Robust Quad-Based Audio Fingerprinting

Reinhard Sonnleitner and Gerhard Widmer

***Abstract*—We propose an audio fingerprinting method that adapts findings from the field of blind astrometry to define simple, efficiently representable characteristic feature combinations called *quads*. Based on these, an audio identification algorithm is described that is robust to noise and severe time-frequency scale distortions and accurately identifies the underlying scale transform factors. The low number and compact representation of content features allows for efficient application of exact fixed-radius near-neighbour search methods for fingerprint matching in large audio collections. We demonstrate the practicability of the method on a collection of 100,000 songs, analyse its performance for a diverse set of noise as well as severe speed, tempo and pitch scale modifications, and identify a number of advantages of our method over two state-of-the-art distortion-robust audio identification algorithms.**

I. Introduction

An audio fingerprinting system identifies a piece of audio from a large collection, given a short query. This is typically done by extracting highly discriminative content features, a "fingerprint", from the collection of audio files as well as the query piece, and subsequently comparing these features.

Established use cases range from the identification of audio which is played in noisy environments to the very large scale application areas of media monitoring as well as copy and plagiarism detection.

Fingerprinting systems used for media monitoring usually analyse audio streams of a large number of broadcast channels or recordings from dance club events, in order to compile lists of the identified content that was played at any given time. As these systems operate on huge amounts of data, the involved data structures should be as compact as possible, and the systems must efficiently operate on large reference databases.

Depending on the application, audio fingerprinting systems should be robust to different kinds of distortions of query audio material. The minimum requirement is robustness to signal distortions from noise and various types of lossy audio compression. Further requirements arise for the task of media monitoring. While for such systems the robustness to noise may not be the main concern, the systems need to recognize audio material that was modified in tempo and/or pitch, and must efficiently operate on large collections of audio. The most challenging application area seems to be the monitoring of DJ sets and DJ (live) performances, due to the large degree of freedom of introduced signal modifications.

The National Institute of Standards and Technology (NIST) acknowledges the probable encounter with heavily modified audio/video content in the scope of automated copy detection systems, and includes several severely distorted files in their TRECVID 2010 copy detection dataset [1]. According to an investigation mentioned in [2], this dataset apparently contains files that are altered in speed from -180% up to $+23\%$.

The main robustness requirements we wish to address in this paper are: robustness to large time and frequency scale distortions in combination with efficient operation on large reference audio collections.

The described signal modifications prove to be substantial challenges for automated audio identification systems, and no method described in the academic literature is shown to satisfy all of these requirements[1] (see Section II).

In this paper, we present a solution to the critical task of developing a system that fulfills the scale change robustness requirements and can efficiently handle large audio collections. While it is directly applicable to established use cases, we think that it also presents a reasonable basis for further study of track identification in DJ mix sets and electronic music performances.

The proposed method is a thoroughly refined and substantially extended version of our initial work on that topic [3][2]. The system we present is, to our knowledge, the first audio fingerprinting method described in the academic literature that meets all of the above requirements. It can efficiently identify audio in large collections, and is robust to noise and audio quality degradation, as well as to severe distortions of speed, tempo and frequency [3]. Moreover, the identification method uncovers and accurately quantifies the scale transform factors of distorted audio queries as an integral part of its matching process.

To make this possible, we adopt research from the field of blind astrometry [4] to the audio domain. The central components in this are local, compact geometric hash representations of audio content that are invariant to translation and scaling. The system thereby overcomes the inherent robustness limitations of methods that depend on equal relative distances of reference and query features, such as the well-known Shazam algorithm [5].

This paper has supplementary downloadable material available at http://ieeexplore.ieee.org, provided by the authors. It includes programs to create all audio data and manipulated queries as used in this paper. This material is 1.3 MB in size.

R. Sonnleitner and G. Widmer are with the Department of Computational Perception, Johannes Kepler University, 4040 Linz, Austria (e-mail: reinhard.sonnleitner@jku.at; gerhard.widmer@jku.at).

[1]There exists an US patent (US7627477B2) covering a system that is claimed to be *immune* to various transformations, including time/frequency scale modifications.

[2]We would like to point the readers who are familiar with [3] to Section VII, which highlights the differences and novelties of the proposed method.

[3]To clarify our terminology: if both scales are changed by the same scale factor, we call this a change in *"speed"*: the song is played faster and at the same time at a higher pitch, which is usually achieved by resampling the audio at different rate. Changing the time scale only will be referred to as a *"tempo"* change: here, the audio is sped up or slowed down without observable changes in pitch. Vice versa, if only the frequency scale is modified, this will be called *"pitch shifting"*.

We analyse the performance of the method based on a freely available audio collection of 100,000 full length songs, which is considerably larger than what is used in the literature, and perform experiments on roughly 450,000 queries that are distorted in speed, tempo and frequency by up to $\pm 30\%$ relative to the reference audio piece. 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.

The paper is organized as follows. Section II discusses related work and identifies two state-of-the-art methods [6], [7], that will act as our main reference here. Section III gives a brief overview of the main points of our new method, in order to set the context for the precise method description, which comes in three parts: Section IV describes the feature extraction process and how to obtain hash representations from these features, which then together constitute the fingerprints. Section V describes the data structures that are used for the audio identification method. Section VI details the identification method, i.e. the process of matching query audio with reference data by comparing the fingerprints. Section VII highlights notable differences and novelties with respect to our previous work [3]. Section VIII systematically evaluates the performance of our method on roughly 450,000 queries with various properties.

II. Related work

From the numerous fingerprinting methods described in the literature we first consider those that can at least meet the following requirements: the algorithms need to operate on excerpts rather than the whole audio file as it is represented in the reference database. In general, such systems extract and compare *local* audio content features. Next, the methods need to be robust to signal degradations like noise and distortions that result from encoding the audio with lossy compression algorithms.

An audio identification system for copy detection is presented in [2]. 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 speed 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 efficiently usable on larger reference collections. The system is evaluated on the TRECVID 2010 dataset [1].

A landmark based system with robustness to moderate speed changes in the range of $\pm 5\%$ is presented in [8]. 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. Similar to [2], this is a limiting factor regarding applicability to larger databases.

The work presented in [9] applies cosine filters to the audio spectrum to compute short-term band energies, which makes the method robust to moderate frequency distortions. The method is perfectly usable with large collections of audio and is robust to audio degradations and to small amounts of tempo and pitch scale distortions.

In [10], audio identification is approached by applying methods from computer vision. 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. The method is very robust to tempo changes in the tested range of $\pm 10\%$, but sensitive to moderate speed changes in the range of $\pm 2\%$.

Regarding the main focus of our work – robustness to *both* time and frequency scale distortions – we could identify three publications which suggest solutions to these criteria. However, to our knowledge, no audio fingerprinting algorithm has been described that in addition to exhibiting large scale change robustness is shown to efficiently operate on large reference collections.

In [6] a method is described that performs the scale invariant feature transform (SIFT [11]) on logarithmically scaled audio spectrograms, compressed into 64 frequency sub-bands. The resulting 128-dimensional SIFT-descriptors represent the fingerprints. Individual fingerprints are matched via nearest neighbour search using locality sensitive hashing (LSH [12]). The results show exceptionally high accuracy for severe speed, tempo and pitch distortions. The evaluation is performed on a database of 10,141 reference audio excerpts of length $60s$. Given the typical average song duration of four minutes, this is the equivalent of about 2535 full length songs.

Another method which focuses on the mentioned challenges is presented in [7]. Its fingerprint hashes consist of triples of feature components similar to what is used in the context of symbolic fingerprinting in [13]. The evaluation of [7] shows high sensitivity to moderate scale distortions of either kind, which limits the applicability of the method. Due to its high efficiency, the method is applicable to large databases. The work shows an evaluation of identification performance under various conditions on the largest audio collection used in this specific context throughout the literature, analysing the performance of the method on 30.000 full length songs. It operates on data structures of small size: the estimated size of a reference database for a million songs is just roughly 28GB.

A fingerprinting algorithm for audio copy detection, that also meets the robustness demands for time-frequency distortions is [14]. The work reports near perfect percentages of correct song association, although on a rather small reference database of roughly 250 songs. The method performs feature extraction on a two-dimensional time-chroma representation of the audio. From this, image patches are extracted and then analysed, and a number of low frequency DCT coefficients of the image patches form the individual 143-dimensional fingerprints. Selection of match candidates is performed by exhaustive nearest neighbour lookups. Compared to other methods, the extraction of fingerprints seems computationally expensive, and because of the exhaustive candidate lookup the

method is not readily applicable to larger reference databases.

The methods described in both publications [6] and [14] report exceptionally high performance for this challenging task and their methods are more robust to scale distortions than other previously described work. However, the system proposed in [7] seems less demanding in terms of computational costs as well as disk space requirements, and seems to be the only method applicable for large scale fingerprinting tasks while still exhibiting limited robustness to moderate scale distortions. We will take the recent work of [6] and [7] as the reference methods in the present paper, because of the strong robustness shown in [6], and the efficiency of [7]. In addition, the information presented in [7] readily enables comparability.

III. Method overview

Our identification method operates on compact four-dimensional, continuous geometric hash representations of quadruples of points, henceforth referred to as "quads" [4]. The points are local maxima in the two-dimensional time-frequency representation of reference audio material, and are referred to as "spectral peaks" throughout the paper. 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 IV 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 fixed-radius near neighbour queries. This stage of the method is described in Section V.

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 index 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 a powerful verification method, which is the key technique that makes the identification algorithm perform with high precision. These steps are described in Section VI.

IV. Feature extraction

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

To begin with, all audio files are downmixed to one-channel monaural representations sampled at 16 kHz. We compute the STFT magnitude spectrogram using a Hann-window of size 2048 samples (128 ms) and a hop size of 64 samples (4 ms), discarding the phases.

A. Constructing quads

The fingerprinting algorithm works on translation- and scale-invariant geometric 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 2D spectrogram space will be used extensively in the following, let us formally introduce the notation:

$$P = (P_x, P_y)$$

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

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 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 seems well applicable to audio spectrograms.

The resulting peak coordinates are now subjected to parabolic interpolation based on their neighbourhood of 3×3 in spectrogram space. If the interpolated value lies outside the neighbourhood in any dimension, the original non-interpolated value is kept. 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* [4], where a quad consists of four spectral peaks A, B, C, D.

$$A_y < B_y \tag{1a}$$
$$A_x < C_x \leq D_x \leq B_x \tag{1b}$$
$$A_y < C_y, D_y \leq B_y \tag{1c}$$

Thus, we define a quad to be *valid* if the points C, D reside in the axis-parallel rectangle that is spanned by 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[4]: we construct a *quad grouping region* of 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 1a. 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 [3]: we attain an almost uniform distribution of quads over time (i.e. per bin). Most importantly, this grouping method extracts the robust quads from the set of all possible quads. Applying this proposed grouping method, we observe increased identification performance of the fingerprinter. Indeed, if the reference database consists of only strong quads, we can simply discard weak quads from query audio to considerably reduce the necessary amount of work for the identification task.

B. 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 to detect. 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} is computed from the reference max filter width m_w^{ref} and ϵ_t using Equation 2:

$$m_w^{query} = m_w^{ref}/(1+\epsilon_t) \tag{2}$$

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

$$m_h^{query} = m_h^{ref} \cdot (1-\epsilon_p) \tag{3}$$

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

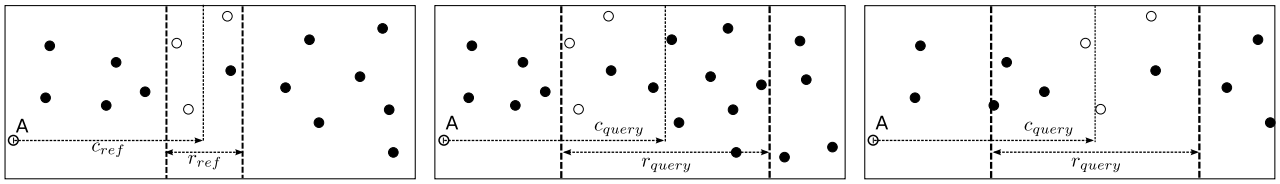
$$r_{start}^{query} = r_{start}^{ref}/(1+\epsilon_t) \tag{4a}$$
$$r_{end}^{query} = r_{end}^{ref}/(1-\epsilon_t) \tag{4b}$$
$$c^{query} = (r_{start}^{query} + r_{end}^{query})/2 \tag{4c}$$

The resulting peaks for a piece of reference and query audio for tolerances $\epsilon_p = \epsilon_t = 0.3$ are depicted in Figure 3. Note that the query peaks (shown as dots) are extracted at a higher density, as given in Equations 2, 3. The other aspects of Figure 3 become relevant in Section VI-C.

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 1, which shows the grouping for a quad for a given root point A. In 1a 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 1b and 1c, respectively. We see that the hollow points, which are 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) 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 1b and 1c show the locations of reference peaks after altering the time scale, but does 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

[4]We will parametrize this process differently, depending on whether we compute quads for the reference database, or for a piece of query audio. To create the reference database, 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 number of quads. The explanation for this will be given later in this section.



(a) Reference quad grouping (b) Query snippet with tempo 130% (c) Query snippet with tempo 70%

Fig. 1. Reference quad grouping (1a) and query quad grouping with increased tempo (1b), and decreased tempo (1c). 1a shows a root point A and the hollow circles B, C, D to group a reference quad. Figures 1b, 1c show the regions for the tolerance of $\pm 30\%$ in the queries, and the different peak densities. Note that r^{query} is scaled according to Equations 4, rather than proportionally to the tolerance bounds of $\pm 30\%$

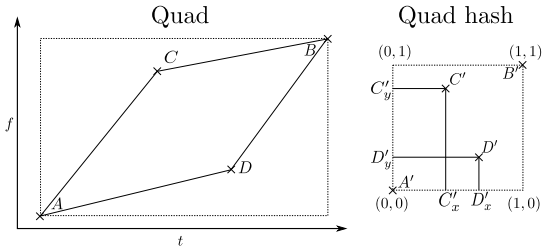


Fig. 2. Example of a valid quad A, B, C, D and its corresponding hash.

in the evaluation results of our previous work [3]. We now alleviate this effect to a great extent by choosing lower values for the parameters c, r. This has the effect that query quads of shorter time-span 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., pitch or speed, but not tempo) can cause relevant peaks to leave the observed bandwidth of the audio signal.

C. 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. That representation should be translation- and scale-invariant, and quickly retrievable. To achieve this, we adapt the hashing model shown in [4]. Instead of 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 spectrogram 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 2. 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 2 shows the hash space for general quads, but the distribution of C'_x, D'_x hash values might be 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 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:

$$C'^{ref}_{x\ min} = D'^{ref}_{x\ min} = \frac{c^{ref} - r^{ref}/2}{c^{ref} + r^{ref}/2} \tag{5}$$

Therefore, all query quads where $C'^{query}_x < C'^{ref}_{x\ min} - \epsilon_L$ 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 $C'_x \leq D'_x$.

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 moved closer to the rootpoint, while at the same time another one moved 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. Naturally, we exclude such irrelevant quads from the set of query quads before we start picking strong quads based on peak magnitude values (see Subsection IV-A). To convey the effectiveness of enforcing that constraint, we measured the amount of possible quads and the size of their relevant subsets: averaged over 11,700 queries (see Figure 5 below), 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.

V. 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 S, where S denotes the height and width of the quad in spectrogram space (i.e. $S_x = B_x - A_x$ and $S_y = B_y - A_y$), 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, S 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 fixed-radius near neighbour searches of quad hashes. We use a variant of a shallow bounding-volume hierarchy [15] 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 [16]. 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 chose to use this tree variant because we already had the implementation along with python bindings, and it is faster than the RTree we used in [3]. We would like to point the reader to the highly optimized kD-tree implementation of [4], 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, and a filter height of 75 frequency bins. 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 VIII 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 peakfile 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 IV-C) and could be scaled and stored as unsigned short16 values. Likewise, the width and height of quads (S as above) could be represented in this format.

VI. Identification algorithm

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 score.

The method of answering a query consists of three stages: 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 VI-A.

Results are then passed to a match sequence estimation stage, in which we efficiently search for sequences within the set of matched candidates. For this we adapt and extend the histogram binning approach that is proposed in [5], combined with 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 VI-B.

Finally, for each match candidate within the filtered sequences we apply a verification step, adapted from the findings in [4]. 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 VI-C.

A. Match candidate selection and filtering

For each query quad hash a fixed-radius near neighbour 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 spectral quads 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 VI-C), 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 in [17]. The filter routine accepts candidates if the following holds:

$$A_y^{query}/A_y^{cand} \geq 1/(1+\epsilon_p) \tag{6a}$$
$$A_y^{query}/A_y^{cand} \leq 1/(1-\epsilon_p) \tag{6b}$$

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 IV-B). This is achieved by looking at the different orientation of the query quads and their candidates and computing the scaling factors for time and frequency as follows:

$$s_{time} = (B_x^{query} - A_x^{query})/(B_x^{ref} - A_x^{ref}) \tag{7a}$$
$$s_{freq} = (B_y^{query} - A_y^{query})/(B_y^{ref} - A_y^{ref}) \tag{7b}$$

Quad candidates are accepted if the following holds:

$$s_{pitch} \geq 1/(1+\epsilon_p) \wedge s_{pitch} \leq 1/(1-\epsilon_p) \tag{8a}$$
$$s_{time} \geq 1/(1+\epsilon_t) \wedge s_{time} \leq 1/(1-\epsilon_t) \tag{8b}$$

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 VI-C).

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 query quads, we can perform a fine pitch coherence filter as follows:

$$|A_p^{query} - A_p^{ref} s_{freq}| \leq \epsilon_{pfine} \tag{9}$$

where ϵ_{pfine} is the fine pitch coherence threshold. Ideally, we would expect $A_p^{query} = A_p^{ref} s_{freq}$, but we have to account for the fact that the location of spectral peaks is only robust, but not invariant to signal distortions. We let $\epsilon_{pfine} = 1.8$, which we determined empirically.

We now sort the remaining accepted candidates by file-ID, and enter the sequence estimation step.

B. 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 used in the Shazam algorithm [5]. We adapt the method such that the query time (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.

C. 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 [4]. 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 7a, 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, below.

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[5].

[5]In our initial work [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 paper. 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%.

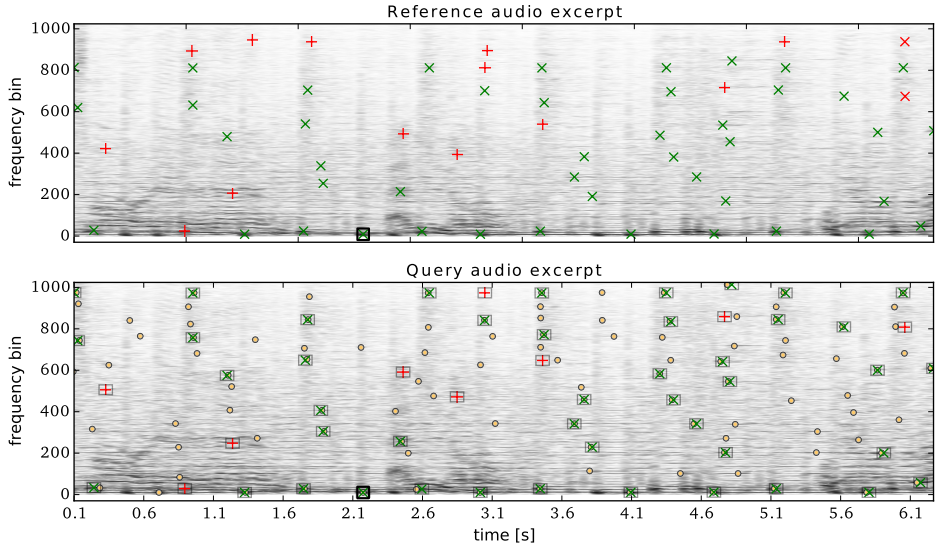


Fig. 3. Verification of a match candidate for a query snippet of 120% pitch (i.e. the query audio has higher pitch than the unmodified reference audio). The two black squares are the root points A^{ref}, A^{query}. The plus symbols and crosses in the upper figure show all reference peaks in proximity 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. 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.

The transformation to align a reference peak P^{ref} with a query peak P^{query} in query space uses the scale transformation estimate s (cf. Equations 7a, 7b):

$$P^{query} = A^{query} + \mathit{off} \tag{10}$$

$$\mathit{off} = (P^{ref} - A^{ref}) \cdot s \tag{11}$$

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 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 3: here, the query audio was modified in pitch, 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 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 percentage v/N (with $v \leq N$) of correctly aligned spectral peaks in the set of nearby peaks. 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 also be applied in the sequence estimation stage, for early rejections.

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 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. In most of the cases the method identifies the correct file-ID, transformation and reference position for a query after

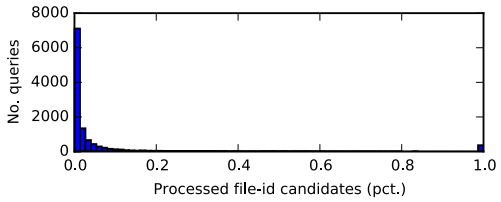


Fig. 4. Histogram of the percentage of candidate file-IDs from the retrieved set of potential file-IDs that had to be processed in order to answer a query, computed from 11,700 queries. Note the bin at $x = 1.0$: here, the contributors are false negatives, and weakly matched file-IDs with exactly t_s matches (either true or false positives).

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.

VII. Comparison to our previous work

In this section we highlight the most important novelty aspects of the proposed method compared to our previous work on that topic [3].

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 of peaks and incorporates post processing to guarantee one peak per window.

Query quad grouping windows are automatically adapted to the given tolerance parameters, a relevant subspace constraint is enforced and the quad grouping process creates quads in an almost uniform distribution over time. Finally, we pick strong quads based on their respective peak magnitudes.

For lookup, we use a different spatial datastructure that has higher performance than libspatialindex's RTree [18]. The quad records are refined to discard the inner points 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. The file ID of quads is incorporated into the quad records to reduce the level of indirection.

Match sequence detection is now performed prior to match verification, which considerably decreases computational costs. It uses outlier detection on candidate sequence members to further reduce the computational load of the subsequent verification stage. We refined the order of the filter chain 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.

While the general concept of the method is the same, the whole implementation and parameterization changed and different data structures are used. Altogether, the novel aspects of the proposed work and the interdependent parameter changes increase the identification performance while considerably decreasing the runtimes.

VIII. Experiments and Results

The basis for all the experiments in this paper is the reference database consisting of 100,011 as described in Section V. The dataset[6] is freely available and we publish information that allows to recreate the reference collections, along with all queries that are performed in the experiments.

To create test queries, we randomly choose and fix a set of 300 reference songs and subject these to different speed, tempo, pitch and noise level modifications. We then randomly select a starting 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 $20s$. These query snippets are used for all experiments.

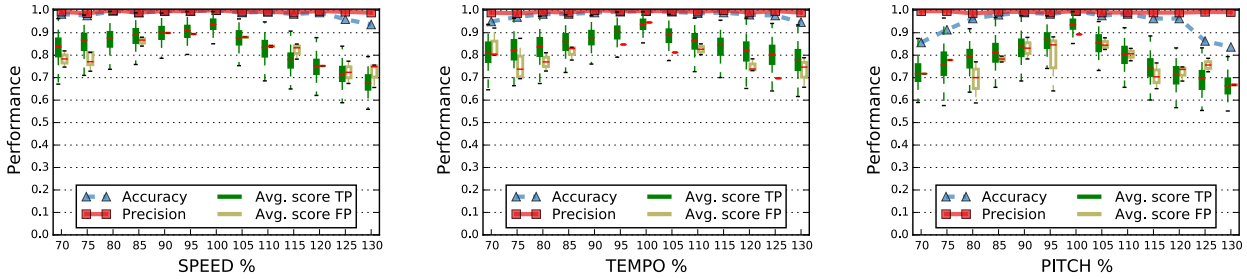
We create the snippets from .mp3 encoded data, and encode the distorted versions in the Ogg Vorbis format [20], using the default compression rate (cr $= 3$). We do this to demonstrate that the system is robust to effects that result from a lossy audio compression. All modifications are realized with the free SoX audio toolkit [21].

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 near-neighbour range 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.

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. *tn* (*true negatives*) is the number of cases in which the system correctly abstains from identifying a reference because there *is* no correct reference. *fp* (*false positives*) is the number of cases in which the system predicts the wrong reference. *fn* (*false negatives*) is the number of cases in which the system fails to return a reference id at all.

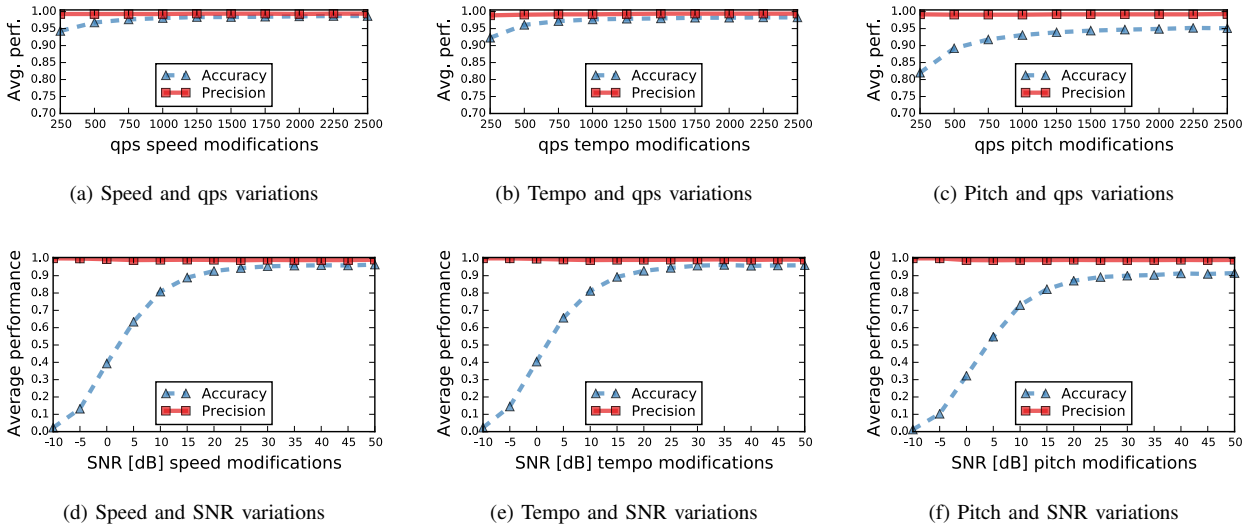
We present a rich set of experiments for various query distortions to evaluate the performance of the system in cases where the correct reference is present in the database, and

[6]The dataset we use consists of more than 100,000 full-length, freely available creative commons licensed music pieces, hosted by the jamendo service [19]



(a) Speed variations (b) Tempo variations (c) Pitch variations

Fig. 5. Precision and accuracy for speed (5a), tempo (5b) and pitch (5c) modifications of $20s$ 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

(d) Speed and SNR variations (e) Tempo and SNR variations (f) Pitch and SNR variations

Fig. 6. Precision and accuracy for speed, tempo and pitch distortions (from left to right) on our reference database of 100,000 songs. Figures 6a to 6c (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 $20s$ 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 5. Figures 6d to 6f show results for a total of 152,100 queries of various SNR, snippet length of $15s$ and $\epsilon_L = 0.01$, where each pair of data points is the average over 3900 queries.

show the capability of the system to report true negatives in a separate experiment. We do this because it is not meaningful to systematically modify queries that are not present in the reference database.

For the set of experiments, where there are no true negatives, we define two performance measures: *Recognition Accuracy* is the proportion of queries whose reference is correctly identified:

$$\text{Accuracy} = \frac{tp}{tp + fp + fn} = \frac{tp}{N} \tag{12}$$

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

$$\text{Precision} = \frac{tp}{tp + fp} \tag{13}$$

Thus, high precision means 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.

To test the system with queries that are not present in the reference database, we define a third performance measure:

$$\text{Specificity} = \frac{tn}{tn + fp} \tag{14}$$

High specificity quantifies the capability of the system to avoid false positives by correctly abstaining from reporting a match.

We now explain the specificity tests and, for convenience, report the results in the following paragraph. The runtimes for the experiments are documented in the bottom two rows of Table I.

To test the specificity of the method, we prepare a second set of $20,000$ query files from the jamendo service [19].

We tried to ensure that the files contained in this set are disjunct with the set of referenced files, by inspecting the available metadata. 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 metadata. We reject music pieces that appear to be duplicates according to the metadata, 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 metadata of some of the high scoring matched files are very similar, but not equal to metadata 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 metadata 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 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 queries of audio that is indexed in the reference database, and show individual results for speed, tempo and pitch distortions. 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 can not be sure, so we regard these cases as false positives.

A. 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 5, and are accompanied by 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 VI-C) 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 I. Because of space constraints we show the *average* performance of the individual types of distortions. Each row of Table I 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 runtimes 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, e.g., 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 treat those cases strictly as false positives.

TABLE I
AVERAGE PERFORMANCE AND RUN TIMES ($qps = 1500$)

l [s]	Speed prec.	Speed acc.	Tempo prec.	Tempo acc.	Pitch prec.	Pitch acc.	mean t [s]	median 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 [19]	18.229	.9617	1.96	1.77
20.0	non-free	50.000	.9726	1.89	1.67

TABLE II
PERFORMANCE ON QUERIES WITHOUT SCALE MODIFICATIONS, THAT ARE DISTORTED BY VARIOUS EFFECTS ($qps = 1500$)

effect type	prec.	acc.
bandpass	.997	.993
chorus	.986	.687
echo	.993	.980
flanger	.984	.827
gsm	.978	.440
tremolo	.990	.977

B. 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 6a to 6c. For each value of q an experiment as in Figure 5 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 6. 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 runtime for the qps range from 250 to 2500 in steps of 250 is: $0.74s$, $0.94s$, $1.14s$, $1.34s$, $1.51s$, $1.69s$, $1.86s$, $2.02s$, $2.17s$ 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.

C. Results on effects and noisy queries

The impact of various effects on the performance of the presented method is evaluated and summarized in Table II. For comparability we chose the same effects with identical

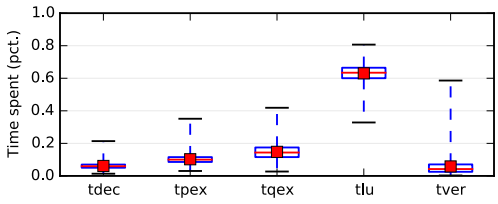


Fig. 7. Fraction of time spent in query processing stages, averaged over the 11,700 queries performed for Figure 5 ($l = 20s$, $\epsilon_L = 0.01$, $qps = 1500$). This figure represents a median query processing time of $1.65s$.

parameterization as in [7]. The most challenging distortions are the gsm codec and the chorus effect, and to a certain extent the flanger effect. For the effects bandpass, 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 given in the second row of Figure 6. 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 $+15dB$, which is a lower SNR value than what we expect to encounter in application scenarios.

D. 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 5, 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 7 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: 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 5, 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 I. We believe the run times are within practical limits, considering the size of the reference database and the large tolerances of scale modifications.

E. Comparison with reference methods

While it is not possible to directly compare our results to those of [6] (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 tempo and pitch scale modifications, our method seems slightly more robust for the ranges we evaluated on, and it has a noticeably higher performance for queries that were severely slowed down in speed. Regarding the efficiency of [6], we do not know runtimes 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 chances 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.

The work presented in [7] allows for comparability by publishing 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 [7] 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 (see Table I). We assume that the limited robustness of [7] 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 aliasing 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. Aliasing effects of quantized hashes are expected even for 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 for strong robustness limitations of quantized hashes for key/value lookup methods.

IX. Discussion

The scale transform tolerances $\pm 31\%$ used in this paper do not reflect the upper bound of what the 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, less 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.

Instead of utilizing quantized hashes, we perform fixed-radius all near neighbour queries in the continuous four-dimensional hash space to retrieve raw candidates. Thus, aliasing effects of quantized hashes, that 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 in [7], hash keys for adjacent bins must be computed for additional queries, which in turn limits the efficiency of such methods. The system described in [4], and our proposed system, allows for intuitive parameterization of the query 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 pitch scale robustness of our method, compared to speed or tempo. 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 with severe signal scale modifications, and not specific to our method.

In [4], using n-tuples instead of quadruples of points is discussed. As proposed in [4], we use quad based hashes and observe that the computational cost and identification performance is 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.

X. Conclusions

We have presented a practical audio identification method that is highly robust to noise, tempo, speed and pitch distortions. Due to the compact fingerprints, subsequent search in hash space can be efficiently performed via an exact fixed-radius near neighbour scheme. We demonstrated high identification performance with low processing run times in experiments of a total of roughly 450,000 queries with various distortions, against a large database consisting of 100,000 full length songs ($\approx 6899h$ of music). The proposed reference data structures take less than 10 GB of disk space. While there is high potential of false positive matches in a database of this size (roughly 209 million quads) in combination with the rather large scale tolerances of $\pm 30\%$, 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.

XI. Acknowledgment

We would like to thank Andreas Arzt, Harald Frostel, Rainer Kelz, Filip Korzeniowski and Andreu Vall for many intense and fruitful discussions, and the anonymous reviewers for their thorough and detailed reviews and suggestions. The research is supported by the Austrian Science Fund FWF under projects TRP307 and Z159.

References

[1] A. F. Smeaton, P. Over, and W. Kraaij, "Evaluation campaigns and trecvid," in *Proc. Intl. Workshop on Multimedia Information Retrieval*. New York, NY, USA: ACM Press, 2006, pp. 321–330.
[2] C. Ouali, P. Dumouchel, and V. Gupta, "A robust audio fingerprinting method for content-based copy detection," in *Intl. Workshop on Content-Based Multimedia Indexing*, Klagenfurt, Austria, 2014, pp. 1–6.
[3] R. Sonnleitner and G. Widmer, "Quad-based audio fingerprinting robust to time and frequency scaling," in *Proc. Intl. Conf. on Digital Audio Effects*, Erlangen, Germany, 2014, pp. 173–180.
[4] D. Lang, D. Hogg, K. Mierle, M. Blanton, and S. Roweis, "Astrometry.net: Blind astrometric calibration of arbitrary astronomical images," *The Astronomical Journal*, vol. 137, pp. 1782–2800, 2010.
[5] A. Wang, "An industrial-strength audio search algorithm," in *Proc. Intl. Conf. on Music Information Retrieval*, Baltimore, Maryland, USA, 2003.
[6] X. Zhang, B. Zhu, L. Li, W. Li, X. Li, W. Wang, P. Lu, and W. Zhang, "Sift-based local spectrogram image descriptor: a novel feature for robust music identification," *EURASIP Journal on Audio, Speech, and Music Processing*, vol. 2015, no. 1, pp. 1–15, 2015.
[7] J. Six and M. Leman, "Panako: a scalable acoustic fingerprinting system handling time-scale and pitch modification," in *Proc. Intl. Conf. Society for Music Information Retrieval*, Taipei, Taiwan, 2014.
[8] E. Dupraz and G. Richard, "Robust frequency-based audio fingerprinting." in *Proc. Intl. Conf. Acoustics, Speech and Signal Processing*. IEEE, 2010, pp. 281–284.
[9] M. Ramona and G. Peeters, "Audioprint: An efficient audio fingerprint system based on a novel cost-less synchronization scheme," in *Proc. Intl. Conf. Acoustics, Speech and Signal Processing*. IEEE, 2013, pp. 818–822.
[10] S. Baluja and M. Covell, "Waveprint: Efficient wavelet-based audio fingerprinting," *Pattern Recognition*, vol. 41, no. 11, pp. 3467–3480, 2008.
[11] D. Lowe, "Distinctive image features from scale-invariant keypoints," *Int. J. Comput. Vision*, vol. 60, no. 2, pp. 91–110, Nov. 2004.
[12] P. Indyk and R. Motwani, "Approximate nearest neighbors: Towards removing the curse of dimensionality," in *Proc. of Symposium on Theory of Computing*, ser. STOC '98. New York, NY, USA: ACM, 1998, pp. 604–613.
[13] A. Arzt, S. Böck, and G. Widmer, "Fast identification of piece and score position via symbolic fingerprinting," in *Proc. Intl. Conf. Society for Music Information Retrieval*, Porto, Portugal, October 8-12 2012, pp. 433–438.
[14] M. Malekesmaeili and R. Ward, "A local fingerprinting approach for audio copy detection," *Signal Processing*, vol. 98, pp. 308–321, 2014.
[15] H. Dammertz, J. Hanika, and A. Keller, "Shallow bounding volume hierarchies for fast simd ray tracing of incoherent rays," in *Proc. Eurographics Conf. on Rendering*. Aire-la-Ville, Switzerland, Switzerland: Eurographics Association, 2008, pp. 1225–1233.
[16] I. Wald, "On fast construction of sah-based bounding volume hierarchies," in *Proc. Symposium on Interactive Ray Tracing*. Washington, DC, USA: IEEE Computer Society, 2007, pp. 33–40.
[17] G. Evangelidis and C. Bauckhage, "Efficient subframe video alignment using short descriptors." *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 35, no. 10, pp. 2371–2386, 2013.
[18] "libspatialindex," available at https://libspatialindex.github.io/.
[19] "Jamendo," available at https://www.jamendo.com/.
[20] "Ogg Vorbis," available at http://www.vorbis.com.
[21] "SoX - Sound eXchange," available at http://sox.sourceforge.net/.

Reinhard Sonnleitner holds a M.Sc. degree in computer science from the Johannes Kepler University, Linz, Austria, where he is a Ph.D. candidate at the Department of Computational Perception. His research interests include machine learning and audio signal processing.

Gerhard Widmer (www.cp.jku.at/people/widmer) is Professor and Head of the Department of Computational Perception at Johannes Kepler University, Linz, Austria, and Head of the Intelligent Music Processing and Machine Learning Group at the Austrian Research Institute for Artificial Intelligence, Vienna, Austria. His research interests include AI, machine learning, and intelligent music processing, and his work is published in a wide range of scientific fields, from AI and machine learning to audio, multimedia, musicology, and music psychology. He is a Fellow of the European Coordinating Committee on Artificial Intelligence (ECCAI), has been awarded Austrias highest research awards, the START Prize (1998) the Wittgenstein Award 2009), and currently holds an ERC Advanced Grant for research on computational models of expressivity of music.