegard Andreas Larsen <vegarl@stud.ntnu.no>
```

```
 */
require_once('include/include.php');
$oDB = CDatabase::Get();

/*
 * Get a list of descriptor names.
 */
$sql = "SELECT ixType, sType FROM tblType";
$rgsTypes = $oDB->extended->GetAssoc($sql, null, null, null,
    MDB2_FETCHMODE_ASSOC, false);

/*
 * Find all the verified matches.
 */
$sql = "SELECT * FROM tblVerifiedMatch";
$rgoVerified = $oDB->extended->GetAll($sql);
$rgVerified = array();
foreach ($rgoVerified as $o)
{
    $rgVerified[$o->ixFile1][$o->ixFile2] = true;
    $rgVerified[$o->ixFile2][$o->ixFile1] = true;
}
$rgoVerified = null;
unset($rgoVerified);

function array_average($rg)
{
    return (array_sum($rg) / (float) count($rg));
}

/*
 * The descriptors to look at.
 */
$rgixType = array(6, 7, 8, 9, 10, 11);
foreach ($rgixType as $ixType)
{
    /*
     * Find all the fingerprints for the verified matches.
     */
    $sql =
        "SELECT f1.sFingerprint AS fp1, f2.sFingerprint AS fp2, ixFile1
            ".
        "FROM tblVerifiedMatch v, tblFingerprint f1, tblFingerprint f2 "
            .
        "WHERE f1.ixFile = v.ixFile1 AND f2.ixFile = v.ixFile2 AND ".
        "f1.ixType = ? AND f2.ixType = f1.ixType";
    $rgoFingerprint = $oDB->extended->GetAll($sql, null,
        array($ixType), array('integer'));

    /*
     * Find the distance for each verified match.
     * This cannot be done in MySQL, because the column
     * that stores the fingerprints is in binary format,
     * to accomodate storing of all types of fingerprints,
     * and MySQL wouldn't know what to do with a
     * subtraction of binary data.
     */
    $rgDiff = array();
    foreach ($rgoFingerprint as $oF)
    {
        $rgDiff[] = abs($oF->fp1 - $oF->fp2);
    }
    sort($rgDiff);
    $avg = array_average($rgDiff);
    $min = $rgDiff[0];
    $max = round($rgDiff[count($rgDiff)-1], 2);

    /*
     * Now lets figure out how many fingerprints we would
     * need to search (on average) when limiting our search
     * to fingerprints within +/- 2 * $avg, and how many of
     * the results we find are verified.
     */
```

```
    $sql = "SELECT ixFile, sFingerprint FROM tblFingerprint WHERE ixType
        = ?";
    $rgFingerprint = $oDB->extended->GetAll($sql, null,
        array($ixType), array('integer'));
    $rgixSearchFor = array_keys($rgoFingerprint);
    $rgValues = array();
    $rgNumVerified = array();
    $delta = $avg * 2;
    foreach ($rgixSearchFor as $ix)
    {
        $dblValue = $rgoFingerprint[$ix]->fp1;
        $ixFile = $rgoFingerprint[$ix]->ixFile1;
        $c = 0;
        $cNumVerified = 0;
        foreach ($rgFingerprint as $oFingerprint)
        {
            $dblFingerprint = (float) $oFingerprint->sFingerprint;
            if ($dblFingerprint < ($dblValue + $delta) &&
                $dblFingerprint > ($dblValue - $delta))
            {
                $c++;
                $ixFile2 = (float) $oFingerprint->ixFile;
                if (isset($rgVerified[$ixFile][$ixFile2]))
                    $cNumVerified++;
            }
        }
        $rgValues[] = $c;
        $rgNumVerified[] = $cNumVerified / count($rgVerified[$ixFile]);
    }
    $c = round(array_average($rgValues), 0);
    $dblFindRate = round(array_average($rgNumVerified), 4);

    echo "avg: $avg\tmin: $min\tmax: $max\tsearch: $c\t"
        ."find rate: $dblFindRate\t\t".$rgsTypes[$ixType]."\n";

?>
```

B.4.6 find_num_correct.php

This script finds the number of verified duplicate pairs in partitions of 100 duplicates, for all the individual descriptors and all the combined solutions.

```
<?php
/**
 * This script finds the number of correct duplicates,
 * divided into partitions of 100 duplicates, for each
 * individual descriptor and each combined solution.
 *
 * @author Vegard Andreas Larsen <vegarl@stud.ntnu.no>
 */
require_once('include/include.php');
$cNumRows = 1000;

/*
 * HTML table headers
 */
?>
<table border="1">
  <tr>
    <th>Type/method</th>
    <th>Name</th>
<?php
    for ($i = 1; $i <= 10; $i++)
    {
        echo "<th>".(($i-1)*100+1)."-".($i*100)."</th>\n";
    }
?>
    <th>%</th>
    <th>Errors in first <?= $cNumRows ?></th>
    <th>First error at #</th>
  </tr>
```

```
<?php
$oDB = CDatabase::Get();

$cNumFetchRows = $cNumRows;


/*
 * Get a list of the verified duplicates.
 */
$sql = "SELECT * FROM tblVerifiedMatch";
$rgoVerified = $oDB->extended->GetAll($sql);
$rgVerified = array();
foreach ($rgoVerified as $o)
{
    $rgVerified[$o->ixFile1][$o->ixFile2] = true;
}
$rgoVerified = null;
unset($rgoVerified);

function display($ixMethod, $sTitle, $rgoScores)
{
    global $rgVerified;
    global $cNumRows;
?>
  <tr>
    <td><?= $ixMethod ?></td>
    <td><?= $sTitle ?></td>
<?php

    $rgoScoresVerified = array();
    $rgoScoresWrong = array();
    $i = 0;
    $cWrong = 0;
    $ixFirstWrong = 0;
    $g = 0; // counter
    $cWrongInPartition = 0;
    for ($z = 0; $z < $cNumRows; $z++)
    {
        $o = $rgoScores[$z];
        if (isset($rgoScores[$z]) &&
            $rgVerified[$o->ixFile1][$o->ixFile2])
        {
            $rgoScoresVerified[] = $o;
        }
        else
        {
            $rgoScoresWrong[] = $o;
            if (empty($ixFirstWrong))
                $ixFirstWrong = $i;
            if ($i < $cNumRows)
            {
                $cWrong++;
                $cWrongInPartition++;
            }
        }
        unset($rgoScores[$k]);
        $i++;
        if ($i % 100 == 0)
        {
            $g++;
            echo "<td>$cWrongInPartition</td>\n";
            $cWrongInPartition = 0;
        }
    }

?>
    <td><?= $cWrong / $cNumRows ?></td>
    <td><?= $cWrong ?></td>
    <td><?= $ixFirstWrong ?></td>
  </tr>
<?php
}

/*
```

```
 * A list of the names of all the scoring methods.
 */
$sqlSelectTitles = "SELECT ixScoreMethod, sTitle FROM tblScoreMethods";
$rgsTitle = $oDB->extended->GetAssoc($sqlSelectTitles, null, null, null,
    MDB2_FETCHMODE_ASSOC, false);
// Get scores for all the scoring methods
$rgixScoreMethods = array_keys($rgsTitle);

/*
 * Now let's examine them.
 */
$sql = "SELECT dblScore, dblSD, ixFile1, ixFile2 ".
    "FROM tblSummedScore ".
    "WHERE ixScoreMethod = ? ".
    "ORDER BY dblScore DESC, dblSD ASC ".
    "LIMIT 0, $cNumFetchRows";

foreach ($rgixScoreMethods as $ixScoreMethod)
{
    $sTitle = $rgsTitle[$ixScoreMethod];
    $rgoScores = $oDB->extended->GetAll($sql, null,
        array($ixScoreMethod), array('integer'));
    display('C'.$ixScoreMethod, $sTitle, $rgoScores);
}

/*
 * Let's look at the individual descriptors
 */

$sqlSelectTypes = "SELECT ixType, sType FROM tblType";
$rgoTypes = $oDB->extended->GetAssoc($sqlSelectTypes, null, null, null,
    MDB2_FETCHMODE_ASSOC, false);

$sql = "SELECT ixFile1, ixFile2, dblScore ".
    "FROM tblScore WHERE ixType = ? ".
    "ORDER BY dblScore DESC LIMIT 0, $cNumFetchRows";

foreach ($rgoTypes as $ixType => $sType)
{
    $rgoScores = $oDB->extended->GetAll($sql, null,
        array($ixType), array('integer'));
    display('I'.$ixType, $sType, $rgoScores);
}

?>
</table>
```

B.4.7 display_duplicates.php

This script displays the potential duplicate pairs, and allows for basic filtering based on whether the file names match, or if the duplicate pair have been verified.

```
<?php
/**
 * This script display potential duplicates in the database,
 * and highlights the duplicates that match by file name,
 * or have been verified to be correct.
 *
 * Options:
 *   fVerified=1        -- show only verified duplicates
 *   fVerified=0        -- show only non-verified duplicates
 *   fMatch=1           -- show only matching file names
 *   fMatch=0           -- show only non-matching file names
 *   ixPage=<page>      -- skip <page> * 1000 results
 *
 * If fVerified and fMatch is not specified, no restrictions
 * are placed on the duplicates status.
 *
 * @author Vegard Andreas Larsen <vegarl@stud.ntnu.no>
 */
```

```
require_once('include/include.php');
$oDB = CDatabase::Get();

$cResultsPerPage = 1000;
$ixPage = 0;
$ixStart = 0;
$ixScoreMethod = 1;

require_once('duplicate_functions.php');

/**
 * Get a list of the verified duplicates.
 */
$sql = "SELECT * FROM tblVerifiedMatch";
$rgoVerified = $oDB->extended->GetAll($sql);
$rgVerified = array();
foreach ($rgoVerified as $o)
{
    $rgVerified[$o->ixFile1][$o->ixFile2] = true;
}
$rgoVerified = null;
unset($rgoVerified);

$sqlSelectFile = "SELECT ixFile, sPath FROM tblFile";
$rgoFiles = $oDB->extended->GetAssoc($sqlSelectFile, null, null, null,
    MDB2_FETCHMODE_ASSOC, false);
$rgoFiles = str_replace('/home/vegard/Music/', '', $rgoFiles);
$rgoFiles = str_replace('/mnt/media/', '', $rgoFiles);


/**
 * Now lets get all the scores from the database
 */

// Lower bound for values extracted.
$dblScoreLimit = 0.05;
if (isset($_GET['ixType']))
{
    $ixType = (int) $_GET['ixType'];
    $sqlSelectScores = "SELECT ixFile1, ixFile2, dblScore ".
        "FROM tblScore WHERE ".
        "dblScore > $dblScoreLimit AND ixType = $ixType ".
        "ORDER BY dblScore DESC LIMIT ?, ?";
}
elseif ($_GET['ixScoreMethod'])
{
    $ixScoreMethod = (int) $_GET['ixScoreMethod'];
    $sqlSelectScores = "SELECT ixFile1, ixFile2, dblScore ".
        "FROM tblSummedScore ".
        "WHERE dblScore > $dblScoreLimit ".
        "AND ixScoreMethod = $ixScoreMethod ".
        "ORDER BY dblScore DESC LIMIT ?, ?";
}
else
{
    /**
     * No type specified, display a menu of choices.
     */
    echo '<h1>Combined methods</h1>';
    $sql = "SELECT ixScoreMethod, sTitle FROM tblScoreMethods";
    $r = $oDB->extended->GetAll($sql);
    foreach ($r as $row)
    {
        echo '<a href="?ixScoreMethod='.$row->ixScoreMethod.'">'.
            $row->sTitle.'</a><br/>';
    }

    echo '<h1>Individual descriptors</h1>';
    $sql = "SELECT ixType, sType FROM tblType";
    $r = $oDB->extended->GetAll($sql);
    foreach ($r as $row)
    {
        echo '<a href="?ixType='.$row->ixType.'">'.
            $row->sType.'</a><br/>';
```

```
    }
    die();
}

/**
 * Allow parameters to change what is fetched.
 */
if ($_GET['ixPage'])
{
    $ixPage = (int) $_GET['ixPage'];
    if ($ixPage < 0) $ixPage = 0;
    $ixStart = $cResultsPerPage * $ixPage;
}

/**
 * Go fetch!
 */
$res = $oDB->extended->GetAll($sqlSelectScores, null,
    array($ixStart, $cResultsPerPage),
    array('integer', 'integer'));

/*
 * Output the HTML page headers
 */
?>
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
  "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml" xml:lang="no" lang="no">
  <head>
    <title>Master thesis</title>
    <link rel="stylesheet" type="text/css" href="css/main.css" />
  </head>
  <body>
<form method="post" action="save_verified.php">
<input type="hidden" name="redirect_to"
  value="display_unverified_scores.php<?php
echo (isset($ixType) ? "?ixType=$ixType" : "?ixScoreMethod=
    $ixScoreMethod");
?>" />
<input type="submit" value="Save verified" />
<table>
  <tr>
    <th><abbr title="Match number">M#</abbr></th>
    <th><abbr title="Do file names match?">M?</abbr></th>
    <th><abbr title="Is this match verified">V?</abbr></th>
    <th><abbr title="Check to verify">?</abbr></th>
    <th><abbr title="File numbers (ixFile)">F#</abbr></th>
    <th>Files</th>
    <th>Score</th>
  </tr>
<?

$j = $ixStart;
foreach ($res as $row)
{

    $fVerified = isset($rgVerified[$row->ixFile1][$row->ixFile2]);

    $fCompares = compare_filename(
        $rgoFiles[$row->ixFile1],
        $rgoFiles[$row->ixFile2]);

    if ((!isset($_GET['fVerified']) ||
        ($_GET['fVerified'] == 1 && $fVerified) ||
        ($_GET['fVerified'] == 0 && !$fVerified))
        &&
        (!isset($_GET['fMatch']) ||
        ($_GET['fMatch'] == 1 && $fCompares) ||
        ($_GET['fMatch'] == 0 && !$fCompares)))
    {
        /*
         * Output ugly HTML table data.
         */
        echo '<tr class="'.($j % 2 == 0 ? 'colored' : '').'">';
```

```
            echo '<td rowspan="2">'.($j+1).'</td>';
            echo '<td rowspan="2"><img src="images/'.
                ($fCompares ? "green" : "red").'" /></td>';
            echo '<td rowspan="2"><img src="images/'.
                ($fVerified ? "green" : "red").'.png" /></td>';
            echo '<td rowspan="2"><input type="checkbox" '.
                'name="match[]" value="'.$row->ixFile1.'-'.
                $row->ixFile2.'" /></td>';
            echo '<td>'.$row->ixFile1.'</td>';
            echo '<td align="right">'.$rgoFiles[$row->ixFile1].'</td>';
            echo '<td rowspan="2">'.($row->dblScore).'</td>';
            echo '</tr>';
            echo '<tr class="bottom '.($j % 2 == 0 ? 'colored' : '').'">';
            echo '<td>'.$row->ixFile2.'</td>';
            echo '<td align="right">'.$rgoFiles[$row->ixFile2].'</td>';
            echo '</tr>';
            $j++;
        }
}

?>
</table>
<input type="submit" value="Save verified" />
</form>

</body>
</html>
```

B.4.8 save_verified.php

This script saves verified duplicate pairs to the database.

```
<?php
/**
 * Simple support script that saves two files as a verified
 * duplicate.
 *
 * @author Vegard Andreas Larsen <vegarl@stud.ntnu.no>
 */
require_once('include/include.php');

$oDB = CDatabase::Get();

$st = $oDB->prepare(
    'INSERT IGNORE INTO tblVerifiedMatch (ixFile1, ixFile2) VALUES (?,
        ?)',
    array('integer', 'integer'), MDB2_PREPARE_MANIP);

/*
 * We get multiple values as an array of strings.
 */
if (isset($_POST['match']))
{
    $rgMatch = $_POST['match'];
    foreach ($rgMatch as $value)
    {
        // The strings are splittable by a hyphen
        list($ixFile1, $ixFile2) = split('-', $value);
        $st->execute(array($ixFile1, $ixFile2));
    }
}

/*
 * Someone might want us to send the user back afterwards.
 */
if (isset($_POST['redirect_to']))
{
    header('Location: '.$_POST['redirect_to']);
    die();
}
?>
```

B.5 Licenses

B.5.1 The GNU General Public License

Version 2, June 1991
Copyright © 1989, 1991 Free Software Foundation, Inc.

51 Franklin Street, Fifth Floor, Boston, MA 02110-1301, USA

Everyone is permitted to copy and distribute verbatim copies of this license document, but changing it is not allowed.

B.5.1.1 Preamble

The licenses for most software are designed to take away your freedom to share and change it. By contrast, the GNU General Public License is intended to guarantee your freedom to share and change free software—to make sure the software is free for all its users. This General Public License applies to most of the Free Software Foundation's software and to any other program whose authors commit to using it. (Some other Free Software Foundation software is covered by the GNU Library General Public License instead.) You can apply it to your programs, too.

When we speak of free software, we are referring to freedom, not price. Our General Public Licenses are designed to make sure that you have the freedom to distribute copies of free software (and charge for this service if you wish), that you receive source code or can get it if you want it, that you can change the software or use pieces of it in new free programs; and that you know you can do these things.

To protect your rights, we need to make restrictions that forbid anyone to deny you these rights or to ask you to surrender the rights. These restrictions translate to certain responsibilities for you if you distribute copies of the software, or if you modify it.

For example, if you distribute copies of such a program, whether gratis or for a fee, you must give the recipients all the rights that you have. You must make sure that they, too, receive or can get the source code. And you must show them these terms so they know their rights.

We protect your rights with two steps: (1) copyright the software, and (2) offer you this license which gives you legal permission to copy, distribute and/or modify the software.

Also, for each author's protection and ours, we want to make certain that everyone understands that there is no warranty for this free software. If the software is modified by someone else and passed on, we want its recipients to know that what they have is not the original, so that any problems introduced by others will not reflect on the original authors' reputations.

Finally, any free program is threatened constantly by software patents. We wish to avoid the danger that redistributors of a free program will individually obtain patent licenses, in effect making the program proprietary. To prevent this, we have made it clear that any patent must be licensed for everyone's free use or not licensed at all.

The precise terms and conditions for copying, distribution and modification follow.

B.5.1.2 Terms and Conditions For Copying, Distribution and Modification

0. This License applies to any program or other work which contains a notice placed by the copyright holder saying it may be distributed under the terms of this General Public License. The "Program", below, refers to any such program or work, and a "work based on the Program" means either the Program or any derivative work under copyright law: that is to say, a work containing the Program or a portion of it, either verbatim or with modifications and/or translated into another language. (Hereinafter, translation is included without limitation in the term "modification".) Each licensee is addressed as "you".

   Activities other than copying, distribution and modification are not covered by this License; they are outside its scope. The act of running the Program is not restricted, and the output from the Program is covered only if its contents constitute a work based on the Program (independent of having been made by running the Program). Whether that is true depends on what the Program does.

1. You may copy and distribute verbatim copies of the Program's source code as you receive it, in any medium, provided that you conspicuously and appropriately publish on each copy an appropriate copyright notice and disclaimer of warranty; keep intact all the notices that refer to this License and to the absence of any warranty; and give any other recipients of the Program a copy of this License along with the Program.

   You may charge a fee for the physical act of transferring a copy, and you may at your option offer warranty protection in exchange for a fee.

2. You may modify your copy or copies of the Program or any portion of it, thus forming a work based on the Program, and copy and distribute such modifications or work under the terms of Section 1 above, provided that you also meet all of these conditions:

   (a) You must cause the modified files to carry prominent notices stating that you changed the files and the date of any change.

   (b) You must cause any work that you distribute or publish, that in whole or in part contains or is derived from the Program or any part thereof, to be licensed as a whole at no charge to all third parties under the terms of this License.

   (c) If the modified program normally reads commands interactively when run, you must cause it, when started running for such interactive use in the most ordinary way, to print or display an announcement including an appropriate copyright notice and a notice that there is no warranty (or else, saying that you provide a warranty) and that users may redistribute the program under these conditions, and telling the user how to view a copy of this License. (Exception: if the Program itself is interactive but does not normally print such an announcement, your work based on the Program is not required to print an announcement.)

These requirements apply to the modified work as a whole. If identifiable sections of that work are not derived from the Program, and can be reasonably considered independent and separate works in themselves, then this License, and its terms, do not apply to those sections when you distribute them as separate works. But when you distribute the same sections as part of a whole which is a work based on the Program, the distribution of the whole must be on the terms of this License, whose permissions for other licensees extend to the entire whole, and thus to each and every part regardless of who wrote it.

Thus, it is not the intent of this section to claim rights or contest your rights to work written entirely by you; rather, the intent is to exercise the right to control the distribution of derivative or collective works based on the Program.

In addition, mere aggregation of another work not based on the Program with the Program (or with a work based on the Program) on a volume of a storage or distribution medium does not bring the other work under the scope of this License.

3. You may copy and distribute the Program (or a work based on it, under Section 2) in object code or executable form under the terms of Sections 1 and 2 above provided that you also do one of the following:

   (a) Accompany it with the complete corresponding machine-readable source code, which must be distributed under the terms of Sections 1 and 2 above on a medium customarily used for software interchange; or,

   (b) Accompany it with a written offer, valid for at least three years, to give any third party, for a charge no more than your cost of physically performing source distribution, a complete machine-readable copy of the corresponding source code, to be distributed under the terms of Sections 1 and 2 above on a medium customarily used for software interchange; or,

   (c) Accompany it with the information you received as to the offer to distribute corresponding source code. (This alternative is allowed only for noncommercial distribution and only if you received the program in object code or executable form with such an offer, in accord with Subsection b above.)

   The source code for a work means the preferred form of the work for making modifications to it. For an executable work, complete source code means all the source code for all modules it contains, plus any associated interface definition files, plus the scripts used to control compilation and installation of the executable. However, as a special exception, the source code distributed need not include anything that is normally distributed (in either source or binary form) with the major components (compiler, kernel, and so on) of the operating system on which the executable runs, unless that component itself accompanies the executable.

   If distribution of executable or object code is made by offering access to copy from a designated place, then offering equivalent access to copy the source code from the same place counts as distribution of the source code, even though third parties are not compelled to copy the source along with the object code.

4. You may not copy, modify, sublicense, or distribute the Program except as expressly provided under this License. Any attempt otherwise to copy, modify, sublicense or distribute the Program is void, and will automatically terminate your rights under this License. However, parties who have received copies, or rights, from you under this License will not have their licenses terminated so long as such parties remain in full compliance.

5. You are not required to accept this License, since you have not signed it. However, nothing else grants you permission to modify or distribute the Program or its derivative works. These actions are prohibited by law if you do not accept this License. Therefore, by modifying or distributing the Program (or any work based on the Program), you indicate your acceptance of this License to do so, and all its terms and conditions for copying, distributing or modifying the Program or works based on it.

6. Each time you redistribute the Program (or any work based on the Program), the recipient automatically receives a license from the original licensor to copy, distribute or modify the Program subject to these terms and conditions. You may not impose any further restrictions on the recipients' exercise of the rights granted herein. You are not responsible for enforcing compliance by third parties to this License.

7. If, as a consequence of a court judgment or allegation of patent infringement or for any other reason (not limited to patent issues), conditions are imposed on you (whether by court order, agreement or otherwise) that contradict the conditions of this License, they do not excuse you from the conditions of this License. If you cannot distribute so as to satisfy simultaneously your obligations under this License and any other pertinent obligations, then as a consequence you may not distribute the Program at all. For example, if a patent license would not permit royalty-free redistribution of the Program by all those who receive copies directly or indirectly through you, then the only way you could satisfy both it and this License would be to refrain entirely from distribution of the Program.

   If any portion of this section is held invalid or unenforceable under any particular circumstance, the balance of the section is intended to apply and the section as a whole is intended to apply in other circumstances.

   It is not the purpose of this section to induce you to infringe any patents or other property right claims or to contest validity of any such claims; this section has the sole purpose of protecting the integrity of the free software distribution system, which is implemented by public license practices. Many people have made generous contributions to the wide range of software distributed through that system in reliance on consistent application of that system; it is up to the author/donor to decide if he or she is willing to distribute software through any other system and a licensee cannot impose that choice.

   This section is intended to make thoroughly clear what is believed to be a consequence of the rest of this License.

8. If the distribution and/or use of the Program is restricted in certain countries either by patents or by copyrighted interfaces, the original copyright holder who places the Program under this License may add an explicit geographical distribution limitation excluding those countries, so that distribution is permitted only in or among countries not thus excluded. In such case, this License incorporates the limitation as if written in the body of this License.

9. The Free Software Foundation may publish revised and/or new versions of the General Public License from time to time. Such new versions will be similar in spirit to the present version, but may differ in detail to address new problems or concerns.

   Each version is given a distinguishing version number. If the Program specifies a version number of this License which applies to it and "any later version", you have the option of following the terms and conditions either of that version or of any later version published by the Free Software Foundation. If the Program does not specify a version number of this License, you may choose any version ever published by the Free Software Foundation.

10. If you wish to incorporate parts of the Program into other free programs whose distribution conditions are different, write to the author to ask for permission. For software which is copyrighted by the Free Software Foundation, write to the Free Software Foundation; we sometimes make exceptions for this. Our decision will be guided by the two goals of preserving the free status of all derivatives of our free software and of promoting the sharing and reuse of software generally.

No Warranty

11. Because the program is licensed free of charge, there is no warranty for the program, to the extent permitted by applicable law. Except when otherwise stated in writing the copyright holders and/or other parties provide the program "as is" without warranty of any kind, either expressed or implied, including, but not limited to, the implied warranties of merchantability and fitness for a particular purpose. The entire risk as to the quality and performance of the program is with you. Should the program prove defective, you assume the cost of all necessary servicing, repair or correction.

12. In no event unless required by applicable law or agreed to in writing will any copyright holder, or any other party who may modify and/or redistribute the program as permitted above, be liable to you for damages, including any general, special, incidental or consequential damages arising out of the use or inability to use the program (including but not limited to loss of data or data being rendered inaccurate or losses sustained by you or third parties or a failure of the program to operate with any other programs), even if such

HOLDER OR OTHER PARTY HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES.

End of Terms and Conditions

Appendix: How to Apply These Terms to Your New Programs

If you develop a new program, and you want it to be of the greatest possible use to the public, the best way to achieve this is to make it free software which everyone can redistribute and change under these terms.

To do so, attach the following notices to the program. It is safest to attach them to the start of each source file to most effectively convey the exclusion of warranty; and each file should have at least the "copyright" line and a pointer to where the full notice is found.

> one line to give the program's name and a brief idea of what it does.
> Copyright (C) yyyy name of author
>
> This program is free software; you can redistribute it and/or modify it under the terms of the GNU General Public License as published by the Free Software Foundation; either version 2 of the License, or (at your option) any later version.
>
> This program is distributed in the hope that it will be useful, but WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU General Public License for more details.
>
> You should have received a copy of the GNU General Public License along with this program; if not, write to the Free Software Foundation, Inc., 51 Franklin Street, Fifth Floor, Boston, MA 02110-1301, USA.

Also add information on how to contact you by electronic and paper mail.

If the program is interactive, make it output a short notice like this when it starts in an interactive mode:

> Gnomovision version 69, Copyright (C) yyyy name of author
> Gnomovision comes with ABSOLUTELY NO WARRANTY; for details type 'show w'.
> This is free software, and you are welcome to redistribute it under certain conditions; type 'show c' for details.

The hypothetical commands `show w` and `show c` should show the appropriate parts of the General Public License. Of course, the commands you use may be called something other than `show w` and `show c`; they could even be mouse-clicks or menu items—whatever suits your program.

You should also get your employer (if you work as a programmer) or your school, if any, to sign a "copyright disclaimer" for the program, if necessary. Here is a sample; alter the names:

Yoyodyne, Inc., hereby disclaims all copyright interest in the program
'Gnomovision' (which makes passes at compilers) written by James Hacker.

signature of Ty Coon, 1 April 1989
Ty Coon, President of Vice

This General Public License does not permit incorporating your program into proprietary programs. If your program is a subroutine library, you may consider it more useful to permit linking proprietary applications with the library. If this is what you want to do, use the GNU Library General Public License instead of this License.

B.5.2 Creative Commons Attribution-Noncommercial-Share Alike 3.0 United States

B.5.2.1 Short text

You are free:

to Share to copy, distribute, display and perform the work

to Remix to make derivative works

Under the following conditions:

Attribution. You must attribute the work in the manner specified by the author or licensor (but not in any way that suggests that they endorse you or your use of the work).

Noncommercial. You may not use this work for commercial purposes.

Share Alike. If you alter, transform, or build upon this work, you may distribute the resulting work only under the same or similar license to this one.

- For any reuse or distribution, you must make clear to others the license terms of this work. The best way to do this is with a link to this web page.
- Any of the above conditions can be waived if you get permission from the copyright holder.
- Apart from the remix rights granted under this license, nothing in this license impairs or restricts the author's moral rights.

B.5.2.2 Full license

THE WORK (AS DEFINED BELOW) IS PROVIDED UNDER THE TERMS OF THIS CREATIVE COMMONS PUBLIC LICENSE ("CCPL" OR "LICENSE"). THE WORK IS PROTECTED BY COPYRIGHT AND/OR OTHER

3. License Grant. Subject to the terms and conditions of this License, Licensor hereby grants You a worldwide, royalty-free, non-exclusive, perpetual (for the duration of the applicable copyright) license to exercise the rights in the Work as stated below:

   (a) to reproduce the Work, to incorporate the Work into one or more Collective Works, and to reproduce the Work as incorporated in the Collective Works;

   (b) to create and reproduce Derivative Works provided that any such Derivative Work, including any translation in any medium, takes reasonable steps to clearly label, demarcate or otherwise identify that changes were made to the original Work. For example, a translation could be marked "The original work was translated from English to Spanish," or a modification could indicate "The original work has been modified.";

   (c) to distribute copies or phonorecords of, display publicly, perform publicly, and perform publicly by means of a digital audio transmission the Work including as incorporated in Collective Works;

   (d) to distribute copies or phonorecords of, display publicly, perform publicly, and perform publicly by means of a digital audio transmission Derivative Works;

   The above rights may be exercised in all media and formats whether now known or hereafter devised. The above rights include the right to make such modifications as are technically necessary to exercise the rights in other media and formats. All rights not expressly granted by Licensor are hereby reserved, including but not limited to the rights set forth in Sections 4(e) and 4(f).

4. Restrictions. The license granted in Section 3 above is expressly made subject to and limited by the following restrictions:

   (a) You may distribute, publicly display, publicly perform, or publicly digitally perform the Work only under the terms of this License, and You must include a copy of, or the Uniform Resource Identifier for, this License with every copy or phonorecord of the Work You distribute, publicly display, publicly perform, or publicly digitally perform. You may not offer or impose any terms on the Work that restrict the terms of this License or the ability of a recipient of the Work to exercise the rights granted to that recipient under the terms of the License. You may not sublicense the Work. You must keep intact all notices that refer to this License and to the disclaimer of warranties. When You distribute, publicly display, publicly perform, or publicly digitally perform the Work, You may not impose any technological measures on the Work that restrict the ability of a recipient of the Work from You to exercise the rights granted to that recipient under the terms of the License. This Section 4(a) applies to the Work as incorporated in a Collective Work, but this does not require the Collective Work apart from the Work itself to be made subject to the terms of this License. If You create a Collective Work, upon notice from any Licensor You must, to the extent practicable, remove from the Collective Work any credit as required by Section 4(d), as requested. If You create a Derivative Work, upon notice from any Licensor You must, to the extent practicable, remove from the Derivative Work any credit as required by Section 4(d), as requested.

(b) You may distribute, publicly display, publicly perform, or publicly digitally perform a Derivative Work only under: (i) the terms of this License; (ii) a later version of this License with the same License Elements as this License; or, (iii) either the unported Creative Commons license or a Creative Commons license for another jurisdiction (either this or a later license version) that contains the same License Elements as this License (e.g. Attribution-NonCommercial-ShareAlike 3.0 (Unported)) ("the Applicable License"). You must include a copy of, or the Uniform Resource Identifier for, the Applicable License with every copy or phonorecord of each Derivative Work You distribute, publicly display, publicly perform, or publicly digitally perform. You may not offer or impose any terms on the Derivative Works that restrict the terms of the Applicable License or the ability of a recipient of the Work to exercise the rights granted to that recipient under the terms of the Applicable License. You must keep intact all notices that refer to the Applicable License and to the disclaimer of warranties. When You distribute, publicly display, publicly perform, or publicly digitally perform the Derivative Work, You may not impose any technological measures on the Derivative Work that restrict the ability of a recipient of the Derivative Work from You to exercise the rights granted to that recipient under the terms of the Applicable License. This Section 4(b) applies to the Derivative Work as incorporated in a Collective Work, but this does not require the Collective Work apart from the Derivative Work itself to be made subject to the terms of the Applicable License.

(c) You may not exercise any of the rights granted to You in Section 3 above in any manner that is primarily intended for or directed toward commercial advantage or private monetary compensation. The exchange of the Work for other copyrighted works by means of digital file-sharing or otherwise shall not be considered to be intended for or directed toward commercial advantage or private monetary compensation, provided there is no payment of any monetary compensation in connection with the exchange of copyrighted works.

(d) If You distribute, publicly display, publicly perform, or publicly digitally perform the Work (as defined in Section 1 above) or any Derivative Works (as defined in Section 1 above) or Collective Works (as defined in Section 1 above), You must, unless a request has been made pursuant to Section 4(a), keep intact all copyright notices for the Work and provide, reasonable to the medium or means You are utilizing: (i) the name of the Original Author (or pseudonym, if applicable) if supplied, and/or (ii) if the Original Author and/or Licensor designate another party or parties (e.g. a sponsor institute, publishing entity, journal) for attribution ("Attribution Parties") in Licensor's copyright notice, terms of service or by other reasonable means, the name of such party or parties; the title of the Work if supplied; to the extent reasonably practicable, the Uniform Resource Identifier, if any, that Licensor specifies to be associated with the Work, unless such URI does not refer to the copyright notice or licensing information for the Work; and, consistent with Section 3(b) in the case of a Derivative Work, a credit identifying the use of the Work in the Derivative Work (e.g., "French translation of the Work by Original Author," or "Screenplay based on original Work by Original Author"). The credit required by this Section 4(d) may be implemented in any reasonable manner; provided, however, that in the case of a Derivative Work or Collective Work, at a minimum such credit will appear, if a credit for all contributing authors of the Derivative Work or Collective Work appears, then as part of these credits and in a manner at least as prominent as the credits for the other contributing authors. For the avoidance of doubt, You may only use the credit required by this Section for the purpose of attribution in the manner set out above and, by exercising Your rights under this License, You may not implicitly or explicitly assert or imply any connection with, sponsorship or endorsement by the Original Author, Licensor and/or Attribution Parties, as appropriate, of You or Your use of the Work, without the separate, express prior written permission of the Original Author, Licensor and/or Attribution Parties.

(e) For the avoidance of doubt, where the Work is a musical composition:

i. Performance Royalties Under Blanket Licenses. Licensor reserves the exclusive right to collect whether individually or, in the event that Licensor is a member of a performance rights society (e.g. ASCAP, BMI, SESAC), via that society, royalties for the public performance or public digital performance (e.g. webcast) of the Work if that performance is primarily intended for or directed toward commercial advantage or private monetary compensation.

ii. Mechanical Rights and Statutory Royalties. Licensor reserves the exclusive right to collect, whether individually or via a music rights agency or designated agent (e.g. Harry Fox Agency), royalties for any phonorecord You create from the Work ("cover version") and distribute, subject to the compulsory license created by 17 USC Section 115 of the US Copyright Act (or the equivalent in other jurisdictions), if Your distribution of such cover version is primarily intended for or directed toward commercial advantage or private monetary compensation.

(f) Webcasting Rights and Statutory Royalties. For the avoidance of doubt, where the Work is a sound recording, Licensor reserves the exclusive right to collect, whether individually or via a performance-rights society (e.g. SoundExchange), royalties for the public digital performance (e.g. webcast) of the Work, subject to the compulsory license created by 17 USC Section 114 of the US Copyright Act (or the equivalent in other jurisdictions), if Your public digital performance is primarily intended for or directed toward commercial advantage or private monetary compensation.

5. Representations, Warranties and Disclaimer
   UNLESS OTHERWISE MUTUALLY AGREED TO BY THE PARTIES IN WRITING, LICENSOR OFFERS THE WORK AS-IS AND ONLY TO THE EXTENT OF ANY RIGHTS HELD IN THE LICENSED WORK BY THE LICENSOR. THE LICENSOR MAKES NO REPRESENTATIONS OR WARRANTIES OF ANY KIND CONCERNING THE WORK, EXPRESS, IMPLIED, STATUTORY OR OTHERWISE, INCLUDING, WITHOUT LIMITATION, WARRANTIES OF TITLE, MARKETABILITY, MERCHANTIBILITY, FITNESS FOR A PARTICULAR PURPOSE, NONINFRINGEMENT, OR THE ABSENCE OF LATENT OR OTHER DEFECTS, ACCURACY, OR THE PRESENCE OF ABSENCE OF ERRORS, WHETHER OR NOT DISCOVERABLE. SOME JURISDICTIONS DO NOT ALLOW THE EXCLUSION OF IMPLIED WARRANTIES, SO SUCH EXCLUSION MAY NOT APPLY TO YOU.

6. Limitation on Liability.
   EXCEPT TO THE EXTENT REQUIRED BY APPLICABLE LAW, IN NO EVENT WILL LICENSOR BE LIABLE TO YOU ON ANY LEGAL THEORY FOR ANY SPECIAL, INCIDENTAL, CONSEQUENTIAL, PUNITIVE OR EXEMPLARY DAMAGES ARISING OUT OF THIS LICENSE OR THE USE OF THE WORK, EVEN IF LICENSOR HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES.

7. Termination

   (a) This License and the rights granted hereunder will terminate automatically upon any breach by You of the terms of this License. Individuals or entities who have received Derivative Works (as defined in Section 1 above) or Collective Works (as defined in Section 1 above) from You under this License, however, will not have their licenses terminated provided such individuals or entities remain in full compliance with those licenses. Sections 1, 2, 5, 6, 7, and 8 will survive any termination of this License.

   (b) Subject to the above terms and conditions, the license granted here is perpetual (for the duration of the applicable copyright in the Work). Notwithstanding the above, Licensor reserves the right to release the Work under different license terms or to stop distributing the Work at any time; provided, however that any such election will not serve to withdraw this License (or any other license that has been, or is required to be, granted under the terms of this License), and this License will continue in full force and effect unless terminated as stated above.

8. Miscellaneous

   (a) Each time You distribute or publicly digitally perform the Work (as defined in Section 1 above) or a Collective Work (as defined in Section 1 above), the Licensor offers to the recipient a license to the Work on the same terms and conditions as the license granted to You under this License.

(b) Each time You distribute or publicly digitally perform a Derivative Work, Licensor offers to the recipient a license to the original Work on the same terms and conditions as the license granted to You under this License.

(c) If any provision of this License is invalid or unenforceable under applicable law, it shall not affect the validity or enforceability of the remainder of the terms of this License, and without further action by the parties to this agreement, such provision shall be reformed to the minimum extent necessary to make such provision valid and enforceable.

(d) No term or provision of this License shall be deemed waived and no breach consented to unless such waiver or consent shall be in writing and signed by the party to be charged with such waiver or consent.

(e) This License constitutes the entire agreement between the parties with respect to the Work licensed here. There are no understandings, agreements or representations with respect to the Work not specified here. Licensor shall not be bound by any additional provisions that may appear in any communication from You. This License may not be modified without the mutual written agreement of the Licensor and You.

Appendix C

Music collection

Due to space constraints only the albums are listed in this appendix. A complete list of songs can be found in the accompanying database. All of the songs were in a folder structure: <artist>/<album>/<song>. This list was generated by parsing the <artist>/<album>-structure, and cleaning the list manually.

A

- ACDC
  - ACDC - High Voltage
  - Back In Black
  - Dirty Deeds Done Dirt Cheap
  - For Those About To Rock
  - High Voltage
  - Highway to Hell
  - If You Want Blood You've Got It
  - Let There Be Rock
  - Powerage
- ATB
  - Addicted To Music
  - Dedicated
  - No Silence
- Aerosmith
  - Aerosmith 1973
  - Get Your Wings 1974
  - Toys In The Attic 1975
  - Rocks 1976
  - Draw the Line 1978
  - Night In The Ruts 1980
  - Rock In A Hard Place 1982
  - Done With Mirrors 1985
  - Permanent Vacation 1987
  - Pump 1989
  - Get A Grip 1993
  - Nine Lives 1998
  - Just Push Play 2001
- Al Di Meola
  - 1976 - Elegant Gypsy
  - 1976 - Land Of The Midnight Sun
  - 1977 - Casino
  - 1979 - Splendido Hotel
  - 1981 - Electric Rendevous
  - 1983 - Scenario
  - 1987 - Tirami Su
  - 1988 - Kiss My Axe
  - 1990 - World Sinfonia
  - 1993 - World Sinfonia - Heart Of Immigrant
  - 1994 - Orange And Blue
- Al Stewart
  - Year of the Cat (remastered)
- Alanis Morissette
  - Jagged Little Pill
  - Jagged Little Pill Acoustic
  - Supposed Former Infatuation Junkie
- Allan Edwall

- Den lilla bäcken
- Edwalls Blandning 1979-84
- Mina Visor 1 - Färdknäpp
- Mina Visor 1 - Grovdoppa
- Mina Visor 3 - Aftonro
- Mina Visor 2 - Gnällspik
- Mina Visor 2 - Vetahuteri
- Ramsor om Dom och Oss

- Alphaville
  - First Harvest 1984-1992
- Amalia
  - Com Que Voz
- Amy Winehouse
  - Back To Black
  - Frank
- Annie Lennox
  - Bare
- Apoptygma Berzerk
  - 7
  - APBL2000 (Live 2000 Version)
  - Black
  - Harmonizer
  - Kathy's Song (6-Track Maxi-Single)
  - The Apopcalyptic Manifesto
  - Welcome To Earth
  - You And Me Against The World
- Arctic Monkeys
  - Favourite Worst Nightmare
- Astor Piazolla
  - Armaguedon (1977)
  - In Concert (1983)
  - La Camorra (1989)
  - Libertango (1981)
  - Tanguedia de Amor (1984)
  - The Rough Dancer And The Cyclical Night (1989)
  - Astor Piazzola & Gary Burton 'The New Tango' (1986)
  - Tango Sensations (1994)
  - Astor Piazalla & David Tonon-baum - El Porteno (1994)
  - Tango - Zero Hour (1986)
  - Eight Seasons (1996)
  - Musicues De Films (Tango, Henri IV) (1986)
  - Gidon Kremer - Hommage a Piazzolla (1996)
  - The Lausanne Concert (1989)
  - El Nuevo Tango de Buenos Aires (1989)
  - El Tango (1997)
  - Bandoneon Sinfonico (1990)
  - Maria De Buenos Aires Tango Operita CD1 (1998)
  - Ballet Tango (1992)
  - Maria De Buenos Aires Tango Operita CD2 (1998)
  - Concierto De Nacar (1997)
  - Tres tangos - concierto (1999)
  - Live At The 'Bouffes Du Nord' (1998)

B

- BT
  - Emotional Technology
- Beatles
  - Let It Be... Naked
  - Revolver & Magical Mystery Tour
  - White Album (disc 1)
  - White Album (disc 2)
- Beethoven
  - Violin Sonatas
- Ben Webster
  - Jazz Ballads CD2
  - Jazz Ballads CD1
- Billie Holiday
  - 1949-52 Radio & TV Broadcasts
  - 1953-56 Radio and TV Broadcasts
  - A Fine Romance
  - Anthology 1944-1959 Disc 1
  - Anthology 1944-1959 Disc 2
  - At Monterey

- Billie Holiday as Stratford '57
- Billie Holiday at Storyville
- Billie's Best
- Billie's Blues
- Broadcaast Performances Volume 4
- Broadcast Performances Volume 3 (1956 - 58)
- Compact Jazz: Billie Holiday
- Control Booth Series, Vol. 1 1940-1941
- Control Booth Series, Vol. 2
- Greatest Hits
- I Loves You Porgy
- Jazz at the Philharmonic
- Lady Day & Prez 1937-1941
- Lady Day - The Best Of Billie Holiday (Disc 1)
- Lady Day - The Best Of Billie Holiday (Disc 2)
- Lady Day - The Storyville Concerts (Disc 2 of 2)
- Lady Day - The Storyville Concerts (Disk 1 of 2)
- Lady Day: The Complete Billie Holiday on Columbia (1933-1944) - Disc 1 through 10
- Lady Live!
- Lady Sings the Blues
- Lady in Satin
- My Man
- New Orleans
- Remixed Hits
- Solitude
- Songs for Distingué Lovers
- Storyville - Masters of Jazz
- The Best of Billie Holiday
- The Commodore Master Takes
- The Complete 1951 Storyville Club Sessions
- The Complete Billie Holiday On Verve, 1945-1959
- The Complete Billie Holiday on Verve 1945-1959 (1 through 10)
- The Complete Commodore Recordings (1 of 2)
- The Complete Commodore Recordings (2 of 2)
- The Complete Decca Recordings (1 of 2)
- The Complete Decca Recordings (2 of 2)
- The Complete Decca Recordings (US Release)
- The Complete Verve Studio Master Takes (Disc 1 through 6)
- The Quintessential Billie Holiday Volume 9 (1940-1942)
- The Quintessential Billie Holiday, Vol. 1 through 9
- The Sensitive Billie Holiday 1940-1949
- Vdisc Musical Contribution For Our Armed Forces Overseas
- Volume 10: 1940-1941

- Billy Idol
  - Greatest Hits
  - Rebel Yell (Expanded Edition)
- Björk
  - Greatest Hits
- Björk Gudmundsdóttir & Trió Gudmundar Ingólfssonar
  - Gling-Gló
- Black Eyed Peas
  - Elephunk
  - Monkey Business
- Bowie, David
  - Reality Bonus CD
- Brian Eno
  - Ambient 1
  - Another Day On Earth
  - Another Green World
  - Before and after Science
  - Nerve Net
- Brian Eno & David Byrne
  - My Life In The Bush Of Ghosts (Remastered)
  - My Life in the Bush of Ghosts
- Brian Eno & John Cale
  - Wrong Way Up
- Bryan Ferry
  - As Time Goes By

C

- Charlie Haden
  - Land of the Sun
- Craig David
  - Born To Do It
- Crash Test Dummies
  - God Shuffled His Feet
- Creedence Clearwater Revival
  - Chronicle- 24-Karat Gold Disc
  - The Concert

D

- DJ Bobo
  - Just For You
  - The Ultimate Megamix 99
- Dalbello
  - whomanfoursays
- Damien Rice
  - O
- Dane Cook
  - Retaliation
- David Bowie
  - David Bowie (1969)
  - Station To Station (1976)
  - Low (1977)
  - Space Oddity (1969)
  - Heroes (1977)
  - The Man Who Sold The World (1970)
  - Hunky Dory (1971)
  - Lodger (1979)
  - Scary Monsters (1980)
  - The Rise and Fall of Ziggy Stardust (1972)
  - Aladdin Sane (1973)
  - Let's Dance (1983)
  - Pin Ups (1973)
  - Tonight (1984)
  - Diamond Dogs (1974)
  - Never Let Me Down (1987)
  - David Live
  - Young Americans (1975)
  - 1.Outside
  - Singles
  - Stage
  - Heathen (Bonus Disc)
  - Heathen (Disc 1)
  - Reality
  - Reality (CD 1)
- David Byrne
  - David Byrne
  - Feelings
  - Grown Backwards
  - Look Into The Eyeball
  - Rei Momo
  - Uh-Oh
- Dido
  - Life For Rent
  - No Angel
- Don McLean
  - American Pie
- Doors
  - Waiting For The Sun
- Dr Hook & The Medicine Show
  - The Very Best Of
- Dune
  - Dune
  - Forever
- deLillos
  - Festen er ikke over... det er kake igjen Disc 1
  - Festen er ikke over... det er kake igjen Disc 2
  - Før Var Det Morsomt Med Sne
  - Ikke gå
  - Kast Alle Papirene
  - Kjerringvik-demoen del 1

- Kjerringvik-demoen del 2
- Mere Disc 1
- Mere Disc 2
- Midt i begynnelsen
- Suser Avgårde
- Varme Mennesker

- deLillos'85
  - Suser videre
  - evig forelsket da

E

- Echo And The Bunnymen
  - Heaven Up Here
- Eiffel 65
  - Europop
- Enya
  - A Day Without Rain
  - Amarantine
  - Paint The Sky With Stars
  - Shepherd Moons
  - The Celts
  - The Memory Of Trees
  - Watermark
- Erik Satie
  - Gymnopedies

F

- Faithless
  - Forever Faithless- The Greatest Hits
- Fragma
  - Embrace
  - Toca
- Franz Ferdinand
  - Franz Ferdinand
  - Franz Ferdinand (Special Edition)
  - Remixes
  - This Fffire
  - You Could Have It So Much Better

G

- Gary Numan
  - Documents
  - Replicas
  - Telekon
  - The Pleasure Principle
  - Tubeway Army
  - Warriors
- Gina G
  - Fresh
- Glenn Miller
  - The Glenn Miller Story
- Green Day
  - International Superhits!
- Grinderman
  - Grinderman
- Gwen Stefani
  - Love Angel Music Baby
  - The Sweet Escape
- Gåte
  - Jygri

H

- Hampton The Hampster
  - The Hampsterdance Song
- Hugh Cornwell
  - Beyond Elysian Fields
  - Footprints In The Desert
  - Guilty
  - Hi Fi

I

- Ian Dury
    - Reasons To Be Cheerful - The Best Of Ian Dury (Disc1)
    - Reasons To Be Cheerful - The Best Of Ian Dury (Disc2)
- Ibrahim Ferrer
    - Buena Vista Social Club Presents Ibrahim Ferrer
- Iggy Pop
    - Blah-Blah-Blah
    - Brick By Brick
    - Lust For Life
    - New Values
    - Pop Music
- Imperiet
    - Alltid Rött Alltid Rätt: en samling 1983-88
- Infernal
    - From Paris To Berlin
- iio
    - Poetica

J

- James Blunt
    - All The Lost Souls (Bonus Track)
    - Back To Bedlam (Edited)
- Jan Johansson
    - Folkvisor
    - Jazz på svenska
- Japan
    - Gentlemen Take Polaroids
    - Tin Drum
- Jerry Harrison
    - Casual Gods
- Jethro Tull
    - Aqualung
- John Cale
    - 5 tracks
    - Black Acetat
    - Fragments Of A Rainy Season
    - Hobosapiens
    - Island Years (Disc 1)
    - Island Years (Disc 2)
    - Walking On Locusts
- John Legend
    - Get Lifted
- Johnny Cash
    - American III - Solitary Man
    - American IV The Man Comes Around
    - American V: A Hundred Highways
    - Unchained
    - Unearthed Volume 4: My Mother's Hymn Book

K

- K.C. & the Sunshine Band
    - Shake Your Booty
- Kaizers Orchestra
    - Død manns tango
    - Evig Pint
    - Maestro
    - Mann Mot Mann [EP]
    - Ompa Til Du Dør
- Karin Krog
    - Where you at?
- Keith Jarret
    - 1977 - Byablue
    - 1980 - Sacred Hymns
    - 1980 - The Celestial Hawk
    - 1981 - Concerts
    - 1983 - Changes
    - 1983 - Standards Vol 1
    - 1983 - Standards Vol 2
    - 1987 - Changeless

- 1990 - Paris Concert
- 1991 - Bye Bye Blackbird
- 1991 - Vienna Concert
- 1999 - The Melody At Night With You

- Keith Jarrett
  - My Song
  - Personal Mountains
- Ketil Bjornstad and David Darling
  - The River
- Kevin Bloody Wilson
  - 20 Years Of Kev
  - Born again piss tank
  - Kalgoorlie Love Songs
  - Kev's back (The Return of the Yobbo)
  - Kev's Kristmas
  - Let's call him ... Kev!
  - Let Loose Live In The Outback
  - My Australian roots
  - The second kumin of Kev
  - The worst of Kevin Bloody Wilson
  - Youre average Australian yobbo
- Kevin Coyne
  - Marjory Razorblade
  - Pointing the Finger
  - Sign Of The Times
- Kraftwerk
  - Autobahn
  - The Man - Machine
- Kylie Minogue
  - Body Language

L

- Laila Dalseth
  - one of a kind - CD1
  - one of a kind - CD2
- Lars Winnerbäck
  - Daugava
- Led Zeppelin
  - Houses Of The Holy
  - Led Zeppelin (Remaster 1994)
  - Led Zeppelin II
  - Led Zeppelin IV
- Linkin Park
  - Dirt Off Your Shoulder-Lying From You- MTV Ultimate Mash-Ups Presents Collision Course (Parental Advisory)
  - Hybrid Theory (Bonus Tracks)
  - Live In Texas
  - Meteora (Bonus Tracks)
  - Minutes To Midnight (Parental Advisory)
  - Reanimation (Bonus Tracks)
- Lisa Ekdahl
  - En Samling Sånger
- Lloyd Cole
  - Antidepressant
  - Bad Vibes
  - Don't Get Weird On Me Baby
  - Love Story
  - Rattlesnakes
- Lomsk
  - Amerikabrevet
- Lou Reed
  - New York
  - Songs For Drella
  - The Very Best of Lou Reed

M

- Madison Avenue
  - The Polyester Embassy
- Madonna
  - American Life (Edited)
  - GHV2
  - Get Together (Maxi-Single)
  - Hung Up (DJ Version)
  - I'm Going To Tell You A Secret (Live)
  - Jump (6-Track Maxi-Single)
  - Like A Virgin
  - Like A Virgin (Remastered-Bonus Tracks)
  - Music [Import Box Set]
  - Ray Of Light
  - Sorry (CD Maxi-Single)
  - The Confessions Tour (Live) (Parental Advisory)
- Madrugada
  - Grit
  - Industrial Silence
  - The Deep End
- Magga Stína
  - syngur Megas
- Mari Boine
  - Gula Gula (1989)
  - Goaskinviellja - Eagle Brother (1993)
  - Leahkastin (Unfolding) (1994)
  - Eallin - Live (1996)
  - Balvvoslatjna (Room of Worship) (1998)
  - Winter In Moscow (2001)
  - Eight Seasons (2001)
- Mariah Carey
  - #1's
- Mark Lanegan
  - Bubblegum
  - Whiskey for the Holy Ghost
- Michael Andrews
  - Donnie Darko (Score)
- Michael Jackson
  - Essential Michael Jackson
  - HIStory- Past, Present And Future, Book I
  - Number Ones
- Mike Scott
  - Bring 'em all in
  - Still Burning
- Miles Davis
  - Porgy and Bess
  - So What
  - Tutu
- Modern Talking
  - The Final Album- The Ultimate Best Of Modern Talking
- Morrissey
  - Bona Drag
- Mr. President
  - Space Gate
- Muse (UK)
  - Black Holes And Revelations

N

- N-Trance
  - Happy Hour
- Natacha Atlas
  - Something dangerous
- Neil Young
  - Everybody's Rockin'
  - Freedom
  - Rust Never Sleeps
  - Sleeps With Angels
- Nena
  - 99 Luftballons
- New Order
  - Get Ready
  - International
  - Republic
  - Technique
  - Waiting For The Sirens' Call
- New York Philharmonic
  - Adagio For Strings-Violin Concerto-In Praise Of Shahn (Expanded Edition)
- Nick Cave & The Bad Seeds
  - Abattoir Blues
  - Acoustic Versions of Songs from "Tender Prey"
  - As I Sat Sadly By Her Side (CD Single)
  - B-Sides & Rarities - Volume I
  - B-Sides & Rarities - Volume II
  - B-Sides & Rarities - Volume III
  - Do You Love Me?
  - Henry's Dream
  - Here Comes the Sun
  - Kicking Against the Pricks
  - Let Love In
  - Love Letter
  - Murder Ballads
  - No More Shall We Part
  - Nocturama
  - Tender Prey
  - The Boatman's Call
  - The Good Son
  - The Lyre Of Orpheus
  - The Mercy Seat
  - The Ship Song
  - What A Wonderful World
  - Where The Wild Roses Grow
- No Doubt
  - No Doubt
  - Return Of Saturn
  - Rock Steady
  - The Beacon Street Collection
  - The singles 1992 - 2003
  - Tragic Kingdom

O

- Oscar Peterson
  - Jazz Ballads 8 - Disc 1
  - Jazz Ballads 8 - Disc 2
  - Night Train
- Oslo Kammerkor - Sondre Bratland - Berit Opheim
  - Dåm

P

- Patti Smith
  - Trampin'
- Paul Oakenfold
  - Bunkka
- Paul Simon
  - The Rhythm of the Saints
- Paul Van Dyk
  - Global
  - Reflections
- Paul Weller
  - Illumination
  - Stanley Road
  - Wild Wood

- Pepito Ross
    - Vamos A La Playa
- Perssons Pack
    - Diamanter
    - Diamanter (cd 2)
    - Kanoner och små, små saker
    - Kärlek och dynamit
    - Nyårsafton i New York
    - Sekunder i Sverige
    - Svenska hjärtan
    - Äkta Hjärtan
- Pet Shop Boys
    - Actually (Remastered)
    - Popart- The Hits
- Peter Gabriel
    - Hit (CD 1)
    - Hit (CD 2)
    - III
    - Up
- Pink Floyd
    - Dark Side of the Moon
    - The Wall (Disc 1)
    - The Wall (Disc 2)
    - Wish You Were Here
- Pixies
    - Surfer Rosa & Come On Pilgrim

Q

- Queen
    - A Day At The Races
    - A Night At The Opera
    - Jazz
    - News of the World

R

- R.E.M
    - In Time- The Best Of R.E.M. 1988-2003 Rarities And B-Sides
    - What's The Frequency, Kenneth-
- R.E.M.
    - In Time 1988-2003
    - Lifes Rich Pageant
- Raga Rockers
    - Forbudte Følelser
- Rage Against the Machine
    - Rage Against The Machine
- Ramones
    - Anthology (Disc 2)
    - Ramones - Anthology - Hey Ho Let's Go 1
- Rednex
    - Cotton Eye Joe (Sex & Violins)
- Robbie Williams
    - Escapology
- Robert Miles
    - Dreamland
- Roxy Music
    - Flesh + Blood
- Rufus Wainwright
    - Poses
    - Rufus Wainwright
    - Want One
    - Want two
- Rush
    - Rush 1974
    - Caress of Steel 1975
    - Fly By Night 1975
    - All The World's A Stage 1976
    - 2112 1976

- A Farewell To Kings 1977
- Hemispheres 1978
- Permanent Waves 1980
- Exit...Stage Left 1981
- Moving Pictures 1981
- Signals 1982
- Grace Under Pressure 1984
- Hold Your Fire
- Retrospective, Vol. 1 (1974-1980)

- Ry Cooder
  - Paris Texas

S

- Scooter
  - 24 Carat Gold
- Seu Jorge
  - The Life Aquatic Studio Sessions
- Sex Pistols
  - Never Mind The Bollocks
- Sharon Jones and the Dap-Kings
  - 100 Days, 100 Nights
- Slade
  - Nobody's fool
  - Slade in flame
  - Whatever Happened To Slade
- Smiths
  - Singles
- Some Like It Hot
  - Some Like It Hot
- Sophie Ellis Bextor
  - Murder On The Dancefloor
- Stefan Sundström
  - Sundström spelar Allan
- Stephen Lynch
  - A Little Bit Special
  - Superhero
  - The Craig Machine
- System of a Down
  - Toxicity

T

- Talking Heads
  - Naked
  - Remain In Light
- Teenage Fanclub
  - Bandwagonesque
  - Grand Prix
- Television
  - Adventure
  - Marquee Moon (2003 Remaster)
- The Aller Værste
  - Disniland I De Tusen Hjem
  - Materialtretthet
  - The Aller Værste
- The Clash
  - London Calling
- The Countdown Quartet
  - Hits Of The 80's
- The Cure
  - Disintegration
  - Galore (The Singles 1987-1997)
  - Kiss Me Kiss Me Kiss Me (Delux Edition - CD1)
  - Staring At The Sea The Singles
  - The Head On The Door
  - The Top
  - Wish
- The Doors
  - Strange Days
  - Waiting for the Sun
- The Dukes of Stratosphear
  - Chips from the Chocolate Fireball

- The Jam
  - All Mod Cons
  - Setting Sons
  - Sound Affects
  - The Gift
- The Killers
  - Hot Fuss
  - Sam's Town
- The Lime Spiders
  - Nine Miles High
- The Prodigy
  - Music For The Jilted Generation
  - Out Of Space
  - The Fat Of The Land (Parental Advisory)
- The Raconteurs
  - Broken Boy Soldiers
- The Real McCoy
  - Platinum & Gold Collection- The Best Of Real McCoy
- The Rolling Stones
  - A Bigger Bang
  - Bridges To Babylon
  - Emotional Rescue
  - Exile On Main Street
  - Forty Licks (CD ONE)
  - Forty Licks (disc 2)
  - Goats Head Soup
  - It's Only Rock 'N Roll
  - Some Girls
  - Steel Wheels
  - Sticky Fingers
  - Tattoo You
  - The Very Best 1962 - 1975
  - Under Cover
  - Voodoo Lounge
- The Stranglers
  - 10
  - Aural Sculpture
  - Dreamtime
  - Feline - Extended Edition (incl. 6 Bonus Tracks)
  - La Folie (Remastered)
  - No More Heroes [Bonus Tracks]
  - Rattus Norvegicus
  - Sweet Smell Of Success Best Of The Epic Years
  - Sweet Smell of Success
  - The Best of the Epic Years
  - The Raven
- The Streets
  - A Grand Don't Come For Free
  - Original Pirate Material
  - The Hardest Way To Make An Easy Living
- The Style Council
  - Our Favourite Shop
  - The Sound Of
- The The
  - 45 RPM
- The Triffids
  - Born Sandy Devotional
  - Born Sandy Devotional (original)
  - Calenture
  - Calenture (bonus disc)
  - Calenture (original)
  - In The Pines (original)
  - In The Pines [2007 Remastered & Expanded]
  - The Black Swan
- The Waterboys
  - Book Of Lightning
  - Dream Harder
  - Fisherman's Blues
  - Room To Roam
  - This Is The Sea [1 of 2]
  - This Is The Sea: Additional Recordings [2 of 2]
  - Universal Hall
- Tiesto
  - Elements Of Life
  - Parade Of The Athletes

- Tom Robinson Band
  - Power in the Darkness
  - TRB TWO
- Tom Waits
  - Bounced Checks
  - One From The Heart OST (Tom Waits and Crystal Gayle)
  - Swordfishtrombones 1983
  - Rain Dogs 1985
  - Franks Wild Years 1987
  - Bone Machine 1992
  - Night On Earth (Soundtrack) 1992
  - The Black Rider 1993
  - Mule Variations 1999
  - Alice (2002)
  - Blood Money (2002)
  - Big Time 1988 (Part2)
  - Orphans: Bastards (d3)
  - Orphans: Bawlers (d2)
  - Orphans: Brawlers (d1)
  - Tales From The Underground (1994) Volume 1 through 6
  - Tom Waits (Gavin Bryars with Tom Waits) - (1993) Jesus' Blood Ne
  - Tom Waits - 'Alice' (The Original Demos)
  - Tom Waits - (1977) Everytime I Hear T., Live in Hamburg, Germany
  - Tom Waits - (1977) Invitation To The Blues, April 26, Germany
  - Tom Waits - (1979) Cold Beer On A Hot Night
  - Tom Waits - (1979) Cold Beer and Warm Women (live Sydney)
  - Tom Waits - (1979) Fast Women & Slow Horses (live) (128)
  - Tom Waits - (1979) On Broadway
  - Tom Waits - (1982) Shadow of intolerance, Ontario
  - Tom Waits - (1998) Dead Man Walking, March 29, Los Angeles
  - Tom Waits - (1999) Hold On (EP)
  - Tom Waits - (1999) VH1 Storytellers
  - Tom Waits - (2000) May 26, Warsaw, Sala Kongresawa
  - Tom Waits - (2000) May 30, Paris, Le Grans Rex
  - Tom Waits - (2004) Amsterdamned, November 21, Amsterdam
  - Tom Waits - Miscellaneous
  - Tom Waits Live
  - Tom Waits Not - Released Studio Records
- Toni Braxton
  - Ultimate Toni Braxton
- Touch & Go
  - I Find You Very Attractive
- Toy Dolls
  - We're Mad (the anthology) - disc 2
  - we're mad! the anthology (Disc.1)
- Travis
  - 12 Memories
  - Good Feeling
  - The Boy With No Name
  - The Invisible Band (Bonus Tracks)
  - The Man Who
- Two to Tango
  - Two To Tango (1)
  - Two To Tango (2)

U

- U2
    - 1980 Boy
    - 1981 October
    - 1983 Under A Blood Red Sky
    - 1983 War
    - 1984 The Unforgettable Fire
    - 1985 Wide Awake In America
    - 1987 The Joshua Tree
    - 1988 Rattle And Hum
    - 1991 Achtung Baby
    - 1993 Zooropa
    - 1997 Pop
    - 1998 The Best Of 1980-1990 & B-sides Disc 1
    - 1998 The Best Of 1980-1990 & B-sides Disc 2
    - 2000 All That You Can't Leave Behind
    - 2002 The Best And The B-Sides Of 1990-2000 CD1
    - 2002 The Best And The B-Sides Of 1990-2000 CD2
    - Achtung Baby
    - How To Dismantle An Atomic Bomb
    - The Best Of 1980 - 1990
    - The Best Of 1990-2000 (A Sides US Version)
    - Who's Gonna Ride Your Wild Horses
- U96
    - Heaven
    - Replugged
- Ultravox
    - 77 Ha-ha-ha
    - 77 Ultravox
- Perssons Pack
    - Nyårsafton i New York

V

- VNV Nation
    - Empires
    - Genesis.2
    - Matter + Form
- Vamp
    - 13 humler
    - En annen sol
    - Flua på veggen
    - Godmorgen, søster
    - Horisonter
    - Månemannen
    - Vamp i full symfoni med kringkastingsorkesteret
    - siste stikk
- Various
    - Buffy The Vampire Slayer - The Album
    - Donnie Darko (Original Soundtrack)
    - I'm Your Fan (Tribute to Leonard Cohen)
    - McMusic 21
    - Suuret suomalaiset tangot : Satumaa
    - Taube 1
    - Taube 2
    - Until the End of the World
    - Viva Cuba!
    - Done Again (In The Style Of The Beatles)- The Beatles, Vol.8
- Vladimir Vissotsky
    - 1
    - 2
- Vladimir Vissotsky & Marina Vlady
    - Vlady Vissotsky
- Vømmøl Spellmannslag
    - Vømlingen
    - Vømmølmusikken