JOHANNES KEPLER
UNIVERSITY LINZ

Submitted by
Reinhard Sonnleitner

Submitted at
Department of Computational Perception

Supervisor and
First Examiner
Gerhard Widmer

Second Examiner
Meinard Müller

April, 2017

Audio Identification via Fingerprinting

Achieving Robustness to Severe Signal Modifications

Doctoral Thesis
to obtain the academic degree of
Doktor der technischen Wissenschaften
in the Doctoral Program
Technische Wissenschaften

JOHANNES KEPLER
UNIVERSITY LINZ
Altenbergerstraße 69
4040 Linz, Österreich
www.jku.at
DVR 0093696

STATUTORY DECLARATION

I hereby declare that the thesis submitted is my own unaided work, that I have not used other than the sources indicated, and that all direct and indirect sources are acknowledged as references. This printed thesis is identical with the electronic version submitted.

Linz, April 2017

Reinhard Sonnleitner

ABSTRACT

In this thesis we approach the task of audio identification via audio fingerprinting, with the special emphasis on the complex task of designing a system that is highly robust to various signal modifications. We build a system that can account for linear and non-linear time-stretching, pitch-shifting and speed changes of query audio excerpts, as well as for severe noise distortions. We motivate the design of yet another fingerprinting method to complement the rich number of proposed methods and research in this field.

In this thesis we propose a novel, efficient, highly accurate and precise fingerprinting method that works on geometric hashes of local maxima of the spectrogram representation of audio signals. We propose to perform the matching of features using efficient range-search, and to subsequently integrate a verification stage for match hypotheses to maintain high precision and specificity on challenging datasets. We gradually refine this method from its early concept to a practically applicable system that is evaluated on queries against a database of 430,000 tracks, with a total duration of 3.37 years of audio content.

Our proposed method is the first in the academic literature that is shown to be able to cope with severe signal modifications while being applicable to large reference collections. This claim is supported via rich evaluation on manually crafted data that is modified in the range of $\pm 30\%$ in speed, time-stretching and pitch scale modifications. We further evaluate the system on noise-distorted queries, and show the influence of various parameters on the resulting identification performance and processing run times.

We identify the task of DJ mix monitoring to be one of the most challenging application areas for audio fingerprinting, due to the vast amount of signal modifications that can be introduced by performers. We observe that the identification performance of systems can suffer tremendously when applied to DJ mixes, much more so than on manually crafted evaluation datasets, since it is hard to create test cases that cover the variety of modifications that can be encountered in DJ-mixes. To close this gap in evaluation methodology, we manually compile and annotate a free dataset of DJ mixes to support the research community in investigating and evaluating particular strengths and weaknesses of proposed systems. In this thesis we make use of this dataset for extensive evaluation of our method. Finally, we show the possibility of building a sequence detection program on top of the fingerprinter, to enable the monitoring of long query recordings for either interactive analysis or fully automated result reporting.

ZUSAMMENFASSUNG

Diese Dissertation befasst sich mit dem Thema Audioidentifikation mittels Audio-Fingerprinting. Im Zuge dieser Arbeit wird der Stand der Forschung durch ein effizientes- und höchst präzises System mit starker Robustheit gegenüber Signalmodifikationen erweitert. Dieses System ist unter anderem robust gegenüber linearen und nicht-linearen Modifikationen der Zeit- und Frequenzdimension von Audiosignalen.

Die Funktionsweise basiert auf skalierungsinvarianten Darstellungen lokaler Maxima im Audiospektrogramm, die mittels Bereichssuche in einem vierdimensionalen Merkmalsraum verglichen werden. Ermittelte Übereinstimmungen werden einer Verifikationsmethode unterzogen, die es ermöglicht Abfragen höchst präzise und spezifisch zu beantworten. Durch schrittweise Verfeinerung wird ein praktisch anwendbares System realisiert, dessen Eigenschaften auf einer Referenzsammlung von 430.000 Audiotracks mit einer Spieldauer von insgesamt etwa $3,37$ Jahren untersucht und evaluiert werden.

Diese Arbeit beschreibt das erste System in der wissenschaftlichen Literatur, das robust gegenüber starken Signalmodifikationen ist und gleichzeitig effizientes Suchen in großen Datensammlungen ermöglicht. Das System wird anhand von Experimenten mit manuell erstellten Testfällen ausgewertet, in denen die Zeit- und Frequenzdimension im Audiosignal Modifikationen von bis zu $\pm 30\%$ unterliegt. Weiters werden Parameterstudien und Laufzeitmessungen präsentiert.

Eine der größten Herausforderungen für automatisiertes Audio-Fingerprinting ist die Track-Identifikation in DJ-Mixes, da ein DJ das Audiosignal nahezu uneingeschränkt modifizieren kann. Die Genauigkeit von Fingerprinting-Systemen kann auf DJ-Mixes weitaus geringer ausfallen als auf manuell modifizierten Testsignalen. Der Grund hierfür liegt in der Schwierigkeit, die Menge der in DJ-Mixes potentiell auftretenden Signalmodifikationen durch manuell erzeugte Testfälle abzudecken. Um diese Diskrepanz zu überbrücken erstellen und veröffentlichen wir einen für Forschungszwecke frei verfügbaren DJ-Mix-Datensatz samt Annotationen, und werten das vorgestellte System mit zahlreichen Experimenten auf diesem Datensatz aus. Weiters wird ein Sequenzdetektor für Media-Monitoring beschrieben, der als Erweiterung des Fingerprinters dazu dient, Abfragen von langer Spieldauer zu analysieren und zu segmentieren.

ACKNOWLEDGMENTS

Thank you, Lisa, for your support and patience.
Many thanks to my family and friends for all their support.

I want to express my sincere gratitude to my supervisor and employer, Prof. Gerhard Widmer, who gave me the opportunity to work at this department, where I could pursue my research interests within a group of profoundly knowledgeable, helpful and friendly colleagues. At the department I enjoyed more than six years in a highly productive and familiar environment, where I could study diverse fields of my own interest. I want to thank Gerhard for his ongoing encouragement and supervision, and for always happily providing insight, advice and immense expertise.

I want to thank Meinard Müller for his efforts spent on reviewing my thesis. This really means a lot to me.

I also want to thank all my colleagues (which are in fact good friends) for the endless and fruitful discussions and their distinguished humour. You all are the reason that working here does not feel like work, but rather like meeting friends to together solve challenging problems and keep each other updated on a vast variety of interesting research topics.

For the work presented in this thesis, I make use of exactly *two* routines that are created by colleagues rather than by myself. For the first of these, I want to thank Jan Schlüter for supplying his Python-script that wraps ffmpeg decoding to RAM. For the second, I want to thank Rainer Kelz for the high-performing peak-extractor that is used from Chapter 4, onwards. Rainer took the time to translate my Python STFT routine together with my original max-min filter based peak-extractor into a C-extension. On top of that, Rainer extended it to guarantee at most one peak per window, perform parabolic interpolation of peak coordinates, and most time-consuming, he carefully refined the memory access patterns for quicker run times in the presence of multi-threading.

The research presented in this thesis was supported by the Austrian Science Fund FWF under projects TRP307 and Z159 (Wittgenstein Award), and by the European Research Council (ERC Grant Agreement 670035, project CON ESPRESSIONE).

CONTENTS

1 INTRODUCTION

This thesis presents the results of our research on automated audio identification via fingerprinting, and reports detailed experiments, descriptions and evaluations. The work describes our initial attempts to solve open challenges, which we subsequently refine to realize a practical, efficient system that can meet robustness demands that are imposed by various application scenarios.

1.1 MOTIVATION

Audio is an omnipresent type of media. Popular audio streaming services host millions of audio files, and at every point in time thousands of broadcast stations transmit audio content. The ever increasing volume of audio data, whether online or on personal devices, creates huge interest in the ability to identify audio content. This can be done via "audio fingerprinting", which is an identification technology that aims at operating at highest precision and specificity.

The industry uses audio fingerprinting systems to monitor radio and TV broadcast channels, in order to create detailed lists of the specific content that was played at any given time. In addition to radio and TV broadcast monitoring, performance rights organizations show interest in monitoring music events, for example in discotheques. Without using automated fingerprinting systems, royalty collection depends on the broadcasters who are expected to create detailed lists of played content.

Fingerprinting systems used for media monitoring usually analyse audio streams of a large number of broadcast channels or recordings from various events. As these systems operate on huge amounts of data, the involved data structures should be as compact as possible while the systems must efficiently operate on large, and ever-growing reference databases. In addition, the application to media monitoring dictates strong robustness demands. While for this use case the sensitivity to noise may not be the main concern, the systems need to recognize audio material that might have been modified by various effects.

In this thesis, we address this open research problem[1] and propose a viable and practically applicable solution.

1 See Section 1.2, where we support this claim.

1.2 CHALLENGES

Audio identification systems usually operate on huge amounts of data, and are expected to meet several robustness requirements depending on the actual use case.

The minimum requirement would seem to be robustness to various types of lossy audio compression and a certain degree of noise.

Systems that aim to identify microphone recordings of short audio pieces require strong robustness to background noise, as for example speech and natural sounds, or even different songs played at the same time that are audible in the near vicinity.

The identification capabilities of fingerprinting systems are commonly used for detection tasks, as for example the detection of song duplicates in private collections, or large scale copy or plagiarism detection systems that aim to report copyright infringements. For copy detection, the National Institute of Standards and Technology (NIST) aims to test systems with heavily modified audio/video content. The TRECVID 2010 copy detection dataset (Smeaton, Over, and Kraaij, 2006) includes several severely distorted files: according to an investigation mentioned by Ouali, Dumouchel, and Gupta (2014), this dataset contains files that are altered in speed from -180% up to $+23\%$.

The most challenging application area seems to be the monitoring of DJ sets and DJ (live) performances. This is due to the large degree of freedom to introduce signal modifications. DJ mixes incorporate a wide variety of signal modifications, e.g. pitch shifting, tempo modifications, loops, cross-fading and beat-matching, to name a few. These modifications can occur in combination and over arbitrary time-spans. Each individual modification is expected to be more severe than what is usually encountered (and tested for) in the monitoring of audio content in radio and TV broadcasts.

Due to the individual artistic style of performers it is hard to quantify the type and severity of signal manipulations that can possibly be encountered. For the same reason we believe it is hard to manually create representative test cases that reflect the possible modifications for system evaluation.

In addition to robustness demands, we have to consider the use-case-dependent severity, or impact, of incorrect results, and have to take into account the required identification performance characteristics of fingerprinting systems.

For example, if a fingerprinting system is used as duplicate song detector and it misses a match, the user might waste a few megabytes of hard disk storage space since it will not get removed. However, if it reports a false match, this could cause a user to delete a unique song. Likewise, copy detection systems that miss occurrences of copyright infringement are not effective, however it is worse if such systems report false positives, i.e. claim a song to be protected even it is not,

because this could lead to expensive lawsuits. Most importantly, for large scale media monitoring, false positives are costly – revenue might be assigned to the wrong artist. Another type of mistake, false negatives, can lead to hours of unidentified content that will have to be investigated with manual effort. Either type of error will increase the maintenance cost of a system.

Track identification in DJ mixes presents an exceptionally hard challenge for automated systems, as it demands high robustness to various signal modifications while at the same time requiring the system to maintain a high level of specificity to avoid false revenue assignment.

We became involved in the topic of audio fingerprinting when we were asked to perform a feasibility study. One goal of this study was to quantify, in an automated fashion, the extent of time and pitch scale modifications [2] that were introduced by DJs in their performances. Using manual techniques, we could show for parts of the performances that these types of modifications frequently occur in combination, but at that time (the year 2013) we could not find a reliable way to show the extent of these modifications in an automated and efficient fashion. In fact, we could not even answer the question whether some of the tracks within the performances were actually present in our reference database. The dilemma was that to answer this question, one would need a robust and reliable fingerprinting system to identify the content.

When designing a system for the monitoring of DJ performances it is advantageous to first establish lower and upper bounds of the extents of scale modifications that are likely to be encountered. Investigating our dataset of recorded DJ performances, via manual spectrogram comparisons we found indications of scale modifications in the range of roughly $\pm 5\%$. This estimate will serve as the required *lower bound* of scale robustness throughout this thesis, i.e. a system should be robust to *at least* this range of scale modifications, but may be required to cope with even more severe alterations. For an estimate of a reasonable *upper bound* – a range where larger extents of modifications are introduced very rarely – one would need a reliable automated approach to quantify encountered scale modifications on a large, representative dataset. Alternatively, one could design a questionnaire for DJs to establish an upper bound. Currently, we do not know of an upper bound and have to leave this question open. In Chapter 8 we propose an automated method to reliably analyse scale modifications in long recordings, which seems to be a promising way to establish an upper bound of scale modifications in the light of a representative dataset. However, this is only valid if the robustness of our method exceeds this hypothetical upper bound.

2 We will clarify our terminology for scale modifications (e.g. speed, tempo, pitch) in Section 1.2.1, below.

1.2.1 *Terminology on Robustness Criteria*

Some important terms on robustness criteria will be used extensively throughout this thesis. Here, we clarify our terminology and will define abbreviations, or short-cut notations, that we will then use to refer to these terms.

For a robust identification of audio signals we have to consider modifications in the time dimension as well as the frequency dimension. Some effects will affect exactly one of these two dimensions, others affect both at the same time. When we talk about the application of an effect, we will use the words modification, change and alteration synonymously.

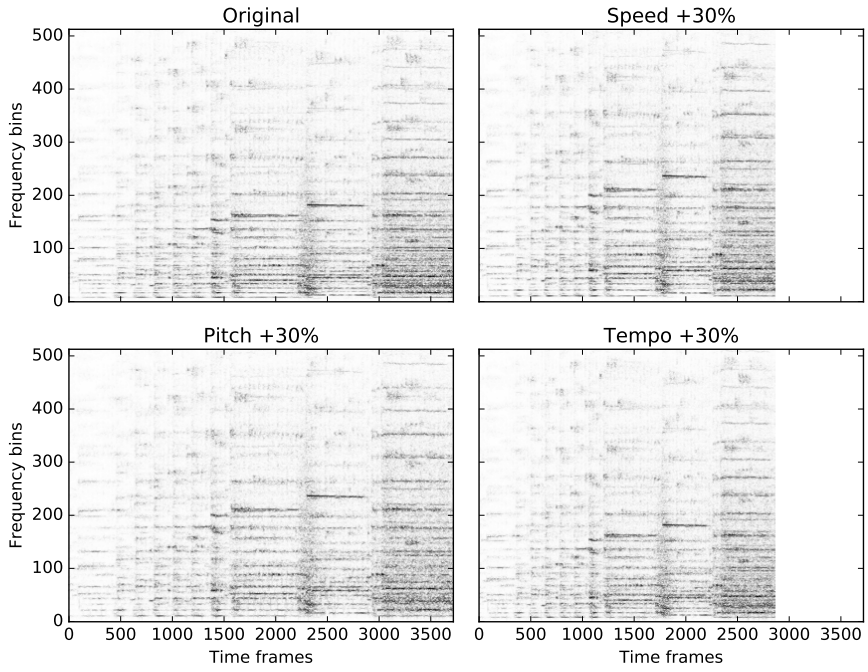


Figure 1.1: The three basic types of time and frequency scale modifications, compared to the original (top left).

Changing the time dimension only is commonly known as a *"tempo"* change: here, the audio is sped up or slowed down without observable changes in pitch. We refer to the tempo-effect as T-modifications.

Vice versa, if only the frequency dimension is modified, without alterations of the time dimension, we call this pitch-effect P-modifications.

If a song is played faster, resulting in a higher pitch, this is called a change in *speed*. Compared to T- and P-modifications, speedup or slowdown is the simplest effect, which can be achieved for example by simply changing the rotational speed of the turntable. Speeding up, or slowing down a recording *proportionally* affects the time (T) and the frequency (or pitch) scale (P). Throughout this document we

will refer to this effect as a modification or change in *TP*. Note that a system which is robust to *TP*-modifications, is not necessarily robust to *P*-modifications or *T*-modifications.

In Figure 1.1 we visualize the effect of audio that is played at higher pitch and changed in tempo and speed.

1.2.2 *The Open Research Problem*

For media monitoring, fingerprinting systems must be robust to scale modifications. For example, Cano, Batlle, Mayer, et al. (2002) observe that broadcasters may speed up audio. Likewise, the broadcasters sometimes apply *P*-modification effects (Fenet, Richard, and Grenier, 2011). From initial experiments on DJ performances we learned that in addition to that, DJs frequently apply *T*-modifications (see Chapter 8). (Malekesmaeili and Ward, 2014) note the need for *T*-modification robustness for copy-detection systems.

From this we conclude that in order to build *reliable* and applicable audio fingerprinting systems, especially for the use in media monitoring and copy detection, at least the following requirements must be met:

- Robustness to background noise and compression
- Robustness to at least $\pm 5\%$ (lower bound) of
  - *T*-modifications
  - *P*-modifications
  - *TP*-modifications
  - Ideally even to non-linear scale alterations (e.g. acceleration and deceleration of audio content).
- Efficient operation on large reference databases: this is dictated by the typical use of media monitoring and copy detection systems.
- High identification performance with the ability to avoid false positive detections.

At the time we started working on this topic (the year 2013), no system had been described in the academic literature that was shown and evaluated to meet this set of requirements. Building a reliable system that is efficiently applicable to these use cases indeed was an open problem – sufficient robustness to the mentioned signal modifications, in combination with high identification performance and efficient operation on large audio collections seem to be substantial challenges for automated audio identification systems. We elaborate on that in Chapter 2, where we give an overview of the literature.

To solve these challenges we propose to adapt an elegant method from the field of astronomy (Lang et al., 2010) for the use of audio fingerprinting. Said method is the most important building block of the method that we propose.

Having stated the task as an open research problem, it is important to discern the *academic world* from the *industry*. Audio fingerprinting systems are claimed to be massively used in the industry, apparently with great success. We do not know what algorithms are in use, and to which degree they have been evaluated. For us, it is impossible to falsify the claims made by the industry, therefore we are forced to conclude that the industry has discovered solutions to the challenges, and in this regard seems to be many years ahead of the advances in the academic world.

1.3 CONTRIBUTIONS

We propose and contribute a robust, high-performing and practical audio fingerprinting system, along with a free dataset of DJ mixes for experimental research purposes.

Our contribution is the first method in the academic literature that meets all of the robustness and performance requirements we mentioned above. Its basic concept is to extract local maxima from the spectrogram, and to group these into four-dimensional feature combinations called "quads", which can be represented in a way that is invariant to translation and non-isotropic scaling (i.e. different scale factors can be applied to either dimension). A query is then answered by comparing the query feature combinations with a collection of reference feature combinations. As an integral part of the matching process, the system uncovers and quantifies potential scale modifications in the query audio.

We show that our method is efficient despite our large reference collections[3], and the large search problem that is caused by scale invariances of hashes and their robustness to signal modifications. Specifically, we designed the proposed system for inexpensive hardware, show its capabilities on a standard desktop computer, and intentionally avoid to make use of expensive and power-hungry GPUs.

The robustness of our method is not limited to linear or non-linear time-scale and frequency-scale modifications of query audio. In addition, the method retains strong robustness to noise distortions, which we demonstrate to be on-par with the Audfprint system's noise robustness (Audfprint implements an algorithm similar to that described by Wang (2003)).

3 Large in the context of the database sizes used for evaluation in the academic literature. We evaluate our system on reference collections of the size of 20,000 full-length songs in Sonnleitner and Widmer (2014), 100,000 in Sonnleitner and Widmer (2016) and 430,000 full-length tracks in Sonnleitner, Arzt, and Widmer (2016), while the industry apparently works on data volumes of tens of millions of reference songs.

According to our experiments and performance evaluations, the system is directly usable for various fingerprinting application areas. Next to typical fingerprinting use cases these are for example audio copy detection, media monitoring, song duplicate detection and the analysis of scale modifications in (long) recordings.

We evaluate our work on a free, Creative-Commons licensed reference collection of 430,000 full-length songs for *T*-, *P*-, and *TP*-alterations of up to $\pm 30\%$. To put this into perspective, listening to the entire reference collection would take over three years and four months (more than 29,500 hours of audio content). The reader might raise the question of whether scale modification robustness to $\pm 30\%$ or more is necessary, or even reasonable. We aim for our system to be able to reliably monitor DJ performances. As we are not aware of a study of typically encountered scale modifications, and we assume that these may likely change due to trends or individual artistic expression, we cannot tell which degree of robustness is *necessary*. In general, we argue that for a system to work reliably, it should not be operated near its limits.

To show the applicability of our method to the monitoring of DJ mixes, we also evaluate the work on two different DJ mix datasets. One contains real-world data – DJ mixes recorded in discotheques and clubs. This data set is proprietary and cannot be made public. The second data set is a contribution of ours to the research community: We create and release a creative-commons licensed dataset (the second dataset) of free DJ mixes along with reference tracks and song-border annotations, and show an evaluation with baseline results of our proposed method on this dataset. We hope that this dataset will be of use to the community and helps to further deepen the research and advances in this field.

Parts of the work presented here have been published in three papers:

- Chapter 3:
  R. Sonnleitner and G. Widmer (2014). "Quad-Based Audio Fingerprinting Robust to Time and Frequency Scaling". In: *Proceedings of the International Conference on Digital Audio Effects (DAFX 2014)*. Erlangen, Germany, pp. 173–180

- Chapter 4:
  R. Sonnleitner and G. Widmer (2016). "Robust Quad-Based Audio Fingerprinting". In: *IEEE/ACM Trans. Audio, Speech & Language Processing* 24.3, pp. 409–421

- Chapter 7:
  R. Sonnleitner, A. Arzt, and G. Widmer (2016). "Landmark-Based Audio Fingerprinting for DJ Mix Monitoring". In: *17th International Society for Music Information Retrieval Conference (ISMIR 2016)*

Other parts of this thesis – notably, Chapters 5, 6 and 8 are unpublished and appear here for the first time.

1.4 AUDIO RETRIEVAL AND IDENTIFICATION

Due to the large amount of audio data, there is notable interest in technologies that allow to search for audio content and support the organisation of multimedia data. Multimedia search can be coarsely categorized into content-based, metadata-based, and multi-modal approaches. We will only consider content-based approaches. These operate solely on a representation derived from the underlying signal. An excellent overview of content-based retrieval and identification is given in Grosche, Müller, and Serrà (2012).

The field of audio identification encompasses topics such as fingerprinting, watermarking, live-song identification, cover-song identification and audio-matching. These pursue different goals, and therefore also use different approaches. However, there exist some common requirements, as for example the requirements for efficient processing and robustness to distortions (see for example Casey, Rhodes, and Slaney (2008), Kurth and Müller (2008), and Grosche and Müller (2012)). Retrieval systems such as cover-song detection (see for example Serrà and Gómez (2008) and Ellis and Poliner (2007)) and audio-matching aim at retrieving lists or sets of audio pieces (or their associated metadata) from known collections via the design and extraction of content-based features, and subsequent *similarity* search in large reference collections. However, their use-cases, applied algorithms, and evaluation methodologies are very different from those that are commonly used in audio fingerprinting. The most apparent differences are that cover-song retrieval, live-song identification and audio-matching have to consider some measure of *similarity* rather than *identity*. For similarity-based retrieval it is not always decidable which result is the correct one, and typically these tasks are formulated as ranking-problems. Further, the methods used in cover-song detection and audio-matching aim to retrieve pieces that *musically* correspond to each other. This motivates the design and use of features and feature-combinations that can capture and describe musical properties. For the complex task of cover-song detection for example, the features need to capture the key, beats and downbeats, structure and maybe even the lyrics of a song, while at the same time representing the features in a way that is robust to modifications thereof. Such feature combinations are in general of high dimensionality. A standard feature that is successfully used in these fields is the "chroma"-feature (a specific type of "pitch class profiles") (Fujishima, 1999; Wakefield, 1999). The most common western music tuning system is the 12 equal temperament (12-TET) system, and can be made robust to P-modifications in the context of western tonal music. However, different scales need

to be processed with a suitable number of band-subdivisions. Work that makes use of chroma-shifting is described for example by Goto (2003) and Müller, Kurth, and Clausen (2005).

Audio identification and fingerprinting, as well as live-song identification (for live-song identification see for example Rafii, Coover, and Han (2014) and Tsai, Prätzlich, and Müller (2016)), are not formulated as retrieval tasks. Rather, these methods make use of exact search strategies. To give an intuitive example, let us compare this to text search. If we search for the next occurrence of a word, we expect to either see that word highlighted, or to be informed that this word does not exist in the text document. We do not expect to get a list of e.g. synonyms of that word that appear in the text. In the case of exact identification there is at most one correct result (given a duplicate-free reference collection). Therefore, automated audio fingerprinting is not formulated as a ranking-problem.

1.5 AUDIO FINGERPRINTING

Audio fingerprinting, the topic of this thesis, is a technology for the exact identification of audio content. Its typical use is to precisely identify a piece of audio from a large collection, given a (short) query – where a *query* is an audio excerpt that is potentially distorted or modified. Audio fingerprinting is of interest to the industry and also is a well established tool in everyday life – millions of users utilize this technology, e.g. via their mobile phones, to learn about a song that is currently played. The media industry uses automated audio fingerprinting systems to collect information of what content was played at which point in time, for various TV and radio broadcast channels all over the world. In addition to that, several media events and discotheques are monitored in order to assist the revenue management of played content.

Historically, audio fingerprinting emerged from watermarking technology. Here, audio signals are modified to carry additional information in a way that does not alter the perceptual quality. In order to identify signals that are augmented in this way, the injected information (the watermark) is extracted and investigated. The disadvantage of watermarking is that it can only be used for identification if the signal was processed with a watermarking system beforehand. In this regard, an advantage of audio fingerprinting is its non-invasive nature, as signals do not need to be actively modified beforehand.

In contrast to e.g. cover-song detection (see Section 1.4), for audio fingerprinting we do not need to extract musical information of any kind. We argue that feature combinations that capture a notion of musical similarity might even be detrimental to the goal of high precision and specificity, especially when large audio collections are used such that there are many songs for each given genre. In this work,

in line with the majority of all described fingerprinting systems, we design a system that is completely agnostic to what "music" is. Rather, we simply treat our input as acoustic signals without any assumptions on their content.

In the context of this work we define the term *audio fingerprinting* as follows:

> An audio fingerprinting system is an identification system that aims to robustly identify "identical" pieces of audio, where the query *originates* from the reference, but may have been altered by background noise, compression, scale modifications and effects.

A piece of audio that is distorted by noise or modified by effects is still considered identical to the original. However, a live version of a song is not identical to e.g. the studio version, because it does not originate from the same signal. Thus, for audio fingerprinting it is important to extract highly *discriminative* content features, so-called "fingerprints", from the collection of audio files as well as the query piece.

In the following we define a list of feature properties which need to be investigated in detail. For this we want to point out the similarity of audio fingerprinting to biometric systems, which are also required to operate at highest precision and specificity, extract robust features, and to search for matches in exceptionally large databases. In the field of biometry, these properties are referred to as "biometric traits" (Jain, A. Ross, and Nandakumar, 2011). Let us transport these traits to the topic of audio fingerprinting (we exclude "measurability" and "acceptability" which are not of relevance to our scenario):

- Universality: All pieces of audio should carry the feature. To give a counter-example let us consider audio features that represent tonality, which are not always decidable in the context of mostly percussive audio content. In general, audio fingerprinting is not limited to *musical* content. We argue that features that try to capture higher-level musical information may violate the "principle of universality" and are not of advantage to audio fingerprinting.
- Uniqueness: The features should be of sufficient discriminability.
- Robustness/Permanence: The feature should be sufficiently invariant (i.e. robust) to various distortions, e.g. encoding, noise, compression artefacts and audio effects such as time and frequency scale modifications.
- Performance: The features should allow for high identification performance as well as low computational costs of the extraction routine and the matching process.

- Circumvention: It should be hard to imitate or remove the feature. While this property is regarded to be of high importance to watermarking systems, it seems that the literature on audio fingerprinting does not consider the problem of circumvention as relevant.

We design our system in the light of the traits of universality, uniqueness, robustness and performance. In the course of this thesis we implicitly show the "universality" by working on large and diverse datasets, including DJ performances. For "uniqueness", we perform an experiment in Chapter 7 to demonstrate the specificity and precision of the method without taking advantage of the verification process (see Section 4.5.3). The key points of "robustness" and "performance" are shown to be satisfied in various experiments for which we give evaluation results and run times (see Section 6.2). Robustness to compression and resulting artefacts is implicitly tested in all the experiments that are presented in this thesis, with the exception of the experiments shown in Chapter 8. We do this by taking the already compressed audio data which typically are in the mp3-format, and convert all queries to the ogg-format, again with compression. Experiments on "circumvention" are not part of this thesis, but we assume that it would be a worthwhile topic of future work, especially for copy detection systems and automated systems that are used as a basis for the distribution of revenue.

1.6 FINGERPRINTER SYSTEM ARCHITECTURE

The abstract overall architecture of a fingerprinting system is based on the building blocks as depicted in Figure 1.2.

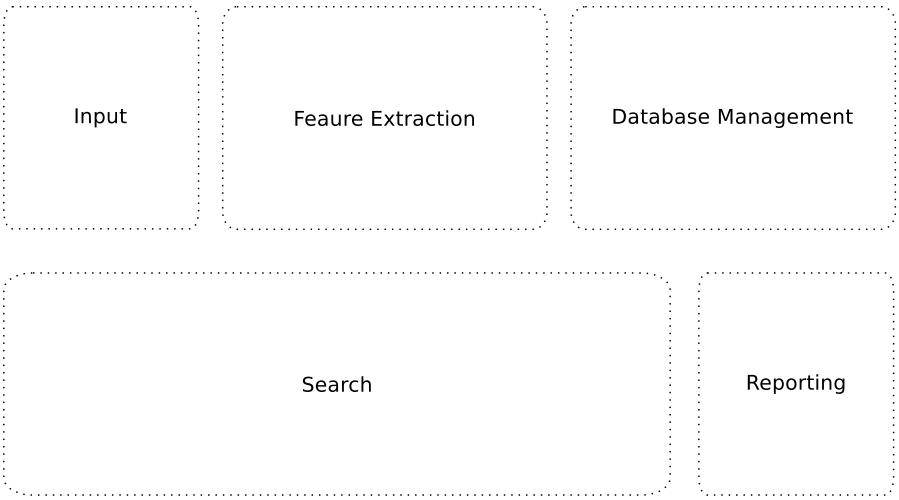


Figure 1.2: Abstract system modules

The systems basically operate in two modi: reference database construction and query processing. The database is usually constructed

from whole tracks or songs, while query audio pieces typically are short excerpts. In both cases, the input is processed by a feature extraction stage, where the audio is decoded, transformed into a suitable representation, and subsequently processed to obtain feature descriptions (i.e. the fingerprints). Ideally, the extracted fingerprints are universal, highly discriminative, robust, cheap to compute and of low storage demands.

For reference database construction, the extracted fingerprints are stored in a way that supports efficient search. After its construction, the reference database comprises all the content that can possibly be identified by the system.

To answer queries, a search method compares the extracted fingerprints from the query excerpt to the fingerprints stored in the reference database. The reporting module then returns the result, with any kind of associated additional information and meta-data.

In the course of this thesis, we will part for part refine this general architecture to the specific method that we propose, and focus on details of each component. In the end, we arrive at descriptions of all components and subcomponents. The result is a complete audio fingerprinting system capable of dealing with noise, effects, and severe linear and non-linear scale modifications. The final architecture of our specific method is depicted in Figure 1.3. Where appropriate, we will use these figures with highlighted components for a quick cue of what the following sections are about.

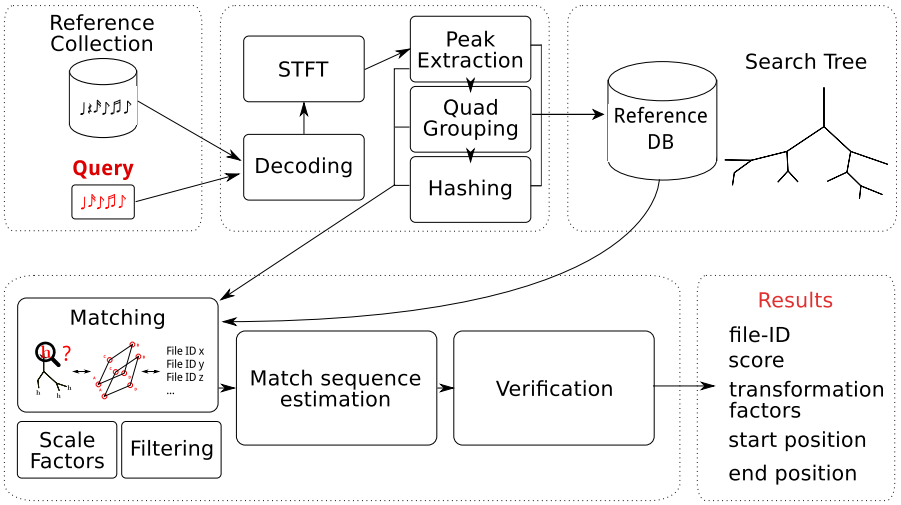


Figure 1.3: Detailed system architecture

1.7 EVALUATION AND PERFORMANCE MEASURES

In this thesis, the performance of our method is evaluated for various distorted queries and long recordings of DJ mixes.

The following terms are used in defining our performance measures:

- *tp*: *true positives* is the number of cases in which the correct reference is identified from the query.
- *fp*: *false positives* is the number of cases in which the system predicts the wrong reference.
- fp_s: *false positives* in the context of specificity are the cases where the system reports a match despite the fact that the query is not identifiable because there is no reference. Thus, we have two types of false positives (*fp* and fp_s), depending on the case whether the reference is present (identifiable) or not (non-identifiable).
- *fn*: *false negatives* is the number of cases in which the system fails to return the result, i.e. instead of reporting the correct match the system returns an empty result.
- *tn*: *true negatives* is the number of cases where the system correctly abstains from identifying a reference because there *is* no correct reference.

Based on these, we define three performance measures:

- *Recognition Accuracy* is the proportion of queries whose reference is correctly identified:

$$\text{Accuracy} = \frac{tp}{tp + fp + fn} = \frac{tp}{N} \tag{1.1}$$

  where N is the sum of all possible outcomes, i.e. true positives, false positives and false negatives. We report the accuracy for cases where the correct reference is represented in the database. In this case the query is *identifiable* and there are no true negatives.

- *Precision* is the proportion of cases, out of all cases where the system claimed to have identified the reference, where its prediction is correct:

$$\text{Precision} = \frac{tp}{tp + fp} \tag{1.2}$$

  Thus, high precision means a low number of false positives. It is important to assess the precision of fingerprinting systems, as in many applications a false positive is regarded more expensive than a false negative. In media monitoring and revenue distribution, a false positive may lead to revenue attribution to the wrong artist, and in copy detection, to false accusations.

- *Specificity* is the proportion of non-identifiable cases where the system correctly abstains from reporting a result (because the query is not present in the reference database).

$$\text{Specificity} = \frac{tn}{tn + fp_s} \tag{1.3}$$

> High specificity quantifies the capability of the system to avoid false positives by correctly abstaining from reporting a match. We will use this performance measure in cases where we know that certain queries are not represented in the database.

In this thesis we show query-based, and recording-based experiments. Query-based experiments consider independent short queries in the range of 3 to 20 seconds. These are explained and evaluated in Chapters 3, 4, 6. For these experiments the performance measures tp, fp, fn are counted based on the *number* of queries, i.e. one correctly identified query is one true positive. For recording-based experiments, where the query content consists of long audio recordings, the measures tp, fp, fn, tn, fp_s reflect the number of *seconds* that are correctly or falsely claimed by the system. Recording-based experiments are shown in Chapters 7, 8.

1.8 ORGANISATION OF THIS THESIS

This document is organised as follows. In this present chapter we have introduced the concepts and the terminology for the subsequent chapters of this thesis. Next, in Chapter 2 we discuss the literature on audio fingerprinting.

We describe our initial research on the topic of robust audio fingerprinting in Chapter 3. It is included in this thesis in order to demonstrate the chronological advancements of our proposed "quad-based" audio fingerprinting method that we call "Qfp". To obtain fingerprints, we construct quadruples of local maxima in the spectrogram, and subsequently represent these as translation invariant hashes, that are also invariant to non-isotropic scaling. The matching of fingerprints is done via range search in tree data structures. To make the system operate with high precision, we integrate a powerful verification process that effectively discards false positive candidates. The method is robust to scale modifications of query audio, and quantifies the scale modification factors that are encountered when processing a query. This chapter is heavily based on the following publication:

> R. Sonnleitner and G. Widmer (2014). "Quad-Based Audio Fingerprinting Robust to Time and Frequency Scaling". In: *Proceedings of the International Conference on Digital Audio Effects (DAFX 2014)*. Erlangen, Germany, pp. 173–180

This chapter is intended to describe the early concept of the method, and since its publication we have managed to considerably improve and refine almost all of its aspects. The chapter might be of interest if a reader is working towards an implementation of our method, without wanting to directly integrate the more complex routines for increased efficiency of the method. The experiments and results in

the chapter allow for comparison on smaller datasets using a non-optimized implementation. A reader who directly wants to learn about the current state of the method with all the refinements is invited to skip Chapter 3, and to proceed with Chapter 4. There, our method "Qfp" is described in its entirety.

In Chapter 4 we refine the initial method and achieve a high-performing and scalable system that is evaluated to be robust to various effects and severe scale modifications in the range of $\pm 30\%$, while operating on a reference collection of 100,000 full-length audio files. To do so, we refine the quad grouping process to extract the fingerprints in an almost uniform distribution over time. Then, quads are filtered to keep only the "strong" ones. Query quad extraction is extended to adapt to tolerance bounds, and we further introduce a subspace constraint in order to dismiss query quad candidates if they originate from an invalid hash-subspace. The search algorithm is replaced, such that we now use a so-called shallow "bounding volume hierarchy" (a 4-ary "BVH") that enables us to perform range search in an efficient way. The chapter is heavily based on the following publication:

> R. Sonnleitner and G. Widmer (2016). "Robust Quad-Based Audio Fingerprinting". In: *IEEE/ACM Trans. Audio, Speech & Language Processing* 24.3, pp. 409–421

Chapter 5 elaborates on the core element of the fingerprinting system: the tree-based search algorithm, the tree construction and its traversal. We refine the internal data organisation and switch to a compressed representation of reference records. This paves the way to lower storage demands and more efficient processing, and allows the system to operate on large reference databases. We here use the largest free and reproducible dataset that we know of: it consists of almost 430,000 full-length audio pieces. These refinements allow us to process queries against the large reference database with even lower run times than what we achieved on the smaller reference database of just 100,000 tracks. Even on this large database, the search algorithm allows us to answer a 20 second long query that may be scale-modified in the range of $\pm 30\%$, within one second.

In Chapter 6 we present more extended and systematic experiments on the large reference database of 430,000 tracks. We show the performance of the algorithm on scale-modified queries in our typical range of $\pm 30\%$, include tests for varying lengths of query snippets, varied numbers of query-quads per second, and non-linear scale modifications as well as noise robustness. Then, we show the runtime that is spent in various stages of our system.

Chapter 7 focuses on our initial research question: Can we apply our proposed system to monitor DJ performances, despite the challenges given by the strong robustness demands? The chapter is heavily based on our third publication in the field:

> R. Sonnleitner, A. Arzt, and G. Widmer (2016). "Landmark-Based Audio Fingerprinting for DJ Mix Monitoring". In: *17th International Society for Music Information Retrieval Conference (ISMIR 2016)*

We discuss the applicability of peak-based methods on DJ performances, and compare the identification performance of our method to two fingerprinting systems that are freely available for research purposes. The comparisons and evaluations are based on two datasets: live DJ mixes that were recorded in discotheques, and a freely available dataset of DJ mixes that we create and publish along with song-border annotations and reference tracks.

In Chapter 8 we give a case study for the application of our system to the task of media monitoring, where we show how to use our proposed system to work on entire hours of media content instead of on successive short and local queries. For this, we build a sequence detector on top of the proposed fingerprinting system, and give an in-depth discussion of the results obtained on the free DJ dataset, and a summarization of results for the non-free and unpublished disco-dataset. The system is directly applicable to fully automated media monitoring tasks, but could also be used to generate reports that visualise the content in an intuitive way that might help in supervised use cases where system results need to be refined by human inspection.

Finally, in Chapter 9 we conclude our work.

1.9 DATASET AVAILABILITY

Information on how to obtain and reconstruct our reference collection that consists of almost 430,000 tracks, and the "mixotic"-set, can be found online on the department's web-page: `http://www.cp.jku.at/datasets/fingerprinting/`

2 LITERATURE OVERVIEW

In the previous introductory chapter we described the topic of audio fingerprinting, and discussed the requirements and challenges to build an efficient and robust automated system. The field of audio fingerprinting is an established research topic and also of interest to the industry, with a rich variety of contributions described in the literature.

In this chapter we discuss previous work and work that is more directly related to the focus of this thesis, which is to build a system that can efficiently and robustly operate on large collections of data.

We organize the literature review according to the robustness requirements that we defined earlier in Section 1.2.2. To do so we categorize the literature as follows.

- Work that does not consider robustness to time or frequency modifications. This category contains the majority of the literature (see Section 2.1).
- Work that proposes solutions for robustness to some types of scale modifications, but not to each of the types (*P*-, *T*-, *TP*–alterations) at the same time (see Section 2.2).
- Work that proposes solutions for robustness to scale modifications in general, i.e. any combination of *P*-, *T*-, *TP*–modifications. This category is of interest for copy-detection and media monitoring use-cases, and our main focus (see Section 2.3).

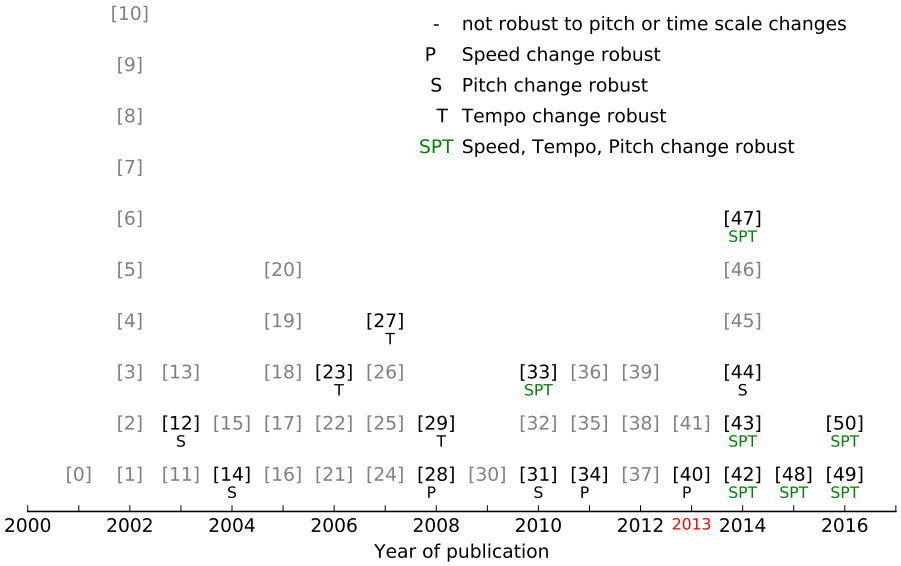
In Figure 2.1 we visualize the publications by year, and highlight those that aim to achieve robustness to changes in pitch or time-scale. The highlighted publications are categorized as described in the list above.

In Table 2.1 we show the sizes of reference collections (in hours of audio) and the reported evaluation measures of the highlighted methods.

The last section in this chapter is devoted to the work that inspired our fingerprinting method.

2.1 LITERATURE NOT CONSIDERING SCALE CHANGE ROBUSTNESS

In this section we give an overview of previous work in the field of audio fingerprinting, that does not consider robustness to effects that arise from time or frequency scaling of query audio.



(a) Literature by year of publication

Idx	Literature	Idx	Literature	Idx	Literature
0	Haitsma, Kalker, and Oostveen (2001)	17	Ke, Hoiem, and Sukthankar (2005)	34	Fenet, Richard, and Grenier (2011)
1	Kurth, Ribbrock, and Clausen (2002)	18	Seo et al. (2005)	35	Chandrasekhar, Sharifi, and D. A. Ross (2011)
2	Sukittanon and Atlas (2002)	19	C. J. C. Burges et al. (2005)	36	Ramona and Peeters (2011)
3	Haitsma and Kalker (2002)	20	Cano et al. (2005)	37	Porter (2012)
4	Batlle, Masip, and Guaus (2002)	21	Smeaton, Over, and Kraaij (2006)	38	Six and Cornelis (2012)
5	Miller, Rodriguez, and Cox (2002)	22	Ramalingam and Krishnan (2006)	39	Anguera, Garzon, and Adamek (2012)
6	Cano, Batlle, Mayer, et al. (2002)	23	Kurth, Gehrmann, and Müller (2006)	40	Ramona and Peeters (2013)
7	Cano et al. (2002)	24	Betser, Collen, and Rault (2007)	41	Fenet (2013)
8	C. J. Burges, Platt, and Jana (2002)	25	Cano Vila and Serra (2007)	42	Sonnleitner and Widmer (2014)
9	Lu (2002)	26	Ogle and Ellis (2007)	43	Malekesmaeili and Ward (2014)
10	Herre, Hellmuth, and Cremer (2002)	27	Baluja and Covell (2007)	44	Ouali, Dumouchel, and Gupta (2014)
11	C. J. Burges, Platt, and Jana (2003)	28	Bellettini and Mazzini (2008)	45	Yu et al. (2014)
12	Haitsma and Kalker (2003)	29	Baluja and Covell (2008)	46	Schreiber and Müller (2014)
13	Wang (2003)	30	Liu, Yun, and Kim (2009)	47	Six and Leman (2014)
14	Bardeli and Kurth (2004)	31	Dupraz and Richard (2010)	48	Zhang et al. (2015)
15	Batlle et al. (2004)	32	Lang et al. (2010)	49	Sonnleitner and Widmer (2016)
16	Goldstein, Plat, and C. J. Burges (2005)	33	Zhu et al. (2010)	50	Sonnleitner, Arzt, and Widmer (2016)

(b) Resolving the citation indices to the bibliography

Figure 2.1: Literature by year of publication (Figure 2.1a), annotated with the type of scale change robustness. Publications of the same year are stacked vertically, without specific order. Grey colored citation indices denote literature that does not consider robustness to any kind of scale modifications. Black citation indices represent work that is robust to at least one kind of scale modifications, where "S" is shorthand for what we defined as *TP*-modification, and "T" and "P" denote *T*- and *P*-modifications. "STP" denotes *P*-, *T*-, *TP*–modification robustness. Table 2.1b resolves the citation indices given in Figure 2.1a.

Idx	Literature	DB size [h]	Eval. measures
12	Haitsma and Kalker (2003)	?	A
14	Bardeli and Kurth (2004)	40	A
23	Kurth, Gehrmann, and Müller (2006)	7	A
27	Baluja and Covell (2007)	667	A
28	Bellettini and Mazzini (2008)	1000	A
29	Baluja and Covell (2007)	667	A
31	Dupraz and Richard (2010)	5	A, P
33	Zhu et al. (2010)	21	A
34	Fenet, Richard, and Grenier (2011)	583	A, P
40	Ramona and Peeters (2013)	8.34	P, R
42	Sonnleitner and Widmer (2014)	1333	A, P
43	Malekesmaeili and Ward (2014)	16.67	A, P
44	Ouali, Dumouchel, and Gupta (2014)	400	P, R (F-measure)
47	Six and Leman (2014)	2000	A, P, S
48	Zhang et al. (2015)	169	A
49	Sonnleitner and Widmer (2016)	6666	A, P, S
50	Sonnleitner, Arzt, and Widmer (2016)	29,500	A, P, S

Table 2.1: Size of reference collections (in hours) and reported evaluation measures for methods robust to time or frequency modifications. For evaluation measures, "A", "P" and "S" denote accuracy, precision and specificity. "R" denotes recall ($tp/(tp+fn)$).

Haitsma, Kalker, and Oostveen (2001) propose to create robust hashes which are represented as bit-strings and allow for subsequent identification of audio by comparing query hashes to reference hashes. This work has become a classic and influential method.

The work of C. J. Burges, Platt, and Jana (2002) investigates means of applying dimensionality reduction on fingerprint hashes, and evaluates the identification performance of the system on 36 hours of audio data.

Cano, Batlle, Mayer, et al. (2002) observe that for the application of audio identification systems to broadcast data, additional robustness demands must be fulfilled. Broadcasters might speed up the audio to play more songs per hour, or to get more attraction to listeners. In this work, *TP*-changes of up to 2.5% are observed in radio broadcasts. Speeding up the audio not only modifies the time scale, but also influences the pitch, and thus can negatively impact the identification performance. The evaluation contains experiments on radio broadcasts with high performance, but the extents of *TP*-modifications in the broadcast dataset are not known.

The work of Miller, Rodriguez, and Cox (2002) explores ways to make computationally expensive nearest neighbour search in high dimensional binary spaces applicable to the task of audio fingerprinting.

A system that operates on features from MPEG-7 content descriptors is given by Herre, Hellmuth, and Cremer (2002).

Kurth, Ribbrock, and Clausen (2002) propose a system for identification of distorted audio, to leverage fingerprinting technology for content that was recorded via mobile phones. The work gives insight in the feature extraction and representation from the view point of coding theory, and highlights the challenges of identifying distorted audio in a highly elaborate way. The description of the feature extraction process mentions an example of the extraction of local signal minima and maxima. Local maxima will later become the base feature representation for an influential identification method, also targeted at the identification of mobile phone recordings (Wang, 2003).

Lu (2002) analyse time-frequency variations in audio signals using the one-dimensional wavelet transform. It seems that at this time, the term fingerprinting gains momentum in the research field, discerning itself from watermarking, and is described as a non-invasive non-watermarking technique. The robustness of watermarking methods is frequently investigated in the context of attack scenarios. Initially, creating different encodings of audio is viewed as a possible attack against fingerprinting systems.

Sukittanon and Atlas (2002) highlight the importance of extracting fingerprints that are invariant to time and frequency distortions, and propose two dimensional modulation frequency features. The evaluation takes into account linear time shifts as well as dynamic time normalization (AGC), and includes tests for misaligned audio of up to five seconds.

An early review of different algorithms for audio fingerprinting is given by Cano et al. (2002). Typical building blocks of fingerprinting system architecture are mapped to a proposed unified framework.

The application of automated audio identification systems to noisy broadcast data is investigated in the work of Batlle, Masip, and Guaus (2002), using an HMM audio model with Viterbi decoding.

C. J. Burges, Platt, and Jana (2003) apply distortion discriminant analysis to achieve robust feature extraction and dimensionality reduction.

Wang (2003) describe a system that creates geometric hashes from local spectrogram maxima. While individual hashes are of low specificity, sequences of matches over time show high specificity. The proposed method of searching for sequences of matched hashes constitutes a very efficient algorithm. This has become a classic and influential method, generally known as “Shazam”. The hashing model, however, does not exhibit robustness to any type of scale modifications. The system is considered to be highly robust to additive noise.

Batlle et al. (2004) present a HMM based fingerprinting method, and discuss scalability issues. Discussing the scalability, the authors split the “cost” of the system in two parts. The first part considers

the feature extraction, and is said to be constant. The other part considers the cost depending on the number of reference pieces, which is described to have linear behaviour. We assume that this property prevents its application to large databases.

C. J. C. Burges et al. (2005) show the applicability of a previously described method (C. J. Burges, Platt, and Jana, 2003) to duplicate song detection and, notably, to audio thumbnailing (i.e., creating a short and representative audio excerpt from the given recording).

Cano et al. (2005) give a review of audio identification methods. The work elaborates on watermarking, fingerprinting and proposes a unified framework for fingerprinting systems. Interestingly, the review does not mention work related to peak-based fingerprinting methods.

Seo et al. (2005) present a system that offers robustness to small linear time scale changes, as well as equalization and mp3 compression. Here, the fingerprints are represented by normalized spectral sub-band centroids.

Goldstein, Plat, and C. J. Burges (2005) develop a way to efficiently search large amounts of fingerprints using tightly bound hyperrectangles that represent high dimensional space, along with redundant bit-vectors to store the index.

Ke, Hoiem, and Sukthankar (2005) show that audio fingerprinting can be approached via computer vision techniques. The system can identify query pieces that are severely distorted by noise.

Ramalingam and Krishnan (2006) represent audio using Gaussian mixture models (GMMs), based on various spectral features, and compare the performance of their method to several different fingerprinting schemes.

The thesis by Cano Vila and Serra (2007) includes a thorough description of audio fingerprinting, along with the description of a general framework.

The application of fingerprinting to detect repeating audio events of personal recordings is investigated by Ogle and Ellis (2007). It successfully retrieves sound events of almost exact identity, like telephone rings. As fingerprinting operates at high discriminativity, it is not applicable to events that are similar rather than identical, like garage door opening sounds and natural sounds.

Betser, Collen, and Rault (2007) present a sinusoidal-component-selection based audio identification system for jingle detection. It proposes sinusoidal fingerprints that are obtained from a four-stage extraction process that first applies a low-pass filter, then peaks are extracted using a Discrete Fourier Interpolator using phases. The third step is to select dominant peaks from the output of the previous stage, and finally performing frequency compression to a 16bit data type. The system is tested on a corpus of 18 hours of French radio broadcast data, and the results are compared to the results obtained by the method of Haitsma, Kalker, and Oostveen (2001) (we assume a

reimplementation of said method was used in the experiments). The proposed method performs better in terms of recall (8 percentage points), and could cover a larger fraction of the duration of played pieces (+19 percentage points). It is reported that both algorithms operate with perfect precision on the test collection.

Liu, Yun, and Kim (2009) propose a multiple-hashing approach via computing the discrete cosine transform (DCT) on sub-bands, and report higher performance compared to the system of Haitsma and Kalker (2002).

A survey of algorithms and methods for mobile query by example applications is given by Chandrasekhar, Sharifi, and D. A. Ross (2011). The work compares three algorithms (Ke, Hoiem, and Sukthankar, 2005; Wang, 2003; Baluja and Covell, 2008) via ROC performance and storage demands. The work discusses the reduction of latency for mobile audio fingerprinting applications.

Anguera, Garzon, and Adamek (2012) propose to compute masks near the spectral peaks in the spectrogram, and use those for robust audio fingerprinting.

Porter (2012) creates and discusses an evaluation framework for audio fingerprinting.

Six and Cornelis (2012) propose a global-feature based identification system via the comparison of pitch class histograms to assist in the cataloguing of ethnic music archives. The system creates similar features for similar-sounding tracks.

The doctoral thesis by Fenet (2013) explores various fingerprinting schemes and the applicability to large-scale search.

Schreiber and Müller (2014) investigate modifications to the method of Haitsma and Kalker (2002) in order to accelerate the lookup and matching process.

Yu et al. (2014) propose hybrid high performance data structures for indexing massive amounts of audio fingerprinting data for efficient search.

2.2 METHODS THAT ARE ROBUST TO SPECIFIC TYPES OF SCALE CHANGES

Here, we introduce related work that focuses on methods that are robust to specific types of scale modifications. Some of these provide solutions for robustness along one dimension, e.g. either time or frequency. These will be introduced first. Next, we describe related work that tries to solve the complex task of building a systems that is robust to modifications in both dimensions at the same time. We observe that this task is rarely addressed, even though it is a requirement for reliable media monitoring and copy detection systems.

2.2.1 *Robustness to P-modifications*

Literature in this category is not robust to *T*- or *TP*-changes, but achieves robustness to *P*-modifications. Work of this category is described by Bellettini and Mazzini (2008), Fenet, Richard, and Grenier (2011), and Ramona and Peeters (2013).

Bellettini and Mazzini (2008) note that robustness to *P*-changes can be commercially relevant for the monitoring of ambient content where pitch shifting might be used to achieve a homogeneous transition of two tracks. The proposed system extends the work of Haitsma, Kalker, and Oostveen (2001) to pitch scale robustness. To do so they propose to exhaustively search for query representations that are *P*-modified within a predefined range. The evaluation considers pitch shifting up to ± 8 semitones, which is in the range of approximately $[-37\%, +59\%]$. The dataset for the experiments consists of 15,000 pieces, and is split in two sets: 4500 Italian pieces, and 10500 pieces of non-Italian, western music. We do not know the length of the pieces and the amount of hours of audio content in this dataset. The evaluation is shown separately for both datasets, and for two versions of the algorithm. Two versions of the proposed method are described, where the considerably higher-performing one applies a 12-TET (12 pitch classes of equal tempered scales. This is often also referred as "chroma") band subdivision. On the (smaller) Italian dataset, the method achieves perfect detection success rates for queries that are pitched in the range of roughly $[-21\%, +33\%]$. Outside of this interval the success rate steeply drops below 40%. On the non-Italian dataset, a perfect success rate is achieved in the range of approximately $[-16\%, +19\%]$ pitch shifted content. For severely pitch shifted audio in the range of roughly $[-25\%, +41\%]$, the success rate is still $\geq 50\%$.

Fenet, Richard, and Grenier (2011) achieve *P*-modification robustness by extending the hashing model of Wang (2003) to include a representation of quantized pitch offsets. The method is evaluated on an audio stream of 7 hours, recorded from a French radio station. For experiments, a small and a large reference database are used, to show the scalability of the method. The small reference database contains roughly 122 hours of audio created from 1 minute long excerpts of a total number of 7309 songs. The query stream contains 459 of the reference pieces. The evaluation follows a comparative setup, where the proposed method with *P*-robustness is compared to the baseline method of Wang (2003), which is not robust to scale modifications, at all. The proposed method achieves to detect 447 of the 459 occurrences (97.4%), while the baseline method detects a lower number of 381 occurrences (83%). From this difference it is concluded that radio stations make considerable use of pitch shifting effects, and that the proposed method effectively manages to achieve *P*-alteration robustness. The large reference database consists of 30,000 song excerpts (500

hours), and the query streams used in this experiment are recordings of 5 days of radio broadcast content. Here, the method achieves to detect 496 out of 506 occurrences. The ground-truth for the evaluation is obtained via manual annotation, and contains the reference track ID together with its time segment within the broadcast. The extents of scale modification that actually occur in the test dataset are unknown. The experiments do not investigate or quantify the extents of *P*-change robustness of the algorithm, and do not report precision and specificity.

Ramona and Peeters (2013) propose to apply cosine filters to the audio spectrum to compute short-term band energies, which makes it robust to moderate frequency distortions. The method is applicable to large collections of audio and is robust to audio degradations and to small amounts of *P*-distortions. According to the evaluation of broadcast data against 8.3 hours of reference audio, the system outperforms the re-implementation of the system by Haitsma and Kalker (2002) by 8.5 percentage points, and the reimplementation of the system by (Wang, 2003) by 13.4 percentage points.

2.2.2 *Robustness to T-modifications*

Methods that incorporate robustness to *T*-modifications, but not to *P*- or *TP*-changes are described by Kurth, Gehrmann, and Müller (2006), Baluja and Covell (2007), and Baluja and Covell (2008). The work proposed by Kurth, Gehrmann, and Müller (2006) strictly speaking is not a fingerprinting method as it aims to identify audio using a higher degree of similarity in order to detect versions or live performances of music pieces. We consider this work in this thesis because the proposed features allow for considerably strong *T*-modification robustness. This is achieved by introducing musically informed, tempo related audio features. Evaluated on 7 hours of audio, the system achieves high identification performance on queries that are time-scaled from 79% to 126% relative to the reference piece.

Baluja and Covell (2008) propose a method commonly known as "Waveprint". The STFT representation of audio is processed in overlapping slices along time. From these slices, wavelets are computed and compared by their magnitude, and the top-200 wavelets are compressed based on individual sign bits. Using Min-Hash, the resulting bit vectors of a spectrogram slice are represented as 200 byte values, which are indexed for nearest neighbour lookup via LSH. The method is robust to *T*-changes in the tested range of $\pm 10\%$, but sensitive to moderate *TP*-changes in the range of $\pm 2\%$. The identification performance is evaluated on a reference database of roughly 583 hours, created from 10,000 songs. The evaluation presents accuracy as the measure for audio identification, and does not report specificity and precision.

2.2.3 *Robustness to TP-modifications*

Methods robust to *TP*-changes, but not to *P*- or *T*-modifications of query audio are described by Haitsma and Kalker (2003), Bardeli and Kurth (2004), Dupraz and Richard (2010), and Ouali, Dumouchel, and Gupta (2014).

Haitsma and Kalker (2003) refine their previous method (Haitsma and Kalker, 2002) to be robust to linear time scale changes of up to 6%. The evaluation is based on within-system measures, i.e. on bit error rates (BER) of the proposed hashes. The experiments are performed on four audio clips, and compared to their previously published method (Haitsma and Kalker, 2002).

The system proposed by Bardeli and Kurth (2004) operates on features that are extracted from segmented audio spectrograms, and then translated into codes. The resulting codes are learned from split datasets. For the proposed algorithm, the identification runtime depends on the signal complexity of the query piece. The method can identify time scaled query audio, and achieves recognition rates of around 85% for audio that was modified in time-scale by $\pm 10\%$. In addition it reports estimates of the underlying time-scale factor with respect to the reference. The evaluation is performed on 600 songs.

The work presented by Dupraz and Richard (2010) extends and optimizes the approach of Betser, Collen, and Rault (2007). The robustness is achieved by an exhaustive preprocessing step that searches for common pitch offsets of query fingerprints against the reference database. If a subset of reference fingerprints exhibit a constant pitch offset to the query, this offset is used to rescale the query for subsequent fingerprinting. The method is robust to *TP*-changes of up to $\pm 5\%$, which is what we defined as lower bound for reliable media monitoring methods in Section 1.2.2. The exhaustive query process may be a limiting factor for its applicability. Two databases are used for evaluation, where the smaller one serves the purpose to assess the scale robustness of the method, and the larger one is used to show the overall system performance for two modes of operation. The smaller database consists of 50 minutes of audio, created from 303 files, each of 10 seconds length. The larger database contains just under 5 hours of audio, created from 1772 files. The evaluation for the robustness to *TP*-changes considers *TP*-modifications for $\pm 1\%$ and $\pm 5\%$, where the results are as follows: 96% true positives, 4% false positives for $\pm 1\%$, and 89% true positives, 11% false positives for $\pm 5\%$ in *TP*-modification. The false positive rate for un-altered queries is 2%.

An audio identification system for copy detection is presented by Ouali, Dumouchel, and Gupta (2014). Quantized spectrogram regions are transformed to a series of horizontal and vertical slices, which are then represented as 48 dimensional fingerprints. Match candidates are determined in an exhaustive fashion, and robustness

to *TP*-changes is achieved by additional search with rescaled versions of the query snippet. While this approach to achieve scale robustness is effective, its exhaustive nature prevents it from being applicable to larger reference collections. The system is evaluated on the TRECVID 2010 dataset (Smeaton, Over, and Kraaij, 2006). The work adds robustness to *TP*-modifications, but not to *T*- or *P*-modifications.

2.3 METHODS THAT ARE ROBUST TO COMBINATIONS OF SCALE MODIFICATION TYPES

Regarding the main focus of our work – building a system that meets typical robustness requirements, and on top of that extends the robustness properties to *both* time and frequency scale modification, i.e. *P*-, *T*-, and *TP*-modifications – at the time we started working on the topic we could identify two publications that suggest promising *features* (Zhu et al., 2010; Malekesmaeili and Ward, 2014). However, these publications do not propose a way to make use of the features in a complete system, i.e. the matching process is left open. The reference collections that are used to evaluate the feature robustness consist of 21 and roughly 17 hours of audio, and thus are too small to gain insight on how the precision and specificity might be impacted in the presence of larger reference collections.

Zhu et al. (2010) propose to use 128-dimensional SIFT features, and seem to be the first to utilize a feature extraction process that is robust to severe changes in both, time- as well as frequency-scale. The work focuses on the evaluation of robustness of the representations, and uses exhaustive search for the matching of candidates. The paper does not propose a way to build a full system for the SIFT features, and focuses on testing the feature applicability, leaving the issue of testing a suited search algorithm unanswered. The identification performance is evaluated on a database that consists of roughly 20.7 hours of audio content, created from 1241 excerpts of 1 minute length, each. The work is reported to achieve near-perfect results for *T*- and *P*-modifications (but not for *TP*-modifications) in the ranges of 65% to 150%, and -50% to $+100\%$, respectively. The work reports the accuracy as evaluation measure.

Malekesmaeili and Ward (2014) propose to compute scale invariant features from two-dimensional time-chroma representations of spectrogram patches. To our knowledge this is the second work in the field of audio fingerprinting that manages to achieve strong robustness to time and frequency scale modifications. In line with the first of such methods (Zhu et al., 2010), the focus is put on the feature extraction, but does not show and evaluate a suited search algorithm to be used with the features, thus the evaluation is performed via exhaustive search. To compute the features, first a set of candidate feature points is selected, which are then purified by extracting and comparing up to

30 two-dimensional image patches of different width, centered around the candidate feature point. A candidate point is selected as feature point if most of the (up to 30) extracted image patches fulfill a similarity criterion. This is determined via k-means clustering, which assigns the extracted patches to a number of c classes. Similarity is calculated by computing low frequency discrete cosine transformation (DCT) coefficients which represent the actual similarity metric. The proposed method performs feature point selection for an average of 20 candidate points per second of audio, of which approximately 40% pass the similarity constraints and are used for fingerprint computation. The actual fingerprints are generated from a number of low frequency DCT coefficients of the extracted image patches, and are scaled and translated to result in vectors of zero mean and unit variance. According to the explanation given in the work, such a fingerprint should result in a vector of 143 floating point values. Fingerprint matching is done by nearest neighbor lookups, with distance defined as the angle of two fingerprint vectors. The work reports near perfect percentages of correct song association, though on a rather small database of only 250 songs.

In the master's thesis by Rotteker (2016), the method (Malekesmaeili and Ward, 2014) is implemented and investigated via experimentation. The identification experiments are performed on a database of 8.3 hours of audio content. It is concluded that the method seems to be prohibitively runtime demanding, such that the extraction of features, together with the stability analysis of time-chroma patches, requires the computation of over 6000 two-dimensional DCTs per second of audio. Altogether, answering a query of 30 seconds in length is reported to take roughly 3.85 minutes. While we assume that this is implemented as so-called "research-code", rather than being a highly optimized system, this report leads us to assume that the method of Malekesmaeili and Ward (2014) might not be directly applicable to large-scale use cases.

In publications that we mentioned here, and in Section 2.2, we can find solutions to some or most of our defined criteria, however scattered among different works. In our research we did not find a promising way to combine some of the promising algorithms to achieve robustness to *T*-, *P*-, and *TP*-modifications. Thus, there remain challenges to build automated systems that exhibit strong robustness and at the same time can be applied in the presence of large reference collections. The features proposed by Zhu et al. (2010) and Malekesmaeili and Ward (2014) are used in combination with exhaustive search, thus, the question on how to build scalable, efficient systems for the specific features is left unanswered. The feature dimensionality of 128 (as used by Zhu et al. (2010)) and 143 (as used by Malekesmaeili and Ward (2014)) implies relatively large reference databases, as for each feature, this number of floating-point values

(we assume single-precision floating point representation with a size of 32 bit) needs to be stored.

At this point in time, from the information given in the literature, we are not aware of a readily applicable or convincingly reliable system for the complex task of media monitoring, including DJ music events.

Chronologically, our initial work on this topic (Sonnleitner and Widmer, 2014) is to be placed here in our list of scale modification robust methods. We propose to group four-dimensional continuous hashes (i.e. real-valued hash representations, rather than using quantisation) from local maxima in the spectrogram. These hashes allow for a simple representation that is invariant to translation and non-isotropic[1] scaling, and enable us to build a high-performing scale modification robust method, evaluated on a dataset of roughly 1333 hours of audio. System performance is evaluated on queries that are scale-modified in the range of $\pm 30\%$. The proposed algorithm achieves near perfect overall performance within scale modification ranges of -10% to $+15\%$ for *TP*-modifications, and $\pm 20\%$ for *T*-modifications. The reported evaluation measures are accuracy and precision, and we further report the run-times of query processing. With the exception of one point, our proposed method meets all the criteria that we defined for robust systems: the shortcoming at this point in time was that we could not convincingly show its scalability to large databases, due to the runtime demands of our unoptimized implementation.

Six and Leman (2014) investigate the use of spectral peak triples for quantised hashes that enable quick search via table lookups, and achieve moderate robustness to any kind of scale modifications. The method can quantify the extents of scale modifications in query content. The system is evaluated on queries against a database of 30 000 full-length songs. The identification performance is highly sensitive to the extent of scale modification in query audio. For *P*-, and *T*-modifications of more than $\pm 2\%$ the accuracy drops below 80%, and to roughly 50% if queries are subjected to more than about 7% of scale modification. Queries that are *TP*-modified in the range of roughly $\pm 5\%$ can be detected with an accuracy of approximately 60%. On the test database, perfect identification specificity is reported. In Chapter 7, we compare our proposed system against this method for their applicability to DJ set monitoring.

Zhang et al. (2015) propose to use locality sensitive hashing (LSH) to overcome the exhaustive search of a previously described system (Zhu et al., 2010). The system is tested on roughly 169 hours of reference audio. The evaluation measure is accuracy. In terms of accuracy, the system is highly robust to *T*-changes in the range of $[-20\%; +40\%]$, but for slow audio (-30%) its performance drops to roughly 79%,

1 Non-isotropic scaling is also referred to as non-uniform-scaling and anisotropic-scaling. It means that different scale factors are applied to either dimension. An object that is non-isotropically scaled will change its shape. In our context, *P*-, *T*-, *TP*–modifications correspond to non-isotropic scaling.

while it has an accuracy of roughly 97% for a tempo change of -20%. The sytem accuracy is quite sensitive to pitch changes, where accuracies of at most 90% are reached for pitch changes of more than $\pm 2\%$. The robustness to *TP*-changes is high: here, the accuracy is between roughly 92% and 100% for *TP*-modifications in the range of $[-20\%; +30\%]$, but the accuracy for *TP*-modifications drops to roughly 68% in the presence of -30% of *TP*-alteration.

In our work (Sonnleitner and Widmer, 2016) we build a complete system by refining the previously published method (Sonnleitner and Widmer, 2014). We considerably reduce the run times of query processing, and achieve higher identification performance. We evaluate this state-of-the-art method in detail and give various parameter studies. Identification performance is reported in terms of accuracy, precision and specificity, on a free and reproducible reference database of 100,000 tracks. This work is described in detail in Chapter 4 of this thesis.

To analyse the capabilities of our method in its target domain we investigate the applicability of spectral-peak-based methods to the task of DJ mix monitoring (Sonnleitner and Widmer, 2016), where we compare the latest modified version of our method to two reference methods: Audfprint, which is a free implementation of the algorithm by Wang (2003), and the method described by Six and Leman (2014). The comparisons are performed on two datasets of DJ mixes, and we then show an evaluation of our method on a large database that consists of roughly 29,500 hours of creative-commons licensed audio content. The database is created from approximately 430,000 tracks, and is extended with the DJ datasets to perform the evaluation. This setting tries to reflect a more realistic scenario in terms of database size, in order to show the applicability of our method to media monitoring. A detailed description is given in Chapter 7. The work contains a free dataset for research purposes to evaluate system performance on DJ mixes.

2.4 THE QUAD-DESCRIPTOR

We kept the most important related work for the end of this chapter: a method from the field of astronomy, described by Lang et al. (2010). It is the foundation that our contributions are built upon. Lang et al. (2010) discover a way to solve a generalisation of the so-called "lost-in-space problem". The task is to identify stars only from the pixel values of input pictures, without knowledge of position and orientation of the camera. To do so, the location of stars is extracted from the input image to obtain a list of two-dimensional points. These are then grouped to a set of n-tuples, which are transformed to a local coordinate system to form the actual feature combinations that we call hashes (or, in our context, fingerprints). The work investigates

3-tuples, 4-tuples and 5-tuples, and convincingly argues for the use of 4-tuples being most advantageous, hence, the name "quads", or "quad-descriptor". The work elaborates on a simple, efficiently computable geometric hash representation of the tuples, which makes the hashes invariant to rotation, translation and to isotropic angle-preserving scaling. These hashes are then associated with their point locations in the input image, and compared against massive databases and sky catalogues for which the features were computed beforehand. For us, a crucial point in their work is the application of *range search* techniques to search for matching hashes. It enables highly robust retrieval, and further, the use of quads with their dimensionality of 4 still is feasible for efficient application of search trees. The second crucial point is the idea of verifying a match hypothesis. Simple and effective, it states that if stars are found near a matching feature in the reference, these should also be present in the input image. We make use of this verification process in our method, to build a fingerprinting system that is highly precise and specific.

For our own fingerprinting method we adapt the hashes, and modify the hashing principle to create invariances to translation and non-isotropic scaling (where the dimensions can be modified with different scaling factors), and dismiss the property of rotational invariance. This way we achieve a method for doing translation- and scale-invariant (but not rotation-invariant) two-dimensional point-cloud-matching. In the context of their work, one could say we obtain an audio signal, compute the spectrogram, and treat it as a picture of the night sky: local maxima will serve as stars and their locations are used to compute the hashes. Our reference collection of audio pieces would be the sky-catalogue to compare the features to. This is the concept of our proposed audio fingerprinting method, which we will refer to as simply "Qfp" – quad-based audio fingerprinter.

3

QUAD-BASED AUDIO FINGERPRINTING: CONCEPT

In previous chapters we introduced the topic and the literature, and from here on we will focus on our proposed method and its implementation. This chapter introduces our initial work on the topic, and while parts of it are superseded by findings that we describe in later chapters, we include this work for completeness. In this thesis we want to describe and document our research on the topic, and our starting point is an important part thereof. That said, readers who are interested in the technical details of the refined version of our method, are invited to skip this chapter and to directly proceed with Chapter 4. To support this, we decided to describe the method in Chapter 4 in its entirety, and do not require the reader to be familiar with this present chapter.

This present chapter is based on the following publication:

- R. Sonnleitner and G. Widmer (2014). "Quad-Based Audio Fingerprinting Robust to Time and Frequency Scaling". In: *Proceedings of the International Conference on Digital Audio Effects (DAFX 2014)*. Erlangen, Germany, pp. 173–180

The content in this description is heavily based on the publication, and contains identical passages. For prototyping, rapid experimentation and evaluation the described system is implemented in the Python programming language, and evaluated on a reference collection consisting of 20,000 tracks.

The paper was well accepted and was granted the "Best Student Paper Award". The evaluation shows state-of-the art results, and the work proposes what seem to be viable solutions for achieving strong robustness. However, at that point in time we could not answer the important question whether the system can scale to a more realistic, far larger number of reference audio pieces. To do this we were still missing results on larger datasets, a fast implementation of the system core, and solutions to challenges that came up during the design and development process. We try to answer these questions in subsequent chapters.

The following sections in this chapter explain the system in detail, where we organise the content as follows: Section 3.2 gives a brief overview of the main points of our proposed method (we simply call it "Qfp"), in order to set the context for the precise method description, which comes in two parts: Section 3.3 describes the process of constructing audio fingerprints – the extraction of features from audio that allow for precise identification. We describe how to obtain

hash representations from these features, and how to store these in data structures for efficient retrieval. The actual identification method, i.e. the process of matching query audio with reference data by using the extracted features and their hash representations, is then explained in Section 3.4. Section 3.5 systematically evaluates the performance of our method in two experiments. The first experiment considers song identification on a database consisting of one thousand full length songs of different musical genres. For the second experiment we extend the database to 20,000 full length songs to investigate the scalability of the method in terms of run times and identification performance.

3.1 INTRODUCTION

We propose an efficient audio fingerprinting method that meets the discussed robustness requirements. It is not only robust to noise and audio quality degradation, but also to large amounts of speed, tempo or frequency and pitch scaling. In addition, it can accurately determine the scaling factors of applied time/frequency distortions. The key technique that makes this possible was found by researchers working on blind astrometry, who use a simple and fast geometric hashing approach to solve a generalization of the "lost in space" problem (Lang et al., 2010) (see Section 2.4). We adapt the algorithm to our needs such that we achieve a representations of fingerprints that are invariant to translation and scaling, and thereby overcome the inherent robustness limitations of systems that depend on equal relative distances of reference and query features to find matches, such as the seminal algorithm by Wang (2003). More precisely, our algorithm uses a compact four-dimensional continuous hash representation of quadruples of points, henceforth referred to as "quads". The quad-descriptor (Lang et al., 2010) has also been adopted to the field of computer vision, for the task of accurate alignment of video streams (Evangelidis and Bauckhage, 2011; Evangelidis and Bauckhage, 2013).

The system we propose can be used for DJ set monitoring and original track identification, audio copy detection, audio alignment, as well as other tasks that demand robustness to certain levels of noise and scale changes.

From the related and previous work, the method described by Malekesmaeili and Ward (2014) will act as our reference method. The work shows extremely high robustness and performance results for a large range of tempo and speed modifications (though based on experiments with a rather small reference database – see Section 3.5 below).

3.2 METHOD OVERVIEW

The basic idea of our proposed method is to extract spectral peaks from the two-dimensional time-frequency representation of reference audio material, then group quadruples of peaks into quads, and create a compact translation- and scale-invariant hash for each quad such that a single hash is a point in a four-dimensional continuous vector space. Quads and their corresponding hashes are stored in different data structures, i.e. quads are appended to a global index, and an inverse index is created to assign the corresponding audio file ID to its quad indices. The continuous hashes are stored, together with the index of their quad, in a spatial data structure, such that the index that is associated with the hash corresponds to the index of the quad that forms the hash.

For querying we extract quads and their hashes from the query audio excerpt. For each query hash we perform a range search in the spatial data structure and collect the indices of search results, which in turn give the indices of matching reference quads in the global index. The time and frequency scaling factor can be determined by comparing a query quad to its matching reference quad. To predict the match file ID for a query snippet, we compare relative distances of matches in the reference and query time series.

3.3 FEATURE EXTRACTION

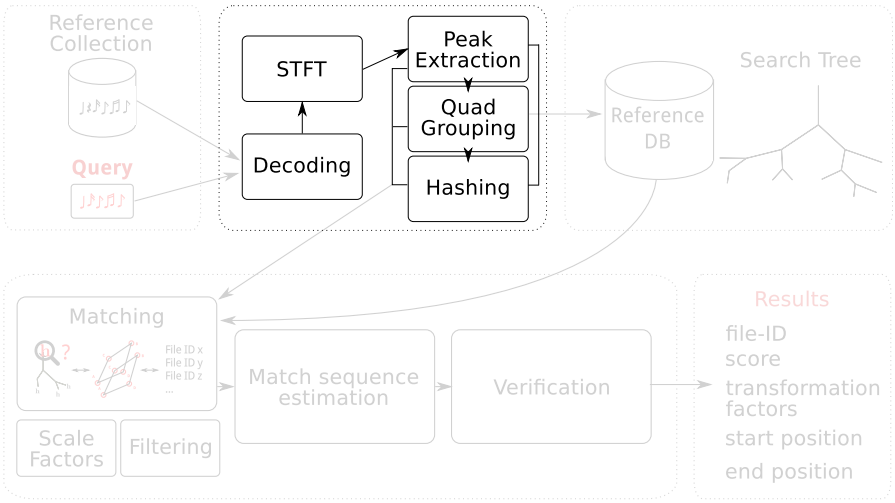


Figure 3.1: Feature extraction components

In this section we describe the extraction of audio features to be used for audio identification, how to obtain hashable representations from these features, and how to finally store these for efficient retrieval. The same feature extraction process is applied to the *reference recordings*

that are used to build the fingerprint database, and the *query audio* that is to be identified in the recognition phase.

To begin with, all audio files are downmixed to one-channel monaural representations and processed with a sampling rate of 16 kHz. We compute the STFT magnitude spectrogram using a Hann-window of size 1024 samples (64ms) and a hopsize of 128 samples (8ms), discarding the phases.

3.3.1 *Constructing Quads*

The fingerprinting algorithm works on translation- and scale-invariant hashes of combinations of spectral peaks. Spectral peaks are local maxima in an STFT magnitude spectrogram, and identified by their coordinates in the spectrogram. Since the notion of a peak P as a point in the two-dimensional spectrogram space will be used extensively in the following, let us formally introduce the notation:

$$P = (P_x, P_y) \tag{3.1}$$

where P_x is the peak's time position (STFT frame index), and P_y is the peak's frequency (index of STFT frequency bin).

The extraction of spectral peaks is implemented via a pair of two-dimensional filters, a max filter and a min filter, where the neighbourhood size is given by the filter window size.

We use the max filter to find the coordinates of spectral peak candidates in the spectrogram, and the min filter with a smaller filter window size to reject peaks that were extracted from uniform regions in the spectrogram, e.g., digital silence. We first explain the algorithms, and then give information on the actual parameterization in Section 3.3.4.

In the following we explain how quads are created from spectral peaks, and how compact hash values are computed from quads.

To create translation- and scale-invariant hashes from spectral peaks, we first have to group peaks into *quads* (Lang et al., 2010). A quad consists of four spectral peaks A, B, C, D, where we define A to be the root point of the quad, which is the peak with the smallest frame index (i.e. the first of the four peaks in time) and B is the most distant point in time from A (thus C, D lie somewhere between the STFT frames of A, B). The quad is *valid* if $B > A$ and C, D lie within the axis-parallel rectangle defined by A, B:

$$A_x < B_x \tag{3.2}$$

$$A_y < B_y \tag{3.3}$$

$$A_x < C_x, D_x \leq B_x \tag{3.4}$$

$$A_y < C_y, D_y \leq B_y \tag{3.5}$$

At the top level, the quad grouping process proceeds through an audio file advancing in time, trying each spectral peak as a potential

root point A of a set of quads, with the goal of creating up to a number of q quads for each peak.

For a given root point A, the process of constructing up to q quads by selecting appropriate sets of B, C, D points is as follows. [1]

We construct a region of width r, spanning r STFT frames, such that the region is centered c STFT-frames from A and A is outside of the region (earlier in time, i.e., the region is located to the right of A), as shown in Figure 3.2. We then sort the peaks that are contained in the region, by time. We let $t = 0$ and pick the first m peaks $p_t, p_{t+1}, \dots p_{t+m-1}$ in the region and try all $\binom{m}{3}$ combinations of 3 peak indices in order to construct a valid quad with root point A and the points from the current combination. If a valid quad can be constructed we append it to a list of quads and proceed until q quads are created.

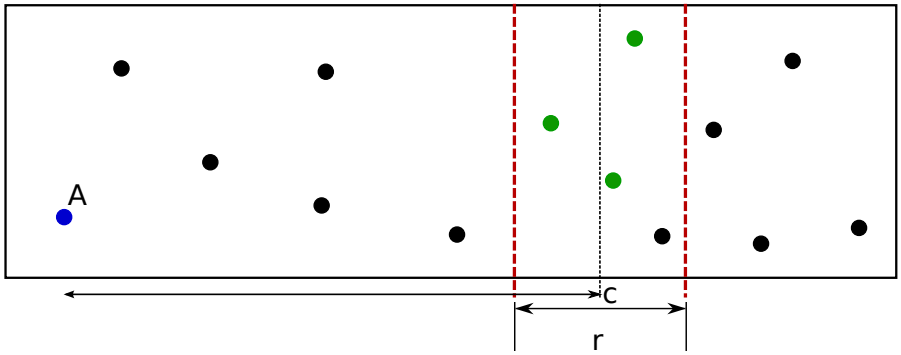
If no valid quad could be constructed, we increase t by one and try again until there are no more peaks in the region.

The total number of resulting valid quads for a given root point A depends not only on the parameter values, but is fundamentally dependent on the specific layout of spectral peaks, and thus on the signal itself. As already mentioned, for creating the reference database we want to create a small number of quads. We therefore choose a small n and a region of small width r. For queries we create an extensive set of quads by choosing a larger n, $r_{\text{query}} \gg r_{\text{ref}}$, and $q_{\text{query}} \gg q_{\text{ref}}$. The center c is the same in both cases.

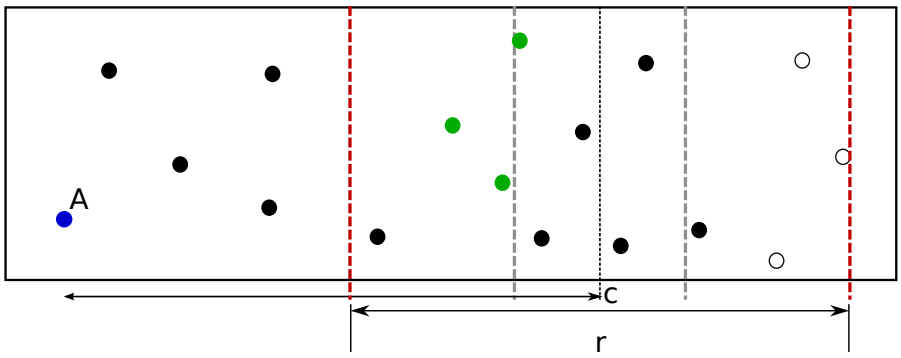
The reason for different parameterization for query quad construction is as follows: When the time scale of a query audio is modified, this affects not only the density of peaks in the given audio snippet, but also their relative positions. An example is given in Figure 3.2, which shows the grouping for a quad for a given root point A. In 3.2a a reference quad is created for a region of width r that is centered c frames from A. The analogous example for grouping a query quad for the same audio, but increased in tempo, or decreased in tempo, is given in 3.2b and 3.2c, respectively. We see that the green points, which are points B, C, D for the reference quad, may happen to move outside of the grouping region of width r if the time scale of the audio is modified. By choosing a larger region width r and a larger number q of quads that may be created for a root point A, we can ensure to obtain a quad that corresponds to the reference quad.

Note that when we consider audio queries of a fixed, limited duration d (e.g., 15 seconds), there is an important difference between increased speed/tempo and decreased speed/tempo. Increasing the tempo of the query audio excerpt relative to the reference leads to

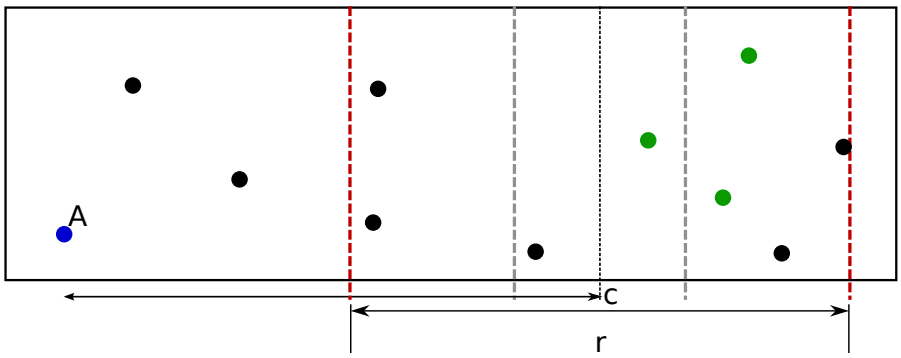
1 We will parametrize this process differently, depending on whether we compute quads for the reference database, or for a piece of query audio. For reference database creation, we choose parameters in such a way that we only create a small number of reference quads to keep the resulting reference database as small as possible. For a query snippet, we will choose parameters to create a large amount of quads. The explanation for this will be given later in this section.



(a) Reference quad grouping



(b) Query quad grouping. Query audio was increased in tempo. New peaks are shown in white facecolor.



(c) Query quad grouping. Query audio was decreased in tempo.

Figure 3.2: *Reference quad grouping (3.2a) and query quad grouping with increased tempo (3.2b), and decreased tempo (3.2c).*

a higher density of relevant audio content; all the content that was used during the phase of reference quad creation is also present when creating the quads for the query. However, decreasing the tempo of the query, i.e., stretching the time scale, may cause some of the relevant spectral peaks to fall out of the 15 second excerpt (i.e. not be part of the query any more), so some important quads do not emerge in the query. This problem arises when tempo or speed are decreased by large amounts.

This difference in increasing vs. decreasing the time scale is actually reflected in the evaluation results (see Section 3.5). To summarize, if the same parameters are used for both reference and query quad grouping, and the time scale changes, it is very likely that no matching quads will be found in subsequent queries.

3.3.2 *From Quads to Translation- and Scale-invariant Hashes*

We now have created quads from spectral peaks in audio, but these quads are not the actual summarizing representation that we later use to find match candidates between a query audio and the reference database. To achieve a suited representation that also is quickly retrievable, we compute *translation-* and *scale-invariant* hashes from the quads. For a given quad (A, B, C, D), the constellation of spectral peaks is translated to the origin and normalized to the unit square, resulting in the four points A', B', C', D' such that $A' = (0, 0)$ and $B' = (1, 1)$, as shown in Figure 4.3 (Section 4.3.3, page 60). The actual continuous hash of the quad is now given by C', D', and is stored as a four-dimensional point (C'_x, C'_y, D'_x, D'_y) in a spatial data structure. Essentially, C', D' are the relative distances of C, D to A, B in time and frequency, respectively. Thus, the hash C', D' is not only

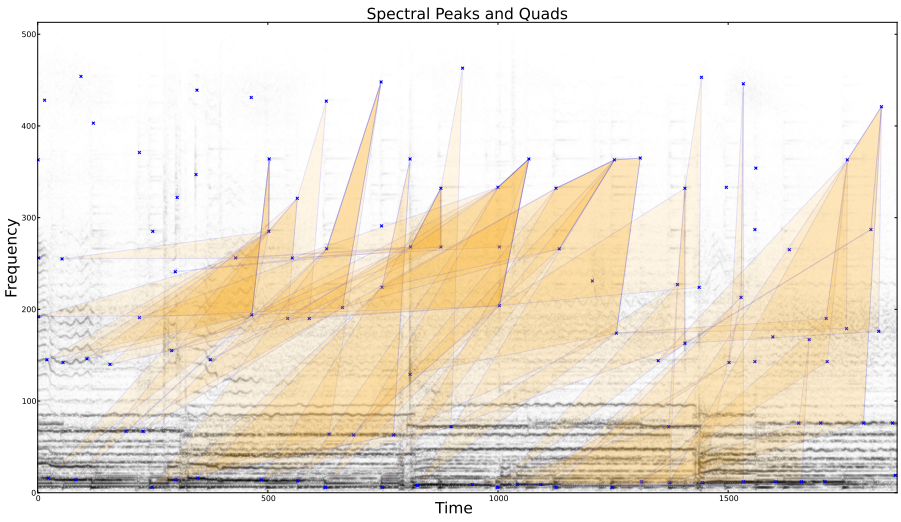


Figure 3.3: *Extracted spectral peaks and grouped quads on a* 15 *seconds excerpt of "Radiohead - Exit Music (For a Film)".*

translation-invariant (A' is always $(0, 0)$), but also scale-invariant. A feature extraction example showing spectral peaks and resulting quads is shown in Figure 3.3

3.3.3 *Fingerprints: Storing Hashes for Efficient Recognition*

This section is about the database component of the fingerprinting system (see Figure 3.4). Here, we explain how the individual results of the feature extraction process are stored by the reference database.

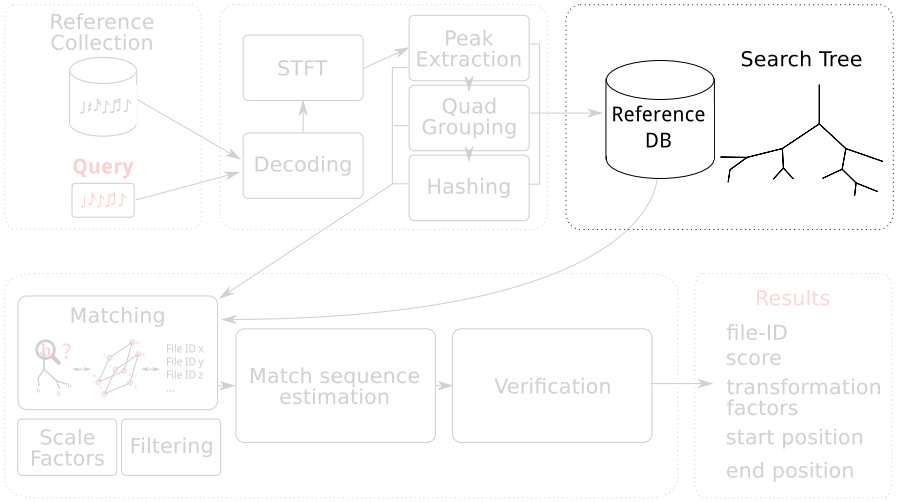


Figure 3.4: Database components

Once peaks, quads and their hashes are computed, we store these in data structures that allow for efficient selection of match candidates and subsequent verification of match hypotheses from query audio. The reference data consist of four data structures:

- quadDB: A file that contains all reference quads (the original, *unnormalized* ones).
- fidindex: An index file that stores file-id, quad index range (into quadDB) and filename for each reference audio file.
- reftree: A spatial data structure containing all reference quad hashes.
- refpeaktrees: Two dimensional search trees for the spectral peaks that are extracted from reference audio files.

The quadDB is a binary file that stores the sequence of quads for all reference files, and the fidindex is an index file which maps each reference file to a unique file-id and also stores the index range (i.e. startindex, number of quads in quadDB) for the sequence of quads that was extracted from the reference files. For the spatial data structure (reftree) we use an R-Tree (Guttman, 1984) with an R* split-heuristic that stores all quad hashes, together with their positional index in the quadDB. R-Rrees are tree data structures that are used in spatial databases to support querying spatial data. R-Trees can be considered as a special instance of so-called bounding volume hierarchies, where

the type of bounding volume is a hyper-rectangle. The difference to a bounding volume hierarchy with rectangular bounding volumes is that R-Trees are usually balanced trees, which are constructed bottom-up rather than top-down. In general, R-Trees are well suited for large out-of-memory databases.

In addition, the R-Tree is well suited for large out-of-memory databases.

The refpeaktrees are used for the verification of match candidates, which will be explained later.

3.3.4 *Chosen Parameter Values*

As mentioned in Section 3.3, the parameters for the STFT are given with a sampling rate of 16kHz, a Hann-window of size 1024 samples (64ms) and a hopsize of 128 samples (8ms). The specific set of parameter values that we chose for our implementation and that are used in the evaluation in Section 3.5, is as follows: The extraction of spectral peaks is performed with a max-filter width of 91 STFT-frames (728ms), and a filter height of 65 frequency bins (1015.625Hz). The min-filter, used to reject maxima that resulted from uniform magnitude regions, has a width of 3 STFT-frames (24ms) and a height of 3 frequency bins (46.875Hz). For reference quad grouping we choose the center of the grouping window c to be four seconds from each root point A. The width r of the region window for reference quad extraction is two seconds. We group $q = 2$ reference quads for each root point A along with a group size of $n = 5$. This results in an average number of roughly 8.7 reference quads per second of audio. For query quad extraction we choose the same c of four seconds, and a large grouping window width r that spans 7.9 seconds. A number of up to $q = 500$ query quads are extracted from a group size of $n = 8$.

3.4 RECOGNITION ALGORITHM

The method for identifying the correct reference recording, given a query audio excerpt, consists of several stages (see Figure 3.5): the selection of match candidates, a filtering stage in which we try to discard false positive candidates, and a verification step adapted from the findings in (Lang et al., 2010). After the verification stage we efficiently estimate a match sequence with the histogram binning algorithm proposed by Wang (2003). In the following the selection of match candidates is explained.

3.4.1 *Match Candidate Selection and Filtering*

For each quad hash that was extracted from a query snippet, a range-search in the reftree is performed. This lookup returns a set of raw

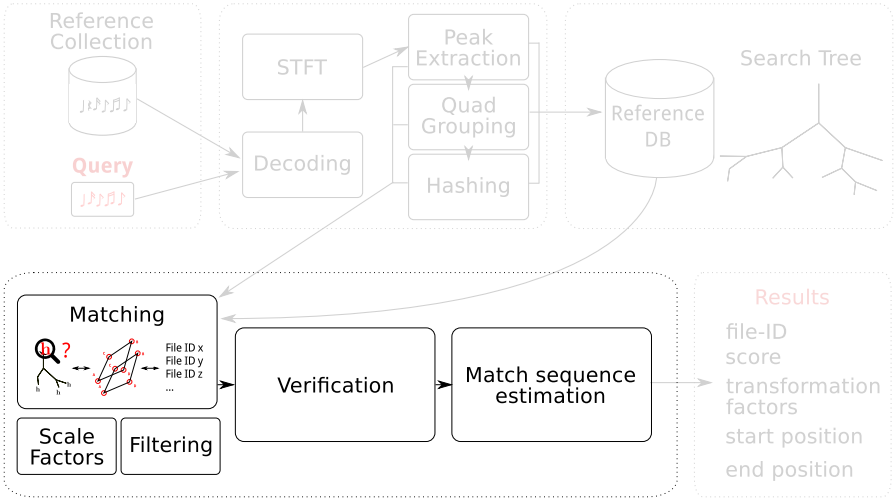


Figure 3.5: Subcomponents of the search module

match candidate indices: the indices of those quads in the quadDB whose quad-hashes are similar (i.e. if C', D' are identical up to epsilon: $C_x'^{query} - \epsilon \leq C_x'^{ref} \leq C_x'^{query} + \epsilon$ etc.) to the query quad-hashes. We call this the set of raw candidates, as it will most likely be a mixture of true positives and a (large) number of false positive matches. The raw candidates are used to obtain estimates of the time/frequency scale modification of the query audio, by looking at the spatial extents of the original (non-normalized) quads corresponding to the query (q) and reference (r) hash, giving us the scaling factors for time and frequency:

$$s_{time} = (B_x^q - A_x^q)/(B_x^r - A_x^r) \tag{3.6}$$

$$s_{freq} = (B_y^q - A_y^q)/(B_y^r - A_y^r) \tag{3.7}$$

It makes sense to parametrize the system with scale tolerance bounds as, depending on the application, one might not be interested in trying to identify audio that is played at, e.g., half the speed or double the tempo, or has undergone extreme pitch-shifting modifications. Such constrained tolerances considerably speed up the subsequent hypothesis testing by rejecting raw match candidates that lie outside the specified bounds.

Instead of directly starting with hypothesis tests on the raw candidate set we first apply filters in order to clean up the match candidates. This filtering process aims at discarding false positive matches while keeping a large number of true positives. In addition to the previously mentioned scale tolerances, we perform a spectral coherence check similar to the spatio-temporal coherence check described by Evangelidis and Bauckhage (2013). Here we reject match candidate quads whose root point A is far away in the frequency domain compared to root point A of the query quad.

We now consult the fidindex to sort the remaining match candidates by file-id, and enter the verification step (Section 3.4.2) – those candidates that pass the following step are considered true matches and are passed to the match-sequence estimation.

3.4.2 *Match Verification and Sequence Estimation*

Match verification is performed once all match candidates for all query quads are collected and filtered as described above. Most likely, the remaining match candidates correspond to a large number of file-ids that are referenced in the database. Since our goal is to identify the correct file-id, we perform this stage of match candidate verification on a per-file-id basis. To do this we consult the fidindex file (cf. Section 3.3.3) and look up the file-ids for all match candidates, and sort the match candidates by file-id.

Verification is based on the insight that spectral peaks that are nearby a reference quad in the reference audio, should also be present in the query audio (Lang et al., 2010). Naturally, depending on the audio compression, the amount of noise or other distortions, there might be a larger or smaller number of nearby peaks in the query. We define the nearby peaks as the set of N peaks closest to the match candidate's root point A (for some fixed N), and retrieve those by performing a k-nearest-neighbor search in the refpeaktrees (cf. Section 3.3.3) for the given file-id. We define a threshold t_{min}, the minimal number of nearby peaks that have to be present in the query in order to consider the candidate an actual match. Note that in order to find relevant nearby peaks in the query, we have to align the query- and reference-peaks by transforming the query peak locations according to the previously estimated time/frequency scale (cf. Section 3.4.1). The candidates that pass the verification step are considered true matches, and they are annotated with the number $v \leq N$ of correctly aligned spectral peaks, and the scale transformation estimates. This number v will be used for an optimization described below.

After match candidates for a given file-id are verified, we try to find a sequence of matches for this file-id by processing the matches with a histogram method similar to the one used in the Shazam algorithm (Wang, 2003), with the difference that the query time (the time value of root point A of each query quad in the sequence) is scaled according to the estimated time scale factor. Finally, the file-id for the largest histogram bin (the longest match sequence) is returned, together with the match position that is given by the minimal time value of the points in the histogram bin. We now know the reference file that identifies the query audio, the position of the query audio in the reference track, and the associated time/frequency scale modification estimates. Note that the reported scale transformation estimates are expected to be quite accurate, because with these estimates, for each

"surviving" match candidate at least t_{min} nearby spectral query peaks could be correctly aligned to corresponding reference peaks during the verification phase. A lookup in the fidindex now gives us the filename of the reference audio and optionally any kind of previously associated meta-data.

To speed up the verification process, we define a threshold for the number of correctly aligned nearby peaks $t_{\text{v}} > t_{\text{min}}$. When the v value of a match reaches or exceeds this threshold, we allow a so-called "early exit" for this file-id. Once all match candidates of an early exit file-id are verified, we directly enter the match sequence estimation for this file-id, without subsequent verification of any other file-id.

3.4.3 *Runtime and Data Size Considerations*

Our system operates on a number of data structures (cf. Section 3.3.3) that together constitute what we call the *reference database*; the largest components are the reftree and the refpeak trees.

The quadDB linearly stores binary records of quads. A quad consists of four two-dimensional discrete points (coordinates in the STFT spectrogram) and can be represented and stored as $8 * 32$bit integers, which amounts to 32 byte per quad. It is not necessary to keep this file in-memory, as the proposed method is designed to operate on big out-of-memory reference data.

There exists exactly one quad hash per quad. A quad hash is a four-dimensional point that is stored as an array of four float32 values by the reftree. The actual number of quads in the quadDB depends on the filter size parameters of the spectral peak extraction and the quad grouping parameters. For an example reference database consisting of 20,000 full length songs we choose the parameters such that we create an average of approximately 8.74 quads per second of audio, with a median of roughly 8.68 and a standard deviation of $\sigma \approx 1.19$. The histogram of the number of quads per second is shown in Figure 3.6. This specific database consists of roughly $4.29 * 10^7$ quads ($\approx$ 1.3GB). The reftree, a four-dimensional R-Tree, consumes approximately 4.8GB. To speed up the verification process we also store trees of the two-dimensional spectral peaks for each file-id, which consume roughly 3GB for 20,000 songs. We currently store the fidindex file as text, along with some meta-data. In this example the size of the fidindex amounts to 2.3MB.

Depending on application scenarios and hardware constraints, it is possible to trade runtime for storage space and vice versa. If minimal space consumption is of priority, one can pack the binary quad records of the quadDB to 16 bytes by exploiting the limited number of STFT frequency bins (i.e. 512), and storing the time values of points B, C, D as offsets from point A. This saves 50% per quad, reducing the size of the quadDB file to roughly 650MB.

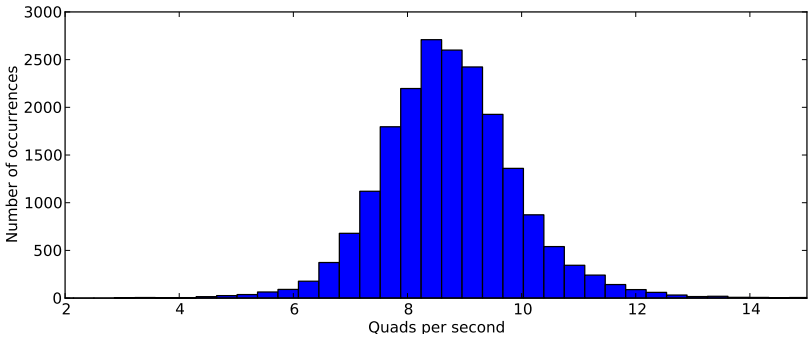


Figure 3.6: *Histogram of the average number of quads per second for all files in a reference database of* 20,000 *songs.*

Regarding the runtime, using our unoptimized pure Python implementation of the method, feature extraction and quad creation for the database of 20,000 songs took approximately 24*h* utilizing seven out of eight logical cores of an Intel Core i7 860 (2.8GHz) Processor.

The runtime for a query is made up of audio decoding, feature extraction, querying the database and filtering the results, match candidate verification and match sequence binning. For a 15 seconds long query against a reference database of 1000 songs, this takes approximately 4 to 12 seconds. Here, half of the time is taken by the preprocessing (decoding, quad extraction).

Querying a larger database of 20,000 songs takes considerably (though, of course, not proportionally) longer. The main reason is the higher number of match candidates that have to be processed. The same query excerpt as used above is processed in approximately 14 to 60 seconds. Here, at least half of the time is consumed by the reftree range queries. Again, this is based on an unoptimized, experimental Python implementation; there is ample room for improvement. Section 3.5.3 gives more detail.

3.5 EVALUATION

We systematically evaluate the performance of the system for different speed, tempo and noise modifications of 15 seconds query audio snippets. The reference database for the first experiment is constructed from 1000 full length songs in "mp3" format.

To create test queries, we randomly choose 100 reference songs and subject these to different speed, tempo, and noise level modifications. We then randomly select a starting position for each selected song, and cut out 15 seconds from the audio, such that we end up with 100 audio queries of 15 seconds, each. The evaluation considers speed and tempo ranges from 70% to 130% in steps of 5%. To evaluate the noise robustness of our system we mix each query snippet with white noise to create noisy audio in SNR level ranges from 0 dB to +50

dB in steps of 5 dB. Furthermore, we create all query audio snippets from .mp3 encoded data, and encode the modified snippets in the Ogg Vorbis format ("Ogg Vorbis" n.d.), using the default compression rate (cr = 3). We do this to show that the system is robust to effects that result from a different lossy audio encoding. All modifications are realized with the free SoX audio toolkit ("SoX - Sound eXchange" n.d.).

3.5.1 *Detailed Results on the Small Reference-DB (*1000 *Songs)*

Each data point in the visualisation shows one of the aforementioned quality measures for 100 queries. The overall system performance for speed, tempo, and SNR changes is shown in Figure 3.7. For this experiment a total of 5900 queries of length 15 seconds were run against the database consisting of thousand songs.

Figures 3.8 and 3.9 show the performance for the tested SNR levels for speed and tempo modifications of 95% and 105%.

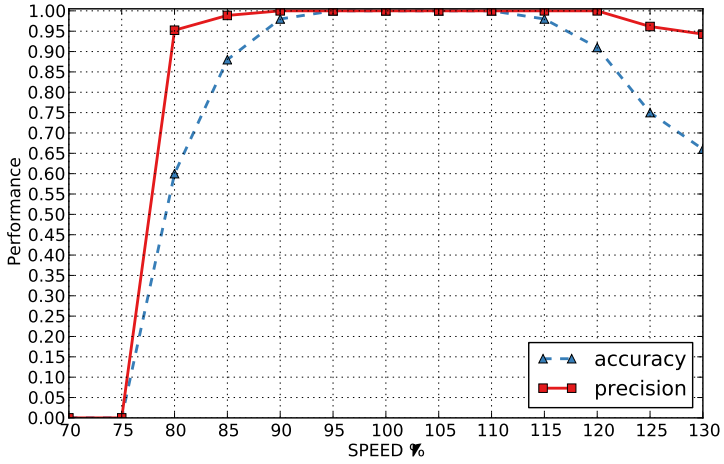
Concerning the noise robustness of the proposed method, the results show that a stable performance of $> 95\%$ for the tested quality measures is achieved for SNR levels down to $+15$dB. According to these results the proposed quad-based hashes seem to be sufficiently robust for queries of various noise levels that may be encountered in real application scenarios.

3.5.2 *Extending the Reference-DB to* 20,000 *Songs*

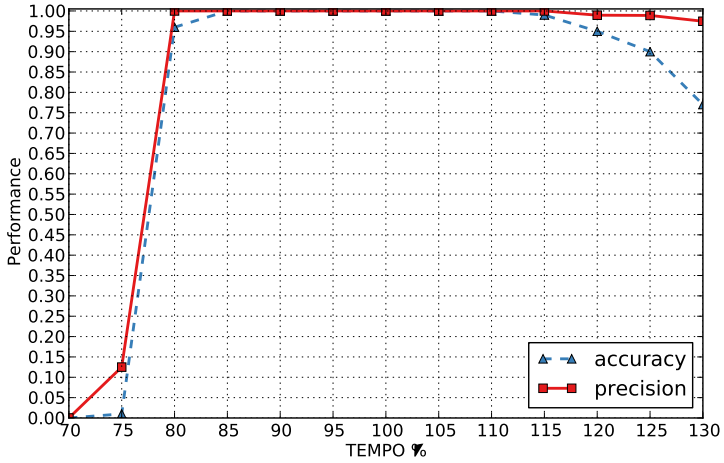
In the previous experiment on a database of thousand songs we reach a very high precision. To further investigate the precision of our proposed algorithm we extend the reference database to 20,000 songs, and query the same audio excerpts that we created for the previous experiment, with the same modifications, against this large database. Figure 3.10 shows that the performance of our approach does not degrade even if there are 20 times as many songs in the reference. Note that we parametrized our system to discard match candidates if their transformation estimates are outside the scale tolerance bounds of $\pm 32\%$ for either frequency and time scale. The performance is comparable to that of the first experiment, resulting in more false positives only for the larger speed modifications. For tempo modifications the system gives basically the same performance as in the first experiment.

3.5.3 *Runtimes*

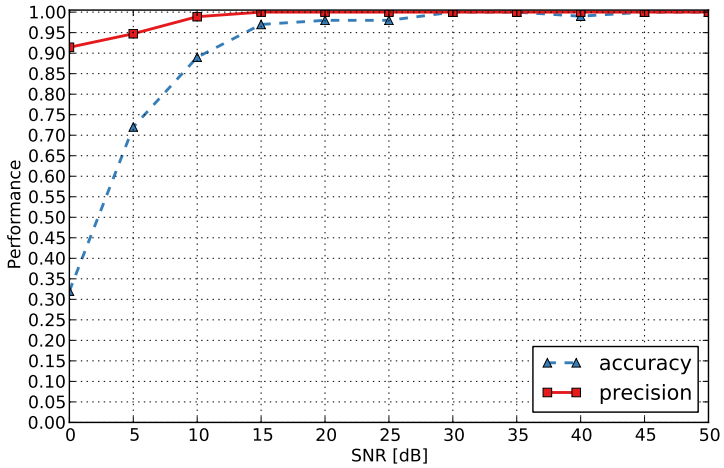
In Table 3.1 we give information about the runtimes observed in the two above experiments. We randomly pick one of the generated audio query excerpts, and compare the query runtime for the small and the large databases for different scale modifications. The increased runtime



(a) Speed variations

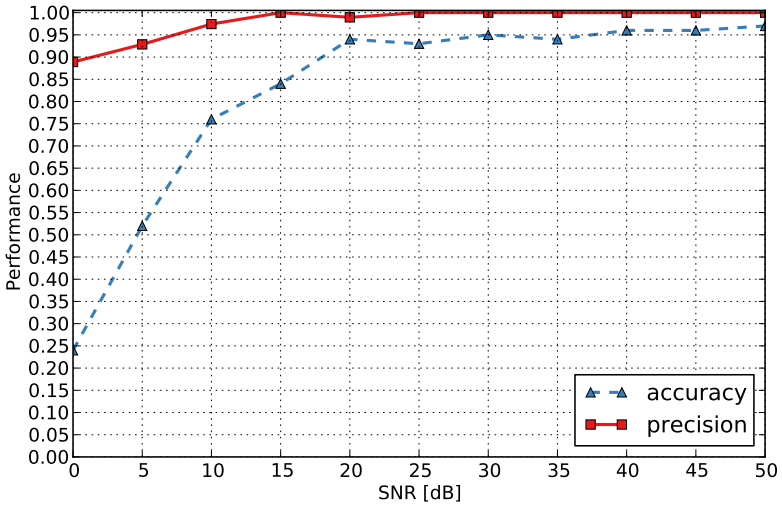


(b) Tempo variations

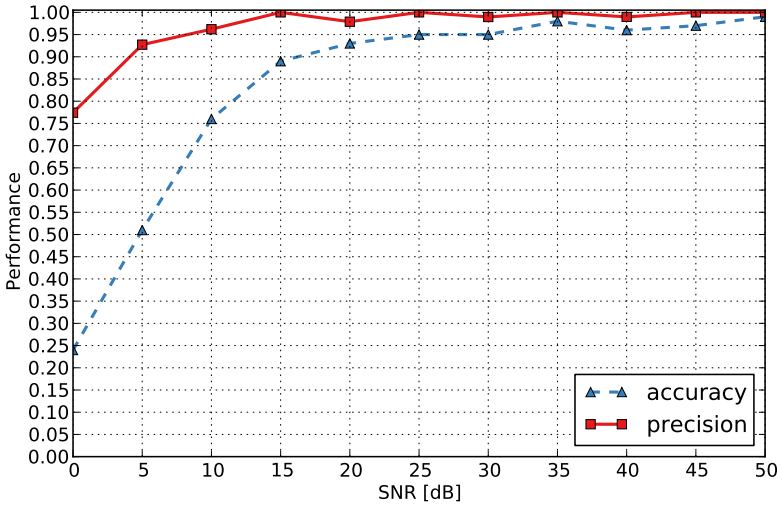


(c) SNR variations

Figure 3.7: *Precision and accuracy for speed (3.7a), tempo (3.7b) and SNR (3.7c) modifications, on a database of* 1000 *songs.*

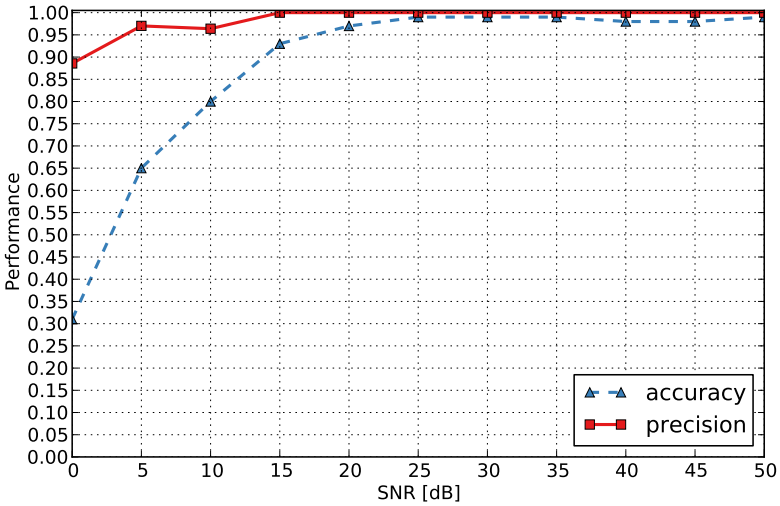


(a) SNR variations for speed 95%

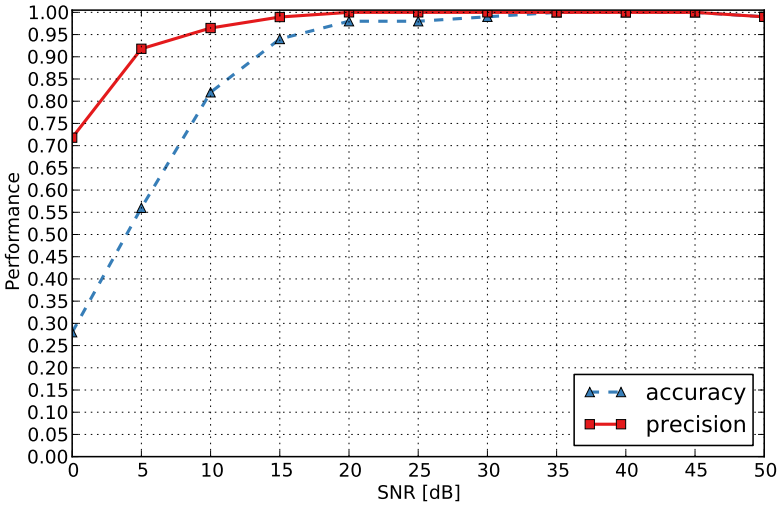


(b) SNR variations for speed 105%

Figure 3.8: *SNR variations for* 95% *and* 105% *speed on a database of* 1000 *songs.*

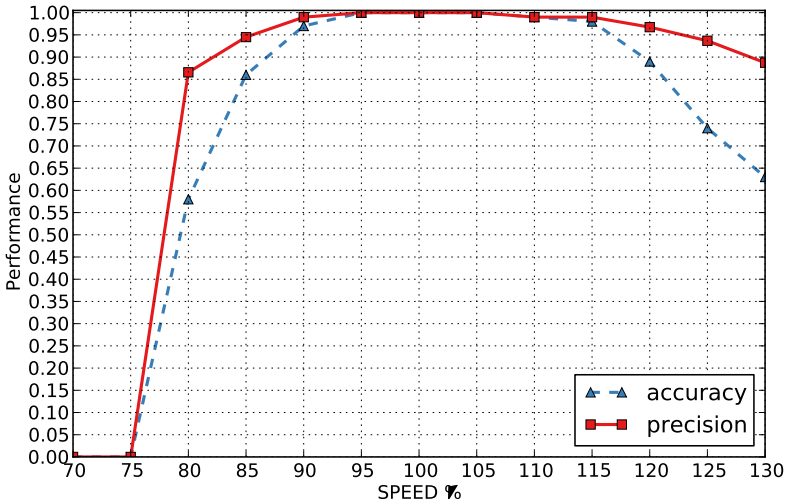


(a) SNR variations for tempo 95%

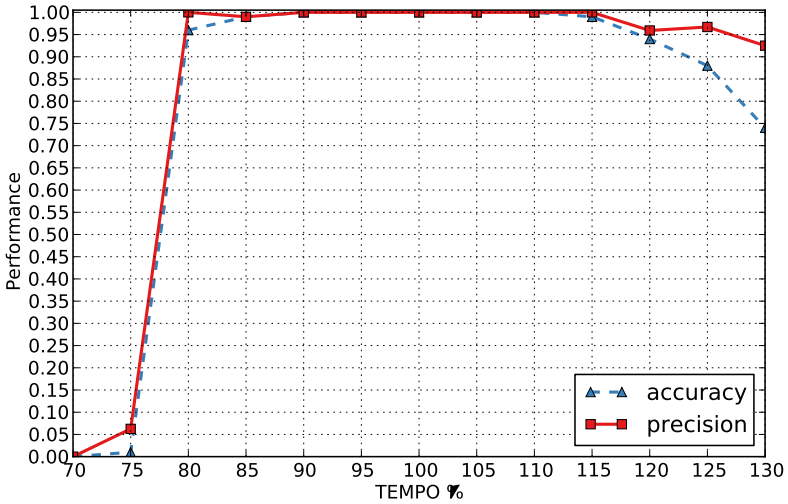


(b) SNR variations for tempo 105%

Figure 3.9: *SNR variations for* 95% *and* 105% *tempo on a database of* 1000 *songs.*



(a) Speed variations



(b) Tempo variations

Figure 3.10: *Precision and accuracy for speed (3.10a) and tempo modifications (3.10b) on a database of* 20,000 *songs.*

	db 1000 songs				db 20,000 songs			
Speed	tot.	tree	feat.	match	tot.	tree	feat.	match
130%	12	1	6	5	60	31	7	22
110%	11	2	6	3	52	28	6	18
100%	9	1	6	2	35	20	6	9
90%	9	1	5	2	37	22	5	10
70%	4	0	3	1	14	7	3	1

Table 3.1: Query runtimes in seconds. "tot" is the total time, "tree" is the time taken by tree intersection, "feat." is the feature extraction time for spectral peaks and quad grouping and "match" is the matching and verification time.

for faster speed and tempo is a result of the higher number of quads in the audio excerpt, for which a larger number of tree-intersections and raw match candidates have to be processed.

3.5.4 *Comparison with a Reference Method*

In this section, we compare our proposed system to the method described by Malekesmaeili and Ward (2014). While it is not possible to directly compare our results to the reference method (because we do not have access to their test data), from the published figures it seems fair to say that in terms of recognition accuracy and robustness, both approaches seem comparable, and both are at the upper end of what can be expected of an audio identification algorithm – for tempo and pitch distortion ranges that are larger than what we expect to encounter in real applications. It should be noted that the results reported in the reference method are based on a small reference collection of approximately 250 songs only and that, unfortunately, no information is given about either the size of the resulting database, or about runtimes.

The advantage of our proposed method is its efficiency in terms of data size and fingerprint computation. In contrast to the reference method, where log-spaced filter banks have to be applied to the spectrogram in order to compute the time-chroma representation, the selection of spectral peaks for quad grouping is done directly on the STFT magnitude spectrogram of the audio, and the quads can be grouped in linear time. Our proposed method chooses spectral peaks that are local maxima of the magnitudes in the spectrogram, in contrast to the compared method, where 600 DCT computations per second of audio have to be performed (similarity computations for 30 rectangular image patches for each of approximately 20 feature candidates per second) in order to find stable feature points. The hash

representation we propose is very compact and can be stored as four float32 values, while the reference algorithm uses fingerprints that are represented by 143-dimensional vectors.

Our match candidates are retrieved via an efficient range search in a spatial data structure, for which we use an R-Tree. The distance of hashes is the euclidean distance between four-dimensional points, while the distance measure used in the reference is the measure of the angles of the 143-dimensional fingerprint vectors.

3.6 CONCLUSIONS

The system we propose seems to be a viable and promising method to approach the task of identifying scale modified audio. While there is a lot of potential for false positive matches in a database of this size (in this chapter our database is an index to roughly 43 million quads) in combination with the rather large tolerated scaling ranges, the method's filtering stage and the subsequent verification process enable the system to maintain high precision and accuracy. The results show a stable high performance for a large range of scale changes, with as few as approximately 9 compact fingerprints per second of audio.

Our proposed method achieves near perfect overall performance within scale modification ranges of 90% to 115% for speed, and 80% to 120% for tempo scale modifications.

4

LARGE-SCALE QUAD-BASED AUDIO FINGERPRINTING

In the previous chapter we introduced the early and basic variant of our proposed robust audio fingerprinter. In this chapter, we refine the system that we presented previously, and introduce and apply insights that lead to increased efficiency in query processing, and to higher identification performance in terms of accuracy and precision. The chapter is based on the following article (The final version of record is available at http://dx.doi.org/10.1109/TASLP.2015.2509248:

> R. Sonnleitner and G. Widmer (2016). "Robust Quad-Based Audio Fingerprinting". In: *IEEE/ACM Trans. Audio, Speech & Language Processing* 24.3, pp. 409–421

This chapter describes the latest version of Qfp in detail, and builds the foundation for extensions and further improvements that we describe in Chapter 5 and Chapter 7.

The system we present in the following is, to our knowledge, the first audio fingerprinting method described in the academic literature that meets all of the robustness requirements we listed in Chapter 2, including the important aspect of demonstrating that the method indeed scales to larger reference collections. The proposed method is robust to noise and audio quality degradation, as well as to severe distortions of the time-scale and frequency-scale. Moreover, the identification method uncovers and accurately quantifies the scale transform factors of distorted audio queries as an integral part of its matching process.

The chapter is organized as follows. Section 4.1 briefly introduces the refinements of the method with respect to the previous chapter. Section 4.2 then gives a brief overview to recall the main points of our method, in order to set the context for the precise method description, which comes in three parts: Section 4.3 describes the feature extraction process and how to obtain hash representations from these features, which then together constitute the fingerprints. Section 4.4 describes the data structures that are used for the audio identification method. Section 4.5 details the identification method, i.e. the process of matching query audio with reference data by comparing the fingerprints. Section 4.6 systematically evaluates the performance of our method on roughly 430,000 queries with various properties against a

reproducible reference collection of 100,000 tracks. To demonstrate the noise robustness of the method we perform experiments on queries with signal to noise ratios (SNR) in the range of $-10dB$ to $+50dB$, and also test the robustness on various other effects.

In the following we give an overview of the refinements with repsect to Chapter 3.

4.1 NOTEWORTHY REFINEMENTS

We use a lower audio sampling rate of 8kHz instead of 16kHz, and reduce the STFT hop size from 128 samples to 32 samples (4ms). This results in an increased time resolution of hashes and contributes to higher identification performance. The decreased sampling rate reduces memory requirements and allows to compute the STFT in less time.

The peak extractor is refined to apply parabolic interpolation (generating a quadratic function through neighbouring points and assigning the maximum of this function to the corresponding peak position dimension) of peaks and incorporates post processing to guarantee that there exists at most one peak per filter window. This turns out to be necessary in some cases when digitally created tracks are ingested in the reference database or used as query excerpts.

On top of the refined peak extraction, we improve the quad grouping algorithm. We describe a way to create quads in an almost uniform distribution over time, for both reference database creation and query quad extraction. In addition, we pick strong quads based on their respective peak magnitudes and refine the quad grouping routine to automatically adapt the grouping window to the given tolerance parameters. We observe that we can considerably reduce the number or query quads by enforcing a so-called "relevant subspace constraint". These changes allow for higher identification performance and lower query run times.

For lookup, we use a different spatial datastructure that in our experiments results in lower run times than libspatialindex's R-tree ("libspatialindex" n.d.). Next, we reduce the storage demands of quad records by overwriting the inner points C, D with the computed quad hashes C', D'. Instead of quad point B of a reference quad, the quad's height and width is stored, which reduces work in the filtering stage. To reduce the level of indirection during the matching stage, we incorporate the file ID of quads into their quad records.

Further, we change the workflow of the last stages, where we change the algorithm to perform match sequence detection prior to match verification, which considerably decreases computational costs. The sequence detection is refined to use a simple outlier detection on candidate sequence members to further reduce the computational

load of the subsequent verification stage. Also, the order of filters in the filter chain is changed for increased efficiency.

The verification stage is changed to bisect into relevant time slices of the peakfile, and no longer depends on storage intensive peak trees. Instead of the more expensive nearest neighbour peak search for alignment, we simply check for a non-zero count of peaks in a parameterizable rectangular region. This allows us to perform the verification without the need of precomputed peak-trees, which demand several gigabytes of memory.

To summarise, while the general concept of the method is the same as described in the previous chapter, the whole implementation and parameterization changed and different data structures are used. Altogether, the novel aspects of the proposed work and their interdependent parameter modifications increase the identification performance while considerably decreasing the query run times.

4.2 METHOD OVERVIEW

To set the stage, we here present a short, three paragraph introduction to the main concepts of Qfp, and then proceed to explain the modified algorithms in the section below.

Our identification method operates on compact four-dimensional, continuous geometric hash representations of quadruples of points, henceforth referred to as "quads" (Lang et al., 2010). The points are local maxima in the two-dimensional time-frequency representation of reference audio material, and are referred to as "spectral peaks". For each quad we create a compact translation- and scale-invariant hash which is represented as a point in a four-dimensional vector space. Section 4.3 describes the feature extraction process in detail.

Quads, along with their hashes and respective source file-IDs of reference audio are the so-called fingerprints that we use for audio identification. The fingerprints are stored as records in a binary file, which is indexed by a data structure for subsequent efficient range-search queries. This stage of the method is described in Section 4.4.

For audio identification, i.e. processing and answering queries, we first extract quads and their hashes from the query audio excerpt. For each query hash we then perform a range-query in the reference data structure. The obtained result sets are filtered and sorted into sequences of match hypotheses for each potentially matching file-ID. Finally, the individual sequences are processed by an effective verification method, which is the key technique that makes the identification algorithm perform with high precision and specificity. These steps are described in Section 4.5.

4.3 FEATURE EXTRACTION

In this section we describe the extraction of audio features and computation of their geometric hash representations.

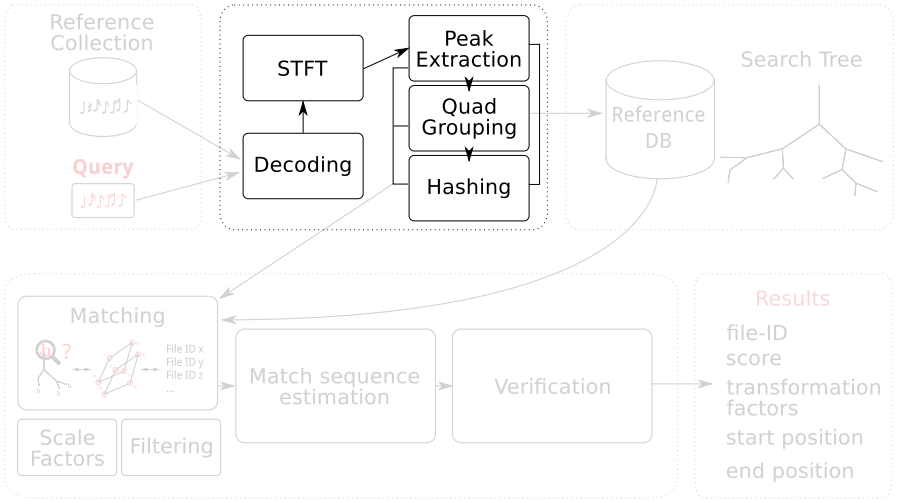


Figure 4.1: Feature extraction components

To begin with, in the decoding stage all audio files are downmixed to one-channel monaural representations sampled at 8 kHz. We then compute the STFT magnitude spectrogram using a Hann-window of size 1024 samples (128 ms) and a hop size of 32 samples (4 ms), discarding the phases. Peak extraction, quad grouping and hashing, as highlighted in Figure 4.1, are explained below.

4.3.1 *Constructing Quads*

This section presents the refined quad construction in detail. In order to explain the methods in their entirety, this section contains some level of redundancy to the explanations given in the previous chapter.

The extraction of peaks is implemented via a pair of two-dimensional filters, a max-filter and a min-filter, which are implemented as two-dimensional sliding window filters, and a post processing step to guarantee that there is at most one peak per window. We first apply the max-filter to the spectrogram. Locations of the spectrogram that are identical to the respective max-filter values are peaks. However, this does not guarantee that there is exactly one peak per window, as a spectrogram might contain regions of uniform magnitude. Examples of such cases would be silence, clicks, or (digitally created) tones with identical magnitudes. According to the max-filter result, each coordinate in such regions is reported as a peak. To clean this up, we use a second sliding window filter, a min-filter of size 3×3, compare the result to the spectrogram, and discard peak candidates if they are detected by both, the min and the max-filter. To give an example, we

assume absolute silence in a region of the spectrogram. Each magnitude within this region is both a maximum and a minimum, therefore each peak in that region will be discarded.

Next, we clean up cases of clicks or tones with a succession of identical magnitude values, since such cases will not be detected by the min-filter if their uniform regions contain parts that are smaller than 3×3. We group all peaks by magnitude, and search for adjacent peaks in each group. If adjacent peaks are found, we keep the first peak in time and frequency and delete all other peaks in the current filter window. This way we ensure to report exactly one peak per max-filter window. In the current implementation the cleanup procedure has a quadratic time complexity in size of peak magnitude groups, but in practice it turns out to be well applicable to audio spectrograms.

The resulting peak coordinates are now processed by parabolic interpolation based on their neighbourhood of 3×3 in time-frequency space. If the interpolated value lies outside the neighbourhood in any dimension, the original non-interpolated value is kept. The interpolation is applied in order to obtain a higher resolution of peak coordinates, in contrast to representing the peak position by the frame-index and frequency-bin index. The advantage of the higher resolution is explained after we introduce the hashing of quads in Section 4.3.3. From this point onwards, peaks are represented as single precision floating point values.

To create translation- and scale-invariant hashes from quadruples of peaks, we first have to group peaks into *quads* (Lang et al., 2010), where a quad consists of four spectral peaks A, B, C, D.

$$A_y < B_y \tag{4.1a}$$
$$A_x < C_x \leq D_x \leq B_x \tag{4.1b}$$
$$A_y < C_y, D_y \leq B_y \tag{4.1c}$$

Thus, we define a quad to be *valid* if the points C, D reside in the axis-parallel rectangle that is spanned by the points A, B. These validity constraints restrict the number and shapes of quads that can be grouped from arbitrary peak constellations. Naturally, there are numerous ways to achieve constrained constellations. We chose the above mainly because of two reasons: it can be efficiently computed and effectively limits the number of possible constellations. The algorithm subsequently strictly operates on *valid* quads only.

At the top level, the quad grouping process proceeds through an audio file from left to right, trying each spectral peak as a potential root point A of a set of quads, and aims to create up to a number of q quads for each second of audio.

The process of constructing quads by selecting appropriate sets of B, C, D points is as follows [1] : we construct a *quad grouping region* of

1 See Footnote 1 in Section 3.3.1 on Page 35.

width r, spanning r STFT frames ranging from r_{start} to r_{end}, such that the region is centered c STFT-frames from A and A is outside of the region (earlier in time, i.e. the region is located to the right of A). This is depicted in Figure 4.2a. For the reference database we want to create a small number of quads. We therefore choose a region of small width r and the center c in near proximity to A. We take all m peaks residing in this region and try all $\binom{m}{3}$ combinations of 3 peak indices in order to construct all valid quads for root point A. Resulting quads are appended to a result list.

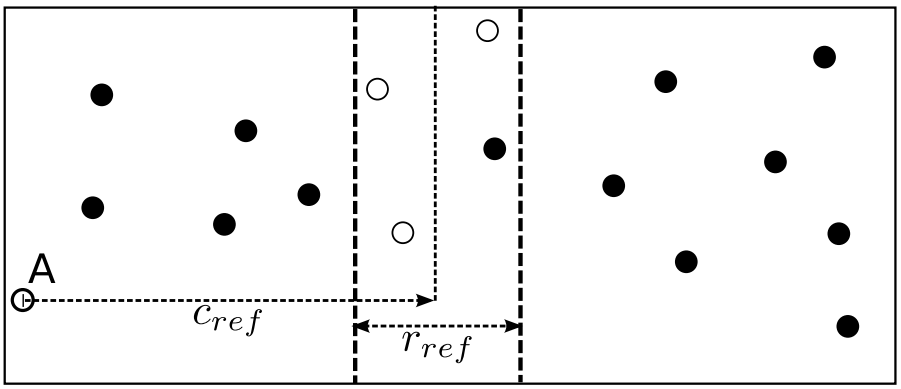
In many cases, creating all $\binom{m}{3}$ quads for a given region results in large amounts of quads. To restrict the total number, we pick "strong" quads from the set of valid quads, based on selecting, for each candidate point B, only a subset of C, D points based on their magnitude. We pick as many of the strongest quads, such that we get close to a maximum of q quads per second of audio. We do this by first creating all valid quads for each root point A. Then we bin the quads into groups according to the time values of their root points A, and select the strong ones for each root point in the binned group until we reach the number q quads per second.

There are notable advantages of this method over the method we proposed in the previous Chapter 3: we attain an almost uniform distribution of quads over time (i.e. per bin). Most importantly, this grouping method extracts the most robust quads from the set of all possible quads. Indeed, if the reference database is guaranteed to consist of only strong quads (i.e. quads that are grouped from peaks with large magnitudes), we can simply discard weak quads from query audio to considerably reduce the necessary amount of work for the identification task. Applying this proposed grouping method, we achieve increased identification performance, reduced reference database sizes, and faster query processing.

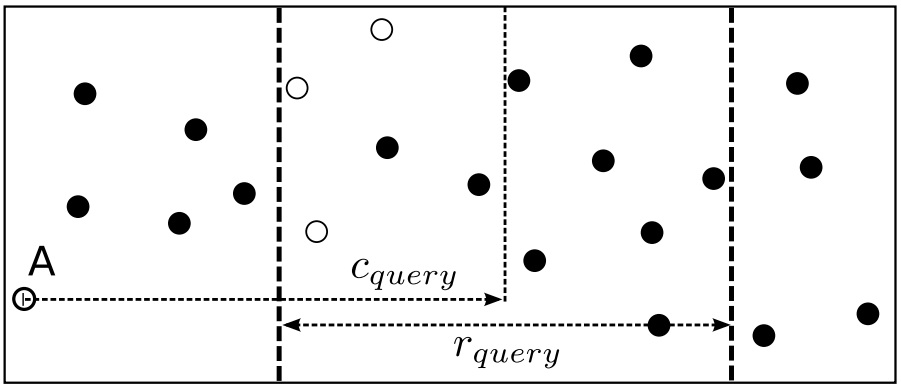
4.3.2 *Query Quad Construction*

The parameters for quad extraction in *queries* are computed from the reference quad extraction parameters and the specified tolerance bounds for pitch (p) and time (t) scale distortions ϵ_p, ϵ_t, that we are interested in detecting. For example, to detect query snippets such that the reference audio is in the range of $\pm 30\%$ with respect to the query, we let the tolerance bounds $\epsilon_p = \epsilon_t = 0.3$.

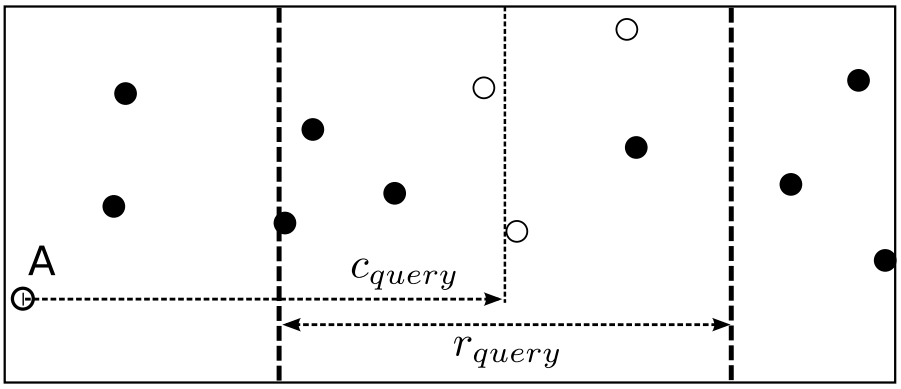
The length or duration of an audio piece with altered time scale is the inverse of the time scale factor. Thus, to identify query audio increased in tempo or speed, the algorithm has to account for the fact that the relevant peaks from the query will be closer to each other than the corresponding peaks from the slower reference. Therefore, we extract peaks from query audio at higher density, by using smaller max-filter sizes: the width (w) of the query max-filter m_w^{query}



(a) Reference quad grouping



(b) Query snippet with tempo 130%



(c) Query snippet with tempo 70%

Figure 4.2: Reference quad grouping (4.2a) and adaptive query quad grouping with increased tempo (4.2b), and decreased tempo (4.2c). 4.2a shows a root point A and the hollow circles B, C, D to group a reference quad. Figures 4.2b, 4.2c show the regions for the tolerance of $\pm 30\%$ in the queries, and the different peak densities. Note the important difference to Figure 3.2: r^{query} is scaled according to Equations 4.4, rather than proportionally to the tolerance bounds of $\pm 30\%$.

is computed from the reference max-filter width m_w^{ref} and ϵ_t using Equation 4.2:

$$m_w^{query} = m_w^{ref} / (1 + \epsilon_t) \tag{4.2}$$

The height (h) of the max-filter is computed as

$$m_h^{query} = m_h^{ref} \cdot (1 - \epsilon_p) \tag{4.3}$$

The borders of the query quad grouping region r_{query}, and the center c (c.f. Figure 4.2) are then obtained in dependency of $\epsilon_t \in [0; 1]$ by:

$$r_{start}^{query} = r_{start}^{ref} / (1 + \epsilon_t) \tag{4.4a}$$
$$r_{end}^{query} = r_{end}^{ref} / (1 - \epsilon_t) \tag{4.4b}$$
$$c^{query} = (r_{start}^{query} + r_{end}^{query}) / 2 \tag{4.4c}$$

The resulting peaks for a piece of reference and query audio for tolerances $\epsilon_p = \epsilon_t = 0.3$ are depicted in Figure 4.6. Note that the query peaks (shown as dots) are extracted at a higher density, as given in Equations 4.2, 4.3. The other aspects of Figure 4.6 become relevant in Section 4.5.3.

To summarize, the reason for different parameterization for query quad construction is as follows: when the time or frequency scale of query audio is modified, this affects not only the density of relevant peaks in the given audio snippet, but also their relative positions. An example is given in Figure 4.2, which shows the grouping for a quad for a given root point A. In 4.2a a reference quad is created for a region of width r that is centered c frames from A. The analogous example for grouping a query quad for the same audio, but increased in tempo, or decreased in tempo, is given in Figure 4.2b and Figure 4.2c, respectively. We see that the hollow circles, which represent the points C, D, B for the reference quad, may happen to move outside of the grouping region of width r if the time scale of the audio is modified. By choosing a larger region width r (see Equations 4.4) and a larger number q of quads per second of audio (qps), in combination with a higher density of extracted peaks, we try to ensure to obtain a quad that corresponds to the reference quad. Note that Figures 4.2b and 4.2c show the locations of reference peaks after altering the time scale, but do not show additional peaks that would emerge due to the higher peak density (i.e. smaller max-filter sizes).

Note that when we consider audio queries of a fixed, limited duration d (e.g., 15s), there is an important difference between increased speed/tempo and decreased speed/tempo. Increasing the tempo of the query audio excerpt relative to the reference leads to a higher density of relevant audio content; all the content that was used during the phase of reference quad grouping is also present when extracting

the quads for the query. However, decreasing the tempo, i.e. stretching the time scale, may cause some of the relevant spectral peaks to fall out of the 15s (i.e. not be part of the query any more), so some important quads do not emerge in the query. This difference in increasing vs. decreasing the time scale was actually reflected in the evaluation results of our previous work (see Section 3.5). We now alleviate this effect to a great extent by choosing smaller values for the parameters c, r. This has the effect that query quads of shorter time-spans are created, and thus more query quads can be extracted towards the end of a short audio snippet. Very similar to the above, severe frequency scale changes (i.e. *P*- or *TP*-modifications, but not *T*-modifications) can cause relevant peaks to leave the observed bandwidth of the audio signal.

4.3.3 *From Quads to Translation- and Scale-Invariant Hashes*

We now have created quads from spectral peaks in audio, but these quads are not the actual summarizing representation that we later use to find match candidates between a query audio and the reference database. We derive a scale-invariant and translation-invariant representation that is quickly retrievable by adapting the hashing model shown by Lang et al. (2010), and instead of exploiting invariance to angle-preserving transformations and rotation, we change the hash model to be invariant to non-isotropic transformations, i.e. different scale factors of either dimension. For our hashes, we deliberately discard the property of rotational invariance, and the resulting model enables us to compute *translation- and scale-invariant hashes* from the quads. This operation normalizes quads from time-frequency space, with spectral peaks A, B representing a rectangular patch with sides always parallel to the axes of the Cartesian coordinate system, into the two-dimensional *unit square*: For a given quad A, B, C, D, the constellation of spectral peaks is translated to the origin and normalized to the unit square, resulting in the four points A', B', C', D' such that $A' = (0, 0)$ and $B' = (1, 1)$, as shown in Figure 4.3. The actual continuous hash of the quad is now given by C', D', and is stored as a four-dimensional point (C'_x, C'_y, D'_x, D'_y) in a spatial data structure. Essentially, C', D' are the relative distances of C, D to A, B in time and frequency, respectively. Thus, the hash C', D' is not only translation invariant (A' is always $(0, 0)$), but also invariant to non-isotropic scaling.

Figure 4.3 shows the hash space for general quads, but the hash values C'_x, D'_x in fact are constrained to a subspace of this hash space. This "relevant subspace" constraint implicitly originates from the *reference* (*ref*) quad grouping stage, and depends on the parameters

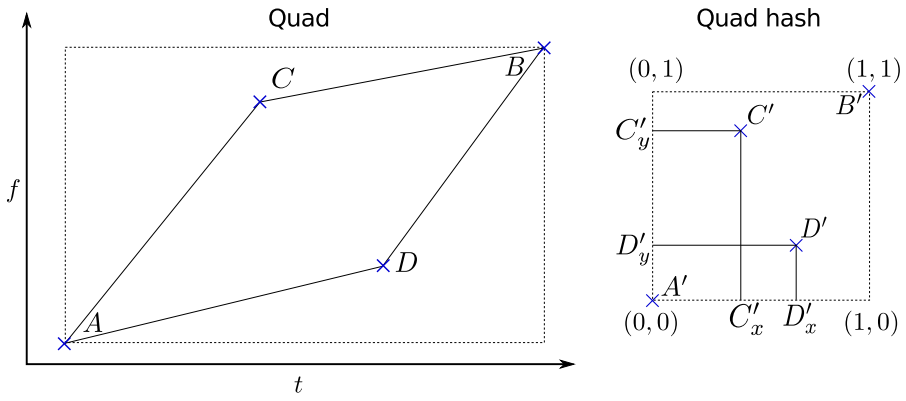


Figure 4.3: Example of a valid quad A, B, C, D and its corresponding hash.

c^{ref}, r^{ref}, which are the distance of the grouping window center from a root point A_x and the width of the grouping window:

$$C_x'^{ref}{}_{min} = D_x'^{ref}{}_{min} = \frac{c^{ref} - r^{ref}/2}{c^{ref} + r^{ref}/2} \tag{4.5}$$

Therefore, all query quads where $C_x'^{query} < C_x'^{ref}{}_{min} - \epsilon_L$ can be discarded – no equivalent quads exist in a reference database, and no range query lookup (L) with radius ϵ_L in the spatial data structure will contain a near neighbour. It is sufficient to only compare against C_x', because $C_x' \leq D_x'$. The relevant subspace is depicted in Figure 4.4.

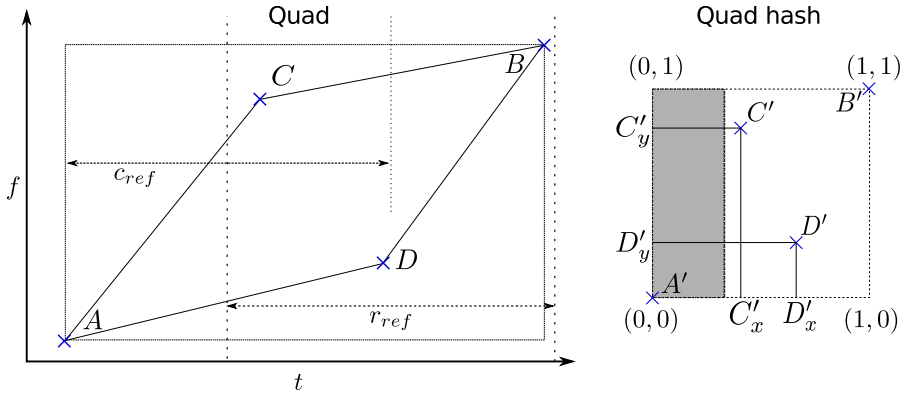


Figure 4.4: Example of a valid quad A, B, C, D that depicts c_{ref}, r_{ref}, and its corresponding hash. The hash space that is not greyed out is the relevant subspace.

An intuitive explanation of what it means when a quad does not originate from the relevant hash subspace is the following: a peak of the *query* quad moves closer to the root point, while at the same time another one retains its position or even moves farther away, which is of course not what we wish to detect since it is impossible that the signal was simultaneously increased and decreased in tempo or speed. Note

that we exclude such irrelevant quads from the set of query quads before we start picking strong quads based on peak magnitude values (see Subsection 4.3.1). To convey the effectiveness of enforcing that constraint, we measured the number of possible quads and the size of their relevant subsets: averaged over 11,700 queries, 44.7% of possible quads could be rejected. Selecting a number of q strong quads strictly from the remaining subsets considerably reduces the number of false negatives in the identification process.

Having introduced the hashing approach, we can now reason about the use of interpolated peak positions. Let us assume we are given a reference quad of a short timespan, e.g. one second between the root point A and the quad's point B. This amounts to 250 time frames. The corresponding general hash space therefore covers 250 frame positions, which would be quantized along the time dimension in steps of 0.004 if non-interpolated peak positions are used. In the case of non-scale-modified query audio, if for example the peak C migrates to a neighbouring STFT-frame, its hash value C'_x has a distance of 0.004 to the original reference hash time value. However, if the query is sped up by TP- or T-modifications, the query quad will cover an even shorter time-span. Considering a speed up of $+20\%$, the quad will span around 208 time frames. If the position of $C_x'^{query}$ migrates to a neighbouring bin, the absolute distance of $C_x'^{query}$ with respect to $C_x'^{ref}$ in hash space will grow to 0.0048, and to 0.0052 for $+30\%$ speed-up. Note that both of the peaks C, D are likely to move when the query audio is distorted, so the distance of query-hashes to reference-hashes is non-zero in the majority of the cases. The interpolation of peak coordinates allows us to use a smaller search range in hash-space when searching for matching quads. A smaller search range leads to faster search and to a lower number of false-positives that are present in a given search region.

4.4 FINGERPRINTS: STORING HASHES FOR EFFICIENT RETRIEVAL

Once peaks, quads and their hashes are computed from a piece of reference audio, we store the data in four data structures that together constitute what we call the reference database, which allows for efficient selection of match candidates and subsequent verification of match hypotheses from query audio. We refer to the four individual data structures as peakfile, refrecords, fidindex and searchtree.

The *peakfile* contains the continuous (interpolated) two-dimensional coordinates of all spectral peaks that were extracted for each piece of reference audio. Note that we do not store the peak magnitudes, as they are only used for the quad grouping to pick strong quads, and are not needed during the processing stages of audio identification. Each peak is represented by two single precision floats, and consumes

8 bytes. The peakfile stores peaks for each audio file as a contiguous sequence of records.

The *refrecords* file stores all quad records (quads and quad hashes, along with their audio file-ID) of all audio data in the reference collection. A quad record consist of spectral peak A and W, where W denotes the spatial extents (height and width) of the quad in time-frequency space (i.e. $W_x = B_x - A_x$ and $W_y = B_y - A_y$), the quad hash C', D', and the file-ID. Spectral peaks C, D are omitted, they are needed only for computing the quad hash.

The data for A, W and the quad hash are represented as four float32 values each, and the file-ID as unsigned int32. In total, a record in this data structure has a size of 36 bytes.

The *fidindex* maps each reference audio file to a unique file-ID and also stores the number of extracted peaks and quads, along with other meta-data. Given a specific audio file-ID, the fidindex is used to find the corresponding record range in the peakfile.

The *searchtree* is used to perform efficient range-search on quad hashes. We use a variant of what is called a shallow bounding-volume hierarchy (Dammertz, Hanika, and Keller, 2008) that stores and references nodes of quad hash ranges. Using this tree variant, it is simple to establish a memory-bounded tree construction. The tree does not create special leaf nodes – instead it marks inner nodes as leaves if they cannot be partitioned any further (e.g. because of the memory bound, or geometric constraints for the bounding volumes) and uses the pointers in the node structure to reference record ranges within our refrecords file. Each node in the hierarchy references a range of records which are stored contiguously in the refrecords file. On a high level, the tree construction can be understood as a kind of quicksort applied to the refrecords data structure, where the pivot-based array partitioning is guided by a binned approximation of the four-dimensional equivalent of the surface area heuristic (Wald, 2007). For each split, the coordinate values of the currently widest axis of the bounding volume are used. In our implementation, the tree construction is guided by a memory bound of 2048MB, and nodes are split as long as their bounding volumes are not too small. We also prohibit node splitting if the number of referenced primitives (i.e. refrecord entries) in a node is lower than a threshold value of 5 hashes. We revisit the search-tree algorithm and explain it in detail in Chapter 5. We also would like to point the reader to the highly optimized kD-tree implementation of Lang et al. (2010), but we did not yet test that implementation in our system.

The extraction of reference peaks is performed with a max-filter width of 151 STFT-frames (604ms), and a filter height of 75 frequency bins (585.9375Hz). The corresponding min-filter has a width of 3 STFT-frames and a height of 3 frequency bins. For reference quad grouping we choose the center of the grouping window c^{ref} to be 1.3 seconds

from each root point A. The width r^{ref} is 0.8 seconds. We group a maximum of $q = 9$ quads per second of reference audio.

The reference database we use for the experiments in Section 4.6 consists of 100,011 full length songs with a total duration of 6899 hours of music, with an average song duration of 248.33s. The indexed audio files altogether consume 550.4GB of disk space. Using the parameters as above, 216,429,829 peaks were extracted and stored in the peak-file which consumes 1.65GB. The peaks were grouped to 209,855,025 quads, thus the refrecords file consumes about 7.2GB of disk space. The nodes of the search tree consume 1068MB, and the fidindex file has a size of 6.7MB. Altogether, our reference database has a size of 9.85GB, which is roughly 1.8% of the size of the audio collection. Creating the reference database, using our Python implementation, took 24.8h (around 67 files per minute), utilizing seven out of eight logical cores of an Intel Core i7-4770 (3.4GHz) Processor.

The largest data structure we use is the refrecords file. While we do not experiment with compressed representations in this work, we want to point out that the refrecords file can be compressed from 36 bytes to 24 bytes per record, by utilizing scaled floating point representation. Hash values are in the range of $[0..1]$ (in fact, time values C'_x, D'_x reside in the *smaller* range of $[C'^{ref}_{x\,min}..1]$ as described in Section 4.3.3) and could be scaled and stored as unsigned short16 values. Likewise, the width and height of quads (W as above) could be represented in this format.

4.5 IDENTIFICATION ALGORITHM

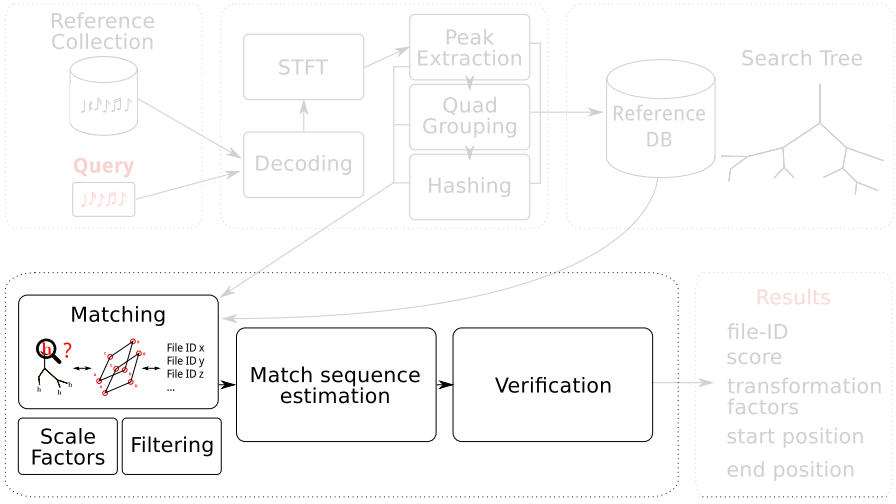


Figure 4.5: Search components

This section describes the algorithm that tries to identify a potentially severely distorted piece of query audio. If a matching audio is found within the reference database, the algorithm reports the

match file-ID, the position of the query piece within the reference audio, the underlying time and frequency scale modifications, and a verification-score.

The method of answering a query consists of three stages (see Figure 4.5): the first processing stage performs the selection of match candidates, followed by a filtering stage in which we try to discard false positive candidates. This is explained in Section 4.5.1.

Results are then passed to a match sequence estimation stage, in which we efficiently search for sequences within the set of matched candidates. This is done by comparing the offsets of match-times in the reference match-time-series, and the query match-time-series. Any line search algorithm can be applied here, but because of its trivial implementation, we adapt and extend the histogram binning approach that is proposed by Wang (2003). We also add an outlier removal step that discards individual match candidates within sequences based on statistics of the respective underlying scale transforms. This is explained in Section 4.5.2.

Finally, for each match candidate within the filtered sequences we apply a verification step, adapted from the findings in Lang et al. (2010). This step is essential to maintain high identification precision on large reference audio collections, especially in the presence of highly repetitive audio material. The verification is explained in Section 4.5.3.

4.5.1 *Match Candidate Selection and Filtering*

For each query quad hash a range-search in the *searchtree* is performed. This lookup returns a set of raw match candidates, which consists of those quad records with hashes that are similar (identical up to the search radius ϵ_L: $C_x'^{query} - \epsilon_L \leq C_x'^{ref} \leq C_x'^{query} + \epsilon_L$ etc.) to the query quad-hashes. We call this the set of raw candidates, as it will most likely be a mixture of true positives and a (large) number of false positive matches. From this point onwards, this stage and subsequent stages operate on the quads (i.e. the points in time-frequency space) rather than on their hashes, thus we can now discard the hashes from the result sets.

The raw candidates are processed by a series of three filters that reject false positives. This considerably reduces the number of raw candidates and therefore reduces the computational load of subsequent steps. Conceptually, if these filters are not applied here, the same effect of candidate rejection takes place in the verification stage (see 4.5.3), although in a less efficient manner.

The first filter tests the query quad and each of its match candidates (*cand*) based on coarse pitch coherence, similar to the spatio-temporal coherence check described by Evangelidis and Bauckhage (2013). This filter has the purpose of constraining the pitch translation invariance of our hash model. It discards raw candidates if these are farther away

along the frequency axis than our tolerance bounds account for. The filter routine accepts candidates if the following holds:

$$A_y^{query} / A_y^{cand} \geq 1/(1+\epsilon_p) \tag{4.6a}$$
$$A_y^{query} / A_y^{cand} \leq 1/(1-\epsilon_p) \tag{4.6b}$$

The next filter tests whether accepted candidates adhere to the previously introduced transform tolerance bounds ϵ_p, ϵ_t for pitch and time (see Subsection 4.3.2). This is achieved by looking at the different spatial extents of the query quads and their candidates and computing the scaling factors for time and frequency as follows:

$$s_{time} = (B_x^{query} - A_x^{query}) / (B_x^{ref} - A_x^{ref}) \tag{4.7a}$$
$$s_{freq} = (B_y^{query} - A_y^{query}) / (B_y^{ref} - A_y^{ref}) \tag{4.7b}$$

Quad candidates are accepted if the following holds:

$$s_{pitch} \geq 1/(1+\epsilon_p) \wedge s_{pitch} \leq 1/(1-\epsilon_p) \tag{4.8a}$$
$$s_{time} \geq 1/(1+\epsilon_t) \ \wedge s_{time} \leq 1/(1-\epsilon_t) \tag{4.8b}$$

The scale factors are stored with the accepted candidates and will be used in the verification stage to align reference peaks to query space (see Section 4.5.3).

The last filter is similar to the coarse pitch coherence filter, but now that we know the scaling factors of the candidates with respect to their reference quads, we can perform a fine pitch coherence filter as follows:

$$|A_p^{query} - A_p^{ref} s_{freq}| \leq \epsilon_{pfine} \tag{4.9}$$

where ϵ_{pfine} is the fine pitch coherence threshold. We set this threshold to the empirically determined value of 1.8. Ideally, we would expect $A_p^{query} = A_p^{ref} s_{freq}$, however, using ϵ_{pfine} we allow small positional inaccuracies along the frequency dimension. This is necessary to account for the fact that the locations of spectral query peaks are not invariant to signal distortions. Rather, locations of spectral peaks are of limited robustness to signal distortions.

After all three filters are applied to the set of raw match candidates, we now have obtained a set of match candidates which adhere to given constraints. We now sort the remaining accepted candidates by file-ID, and enter the sequence estimation step.

4.5.2 *Match-Sequence Estimation*

We perform this stage and the next stage of match candidate verification on a per-file-ID basis. Therefore we group the match candidates by file-ID, and sort the groups by the number of match candidates, in decreasing order.

Per file-ID, we try to find a sequence of candidates by processing the matches with a histogram method similar to the one described in Wang (2003). We adapt the method such that the query time A_x (the time value of root point A of each query quad in the sequence) is scaled according to the uncovered time scale factor s_{time}. The file-ID for the largest histogram bin (the longest match sequence) is returned, together with the match position, which is the minimal time value A_x of the peaks in the histogram bin. This time value localizes the query snippet within the reference audio.

Resulting sequences with a variance of scale transforms larger than a threshold value are cleaned up using a simple variance based outlier detection method.

Finally, if match sequences are found for a given file-ID, and their number of matched candidates is larger than a threshold value t_s, we try to verify these sequences match-by-match.

4.5.3 *Match Verification*

Verification of match hypotheses is based on the insight that spectral peaks that were extracted nearby a matching reference quad in the reference audio should also be present in the query audio, analogous to what is described by Lang et al. (2010). For this verification process we have to align the peaks of the relevant part of reference audio with the corresponding part of query audio, by aligning regions around the respective locations of the match hypothesis. In order to do so we use the previously computed scale transformation factors (cf. Equations 4.7a, 4.7b) for the current match candidate and then count the number of peaks that match in aligned space. We first describe the verification process and then show a visualization of the method.

Naturally, the number of peaks in proximity of the reference quad will differ from the number of peaks near the matched query quad. This depends on the query peak extraction density, and therefore on the scale tolerance parameters ϵ_p, ϵ_t, but also on the encoding, noise, and other distortions. Thus, the verification process must be robust to the existence of additional and missing peaks, so called *distractors* and *dropouts*. Because we search for the existence of reference peaks in the query audio, and not vice versa, distractors tend to get problematic only if the query peaks are extracted at very high densities – in this case the probability for false verifications increases. Dropouts on the other hand, may lead to false rejections. Therefore, we try not to find all, but just a percentage of t_{min} of the nearby reference peaks within the local query excerpt.

We define the nearby reference peaks as the set of N spectral peaks in the reference audio that exist within a timespan around the match candidate's root point A (for some fixed timespan), and retrieve those via a lookup for the peaks of the file-ID in the peakfile of our reference

database, and then bisecting the time values. In our initial work as described in Chapter 3, we proposed to search for nearby peaks in peak trees that are part of the reference database. These peak trees consume more than 13GB for the audio collection we use in this present chapter. Using the variant we propose here enables us to perform the verification without the need for these data structures, and consequently reduce the total disk space requirements of the reference data structures by more than 50%.

The transformation to align a reference peak P^{ref} with a query peak P^{query} in query space uses the scale transformation estimate S, with its two components S_{pitch} and S_{time} (cf. Equations 4.7a, 4.7b):

$$P^{query} = A^{query} + \mathit{off} \tag{4.10}$$

$$\mathit{off} = (P^{ref} - A^{ref}) \cdot S \tag{4.11}$$

where A^{query} is the root point of a query quad and A^{ref} the root point of an associated reference match candidate. The vector off is the transformed offset (i.e. the relative position) in pitch and time from A^{ref} to a nearby peak P^{ref}. P^{query} is the location estimate in query space where we expect to find a query peak that corresponds to the reference peak P^{ref}. Note that we search for P^{ref} in a small rectangular region which is centered at the location estimate P^{query}. In this specific implementation with $\epsilon_p = \epsilon_t = \pm 30\%$, we parameterize the rectangular region to span 12 frequency bins and 18 STFT frames (0.072s), i.e. we allow larger tolerances to align nearby peaks than during the match candidate filtering stage, where we aligned A_p^{ref}, A_p^{query} according to Equation 4.9. Here, we are less strict, to allow miniscule inaccuracies in the obtained scale factors, but most importantly to tolerate deviations from expected locations of spectral peaks that can occur due to audio signal distortions.

An example verification is depicted in Figure 4.6: here, the query audio was P-modified, such that the query has a pitch of 120% relative to the reference song. The upper figure shows an excerpt of the reference audio around the reference match candidate's root point A. The plus symbols and crosses represent the locations of the spectral peaks from this reference audio excerpt. The lower figure shows the query audio excerpt, where the dots represent the spectral peaks extracted for this query snippet. The plus symbols and crosses in the lower figure show the estimated locations of reference peaks in the query audio, and the rectangles depict the tolerance regions in which we search for the existence of a query peak. Crosses represent successfully aligned peaks and therefore always coincide with a dot. Plus symbols show failed alignments, respectively. In this example, 75% of the nearby reference peaks could be aligned.

The candidates that pass the verification step are considered true matches, and are associated with two measures, a verification score, and the uncovered scale transforms S. The verification score is the

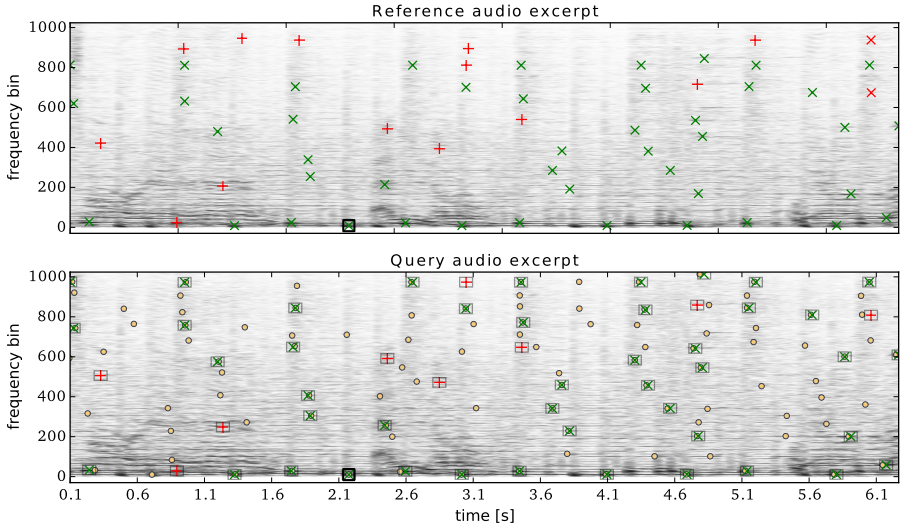


Figure 4.6: Verification of a match candidate for a query snippet of +120% pitch (i.e. the query audio has higher pitch than the unmodified reference audio). The two black squares are the root points A^{ref}, A^{query}. The plus symbols and crosses in the upper figure show all reference peaks in proximity of A^{ref}. The lower figure shows the location estimates of peaks in query space. The plus symbol means "not verified", the cross means "verified". The dots in the lower figure show all additional peaks which are extracted due to smaller peak filter sizes. Note that crosses always coincide with a dot, while this is never the case for the plus symbols. The rectangles show the tolerance regions for the alignments. The figure allows to observe that the high frequency reference peaks were shifted out of the query content. Note that this shows an example using a sampling rate of 16kHz, window size of 2048 samples and a hop size of 64 samples.

proportion v/N (with $v \leq N$) of correctly aligned spectral peaks in the set of nearby peaks (which in this example amounts to 0.75). Finally, if the sequence of verified matches for the given file-ID covers at least 15% of the query snippet length, we report the match. Note that this test should already be applied in the sequence estimation stage to achieve an rejection of candidate sequences as soon as these turn out to be too short.

At this point, we finally know the reference file-ID that identifies the query audio, the position of the query audio in the reference track, and the associated time-frequency scale modification S.

As we are interested in the best matching file-ID only, we can return the best verified sequence of the current file-ID as soon as it becomes evident that the next file-ID to process has a smaller number of associated match candidates than the number of verified matches in the current file-ID's best sequence (for this, the file-IDs are sorted by

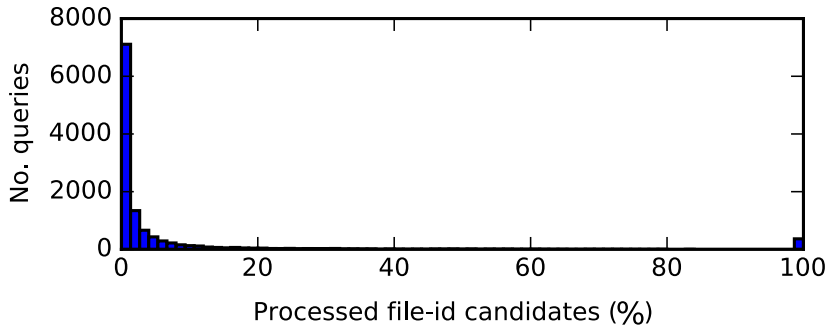


Figure 4.7: Histogram of the percentage of candidate file-IDs from the retrieved set of potential file-IDs that had to be processed in order to answer a query, computed from 11,700 queries. Note the bin at $x = 100\%$: here, the contributors are false negatives, and weakly matched file-IDs with exactly t_s matches (either true or false positives).

the number of match candidates in decreasing order before entering the sequence estimation step).

This early exit considerably reduces the amount of work to be done in this processing stage, and its effectivity is shown in Figure 4.7. In most of the cases the method identifies the correct file-ID, transformation and reference position for a query after processing the first few candidate file-IDs. Weak matches (with a low number of quads in a verified match sequence approaching the match threshold t_s) and false negatives make it necessary to process all candidate file-IDs, i.e. it is not possible to take the early exit.

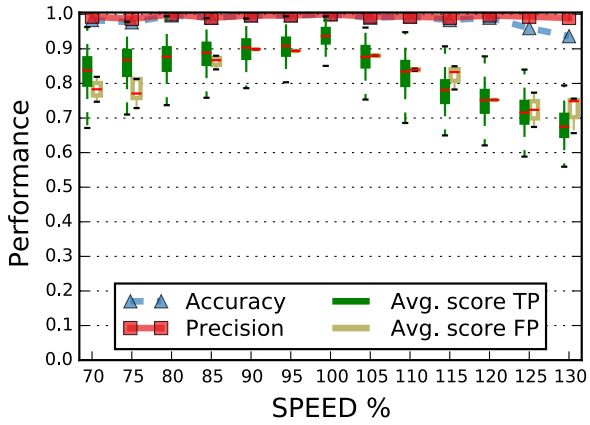
4.6 EXPERIMENTS AND RESULTS

The basis for all the experiments in this chapter is the reference database consisting of 100,011 as described in Section 4.4. The dataset[2] is freely available and we publish information that allows to recreate the reference collections, along with all queries that are performed in the experiments.

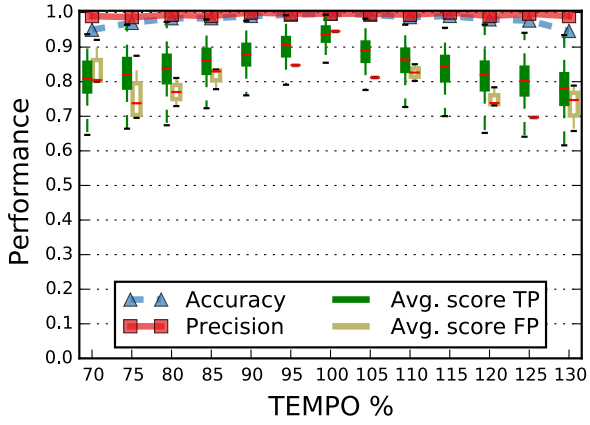
The performance of the method is evaluated using the measures that we defined in Section 1.7.

To create test queries, we randomly choose and fix a set of 300 reference songs and alter these with different *P*-, *T*-, *TP*–modifications as well as noise level modifications. We then randomly select a starting

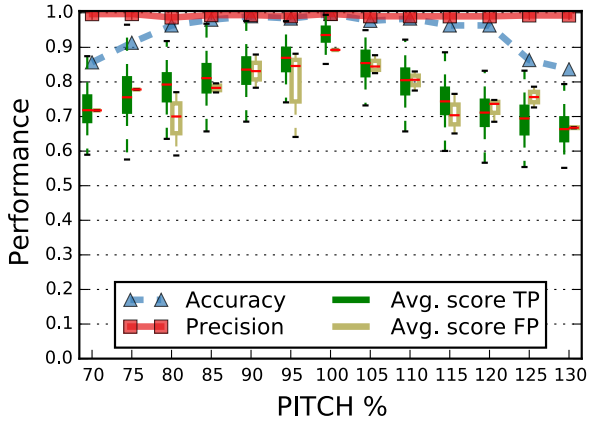
2 The dataset consists of freely available creative commons licensed music pieces, hosted by the jamendo service ("Jamendo Service" n.d.). Information to recreate the dataset is published with IEEE and can be downloaded from the media section at http://dx.doi.org/10.1109/TASLP.2015.2509248



(a) Speed variations

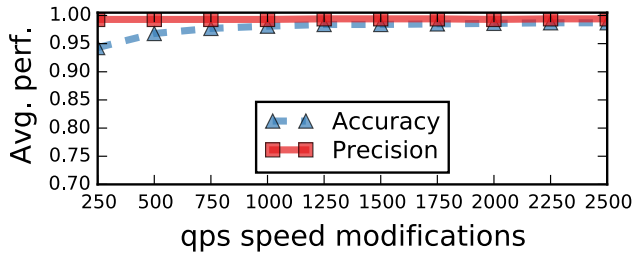


(b) Tempo variations

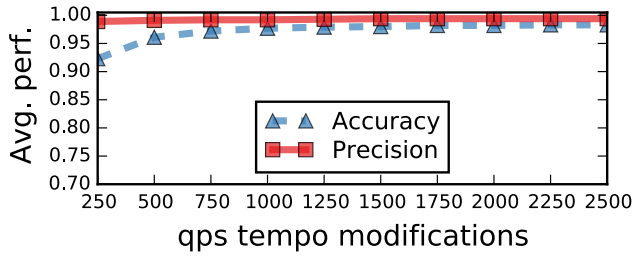


(c) Pitch variations

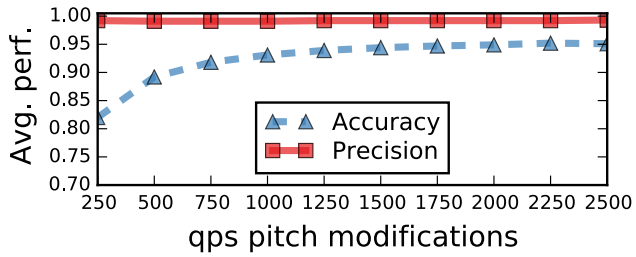
Figure 4.8: Precision and accuracy for *TP*- (4.8a), *T*- (4.8b) and *P*-modifications (4.8c) of 20*s* queries with a near-neighbour search radius $\epsilon_L = 0.01$ on a database of 100,000 songs. The figures show results of a total of 11,700 queries (3 kinds of distortions for 13 values over 300 queries), where each pair of data points shows the result of 300 queries. The boxplot pairs show the average verification scores of *tp* (left) and *fp* (right) sequences. In cases with perfect precision, no *fp* boxplots are shown.



(a) Speed and qps variations

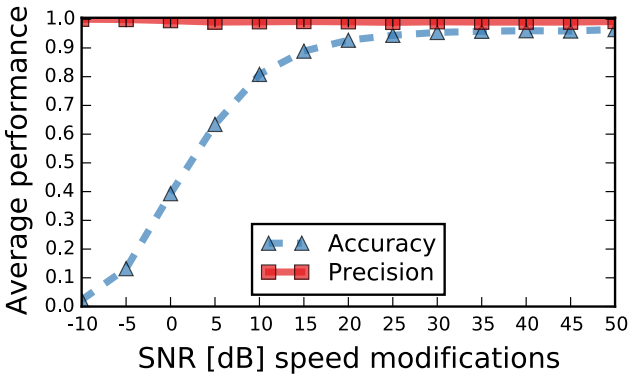


(b) Tempo and qps variations

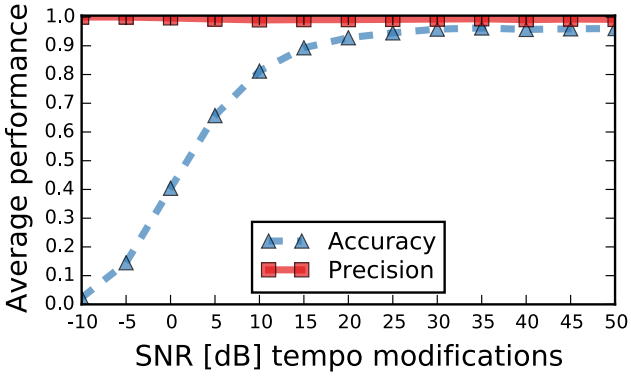


(c) Pitch and qps variations

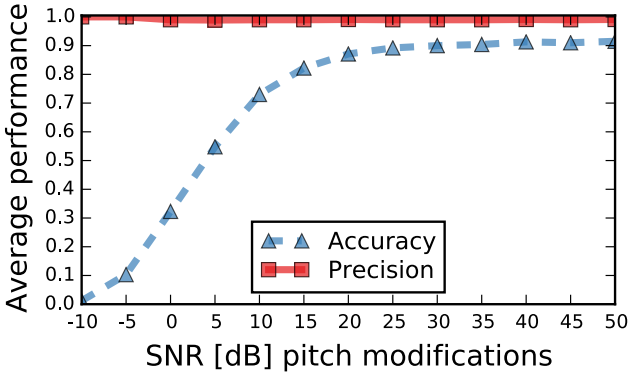
Figure 4.9: Precision and accuracy for *TP*-, *T*- and *P*-modifications (from top to bottom) on our reference database of 100,000 songs. Figures 4.9a to 4.9c (note the y-axis range) show results for a total of 117,000 queries, for various values of quads per second (qps, parameter q) and a query snippet length of 20*s* and $\epsilon_L = 0.01$. Each pair of data points shows the average precision and accuracy of 3900 queries for scale modifications in the range of $\pm 30\%$ of the individual type, as in the experiment shown in Figure 4.8.



(a) Speed and SNR variations



(b) Tempo and SNR variations



(c) Pitch and SNR variations

Figure 4.10: *TP*-, *T*- and *P*-modifications (from top to bottom) for varying SNRs. The figures show results for a total of 152,100 queries of various SNR, snippet length of 15*s* and $\epsilon_L = 0.01$, where each pair of data points is the average over 3900 queries using our reference collection of 100,000 tracks. Figure 4.10a shows *TP*-modified queries for 13 *TP*-modification values in the range of ±30%. Figure 4.10b and Figure 4.10c show *T*- and *P*-modifications, respectively, such that each pair of datapoints is one single experiment as depicted in Figure 4.8.

position for each selected song, and cut out 20 seconds from the audio, such that we end up with 300 query snippets with a duration of 20 seconds. These query snippets are used for all experiments.

We create the snippets from mp3 encoded data, and encode the distorted versions in the Ogg Vorbis format ("Ogg Vorbis" n.d.), using the default compression rate (cr $= 3$). We do this to demonstrate that the system is robust to effects that result from a lossy audio compression. All modifications are realized with the free SoX audio toolkit ("SoX - Sound eXchange" n.d.).

For all experiments in this work, the fingerprinter is configured with scale transform tolerances $\epsilon_q = \epsilon_t = 0.31$. Query peak extraction is performed with max-filter sizes of 51 frequency bins and 113 frames. The radius for range-search queries is $\epsilon_L = 0.01$. Sequences must contain at least $t_s = 4$ matches, and the threshold for the average verification score of sequences is set to 0.53. The verification stage considers peaks in the range of $\pm 1.8s$ near the reference candidate rootpoint, and uses rectangular alignment regions of height 12 frequency bins and 18 time frames. For the experiments we use a preliminary parallel implementation of our method, utilizing 7 workers and one master process on a Intel Core i7-4770 (3.4GHz) machine. All building blocks of the method are implemented in Python, except the peak extractor and the searchtree, which are implemented as C extensions.

We now explain the specificity tests and, for convenience, report the results in the following paragraph. The run times for the specificity experiments are documented in the last two rows of Table 4.1.

To test the specificity of the method, we prepare a second set of 20,000 query files from the jamendo service ("Jamendo Service" n.d.). We tried to ensure that the files contained in this set are disjoint with the set of referenced files, by inspecting the available meta-data. We found it is difficult to correctly clean up duplicates, because often the same song is present in another album of the same artist, referenced with another track ID, and inconsistent meta-data. We reject music pieces that appear to be duplicates according to the meta-data, and finally keep a subset of 18,229 files. The system's specificity on this set is 0.9617. Inspecting the results leads us to believe that there are still duplicates present. The meta-data of some of the high scoring matched files are very similar, but not equal to meta-data of the query audio piece. To work around the problem of duplicates, we perform a second experiment, on yet another dataset that we prepared to test the specificity of our system. This dataset consists 50,000 non-free songs from a different collection that we cannot make available. This experiment results in a higher specificity of 0.9726, but even in this experiment the top matching query audio seems to be a duplicate of a song that is present in our reference database. Because the meta-data are inconsistent, and we do not have a ground truth reference, we cannot be sure if some of these cases are duplicates or versions of the

	Speed (*TP*)		Tempo (*T*)		Pitch (*P*)		mean	median
l [s]	prec.	acc.	prec.	acc.	prec.	acc.	t [s]	t [s]
20.0	.994	.984	.994	.980	.992	.944	1.69	1.65
17.5	.993	.981	.992	.976	.991	.929	1.49	1.45
15.0	.991	.976	.991	.969	.992	.905	1.29	1.26
12.5	.990	.964	.990	.955	.990	.873	1.06	1.03
10.0	.990	.948	.989	.929	.987	.833	0.83	0.80
7.5	.987	.908	.988	.865	.979	.735	0.59	0.57
5.0	.980	.770	.978	.709	.953	.563	0.37	0.35
2.5	.798	.293	.777	.281	.668	.184	0.19	0.16

l [s]	*tn* dataset	No. queries	Specificity	mean	median
20.0	jamendo	18,229	.9617	1.96	1.77
20.0	non-free	50,000	.9726	1.89	1.67

Table 4.1: Average performance and run times ($qps = 1500$).

matched song, and therefore have to treat such cases as false positives. For both experiments the queries are of length 20s, starting at position 40s of the query audio piece.

All experiments presented in the following regard identifiable queries, and show individual results for *P*-, *T*-, *TP*–modifications. Note that in the vast majority of experiments, the reported precision is impacted by a set of three songs which we believe to be duplicates, but without the corresponding ground truth we cannot be sure, so we strictly count these cases as false positives.

4.6.1 *Results on Scale-Modified Queries*

We consider scale distortions in the range from 70% to 130% in steps of 5 percentage points, and extract a number of $q \approx 1500$ quads per second of query audio of length 20s. The results are shown in Figures 4.8, and are annotated with box plots of the average scores of matched sequences, which are shown for true positives and false positives if such sequences were predicted. Increasing the verification threshold (cf. Section 4.5.3) reduces the number of false positives, at the cost of introducing a number of false negatives for larger scale transform modifications. For industrial applications, this threshold should be optimized via large-scale experiments.

Results for the same kind of experiment, for various lengths of query snippets are given in Table 4.1, where we show the *average* performance of the individual types of distortions. Each row of Table 4.1 shows results of 11,700 queries (300 queries are tested for 3 types of scale distortion, over a range of 13 different scale factors), except for the last three rows, which show the system's specificity and run times for queries that are not present in the reference database.

We observe that false positives with high verification scores tend to be cases where the fingerprinter confuses different *versions* of a song. Specifically, the high-scoring false positives are due to the fact that it is hard for the fingerprinter to distinguish *scale modified* and *highly repetitive* short snippets of long remixes of electronic music from the original version, a problem which is equally hard for human listeners (especially if the major difference of the segment of the remix is the altered tempo or pitch). Depending on the application it might be advantageous to maintain a dictionary that relates versions of a song to a reference version. We currently do not have such a dictionary for our data set, so we strictly treat those cases strictly as false positives.

4.6.2 *Results on qps Values*

Identification results for various values of the number of extracted query quads per second (qps, parameter q) are shown in Figures 4.9a to 4.9c. For each value of q an experiment as in Figure 4.8 is performed, and the averaged results over the total of 117,000 queries (11,700 queries for 10 values of q) are shown in the first row of Figure 4.10. Decreasing the number of query quads negatively affects the identification performance for the most severe transformations only, while greatly reducing the run time of query processing: The mean query run times for the qps range from 250 to 2500 in steps of 250 are: 0.74s, 0.94s, 1.14s, 1.34s, 1.51s, 1.69s, 1.86s, 2.02s, 2.17s and 2.33s.

Thus, the proposed method of picking strong quads seems indeed to be effective in discarding irrelevant quads. Note that without imposing a limit of q quads per second, in many cases more than 4500 quads would be extracted per second of query audio.

4.6.3 *Results on Effects and Noisy Queries*

The impact of various effects on the performance of the presented method is evaluated and summarized in Table 4.2. For comparability we chose the same effects with identical parameterization as described by Six and Leman (2014). In total, a number of 6 effects is used, which are processed using the "SoX" toolkit ("SoX - Sound eXchange" n.d.). To describe the effects we take the liberty of showing excerpts from the man-page of "SOX" for information on chorus, echo, flanger and GSM effects:

- Bandpass: A two-pole Butterworth band pass filter.
- Chorus: The chorus effect has its name because it will often be used to make a single vocal sound like a chorus. But it can be applied to other instrument samples too. It works like the echo effect with a short delay, but the delay isn't constant. The delay is varied using a sinusoidal or triangular modulation. The

	Qfp		Panako	
effect type	prec.	acc.	prec.	acc.
Bandpass	.997	.993	-	.876
Chorus	.986	.687	-	.33
Echo	.993	.980	-	.686
Flanger	.984	.827	-	.573
Gsm	.978	.440	-	.364
Tremolo	.990	.977	-	.673

Table 4.2: Performance on 20*s*-queries without scale modifications, that are distorted by various effects. Column "Qfp" shows the results using our method (with $qps = 1500$). Column "Panako" shows the transcribed results for the same effects as presented by Six and Leman (2014), on their smaller reference collection of 30,000 tracks. A visualisation of example spectrograms for the effects is given in Figure 6.4

modulation depth defines the range the modulated delay is played before or after the delay. Hence the delayed sound will sound slower or faster, that is the delayed sound tuned around the original one, like in a chorus where some vocals are a bit out of tune.

- Echo: An echo effect can be naturally found in the mountains, standing somewhere on a mountain and shouting a single word will result in one or more repetitions of the word (if not, turn a bit around and try again, or climb to the next mountain). However, the time difference between shouting and repeating is the delay (time), its loudness is the decay. Multiple echos can have different delays and decays. It is very popular to use echos to play an instrument with itself together, like some guitar players (Brain May from Queen) or vocalists are doing. For music samples of more than one instrument, echo can be used to add a second sample shortly after the original one.

- Flanger: The flanger effect is like the chorus effect, but the delay varies between oms and maximal 5ms. It sound like wind blowing, sometimes faster or slower including changes of the speed. The flanger effect is widely used in funk and soul music, where the guitar sound varies frequently slow or a bit faster.

- GSM: The GSM audio file format. SoX has support for GSM's original 13kbps 'Full Rate' audio format.

- Tremolo: Low frequency amplitude modulation effect.

The most challenging distortions are the gsm codec and the chorus effect, and to a certain extent the flanger effect. For the effects bandpass,

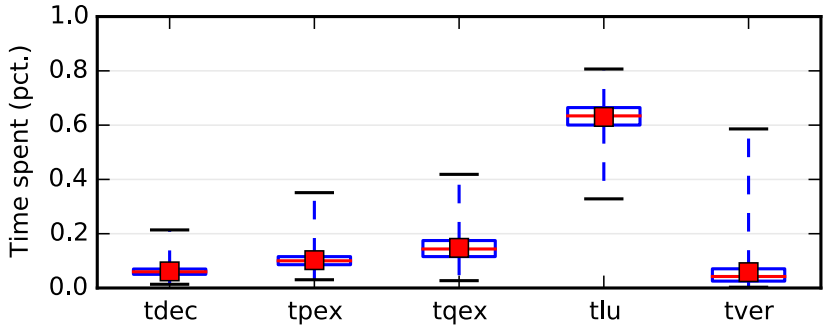


Figure 4.11: Fraction of time spent in the processing stages by the average query, averaged over the 11,700 queries performed for Figure 4.8 ($l = 20s$, $\epsilon_L = 0.01$, $qps = 1500$). This figure represents a median query processing time of $1.65s$.

echo and tremolo, the system maintains high performance with more than 97% accuracy and precision.

To evaluate the performance on white noise, we modify each query snippet of the individual scale transforms and create noisy versions in SNR ranges from $-10dB$ to $+50dB$ in steps of $5dB$. The results are visualized in Figure 4.10. For this experiment, a total of 152,100 queries were processed (11,700 queries for each of the 13 SNR values).

The results show a stable and high performance for SNR down to $+15dB$ (with *P*-, *T*-, *TP*–modifications in the range of $\pm 30\%$), which is a lower SNR value than what we expect to encounter in application scenarios.

4.6.4 *Run Times*

We identify five building blocks in our method, and take note of the time spent in these stages over the 11,700 queries performed for the experiment shown in Figure 4.8, using query snippet lengths of $20s$ and a search radius of $\epsilon_L = 0.01$. The stages are audio decoding (tdec), peak extraction (tpex), quad extraction (tqex), tree lookup and filtering (tlu), as well as match sequence estimation and verification (tver).

Figure 4.11 shows the fraction of time spent for the individual stages. Time spent on the lookup depends on the number of box-box intersections during tree traversal, and the size of the result set that is then filtered for the range constraint ϵ_L, pitch coherency and transform tolerances. The outliers for match sequence estimation and verification can be explained in context of Figure 4.7: in a great majority of all queries, the method takes the early exit after the first few candidate file-IDs, which means that only a small fraction of all potential file-IDs are processed in this stage. For the case of false negative matches,

or weak matches (i.e. approaching the matching threshold t_s) many candidates have to be processed.

In the reasonable scenario where a machine is dedicated to fingerprinting (we use 7 worker processes), with the settings as in Figure 4.8, we can process 213 20s-queries per minute, and 280 15s-queries per minute. Single-worker run times for various snippet lengths are given in Table 4.1. We believe the run times are within practical limits, considering the size of the reference database and the large tolerances of scale modifications.

4.6.5 *Comparison with Reference Methods*

While it is not possible to directly compare our results to those of Zhang et al. (2015) (because we do not have access to their test data), from the published figures it seems fair to say that in terms of recognition accuracy and robustness, both methods seem comparable, and both seem to approach the upper end of what can be expected of an automated audio identification system. For T- and P-modifications, our method seems slightly more robust for the ranges we evaluated, and it has a noticeably higher performance for queries that were severely slowed down in speed. Regarding the efficiency of Zhang et al. (2015), we do not know run times of the reference implementation, and we do not know how large the data structures for reference databases will be in practice. We cannot directly answer if the nearest neighbour search using LSH will still be that effective in the presence of very large databases, because the chance that the nearest neighbour constitutes the correct candidate decreases with the number of referenced fingerprints. However, given that LSH allows to be tuned in various ways, we assume that the method will be well applicable in practice.

A direct comparison to the algorithm presented in (Six and Leman, 2014) is possible, because the authors publish code and data. Given the strong limitations of the system with regard to scale change robustness, with a true positive rate of roughly 50% for scale distortions in the range of $\pm 5\%$ for 20s query snippets, we do not evaluate that system on our large reference database. However, we utilize the information given in Six and Leman (2014) to recreate the identical distortions for the robustness experiments, and present the results as part of our evaluation on our reference database consisting of 100,000 full length audio pieces. The results on queries that are modified with effects are given in Table 4.2. We assume that the limited robustness of Six and Leman (2014) is because of the specific hashing model used in combination with spectral peaks: it contains absolute values of quantized frequency components of feature points, and deltas of the quantized time values as well as frequency values, which are directly used to create the hash. This resulting representation is prone

to quantization effects, which occur if one or more peaks of the triple migrate into a neighbouring bin. Such cases will result in a hash that is different to the original, even though the triples are highly similar. This is an inherent property of quantized hash models, and here, its effect is exposed by the limited location robustness of the spectral peaks. Quantization effects expected even for hashes that are obtained from unmodified audio: depending on the start position of query snippet decoding, query peaks sometimes migrate to a neighbouring bin with respect to the other query peaks. If just one peak of the triple moves relative to the other two peaks (e.g. assume it is assigned to bin_{t-1} instead of bin_t along the time axis), the time deltas within the triple change and the triple is likely to be assigned to a different hash. We assume that this could be the reason for the large number of false negatives. The reported false negatives that occur even in the case of unmodified audio queries are an indicator of strong robustness limitations of quantized hashes for key/value lookup methods.

4.7 DISCUSSION

The scale transform tolerances of $\pm 31\%$ used in this thesis do not reflect the upper bound of what our method can handle. It can detect more severe scale modifications, but the runtime increases with larger values of ϵ_p, ϵ_t, and decreases with lower values. This is because larger tolerance values demand a higher density of query peaks, and due to the implicitly increasing size of the grouping window width r, a greater number of query quads will be grouped before the number of qps strong peaks will be selected. Also, fewer match candidates will be rejected due to larger transform tolerances, and therefore more candidates are passed to the fine pitch coherence filter and possibly even to the verification stage.

Speaking of the verification, we would like to shortly revisit this topic. While the verification is a highly effective way to drastically reduce false positives, there still is some room for improvement, because there remain some (rare) cases of correctly verified matches that in fact are false positives. To give an example, we visualise the correct verification of an incorrect match candidate in Figure 4.12. In some cases with mostly percussive content the transformation invariances of our hashes could cause the verification to match the strong onsets. However, there seems to be a simple solution to that, by storing a larger number of reference peaks, but without creating more reference quads. To do so, after creating the reference quads, one could simply perform a second pass of peak extraction, with higher density. The advantage would then be an even more effective verification, and the downside would be more storage consumption for the pdb and a small run time penalty for creating the reference database (because of the second pass of peak extraction).

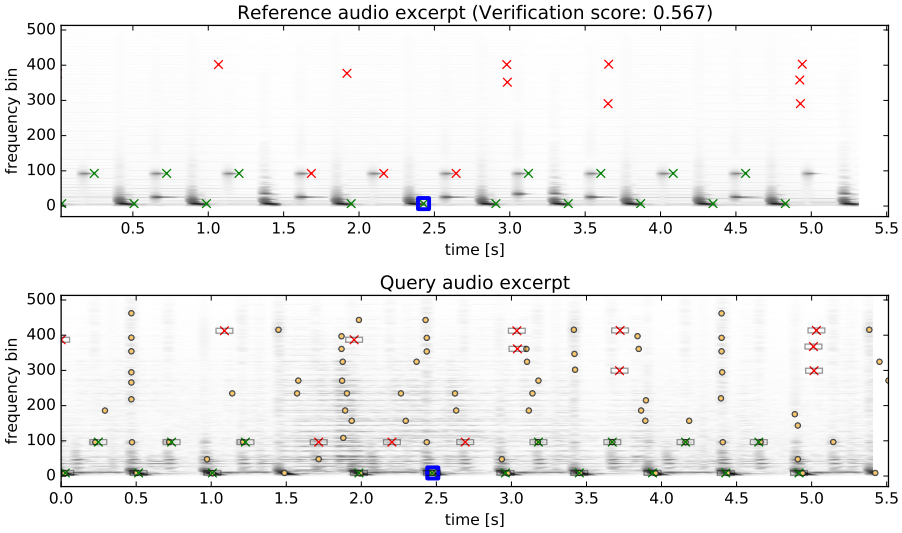


Figure 4.12: Example of an incorrect verification of a single match-candidate, resulting in a false positive match. Note, to report a false positive *sequence*, the verification step has to fail on several match candidates of the same incorrect file-ID, and these matches still have to end up in the same bin during sequence estimation. The meaning of the markers (cross, plus, circle, box) is described in Section 4.5.3 and Figure 4.6.

To summarize and conclude, instead of utilizing quantized hashes, we perform range search in the continuous four-dimensional hash space to retrieve raw candidates. Thus, quantization effects of hashes, as they predominantly emerge in combination with lookup tables, are not of concern. To loosen the strict hash similarity constraints of systems that use key-value lookup methods, as the work described by Six and Leman (2014), hash keys for adjacent bins must be computed for additional queries, which in turn limits the efficiency of such methods. The system described by Lang et al. (2010), and our proposed system, allows for intuitive parameterization of the range-search radius ϵ_L to adapt to robustness requirements.

We identify two reasons why our method on the whole exhibits *robustness* to scale changes, rather than *invariance*, despite the invariant hashing model: the very basis of the method are spectral peaks, and their respective locations are not invariant to scale changes or introduced digital artifacts. This is predominantly shown by the lower *P*-modification robustness of our method, compared to *TP*- and *T*-changes. The second reason is, that under severe scale changes, relevant peaks leave the observable regions of query audio, thus some important peaks cannot be found. This, however, is a general issue and not specific to our method.

Lang et al. (2010) discuss the use of n-tuples rather than strictly quadruples of points. As suggested in their work we use quad based hashes and observe that the computational cost and identification performance are well within reasonable limits. Using triples will inevitably result in much larger sets of retrieved match candidates (that need to be filtered and verified subsequently), because triples have a lower discriminability than quadruples of points.

5

SEARCH ALGORITHM

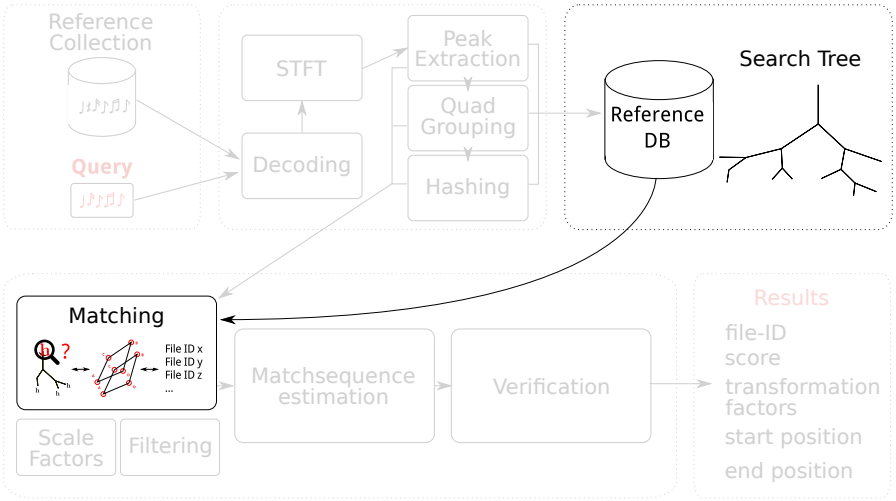


Figure 5.1: System components for searching

This chapter details the search algorithm that we implement to answer range-search queries. The search algorithm involves the construction of the search tree and its traversal, as sketched in Figure 5.1.

5.1 OVERVIEW

In our specific problem setting, we found via extensive experimentation that nearest neighbour search, k-NN search, as well as approximate variations thereof result in poor identification performance. The search space is densely, but non-uniformly populated by hashes which are computed from spectral peak coordinates. These coordinates have limited robustness to signal modifications, distortions and compression artefacts, but may also depend on the actual starting point in time (i.e. the signal sample) of the decoding process.

In experiments we learned that it is highly unlikely that the nearest neighbour of a query point is the correct match. When extending nearest neighbour search to k nearest neighbour search (k-NN), it is not readily apparent how to choose k. For less densely populated hash space regions, the volume spanned by k neighbours might be larger than we want. Vice versa, densely populated subspaces will return k candidate points that occupy a small volume of space – possibly smaller than the tolerances we have to account for.

When refining the search stage, we initially experimented with grid search, simply because of its trivial implementation. For range-search,

neighbouring grid cells have to be inspected. In a four dimensional search space, a grid cell has $3^4 - 1 = 80$ neighbours. In our problem setting, this turned out to result in prohibitively long run times. While one can guarantee fewer neighbour lookups by choosing larger cells, this comes at the cost of each cell containing larger numbers of reference points. However, by experimenting with (run time intensive) grid search we could validate our intuition that the identification performance greatly benefits from range search (all near neighbour search). A highly appreciated property is that range-search makes it possible to intuitively tune the trade-off between identification performance and runtime performance by experimenting with different sizes of the neighbourhood radius. To clarify, the term neighbourhood radius implies a hypersphere as neighbourhood, which makes sense if the same tolerances must be used for each dimension. However, we found that we need higher tolerances for the time domain than for the frequency domain. Thus, our search neighbourhood is a hyper-rectangle instead of a hyper-sphere, and is set to cover wider spatial extents for the two time-related dimensions in hash space (C'_x, D'_x) than for the pitch related dimensions (C'_y, D'_y).

We want to efficiently retrieve all reference records if their hashes reside in a given range (a neighbourhood) around the query point. We visualize the concept in two dimensional space, but these concepts trivially extend to higher dimensional space. The problem setting can be depicted as shown in Figure 5.2.

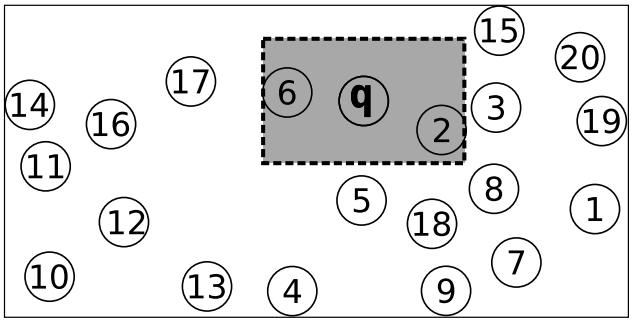


Figure 5.2: Example of range-search. The figure shows the search space that is populated with reference points (labelled by their IDs). The circle q is the query point with its search neighbourhood (grey area). The expected result set consists of the records 6 and 2.

For lookup (L) tolerances in time (t) and pitch (p), $\epsilon_{L_t}, \epsilon_{L_p}$, the width of the neighbourhood is $2\epsilon_{L_t}$, and the height is $2\epsilon_{L_p}$. Thus, for the four-dimensional quad hashes the neighbourhood around a query hash C', D' is parameterized as follows:

$$\epsilon_L = C'_x \pm \epsilon_{L_t}, C'_y \pm \epsilon_{L_p}, D'_x \pm \epsilon_{L_t}, D'_y \pm \epsilon_{L_p}$$

.

To construct a search tree, we choose a variant of a bounding volume hierarchy (see Section 5.2 below), and implement the tree construction and traversal specifically for the AMD64 architecture, with dependencies on the SSE3 instruction set.

5.2 BOUNDING VOLUME HIERARCHIES

A bounding volume hierarchy (BVH, Goldsmith and Salmon (1987)) is an object hierarchy. Rather than subdividing cells of space, it is a partitioning scheme that recursively splits object lists into disjoint partitions. The splits take into account the spatial extents of the primitives residing in the object lists.

We choose to implement a BVH variant that is called shallow bounding volume hierarchy (Dammertz, Hanika, and Keller, 2008). It is a 4-ary BVH that is designed to exploit the SIMD capabilities of CPUs.

This data structure exhibits a low memory footprint, and can be restricted to use a predefined amount of memory. Excessive nodes that do not fit into the allowed memory pool will then simply reference more hashes. One reason for the low memory footprint is that the leaves are not explicitly stored (i.e. leaves do not consume memory), rather the parent nodes are marked as leaves and directly index into contiguous regions of reference hashes: if a node split results in a constellation of points with small spatial extent or low number of points, instead of creating a leaf node, the node is marked as leaf and its pointers are adjusted to point to the contiguous regions of memory that hold the primitives.

5.3 DATA LAYOUT

The tree search combined with the filtering stage is the most run time expensive processing stage in our system. After peak extraction and quad grouping for a query audio piece, for thousands of query quads we have to retrieve all nearby reference quads from data structures that (currently, in our case) exceed 20GB in size. To do this efficiently, the highest priority is to design a well suited data layout scheme.

Memory and upper cache level accesses are the operations where most of the time is spent unproductively, i.e. the system processes data faster than it manages to transfer it to the CPU. The goal is to achieve a throughput that is limited by the CPU rather than by the memory subsystems, which can be a highly challenging task in practice. A general guideline is to organize the data such that consecutively accessed data are close together in memory, and algorithms should be implemented to reduce the need for scattered memory accesses in favour of more predictable and linear memory access patterns.

Our data layout scheme is as follows: The 4-ary BVH operates on a memory mapped contiguous region of storage that holds the data

of the entire reference database. This storage contains reference quad records, and in terms of BVH this is the "object list" that will be partitioned to resemble a hierarchy in the form of a 4-ary search tree.

A reference quad record consists of the coordinates of two spectral peaks A, B, the quad hash C', D', and its file-ID. We compress the quad hash components to 16bit each, by scaling the floating point representations. The actual reference quad records thus are as follows: A_t, A_p, B_t, B_p are single precision floating point coordinates in 2-D spectrogram space, C'_x, C'_y, D'_x, D'_y are four unsigned int16 values and the file-ID as 32-bit data type. Instead of actually storing the spectral peaks B_t, B_p of quads, we store the width and height of the quad: W_t, W_p with $W_t = B_t - A_t, W_p = B_p - A_p$. This will save some computations later on when the filter chain processes the lookup results. In total, a single quad record consumes $16 + 8 + 4 = 28$bytes In Chapter 4 the reference quad record size was $16 + 16 + 4 = 36$bytes.

Both the compressed as well as the uncompressed records lead to suboptimal data layout, as individual records are not aligned to 16 byte boundaries in memory. Padding the compressed records to 32bytes would be a waste of memory. To solve this we reorganize the data layout. As the range-search algorithm solely operates on the hashes, and do not need to access A, W and the file-ID, we split the reference quad records into three separate contiguous regions of memory:

1. *hashes*: (C'_x, C'_y, D'_x, D'_y) in compressed representation
2. *qaws*[1]: the quad points A, W, i.e. spectral peak A_t, A_p and the width and height W_t, W_p of the quad in spectrogram space.
3. *fids*: the file-IDs

The order of records is kept, thus the original reference quad record with index i can be obtained by reading the i-th entries of the *hashes*, *qaws* and *fids* from their separate storage areas. The storage areas are aligned to memory regions starting at 16 byte boundaries. The subsequent filter chain has to access the *qaws* and *fids*, which are now conveniently stored in separate contiguous regions of memory.

5.4 TREE CONSTRUCTION

The tree is a hierarchy of quad record partitions. The partitions are created according to the spatial extents of the contained quad records, which are represented as axis aligned bounding boxes (aabb). A tree node consists of four aabbs, and indices into the contiguous regions of the reference records that reside in each box.

To construct the tree, first the aabb of the entire collection of quad hashes is computed. This box represents the minima and maxima of C'_x, C'_y, D'_x, D'_y of all quad hashes. These extrema are found in a linear

1 For lack of a better short name. Read qaws as "quad point A and width W"

pass over all hashes by keeping the running minima and maxima. The aabb of reference hashes is necessarily smaller or equal in size to the hash space.

The tree construction is a recursive process to subdivide the reference records according to the spatial extents of their axis aligned bounding boxes. A subdivision is performed by splitting the bounding box into two boxes, which is done by choosing a split axis as well as a split position along that axis. Then, the quad records array is modified in-place, such that a split results in two partitions of the array, where each partition consists of the quad records that lie in the sub-split child of the parent bounding box. This can be done efficiently with a pass of swap operations over the data points. We implement this step using the Hoare partitioning scheme. An example of an arbitrary box split for a 4-ary BVH is shown in Figure 5.3.

Intuitively, if we think of quicksort, the entire tree construction is quite similar in that the choice of the split position would resemble the choice of the pivot element. As in quicksort, the resulting two partitions are created according to the split position only, but the elements within either of the two partitions remain unsorted.

To obtain a high quality tree, we construct the tree guided by an approximation of the surface area heuristic (SAH), as proposed by Wald (2007).

5.4.1 *Surface Area Heuristic (SAH)*

The surface area heuristic builds on the fact that the chance for a random point in the search space to intersect an aabb is proportional to the surface area of this aabb. To efficiently construct trees in this way, Wald (2007) proposes an approximation of the surface area heuristic (SAH), which is computed as follows:

$$cost = n_l a_l + n_r a_r \tag{5.1}$$

In Equation (5.1), n_l, n_r is the number of primitives in the two boxes that result from the split (with $n_l + n_r = n$), and a_l, a_r denote the surface area of the boxes. The split-candidate with the lowest cost is then used to perform the split and reorder the contained primitives into their partitions.

The *binned* approximation of the SAH avoids computing all possible split positions in order to determine the lowest cost. Rather, it evaluates the $N - 1$ split positions that represent the borders of N uniform bins over the chosen split-axis.

This process is shown in Figure 5.4. The figure shows an example (in two dimensions) of the SAH binning of the first split-candidate positions on the x-axis, using $N = 8$ bins, resulting in $N - 1 = 7$ split positions to evaluate the SAH.

The tree is now constructed recursively using SAH guided splits in a greedy, top-down fashion.

5.5 TRAVERSAL

For each query point, we aim to efficiently access nearby reference hashes that are stored in the reference database. Once the tree is constructed, this is done by traversing the tree structure to its lowest level, the leaf nodes, where we store the indices of referenced records that will then be passed to the filter chain, together with the query quads. Note that there are no actual leaf nodes for this kind of tree (see Section 5.2), but there exist nodes that are *marked* as leaves. For simplicity we will still refer to these nodes as leaf nodes.

As we aim to perform a range-search, we represent the query points as search ranges simply by constructing an aabb around each query hash such that the boxes are centered at the query hash (see Figure 5.2). This query-aabb descends the tree structure during traversal. The specific path is determined by the results of query-aabb vs treenode-aabb intersection tests for inner nodes. The aabb intersection tests are simple and can be implemented efficiently: aabbs store the minimum and the maximum value for each axis in our four dimensional hash space. An aabb is intersected by another aabb if their ranges overlap for all axes. Given two aabbs box_a, box_b, the intersection tests whether the maxima of box_a are greater or equal to the minima of box_b, and whether the minima of box_a are smaller or equal to the maxima of box_b. Upon valid intersections, query aabbs are subsequently tested against the aabbs in corresponding child nodes.

For the implementation we adapt the method proposed by Tsakok (2009). Instead of traversing the tree for each query point in isolation, the idea is to create groups of query points that together traverse the tree. The purpose is to exploit hidden spatial coherence without the need to first spatially sort the query points, i.e. query points that are spatially close will share a common part of the traversal-path through the tree structure. Even if it turns out that there is no hidden spatial coherency of query points, the algorithm reduces the number of scattered memory accesses to the nodes, and amortizes the cost of node fetching over a larger number of intersection tests. Each group of query hashes traverses the tree in breadth first fashion, and collects lookup results. We then have collected the set of all record ranges whose aabbs are overlapping with the query point aabbs of the current group.

Query quad records consume 24 bytes. The data structures are identical to reference quad records, with the exception that there exist no file-IDs for query points. Query quads are partitioned into groups of size s quad records, where we separate the hashes from the points A, W and keep these in individual contiguous regions of

memory, the *hashes* and *qaws*, as described for the tree construction data layout. Before finally passing the results to the filter-chain, we have to perform one last cleanup step: the so-called leaf-boxes may be larger than the query boxes, thus we have to filter the referenced hashes according to whether these actually lie within the bounds of the query box. To do so, we iterate over the referenced hashes and create a dense contiguous memory region of points that lie inside the query-aabb, while moving invalid points outside of the region, using a SIMD stream compaction approach. After that, the valid hashes reside in the region in continuous fashion.

Before we continue to traverse the next query quad group, we pass the valid reference hashes to the filter stage, which computes the scale transform factors, and filters the results according to coarse and fine pitch coherence results. Valid matches are collected in a list, and the traversal proceeds with the next group. When all groups are processed, the collected results are passed to the match-sequence estimation stage.

5.6 CONSTRUCTION AND TRAVERSAL PARAMETERS

Tree construction is guided by parameters for the SAH, and parameters that constrain the splitting of nodes. We approximate the SAH using 16 bins, i.e. we have to evaluate 15 split-positions. Nodes are split if the extents of any aabb-dimension is larger than 0.005, and if nodes contain more than 64 elements. A run time comparison for two trees that differ in this minimum number of elements is given in Chapter 6, where we measure the traversal run times for values 32 and 64.

For our implementation, trees with nodes that contain at most 64 elements give the lowest query run times.

Traversal parameters include the search range and the group size. We set the time and frequency related search distances to 0.0115 and 0.0035, respectively. The group size is set to 32.

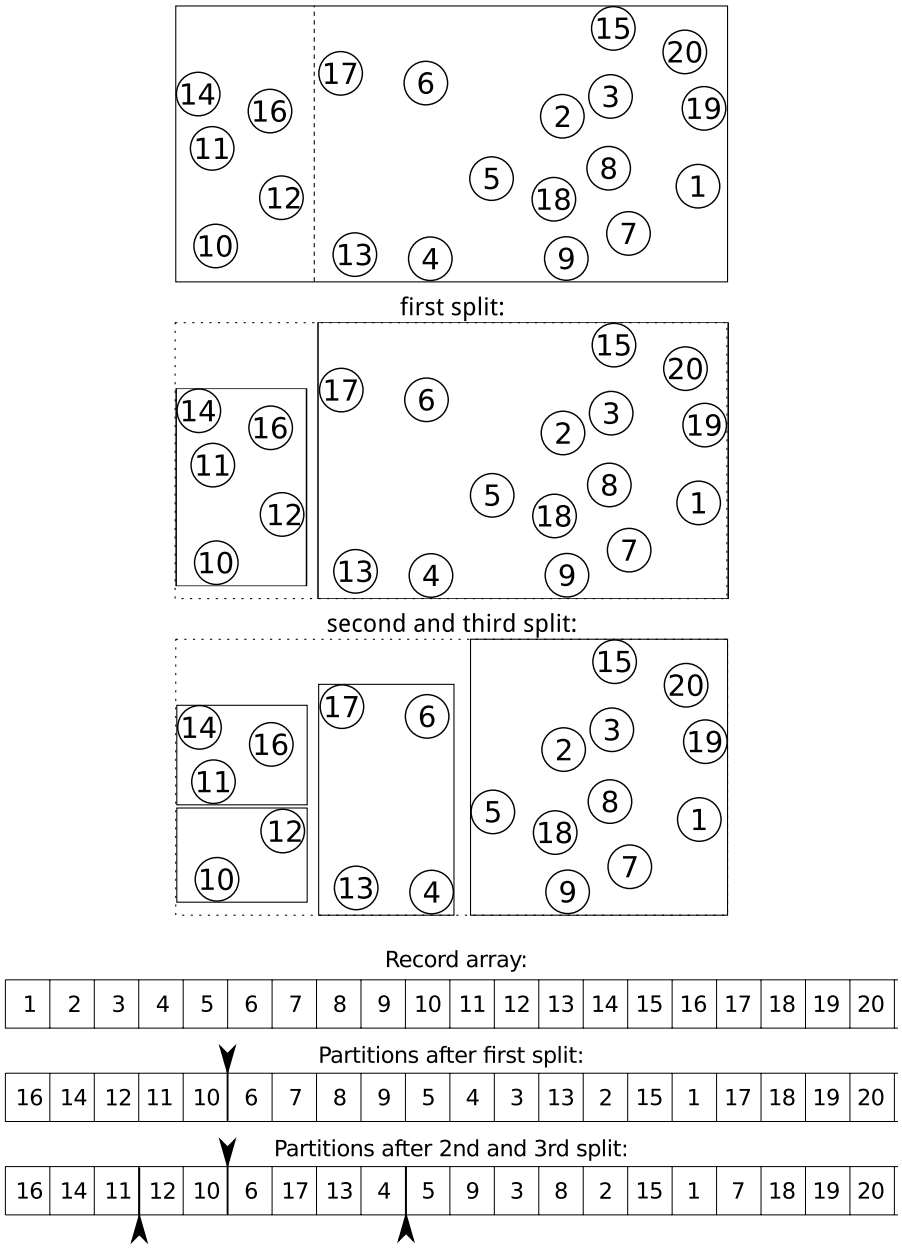


Figure 5.3: Example of a split operation for a 4-ary BVH. The numbered circles represent data points in two dimensional space. The top box shows the aabb for the collection of data points, where the widest axis is chosen as split axis. The split position is shown as a dotted line. The second box shows the result of the first split, and the third box shows the results of the second and third split. The corresponding partitioned record arrays obtained with the Hoare partitioning scheme are shown below. This example shows arbitrary, non-optimal split positions.

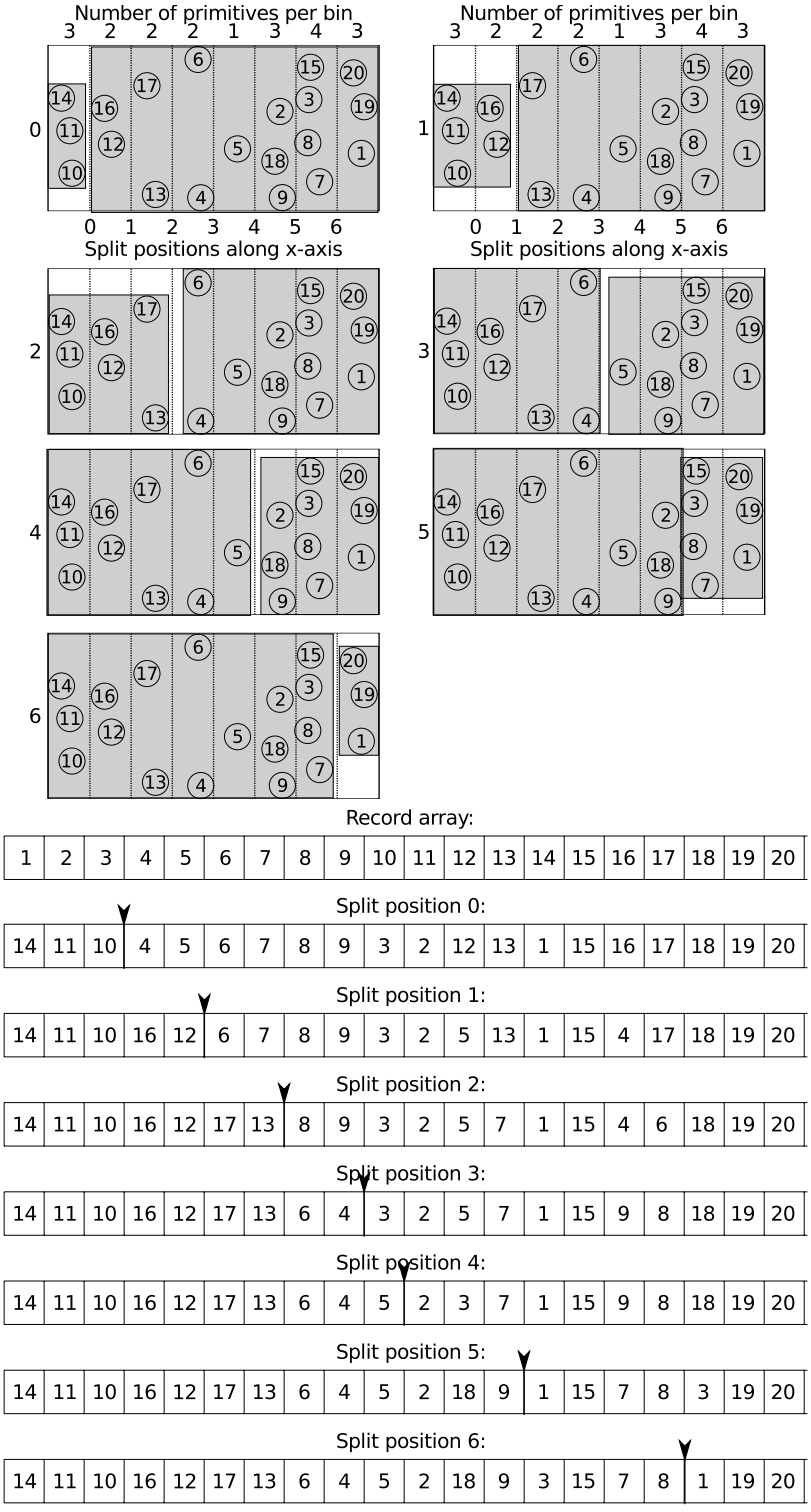


Figure 5.4: Evaluating split positions for the SAH sweep over the x-axis. The record partitions for each split candidate are shown in the lower half of the figure. The arrows denote the start of the record range of the second box.

6 EXTENDED EXPERIMENTS

In previous chapters we explained the method that we propose in detail, and gradually refined the internal data representation, the quad grouping and search algorithm. In this chapter we now show the scalability and performance of the refined method on a larger reference database:

In Chapter 4, we tested Qfp on a reference collection of 100,000 tracks, for *P*-, *T*-, *TP*–modifications in the range of $\pm 30\%$ (see Section 4.6.1), using 300 tracks that are systematically modified. In addition we demonstrated the effect of various values of the qps parameter (see Section 4.6.2). Results on query lengths from 2.5*s* to 20*s*, along with run time measurements, are described in Table 4.1. Noise robustness was tested in the range of -10 to $+50$dB. The robustness of our method to various effects is given in Table 4.2. The specificity on the database we used in Chapter 4 is roughly 0.96%.

To highlight the scalability of Qfp, in this present chapter we re-run these experiments against our large reference collection of 430,000 tracks. In addition we extend the experiments: We elaborate on the possibility of tracking duplicate songs. This information may then be used by the evaluation program. Then, we compare Qfp's noise robustness to that of Audfprint, show the capability of Qfp to identify queries that are scale-modified in a non-linear fashion, e.g. constant acceleration in time and its equivalent in pitch, and perform run time measurements using different search-tree parameterizations.

In order to facilitate reproducible research, and to allow interested parties to compare their methods to Qfp, we publish all information necessary to recreate the database and the query files from the freely available tracks. Information on how to recreate the our reference collection and query audio modifications can be found at `http://www.cp.jku.at/datasets/fingerprinting/`.

In this chapter we evaluate the performance using a rich set of experiments that may help to get a deeper understanding of its strengths and potential weaknesses. However, we cannot exhaustively test all cases, as for example acceleration of audio that is already *P*-modified at $+5\%$, while being subject to an echo effect. The number of queries for exhaustive experiments grows very large, and the results are hard to visualize. As we already mentioned in the introductory sections of this thesis, in the following chapters we will investigate the performance of Qfp on actual DJ mixes, rather than using manually crafted test cases.

In Section 6.1 and Section 6.1.2 we report the specific parameter settings used for the experiments in this chapter. In Section 6.2 we re-run the experiments as shown in Section 4.6 on our large reference collection. In Section 6.2.1 we test the robustness of Qfp on various effects, and in Section 6.2.2 we show the robustness to non-linear modifications of the time and frequency scale. In Section 6.2.4 we test the performance of Qfp for various query snippet lengths, and Section 6.2.5 shows an extensive experiment for the effect of choosing different numbers of query-quads per second on the identification performance. In Section 6.2.6 we test the specificity of Qfp using 100,000 queries from a different reference collection.

6.1 REFERENCE DATABASE AND PARAMETERIZATION

We extend the previously used reference database of 100,000 freely available tracks from the Jamendo service to a number of almost 430,000 full-length tracks (to be accurate, the database contains precisely 429,741 tracks). All the data are freely available, licensed under the Creative-Commons license and are accessible via the Jamendo service. The list of tracks as well as additional information on how to recreate the reference collection can be found at `http://www.cp.jku.at/datasets/fingerprinting/`.

The total length of the audio files amounts to just over 29,515 hours of music (or equivalently, 3.37 years worth of non-stop music), with an average track length of 247.25*s* and a median track length of 223.3*s*.

To create the reference database, and to process query data, we compute the STFT on monaural signals sampled at 8kHz, using an FFT window size of 1024 samples, and a hopsize of 32 samples.

We try to group 7.5 reference quads per second from extracted peaks. The bin-size for grouping an almost uniform amount of quads over time is set to 5 seconds. For comparison, in Chapter 4, where we evaluate the system on 100,000 tracks, we extract 9 reference quads per second.

The peak extraction max filter spans 157 time frames, and 81 frequency bins. The min filter sizes remain at 3 time frames and 3 frequency bins. The center of the quad grouping window is located 1 second from a root point, and the quad grouping region spans 0.9 seconds.

The reference database is created from roughly 430,000 tracks, and results in a collection of 826,107,201 spectral peaks and 757,628,895 quad records. Using the parameter settings that we propose, we realize a hash-to-peak rate of 0.917, which means that we store a lower number of feature combinations (hashes) than the number of extracted spectral peaks. The compressed "quadrecords" consume roughly 20GB

of storage space, the tree has a size of 1.4GB[1] and the spectral peaks consume approximately 9.3*GB*. Altogether the reference data consume 30.7GB. To recall, the quadrecords and the tree are needed to find matches, while the stored spectral peaks are needed for the verification of these matches.

6.1.1 *Duplicate Detection*

An audio collection of this size likely contains duplicate tracks which can negatively influence the evaluation results. To cope with this situation, we extend Qfp with a duplicate detector that automatically keeps track of encountered duplicates. We are aware that automatically generated lists of duplicates are risky in the light of fair evaluation, therefore we present the evaluation results *without* making use of this information, i.e. results with a wrong file-ID are strictly treated as false-positive. However, we also add the results obtained from considering the information about duplicates, and will mark these cases with an additional "$^{\text{dup}}$".

The fingerprinter answers queries by returning result dictionaries that contain the following information:

- Reference file-ID
- Average verification score
- Number of matches
- Average transformation factors for pitch and time scales
- The matching segment in the reference audio, with its start position and duration.
- The matching segment in the query audio

In cases of detected duplicates, the fingerprinter will return a *list* of results rather than a single result. Here, the average score, the number of matches, transformation factors and the duration of the reference segment (but not the start point) are equal up to a float epsilon of $1e^{-5}$. These are the trivial cases of encountered duplicates, which are automatically inserted into a duplicate-dictionary.

In other cases, the result values are not equal, but highly similar. This may occur if two audio pieces of the same recording are present with highly different encoding or bit-rates, or if the pieces are very similar – almost identical – versions of the track. In these cases the user is informed, and it is suggested to add the given file-IDs to the duplicates dictionary.

1 The tree consumes 1.4GB if we stop splitting nodes as soon as fewer than 64 hashes are referenced. If we build a deeper tree, and stop splitting with fewer than 32 hashes, the tree consumes 2.6GB

While the system keeps track of (potential) duplicates, the fingerprinter itself is agnostic to this information. Whether or not to use information on duplicates is dictated by a parameter for the evaluation program.

In any case, if duplicates are added or suggested, these were manually checked by listening to the files and by looking at the spectrograms. Nevertheless, we cannot be sure that these are duplicates rather than *versions* of the original track with just minor differences, therefore we present the results that are obtained with information on duplicates in separate columns of Table 6.3 and Table 6.4.

6.1.2 *Query Processing Parameters*

The query processing parameters are set as follows. As in previous experiments, we let tolerances $\epsilon_p, \epsilon_t = 0.31$, i.e. we want to identify audio if the corresponding reference is not farther off than $\pm 31\%$ in the time and pitch scale. The search-range for the near neighbour search in the hash space is given by an axis-aligned hyper-rectangle (centered at the query point) with a width of 0.0115 and a height of 0.0035. Candidate sequences of at least 4 matches are passed to the verification stage. In addition, we require the candidate sequence to cover at least 15% of the query audio snippet. This is to avoid sequences that span a very short segment, and especially in the presence of highly repetitive and percussive content this constraint helps to reduce the workload in the subsequent verification stage.

Verification parameter settings use an alignment region that spans 14 time frames, and 3.5 frequency bins. The verification considers reference peaks in a window of $2.5s$ around the root point of a match candidate. A sequence is allowed to consist of matches that have a verification score of at least 0.53.

In all experiments that we describe in the following sections, if not stated otherwise, we compute $q = 1500$ query quads per second (qps), and operate on query excerpts of 20 seconds in length.

6.2 SCALABILITY: METHOD EVALUATION ON 430,000 TRACKS

In this section we try to investigate the scalability and applicability of our proposed fingerprinting method to large scale tasks, for which we make use of our large database.

The first experiment tests all scale modifications between 70% and 130% in steps of 5 percentage points. This will serve as our default experiment, and consists of 11,700 queries (three types of scale modification using 13 values thereof for a number of 300 queries). The result is depicted in Figure 6.2, where we choose to split the results into three figures for the *P-*, *T-*, *TP*–modification types. To obtain the overall performance, we accumulate all tp, fp, fn. From these we then

compute the accuracy and precision. In this experiment, Qfp has an overall accuracy of 0.959, and an overall precision of 0.984. Note that this reflects a *pessimistic* scenario, because in our experiments we assume that all three kinds of scale modifications and all tested extents thereof are equally likely to be encountered, which is highly unlikely in actual application scenarios. In the larger experiments shown below, an experiment like this will serve as one single data-point.

In the subsequent experiments we always test scale modifications of three types: speed, tempo and pitch. For each of those, we test 13 values, from 70% to 130% in steps of 5 percentage points.

The efficiency of the matching process can be demonstrated by computing the fraction of candidate file-IDs, out of the set of all returned candidate file-IDs. For processing the candidates we use the early-exit strategy as described in Section 4.5.3. The fraction of processed file-IDs is shown in Figure 6.1. Even though our reference collection is roughly 4.3 times as large as the one we used in Chapter 4, the increased amount of reference tracks does not noticeably influence the result set processing.

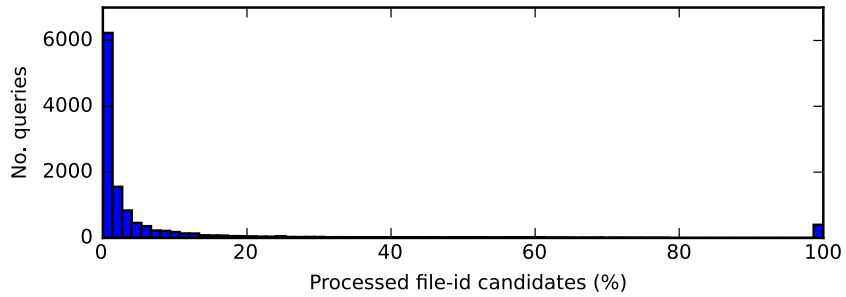
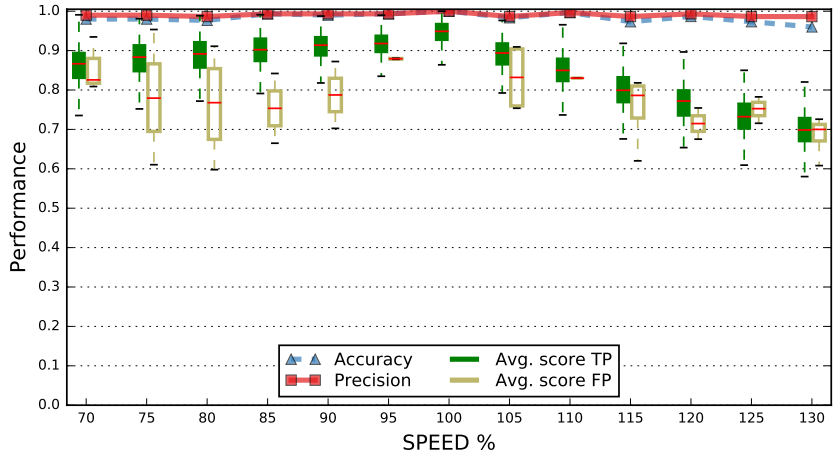
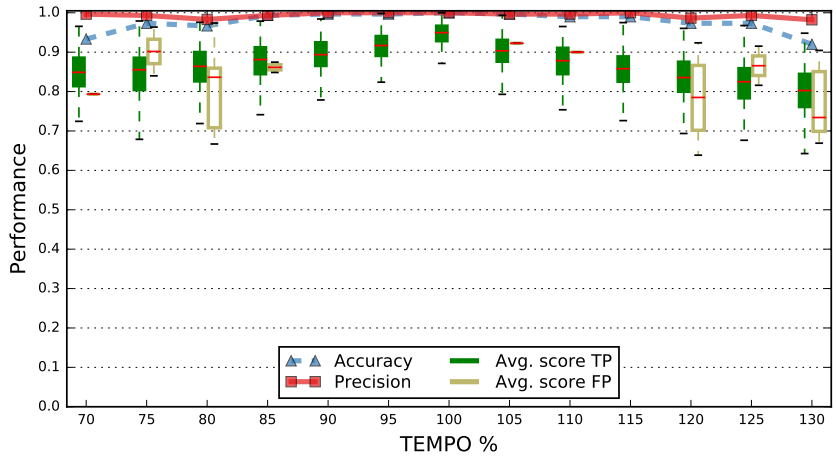


Figure 6.1: Histogram of the fraction of *candidate* file-IDs out of the set of retrieved file-IDs) that had to be processed to answer a query, computed from 11,700 queries. This figure highlights the effectiveness of the early-exit behaviour of the matching process. Note the bin to the right. It represents weakly matched queries and false positives.

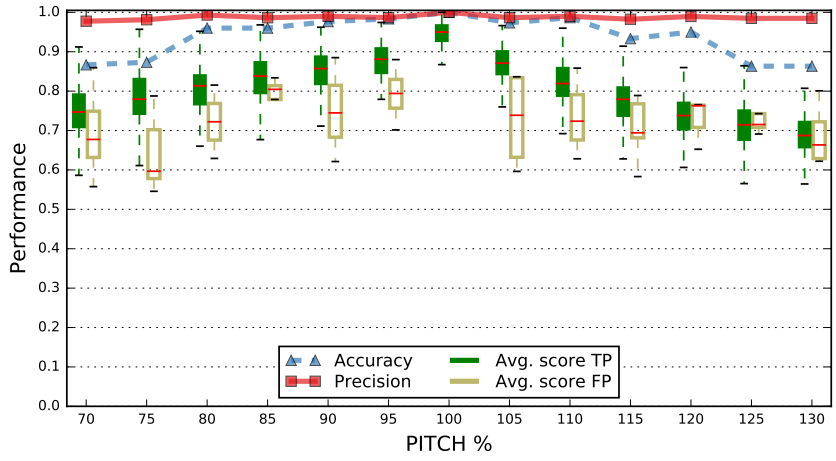
The time spent in processing stages using the refined method is summarized by Figure 6.3. The figures show the impact of two different tree construction parameterizations, one that stops splitting nodes as soon fewer than 32 hashes are contained, and the other stops splitting if fewer than 64 nodes are contained. The third figure shows the second tree version using just 2 worker threads instead of the typically used 7 worker threads. The processing time when using 2 threads averages at around 1 second per query, while it takes in average roughly 1.6 seconds to process a query when 7 threads are active. This difference in processing time is mostly due to cache thrashing,



(a) Speed (*TP*-modifications)



(b) Tempo (*T*-modifications)



(c) Pitch (*P*-modifications)

Figure 6.2: Speed, tempo and pitch modifications in the range of ±30% on the database of roughly 430,000 tracks.

i.e. using more threads causes the eviction of cache lines that would still be needed by other threads.

6.2.1 *Robustness to Various Effects*

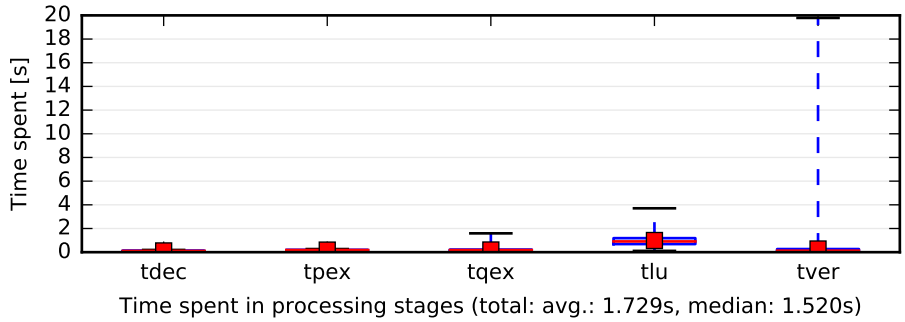
Table 6.1 compares the results to the results that we obtained from experiments on the smaller reference collection of 100,000 tracks given in Table 4.2. The overall precision is slightly lower, while the overall accuracy is increased. The slightly lower precision seems to be caused by the larger number of reference tracks (roughly 4.3 times as many tracks), while the higher accuracy is due to the changed parameter setting of Qfp as described in Section 6.1 and Section 6.1.2. A visualisation of spectrogram excerpts of audio that was modified with these effects is given in Figure 6.4.

	430,000 Tracks		100,000 Tracks	
Effect type	prec.	acc.	prec.	acc.
Bandpass	.993	.999	.997	.993
Chorus	.951	.713	.986	.687
Echo	.987	.980	.993	.980
Flanger	.953	.813	.984	.827
GSM	.919	.608	.978	.440
Tremolo	.993	.990	.990	.977

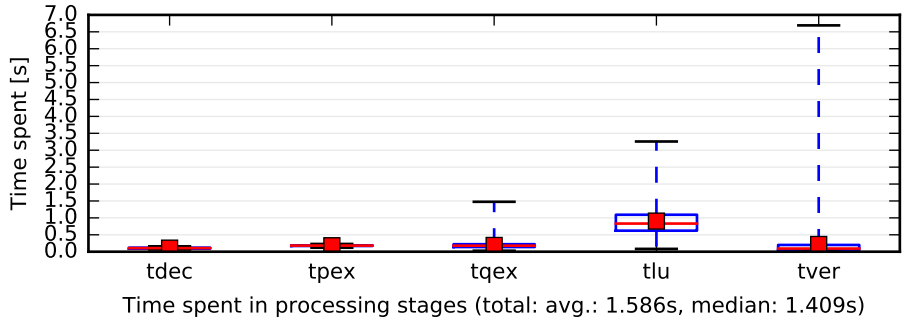
Table 6.1: Performance on queries (without scale modifications) that are distorted by various effects ($qps = 1500$). A visualisation of the corresponding spectrograms is given in Figure 6.4.

6.2.2 *Robustness to Non-Linear Scale Modifications*

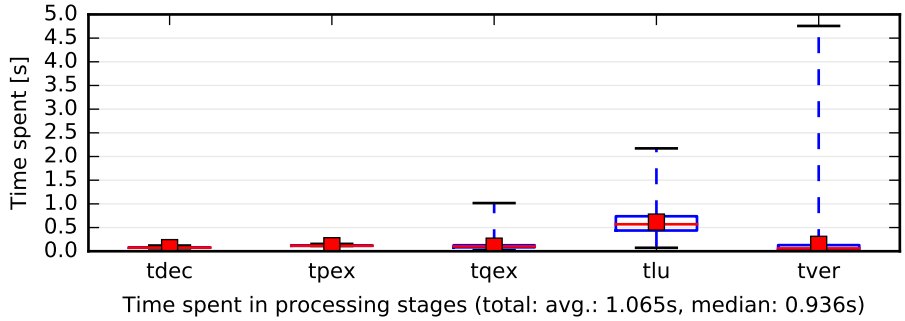
In previous experiments we extensively tested the method on T, P, and TP-modifications in the range of $\pm 30\%$. In these tests we assume a linear, constant scale modification of either type. In DJ mixes, however, we potentially encounter non-linear modifications such as acceleration or deceleration in time, or the equivalent effect along the frequency axis. We argue that for copy detection systems and for media monitoring of digital music content the robustness to acceleration or deceleration is important. For example, a copy detection system that is known not to be robust to a certain degree of non-linear scale modifications can be circumvented by crafting a copy of a track that is perpetually accelerated and decelerated by minuscule amounts. We are not aware of other work in the audio fingerprinting domain that includes an experiment on non-linear modifications. To create the query excerpts we use the sliding window scale effects of the audio



(a) Average time per query that is spent in the processing stages, when 7 parallel processes are used. The timings of the stages represent the mean over 11,700 queries. The tree stops splitting nodes if $n < 32$ points are referenced.



(b) Average time per query that is spent in the processing stages, when 7 parallel processes are used. The timings of the stages represent the mean over 11,700 queries. The tree stops splitting nodes if $n < 64$ points are referenced.



(c) Average time per query that is spent in the processing stages, when 2 parallel processes are used. The timings of the stages represent the mean over 11,700 queries. The tree stops splitting nodes if $n < 64$ points are referenced.

Figure 6.3: Average time spent by a query in the processing stages. The average time is computed from 11,700 queries (3 types of modifications with 13 values of 300 snippets) using 7 worker threads (top and middle figure), or 2 worker threads (bottom figure). For the top figure, a deeper tree is used, such that splitting is stopped when less than 32 hashes are referenced in a node. The middle and bottom figure use a tree that stops splitting when less than 64 elements are referenced.

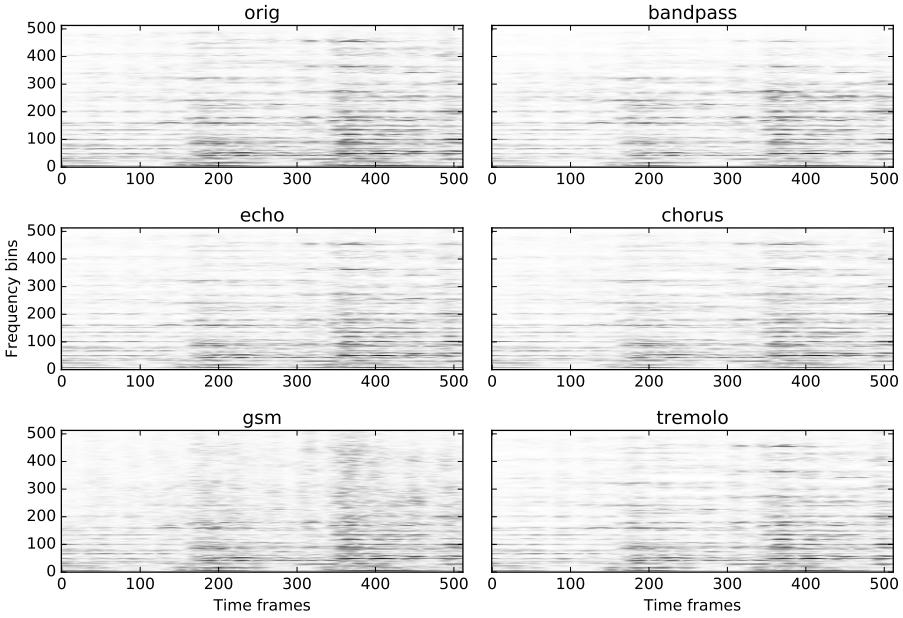


Figure 6.4: Visualisation of the effects described in Table 6.1. The spectrograms show roughly two seconds of audio.

processing tool "Audacity[2]" and specify the initial and final value of *P*-, *T*-, *TP*–modifications. For example, a *T*-scaling value of -50% means half the tempo, i.e. double the length. A value of $+100\%$ refers to double the tempo, or half the duration.

To demonstrate the severity of the non-linear modifications, in Figure 6.5 we visualize the original spectrogram and the corresponding modified spectrograms of the audio snippet we already used in Figure 1.1. Note that the audio snippet length is 20 seconds for this example, in contrast to the 15 seconds in Figure 1.1. The start position of the audio snippet is the same in both figures, however in this experiment the duration of the query audio is 5 seconds longer.

Using our typical query excerpts, we create non-linear scale modifications using two values, $\pm 5\%$ and $\pm 10\%$. For a given modification value of $\pm x\%$, each query excerpt of 20 seconds in length is modified such that the effect takes a value of $-x\%$ at the start position of the query and reaches a value of $+x\%$ at the end position of the query excerpt. Note that non-linear modifications (*P*-, *T*-, *TP*–modifications) from e.g. -10% to $+10$ over the course of 20 seconds are *extreme* and very noticeable to a listener. The results of this experiment are listed in Table 6.2 and Figure 6.5.

Given the severity of the non-linear scale modifications, Qfp's precision and accuracy turns out to be surprisingly high, notably so for the cases where a 20 second query is accelerated from -5% to $+5\%$ in either dimension. We attribute Qfp's robustness to non-linear scaling to a combination of two design choices: the range search that we use

2 Audacity is available at https://www.audacityteam.org

to obtain match-candidates, and the sequence estimation stage (see Section 4.5.2) that accepts sequence candidates even if there exists a certain variance within the estimated scale transform factors.

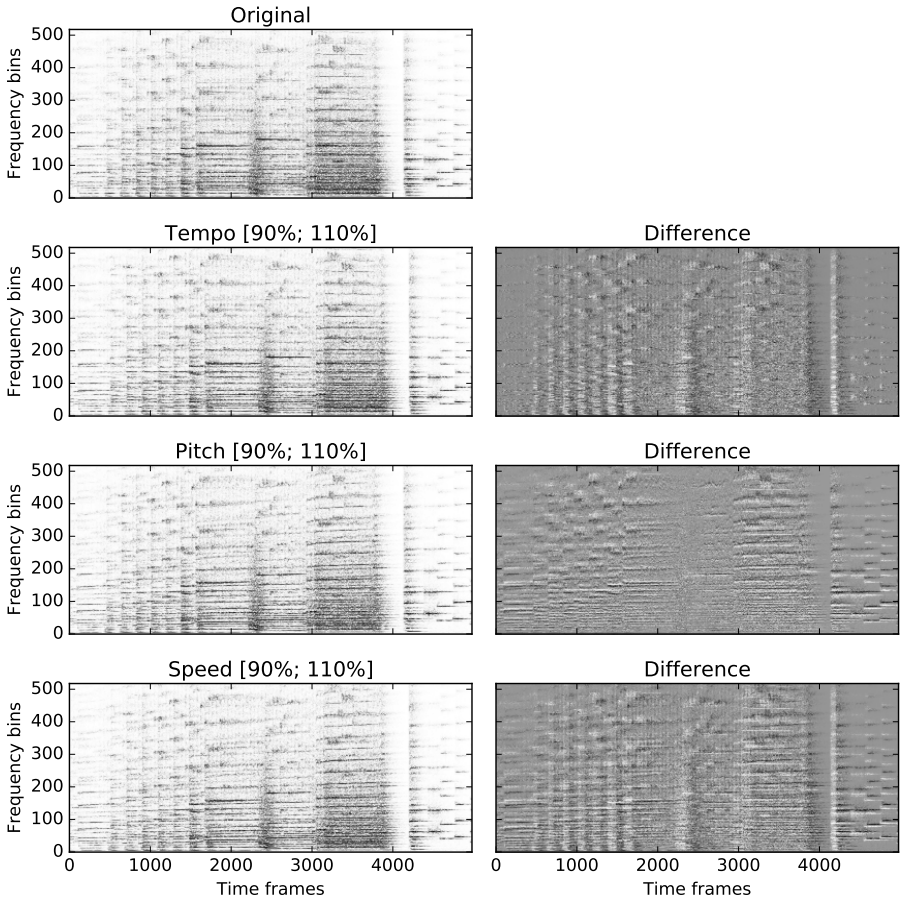


Figure 6.5: Non-linear scale modifications in the range of $\pm 10\%$. The rows correspond to the type of scale modification. The left column shows the spectrogram content of the modified piece. The right column shows the difference to the original.

6.2.3 *Noise Robustness*

We here demonstrate the noise robustness of Qfp. The queries are distorted with additive white noise such that 13 versions of each query excerpt are created. The 13 values of signal to noise ratios (SNR) are in the range of $[-10; 50]$dB, with a step of 5dB. The experiments in two parts: the first experiment tests the noise robustness of our method *without* adding additional *P*-, *T*-, *TP*–modifications. This experiment will serve as a baseline for the second experiment. For comparison we also show the results of the fingerprinting implementation "Audfprint" (see Section 7.2 below). Audfprint is a peak-based method that implements what is known about the "Shazam"-algorithm, but also has to make assumptions about parameters and implements additional

concepts. It groups quantized pairs of spectral peaks and uses these as keys for a lookup-table. The method is generally considered as highly robust to noise. However, because the it is not robust to even the smallest scale changes, we here test its performance on queries that are not *P-*, *T-*, *TP*–modified. In this experiment, for Audfprint we use a very small reference collection of just 1023 tracks (because we simply want to establish a baseline). It consists of 723 tracks of a dataset that we describe in Chapter 7 below (the "Mixotic"-set), and the 300 references that correspond to the 300 query pieces, to make these identifiable. The results are shown in Figure 6.6. Qfp performs at higher accuracy for SNR values down to +5dB, but performs noticeably worse for lower SNR values. The precision of both methods is equal down to SNR values of +10dB. For lower SNR values, the precision of Audfprint is lower than that of Qfp. Note that Qfp returns fewer false positives than Audfprint, even though its reference collection is more than 420 times as large. Further, using its default settings, Audfprint targets a rate of 20 hashes per second, Qfp targets a hash rate of 7.5 hashes per second. That is, Audfprint uses roughly 2.67 times as many hashes, and Qfp operates at a lower false negative rate for the SNRs down to +5dB.

The second experiment considers *P-*, *T-*, *TP*–modifications in the range of ±30% in addition to the 13 values of noise distortions (152,100 queries in total). The results for this experiment are shown in Figure 6.7. The Qfp results that we obtained in the first experiment are added to the figures as a baseline.

6.2.4 *Performance for Various Query Excerpt Lengths*

We here show the performance of Qfp for query snippets between three and twenty seconds in length. The purpose of this experiment is to gather data-sheet like data, that should help readers to decide whether Qfp might be applicable to their use-cases if specific query-lengths are of interest. If a use-case considers very short queries only, the performance for this case can be increased by extracting more than our average of 7.5 hashes per second of reference audio.

The results are given, along with run times, in Table 6.3. We also show the results of the duplicate-aware evaluation, that counts cases where a file-ID is returned that is different from the expected file-ID as *tp*, if the duplicate dictionary contains that pair of file-IDs. The performance measures are the total average over *P-*, *T-*, *TP*–modified queries with the given query length. Thus, each row in Table 6.3 represents 300 queries for 3 types of modification over 13 values of the modifications. From these 11,700 queries per row, we dismiss 600 queries such that unmodified queries are counted exactly once (i.e. from the 900 unmodified queries of *P-*, *T-*, *TP*–modifications, we only count 300 unique unmodified queries).

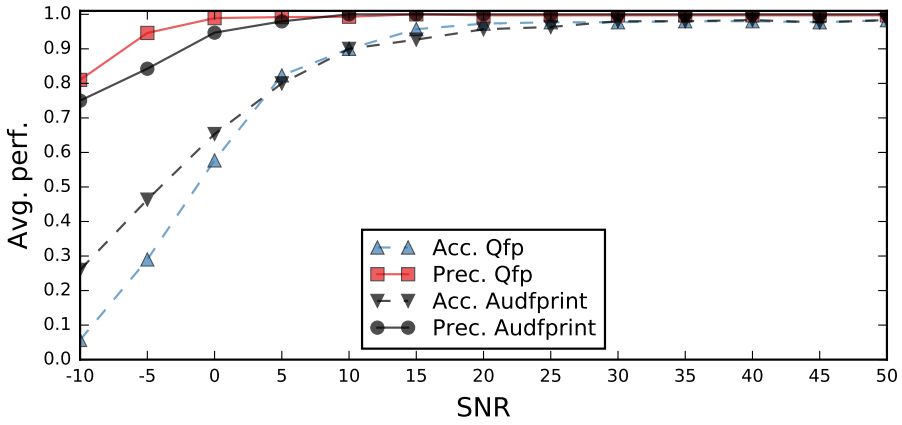


Figure 6.6: Robustness to white noise in the range of $[-10; 50]$dB, for queries that are *not* *P*-, *T*-, *TP*–modified. The figure shows a comparison of Qfp (square and upward-triangle markers) to the peak-based method "Audfprint" (circle and downward-triangle markers), which is considered to be exceptionally robust to noise. Note that Qfp operates with high precision on 430,000 tracks (with 7.5 hashes per second) while for Audfprint we use a small example-collection that contains a subset of 1023 tracks (with the default of 20 hashes per second).

The results demonstrate that the precision is barely impacted by the length of the query audio excerpt. The accuracy drops with shorter query lengths, which is expected since the time-dimension carries important information, i.e. the match sequence estimation needs to find a number of match-candidates that form a sequence over time. For ten seconds of query audio, Qfp achieves an average accuracy of over 86%, which includes all the *P*-, *T*-, *TP*–modified queries. While we believe that this demonstrates excellent performance, we highlight that for use-cases where shorter queries have to processed we suggest to increase the hash-rate for reference feature extraction. This will result in higher accuracy for short queries.

6.2.5 *Performance for Varied Numbers of Query-Quads*

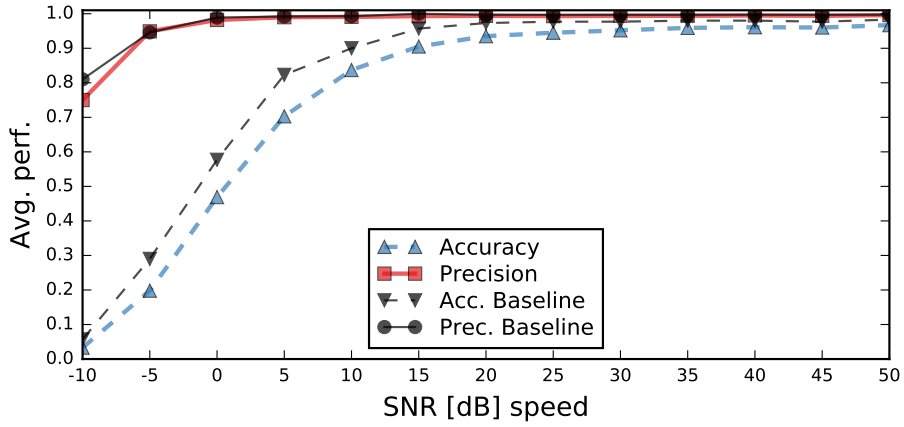
Here, we repeat the experiment for varying the number of query quads per second (qps), similar to the experiment as given in Figure 4.9, to gain insight if the larger reference database has a negative impact on the results, and to find out if the current parameterization of Qfp improves its performance. Compared to Figure 4.9, the overall performance slightly increased, especially for the low numbers of qps (in the higher number ranges of qps there is not much room left for improvement). For the low value of 250 qps the accuracy for *P*- and *TP*-modifications is increased by roughly 3 percentage points. We

Initial value	Final value	Type	Accuracy	Precision
−5%	+5%	*T*	0.89	0.982
		P	0.81	0.98
		TP	0.903	0.993
−10%	+10%	*T*	0.81	0.987
		P	0.433	0.963
		TP	0.47	0.972

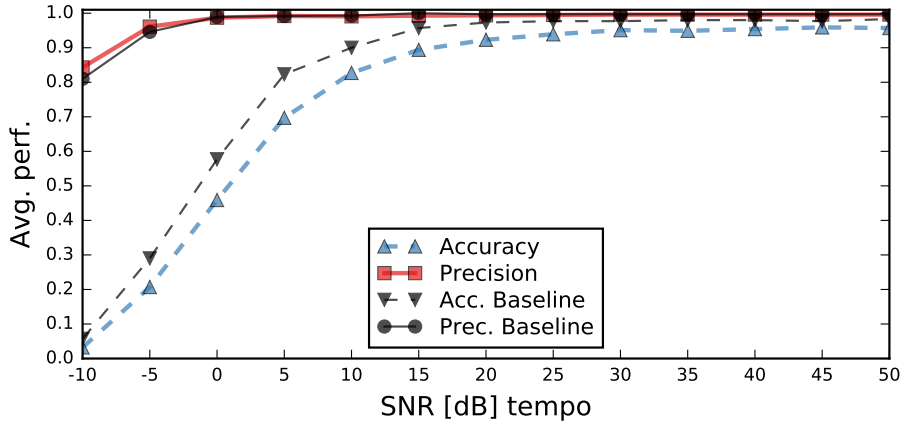
Table 6.2: Performance on 20*s* queries with non-linear scale modifications (constant "acceleration").

	11,100 queries / row				11,700 queries / row	
qlen [s]	acc.	prec.	acc.^{dup}	prec.^{dup}	mean t [s]	median t [s]
3	0.214	0.925	0.217	0.942	0.28	0.21
4	0.418	0.962	0.423	0.974	0.32	0.26
5	0.565	0.973	0.570	0.983	0.39	0.31
6	0.674	0.972	0.682	0.984	0.52	0.41
7	0.746	0.976	0.755	0.987	0.57	0.47
8	0.793	0.976	0.803	0.988	0.65	0.53
9	0.832	0.976	0.843	0.989	0.76	0.63
10	0.863	0.978	0.873	0.990	0.82	0.69
11	0.885	0.980	0.895	0.991	0.90	0.76
12	0.902	0.980	0.913	0.992	1.00	0.85
13	0.913	0.981	0.924	0.992	1.07	0.92
14	0.922	0.981	0.933	0.993	1.14	0.98
15	0.932	0.983	0.943	0.994	1.25	1.08
16	0.939	0.982	0.950	0.993	1.31	1.15
17	0.946	0.983	0.956	0.994	1.38	1.21
18	0.950	0.983	0.960	0.994	1.48	1.30
19	0.954	0.983	0.965	0.994	1.54	1.37
20	**0.957**	**0.983**	**0.968**	**0.994**	1.59	1.41

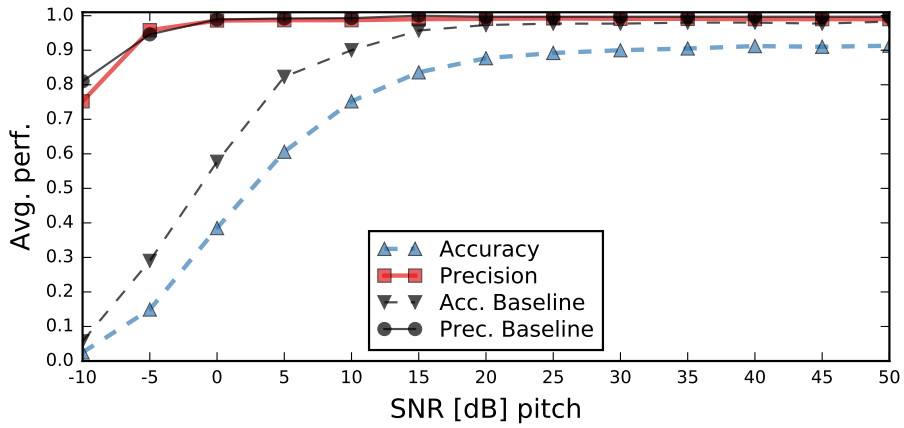
Table 6.3: Performance on query lengths (qlen) in the range of $[3;20]$ seconds. The accuracy is given in columns "acc." and (duplicate-aware) in "acc.^{dup}", the precision is given in columns "prec." and "prec.^{dup}". The third and fourth column represents the results when using duplicate track information. The last row (bold) represents the common experiment using 20 second queries. The two rightmost columns show the runtime mean and median.



(a) SNR and speed modifications



(b) SNR and tempo modifications



(c) SNR and pitch modifications

Figure 6.7: Robustness to white noise for SNR levels from −10dB to 50dB, with *P*-, *T*-, *TP*–modifications in the range of ±30%. The thin black curves represent the baseline (see "Acc. Qfp" and "Prec. Qfp" in Figure 6.6). For the blue and red curves, each pair of points shows precision and accuracy average over 3900 queries. The experiment consists of a total number of 152,100 queries (3 types of modifications for 13 SNR values of 3900 queries).

do not observe any negative impact caused by the larger number of reference tracks.

6.2.6 *Specificity Experiment*

The specificity is about the number of false positives in cases where the correct reference is not present in the reference collection. Thus, in these cases a system needs to refrain from claiming a match, in order to operate at high specificity.

To test the specificity of our proposed method, we need to create queries from pieces of audio that are not represented in our reference collection. Therefore, we create the queries from a different, proprietary collection that represents songs of a wide variety of genres. This collection consists of 104,011 tracks. However, as described in Section 4.6, it is hard to guarantee that the collections are indeed disjoint. If we query each track of this collection (with 20 second excerpts starting at second 20) against our reference database, we get a total of 8144 matched file-IDs. Assuming that the collections are disjoint, all these matches are fp_s. This results in a specificity of 0.922.

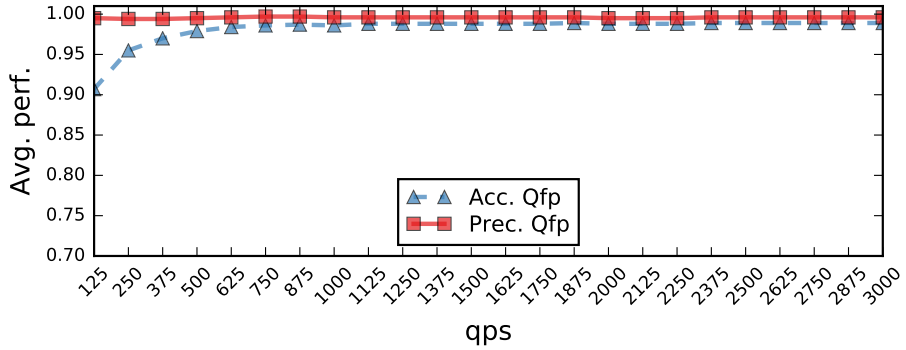
However, we know that the test collection itself contains duplicates, which means that we now potentially count some fp_s more than once: if the fingerprinter returns a match for a given query-track, it will necessarily return the same match for each duplicate of that query-track, resulting in an unfair negative impact on the specificity. To circumvent this problem, we first perform a duplicate detection on the test-collection, and then dismiss all fp_s results that correspond to duplicate queries. This results in a number of 7897 matches. Discarding these duplicate queries, Qfp has a slightly higher specificity of 0.924.

Because we assume that the query collection and reference collection might not be disjunct (in contrast to what we hoped for), we have to investigate the results in order to find out whether some of the fp_s results might be *tp* after all. To do so, we group the remaining results (which all should be *fp* if the collections are disjoint) by their verification scores and uncovered transformation factors, and listen to the results. The histograms of the verification scores and the uncovered scale factors of these false positives are visualized in Figure 6.9.

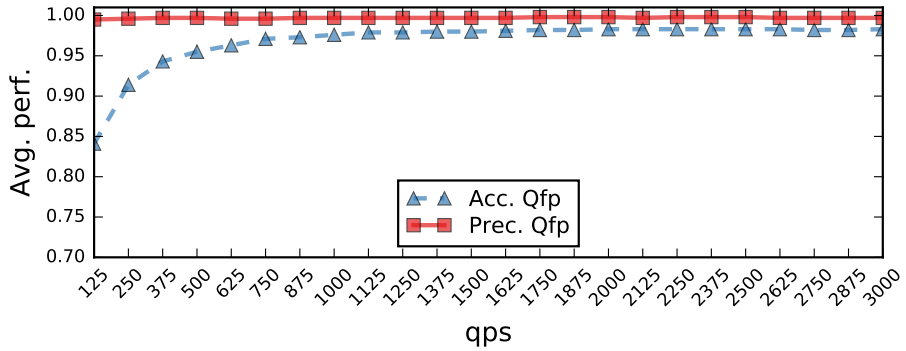
The listening test uncovers interesting results, and in many cases points out possible mistakes by the users of the Jamendo service who seem to have uploaded content which is not available under the Creative Commons License, or vice versa, there seem to be cases of audio-content that is said to be copyrighted material (as it appears in our test-collection of copyrighted material) while in fact it is not.

Let us list a few of the roughly 350 examples that we listened to ("ID" refers to the unique Jamendo file-ID):

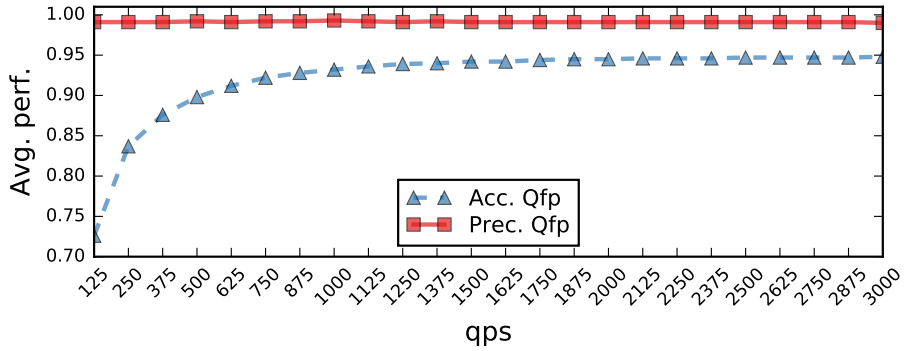
- ID 1093751 at second 161 plays a song by James Brown



(a) Speed and qps variations



(b) Tempo and qps variations

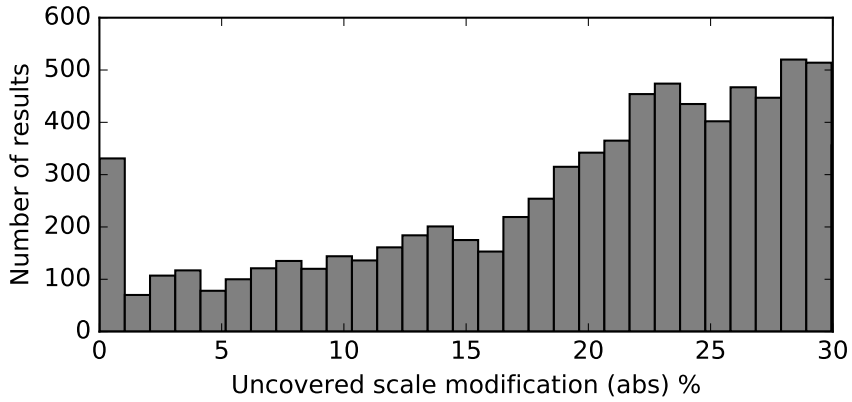


(c) Pitch and qps variations

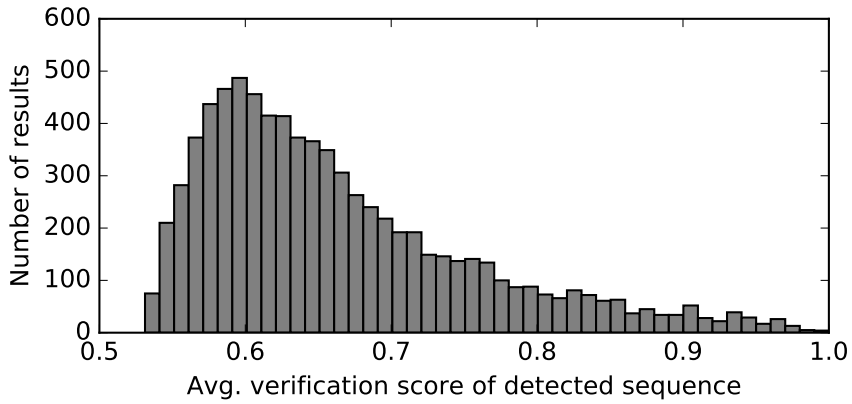
Figure 6.8: Precision and accuracy for *TP*-, *T*- and *P*-modifications (from top to bottom) on our reference database of 430,000 songs. Figures 6.8a to 6.8c (note the y-axis range) show results for a total of 280,800 queries, for 24 values of quads per second (qps, parameter q) in the range of $[125;3000]$ in steps of 125 qps, and a query snippet length of $20s$. Each pair of data points shows the average precision and accuracy of 3900 queries for scale modifications in the range of $\pm 30\%$ of the individual type.

	11,100 queries / row				11,700 queries / row	
qps	acc.	prec.	acc.^{dup}	prec.^{dup}	mean t [s]	median t [s]
125	0.808	0.983	0.816	0.994	0.59	0.55
250	0.887	0.983	0.897	0.994	0.72	0.65
375	0.916	0.983	0.926	0.994	0.84	0.74
500	0.931	0.984	0.941	0.994	0.93	0.82
625	0.941	0.984	0.951	0.994	1.04	0.92
750	0.948	0.985	0.958	0.995	1.12	0.99
875	0.950	0.984	0.961	0.995	1.23	1.09
1000	0.953	0.984	0.963	0.995	1.30	1.15
1125	0.956	0.984	0.966	0.995	1.38	1.23
1250	0.956	0.984	0.967	0.994	1.48	1.31
1375	0.957	0.984	0.968	0.995	1.55	1.37
1500	0.957	0.983	0.968	0.994	1.60	1.44
1625	0.958	0.984	0.969	0.994	1.69	1.50
1750	0.959	0.984	0.970	0.995	1.75	1.55
1875	0.960	0.984	0.971	0.995	1.82	1.60
2000	0.960	0.983	0.971	0.994	1.89	1.66
2125	0.960	0.983	0.971	0.994	1.92	1.70
2250	0.960	0.983	0.971	0.994	2.00	1.76
2375	0.960	0.983	0.971	0.995	2.01	1.77
2500	0.960	0.983	0.972	0.994	2.10	1.84
2625	0.960	0.983	0.971	0.994	2.12	1.85
2750	0.960	0.983	0.971	0.994	2.15	1.88
2875	0.960	0.983	0.972	0.994	2.18	1.91
3000	0.961	0.983	0.972	0.994	2.21	1.92

Table 6.4: Performance on 20 seconds long queries with varied numbers of query quads per second (qps), in the range of $[125; 3000]$ qps, in steps of 125 qps. The table shows the total averaged accuracy and performance (evaluated with and without awareness of duplicates). The average is obtained by accumulating the *tp*,*fp*,*fn* for *P*-, *T*-, *TP*–modifications. The accuracy is given in columns "acc." and (duplicate-aware) in "acc.^{dup}", the precision is given in columns "prec." and "prec.^{dup}". The two rightmost columns show the mean and median run time.



(a) Histogram of claimed scale modifications (absolute values)



(b) Histogram of verification scores of reported sequences

Figure 6.9: Reported scale modification factors (Figure 6.9a) and verification scores (Figure 6.9b) of the reported results (false-positives) of the specificity experiment.

- ID 1050228 plays a song by the band Rednex
- ID 733687 seems to make use of the beats and melody of a song by Dr. Dre and Eminem.
- ID 786051 contains beats and melody of a Snoop Dogg and Dr. Dre track.
- ID 979780 makes use of content by Beyonce.
- ID 828010 likwise uses content by Lady Gaga
- ID 1077380 uses beats and melody by Gentleman
- ID 594815 might be from a movie scene where people have a conversation in a bar, while faintly in the background the song “Love her Madly” by “The Doors” is played (around second 98).

Having listened to roughly 350 of such cases we point out that the specificity of 0.924 reached by our method represents a pessimistic value, due to the overlaps between the two data-collections.

Finally, we want to shortly elaborate on the run time impact on non-identifiable queries: on average it takes a bit longer to process a query that does not have a true reference in the reference collection, compared to cases where a query is identifiable. This is because all match-candidates have to be processed in order to report that no valid match could be determined. In a figure like Figure 6.1, all correctly processed queries of this specificity experiment will end up in the rightmost bin. Because of the increased workload, the average query run time of a query used in this present experiment is 2.11*s*, and the median run time is 1.55*s*, while queries of the same length that in fact are identifiable can be processed on average in 1.59*s*, with a median run time of 1.41*s* (see Table 6.3).

7 DJ MIX MONITORING: METHOD COMPARISON

In this chapter we assess the fitness of three peak-based audio fingerprinting systems with different properties on real-world data – DJ mixes that were recorded in discotheques and clubs.

Not surprisingly, the amount and variety of incorporated signal modifications shows that identification of tracks in DJ mixes is a highly challenging task. The performance gap between evaluations on manually crafted test data and evaluations on actual DJ mix sets can be considerable. To address this evaluation issue, and to enable the research community to evaluate systems on DJ mixes, we also create and publish a freely available, creative-commons licensed dataset of DJ mixes along with their reference tracks and song-border annotations[1]. Experiments on these datasets reveal that our method, together with the refined search algorithm as described in Chapter 5 achieves considerably higher performance on this task than the other methods.

This chapter is heavily based on our third work in the field, in which we finally apply our matured system to the target application domain: media monitoring for discotheques.

> R. Sonnleitner, A. Arzt, and G. Widmer (2016). "Landmark-Based Audio Fingerprinting for DJ Mix Monitoring". In: *17th International Society for Music Information Retrieval Conference (ISMIR 2016)*

The chapter is organized as follows. In Section 7.1 we introduce the datasets that are the basis for the experiments and analysis and interpretation of results. Section 7.2 gives an overview of the methods we test in this work. Then, in Section 7.3 we describe the setup of experiments and their evaluation. An analysis of the different properties of the tested methods is given in Section 7.4.

7.1 DATA SETS

We perform experiments on two different datasets, called disco set, and mixotic set. The disco set is a non-free dataset that we unfortunately cannot make publicly available. Therefore, we compile and annotate a second dataset, the mixotic set, which is freely available for research purposes. We think that the mixotic set may be useful to the research community, and could help to design well balanced identification systems and to uncover specific strengths and potential shortcomings

1 The mixotic-dataset is available at http://www.cp.jku.at/datasets/fingerprinting/

of various methods, therefore we publish the mixotic set along with the annotations.

For both datasets we have a larger number of reference tracks than the number of tracks that are actually played in the performances. The superfluous reference tracks allow to investigate if a system can discriminate tracks even in the presence of larger reference collections. Furhter, the reference collections are incomplete. For the disco set, we were not supplied with a complete reference collection, and for the mixotic set we could not find all the references for played tracks. However, this reflects a realistic use-case for media monitoring: fist, we cannot expect to have all the references to all songs that could possibly be performed, and second, we will almost always use a larger reference collection than the number of tracks that are played in a single monitored event. Due to the missing reference tracks we can evaluate an extremely important property of automated systems: their capability of not answering a query if the query piece is not represented in the reference database. A system that successfully refrains from claiming a match in these cases operates at high specificity.

In these datasets, the DJ mix recordings will serve as query content, and the collected reference tracks serve as the reference database. In the following we introduce these datasets, and summarize their properties in Table 7.1.

7.1.1 *The non-free Disco Dataset*

The first dataset, the *disco set*, contains eight mixes that were performed in discotheques, and digitally recorded from the DJ mixing desk. The duration of the mixes is approximately 7 hours and 16 minutes. For this dataset we have 296 reference tracks, only some of which are actually played in the mixes. The genres of the mixes include pop and rock, electronic music and German folk.

Because of copyright reasons, we cannot make the disco set publicly available.

7.1.2 *The Mixotic Dataset for Research*

We compile a second dataset, called *mixotic set*. We created this dataset from free, CC-licensed DJ mixes that were published on the mixotic netlabel[2]. We manually collected the reference songs via web-search, and integrate those which are available under the same license in the dataset. Currently, the mixotic set consists of 10 mixes with a total duration of 11 hours and 23 minutes. For this dataset we collected a set of 723 reference tracks, 118 of which are actually played in the mixes. According to the artists, this set contains genres like Techno, Chicago House, Deep-Tech, Dub-Techno, Tech-House, and the like. To

2 Mixotic is accessible via http://www.mixotic.net.

Disco	tracks	ref.	$+[s]$	$-[s]$
set0	25	18	5661	2179
set1	12	12	3760	0
set2	12	11	3206	294
set3	11	4	1054	2006
set20	19	17	3123	457
set35	20	7	324	996
set36	28	13	872	768
set37	21	10	720	720
total: 8	148	92	18 720	7420

Mixotic	tracks	ref.	$+[s]$	$-[s]$
set044	14	14	4640	0
set123	12	12	3320	0
set222	18	11	3543	2097
set230	9	7	2560	780
set275	17	11	3398	1622
set278	12	11	3576	284
set281	18	15	3300	280
set282	14	8	2200	1740
set285	15	15	4540	0
set286	14	14	3140	0
total: 10	143	118	34 217	6803

Table 7.1: Data set properties of the disco set (top) and the mixotic set (bottom). The column "tracks" gives the number of played tracks in the DJ mix, "ref" denotes the number of these tracks that are present in the reference database, and the columns "$+[s]$, $-[s]$" hold the number of seconds of referenced audio and not-referenced audio for the individual DJ mixes.

be able to evaluate the fingerprinting results, we annotated the song borders of the tracks that are played in the individual mixes. Due to the long fading regions and sometimes very homogeneous track transitions, these annotations cannot be exact. We tried to mark the positions in time where the previous track is fully faded out.

7.2 OVERVIEW OF METHODS: AUDFPRINT AND PANAKO

We use the datasets that we described in the previous section to experiment with the following three methods: *Audfprint*, *Panako* and our own proposed method, the quad based audio fingerprinter *Qfp* that we already discussed in great detail.

An evaluation of experiments using Audfprint and Panako (on other audio data) is given in Six and Leman (2014). While all three methods are peak-based, the systems employ different inner mechanisms and thus are expected to perform differently on the datasets used in this thesis. Note that we use Audfprint and Panako as published, without tuning to the task at hand. We do this because we believe that the methods are published with a set of standard parameters that turned out to be well suited for general use cases according to experimentation performed by their authors. Likewise, we use the *same* set of parameters for Qfp, as they are described in Chapter 4, and incorporate the improvements that we described in Chapter 5. For the task at hand, we want to investigate the fitness of the underlying algorithms of the methods, rather than discussing their specific implementations.

AUDFPRINT Audfprint is an MIT-licensed implementation[3] of a peak-based audio identification algorithm based on the method described by Wang (2003). The published algorithm utilizes quantized hash fingerprints that represent pairs of spectral peaks. The hashes are described by the time-frequency position of the first peak and its distance in time and frequency to the second peak. The hashes that are computed from a snippet of query audio are used as the keys into a suitable reference data structure, e.g. a hash table, to retrieve reference hashes with the same key. For each query hash, a lookup is performed and the result sets are collected. Matched query and reference hashes which happen to have a constant time offset in their individual peak-time identify the reference audio, along with its position in which the query snippet could be located.

PANAKO Panako (Six and Leman, 2014), available[4] under the GNU Affero General Public License is a free audio identification system. It transforms the time domain audio signal into a two dimensional time frequency representation using the Constant Q transform, from

3 Audfprint is available on https://github.com/dpwe/audfprint.
4 Panako is available on http://www.panako.be/.

which it extracts event coordinates. Instead of peak pairs, the method uses triples, which allows for a hash representation that is robust to small time and pitch scale modifications of the query audio. Thus, the system can also report the scale change factors of the query audio with respect to the identified reference. The system was evaluated in (Six and Leman, 2014) on queries against a database of 30,000 full length songs, and on this data set achieves perfect specificity while being able to detect queries that were changed in time or frequency scale of up to around 8%. In this work we use Version 1.4 of Panako.

7.3 EXPERIMENT SETUP

Disco-set			identifiable [s]					non-identifiable [s]		
$+[s]$	$-[s]$	M.	tp	fp	fn	**acc.**	**prec.**	tn	fp	**spec.**
18720	7420	**A**	7838	7440	3442	0.419	0.513	3611	3809	0.487
		P	4624	5596	8500	0.247	0.452	5539	1881	0.746
		Q	13879	1253	3588	**0.741**	**0.917**	6996	424	**0.942**
		Q^v	14316	1523	2881	0.765	0.904	6587	833	0.888
		Q^ϵ	3423	152	15145	0.183	0.957	7413	7	0.999
Mixotic-set			identifiable [s]					non-identifiable [s]		
$+[s]$	$-[s]$	M.	tp	fp	fn	**acc.**	**prec.**	tn	fp	**spec.**
34 217	6803	**A**	21783	10233	2201	0.637	0.680	1735	5068	0.255
		P	12326	16181	5710	0.360	0.432	2371	4432	0.349
		Q	29985	1262	2970	**0.876**	**0.959**	6304	499	**0.927**
		Q^v	30445	1680	2092	0.889	0.948	4395	2408	0.647
		Q^ϵ	19497	349	14371	0.570	0.982	6715	88	0.987

Table 7.2: Evaluation results for the data sets. The column "+" shows the number of seconds of the DJ mix, for which a reference is present. The column "−" likewise gives the number of seconds for which no reference track is present. The methods (M.) Audfprint, Panako and Qfp are abbreviated as "A", "P" and "Q". The column "Q^v" shows Qfp results without the verification stage, and "Q^ϵ" shows results for the reduced search neighbourhood. "acc." is the accuracy, "prec." is the precision and "spec." is the specificity (see Section 1.7). The experiment setup is defined in Section 7.3. In this chapter we omit showing the individual statistics of each DJ mix that is contained in the dataset, and directly present the overall values. Detailed results and additional experiments are given in the subsequent chapter.

Experiments are performed individually on the datasets we described in Section 7.1. The general experimental setup is as follows. The mixes are split into non-overlapping query snippets of 20 seconds in length. To create query snippets from the DJ mix we use the tool SoX[5] along with arguments to prevent clipping, and convert the snippets into .wav files.

The methods process each individual, local and independent query snippet, and store the results. The implementations of the three tested systems behave differently in answering a query: if the query excerpt could be matched, Audfprint and Panako by default report the whole query duration as matched sequence. Qfp gives a more detailed answer and reports the start time and end time of the matched portion within the query excerpt. Likewise, as Qfp, Audfprint allows to report the exact part of the query that it could actually match (using the option `-find-time-range`), but for Panako we did not find such an option. For best comparability of the evaluation results, for all of the three methods we assign the reported match file ID to its whole query of 20 seconds. Reporting the exact duration (in seconds) of identified content would seem to be of high relevance if the industry is going to aim for fair revenue distribution. In that case, the duration of played content will impact the assigned revenue, which is not possible if simply binary decisions are made whether a track is played or not.

In this experiment we make the assumption that there exists exactly one correct result per query excerpt, which can result in one of the five types of confusions (tp, fp, fp_s, fn, tn). This assumption may seem flawed, since there could be more than one tracks being played at the same time, for example when two tracks are being overlaid, or in the case of cross-fading. However, we have to follow our assumption of one result per query snippet because of two reasons. First, we do not have an exact ground-truth, as the DJs who created the mixes report a list of tracks that are contained in the mixes, but no information on how these are used to compose the mix. A correct manual annotation of cross-fading events, with exact start and end positions in time turns out to be highly challenging - therefore we mark a single point in time for each transition of a track into the next one. The second reason for our assumption concerns the fairness of the evaluation. The three tested methods behave differently in answering a query. In some cases Panako returns more than one result in ranked order. Audfprint returns exactly one result per query, at least with its default parametrization. If we assume that potentially more than one result may be correct for a given query, we would have to adapt the evaluation method. Audfprint would have to be assigned an additional *fn* for each query with more than one correct result, because it reports exactly one result. For Panako, in this case we would have to assign additional *fp* or *fn*. Indeed, if there exist true cases with more than one

5 SoX is available on http://sox.sourceforge.net/.

correct result, we cannot simply ignore superfluous ranked outputs of either method for queries in which only *one* song is played. Since the three methods behave differently, we believe the most fair and straightforward way is to consider the top result that is returned, and always ignore further ranked results. We further discuss this in the subsequent chapter, where we propose to incorporate more context for the monitoring of DJ mixes.

It is important to note that we do not perform smoothing over time on the individual results but rather test the local and independent *raw* identification performance of each method based on each individual query.

We compare the fingerprinting results to the ground truth on a one second basis, i.e. for each second of the DJ mix we check whether the corresponding query result is correct.

Here we distinguish the following two cases: Case 1 $(C1)$ *identifiable*, and Case 2 $(C2)$ *non-identifiable* portions of the mixes. We investigate how the systems perform in cases where a song is identifiable, because it is present in the reference database $(C1)$, and how well behaving a system is in not producing a match result in cases where this is correct, i.e. because the track is in fact not present in the reference $(C2)$.

For all cases $(C1)$, we count the number of seconds of true positives (*tp*), false positives (*fp*) and false negatives (*fn*). True positives are cases in which the system correctly identified a track from a query. The false positives denote situations in which the wrong track is claimed to be present, and the false negatives are cases in which the system did not report a result at all. For this evaluation there exist no true negatives, i.e. $tp + fp + fn = N$. For this case $(C1)$ we use the performance measures *accuracy* and *precision*.

To assess system performance for cases $(C2)$, in which the reference track is unknown, i.e. not present in the database, we compute a third evaluation measure, the *specificity*.

The identification performance of all three methods is listed in Table 7.2. We will discuss the results in the Section below, and analyze the properties and differences of the methods.

7.4 DISCUSSION OF RESULTS

Table 7.2 summarizes the results of each method on the disco set and the mixotic set (rows Q^v and Q^ϵ become relevant at a later point of this section). For the disco set, the accuracy shows that just between 25% and 74% of identifiable portions were assigned to the correct reference track. This reveals that DJ mix track identification indeed is a tough problem. The precision values show that Audfprint and Panako claim a wrong track in around 50% of the identifiable cases. The specificity of the systems shows that Audfprint correctly abstains from claiming a match in roughly 50% of the non-identifiable cases. Panako shows

higher specificity of around 75%. Qfp manages to correctly treat *tn* in 94% of the cases.

The results obtained from the experiment on the mixotic set show higher accuracy for all three methods, and Audfprint and Qfp operate with higher precision than on the disco set. For the mixotic set, all three systems show lower specificity than for the disco set. We believe that this is a result of the larger reference database (723 songs in contrast to the 296 in the disco set) and the highly repetitive tracks in the mixotic set. In general, Qfp performs at higher accuracy, precision and specificity than Audfprint and Panako. Panako shows higher specificity than Audfprint on both datasets.

The low specificity of the algorithm that is implemented in Audfprint indicates that its fingerprints are too general and therefore seem to violate the trait of uniqueness. Panako uses triples of peaks, which inherently capture more specific information of the local signal. Indeed, its specificity on the disco set is considerably higher than that of Audfprint, i.e. its fingerprint descriptors are less general, which may be the reason for it to correctly refuse to make a claim in around 75% of the cases on the disco set, and in roughly 35% of the cases on the mixotic set.

ANALYSIS Qfp performs best on the tested datasets. To find out which properties of the system are responsible for that, we perform two additional experiments. The first experiment is intended to investigate the impact of the verification process, and the second experiment highlights the effect of the range query for Qfp. For a detailed explanation on the parameters that are mentioned in this section, we ask the reader to consult the Chapters 4 and 5.

First, we want to find out if it is the verification process that allows Qfp to maintain high performance. If we switch off the verification[6] and run the experiments, this results in an overall accuracy of 0.76, a precision of 0.90, and a specificity of 0.89 on the disco set. For the mixotic set this results in the accuracy of 0.89, precision of 0.95 and a specificity of 0.65 (c.f. Table 7.2, row Q^v). In terms of accuracy and precision, the results for both datasets are comparable to those with active verification. The specificity on the mixotic set, however, is notably lower.

We now investigate the performance of Qfp using a reduced neighbourhood for the range queries. We argue that this loosely translates to using quantized hashes with the same effect of when a query peak moves with respect to the others, the corresponding reference hash cannot be retrieved. This neighbourhood is specified as distance in the hash-space of the quad descriptor. For this experiment we re-

6 Strictly speaking, the implementation does not allow to switch off the verification. Therefore we instead relax the verification constraints such that no candidate can be rejected.

duce this distance from 0.0035, 0.0115 for pitch and time to 0.001, 0.001 for pitch and time. For the disco set, this results in a low accuracy of 0.18, precision of 0.96 and specificity of 0.99. On the mixotic set, the small range-search neighbourhoods result in an accuracy and precision of 0.57 and 0.98, and specificity of 0.99 (c.f. Table 7.2, Q^{ϵ}). Audfprint achieves roughly 42% accuracy on the disco-set, where Panako achieves roughly 25% accuracy. In this experiment we strongly reduced the search-range of Qfp, and as expected, the resulting accuracy is worse than that of the other two tested algorithms. This is because we have to retrieve four "surviving" peaks for a matching quad, while Panako needs to retrieve three peaks, and Audfprint just two peaks in order to find a single match. If the same peak extractor would be used for all three methods, and the probability of a peak *not* migrating to a neighbouring bin is denoted as p, the probability of retrieving a given hash would be p^2 for Audfprint, p^3 for Panako, and p^4 for Qfp, which fits our observed low performance of Qfp when we omit the range-search.

For the mixotic set we obtain very similar results. Audfprint manages to operate at an accuracy of roughly 64%, Qfp achieves 57% and Panako operates at 36% accuracy. For this dataset the "ranks" of Qfp and Panako switched, but this is due to the low precision of Panako in the presence of highly repetitive tracks, and a low precision necessarily impacts the accuracy.

This experiments highlights the importance of range-search: by using a reasonable search range, migrating peaks are not of any concern.

EXTENDED DATABASE We now add the reference tracks of both the disco set and the mixotic set to our reference database that consists of 430,000 full length tracks (this captures almost the entire Jamendo corpus[7]), and inspect how Qfp copes with this amount of additional tracks. The results are compared to the previous experiment in Table 7.3.

The overall result for the disco set (with standard settings for the range-search and verification) is 0.69 for accuracy and 0.80 for precision. The specificity is 0.71. On the mixotic set, the accuracy is 0.83, the precision amounts to 0.87 and the achieved specificity is 0.56. The low specificity here is also caused by a song duplicate in the DJ mixes and Jamendo corpus, i.e. in the case of mixotic set 282, Qfp could correctly identify the track "Akusmatic - Scamos" within the additional 430,000 songs, but the evaluation treats this as *fp*, because according to the ground truth this track is not present. The issue with song duplicates does not influence any other experiments in this chapter, since we use the extended reference database only with the Qfp method.

7 Jamendo is accessible via https://www.jamendo.com.

Disco-set	identifiable		non-id.
M.	acc.	prec.	spec.
Q	0.74	0.92	0.94
Q_{large}	0.69	0.80	0.71
Q^{part}_{large}	0.60	0.88	0.89
Mixotic-set	**identifiable**		**non-id.**
M.	acc.	prec.	spec.
Q	0.88	0.96	0.93
Q_{large}	0.83	0.87	0.56
Q^{part}_{large}	0.76	0.93	0.80

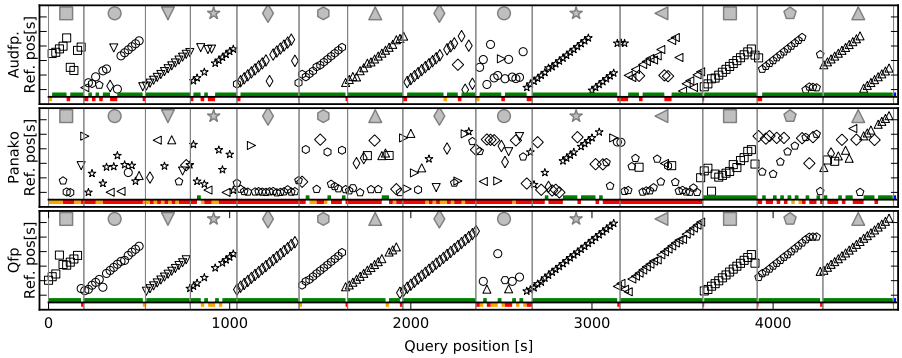
Table 7.3: Evaluation results for the data sets within the large reference collection, compared to the results using the small reference collections. Q denotes the previous results as shown in Table 7.2, Q_{large} denotes the results on the large database of 430,000 tracks, and Q^{part}_{large} denotes the corresponding results when considering the start and end point of the matched query segment rather than the whole 20 seconds.

The experiment shows that there is a certain negative impact, causing more *fp* when trying to identify tracks in DJ mixes on larger databases. Note that these results also depend on the experiment setup as defined in Section 7.3, where we chose to assign the identified track ID to the whole query of 20 seconds in length. If we consider the reported start and end time of identified queries, the results on the disco set give an accuracy of 0.60, precision of 0.88, and a specificity of 0.89. For the mixotic set the accuracy then is 0.76, precision is 0.93 and the specificity results in 0.80.

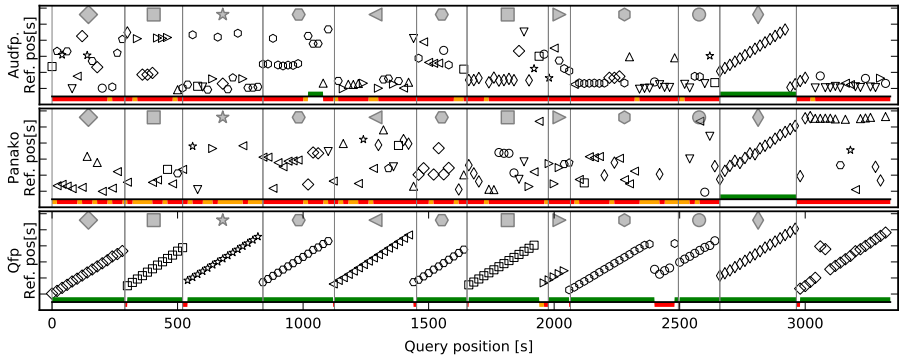
Qfp turns out to maintain – what we believe is – acceptable performance, on a database with 430,000 full length songs. According to precision and specificity, the other methods tested in this work report large numbers of false positives despite the very small reference collection of 723 songs. This leads us to suggest that the monitoring of DJ mixes via automated fingerprinting systems indeed is a highly challenging task.

VISUAL ANALYSIS The different behaviour of the three methods can be conveyed visually. In Figures 7.1, 7.2, 7.3 we show the predictions of the three systems on the mixotic dataset [8]. Vertical lines represent song borders. The figures show the scattered query identification results, where the x-axis position is the query time, and the y-axis position locates the query within the reference song that the system could

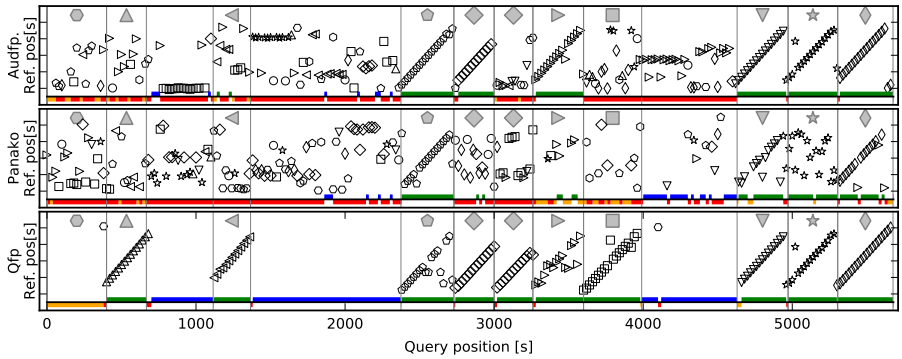
8 The mix-IDs are listed and explained in the published dataset which is available at http://www.cp.jku.at/datasets/fingerprinting/



(a) Set044

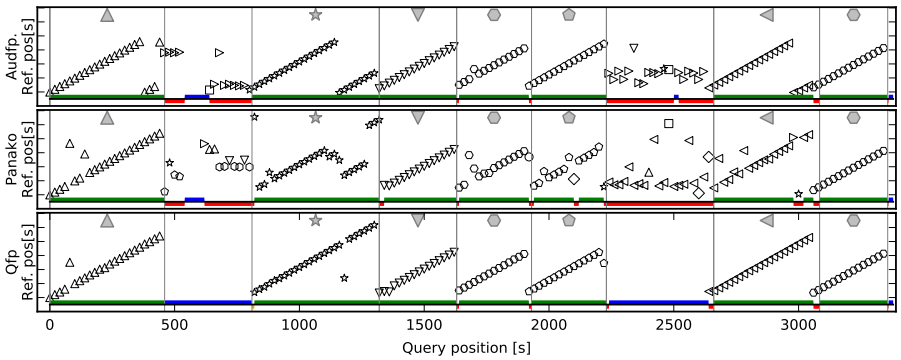


(b) Set123

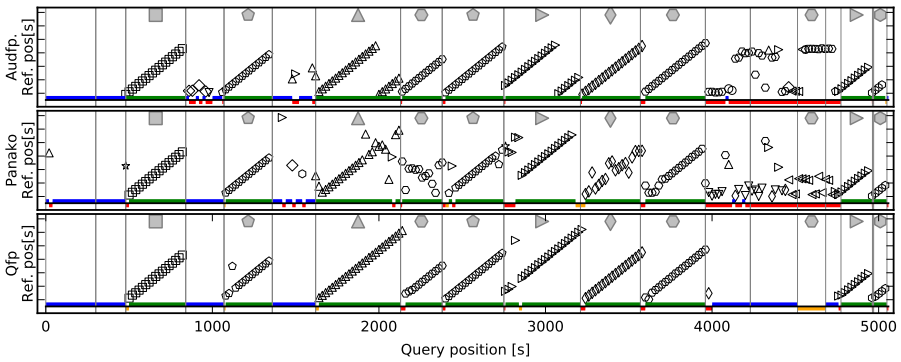


(c) Set222

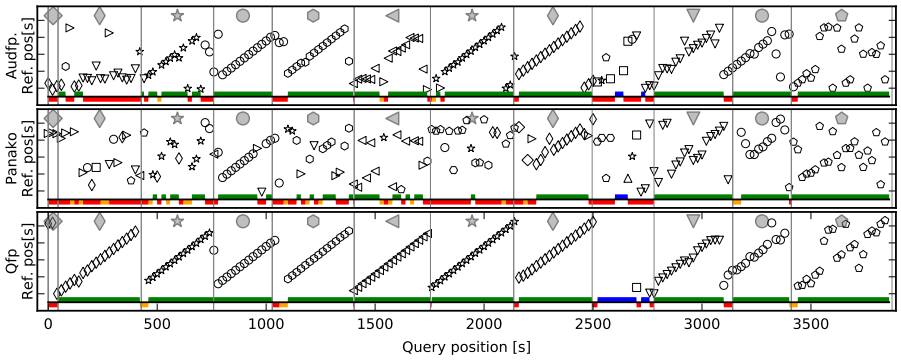
Figure 7.1: Query visualisations for mixotic sets. The rows show the results of individual, non-overlapping 20*s* queries without smoothing of predictions for Audfprint (top), Panako (middle) and Qfp (bottom). The vertical lines are the annotated song borders. The identification claims of the systems are encoded in the shown markers, where each marker represents a reference track. The x-axis position shows the query excerpt position, and y-axis the location of the matched query within the identified reference track. A missing large marker indicates a missing reference track. The figures show a bar at the bottom, which represents the confusions. *tp* (green) and *tn* (blue) are shown on top of the horizontal line, *fp* (red) and *fn* (yellow) are shown below.



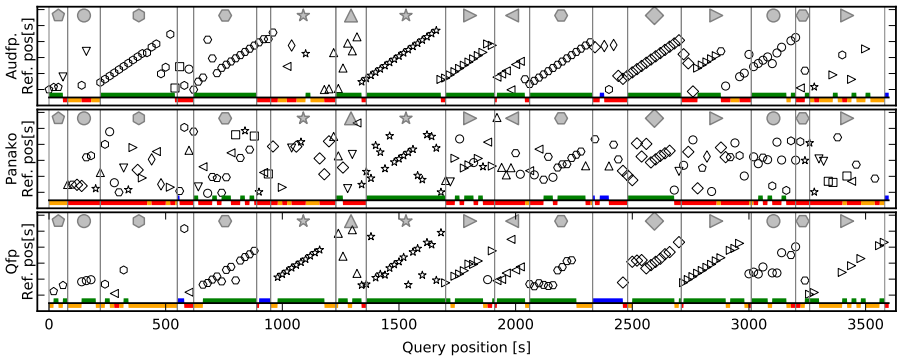
(a) Set230



(b) Set275

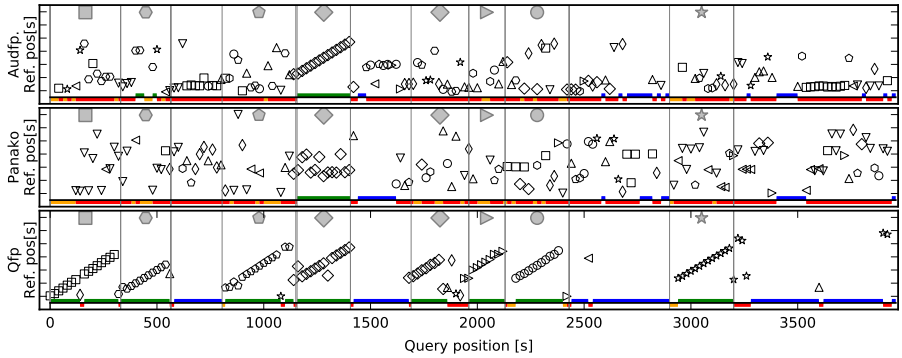


(c) Set278

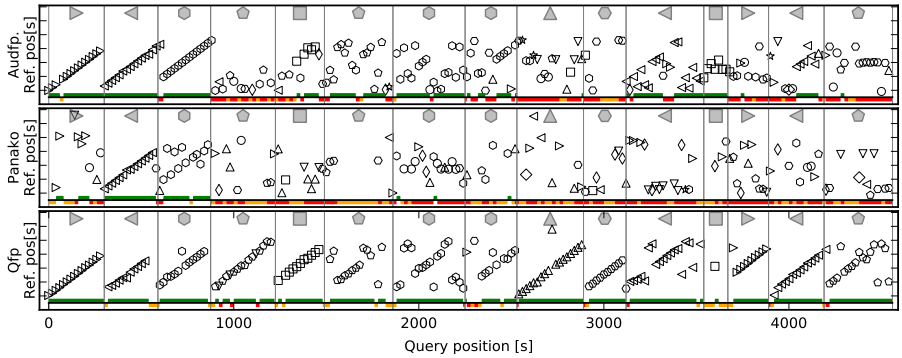


(d) Set281

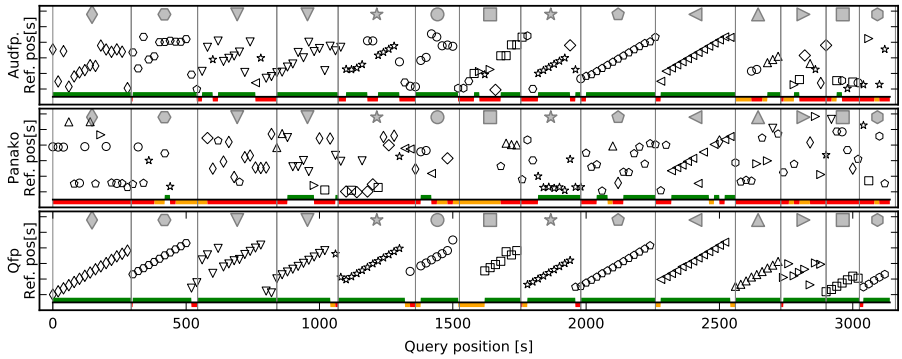
Figure 7.2: Visualisation of mixotic sets 230, 275, 278 and 281. Please refer to Figure 7.2 for explanations.



(a) Set282



(b) Set285



(c) Set286

Figure 7.3: Visualisation of mixotic sets 282, 285 and 286. Please refer to Figure 7.2 for explanations.

identify. Thus, scattered positions of songs that are correctly identified over several successive queries usually take the shape of a sawtooth function. In DJ mixes this will not always be the case, as the DJ can loop content. The different track names are encoded using markers, to be able to see if a system tends to confuse the same two tracks, or whether it reports many different tracks for a portion that it fails to identify correctly. The large markers shown on top, between song borders, are the reference. A missing reference marker means that the song is not present in the database, and therefore is non-identifiable. Note that the evaluation does not consider whether the predicted position within the reference is correct, as this is not meaningful for highly repetitive musical content. If two or more missing references occur after each other, we sometimes omit the song-border between these unknown tracks. We do this since it is hard to annotate the song boundaries without being able to listen to the original references.

The results obtained from the experiments shown in this chapter support the initial premise of this thesis, where we state that automated audio identification on DJ mixes is a challenging problem. We observe that Qfp performs best on the tested datasets, and believe that it constitutes a well suited method to further investigate the analysis of DJ mixes via audio fingerprinting. Further work that goes into more detail of the published mixotic dataset is presented in Chapter 8 of this document.

8

INTERACTIVE AND AUTOMATIC MONITORING OF LONG RECORDINGS

In this chapter we discuss how to build a media monitoring system on top of the fingerprinter. The goal of monitoring is to identify and accurately locate the tracks that are played within a recording. Basically, media monitoring can be seen as a segmentation task: locating a track means to detect and describe its time segment within the query, such that the start-positions and durations of present tracks are revealed. Knowing the duration of identified tracks (in seconds) will pave the way to fair revenue distribution.

We will use a "global sequence detection" algorithm to accomplish that, which is the main difference to what we demonstrated in Chapter 7. The global approach changes the granularity of the matching process as follows. In Chapter 7 we computed and reported the top result for each independent local short query of 20 seconds in length. There, one single query was the entire context based on which the fingerprinter could report results (i.e. local context). In contrast to that, the approach that we propose in this present chapter is to compute all matches of the entire (in our datasets up to 130 minutes long) query recording at once, and then try to uncover sequences to determine time-segments for each identified track within the long query recording. Having computed all match-candidates, the next stage that tries to form match-sequences can operate on global context, thus we will refer to this concept as "global sequence detection". In our experiments, global sequence detection increases the overall identification performance compared to the local query processing approach that we described in Chapter 7.

Global sequence detection can be categorized into the reporting module of our overall architecture, as depicted in Figure 8.1.

Throughout this chapter we mainly make use of our mixotic-dataset (see Section 7.1). It consists of 10 DJ mixes, and 723 reference songs which now are included into our database of 430,000 tracks. All experiments in this chapter are performed on the large database, which in our opinion reflects a quite realistic use-case, considering the volume of the database. We present detailed figures and tables for each mix that is contained in the dataset, to document the current state of our method and to serve as a baseline for future developments. Note that we introduce minor changes to the dataset: the contents of the dataset are as previously described in Section 7.1, however, the total number of seconds is slightly changed. The changes have two reasons: first, we found that in the experiment setup of the previous chapter we

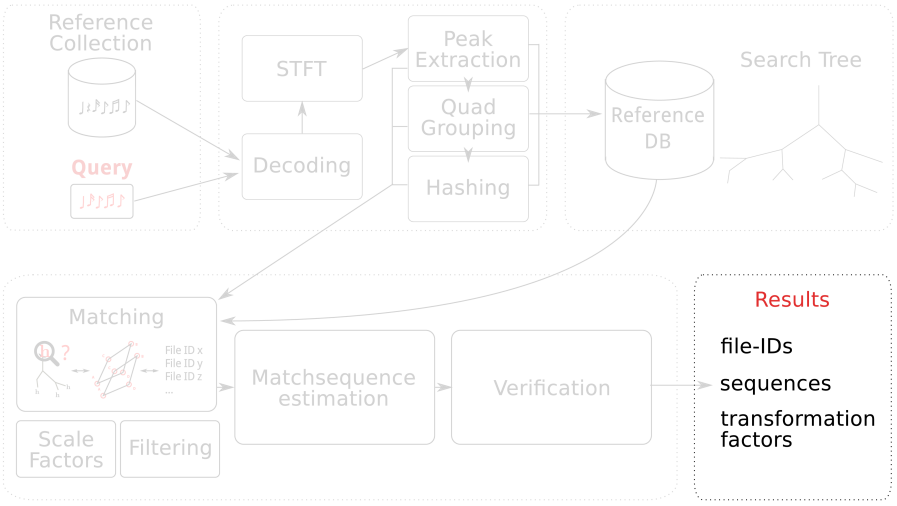


Figure 8.1: Sequence detection as part of the reporting module.

erroneously cut away the last seconds of some sets, and second, in this chapter we process the entire recordings rather than individual query-snippets of 20 seconds in length. The updated dataset statistics are as follows: The mixotic-set encompasses 10 query sets with a total length of 41,231 seconds (roughly 11.5h, and exactly 221 seconds more than in Section 7.1). From the played tracks, a total number of 34,663 seconds is referenced in our database, and the remaining 6568 seconds are unknown (9.6h and 1.8h, respectively).

This chapter is organised as follows. First, in Section 8.1 we describe the approach of collecting global fingerprinting results for the monitoring of long recordings. We here include a simple example to introduce the content of visualisations that we will then use throughout the following sections of this chapter. Then, in Section 8.2 we describe the algorithm that is used to implement our global sequence detector. Next, we describe an *interactive* monitoring setup where the user is in control of the sequence detection parameters. The purpose of this is to help with the manual validation of results.

Section 8.3 then introduces the *automatic* detection of sequences: we fix a set of parameters such that the sequence detector can be used to monitor long recordings without requiring human interaction.

At the end of this chapter, the automatic detection approach is used in a case study, where we process the entire mixotic set and discuss two main points: the scale modifications that we uncover in the individual mixes, and the performance of our proposed system. Given the overall high performance, we will focus on errors that are introduced. The case study is presented in Section 8.4. Finally, we discuss the approach in Section 8.5.

8.1 SEQUENCE DETECTION ON GLOBAL RESULTS

In Chapter 7 we demonstrated how to use fingerprinters to process long recordings by creating successive local short queries that do not overlap. There, each 20*s*-query is processed independently. The best result for this query is returned and all weaker results are discarded. While this approach is simple, the downside is that processing recordings in this way discards potentially useful information concerning the weaker match-candidates and sequences.

Here, instead of independently processing short local queries, the entire recording is passed to the fingerprinter as one large query. To do so, we parameterize Qfp exactly as described in Section 4.5: At least four matches of a candidate file-ID have to be found, and these are required to form a sequence. In addition, their average verification score has to meet a threshold. To collect all the (potentially weak) matches of an entire DJ mix recording, we have to disable Qfp's "early exit" optimization (see Section 4.5.3). The set of collected global results now represents all valid match-candidates that adhere to the parameterized constraints of Qfp.

Of course, deactivating the early-exit results in longer query processing run times as we now have to process all match-candidates for each candidate file-ID that is found in the query recording. However, in our tests where we process DJ mixes between one and two hours in length, the processing time is still shorter than the duration of the query audio, even when using a only a single core. Note that the sequence detector is currently not run time optimized.

The global result consists of valid match-candidates, which essentially are "sequence-candidates", since the constraints of Qfp require at least four match-candidates for a file-ID to form a sequence. These global results are basically sub-sequences of various file-IDs that are scattered throughout the long query, and many of these sub-sequences may be weakly matched in terms of verification score[1]. These global results are now the input to a "global sequence detection" routine that tries to form sequences of higher "quality" (in terms of the number of matches, verification-score, etc.) by making use of the global context of the input. It then reports detailed results for each sequence that could be determined:

- The reference file-ID
- Start-position and duration in seconds
- Average time and pitch transformation factors of the detected sequence with respect to the reference

1 The majority of these sub-sequences would have been discarded by the approach taken in Chapter 7, where local short queries are processed individually, and only the top match is of relevance.

- Number of matches
- Average verification score

This is done by merging sub-sequences of the same file-IDs, and by discarding sub-sequences that violate constraints. The sequence detection algorithm is described in Section 8.2.

The system can be used as a fully automatic sequence detector for the monitoring of long query recordings, or in an interactive mode for use-cases where human post-processing is desired. For the automatic monitoring, the sequence detector is parameterized to automatically clean up regions where several (potentially contradicting) sequences are found, and discard all but the "strongest" sequences by computing some quality measures and reporting the sequences of highest quality.

We add an interactive monitoring mode because we can imagine that the reported results of a system that is used to distribute royalties (among artists, record-labels and rights holders) may be required to be validated by human experts, at least in the early test stages of system application. The interactive analysis is intended to speed up this process, and therefore reduces the overall cost of the system. In case of the interactive mode for example, by visualizing "hidden" sequences (i.e. sequences that would not be revealed when considering just the *best* match of each short local query) we can convey a more complete picture of what is happening in a monitored performance, and how many contradicting results are given for some time segment of the query. The interactive mode supports the manual post processing such that is helps the user to quickly focus on regions with sequence overlaps by selecting an excerpt and visualising the query and reference spectrograms along with verified peaks while listening to the selected excerpt. An example of interactive monitoring is given in Section 8.2.1.

The following section introduces the concept of global result processing using a simple manually crafted example.

8.1.1 *Toy Example*

We start with a simple example to introduce the concept of global sequence detection. This example shall introduce how we will visualize the figures that we present throughput this chapter. We manually modify the time and frequency scale of three reference tracks and concatenate the tracks to obtain a single piece of audio, which will serve as our query recording. The first track is pitched down by 10%, the second is sped up by 20% and the third is slowed down in tempo by 25%. In Figure 8.2 we visualize the sequences that are formed by verified matches, and show the corresponding scale transformation factors (*query*/*ref*) that the system uncovers. The upper figure shows the sequences that could be detected by the fingerprinter. The x-axis represents the time position of the root point of the query quad in

the query audio, and the y-axis shows the root point of the reference quad. Note that in all examples the y-axis ranges from 0 seconds to the maximum reference time position of match-candidates for the claimed file-ID. Each match-candidate reference position will be positioned at its corresponding point in time along the y-axis.

The lower figure shows the uncovered time and pitch scale factors that correspond to the matches within the sequences.

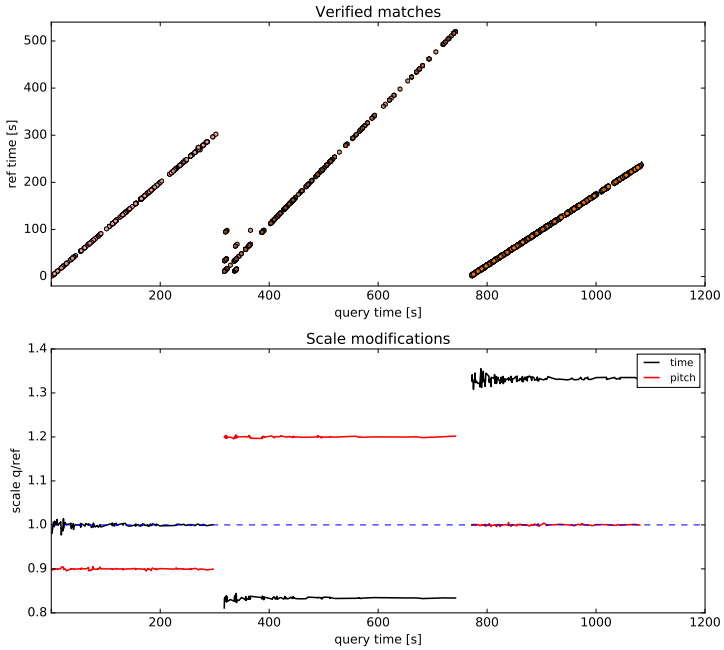


Figure 8.2: An introductory example of sequence detection on global results. The upper sub-figure shows the collected verified matches that adhere to Qfp's set of constraints. The lower figure shows the uncovered scale transformations (first sequence: pitch −10%, second sequence: speed +20%, third sequence: tempo −25%).

In the upper part of the figure we can quickly see that there are three sequences (we concatenated three tracks), and that these sequences have different slopes, which in turn means that the tracks were processed at least with different *TP*- or *T*-modifications. A query song that is played faster compared to the reference will have a slope of more than 45°, and a song that is played slower will have a slope of less than 45°. Note that *P*-effects do not influence the slope of the sequence visualisation.

The lower figure here shows three parts over the time of the query recording. These three parts correspond to the reported sequences of the upper figure, and show the uncovered pitch and time modifications. The first sequence of scale modifications shows that the first track was changed in pitch only, where (in average) no change of the time scale is reported. The pitch scale is reported to be modified by a factor

(query/ref) of approximately 0.9, which means the query has a lower pitch than the reference. The second part, starting at around second 300, indicates modification in pitch as well as time. The pitch value is 120%, and the time value is 83.3% which corresponds to the speed change of +20%. It is important to note that these visualisations show the time scale with its inverse relationship to the duration of the song, i.e. $1/0.833 \approx 1.2$, which in turn indicates a 20% longer time duration of the reference with respect to the query. Recall, that if the pitch is changed at the same rate as the time duration, we call this a modification in *speed* (*TP*-modification). The third part, at around second 800, shows the that the third sequence is modified in tempo, i.e. the pitch was left unaltered and the duration increased to a factor of 1.33, which is a slowdown in tempo of 25%.

In this simple example we just concatenated three tracks, so there is no overlap of sequences that need to be resolved by a sequence detector. More complex and realistic cases from monitoring the mixotic-set are discussed in Section 8.4.

8.2 SEQUENCE DETECTION ALGORITHM FOR THE SEGMENTATION OF QUERY RECORDINGS

In this section we describe how the sequence detector processes the global results that were collected by the fingerprinter for the entire query. This is the collection of all verified sequence candidates that again each consist of a minimum number of 4 verified match-candidates.

The sequence detector has access to the query content, and knows the parameterization of the query process (as for example the STFT settings and the query tolerances). The sequence detector itself is called with a set of threshold parameters that guide the process to discard or keep sequence candidates. The parameters are the minimum number of matches per sequence, the minimum duration of a sequence, the minimum verification score and a minimum sequence score.

The actual parameter values that we use with the sequence detection algorithm will be described in Section 8.3, while in this present section we will focus on the algorithm without suggesting values for the parameterization.

We assign scores to each sequence to be able to compare their quality to other sequences. There are many ways to assign a quality measure to reported sequences of matches, and currently we use empirically determined parameters to assign a score to sequences. Sequence scores take into account the information that is associated with individual verified matches, and ideally would also depend on temporal properties of sequences. For individual matches we know the following match-attributes:

- The query and reference quads, and their spectral peak positions

- The time and pitch transformation factors of each matching pair of quads
- The verification scores

From these attributes we suggest to compute and incorporate temporal properties of the sequences:

- Sequence density
- Sequence segment coverage: is the segment covered by matches, and if not, what is the number and size of gaps within the sequence where no matches are reported.
- The "result-noise", which is the number of contradicting results (sequences of different file-IDs) for the time segment of a given sequence

We currently do not use all of these measures, as in our experiments with our two small datasets of DJ mixes (the mixotic-set and the disco-set) we achieve good results without using the sequence density and segment coverage. If we had more data together with a high quality ground-truth, it would be interesting to formulate the sequence detection as a learning problem.

Note that "sequence" and "segment" refer to very different things. A segment is some time-span in the audio, with a known start-position and duration. A sequence on the other hand is a series of verified matches for the same file-ID. If for example two arbitrary sequences overlap, we will refer to the region of overlap as segment. The region that is covered by a sequence is also called a segment.

To compute sequences we pass the collected sequence candidates in a large dictionary that associates the attributes to their sequence-candidates. In this dictionary, the keys are the sequence candidate file-IDs and the values are a list of match-attributes.

The sequence detector starts to process this input on a per-file-ID basis. That means all results that the fingerprinter reported for a given file-ID are accessed, and the sequence candidates of this file-ID are sorted according to their start-position in time.

Individual sequences in the list of sorted sequences for a given file-ID might overlap (in time), or they might even be fully included in the time-span of a larger sequence. In such cases these sequence candidates (all of which belong to the same file-ID) are merged using a straightforward recursive merging algorithm. Merging of two given overlapping sequences is done by collecting and combining their match attributes, and to create a new sequence from the combined match attributes. From this we get an updated list of matching quad pairs, that we store as lists of query-times, reference-times, verification-score, pitch factors, and time factors. The order is preserved, such that the i-th entry in the list of query-times corresponds to the i-th entry in

the lists of attributes. The merging routine knows the thresholds of the sequence detector, and uses these as constraints to create merged sequences. After this merging step, the number of sequence candidates for all candidate file-IDs is usually drastically reduced.

In our scenario, the next step is to process regions where sequences of different file-IDs are reported. We call these the "contradicting segments". In a production system, it may be desired to not compute this step to be able to report overlapping sequences. In such cases revenue could be split among the artists of the co-occurring tracks. However, we have to compute this step because we we do not have a ground-truth for overlapping sequences or cross-fading of different tracks, and therefore cannot evaluate any predicted overlaps. A contradicting segment is resolved by simply letting the *stronger* sequence win, such that the time-segment of overlap is assigned to the stronger sequence, where "strong" means that it achieves a higher *score* within the segment of overlap.

The score for a sequence is computed from three measures: the number of matches within the segment, the average verification score of the sequence, and the result-noise in the segment.

The result noise is basically intended to be a measure of how many different file-IDs compete for a given segment of the query audio. In most musical genres, when using Qfp, we found the result noise to be very low – in most cases just a single file-ID is reported for a given time-segment, which is due to the verification step of Qfp. For fast and repetitive electronic music genres however, we observe a higher number of competing match-candidates. An intuitive way to interpret the result-noise is to think of it as a measure of how hard it is for Qfp to determine the correct match. The result-noise is computed as the sum of contradicting file-IDs that are reported in a given segment of interest, which in our task is the segment in which the overlap occurs. For the segment of the sequence overlap we bin each query-quad root-point's time position into bins of one second in size, and store the bin-indices. We now do the same for all root-points of the reported global results. After that we inspect the bins at the previously stored indices, and count the unique file-IDs of the root-points that were assigned to these bins. The sum of these counts is now what we call the result-noise.

To assign a segment with overlapping sequences to the strongest sequence we compute their score as follows:

$$score = score_v \, n / (\log(noise + 1) + 1) \tag{8.1}$$

where $score_v$ is the verification score of the sequence as reported by Qfp, n is the number of matches in the sequence and *noise* is the result-noise that corresponds to the binned root-point time positions in the overlap-segment. The idea behind this way of computing the score is the following. The higher the result-noise the more sequences

compete, thus, it is more likely to accept a false positive sequence. Therefore we compute lower scores for noisy segments and assign the segment to that sequence that stands out in terms of score.

The overlap-segment is now assigned to highest-scoring sequence, and all weaker (sub-)sequences in this segment are clipped, i.e. their matches that occur in the segment are deleted. Deleting these matches will cause the lower scoring sequences to either become shorter in time, or create gaps. In any case, the weaker sequences will then consist of a lower number of matches.

At this point, after cleaning up segments with overlap, the existing sequences will have changed their appearance noticeably. First sequences of the same file-ID were merged, and then parts of weaker sequences were deleted.

The next processing step of the sequence detector is to filter the sequences that we obtained from computing the merging step and overlap assignment step. This is done in three steps: merging neighbouring sequences of the same file-ID if these are not too far apart, then deleting sequences that turn out to be short, and finally a second invocation of the merging step.

At this point the sequence detection is finished, and the result is a segmented query recording such that each segment corresponds to a sequence of an identified file-ID.

Note that we also tried to incorporate temporal properties such as sequence density or segment coverage of sequence candidates into this filtering process, but with our limited data volume and ground-truth we could not yet find a reliable way to make temporal properties a useful feature.

8.2.1 *Interactive Sequence Detection*

In this section we demonstrate the interactive analysis of DJ mixes. To keep the example simple, we show the results from changing the parameter for the minimum number of matches, and visualize the results as well as their uncovered time and pitch scale modifications. We gradually increase the minimum required number of matches within a sequence in order to treat this sequence as valid. The query recording of mixotic-set282 will serve as example, based on which we show the sequences that are returned for a minimum of $4, 10, 30, 50, 70$ and 100 matches in Figure 8.3 and Figure 8.4. The first of these figures, Figure 8.3a shows the global sequence-candidates that are returned from the fingerprinter without any post processing. The following figures show the result processing using increasing values for the minimum number of matches. Note, that in addition to this constraint one could also change the parameters for sequence density, sequence duration in time, minimum average verification score, etc. Note that simply changing the number of matches only will not allow for high

identification performance, however, it effectively cleans up results for human validation and post processing.

8.3 AUTOMATIC SEQUENCE DETECTION

In this section we demonstrate the automatic sequence detection, that will set the context for the next section where we perform a case study on this topic.

We parameterize the sequence detector as follows. We require a minimum of 10 matches per sequence, a minimum average verification score of 0.53 and a minimum total sequence score of 5. Finally, we discard all sequences that are shorter than 50 seconds.

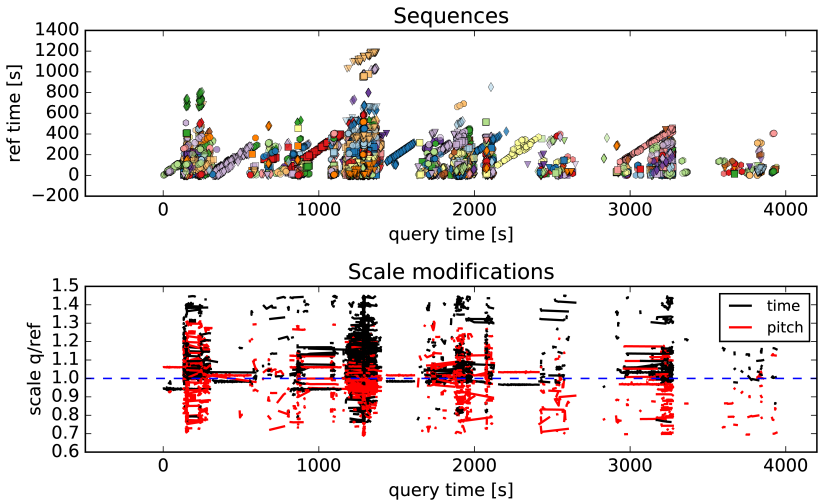
In this section we first show an example using mixotic-set282, and demonstrate the method using the small reference database of 723 tracks. Then, we perform the same experiment on our large database of 430,000 which allows us to observe the increased problem complexity and scalability of Qfp.

Having introduced the concept, we then apply the approach to the whole mixotic dataset. Next, we present evaluation measures and discuss and analyse the mixes that are contained in the dataset.

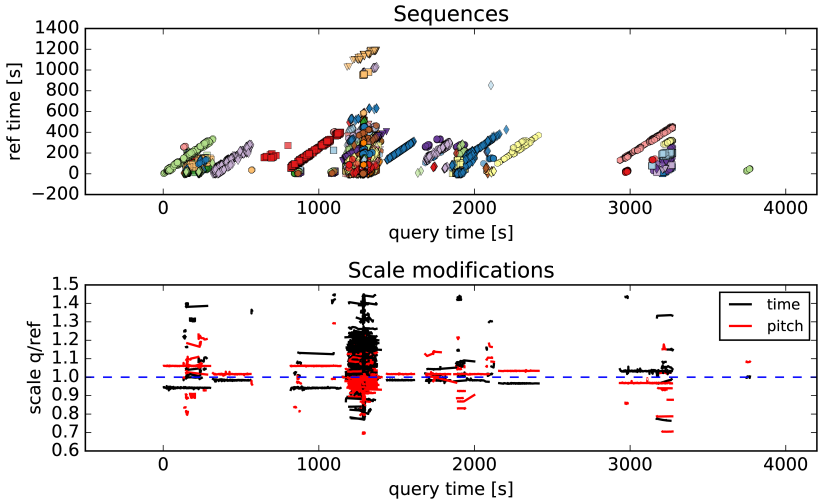
Mixotic-set282 has a duration of roughly 65 minutes. The results using the reference database of 723 tracks as it is described in Chapter 7 is shown in Figure 8.5. The visualisation follows the same concept as described in the previous section, and it is further extended to show the sequences that remain after we apply the sequence detector. This is shown in the middle figure. The returned sequences (shown in the middle figure of Figure 8.5) match well with the ground truth (vertical lines in Figure 8.5). The rightmost sequence is a false positive result, where the system claims a series of matches that were pitched down by roughly 25% while being increased in tempo by roughly 3%. This example shows the automatic sequence cleanup on results that are filtered from a total number of 15,717 matches.

The next experiment follows the same setup as described above, but uses the large reference database that consists of 430,000 tracks (see Chapter 6). Instead of sequences made from 15,717 matches we here obtain a result set that consists of 89,257 matches, but the growth in matches is of course not proportional. We obtain roughly 5.7 times as many matches in sequences, but the database is roughly 595 times as large.

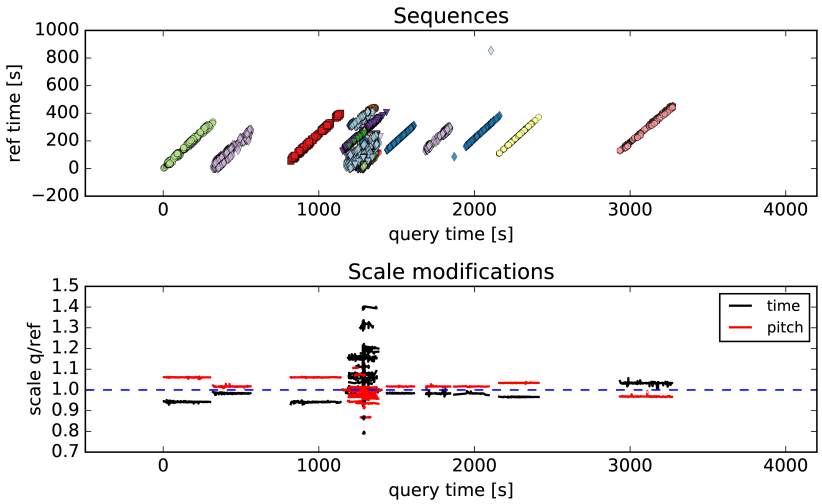
As the individual mixes are of different duration, we do not average the performance measures over the 10 mixes, but treat the mixotic set as one very long mix, and accumulate the confusions. This results in a overall accuracy of 0.88, a precision of roughly 0.96, and a specificity of roughly 0.90. Compared to the local query evaluation on the large database, as given in Section 7.4 of Chapter 7, the automatic sequence detection on global results (using this specific parameterization) in-



(a) All possible sequences on mixotic set282 (minimum number of matches: 4).

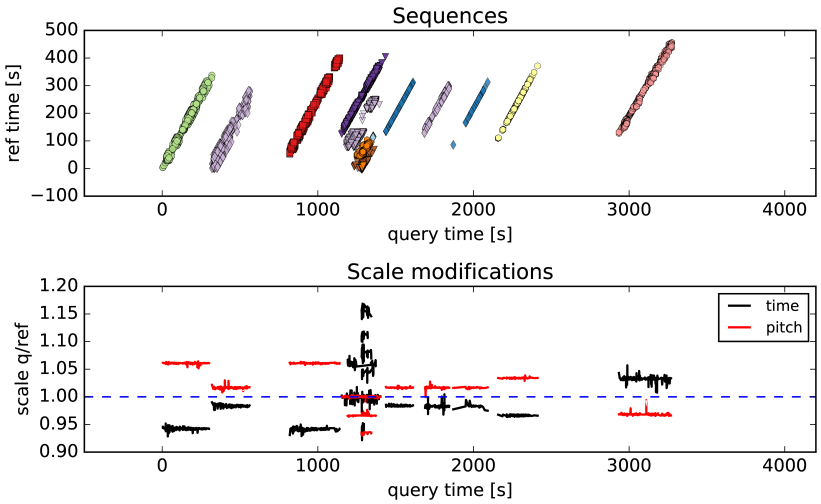


(b) Minium number of matches: 10

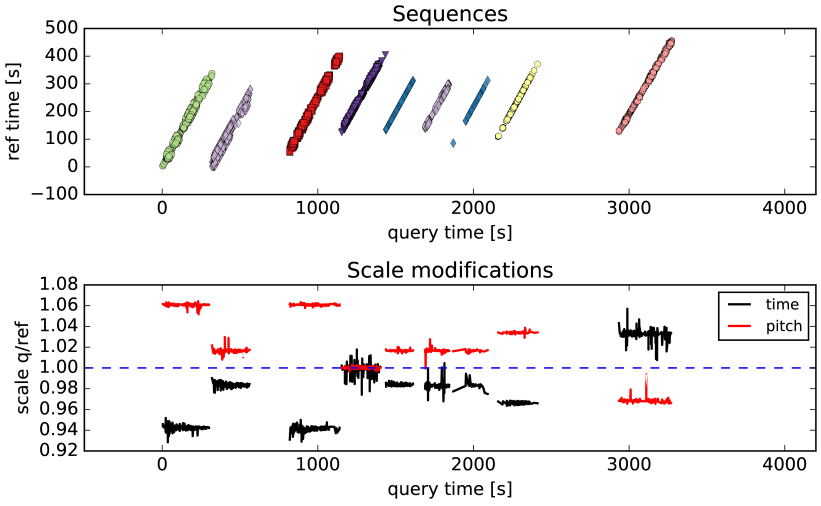


(c) Minium number of matches: 30

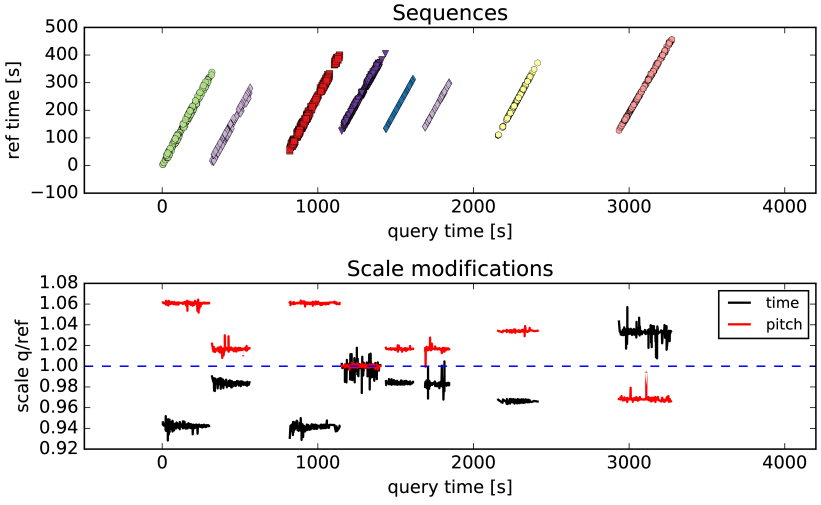
Figure 8.3: Semi-automatic analysis: varying the number of matches.



(a) Minium number of matches: 50



(b) Minium number of matches: 70



(c) Minium number of matches: 100

Figure 8.4: Semi-automatic analysis: varying the number of matches.

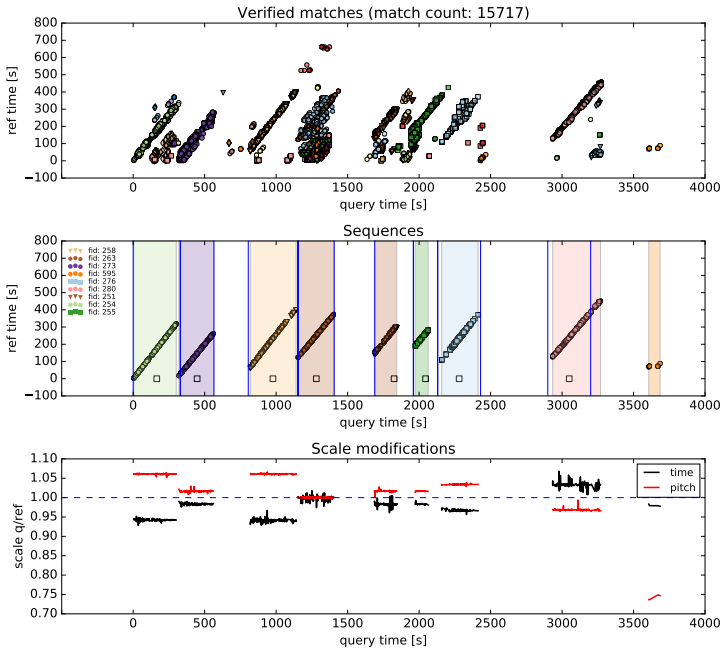


Figure 8.5: Automatic sequence detection on set282 of the mixotic data set, using the small reference database that contains the 723 published reference tracks. The upper sub-figure shows the collected verified matches. The middle figure shows the result of the sequence detector, where the colored regions show the segments are claimed for an identified track, and the vertical bars show the song-border annotations of the ground-truth. The lower figure shows the uncovered pitch- and time-scale transformations.

creases the overall performance: on the mixotic-set the accuracy is increased by 5 percentage points, the precision by 9 percentage points and the specificity by 34 percentage points [2]. On the disco set, the accuracy increases by 8 percentage points, and precision and specificity benefit with an increase of 14 and 20 percentage points, respectively.

In Table 8.1 the evaluation of the automatic sequence detector on the 10 DJ mixes that are contained in the mixotic-set is presented. The parameters as described above are found empirically to give acceptable results on the mixotic- as well as the disco-set, although, we need more data for an absolutely independent parameter study and evaluation.

Note, that it is of course possible to parameterize the automatic sequence detector to focus on specific performance characteristics, for example it could be parameterized to try to increase the accuracy of the system at the cost of precision and specificity.

2 To be precise, in Section 7.4 we noted the impact of a song that is not present in the reference recordings of the published dataset, however it is present in the Jamendo corpus. This resulted in a sequence that was incorrectly labelled as false positive, and impacted the specificity. Here, we fixed this issue and treat this sequence as true positive. The influenced values are given bold in Figure 8.14for mixotic set282.

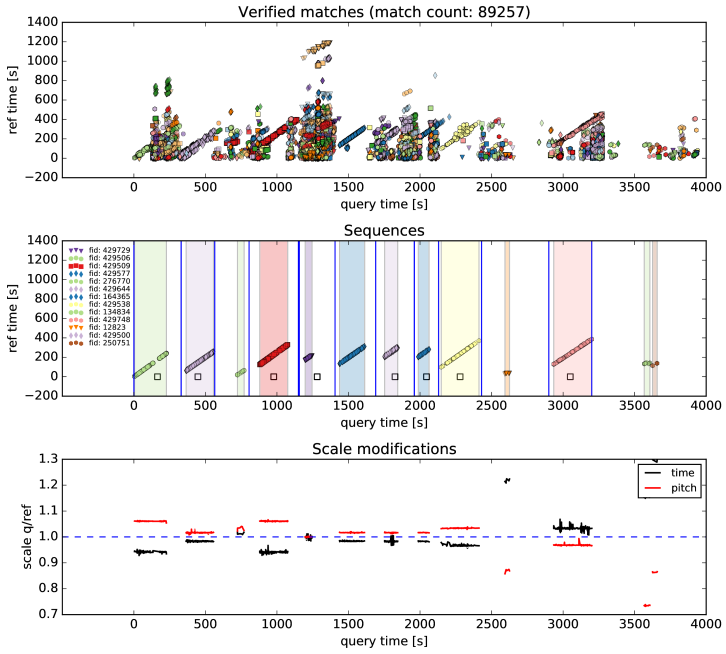


Figure 8.6: Automatic sequence detection on set282 of the mixotic data set, using our reference database that contains 430,000 tracks, including the published mixotic reference tracks. The upper sub-figure shows the collected verified matches. The middle figure shows the result of the sequence detector, where the colored regions show the segments that correspond to an identified track, and the vertical bars show the song-border annotations of the ground-truth. The lower figure shows the uncovered pitch- and time-scale transformations.

8.4 CASE STUDY: ANALYSIS OF THE MIXOTIC DATASET

In Figures 8.5, 8.6 the bottom sub-figures demonstrate the variety of scale modifications that are applied by the DJs. In this section we look at the entire mixotic-dataset in detail, and discuss what can be observed based on the results of our proposed method.

We will organise the analysis using a separate double-page for each mix, where we discuss the query recording properties and the behaviour of the method on the left-hand page, and show the visualisation of the sequence detector and the result table on the right-hand page.

8.4.1 *Descriptions of Tables and Figures*

The sequence detection figures that we present in the following subsections consist of three sub-figures. In these, the top sub-figure shows the collection of verified matches that are reported by Qfp. The middle

Set	tp	fp	fn	tn	fp_s	Acc.	Prec.	Spec.
044	4126	144	395	0	0	0.88	0.96	–
123	3078	149	113	0	0	0.92	0.95	–
222	3053	26	500	1742	355	0.85	0.99	0.83
230	2494	36	47	778	5	0.96	0.98	0.99
275	3080	177	262	1515	15	0.87	0.94	0.99
278	3245	161	182	265	19	0.90	0.95	0.93
281	2593	151	558	285	12	0.78	0.94	0.95
282	2095	100	180	1345	232	0.88	0.95	0.85
285	3932	31	596	0	0	0.86	0.99	–
286	2815	180	164	0	0	0.89	0.93	–
All:	30511	1155	2997	5930	638	**0.88**	**0.96**	**0.90**

Table 8.1: Evaluation of the automatic sequence detection on the complete **mixotic**-set, using our large reference database. tp,fp,fn show the confusions in seconds, concerning those parts of the mixes for which the reference song is present in the database. tn,fp_s show true-negatives and false-positives for the parts for which we do not have the reference. Acc. is the accuracy, Prec. the precision and Spec. the specificity.

Set	tp	fp	fn	tn	fp_s	Acc.	Prec.	Spec.
set0	4608	138	922	1941	238	0.81	0.97	0.89
set1	3129	213	111	286	27	0.90	0.93	0.91
set2	2377	248	583	294	0	0.74	0.90	1.0
set3	895	0	159	1537	469	0.84	1.0	0.76
set20	2433	365	339	456	1	0.77	0.86	0.99
set35	113	0	218	1004	0	0.34	1.0	1.0
set36	178	0	694	768	0	0.20	1.0	1.0
set37	429	0	291	730	0	0.59	1.0	1.0
All:	14162	964	3317	7016	735	**0.77**	**0.94**	**0.91**

Table 8.2: Evaluation of the automatic sequence detection on the complete (non-free and unpublished) **disco**-set, using our large reference database. See Table 8.1 for a description of the columns. The confusions are given in seconds.

sub-figure shows the detected sequences. There, the ground truth annotations of the song-boundaries as shown as vertical bars, and the markers represent the reference file-ID (as there are less markers than file-IDs we simply cyclically index into a set of markers). The horizontal bar at the bottom of each middle sub-figure represents the so-called confusions in seconds: true positives are shown in green, false positives are red, true negatives are shown in blue and false negatives are shown in yellow. The legend on the left-hand side of the

middle sub-figure gives Qfp's file-IDs of identified tracks. The bottom sub-figures show the uncovered time and pitch transformations that correspond to their sequences.

The result tables shown below use eight columns as follows:

1. "nm" is the number of matches within a final sequence. Large numbers of matches typically indicate that the query content is not mixed with additional sources, but still might be subject to severe scale modifications. Low numbers of matches may indicate false positives, or the presence of highly distorting digital audio effects, or for example a track being overlaid with another track.

2. "tquery" gives both, the start and end time in seconds.

3. "len" is the sequence length that we include for convenience, it could also be determined from tquery.

4. "pf.est" represents the estimated pitch scale factor (query vs. reference, averaged over all matches contained in the final sequence)

5. "tf.est" shows the estimated time scale factor (query vs. reference, averaged over all matches contained in the sequence). A value lower than 1.0 means that the time is compressed with respect to the reference track, i.e. the query is played faster. Values larger than 1.0 show a stretched query, meaning that the query is player slower.

6. "vscore" is the verification score averaged over all contained matches within the sequence. The minimum verification score for the fingerprinter, before passing results on to the sequence detector, is 0.53. A low average verification score tells us that there are a number of missing peaks in the query spectrogram. This may be due to severe signal distortions or cross-mixing. For severe non-linear scale modifications, as for example when the query audio is rapidly accelerated or decelerated the vscore will be lower in comparison to linearly scaled modifications. This is because non-linear transformations will cause the fingerprinter to compute non-representative scale modifications, which can prevent the algorithm from aligning some present spectral peaks.

7. "score" represents the overall strength of the sequence (see Section 8.2). The score is not normalized, and does not have the same range over different recordings. At this time, the score reflects our attempt to find a summarizing measure for the quality of a matched sequence. We want to refine our scoring method as soon as we have more data.

8. "title" gives the first ten letters of artist/track information. We shorten this due to space constraints. The full titles are available in the ground-truth files of our published dataset[3]. In a few cases, the evaluation table shows a title that is just a long number. In our reference collection such numbers represent Jamendo file-IDs. Since the mixotic-set does not contain tracks from the Jamendo dataset [4], these cases are false-positive sequences.

The lower part of each table shows the evaluation results for the DJ-mix. The confusions are counted in seconds, and the fields "Acc.", "Prec." and "Spec." denote the accuracy, precision and specificity. The fields "Spec.TN" and "Spec.FP" denote the true negatives and false positives in seconds for cases in which the correct track is non-identifiable. In cases of identifiable content the specificity is denoted as "−".

3 The mixotic dataset is available at http://www.cp.jku.at/datasets/fingerprinting/

4 There is one exception to this. One track of the mixotic-set 282 (see Figure 8.14a) is contained in our Jamendo dataset.

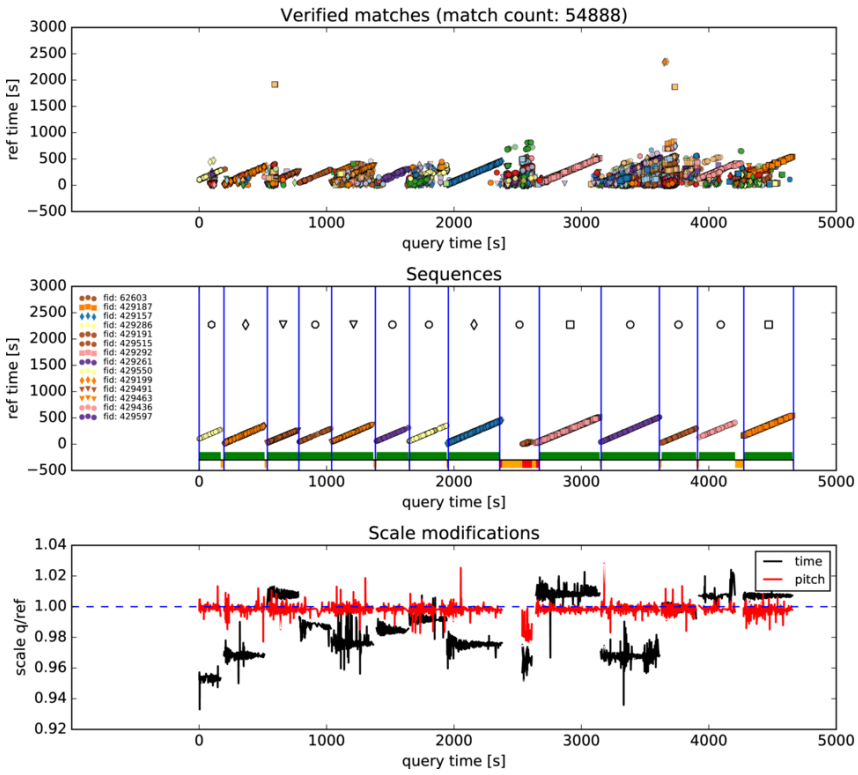
8.4.2 *Set 044*

The results for this mix are presented in Figure 8.7.

- Number of identifiable tracks: 14
- Number of non-identifiable tracks: 0
- Accuracy: 0.88
- Precision: 0.96
- Specificity: –

According to Table 8.7b, the method achieves an accuracy of 0.88 and a precision of 0.96. All played tracks are present in the reference database, so specificity is not of relevance – there are no true negatives. The accuracy is impacted by a missed sequence, parts of which even give rise to a falsely claimed sequence with 70 matches and a duration of 75 seconds. To see whether the correct matches are overwritten by false sequences that were discarded in the end, we can consult the evaluation of local queries as given in Chapter 7, Figure 7.1a. We can see that there we also fail to identify large portions of that track, preventing the sequence detector to reveal this part as sequence. The method Audfprint manages to find large portions of this track at its expected position in the query recording, but the claimed reference time positions are rather chaotic. This could indicate that the content is either highly repetitive or looped. It could also mean that this track is mixed with another track. In such cases we expect Audfprint's chances to detect the part to be higher than for our method, as for Audfprint only two peaks have to be found to match a single fingerprint while our method needs to find four peaks to do so. In the case of a track that is "overlaid" with another track, chances are high that query quads contain peaks of both signals, and therefore will rarely match the reference.

Overall, the pitch of played content is hardly modified, while the time scale is modified in all tracks. This means the performer used *T*-modifications: Four tracks are played a bit slower, the remaining ones are played faster. The most severe change seems to be the tempo of the first track, which is approximately 5% faster.



(a) Sequence detection

set044							
nm	tquery [s]	len	*pf.est*	*tf.est*	*vscore*	*score*	Title
244	1; 169	168	1.000	0.953	0.791	192.940	pntg012-04
352	195; 513	318	0.999	0.968	0.830	292.238	pntg010-02
634	532; 779	247	0.998	1.008	0.911	577.870	pntg012-01
187	786; 1035	249	0.998	0.987	0.790	147.808	pntg003-04
513	1038; 1367	329	0.999	0.976	0.847	434.462	pntg007-01
429	1393; 1643	250	0.998	0.985	0.891	382.092	pntg009-04
580	1649; 1943	294	0.998	0.992	0.899	521.479	pntg012-08
645	1946; 2377	431	0.998	0.976	0.877	565.697	pntg012-03
70	2536; 2611	75	0.984	0.966	0.688	19.111	1171351
1003	2644; 3143	499	0.998	1.009	0.869	821.058	pntg012-02
803	3152; 3612	460	0.999	0.968	0.823	303.110	pntg012-05
1137	3630; 3901	271	0.998	0.999	0.951	1081.152	pntg005-03
126	3924; 4206	282	0.999	1.007	0.768	96.796	pntg006-03
792	4273; 4656	383	0.998	1.007	0.899	712.148	pntg011-02
Eval :							
TP	FP	FN	Spec.TN	Spec.FP	Acc.	Prec.	Spec.
4126	144	395	0	0	0.88	0.96	-

(b) Results

Figure 8.7: Automatic sequence detection on mixotic-set 044

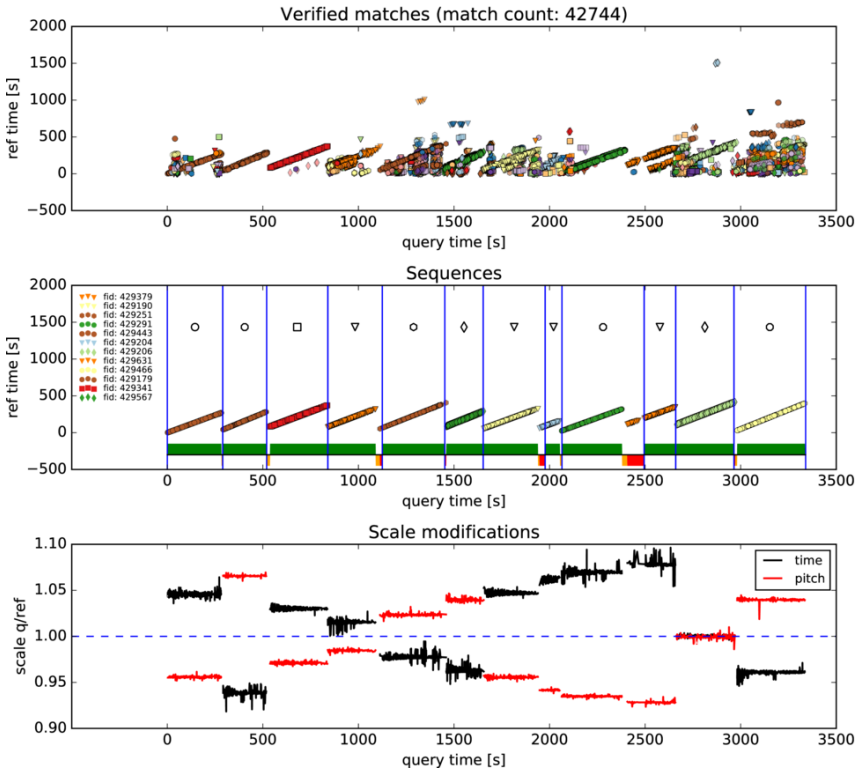
8.4.3 *Set 123*

The results for this mix are presented in Figure 8.8.

- Number of identifiable tracks: 12
- Number of non-identifiable tracks: 0
- Accuracy: 0.92
- Precision: 0.95
- Specificity: –

This mix contains twelve tracks, all of which are referenced, so there are no true negatives. Each detected sequence shows a perfect diagonal, with no disagreement in the reference time positions. The accuracy of 0.92 and precision of 0.95 seems to be negatively impacted by what seems to be an inaccurate ground-truth annotation, which we should revisit. We think we might have put the song border of track number 9 too late. If we look at the results presented in Figure 7.1b (for this region before second 2500), neither of the three methods detect what is claimed by the ground-truth.

In this mix, the performer incorporates *TP*-modifications to all but one tracks. In seven cases, the content is played slower together with a lower pitch. In four cases the content is sped up, and the one remaining track is not modified in scale at all. Some speed changes are quite severe, for example the third to last track was played roughly 8% slower.



(a) Sequence detection

set123							
nm	tquery [s]	len	*pf.est*	*tf.est*	*vscore*	*score*	Title
1058	3; 283	280	0.956	1.046	0.942	997.032	04_Peter_G
445	292; 518	226	1.066	0.939	0.811	361.013	04_Vizar_-
1019	538; 838	300	0.971	1.030	0.947	965.400	04_Sascha_
714	840; 1090	250	0.985	1.015	0.915	653.093	04_Katsuyu
1050	1113; 1460	347	1.023	0.977	0.887	931.809	01_Sascha_
874	1460; 1655	195	1.039	0.963	0.901	654.145	03_Vizar_-
690	1656; 1939	283	0.956	1.047	0.918	459.037	03_Hiroshi
194	1948; 2054	106	0.942	1.061	0.885	171.735	04_Pepe_Ar
613	2063; 2379	316	0.935	1.069	0.814	499.168	02_Sascha_
301	2407; 2659	252	0.928	1.079	0.841	253.025	04_Roger_M
1065	2664; 2971	307	1.000	1.000	0.924	984.461	08_Banding
1150	2982; 3336	354	1.040	0.961	0.909	1044.783	04_Hermeti
Eval :							
TP	FP	FN	Spec.TN	Spec.FP	Acc.	Prec.	Spec.
3078	149	113	0	0	0.92	0.95	-

(b) Results

Figure 8.8: Automatic sequence detection on mixotic-set 123.

8.4.4 *Set 222*

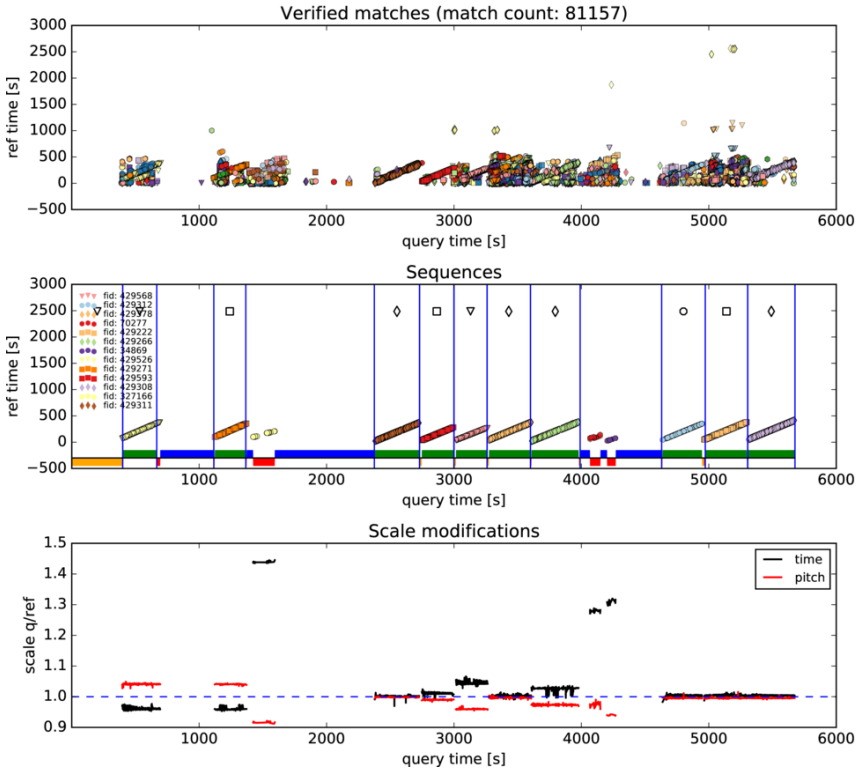
The results for this mix are presented in Figure 8.9.

- Number of identifiable tracks: 11
- Number of non-identifiable tracks: 7
- Accuracy: 0.85
- Precision: 0.99
- Specificity: 0.83

This mix contains *TP*-modifications not stronger than roughly 4 percent. Remaining tracks are changed in scale only moderately, or not at all.

Our method completely misses the entire first sequence, and the automatic sequence detector gives three short false positives, all coming from regions where many file-ID candidates were returned from the fingerprinter. From experience we know, that such noisy regions tend to contain highly repetitive content.

The missed sequence causes the low accuracy of only 85%. The wrongly claimed sequences occur in locations for which there is no reference. This noticeably impacts the specificity, which is only 83%. The overall precision of 99% is almost perfect with only 26 false positive seconds (in a mix that is longer than 90 minutes) for regions where the reference is present.



(a) Sequence detection

set222							
nm	tquery [s]	len	*pf.est*	*tf.est*	*vscore*	*score*	Title
702	396; 694	298	1.041	0.961	0.870	442.404	[thn109]-3
374	1123; 1372	249	1.041	0.962	0.860	321.822	mr-cloudy-
48	1424; 1593	169	0.916	1.440	0.593	14.998	346520
1539	2379; 2727	348	0.999	1.002	0.978	1504.803	brq60_nied
770	2748; 2998	250	0.990	1.011	0.939	722.967	Bactee_Tit
635	3016; 3265	249	0.960	1.044	0.856	543.406	Zimmer046.
846	3275; 3607	332	0.997	1.003	0.861	728.649	Max_Cavale
816	3608; 3978	370	0.974	1.027	0.909	741.617	2_DML_-_Fi
22	4068; 4149	81	0.973	1.278	0.638	5.977	394114
15	4202; 4269	67	0.940	1.308	0.639	9.590	964433
1989	4637; 4946	309	0.996	1.004	0.938	1287.533	DML_-_Arbe
1234	4964; 5298	334	0.996	1.004	0.920	1134.841	brq50_max_
1227	5309; 5674	365	0.996	1.004	0.948	1163.004	Patrick Di
Eval :							
TP	FP	FN	Spec.TN	Spec.FP	Acc.	Prec.	Spec.
3053	26	500	1742	355	0.85	0.99	0.83

(b) Results

Figure 8.9: Automatic sequence detection on mixotic-set 222.

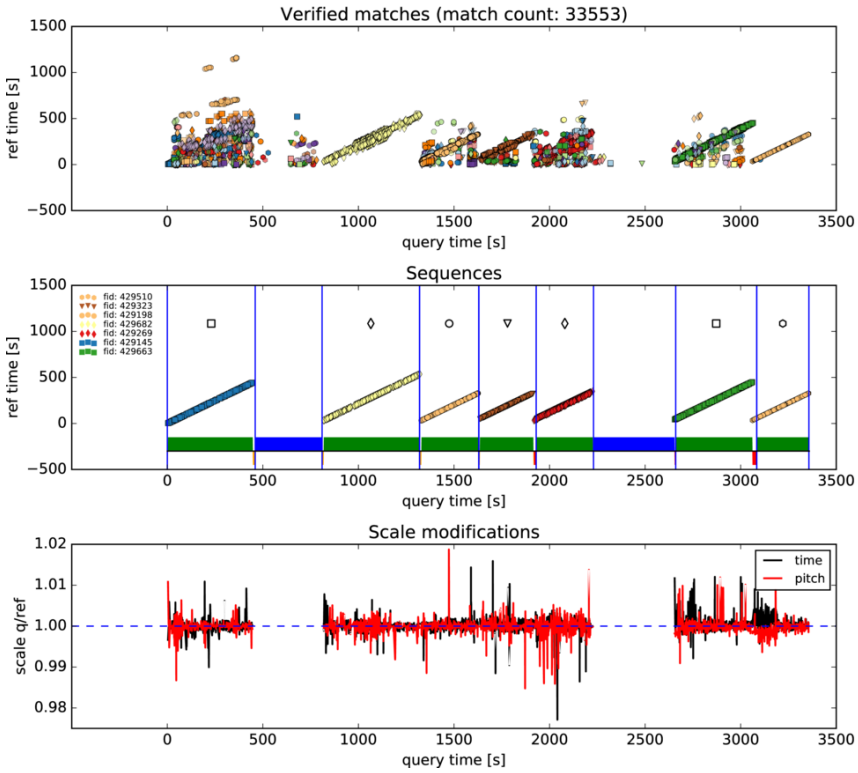
8.4.5 *Set 230*

The results for this mix are presented in Figure 8.10.

- Number of identifiable tracks: 7
- Number of non-identifiable tracks: 2
- Accuracy: 0.96
- Precision: 0.98
- Specificity: 0.99

Averaged over the duration of the contained tracks, there occur no scale modifications at all.

Two tracks are not referenced in the database, and in both cases the sequence detector manages to correctly abstain from reporting matches. The results achieved by our method are almost perfect in this case.



(a) Sequence detection

set230							
nm	tquery [s]	len	*pf.est*	*tf.est*	*vscore*	*score*	Title
1165	4; 447	443	1.000	1.000	0.923	1074.894	DN015_04-a
994	819; 1322	503	1.000	1.000	0.892	886.362	thn010-02-
1442	1330; 1626	296	1.000	1.000	0.955	1376.897	DN020-08_f
1147	1636; 1915	279	1.000	1.000	0.901	1033.343	DN005_04_l
1385	1919; 2226	307	1.000	1.000	0.923	833.818	DN006_01_k
1491	2655; 3062	407	1.000	1.000	0.918	1369.285	DN020-12_t
1579	3062; 3356	294	0.999	1.001	0.981	1549.672	DN015_10-n
Eval :							
TP	FP	FN	Spec.TN	Spec.FP	Acc.	Prec.	Spec.
2494	36	47	778	5	0.96	0.98	0.99

(b) Results

Figure 8.10: Automatic sequence detection on mixotic-set 230.

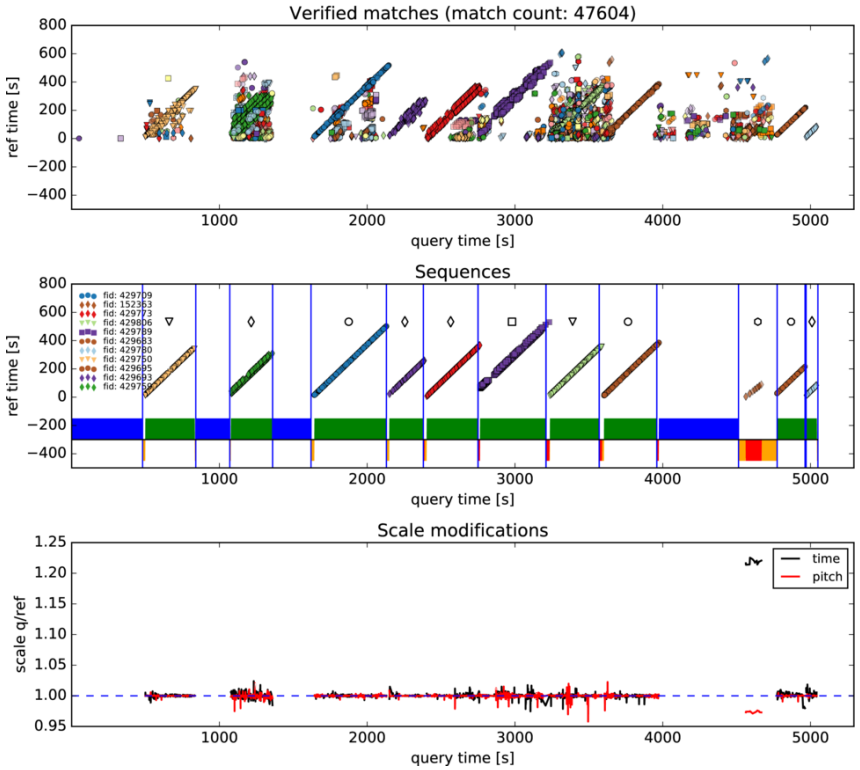
8.4.6 *Set 275*

The results for this mix are presented in Figure 8.11.

- Number of identifiable tracks: 11
- Number of non-identifiable tracks: 6
- Accuracy: 0.87
- Precision: 0.94
- Specificity: 0.99

Four tracks that are played in this mix are not present in the reference database. These cases are correctly handled by the method, which here performs with a specificity of 99%. The total number of 177 seconds of false positives and 262 seconds of false negatives are due to a missed sequence. Here, the missed sequence gives room for the false-positive sequence, which in turn lowers the precision to 94%. This wrongly claimed sequence has a low average verification score of 0.59, and a very low number of matches (12 matches) compared to the other results in this DJ mix.

As in mixotic-set 230, the correct sequences show no pitch or time scale modifications.



(a) Sequence detection

set275							
nm	tquery [s]	len	*pf.est*	*tf.est*	*vscore*	*score*	Title
1453	499; 833	334	1.000	1.000	0.978	1421.308	atabey - j
1806	1078; 1359	281	1.000	1.000	0.944	1054.458	02.Stereop
2187	1643; 2131	488	1.000	1.000	0.961	2101.032	Krisz Deak
907	2150; 2388	238	1.000	1.000	0.966	876.436	01_F.l.o._
2013	2403; 2763	360	1.000	1.000	0.979	1971.553	02.Echoton
1792	2764; 3233	469	1.000	1.001	0.914	1413.682	Roberto Fi
1839	3237; 3589	352	1.000	1.000	0.966	1777.378	03_diarmai
3546	3603; 3973	370	1.000	1.000	0.939	2827.459	03_kwartz_
12	4562; 4669	107	0.974	1.219	0.590	7.083	139248
554	4774; 4963	189	1.000	1.001	0.962	511.012	01_the_thi
437	4970; 5043	73	1.000	1.001	0.982	429.305	01_kwartz_
Eval :							
TP	FP	FN	Spec.TN	Spec.FP	Acc.	Prec.	Spec.
3080	177	262	1515	15	0.87	0.94	0.99

(b) Results

Figure 8.11: Automatic sequence detection on mixotic-set 275.

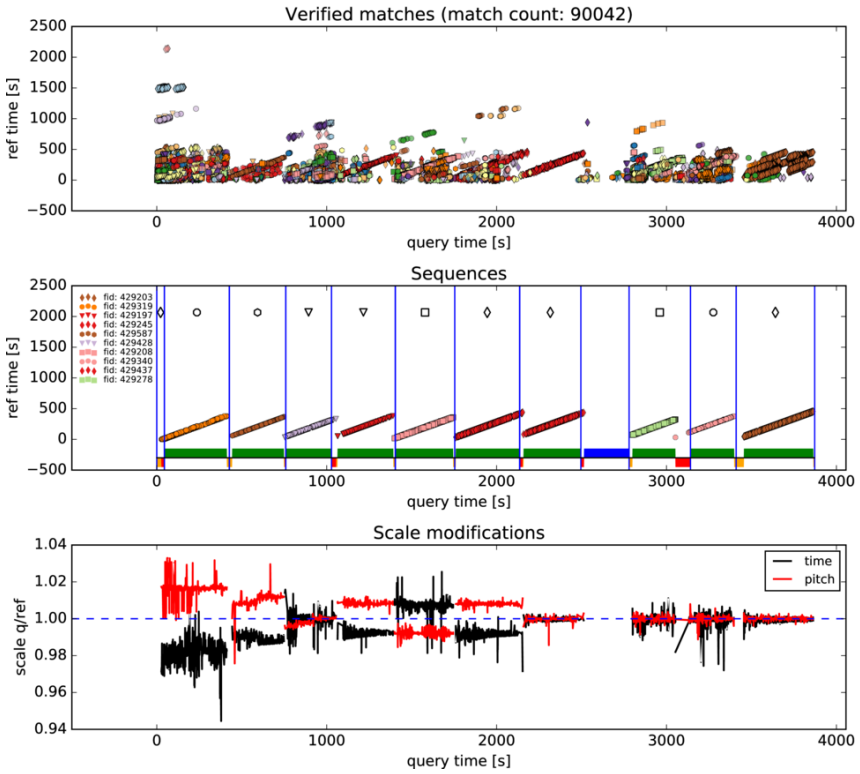
8.4.7 *Set 278*

The results for this mix are presented in Figure 8.12.

- Number of identifiable tracks: 11
- Number of non-identifiable tracks: 1
- Accuracy: 0.90
- Precision: 0.95
- Specificity: 0.93

Apart from one correctly interpreted missing reference track, all sequences are identified. There are minor errors near the annotated borders, which may be due to long fading regions.

Scale modifications are quite moderate, and lower than two percent. Modified content is mostly sped up, and slowed down in only one of the contained tracks.



(a) Sequence detection

set278							
nm	tquery [s]	len	*pf.est*	*tf.est*	*vscore*	*score*	Title
1300	27; 412	385	1.017	0.981	0.904	459.387	hryk - A.D
927	444; 748	304	1.010	0.989	0.909	690.240	ivity - ce
814	754; 1055	301	0.999	0.999	0.905	736.381	Dynastic_-
736	1064; 1392	328	1.008	0.992	0.921	677.730	Dynastic_-
880	1400; 1744	344	0.992	1.007	0.924	509.528	hryk - par
1343	1760; 2153	393	1.008	0.991	0.952	932.248	Flip & Tou
1536	2159; 2514	355	1.000	1.000	0.962	1477.365	l_cio - al
1035	2800; 3052	252	1.000	0.999	0.897	549.579	whimee - a
1268	3053; 3398	345	1.000	0.999	0.916	1161.334	o6 6. Redu
1847	3456; 3864	408	1.000	1.000	0.951	1757.101	The Nautil
Eval :							
TP	FP	FN	Spec.TN	Spec.FP	Acc.	Prec.	Spec.
3245	161	182	265	19	0.90	0.95	0.93

(b) Results

Figure 8.12: Automatic sequence detection on mixotic-set 278.

8.4.8 *Set 281*

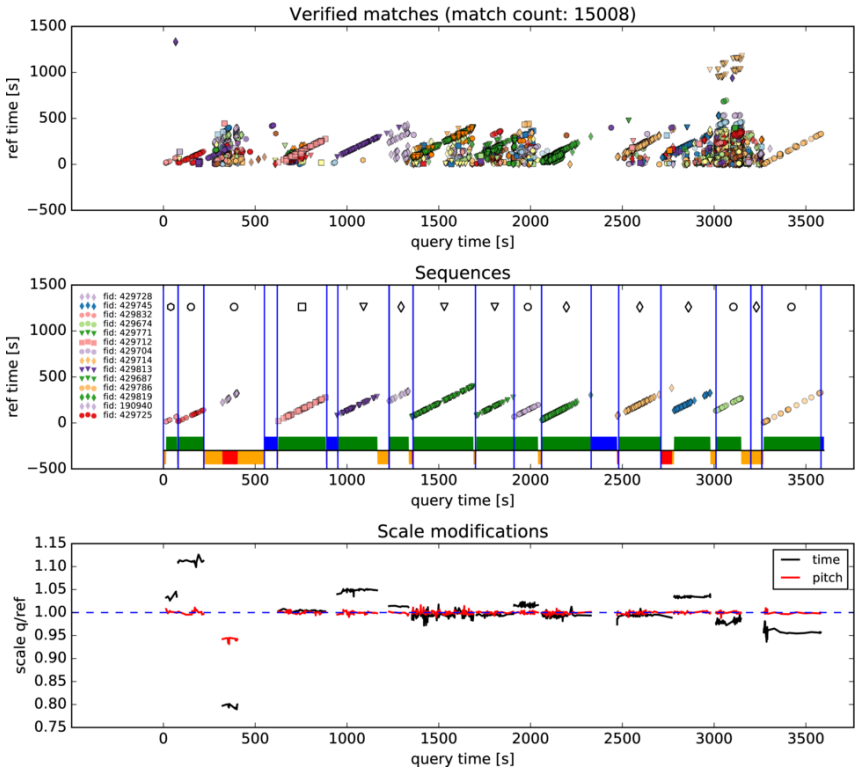
The results for this mix are presented in Figure 8.13.

- Number of identifiable tracks: 15
- Number of non-identifiable tracks: 3
- Accuracy: 0.78
- Precision: 0.94
- Specificity: 0.95

This mix contains 18 tracks, three of which are not referenced in the database. The obtained accuracy of 78% is relatively low, and is explained by the method failing to detect two sequences, one of which is of long duration.

In this mix we only observe *T*-modifications. In six cases the tempo is reduced, and in two cases it is sped up. This mix contains one of the more severe scale changes, where the second track is played 10% slower.

However, there is something odd in this mix. The number of matches for the sequences is extremely low, and all but two sequences have less than 100 matches, which is not the case in other mixes in the dataset. Typically, there are hundreds of matches for a correctly identified sequence. Consulting Figure 7.2d, we can see the good performance of Audfprint for this mix (but on the small database of 723 tracks). We listened to the some of the tracks and their references, and found that these indeed are heavily modified with additional samples and effects.



(a) Sequence detection

set281							
nm	tquery [s]	len	*pf.est*	*tf.est*	*vscore*	*score*	Title
12	15; 74	59	1.002	1.035	0.613	7.354	Zimmer110.
17	80; 220	140	1.001	1.111	0.673	11.448	4.Rabitza-
29	322; 404	82	0.940	0.795	0.613	17.786	267695
54	626; 884	258	1.000	1.003	0.726	39.227	01.Brandon
62	947; 1165	218	0.999	1.048	0.758	47.021	[DigitalDi
13	1229; 1336	107	0.998	1.013	0.710	9.225	CODA013_Dr
230	1354; 1688	334	1.000	0.995	0.823	139.727	[DigitalDi
59	1705; 1906	201	1.000	0.995	0.730	43.042	[DigitalDi
96	1910; 2040	130	1.000	1.016	0.758	45.584	[DigitalDi
199	2060; 2329	269	0.999	0.994	0.830	165.200	[DigitalDi
111	2472; 2770	298	1.000	0.994	0.711	78.929	[DigitalDi
50	2782; 2979	197	1.000	1.034	0.840	28.103	brq101_nic
95	3010; 3147	137	1.000	0.980	0.849	80.701	02_kreisla
54	3270; 3582	312	1.001	0.956	0.731	39.494	7.ALW042_P
Eval :							
TP	FP	FN	Spec.TN	Spec.FP	Acc.	Prec.	Spec.
2593	151	558	285	12	0.78	0.94	0.95

(b) Results

Figure 8.13: Automatic sequence detection on mixotic-set 281.

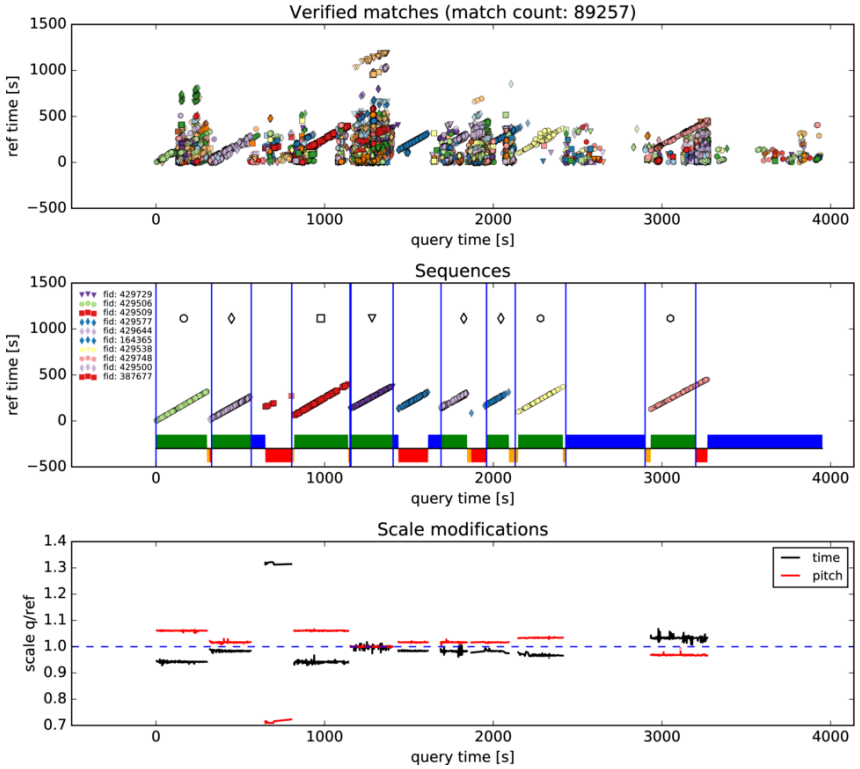
8.4.9 *Set 282*

The results for this mix are presented in Figure 8.14.

- Number of identifiable tracks: 8
- Number of non-identifiable tracks: 6
- Accuracy: 0.88
- Precision: 0.95
- Specificity: 0.85

For this mix we found the correct reference for track number 7 in our large reference database, but this track is not present in our published dataset. In Chapter 7 we treat this as false positive, as the ground truth claims that this track is not present. Here, we treat this track correctly, and manually fix the wrongly assigned confusions. These cases are typed bold in Table 8.14b. Apart from the issues with that specific track, we falsely claim a detected sequence in a region where the reference is missing, and only reach a specificity of 85%. The last sequence is noticeably longer than the corresponding region of the ground truth, which also negatively impacts the specificity.

The scale modifications are done with at most 6% of speed-up. One track is slowed down by roughly 3%.



(a) Sequence detection

set282							
nm	tquery [s]	len	*pf.est*	*tf.est*	*vscore*	*score*	Title
627	4; 302	298	1.061	0.942	0.893	560.032	(05) Smoot
446	320; 562	242	1.016	0.983	0.869	387.629	08 Baumfre
12	649; 804	155	0.712	1.318	0.657	7.878	730347
540	820; 1139	319	1.061	0.941	0.807	334.292	[04] Smoot
795	1153; 1404	251	1.000	0.999	0.934	742.274	04 - Spira
195	1438; 1613	**175**	1.017	0.984	0.823	160.504	**! Scamos !**
356	1688; 1844	156	1.017	0.983	0.813	196.886	(03) Smoot
74	1870; 2092	222	1.017	0.983	0.835	61.780	(02) Smoot
588	2148; 2412	264	1.034	0.967	0.907	475.255	10 Martin
614	2934; 3269	335	0.968	1.033	0.832	341.220	04 - Subli
Eval :							
TP	FP	FN	Spec.TN	Spec.FP	Acc.	Prec.	Spec.
2095	100	180	1345	**232**	**0.88**	0.95	**0.85**

(b) Results

Figure 8.14: Automatic sequence detection on mixotic-set 282.

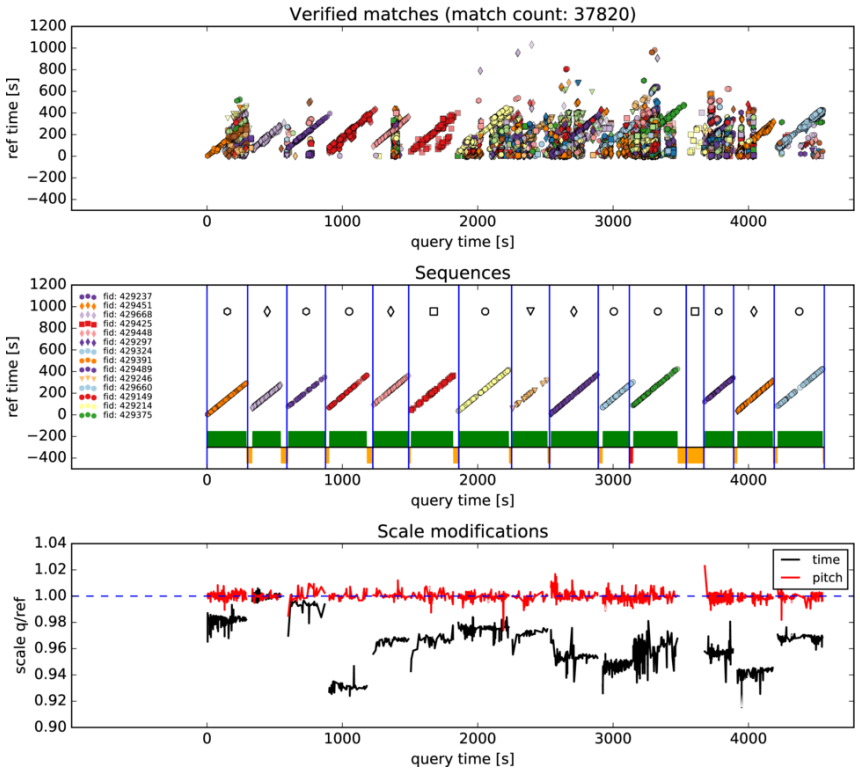
8.4.10 *Set 285*

The results for this mix are presented in Figure 8.15.

- Number of identifiable tracks: 15
- Number of non-identifiable tracks: 0
- Accuracy: 0.86
- Precision: 0.99
- Specificity: –

All of the 15 tracks are present in the reference database. For one of the tracks our method fails to report the correct reference, but does not introduce false positives for this region. The missed sequence as well as mistakes near the annotated song borders negatively impact the accuracy to 86%.

All but one track are modified with higher tempo (*T*-modification). The remaining one is not scale-modified at all. The most severe scale modification is roughly +7.4% faster tempo. An interesting observation is that the content starting at around second 3000 is slowly but steadily decelerated (non-linear scale modification), but still our method manages to correctly identify the track.



(a) Sequence detection

set285							
nm	tquery [s]	len	*pf.est*	*tf.est*	*vscore*	*score*	Title
247	4; 291	287	1.000	0.982	0.845	208.610	7-04 - psy
226	335; 545	210	1.000	1.000	0.745	168.279	06 - Hudoz
62	600; 871	271	1.001	0.993	0.798	49.449	04 - Alex
71	903; 1180	277	1.000	0.931	0.729	41.843	15 - Bet O
77	1228; 1490	262	1.001	0.966	0.790	60.808	01 - Fcode
75	1506; 1818	312	1.000	0.966	0.800	57.016	04 - Mudar
257	1857; 2230	373	0.999	0.975	0.812	208.611	079 - DJ Z
49	2256; 2515	259	0.999	0.971	0.780	38.204	28 - Whoon
117	2540; 2885	345	1.001	0.954	0.750	87.738	02 - Mr Ka
175	2923; 3146	223	0.999	0.946	0.806	92.261	02 - Faris
136	3149; 3476	327	1.000	0.959	0.778	105.844	03 - Mudar
189	3677; 3885	208	0.999	0.957	0.787	62.880	02 - AFTec
243	3918; 4180	262	0.999	0.941	0.807	196.092	02 - Lenne
159	4215; 4548	333	1.000	0.968	0.722	114.775	093 - BKMZ
Eval :							
TP	FP	FN	Spec.TN	Spec.FP	Acc.	Prec.	Spec.
3932	31	596	0	0	0.86	0.99	-

(b) Results

Figure 8.15: Automatic sequence detection on mixotic-set 285.

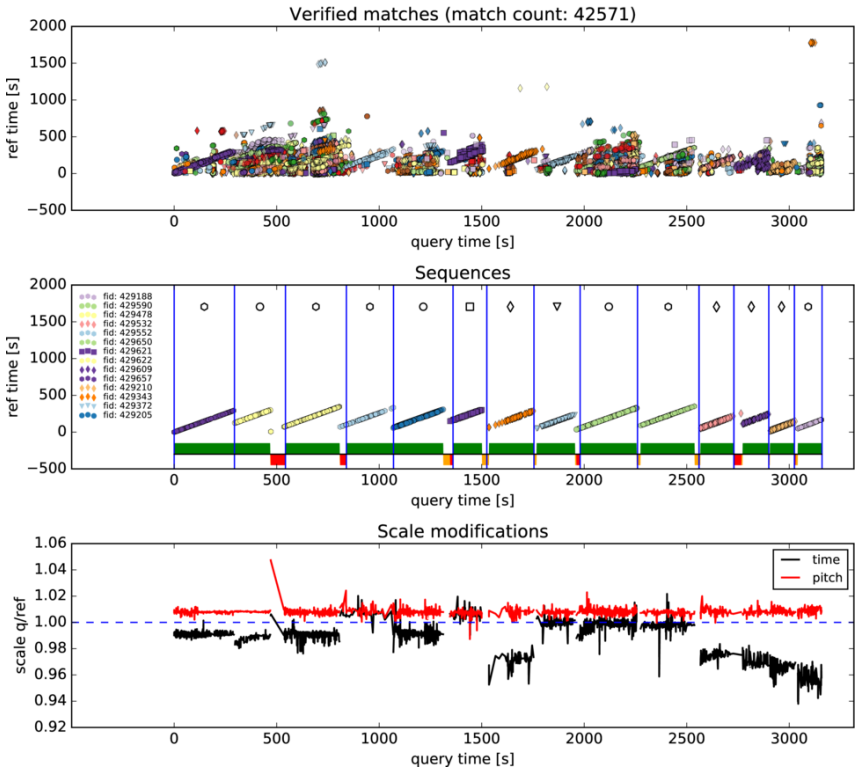
8.4.11 *Set 286*

The results for this mix are presented in Figure 8.16.

- Number of identifiable tracks: 14
- Number of non-identifiable tracks: 0
- Accuracy: 0.89
- Precision: 0.93
- Specificity: –

Apart from one sequence, which (according to the ground-truth) seems to start far too early (see the region at second 500), everything is mostly correctly detected, but some mistakes are made around the song borders

All songs are played approximately one percent faster, but two performed tracks in this mix stand out: Sequences number 5 and 7 (around second 1000 and 1500, respectively) are played slower, but also with a higher pitch. These cases are examples of a *combination* of time as well as pitch scale modifications.



(a) Sequence detection

set286							
nm	tquery [s]	len	*pf.est*	*tf.est*	*vscore*	*score*	Title
670	0; 292	292	1.008	0.991	0.880	589.388	Zimmer119.
210	297; 468	171	1.008	0.988	0.928	92.265	Zimmer119.
559	471; 807	336	1.008	0.991	0.863	482.676	Zimmer120.
108	810; 1069	259	1.009	1.007	0.799	74.806	Zimmer121.
1036	1069; 1312	243	1.008	0.991	0.867	612.419	Zimmer121.
196	1346; 1501	155	1.007	1.006	0.842	165.011	Zimmer121.
97	1536; 1753	217	1.008	0.971	0.824	67.412	Zimmer121.
230	1768; 1955	187	1.008	1.000	0.844	194.045	01_hermeti
777	1962; 2255	293	1.008	0.999	0.860	319.258	02_hermeti
234	2275; 2538	263	1.008	0.998	0.830	194.264	Track_Jack
147	2567; 2766	199	1.008	0.976	0.853	125.427	Track_Jack
78	2773; 2900	127	1.008	0.971	0.837	59.702	A3 Miami S
122	2906; 3031	125	1.009	0.966	0.844	102.972	B1 Miami S
145	3042; 3157	115	1.009	0.958	0.811	58.068	Track_Jack
Eval :							
TP	FP	FN	Spec.TN	Spec.FP	Acc.	Prec.	Spec.
2815	180	164	0	0	0.89	0.93	-

(b) Results

Figure 8.16: Automatic sequence detection on mixotic-set 286.

8.5 DISCUSSION

In this chapter we demonstrated how to take advantage of first collecting all fingerprinted match-candidates of an entire query recording, and then use this collection to segment the query recording and report sequences. We refer to this as a "global" approach to sequence detection.

We demonstrate that the use of global information supports the capabilities of an automated system to maintain a higher identification performance in comparison to short, local query snippets as used in Chapter 7. For interactive post-processing the gradually refined visualisations seem to help – one can see regions of the mix where a lot of repetitive content is played, and where the largest amount of competing file-IDs are returned. One could change the sequence detector parameters in an interactive fashion to clean up complicated regions of the result set. On the other hand, when considering the local query results that were stripped of weak results beforehand, information is lost that otherwise could turn out to be helpful to the task. We argue that for interactive post-processing this kind of visualisations help to speed up the validation, and allow a user to focus on apparently incorrect, or dubious results.

For fully automatic sequence detection this approach successfully increases all three performance measures, i.e. the accuracy, precision and specificity. On our dataset, the most impact for the increase in performance is the successful deletion of false positive sequences, for which simpler and cheaper approaches might work as well. However, we could in fact increase the accuracy by 5 percentage points on the mixotic-set and by 8 percentage points on the disco-set.

Note that the sequence detector parameters should be tuned on large and independent data sets. Unfortunately, we currently do not have enough data for an independent parameter study.

9 CONCLUSION

This thesis has presented our work on audio fingerprinting. We aimed at solving the complex problem of building an automated system that is practically applicable to a wide variety of use cases, most notably, to a topic that is of high relevance to the industry: the monitoring of DJ performances for transparent distribution of revenue.

For the application to a wide variety of use cases, systems have to be robust to background noise, introduced signal artifacts caused by lossy audio compression, and most importantly, scale modifications in time and frequency. These requirements have to be met without limiting the identification performance of systems, or their applicability to large-scale tasks.

For this, we defined a set of robustness requirements that have to be met for system application to media monitoring and audio copy detection tasks. To do so we cite sources that observe the importance of systems being robust to changes in the pitch and time scale of query audio. We further investigate DJ performances to establish a lower bound of $\pm 5\%$ of scale robustness. However, it turned out that this lower bound seems to be too optimistic: in our small datasets we encounter scale changes of up to 10%, which calls for even more robust systems. To meet this demand, we designed Qfp to be highly robust, and evaluate all experiments in the range $\pm 30\%$ of scale modifications.

Studying previous and related work in the academic literature, we concluded that at the time we started working on the topic, the task of building an automated system that can meet the criteria indeed was an unsolved research problem.

In this thesis we propose a novel audio fingerprinting method, "Qfp". It is based on simple and elegant methods that have been discovered by researchers working in the domain of astronomy. According to the rich evaluation reported in this thesis, Qfp seems to be able to meet the established robustness criteria. To our knowledge, Qfp is the first system to be proposed in the academic literature, that can satisfy all of the defined criteria at the same time – even if the lower bound of scale robustness is to be raised from 5% to 10%.

Due to the compact fingerprints used in Qfp, the four-dimensional hash space can be efficiently searched via range search techniques. We demonstrated high identification performance with low processing run times in experiments of more than a million of queries with various distortions, against a large database consisting of up to 430,000 full-length songs (roughly 3.37 years of music). While there is high potential of false positive matches in a database of this size (roughly 757 million

quads) in combination with the rather large scale tolerances of $\pm 30\%$ of our system, the proposed filtering stage and verification of match sequence hypotheses enable the system to maintain high precision and specificity, even for musical genres with highly repetitive content. The database is created from freely available (for non-commercial use) Creative-Commons licensed tracks that are hosted by the Jamendo service, i.e. we perform the experiments on a reproducible reference collection.

We assume that the monitoring of DJ performances could be seen as the "grand challenge" of audio fingerprinting, due to the enormous robustness demands that are caused by the large degree of freedom for DJs to introduce effects that modify the audio signal. For evaluating the system, we aimed at constructing a realistic test scenario and tested the system on real world data – actual DJ performances over a variety of music genres. In the experiments, Qfp is compared to two further methods, both of which are freely available for research purposes. With this setup we intend to facilitate reproducible research. The method we propose turned out to operate at considerably higher performance than the other tested methods. The results allow to observe that this use-case indeed reveals issues that are not encountered when evaluating systems on manually crafted evaluation data. To facilitate research in this direction, we created and published a free dataset for this task.

Then, we proceed to append the reference tracks of our two DJ performance datasets to our large reference database, in order to perform experiments that, we believe, are quite realistic scenarios for the application in industry, because of the large reference collection. Here, apart from maintaining high identification performance, systems must be able to operate on very large reference collections, and at the same time operate with high efficiency. The results highlight that our proposed system seems to be a highly promising method.

To further increase the identification performance for media monitoring, we propose to build a sequence detector on top of Qfp. The underlying idea is to first collect all match-candidates by fingerprinting the entire query recording to then filter this global result set using a sequence detector. We observe that despite the verification process of Qfp, the system reports large numbers of potentially correct match hypotheses of various, contradicting file-IDs. This is to be expected in the presence of highly repetitive electronic music. To increase the performance of the system in these cases, we perform sequence estimation on global context, rather than on successive short and local queries. This also seems to be a promising way to accelerate the verification of results in a supervised setting, where manual efforts must be invested to validate the system output. While the sequence detection on global results is a more runtime demanding process than performing local queries, we show that the proposed system is highly beneficial in

terms of accuracy, precision and specificity. According to a case study that we perform in this thesis, we could increase the overall identification performance. In terms of accuracy, precision and specificity, for the first dataset (called "disco-set") we achieved an increase in accuracy by 8 percentage points, precision by 14 percentage points, and specificity by 20 percentage points, respectively. On the second dataset (called "mixotic-set"), we could improve the results by 5, 9, and 34 percentage points. However, it remains to be seen how this method performs on a large and representative dataset.

In the course of this thesis we established, via extensive experimentation, that our own proposed method "Qfp" reports reliable results and is efficiently applicable to large collections of data. Qfp seems to be a promising method to be applied to a task that is highly demanded by artists: artists claim the right of *fair* and *transparent* distribution of revenue. For this, it is not sufficient to simply report which tracks occurred in the processed recordings. Rather, to distribute revenue in an accurate and fair way, it is important to assess the amount of seconds that a given song was played. In our experiments on media monitoring, our evaluation methodology is designed to report this kind of information in detail.

For future analysis and experiments in this direction we think it would be worthwhile to collect DJ mixes with accurate annotations and timestamps that are exported from the specific software or the midi controller used by the DJ. This would allow to gain insight on what kinds of effects and combinations thereof prevent automated identification systems from correctly identifying certain portions of query audio.

We hope that our contributions can complement the active field of research on audio fingerprinting, and that the level of detail in our evaluation methodology, together with reproducible reference data collections [1], allows our system to serve as a baseline for further research in this field.

1 The information to reproduce the data is available at http://www.cp.jku.at/datasets/fingerprinting/

LIST OF FIGURES

BIBLIOGRAPHY

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

Baluja, S. and M. Covell (2007). "Audio Fingerprinting: Combining Computer Vision and Data Stream Processing". In: *IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP 2007)*. Vol. 2, pp. 213–216.

Baluja, S. and M. Covell (2008). "Waveprint: Efficient Wavelet-Based Audio Fingerprinting". In: *Pattern Recognition* 41.11, pp. 3467–3480. ISSN: 0031-3203.

Bardeli, R. and F. Kurth (2004). "Robust Identification of Time-Scaled Audio". In: *Audio Engineering Society (AES 2004), 25th International Conference on Metadata for Audio, London, UK*.

Batlle, E., J. Masip, E. Guaus, and P. Cano (2004). "Scalability Issues in an HMM-Based Audio Fingerprinting". In: *IEEE International Conference on Multimedia and Expo (ICME 2004)*. Vol. 1, 735–738 Vol.1.

Batlle, E., J. Masip, and E. Guaus (2002). "Automatic Song Identification in Noisy Broadcast Audio". In: *International Conference on Signal and Image Processing (IASTED 2002)*.

Bellettini, C. and G. Mazzini (2008). "Reliable Automatic Recognition for Pitch-Shifted Audio". In: *Proceedings of 17th International Conference on Computer Communications and Networks, ICCCN'08, St. Thomas, U.S. Virgin Islands, August 3-7, 2008*. IEEE, pp. 838–843.

Betser, M., P. Collen, and J.-B. Rault (2007). "Audio Identification Using Sinusoidal Modeling and Application to Jingle Detection." In: *Proceedings of the 8th International Conference on Music Information Retrieval (ISMIR 2007)*. Vienna, Austria, pp. 139–142.

Burges, C. J. C., D. Plastina, J. C. Platt, E. Renshaw, and H. S. Malvar (2005). "Using Audio Fingerprinting for Duplicate Detection and Thumbnail Generation". In: *IEEE International Conference on Acoustics, Speech, and Signal Processing (ICASSP 2005)*. Vol. 3, iii/9–iii12 Vol. 3.

Burges, C. J., J. C. Platt, and S. Jana (2002). "Extracting Noise-Robust Features from Audio Data". In: *IEEE International Conference on Acoustics, Speech, and Signal Processing (ICASSP 2002)*. Vol. 1. IEEE, pp. I–1021.

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

Cano Vila, P. and X. Serra (2007). *Content-Based Audio Search: from Fingerprinting to Semantic Audio Retrieval*. English. Barcelona: Universitat Pompeu Fabra. ISBN: 978-84-691-1205-2.

Cano, P., E. Batlle, T. Kalker, and J. Haitsma (2002). "A Review of Algorithms for Audio Fingerprinting". In: *IEEE Workshop on Multimedia Signal Processing*, pp. 169–173.

Cano, P., E. Batlle, T. Kalker, and J. Haitsma (2005). "A Review of Audio Fingerprinting". en. In: *Journal of VLSI signal processing systems for signal, image and video technology* 41.3, pp. 271–284. ISSN: 0922-5773.

Cano, P., E. Batlle, H. Mayer, and H. Neuschmied (2002). "Robust Sound Modeling for Song Detection in Broadcast Audio". In: pp. 1–7.

Casey, M., C. Rhodes, and M. Slaney (2008). "Analysis of Minimum Distances in High-Dimensional Musical Spaces". In: *Trans. Audio, Speech and Lang. Proc.* 16.5, pp. 1015–1028. ISSN: 1558-7916.

Chandrasekhar, V., M. Sharifi, and D. A. Ross (2011). "Survey and Evaluation of Audio Fingerprinting Schemes for Mobile Query-by-Example Applications." In: *International Society for Music Information Retrieval Conference (ISMIR 2011)*. Vol. 20, pp. 801–806.

Dammertz, H., J. Hanika, and A. Keller (2008). "Shallow Bounding Volume Hierarchies for Fast SIMD Ray Tracing of Incoherent Rays". In: *Proceedings Eurographics Conference on Rendering*. Aire-la-Ville, Switzerland, Switzerland: Eurographics Association, pp. 1225–1233.

Dupraz, E. and G. Richard (2010). "Robust Frequency-Based Audio Fingerprinting". In: *IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP 2010)*, pp. 281–284.

Ellis, D. P. and G. E. Poliner (2007). "Identifying 'Cover Songs' with Chroma Features and Dynamic Programming Beat Tracking". In: *Proceedings of the IEEE International Conference on Acoustics, Speech, and Signal Processing (ICASSP 2007)*. Vol. 4. Honolulu, Hawaii, USA.

Evangelidis, G. and C. Bauckhage (2011). "Efficient and Robust Alignment of Unsynchronized Video Sequences." In: *DAGM-Symposium*. Vol. 6835. Lecture Notes in Computer Science. Springer, pp. 286–295. ISBN: 978-3-642-23122-3.

Evangelidis, G. and C. Bauckhage (2013). "Efficient Subframe Video Alignment Using Short Descriptors." In: *IEEE Trans. Pattern Anal. Mach. Intell.* 35.10, pp. 2371–2386.

Fenet, S. (2013). "PhD thesis. Telecom-ParisTech". PhD thesis. telecom-paristech.

Fenet, S., G. Richard, and Y. Grenier (2011). "A Scalable Audio Fingerprint Method with Robustness to Pitch-Shifting." In: *Proceedings of the 12th International Conference on Music Information Retrieval (ISMIR 2011)*, pp. 121–126.

Fujishima, T. (1999). "Realtime Chord Recognition of Musical Sound: a System Using Common Lisp Music". In: *Proceedings of the 1999 International Computer Music Conference, ICMC 1999, Beijing, China, October 22-27, 1999*.

Goldsmith, J. and J. Salmon (1987). "Automatic Creation of Object Hierarchies for Ray Tracing". In: *IEEE Computer Graphics and Applications* 7.5, pp. 14–20.

Goldstein, J., J. C. Plat, and C. J. Burges (2005). "Redundant Bit Vectors for Quickly Searching High-Dimensional Regions". In: *Deterministic and Statistical Methods in Machine Learning*. Springer, pp. 137–158.

Goto, M. (2003). "A Chorus-Section Detecting Method for Musical Audio Signals". In: *Proceedings of the IEEE International Conference on Acoustics, Speech, and Signal Processing (ICASSP 2003)*. Hong Kong, China, pp. 437–440.

Grosche, P. and M. Müller (2012). "Toward Characteristic Audio Shingles for Efficient Cross-Version Music Retrieval". In: *IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP 2012), Kyoto, Japan, March 25-30, 2012*, pp. 473–476.

Grosche, P., M. Müller, and J. Serrà (2012). "Audio Content-Based Music Retrieval". In: *Multimodal Music Processing*. Ed. by M. Müller, M. Goto, and M. Schedl. Vol. 3. Dagstuhl Follow-Ups. Dagstuhl, Germany: Schloss Dagstuhl–Leibniz-Zentrum für Informatik, pp. 157–174.

Guttman, A. (1984). "R-trees: A Dynamic Index Structure for Spatial Searching". In: *SIGMOD Rec.* 14.2, pp. 47–57. ISSN: 0163-5808.

Haitsma, J. and T. Kalker (2002). "A Highly Robust Audio Fingerprinting System". In: *International Society for Music Information Retrieval Conference (ISMIR 2002)*.

Haitsma, J. and T. Kalker (2003). "Speed-Change Resistant Audio Fingerprinting using Auto-Correlation". In: *IEEE International Conference on Acoustics, Speech, and Signal Processing (ICASSP '03)*. Vol. 4, pp. 728–731.

Haitsma, J., T. Kalker, and J. Oostveen (2001). "Robust Audio Hashing for Content Identification". In: *International Workshop on Content-Based Multimedia Indexing*. Vol. 4. Citeseer, pp. 117–124.

Herre, J., O. Hellmuth, and M. Cremer (2002). "Scalable Robust Audio Fingerprinting using MPEG-7 Content Description". In: *IEEE Workshop on Multimedia Signal Processing*, pp. 165–168.

Jain, A., A. Ross, and K. Nandakumar (2011). *Introduction to Biometrics*. SpringerLink : Bücher. Springer US. ISBN: 9780387773261.

"Jamendo Service". Available at https://www.jamendo.com.

Ke, Y., D. Hoiem, and R. Sukthankar (2005). "Computer Vision for Music Identification". In: *2005 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR 2005), 20-26 June 2005, San Diego, CA, USA*, pp. 597–604.

Kurth, F., T. Gehrmann, and M. Müller (2006). "The Cyclic Beat Spectrum: Tempo-Related Audio Features for Time-Scale Invariant Audio Identification." In: *International Society for Music Information Retrieval Conference (ISMIR 2011)*, pp. 35–40.

Kurth, F. and M. Müller (2008). "Efficient Index-Based Audio Matching". In: *IEEE Transactions on Audio, Speech, and Language Processing* 16.2, pp. 382–395.

Kurth, F., A. Ribbrock, and M. Clausen (2002). "Identification of Highly Distorted Audio Material for Querying Large Scale Data Bases". In: *International Audio Engineering Society Convention 112*. Audio Engineering Society.

Lang, D., D. Hogg, K. Mierle, M. Blanton, and S. Roweis (2010). "Astrometry.net: Blind Astrometric Calibration of Arbitrary Astronomical Images". In: *The Astronomical Journal* 137, pp. 1782–2800.

"libspatialindex". Available at https://libspatialindex.github.io/.

Liu, Y., H. S. Yun, and N. S. Kim (2009). "Audio Fingerprinting Based on Multiple Hashing in DCT Domain". In: *IEEE Signal Processing Letters* 16.6, pp. 525–528. ISSN: 1070-9908.

Lu, C.-S. (2002). "Audio Fingerprinting based on Analyzing Time-Frequency Localization of Signals". In: *2002 IEEE Workshop on Multimedia Signal Processing*, pp. 174–177.

Malekesmaeili, M. and R. K. Ward (2014). "A Local Fingerprinting Approach for Audio Copy Detection". In: *Signal Processing* 98, pp. 308–321.

Miller, M. L., M. A. Rodriguez, and I. J. Cox (2002). "Audio Fingerprinting: Nearest Neighbor Search in High Dimensional Binary Spaces". In: *IEEE Workshop on Multimedia Signal Processing*, pp. 182–185.

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

"Ogg Vorbis". Available at http://www.vorbis.com.

Ogle, J. P. and D. P. Ellis (2007). "Fingerprinting to Identify Repeated Sound Events in Long-Duration Personal Audio Recordings". In: *IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP 2007)*. Vol. 1. IEEE, pp. I–233.

Ouali, C., P. Dumouchel, and V. Gupta (2014). "A Robust Audio Fingerprinting Method for Content-Based Copy Detection". In: *12th International Workshop on Content-Based Multimedia Indexing (CBMI), 2014*. IEEE, pp. 1–6.

Porter, A. (2012). "Evaluating Musical Fingerprinting Systems". PhD thesis. McGill University.

Rafii, Z., B. Coover, and J. Han (2014). "An Audio Fingerprinting System for Live Version Identification using Image Processing Techniques". In: *IEEE International Conference on Acoustics, Speech*

and Signal Processing, (ICASSP 2014), Florence, Italy, May 4-9, 2014. IEEE, pp. 644–648.

Ramalingam, A. and S. Krishnan (2006). "Gaussian Mixture Modeling of Short-Time Fourier Transform Features for Audio Fingerprinting". In: *IEEE Transactions on Information Forensics and Security* 1.4, pp. 457–463. ISSN: 1556-6013.

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

Ramona, M. and G. Peeters (2013). "AudioPrint: An Efficient Audio Fingerprint System Based on a Novel Cost-Less Synchronization Scheme". In: *IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP 2013)*. IEEE, pp. 818–822.

Rotteker, V. (2016). "Evaluation of Pitch Shift and Time Stretch Invariant Acoustic Fingerprinting Systems". MA thesis. Technische Universität Berlin.

Schreiber, H. and M. Müller (2014). "Accelerating Index-Based Audio Identification". In: *IEEE Transactions on Multimedia* 16.6, pp. 1654–1664. ISSN: 1520-9210, 1941-0077.

Seo, J. S., M. Jin, S. Lee, D. Jang, S. Lee, and C. D. Yoo (2005). "Audio Fingerprinting based on Normalized Spectral Subband Centroids". In: *IEEE International Conference on Acoustics, Speech, and Signal Processing, 2005. Proceedings.(ICASSP 2005).* Vol. 3. IEEE, pp. iii–213.

Serrà, J. and E. Gómez (2008). "Audio Cover Song Identification Based on Tonal Sequence Alignment". In: *IEEE International Conference on Acoustics, Speech and Signal processing (ICASSP 2008)*. Las Vegas, USA, pp. 61–64.

Six, J. and O. Cornelis (2012). "A Robust Audio Fingerprinter Based on Pitch Class Histograms Applications for Ethnic Music Archives". In: *Proceedings of the Folk Music Analysis conference (FMA 2012)*.

Six, J. and M. Leman (2014). "Panako: a Scalable Acoustic Fingerprinting System Handling Time-Scale And Pitch Modification". In: *15th International Society for Music Information Retrieval Conference (ISMIR 2014)*.

Smeaton, A. F., P. Over, and W. Kraaij (2006). "Evaluation Campaigns and TRECVid". In: *Proceedings of the International Workshop on Multimedia Information Retrieval*. New York, NY, USA: ACM Press, pp. 321–330. ISBN: 1-59593-495-2.

Sonnleitner, R. and G. Widmer (2014). "Quad-Based Audio Fingerprinting Robust to Time and Frequency Scaling". In: *Proceedings of the International Conference on Digital Audio Effects (DAFX 2014)*. Erlangen, Germany, pp. 173–180.

Sonnleitner, R., A. Arzt, and G. Widmer (2016). "Landmark-Based Audio Fingerprinting for DJ Mix Monitoring". In: *17th International Society for Music Information Retrieval Conference (ISMIR 2016)*.

Sonnleitner, R. and G. Widmer (2016). "Robust Quad-Based Audio Fingerprinting". In: *IEEE/ACM Trans. Audio, Speech & Language Processing* 24.3, pp. 409–421.

"SoX - Sound eXchange". Available at http://sox.sourceforge.net/.

Sukittanon, S. and L. E. Atlas (2002). "Modulation Frequency Features for Audio Fingerprinting". In: *IEEE International Conference on Acoustics, Speech, and Signal Processing (ICASSP 2002)*. Vol. 2, pp. 1773–1776.

Tsai, T., T. Prätzlich, and M. Müller (2016). "Known-Artist Live Song ID: A Hashprint Approach". In: *Proceedings of the International Conference on Music Information Retrieval (ISMIR 2016)*. New York, USA, pp. 427–433.

Tsakok, J. A. (2009). "Faster Incoherent Rays: Multi-BVH Ray Stream Tracing". In: *Proceedings of the Conference on High Performance Graphics 2009*. HPG '09. New York, NY, USA: ACM, pp. 151–158.

Wakefield, G. (1999). "Mathematical Representation of Joint Time-Chroma Distributions". In: *International Symposium on Optical Science, Engineering, and Instrumentation, SPIE*. Vol. 99, pp. 18–23.

Wald, I. (2007). "On Fast Construction of SAH-based Bounding Volume Hierarchies". In: *Proc. Symposium on Interactive Ray Tracing*. Washington, DC, USA: IEEE Computer Society, pp. 33–40.

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

Yu, C., R. Wang, J. Xiao, and J. Sun (2014). "High Performance Indexing for Massive Audio Fingerprint Data". In: *IEEE Transactions on Consumer Electronics* 60.4, pp. 690–695. ISSN: 0098-3063.

Zhang, X., B. Zhu, L. Li, W. Li, X. Li, W. Wang, P. Lu, and W. Zhang (2015). "SIFT-Based Local Spectrogram Image Descriptor: a Novel Feature for Robust Music Identification". en. In: *EURASIP Journal on Audio, Speech, and Music Processing* 2015.1. ISSN: 1687-4722.

Zhu, B., W. Li, Z. Wang, and X. Xue (2010). "A Novel Audio Fingerprinting Method Robust to Time Scale Modification and Pitch Shifting". In: *Proceedings of the 18th ACM international conference on Multimedia*. ACM, pp. 987–990.