Fast Intra-Collection Audio Matching

Verena Thomas, Sebastian Ewert*, Michael Clausen
Computer Science III
University of Bonn
{thomas,ewerts,clausen}@cs.uni-bonn.de

ABSTRACT

The general goal of audio matching is to identify all audio extracts of a music collection that are similar to a given query snippet. Over the last years, several approaches to this task have been presented. However, due to the complexity of audio matching the proposed approaches usually either yield excellent matches but have a poor runtime or provide quick responses albeit calculate less satisfying retrieval results. In this paper, we present a novel procedure that combines the positive aspects and efficiently computes good retrieval results. Our idea is to exploit the fact that in some practical applications queries are not arbitrary audio snippets but are rather given as extracts from the music collection itself (*intra-collection query*). This allows us to split the audio collection into equal sized overlapping segments and to precompute their retrieval results using dynamic time warping (DTW). Storing these matches in appropriate index structures enables us to efficiently recombine them at runtime. Our experiments indicate a significant speedup compared to classical DTW-based audio retrieval while achieving nearly the same retrieval quality.

Categories and Subject Descriptors

H.2.4 [**Information Systems**]: Systems—*Multimedia databases*; H.5.5 [**Information Interfaces and Presentation**]: Sound and Music Computing

Keywords

Audio matching, dynamic time warping, music retrieval

1. INTRODUCTION

Digital audio formats in combination with today's storage capabilities enabled the development of digital audio collections on a grand scale. Thereby, content-based retrieval techniques are becoming increasingly important. For audio files content-based retrieval can be categorized into four types. In *audio identification*, for a given audio snippet, the exact audio recording it originates from has to be determined whereas in *version identification* or *cover song retrieval* all different recordings of the queried piece of music (e.g., played by different musicians or using a different instrumentation) are to be retrieved. In *audio matching*, rather than whole recordings, audio extracts that are similar to a query snippet are reported as retrieval results. The last retrieval type is *category-based music retrieval*, where the recordings in the music collection are categorized or clustered w.r.t. some predefined cultural or musicological category, e.g., genre or mood. Given a recording, category-based retrieval can then propose similar songs to the user. For details on the different approaches and an extensive bibliography we refer to [6].

In this paper, we focus on the audio matching scenario. Here, queries are formed (at least implicitly, see Section 2) by audio examples of arbitrary length and the retrieval results should be arbitrary audio extracts from the collection that present some similarity with the query. In particular, one recording might even contain several hits. Therefore, to find all occurrences of a given query, a comparison of the query with all feasible contiguous subsegments of the collection is necessary. Two prominent approaches for solving this task are *diagonal matching* (or *linear scan*) and (subsequence) *dynamic time warping* (DTW), see [9] for details on both approaches. In diagonal matching, a sequential warping-free comparison of the query with subsegments of the collection (having the same length) is performed. By using appropriate feature representations, local variations (e.g., in timbre, harmony, instrumentation) can be canceled out. However, the approach is constraint to comparing sequences of equal length and therefore does not consider tempo variations. To allow for global tempo variations of ±40%, Kurth and Müller propose the retrieval of several time stretched versions of the query [8]. In contrast, subsequence DTW enables the comparison of feature sequences with different length by performing non-linear warping (see Section 3.1). Thereby, DTW allows for both global and local tempo variations. In addition, a higher robustness towards local variations (e.g., insertions and deletions) is achieved.

Unfortunately, the good retrieval quality comes at the cost of DTW being rather slow.[1] One approach for speeding up the calculation is the imposition of global constraints on the admissible warping paths [9]. However, in the context of subsequence DTW-based audio matching this approach is not applicable. Besides, the runtime of classical diagonal matching becomes problematic for large datasets as well.

*Is now with Queen Mary, University of London, UK.

MIRUM'12, November 2, 2012, Nara, Japan.

[1]Given two sequences of length N and M, the computational complexity of DTW is $O(NM)$.

For both approaches, the pre-calculation of a retrieval index has proven to be a reasonable approach for handling large collections. Kurth and Müller propose an indexing approach for diagonal matching that significantly speeds up the retrieval process while causing only minor quality losses [8]. The general idea is to use a set of reference feature vectors (codebook vectors) and to store their occurrence positions in the database using inverted lists. During retrieval the matches can be determined using shifted versions of the relevant inverted lists in combination with fast list intersections. Another approach widely used in audio identification and version identification is indexing based on locality sensitive hashing (LSH) (see, e.g., [2, 5, 14]). After converting the audio data into some feature representation, the feature sequence of the audio collection is segmented into equal-sized subsequences (*shingles*) which are subsequently stored using LSH. During retrieval, the feature sequence of a query is split into shingles as well. For each query shingle all shingles with the same hash value are retrieved from the index before applying some merging approach to determine valid matches (e.g., Casey et. al. [2] count the number of matching shingles while Yang [14] employs the Hough Transform).

The issue with DTW is that it does not obey the triangular inequality and thus cannot be indexed without loss of quality. Instead, several approaches suggest the introduction of a lower bounding function (LBF) for DTW that enables a fast filtering of match candidates and can be indexed using multidimensional indexing methods (e.g., R-tree), e.g., [4, 7, 12, 15]. After index look-up these candidates are subsequently verified by calculating the DTW-distance. Agrawal et. al. [1] combine the LBF-based indexing approach with the previously mentioned shingling to deal with subsequence matches (i.e., the query is part of one of the documents). However, all of these approaches are constraint to DTW-comparisons of the query (or query shingles) with equal-sized subsequences from the database and thus weaken the benefits of DTW-based matching.

In conclusion, diagonal matching achieves short response times but lacks the flexibility of DTW to search for arbitrarily time-warped (globally and locally) versions of a query. Similarly, the discussed approaches to indexing of DTW can only retrieve matches of the same length as the query and require online verification of candidates via DTW calculation. For large candidate lists this step can potentially result in long response times. In this paper, we propose a new procedure whereby the advantages of index-based retrieval and subsequence DTW are combined. To this end, we utilize the fact that in practical applications queries are often given as audio extracts from the music collection itself (intra-collection query), see Section 2. As we will see, this leads to a simple, yet very efficient and effective retrieval approach that combines the efficiency of indexing techniques with the retrieval quality of classical DTW-based matching. Evidently, indexing the retrieval results for all possible queries—which would be the obvious first idea—is not feasible for larger collections. Instead, we follow the shingling approach and split the dataset into overlapping segments of equal length, calculate the corresponding audio matches, and store them as search index. During query processing the indexes of the segments covering the query are merged to calculate the retrieval result. Depending on the size of the music collection we observed speedup factors between 42 and 311 in comparison to DTW-based audio matching.

The remainder of this paper is organized as follows. In the next section, applications for intra-collection audio matching are discussed. In Section 3, we present the proposed index-based audio matching procedure. Section 4 discusses various experiments on retrieval speed and result quality before we conclude the paper with an outlook on future work.

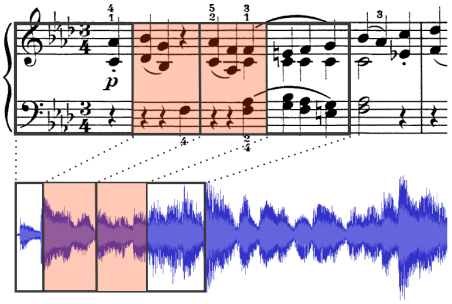

Figure 1: Schematic example for score-audio synchronization. The calculated alignments allow for cross-modal audio retrieval where the score is used for query formulation (red selection).

2. APPLICATIONS

Usually, libraries and museums that provide access to their digital audio collections (typically consisting of thousands of audio tracks) do not allow visitors to connect their USB devices to upload queries. Therefore, the users are constrained to searching within the given collection using extracts of the available audio as queries (i.e., intra-collection search). The method proposed in this article was designed specifically with such library systems in mind and aims at providing fast and accurate retrieval results for these intra-collection query scenarios. This way, our procedure allows users, for example, to quickly find and access repetitions of a music extract in all available recordings of the underlying piece or to search a database for pieces of music that borrow ideas from other pieces.

Intra-collection retrieval is particularly interesting as it allows for creating convenient user interfaces for query formulation. For example, a library system can offer rich audio visualizations (e.g. spectrogram, structure analysis) to assist the user in selecting a query within the audio collection. Furthermore, for a given music recording, digital libraries often provide corresponding scanned score sheets. In these cases, using score-audio synchronization techniques, each position in the score can be linked to a corresponding position in an audio recording [9], see Figure 1. Thus, queries can be formulated using an intuitive score-based interface [3] where the computed linking information is used to automatically translate the queries into the audio domain (*cross-modal audio retrieval*).

To relax the restriction on intra-collection queries, the system could easily be extended by an audio identification step [6, 13]. In such a preprocessing step, a given external audio snippet could be tested for membership with the collection. If this is the case, the according audio snippet from the collection could be used instead (to benefit from intra-collection search). Otherwise, the system could fall back on diagonal matching (accepting inferior results) or classical DTW-based audio matching (accepting long response times). Our experiments in Section 4 in combination with the reported performance of audio identification suggest that this approach would still yield orders of magnitude better response times than classical DTW-based audio matching (for queries that are part of the collection).

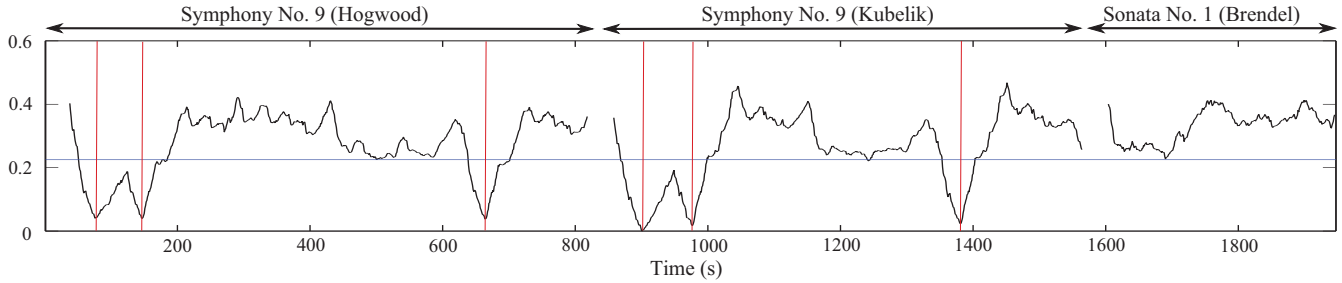

Figure 2: Distance function with respect to a query consisting of the beginning of *Symphony No. 9, Molto Vivace* by L. v. Beethoven in an interpretation conducted by R. Kubelik. For two different interpretations of the *Molto Vivace*, the distinct peaks at the three match locations in each document are visible (vertical red lines). The horizontal blue line indicates the ranking threshold $\theta = 0.225$. As expected, for a different piece (*Piano Sonata No.1* by Beethoven), no matches are reported.

3. INDEXING AND RETRIEVAL METHOD

In this section, we first describe the DTW-based audio matching procedure we employ. Our approach closely resembles the methods described in [8, 9]. Afterwards we explain the index creation and the corresponding retrieval process to handle intra-collection queries.

3.1 Audio Retrieval

Let Q be a query audio clip and let $(D_0, D_1, \ldots, D_N)$ be a collection of audio recordings. To simplify matters, we create a large dataset document D by concatenating all documents $D_0, D_1, \ldots, D_N$, where we keep track of the document boundaries in a supplemental data structure. Then, the goal of audio matching is to find all contiguous subsegments in D that are similar to the given query Q.

For this purpose, Q and D are transformed into suitable feature sequences $\mathbf{Q} = (\mathbf{Q}_0, \mathbf{Q}_1, \ldots, \mathbf{Q}_K) \in \mathcal{F}^{K+1}$ (with $|\mathbf{Q}| = K+1$) and $\mathbf{D} = (\mathbf{D}_0, \mathbf{D}_1, \ldots, \mathbf{D}_L) \in \mathcal{F}^{L+1}$, respectively. Here, $\mathcal{F}$ denotes the underlying feature space. In our implementation, we chose the CRP (chroma DCT-reduced log pitch) features introduced by Müller and Ewert [10] using their implementation provided by the Chroma Toolbox [11]. In our procedure, we employ non-overlapping features with a window size of one second. For CRP-features the feature space $\mathcal{F}$ consists of all elements in $[-1, 1]^{12}$ which have euclidean length 1.[2] We define the cost measure $c : \mathcal{F} \times \mathcal{F} \to [0, 2]$ on $\mathcal{F}$ as $c(x, y) := 1 - \langle x, y \rangle$ (which is the cosine measure for normalized vectors). Then, we define a distance function $\Delta_{\mathbf{Q}}^{\mathbf{D}} : [0\colon L] \to [0, \infty]$ between $\mathbf{Q}$ and $\mathbf{D}$ that locally compares $\mathbf{Q}$ to subsequences of $\mathbf{D}$

$$\Delta_{\mathbf{Q}}^{\mathbf{D}}(\ell) = |\mathbf{Q}|^{-1} \min_{a \in [0\colon \ell]} (\mathrm{DTW}(\mathbf{Q}, \mathbf{D}(a\colon \ell)). \tag{1}$$

Here, $\mathbf{D}(a\colon \ell)$ denotes the subsequence of $\mathbf{D}$ starting at index a and ending at index ℓ and $\mathrm{DTW}(\mathbf{Q}, \mathbf{D}(a\colon \ell))$ denotes the DTW distance between $\mathbf{Q}$ and $\mathbf{D}(a\colon \ell)$ with respect to the cost measure c. For details on the efficient computation of this distance using dynamic programming, we refer to [9, Section 4.4].

Each entry $\Delta_{\mathbf{Q}}^{\mathbf{D}}(\ell)$ of the distance function measures the distance between $\mathbf{Q}$ and the subsequence $\mathbf{D}(a_\ell\colon \ell)$ of $\mathbf{D}$, where $a_\ell = a_\ell(\mathbf{Q}, \mathbf{D})$ denotes the minimizing index in equation (1), see Figure 2 for an example. As we apply DTW, it is usually true that $|\mathbf{Q}| \neq |\mathbf{D}(a_\ell\colon \ell)|$.

The best match between Q and D is now encoded by the index $\ell_0 \in [0\colon L]$ minimizing $\Delta_{\mathbf{Q}}^{\mathbf{D}}$. The distance value $\Delta_{\mathbf{Q}}^{\mathbf{D}}(\ell_0)$ is also referred to as the *ranking value* of the match corresponding to the feature sequence $\mathbf{D}(a_{\ell_0}\colon \ell_0)$. As the goal is to find all audio extracts that are similar to the query, the calculation then continues by searching for the second best match. But first a neighborhood of ℓ_0 is excluded from further considerations to avoid overlaps between matches. In our implementation, we exclude the region $[a_{\ell_0}\colon \ell_0 + 0.85 \cdot (\ell_0 - a_{\ell_0})]$ by setting the respective $\Delta_{\mathbf{Q}}^{\mathbf{D}}$-values to ∞. Then, to find subsequent matches, the above procedure of identifying the minimum of $\Delta_{\mathbf{Q}}^{\mathbf{D}}$ is performed repeatedly until the minimal distance exceeds a specified distance threshold θ (we use $\theta = 0.225$) or until a certain number of matches is obtained. This way, we iteratively compute all matches of $\mathbf{Q}$ in $\mathbf{D}$

$$H(\mathbf{Q}) := \left\{ (a_\ell, \ell, r) \mid \ell \in [0\colon L], r = \Delta_{\mathbf{Q}}^{\mathbf{D}}(\ell) \leq \theta \right\}.$$

By employing a DTW-based distance function, a high robustness towards global and local tempo variations as well as small local variations is achieved. However, for large datasets, the described sequential scanning approach results in long response times. For applications where the query originates from within the dataset, we therefore propose the calculation of an audio matching index that allows for fast DTW-based audio matching in larger datasets. The general idea is to split the dataset into small overlapping segments, perform the previously described audio matching procedure, and to store the result lists as retrieval index. During retrieval, those lists are used to efficiently determine the matches for a given query.

3.2 Index Creation

Given a segment length $\lambda > 0$ and a step size $\tau \leq \lambda$, the segmentation $S_{\mathbf{D}}$ of the feature sequence $\mathbf{D}$ is defined as

$$S_{\mathbf{D}} = (S_0, S_1, \ldots, S_M)$$

where $M = \lceil \frac{|\mathbf{D}| - \lambda}{\tau} \rceil$ and $S_m = \mathbf{D}(m\tau\colon m\tau + \lambda - 1)$ for $m \in [0\colon M-1]$ and $S_M = \mathbf{D}(M\tau\colon |\mathbf{D}| - 1)$, see Figure 3. If not stated otherwise, we use $\lambda = 20, \tau = 5$.

For each segment S_m, $m \in [0\colon M]$, we perform the audio matching procedure presented in Section 3.1 and store the respective retrieval results as inverted lists

$$L(m) = \{m\} \times H(S_m)$$

containing all tuples (m, a_ℓ, ℓ, r) with $\ell \in [0\colon L]$ and $r = \Delta_{S_m}^{\mathbf{D}}(\ell) \leq \theta$.

The computational complexity of this preprocessing step is $O(\lambda \cdot M \cdot |\mathbf{D}|) = O(|\mathbf{D}|^2)$. Therefore, its calculation becomes rather time consuming for larger collections. However, the index creation can be accelerated significantly by

[2] We distinguish between $[0\colon n] := \{x \in \mathbb{Z} \mid 0 \leq x \leq n\}$ and $[0, n] := \{x \in \mathbb{R} \mid 0 \leq x \leq n\}$.

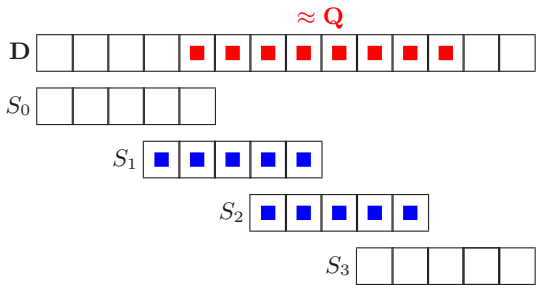

Figure 3: Segmentation of $\mathbf{D}$ ($\lambda = 5$, $\tau = 3$). For the approximate feature representation $\mathbf{D}(4\,{:}\,11)$ of query Q (red) the best fitting segment subsequence is $S_{\mathbf{D}}(1\,{:}\,2)$. In a sense, this best fitting coverage corresponds to minimizing the symmetric difference between the red and the blue squares projected onto the $\mathbf{D}$-strip.

employing distributed processing. Furthermore, a given index does not need to be recalculated if a new audio track D_{N+1} is added. Instead, two steps are required. First, the inverted lists for all segments in D_{N+1} have to be calculated (using $(D_0, D_1, \ldots, D_{N+1})$ as music collection). Second, all segments in $(D_0, D_1, \ldots, D_N)$ need to be queried in D_{N+1} to update their respective inverted lists. In this way, the computational complexity for adding a new document D_{N+1} (usually: $|\mathbf{D}_{N+1}| \ll |\mathbf{D}|$) is $O(|\mathbf{D}_{N+1}| \cdot |\mathbf{D}|)$.

3.3 Retrieval Method

As the given query Q is a subsegment of D, we can approximate its feature representation by a subsequence $\mathbf{D}(i\,{:}\,j)$ of $\mathbf{D}$, with $0 \leq i < j < |\mathbf{D}|$. Then, we determine the subsequence $S_{\mathbf{D}}(i^*\,{:}\,j^*)$ of segments in $S_{\mathbf{D}}$ through which the query feature sequence $\mathbf{D}(i\,{:}\,j)$ is properly represented. We use a best fitting coverage of the query by setting $i^* = \lfloor \frac{i}{\tau} \rceil$ and $j^* = \lfloor \frac{j+1-\lambda}{\tau} \rceil$, see Figure 3.

Subsequently, the inverted lists $L(m), m \in [i^*\,{:}\,j^*]$, of all segments in $S_{\mathbf{D}}(i^*\,{:}\,j^*)$ are loaded. Using these, we will now calculate the index-based (approximate) set of matches $\tilde{H}(\mathbf{Q})$.[3] To this end, we employ a similar approach as in [8]. A fundamental observation is that for a subsegment $\mathbf{D}(u\,{:}\,v)$ to be a valid retrieval result, a sufficient number of inverted lists $L(m), m \in [i^*\,{:}\,j^*]$ need to contain a match whose start and end points lie within $[u\,{:}\,v]$. To determine such subsegments $\mathbf{D}(u\,{:}\,v)$ efficiently, we exploit the fact that the regions of elements in the individual inverted lists belonging to the same match can be made overlapping by shifting them appropriately.[4] We therefore define the *p-shifted* version of a list $L(m)$, $m \in [0\,{:}\,M]$, as

$$L(m) - p := \{(m, a_\ell - p, \ell - p, r) \mid (m, a_\ell, \ell, r) \in L(m)\}$$

and create the list of all shifted segment retrieval results

$$B(\mathbf{Q}) = \bigcup_{m \in [i^*\,:\,j^*]} L(m) - m\tau.$$

Using $B(\mathbf{Q})$ we will now calculate $\tilde{H}(\mathbf{Q})$ in two steps. First, we determine all regions $\mathbf{D}(u\,{:}\,v)$ that contain sufficiently overlapping elements of $B(\mathbf{Q})$ (Algorithm 1). Second, the ranking value of these regions is calculated and their eligibility as retrieval results is tested (Algorithm 2).

[3] For $i^* = j^*$, $\tilde{H}(\mathbf{Q}) = L(i^*)$.

[4] As we use DTW, the regions are of varying length and therefore do not become identical after list shifting (in contrast to [8]).

Algorithm 1 merge regions

1: $\mathcal{I}(\mathbf{Q}) \leftarrow \emptyset$
2: $k \leftarrow 0$
3: **while** $k < |\mathcal{B}(\mathbf{Q})|$ **do**
4: search maximum $t \in [0\,{:}\,|\mathcal{B}(\mathbf{Q})| - 1 - k]$ with
(1) $|\boldsymbol{a}_{k+t} - \boldsymbol{a}_k| < \lambda$
(2) $\forall p < t : |\boldsymbol{a}_{k+p+1} - \boldsymbol{a}_{k+p}| \leq \lambda/4$
5: **if** $t > 0$ **then**
6: $\mathcal{I}(\mathbf{Q}) \leftarrow \mathcal{I}(\mathbf{Q}) \cup \{(k, k+t)\}$
7: **end if**
8: $k \leftarrow k + |\mathcal{S}_{k,k+t}|$
9: **end while**

Algorithm 2 verify candidates

1: $\tilde{H}(\mathbf{Q}) \leftarrow \emptyset$
2: **for all** $(u, v) \in \mathcal{I}(\mathbf{Q})$ **do**
3: $R_{u,v} := \left(\sum_{j \in \mathcal{S}_{u,v}} r_j\right) \cdot (j^* - i^*) \cdot |\mathcal{S}_{u,v}|^{-2}$
4: **if** $\left[R_{u,v} \leq \theta \text{ and } |\mathcal{S}_{u,v}| \geq \frac{1}{2}(j^* - i^*)\right]$
or $\left[R_{u,v} \cdot (j^* - i^*)^{-1} \cdot |\mathcal{S}_{u,v}| < 0.1 \cdot \theta\right]$ **then**
5: $\tilde{H}(\mathbf{Q}) \leftarrow \tilde{H}(\mathbf{Q}) \cup \left\{\left(\min_{t \in [u\,:\,v]}(a_{\ell_t}), \max_{t \in [u\,:\,v]}(\ell_t), R_{u,v}\right)\right\}$
6: **end if**
7: **end for**

For a match $(m, a_\ell, \ell, r) \in L(m)$, let $\overline{a}_\ell := a_\ell - m\tau$ and $\overline{\ell} := \ell - m\tau$ represent the $m\tau$-shifted versions of the start and end positions of the match, a_ℓ and ℓ, respectively. By sorting $B(\mathbf{Q})$ by the shifted start indexes of the matches, we now derive the sequence $\mathcal{B}(\mathbf{Q}) = ((s_k, \boldsymbol{a}_k, \boldsymbol{\ell}_k, r_k))_{k \in [0\,:\,|B(\mathbf{Q})|-1]}$ with $\boldsymbol{a}_k := \overline{a}_{\ell_k}, \boldsymbol{\ell}_k := \overline{\ell}_k$ and $\boldsymbol{a}_0 \leq \boldsymbol{a}_1 \leq \ldots \leq \boldsymbol{a}_{|B(\mathbf{Q})|-1}$. Further, s_k denotes the segment index the according match originates from and $r_k = \Delta^{\mathbf{D}}_{S_{s_k}}(\ell_k)$ is its ranking value. Then, we define $\mathcal{S}_{u,v} := \{s_u, s_{u+1}, \ldots, s_v\}, u \leq v \in [0\,{:}\,|\mathcal{B}(\mathbf{Q})| - 1]$, as the duplicate-free set of segment indexes between u and v. In Algorithm 1, we now step through $\mathcal{B}(\mathbf{Q})$ and combine successive entries to form the set $\mathcal{I}(\mathbf{Q})$ of merged retrieval candidate regions. Here, only those segments that overlap sufficiently with the other segments of a candidate region are added to this region (see, line 4 in Algorithm 1 for the exact overlap-conditions).

The ranking value of a region in $\mathcal{I}(\mathbf{Q})$ is defined as the average mean of the ranking values from the partial matches in the segments $S(i^*\,{:}\,j^*)$ (only one ranking value per segment is used). In addition, we apply a penalty factor $(j^* - i^*) \cdot |\mathcal{S}_{u,v}|^{-1}$ whereby the ranking value of matches with few contributing segments is degraded, see line 3 in Algorithm 2.

In line 4 of Algorithm 2, the eligibility of a match as a retrieval result for Q is tested. To this end, we apply two conditions. First, matches formed by at least half the segments present in the query with a ranking value $R_{u,v} \leq \theta$ are valid matches. However, by means of the second condition, we also allow partial matches with $|\mathcal{S}_{u,v}| < \frac{1}{2}(j^* - i^*)$ that have a very good ranking.

4. EXPERIMENTS

In this section, we report on a series of experiments to indicate how the proposed index-based audio matching approach for intra-collection retrieval performs in comparison to the DTW-based approach. First, we show that through

query	C_1			C_2		
length (s)	DTW	$\text{DTW}^{\leq\theta}_{\text{index}}$	$\text{DTW}^{1,000}_{\text{index}}$	DTW	$\text{DTW}^{\leq\theta}_{\text{index}}$	$\text{DTW}^{1,000}_{\text{index}}$
25	3.15	0.13	0.08	19.51	0.28	0.06
45	4.45	0.15	0.08	24.43	0.30	0.08
65	5.79	0.20	0.11	30.35	0.45	0.11
85	7.23	0.23	0.12	34.84	0.51	0.13
105	8.85	0.27	0.15	39.81	0.61	0.16
125	10.63	0.29	0.16	45.02	0.66	0.18

Table 1: Comparison of the response times (in seconds) for DTW, $\text{DTW}^{\leq\theta}_{\text{index}}$ and $\text{DTW}^{1,000}_{\text{index}}$.

indexing the response times decrease considerably. Afterwards, we examine the quality of the reported matches.

4.1 Datasets

For the presented evaluations, we prepared two collections featuring audio recordings of Western classical music. The first dataset C_1 comprises 444 tracks containing four different interpretations of all piano sonatas by L. v. Beethoven resulting in 44.7 hours of audio. While the first collection is constrained to piano music, the second set C_2 ($C_2 \supset C_1$) additionally contains orchestral music and several songs for voice and piano (e.g., *Winterreise* by F. Schubert). In particular, C_2 contains six recordings of the *Symphony No. 9* by L. v. Beethoven, two of which are piano versions based on the piano transcription by F. Liszt. Overall, the second collection is significantly larger as it comprises $2,012$ audio tracks yielding a total of 141 hours of music.

4.2 Query Length and Response Time

In this experiment, we compare the performance of the classical DTW-based audio matching procedure described in Section 3.1 to the performance of our index-based matching approach ($\text{DTW}_{\text{index}}$) by measuring the average response times. All of the proposed algorithms were implemented in the MATLAB (7.11.0) environment and tests were run on an Intel Core 2 Quad Core (2.83 GHz) PC with 4 GB of main memory under MS Windows 7 (64-bit). Besides comparing the two approaches, we focused on three aspects. First, the impact of the query length on the runtime by using audio snippets with durations of $25-125$ seconds as queries. Second, the impact of the size of the data set on the response time by performing searches for all queries in C_1 as well as C_2. Third, we compare the runtime of the index-based method for indexes consisting of all retrieval results with a ranking value $\leq \theta$ ($\text{DTW}^{\leq\theta}_{\text{index}}$) to indexes containing at most the best $1,000$ matches of each segment ($\text{DTW}^{1,000}_{\text{index}}$). For each setup, we performed 24 runs. The measured average runtimes are depicted in Table 1.

For $\text{DTW}^{\leq\theta}_{\text{index}}$ the speedup factors range from 25 (25s query in C_1) to 80 (45s query in C_2) whereas for $\text{DTW}^{1,000}_{\text{index}}$ the speedup factors increase even further and go from 42 to 311. In addition, comparing the runtimes for C_1 and C_2, the scalability of the index-based procedure w.r.t. the size of the audio collection becomes apparent. While for DTW the response times for queries in C_2 on average increase by a factor of 5 (compared to C_1), with $\text{DTW}^{1,000}_{\text{index}}$ they remain nearly constant.

Furthermore, the evaluations show that with increasing query length both approaches decrease in their performance. However, for the 125s queries $\text{DTW}^{1,000}_{\text{index}}$ still achieves response times below 0.2s. Finally, with respect to the runtime a clear advantage of applying a top-$1,000$ strategy—especially for larger datasets—over the threshold-based indexing becomes apparent (up to 4.5 times faster responses).

C_1	$\lambda = 10$			$\lambda = 20$		
	EM	T 20	T 30	EM	T 20	T 30
Q_1	1.00	0.45	0.53	1.00	0.80	0.70
Q_2	1.00	0.60	0.53	1.00	0.75	0.67
Q_3	1.00	0.55	0.40	1.00	0.65	0.60
Q_4	1.00	0.40	0.33	1.00	0.65	0.60
Q_5	1.00	0.65	0.50	1.00	0.70	0.53
Q_6	1.00	0.55	0.50	1.00	0.80	0.77
Q_7	0.88	0.60	0.43	1.00	0.85	0.80
Q_8	0.75	0.30	0.23	1.00	0.80	0.67
∅	**0.95**	**0.51**	**0.43**	**1.00**	**0.75**	**0.67**

Table 2: Recall of $\text{DTW}^{\leq\theta}_{\text{index}}$ for different segment length λ in relation to the matches of DTW-based audio matching. The results of $\text{DTW}^{1,000}_{\text{index}}$ coincide with the depicted values.

In a further set of experiments we evaluate the competitiveness (in terms of runtime) of our approach with DTW indexing methods applying a lower bounding function (e.g., [7, 15]). For this purpose, we calculated the average runtime of the required candidate verification step, which yields a lower bound on the total runtime. Using queries of $25-125$s length the verification of 20 candidates required $0.15-0.24$s. For 100 candidates we observed response times between 0.66 to 1.22s. These results suggest that our method is a competitive alternative for intra-collection audio matching scenarios.

All in all, the presented evaluations show a significant efficiency boost by applying the proposed index-based audio matching method. However, to access its practical usability one should also examine the quality of the created matches.

4.3 Matching Quality

We present a variety of experiments on the performance of $\text{DTW}_{\text{index}}$ in terms of the matching quality. As the proposed method is intended as a fast index-based approximation of DTW-based audio matching, we compare the generated matches to those calculated by DTW-based matching.

In the first set of experiments, we evaluate the impact of the segment length λ used during index creation (see Section 3.2) on the quality of the retrieval results. On the one hand, the selected segment length obviously directly influences the minimal query length processable by $\text{DTW}_{\text{index}}$. Therefore, too large values will render the approach useless for real-life applications. On the other hand, too short queries usually result in piles of insignificant matches. We chose to compare the performance of $\text{DTW}_{\text{index}}$ for $\lambda = 10$ and $\lambda = 20$. In our experiment, we use C_1 as data collection and take the first 21 measures of the *Piano Sonata No.* 1, *Op.* 2, 1 by L. v. Beethoven as query. This extract is played twice during each of the four performances in C_1 (due to a repetition). We use both the first and the second repetition in each performance as query, thereby receiving a total of eight queries ($Q_1 - Q_8$) with durations between 21 and 25 seconds. Obviously, for each query the collection contains eight exact matches. The classical DTW approach ranks them as the top eight matches (no matter which query $Q_1 - Q_8$ is used). The column labeled "EM" in Table 2 presents the recall values for those exact matches using $\text{DTW}_{\text{index}}$ (i.e., ratio of exact matches occurring in the top eight matches). While for $\lambda = 10$ some queries do not result in a perfect recall, no qualitative difference to the classical DTW approach is observable for $\lambda = 20$.

Furthermore, we evaluate the recall for the 20/30 best

C_2	$\text{DTW}_{\text{index}}^{\leq\theta}$			$\text{DTW}_{\text{index}}^{1,000}$		
	EM	T 20	T 30	EM	T 20	T 30
Q_1	1.00	0.75	0.63	1.00	0.75	0.63
Q_2	1.00	0.70	0.60	1.00	0.70	0.60
Q_3	1.00	0.85	0.70	1.00	0.85	0.70
Q_4	1.00	0.75	0.63	1.00	0.75	0.63
Q_5	1.00	0.80	0.83	1.00	0.80	0.83
Q_6	1.00	0.75	0.77	1.00	0.75	0.77
Q_7	1.00	0.65	0.87	1.00	0.65	0.87
Q_8	1.00	0.70	0.63	1.00	0.70	0.63
∅	**1.00**	**0.74**	**0.71**	**1.00**	**0.74**	**0.71**
Q_9	1.00	0.90	0.63	1.00	0.90	0.67
Q_{10}	1.00	0.90	0.63	1.00	0.90	0.67
Q_{11}	1.00	0.90	0.67	1.00	0.90	0.63
Q_{12}	1.00	0.90	0.63	1.00	0.90	0.67
Q_{13}	1.00	0.90	0.67	1.00	0.95	0.67
Q_{14}	1.00	0.95	0.70	1.00	0.90	0.73
∅	**1.00**	**0.91**	**0.66**	**1.00**	**0.91**	**0.67**

Table 3: Recall values for $\text{DTW}_{\text{index}}^{\leq\theta}$ and $\text{DTW}_{\text{index}}^{1,000}$.

ranked queries (i.e., ratio of 20/30 best matches calculated by DTW also belonging to the 20/30 best retrieval results when using $\text{DTW}_{\text{index}}$). The results are shown in the columns "T 20" and "T 30" of Table 2. Here, the impact of the segment length on the result quality becomes even more distinct. While with $\lambda = 10$ only one half of the top 20 matches was retrieved, for $\lambda = 20$ an accordance of 0.75 could be achieved. This supports our assumption that too short queries seem to result in a lot of insignificant matches which consequently reduce the overall retrieval accuracy.

In Section 4.2, we showed that the truncated index $\text{DTW}_{\text{index}}^{1,000}$ attains up to 4.5 times better response times compared to $\text{DTW}_{\text{index}}^{\leq\theta}$ (see Table 1). In the next experiment, we now compare the performance of the two indexes in terms of matching quality. Furthermore, we will extend our experiments to the larger collection C_2 and introduce a second set of queries to illustrate the effect of larger collections and the chosen query on the retrieval results.

Again, we use $Q_1 - Q_8$ as queries and perform audio matching on C_1 (see Table 2, results for $\lambda = 20$) as well as C_2 (see Table 3). For C_2 we additionally use audio snippets from the beginning of *Symphony No. 9, Molto vivace* by L. v. Beethoven as queries ($Q_9 - Q_{14}$, 70 − 76s). As the beginning of the *Molto vivace* is repeated twice during the piece, each interpretation contributes three exact matches, thereby generating a total of 18 exact matches for $Q_9 - Q_{14}$ (six of which are extracts of the piano recordings).

For $Q_1 - Q_8$ (both, in C_1 and in C_2) no differences between the matching results of $\text{DTW}_{\text{index}}^{1,000}$ and $\text{DTW}_{\text{index}}^{\leq\theta}$ could be detected. In contrast, subtle difference could be observed for $Q_9 - Q_{14}$. However, the overall performance remains unchanged (see Table 3).

The presented results, in combination with the runtime evaluations discussed in the previous section, suggest that $\text{DTW}_{\text{index}}^{1,000}$ with $\lambda = 20$ constitutes a good trade-off between speed, processable query length, and performance.

5. OUTLOOK

We introduced a novel audio matching procedure for intra-collection queries. The basic idea is to split the collection into segments of equal length, to then calculate the corresponding matches and to store them as indexes. During retrieval, those indexes are accessed and joint accordingly to determine the audio matches for the given query.

In the presented approach segment matches are merged employing conditions only depending on the segment length λ. Instead, one could imagine conditions that also consider the overlap τ and the query length. In the future, we intend to test alternative merging and ranking approaches with the goal of further improving the matching quality.

In addition, we plan on incorporating the proposed matching procedure into our digital library system PROBADO [3]. We expect this to result in a significant gain in retrieval quality for intra-collection queries without sacrificing good response times.

6. REFERENCES

[1] R. Agrawal, K.-I. Lin, H. S. Sawhney, and K. Shim. Fast similarity search in the presence of noise, scaling, and translation in time-series databases. In *Proceedings of International Conference on Very Large Data Bases (VLDB)*, pages 490–501, 1995.

[2] M. Casey, C. Rhodes, and M. Slaney. Analysis of minimum distances in high-dimensional musical spaces. *IEEE Transactions on Audio, Speech & Language Processing*, 16(5):1015–1028, 2008.

[3] D. Damm, C. Fremerey, V. Thomas, M. Clausen, F. Kurth, and M. Müller. A digital library framework for heterogeneous music collections—from document acquisition to cross-modal interaction. *International Journal on Digital Libraries: Special Issue on Music Digital Libraries*, 2012.

[4] C. Faloutsos, M. Ranganathan, and Y. Manolopoulos. Fast subsequence matching in time-series databases. In *ACM International Conference on Management of Data (SIGMOD)*, pages 419–429, 1994.

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

[6] P. Grosche, M. Müller, and J. Serrà. Audio Content-Based Music Retrieval. In M. Müller, M. Goto, and M. Schedl, editors, *Multimodal Music Processing*, volume 3 of *Dagstuhl Follow-Ups*, pages 157–174. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik, Dagstuhl, Germany, 2012.

[7] E. Keogh and C. A. Ratanamahatana. Exact indexing of dynamic time warping. *Knowledge and Information Systems*, 7(3):358–386, 2005.

[8] F. Kurth and M. Müller. Efficient index-based audio matching. *IEEE Transactions on Audio, Speech, and Language Processing*, 16(2):382–395, 2008.

[9] M. Müller. *Information Retrieval for Music and Motion*. Springer Verlag, 2007.

[10] M. Müller and S. Ewert. Towards timbre-invariant audio features for harmony-based music. *IEEE Transactions on Audio, Speech, and Language Processing*, 18(3):649–662, 2010.

[11] M. Müller and S. Ewert. Chroma Toolbox: MATLAB implementations for extracting variants of chroma-based audio features. In *Proceedings of the International Conference on Music Information Retrieval (ISMIR)*, pages 215–220, 2011.

[12] P. Papapetrou, V. Athitsos, G. Kollios, and D. Gunopulos. Embedding-based subsequence matching in time series databases. *ACM Transactions on Database Systems*, 36(3), 2011.

[13] A. Wang. An industrial strength audio search algorithm. In *Proceedings of the International Conference on Music Information Retrieval (ISMIR)*, pages 7–13, 2003.

[14] C. Yang. Efficient acoustic index for music retrieval with various degrees of similarity. In *ACM Multimedia*, pages 584–591, 2002.

[15] Y. Zhu and D. Shasha. Warping indexes with envelope transforms for query by humming. In *Proceedings of the ACM International Conference on Management of Data (SIGMOD)*, pages 181–192, 2003.